

e-Informatica

software engineering journal

2020

volume 14

issue 1



e-Informatica

e-Informatica
software engineering journal

2020 volume 14 issue 1



e-Informatica



Wrocław University of Science and Technology

Editor-in-Chief

Lech Madeyski (*Lech.Madeyski@pwr.edu.pl*, <http://madeyski.e-informatyka.pl>)

Editor-in-Chief Emeritus

Zbigniew Huzar (*Zbigniew.Huzar@pwr.edu.pl*)

Department of Software Engineering, Faculty of Computer Science and Management,
Wrocław University of Science and Technology,
50-370 Wrocław, Wybrzeże Wyspiańskiego 27, Poland

e-Informatica Software Engineering Journal

www.e-informatyka.pl, DOI: 10.37190/e-inf

Editorial Office Manager: Wojciech Thomas

Typeset by Wojciech Myszka with the L^AT_EX 2_ε Documentation Preparation System

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, transmitted in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publishers.

© Copyright by Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2020

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

www.oficyna.pwr.edu.pl;

e-mail: oficwyd@pwr.edu.pl; zamawianie.ksiazek@pwr.edu.pl

ISSN 1897-7979

Print and binding: beta-druk, www.betadruk.pl

Editorial Board

Editor-in-Chief

Lech Madeyski (Wrocław University of Science and Technology, Poland)

Editor-in-Chief Emeritus

Zbigniew Huzar (Wrocław University of Science and Technology, Poland)

Editorial Board Members

Pekka Abrahamsson (NTNU, Norway)

Apostolos Ampatzoglou (University of Macedonia, Thessaloniki, Greece)

Sami Beydeda (ZIVIT, Germany)

Miklós Biró (Software Competence Center Hagenberg, Austria)

Markus Borg (SICS Swedish ICT AB Lund, Sweden)

Pearl Brereton (Keele University, UK)

Mel Ó Cinnéide (UCD School of Computer Science & Informatics, Ireland)

Steve Counsell (Brunel University, UK)

Maya Daneva (University of Twente The Netherlands)

Norman Fenton (Queen Mary University of London, UK)

Joaquim Filipe (Polytechnic Institute of Setúbal/INSTICC, Portugal)

Thomas Flohr (University of Hannover, Germany)

Francesca Arcelli Fontana (University of Milano-Bicocca, Italy)

Félix García (University of Castilla-La Mancha, Spain)

Carlo Ghezzi (Politecnico di Milano, Italy)

Janusz Górski (Gdańsk University of Technology, Poland)

Tracy Hall (Lancaster University, UK)

Andreas Jedlitschka (Fraunhofer IESE, Germany)

Barbara Kitchenham (Keele University, UK)

Stanisław Kozielski (Silesian University of Technology, Poland)

Ludwik Kuźniarz (Blekinge Institute of Technology, Sweden)

Pericles Loucopoulos (The University of Manchester, UK)

Kalle Lyytinen (Case Western Reserve University, USA)

Leszek A. Maciaszek (Wrocław University of Economics, Poland)

and Macquarie University Sydney, Australia)

Jan Magott (Wrocław University of Science and Technology, Poland)

Zygmunt Mazur (Wrocław University of Science and Technology, Poland)

Bertrand Meyer (ETH Zurich, Switzerland)

Matthias Müller (IDOS Software AG, Germany)

Jürgen Münch (University of Helsinki, Finland)

Jerzy Nawrocki (Poznan University of Technology, Poland)

Mirosław Ochodek (Poznan University of Technology, Poland)

Janis Osis (Riga Technical University, Latvia)

Mike Papadakis (Luxembourg University, Luxembourg)

Kai Petersen (Hochschule Flensburg, University of Applied Sciences, Germany)

Łukasz Radliński (West Pomeranian University of Technology in Szczecin, Poland)

Guenther Ruhe (University of Calgary, Canada)

Krzysztof Sacha (Warsaw University of Technology, Poland)

Martin Shepperd (Brunel University London, UK)

Rini van Solingen (Drenthe University, The Netherlands)

Mirosław Staron (IT University of Göteborg, Sweden)

Tomasz Szmuc (AGH University of Science and Technology Kraków, Poland)

Iwan Tabakow (Wrocław University of Science and Technology, Poland)

Guilherme Horta Travassos (Federal University of Rio de Janeiro, Brazil)

Adam Trendowicz (Fraunhofer IESE, Germany)

Burak Turhan (University of Oulu, Finland)

Rainer Unland (University of Duisburg-Essen, Germany)

Sira Vegas (Polytechnic University of Madrid, Spain)

Corrado Aaron Visaggio (University of Sannio, Italy)

Bartosz Walter (Poznan University of Technology, Poland)

Dietmar Winkler (Technische Universität Wien, Austria)

Bogdan Wiszniewski (Gdańsk University of Technology, Poland)

Marco Zanoni (University of Milano-Bicocca, Italy)

Jaroslav Zendulka (Brno University of Technology, The Czech Republic)

Krzysztof Zieliński (AGH University of Science and Technology Kraków, Poland)

Contents

What Support do Systematic Reviews Provide for Evidence-informed Teaching about Software Engineering Practice? <i>David Budgen, Pearl Brereton, Nikki Williams, Sarah Drummond</i>	7
Technical Debt Aware Estimations in Software Engineering: A Systematic Mapping Study <i>Pawel Klimczyk, Lech Madeyski</i>	61
SIoT Framework: Towards an Approach for Early Identification of Security Requirements for Internet-of-things Applications <i>Ronald Jabangwe, Anh Nguyen-Duc</i>	77
Extending UML Use Case Diagrams to Represent Non-Interactive Functional Requirements <i>Saqib Iqbal, Issam Al-Azzoni, Gary Allen, Hikmat Ullah Khan</i>	97
System performance requirements: A standards-based model for early identification, allocation to software functions and size measurement <i>Khalid T. Al-Sarayreh, Kenza Meridji, Alain Abran, Sylvie Trudel</i>	117

What Support do Systematic Reviews Provide for Evidence-informed Teaching about Software Engineering Practice?

David Budgen*, Pearl Brereton**, Nikki Williams***, Sarah Drummond****

**Department of Computer Science, Durham University*

***School of Computing and Maths, Keele University*

****Centre for Electronic Warfare, Information and Cyber, Cranfield University*

*****Department of Computer Science, Durham University*

david.budgen@durham.ac.uk, o.p.brereton@keele.ac.uk, nikki.williams@cranfield.ac.uk,

Abstract

Background: The adoption of the evidence-based research paradigm by software engineering researchers has created a growing knowledge base provided by the outcomes from systematic reviews.

Aim: We set out to identify and catalogue a sample of the knowledge provided by systematic reviews, to determine what support they can provide for an evidence-informed approach to teaching about software engineering practice.

Method: We undertook a tertiary study (a mapping study of systematic reviews) covering the period to the end of 2015. We identified and catalogued those reviews that had findings or made recommendations that were considered relevant to teaching about industry practice.

Results: We examined a sample of 276 systematic reviews, selecting 49 for which we could clearly identify practice-oriented findings and recommendations that were supported by the data analysis provided in the review. We have classified these against established software engineering education knowledge categories and discuss the extent and forms of knowledge provided for each category.

Conclusion: While systematic reviews can provide knowledge that can inform teaching about practice, relatively few systematic reviews present the outcomes in a form suitable for this purpose. Using a suitable format for presenting a summary of outcomes could improve this. Additionally, the increasing number of published systematic reviews suggests that there is a need for greater coordination regarding the cataloguing of their findings and recommendations.

Keywords: Systematic review, education, provenance

1. Introduction

Over the half-century since software engineering became recognised as a distinct sub-discipline of computing [1], a degree of consensus has emerged about what it encompasses [2], as well as about the skills and knowledge that are needed by software engineers. For the latter, the ACM and IEEE produced a set of curriculum guidelines in 2004 aimed at consolidating ideas about what

a software engineer should acquire from an undergraduate education, and this was updated in 2015 after wide consultation across academia and industry [3].

However, although there is fairly general agreement about *what* a software engineer should know, much less attention has been given to *how* that knowledge might be obtained. Indeed, much of our knowledge is still based upon “expert opinion”, and although this is largely derived from

The work reported in this paper was partly undertaken when Nikki Williams was employed by Keele University.

experience, it lacks rigour as the foundation for what aspires to be an engineering discipline [4]. And, even when more systematically-acquired evidence is available, this does not necessarily mean that it will be readily accepted or adopted by practitioners [5, 6].

This raises two related questions. The first is concerned with how rigorous knowledge about the effectiveness of software engineering procedures might be derived (that is, how can we identify what works or doesn't work, and under what conditions?). And then when we have such knowledge, how can it most usefully be used for educating students?

In many disciplines, the major source of such knowledge is practice-related research, which is usually derived from "field studies" of the effects that arise from the use of some intervention. (For software engineering, the interventions might be the introduction of innovative technologies or processes, such as the use of agile practices.)

In software engineering research, there has been increasing use of empirical studies as a means of obtaining knowledge about software engineering practice. A comparison of the characteristics of papers submitted to, and accepted by, the ICSE conferences in 2002 and 2016 shows a significant increase in the reporting of empirical studies and the use of empirical models [7]. In particular, while no papers reporting empirical studies were accepted in 2002, this category made up 30% of the accepted papers in 2016.

Researchers have also adopted the *evidence-based* paradigm as a means of aggregating the knowledge available from a set of "primary" studies that investigate a given topic, based upon the use of the *systematic review* as its main tool [8]. This in turn has helped to create a growing knowledge base of research findings about software engineering procedures that should potentially be able to inform teaching (and hence, implicitly, inform practitioners). In [8], the authors suggested that adopting evidence-based software engineering (EBSE) would potentially provide:

- "A common goal for individual researchers and research groups to ensure that their research is directed to the requirements of industry and other stakeholder groups."

- "A means by which industry practitioners can make rational decisions about technology adoption."

For the study reported here we consider teachers and students to be additional stakeholders. Teachers can be regarded as being direct beneficiaries, as such knowledge can lend appropriate authority to reinforce teaching about software engineering topics. We view students as being indirect stakeholders, largely benefiting through the material presented by their teachers, rather than through direct use of the findings from systematic reviews.

To set this paper into context, we explain here how it originated and how it relates to other analyses that we have published. As experienced teachers, we wondered whether knowledge derived from the use of EBSE might be used in support of our teaching about software development practices. We envisaged that this support would have a number of forms, but our main expectation was that they might provide some authoritative support for the use of particular practices, or at least, an indication of when these were likely to be effective (or otherwise). In addition we expected that we might obtain some examples from experience about how or when to adopt new technologies.

In order to identify the extent and forms of knowledge about practice that was available, we originally undertook a study of a sample of the systematic reviews that were available up to the middle of 2011, selected on the basis that their topics related to practice, with our findings being reported in [9]. Although that study identified a set of potentially useful systematic reviews, in trying to use these to inform our teaching, we realised that they rarely presented their findings in a readily-usable form. So, beginning in 2016, we undertook a further study (reported here) that extended the earlier one in two ways. Firstly, we included systematic reviews published to the end of 2015, so including more reviews that were undertaken when their form had become more established. Secondly, we have performed a more comprehensive process of selection and analysis, requiring that a review should not only cover a topic relevant to practice, but also provide

topic-related findings that were supported by its analysis of the available data.

While conducting this review, the problems encountered in identifying both relevant information about the processes followed in the reviews, as well as about their findings, led us to use our material to analyse and report on the ways that systematic reviews in software engineering were being reported [10] before writing a summary of our findings (this paper). Our aim was to persuade authors and reviewers of the urgent need to improve the quality of published reviews.

A separate question that arose was how extensively practitioners formed the participants in the primary studies used in our set of 49 systematic reviews and to what extent these were conducted in an industry setting (“field studies”)? Addressing this involved further additional data extraction, with the outcomes reported in [11]. We do discuss some of the findings from this analysis later, as they provide useful supplemental information about the context of the knowledge available from the systematic reviews.

We begin by examining the evidence-based paradigm and the way that this has been employed by other disciplines. We then examine how its use has been adopted in software engineering; identify the forms of knowledge systematic reviews can provide; describe the design and conduct of our own study; and report our findings. We also examine the ways that other disciplines have used such knowledge to inform their teaching, and what we might learn from their experiences.

2. The evidence-based paradigm and software engineering

Use of the evidence-based paradigm originated in what has become known as *Evidence-Based Medicine* (EBM), by which a medical practitioner can draw upon the findings and recommendations from systematic reviews to aid them in making decisions about how to treat individual patients. Some of its success, in terms of its widespread adoption, derives from the nature of the clinical studies used in the reviews. While these are human-centric, the participants are usually *recip-*

ients of the treatment being studied, and so any variation in the outcomes is likely to occur mainly because of physiological differences among the participants. This, together with the extensive use of Randomised Controlled Trials (RCTs) – which are field experiments with rigorous controls, allows the findings from a set of primary studies to be synthesised using statistical meta-analysis [12]. Use of such forms of analysis makes it possible to assign a high level of confidence to the outcomes.

The evidence-based paradigm has also been successfully adapted to the needs of other “social” disciplines (in which humans *interact* with each other), including management, education and psychology as well as to more general social and health-related fields [13–15]. For these, other forms of synthesis that may be more appropriate to particular forms and mixes of primary studies have been developed. An overview of the forms that are potentially useful for software engineering is provided in [16], and in addition, a form of synthesis that can aggregate qualitative and quantitative evidence has been proposed for use in software engineering research [17].

While the term evidence-based software engineering (EBSE) is often used in analogy with evidence-based medicine (EBM), this can lead to inflated expectations. Rather than RCTs, empirical research in software engineering employs a mix of primary study forms that is actually more typical of the social sciences. In addition, the “treatment” used in software engineering studies usually involves participants in actively performing creative tasks related to software development, rather than being passive recipients. Since these tasks are likely to differ in detail between studies, this makes it more difficult to synthesise the data using forms such as statistical meta-analysis. (Comparison with a number of other disciplines using systematic reviews suggests that the discipline most similar to software engineering is that of Nursing and Midwifery [18], which helps to highlight the “social” nature of software engineering, where humans both interact with each other, and also with (or via) technology.)

For many disciplines, systematic reviews, are apt to be commissioned by policy-makers and

research agencies, and hence the topics studied are likely to be ones considered to be of strategic importance to that discipline and its practitioners. In addition, the task of searching for primary studies will often be performed by trained librarians [15]. In contrast, for systematic reviews on software engineering topics:

- coverage of key topics is uneven (see Appendix B) and the choice of topics appears to be almost entirely researcher-driven, with little to indicate that professional bodies, research agencies or industry have so far taken much interest in identifying suitable topics;
- the quality of reviews is apt to be uneven, particularly with regard to the rigour with which the primary studies are selected, categorised and synthesised [19].
- reporting is apt to be poorly structured and findings are not presented clearly [10];
- many studies use unnecessarily weak forms of synthesis [16];

Together, these influence the form and quality of the available knowledge.

3. The nature of software engineering knowledge

In this section we consider what forms of knowledge useful for teaching about software engineering practice can be provided from systematic reviews.

3.1. The nature of the knowledge provided from systematic reviews

The knowledge provided from any systematic review can be expected to be organised around the research question that the review is seeking to answer, as well as whether this question is concerned with issues related to research or to practice. Three important aspects of this knowledge are: the form in which the findings are presented; the strength of evidence supporting these findings; and how useful they are.

In terms of their usefulness for teaching, in examining the reviews we selected, we have observed that systematic reviews commonly provide

knowledge about practice in three different forms (and obviously, the findings of any review may consist of a mix of these).

The first way in which the presentation of the findings is structured is concerned with knowledge that has been derived from the *experiences* of others, in the form of lessons that have been derived about particular software engineering activities. The investigation of the effects of user participation in software development reported in [20] offers a good example of a topic where presenting qualitative knowledge about the experiences of others may well be the most useful form of knowledge to provide. Pedagogically, this can be viewed as providing broader knowledge about software engineering activities than can usually be provided in the classroom, or through practical exercises.

A rather different way of presenting knowledge that has been derived from *experience* is to provide a list of *factors* that should be considered when undertaking some task or adopting a technique. A good example is provided by [21], where the authors identify the factors that can make for the effective adoption of global software development practices. This type of knowledge can provide more directly useable guidance, possibly in the form of checklists, and hence can usefully be used to supplement classroom teaching about a given topic.

The third way to present knowledge is largely concerned with providing guidance about *choices* between different techniques. Such knowledge is more quantitative in its nature, and may well be involve *ranking* the different options in some way. A good example of using such a form is provided in the review by Dieste and Juristo [22] that assesses the effectiveness of different requirements elicitation techniques. From a pedagogical perspective, where the findings are organised in this form, they can be used to provide an authoritative basis for choosing to use particular practices.

The usefulness of any systematic review is also dependent upon the *provenance* for its findings – that is, how far we can be confident that the original primary studies are reliable and relevant. One reason for systematic reviewers to perform a quality check on the primary studies when per-

forming a systematic review is to help make some assessment of their reliability, in order to inform the process of synthesis. If they conclude that a primary study was conducted well, this provides some reassurance that including its findings in a synthesis procedure will help with producing sound findings from that process. This in turn provides scope for assessing the *strength of evidence* supporting the findings from a review [23].

The issue of the *relevance* of the findings from the primary studies used in a review is more challenging. In [24] this is defined as the “potential impact the research has on both academia and industry”, and the authors observe that the long maturation period for technology makes it “infeasible to use the actual uptake of research results by industry” as an evaluation tool. They propose an evaluation model for relevance that is based upon “potential for impact” and that uses four aspects: subjects; context; scale; and research method. Unfortunately, the reports of systematic reviews rarely provide much detail about these characteristics of the primary studies, and particularly about their context and scale, so we were not able to apply this model retrospectively in our analysis.

From the perspective of the teacher wishing to use the findings as supplemental material, the first aspect should require little more than explaining to students about the nature of a systematic review, in order for them to understand the nature of the evidence. An appreciation of the second (and to some extent the third) aspect may require a rather fuller explanation of the forms and limitations of empirical software engineering studies. However, since few software engineering systematic reviews provide any information about the strength of evidence, our own experiences suggest that this explanation need not be particularly detailed or extensive.

3.2. Categorising software engineering knowledge

Categorising and organising software engineering knowledge has been the goal of a number of initiatives. ACM and IEEE have jointly sponsored two that are relevant to this paper.

- The *Software Engineering Body of Knowledge* (SWEBOK) [2].
- The software engineering undergraduate curriculum guidelines (SE2014), and within this, the Software Engineering Education Knowledge (SEEK) categorisation of relevant knowledge [3].

The first of these is largely concerned with identifying the topics that collectively comprise the activities that make up software engineering practices, and where possible, identifying good sources of material related to these. So it can be considered to provide an expert interpretation of the nature of software engineering itself.

The second is concerned with identifying what an undergraduate studying software engineering should know, and hence the SEEK has been used in this paper to categorise the systematic reviews identified within the tertiary study. Even where a student is not studying software engineering as the major element of a degree, these are still topics that they need to be aware of, although perhaps in less detail than would be appropriate for a specialist course.

Table 1. Knowledge Areas used to categorise the SEEK

Knowledge Area	Key
Computing Essentials	CMP
Mathematical and Engineering Fundamentals	FND
Professional Practice	PRF
Software Modelling and Analysis	MAA
Requirements Analysis and Specification	REQ
Software Design	DES
Software Verification and Validation	VAV
Software Process	PRO
Software Quality	QUA
Security	SEC

As a framework, while the SEEK can appear to be organised around technology issues rather than “social” issues, this impression is misleading. Table 1 lists the major *Knowledge Areas* (KAs) used to structure the 2014 version of the SEEK. Each Knowledge Area is organised as a set of *Knowledge Units* (KUs) and both in these and in the guidelines there is quite extensive emphasis

upon the importance of more “social” aspects of software engineering such as the human interactions that occur in agile development and groupwork. Also, as emphasised in the Curriculum Guidelines, the Knowledge Areas are not meant to be templates for modules.

4. Research method

In order to answer the question posed in the title of this paper, we divided this into two separate, but linked, research questions, as follows.

RQ1: *Which systematic reviews published up to the end of 2015 produced findings that were relevant to teaching about practice in software engineering?*

RQ2: *What guidance did each systematic review provide that could help a student (or practitioner) to understand how to make an effective choice or use of a technology or practice?*

To answer RQ1, we conducted a systematic mapping study of published systematic reviews (a tertiary study). We then used the Knowledge Areas from the SEEK to categorise those that were selected as being relevant. To answer RQ2, we analysed the outcomes from each of the systematic reviews that we included, in order to identify relevant findings and explicit recommendations. As a point of clarification regarding RQ2, we did expect that for students, the process of understanding this guidance was something that would usually be mediated by a teacher. Indeed, for both students and practitioners, we expected that the findings of a review would mainly provide “help” by identifying those circumstances where a technique or practice might be most effectively employed (or where it would be inappropriate to employ it).

In the rest of this section, we explain our choices for the procedures required to answer these two questions, and then the following section describes how these procedures were implemented.

4.1. Scope of the study

For a systematic review, the aim should be to find *all* of the primary studies that can provide findings relevant to the topic of the review, in

order to avoid bias. Because a mapping study has the purpose of creating a “map” of the knowledge available about a topic, rather than synthesising its inputs, it does not usually need to be quite as comprehensive. Our aim, as posed in the title and RQ1, is concerned with establishing whether teaching about practice *could* be supported by the findings of systematic reviews. We therefore considered that our question could be answered from a suitably large sample of systematic reviews.

We also restricted the scope of our study to those systematic reviews for which the findings were published in journals. The page constraints of conference proceedings often means that reports of systematic reviews have to omit important details. Additionally, while many systematic reviews are first reported in conference proceedings, it is quite common for a later and fuller version to also be published as a journal paper. Since we were concerned with finding those systematic reviews that were reported in sufficient detail to be of use in making decisions and choices, we felt that it was appropriate to constrain our study to reviews published in journals. It was also considered that this would make our final selection more readily accessible for teachers, students and practitioners.

For the period to the end of 2009, we selected the journal papers from three existing “broad” tertiary studies to form our set of candidate systematic reviews [25–27]. These studies used a mix of manual and electronic searching to achieve a comprehensive degree of coverage for that period. As no equivalent sources were available for the period January 2010 to end 2015 and the number of published systematic reviews was rapidly increasing, we searched five major software engineering journals for those systematic reviews published in this later period. These were IEEE Transactions on Software Engineering, Empirical Software Engineering, Information and Software Technology, Journal of Systems and Software, and Software Practice and Experience.

Our choice of these journals was made on the basis that these were major publishers of systematic reviews addressing software engineering practices. One of the journals (*Information and Software Technology*) also had a special section for systematic reviews.

4.2. The inclusion/exclusion criteria

We required that any reviews included in our study should address a topic relevant to practice (rather than research) and that these topics should also be relevant to “introductory” – as opposed to “advanced postgraduate” – teaching. (Since our model for this was based on the SEEK, it could be considered to cover anything that would be material for an undergraduate degree programme.) In addition the review needed to provide some knowledge about practice that was explicitly supported by a synthesis of the findings of the primary studies. The resulting inclusion/exclusion criteria for the study are described in Table 2.

To be included, a systematic review needed to meet all of the inclusion criteria, while it could be excluded if it met any of the exclusion criteria. Using the SEEK gave us a reasonably clear measure of the set of topics that we considered appropriate for answering our research question. In particular, even where a review might meet all of the inclusion criteria, if we considered its topic as inappropriate for an introductory course, it could still be excluded (Excl-4). Typically such reviews were on relatively advanced topics that combined different aspects of software engineering, such as “security in process-aware information systems” [28].

4.3. Searching for systematic reviews

Our decisions about scope, as described above, meant that the searching process was relatively straightforward. The set of 120 papers from the

three broad tertiary studies were listed in the reports, and for the journals we employed a manual search of index sections. We complemented the manual search by using an electronic search to check for any systematic reviews that might have been missed (not all systematic reviews have titles that explicitly identify them as being reviews).

4.4. Quality assessment

Quality assessment of systematic reviews is commonly performed by using the DARE criteria (Database of Attributes of Reviews)¹ that were originally devised for use in clinical medicine. In its current form, the DARE assessment is based upon the following five questions.

1. Are the review’s inclusion and exclusion criteria described and appropriate?
2. Is the literature search likely to have covered all relevant studies?
3. Did the reviewers assess the quality/validity of the included studies?
4. Were basic data/studies adequately described?
5. Were the included studies synthesised?

For this study we adopted the use of DARE as a means of providing an assessment of how thoroughly each systematic review had been performed, and hence some indication of how reliable the findings from it might be. (This was also employed in the three broad tertiary studies.) In doing so we also adopted the widely-used convention of scoring each question using a three-point scale: yes (1); partly (0.5); no (0), with the max-

Table 2. The Inclusion and exclusion criteria adopted for this study

	Criteria
Inc-1	The paper is published in a journal and either included in the three broad tertiary studies or in one of the five journals (depending on publication date).
Inc-2	The topic of the paper is related to practice and is considered appropriate for use with introductory teaching of SE, as defined by the SEEK.
Inc-3	The paper contains findings and/or recommendations that are explicitly supported by the reviewers’ analysis.
Excl-1	Systematic reviews addressing research trends.
Excl-2	Systematic reviews addressing issues related to research methodology.
Excl-3	Mapping studies with no synthesis of data.
Excl-4	Systematic reviews that address topics not considered to be relevant for introductory teaching of SE.

¹<http://www.crd.york.ac.uk/CRDWeb/AboutPage.asp>.

Table 3. Interpretation of the DARE Criteria used for the tertiary study

Criterion	Score	Interpretation
Inclusion and exclusion	yes	The criteria used are explicitly defined in the paper.
	partly	The inclusion/exclusion criteria are implicit.
	no	The criteria are not defined and cannot be readily inferred.
Search coverage	yes	The authors have searched four or more digital libraries and included additional search strategies OR identified and referenced all journals addressing the topic of interest.
	partly	Searched three or four digital libraries with no extra search strategies OR searched a defined but restricted set of journals and conference proceedings.
	no	Searched up to two digital libraries or an extremely restricted set of journals.
Assessment of quality	yes	The authors have explicitly defined quality criteria and extracted them from each primary study.
	partly	The research question involved quality issues that are addressed by the study.
	no	No explicit quality assessment of individual papers has been attempted.
Study description	yes	Detailed information is presented about each study.
	partly	Only summary information is presented about the studies.
	no	Details of the studies are not provided.
Synthesis of studies	yes	The authors have performed a meta-analysis or used another form of synthesis for all the data of the study.
	partly	Synthesis has been performed for some of the data from some of the primary studies.
	no	No explicit synthesis has been performed (as in a mapping study).

imum score then being 5.0. Table 3 explains how the scoring was interpreted for the DARE criteria in the case of this study.

For each of the DARE questions, a score of “no” was awarded where there was an absence of information (apart from “search coverage” where we had defined a lower bound). Likewise, a “yes” indicated that the description or related operations for that criterion exceeded some threshold. A score of “partly” indicated that, while something was provided, it might only be for some of the primary studies (say), or that it was provided in some aggregate form. Hence a rating of “partly” could be interpreted as “present but incomplete”.

We should also note that DARE is only concerned with the systematic review *process* and whether these activities have been performed, rather than how well they have been done. However, until we have better reporting of systematic

reviews performed in software engineering, it does not seem practical to employ some of the other forms of assessment discussed in [23] and [29].

4.5. Data extraction

Our inclusion criteria, as summarised in Table 2, required that we should be able to identify *findings* and *recommendations* for any systematic review that was to be included. This stemmed from a concern that, to be of use, a study had to present results that end-users could readily employ. For the purpose of data extraction, we used the following descriptions.

- A *finding* provides knowledge about the topic that an end-user might find useful in order to gain knowledge about the topic². However it is not of such a nature, or accompanied by such a degree of confidence, as to be able to

²In [10] we used the term “conclusion” rather than “finding”. Upon reflection, we felt that this could be ambiguous in this context, and so have adopted the use of “finding” in this paper.

Table 4. Core data extraction from systematic reviews

Item	Description
1.	Bibliographic information (title, authors, publication details).
2.	Our scores for the DARE criteria. (As interpreted in Table 3).
3.	Data about any quality assessment performed in the systematic review for the primary studies, including details about any checklist used for this.
4.	Details of how the quality scores from Item 3 were actually used in the systematic review.
5.	The size and nature of the <i>body of evidence</i> used in the review (numbers and types of primary study).
6.	The <i>context</i> relating to the body of evidence: details of participant types, period covered by the searching, search engines used, details of any manual searches, use of snowballing, number of studies retained at each stage of inclusion/exclusion.
7.	Any <i>findings</i> that are reported, or that could be derived from the later sections of the paper.
8.	Any <i>recommendations</i> reported or that could be derived.

act as the source of explicit advice about good or undesirable practice related to that topic.

- A *recommendation* provides an operationalisation of a finding that provides deeper understanding and that can be taken into consideration when making decisions about practice. So if possible, a recommendation should be accompanied by some measure of its *strength*, derived from the evidence available from the systematic review.

Because the presence of these could only be determined with certainty at the stage of data extraction, we accepted that some decisions about exclusion would occur during data extraction. The data extracted from each study is itemised in Table 4.

5. Conduct of the study

The study was conducted according to the plan, and this section provides some details about the procedures followed as well as the outcomes.

Because there was some overlap between the set of systematic reviews selected for our earlier study [9], for brevity when comparing the two studies, we refer to that study as EPTS1 (Education and Practice Tertiary Study 1), and refer to this study as EPTS2.

5.1. Study identification

As the set of papers found by the three broad tertiary studies was already determined, searching was only necessary for the papers from the

five journals published in the period 2010–2015. The manual search process was conducted by one of the authors (DB) and involved reading through the contents pages of the five journals examining titles of papers, and where necessary, also inspecting the abstracts.

To complement the manual search, an electronic search was also performed by an independent researcher. This was undertaken in two stages. In the first of these, covering the period 2010–2014, the *Scopus* digital library was used to perform a forward citation analysis of six papers that discussed the principles of EBSE and systematic reviews (listed in Table 5). This was performed in April 2016. The papers identified as being systematic reviews or mapping studies and published in the five journals were compared with the papers that had been found by manual search. However, this identified a large number of false positives, and for papers published in 2015, this problem became much greater. So for the second stage (period) for 2015, Scopus was searched using the terms: TITLE-ABS-KEY (“systematic literature review” OR “systematic review” OR “systematic mapping study” OR “mapping study”) AND DOCTYPE (ar OR re) AND PUBYEAR = 2015 AND (LIMIT_TO (SUBJAREA, “COMP”)). The results from this were sub-setted to select studies from each of the five journals and the papers that were identified as being mapping studies and systematic reviews were compared with the papers found by the manual search. This second search took place in May 2016.

Table 5. Papers used for forward citation analysis in Scopus

Title	Reference
Evidence-Based Software Engineering	[8]
Evidence-Based Software Engineering for Practitioners	[30]
Procedures for Undertaking Systematic Reviews	[31]
Guidelines for Performing Systematic Literature Reviews in Software Engineering	[32]
Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain	[33]
Systematic Review in Software Engineering	[34]

The manual search identified 140 papers and the electronic search added a further 16, giving a total of 156 systematic reviews from searching the journals. All studies were allocated an index number, those from the broad tertiary studies being numbered #1–120, and those from the journals #121–276.

Our sources are described in Table 6. For ease of reference, we have labelled these as *Source-set1* and *Source-set2*. We have also indicated the number of papers obtained from each of these sources.

Table 6. Details of the sources used

Period	Sources	Count
2004–2009 (Source-set1)	TS1: Tertiary Study 1 [25]	20
	TS2: Tertiary Study 2 [26]	33
	TS3: Tertiary Study 3 [27]	67
		120
2010–2015 (Source-set2)	IEEE Transactions on S/W Eng.	13
	Empirical Software Engineering	10
	Information and Software Technology	97
	Journal of Systems and Software	31
	Software Practice and Experience	5
		156

5.2. The inclusion-exclusion process

The process of inclusion/exclusion was performed in two stages. This was because the relevance of the topic could be fairly easily determined from the title and abstract, whereas determining the availability of appropriate findings and recom-

mendations (Inc-3) did require that the complete paper had to be read.

In the first stage the two criteria used were whether or not a study was a systematic review, published in a journal, that addressed a potentially relevant topic (Inc-1 and Inc-2). The studies that had earlier been included in EPTS1, published in the period up to mid-2011 and described in [9], had already been identified as meeting the second criterion, and so the only action required was to remove those published in conferences. Hence a full selection process was only performed for the studies with index values #146–276, which were those published from mid-2011 onwards and hence had not been used in EPTS1. This was performed by all four authors, working in different pairings that were allocated on a random basis. The only exceptions were the papers for which two of us (DB and PB) were authors, which were assessed by the other two reviewers. If the reviewers were unable to agree on exclusion of a paper, it was retained for the second stage. Using the Fleiss' Kappa [35] to assess the level of rater agreement for this first stage, as we were using multiple raters, produced a score of 0.490, which indicates *moderate* agreement, falling into the band of values usually considered as being acceptable (“fair to good”) [36].

The second stage was combined with the process of data extraction, which was based upon the data extraction model described in Table 4. This was applied to all of the reviews identified from the first stage, and all data extractions were performed by two members of the team, working independently, who then resolved any differences to produce an agreed dataset for a review.

Studies were only retained at this stage if we could identify clear findings and/or recommendations that could be linked to the data extracted as part of the systematic review (criterion Inc-3).

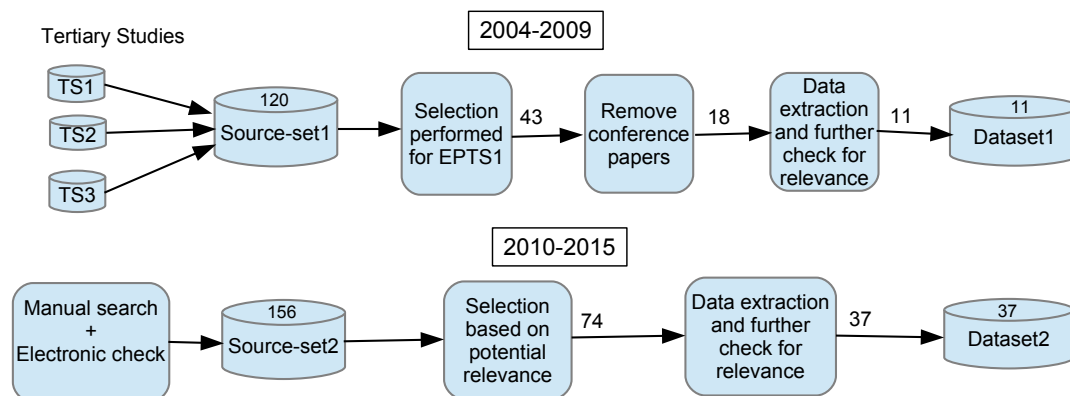


Figure 1. The overall selection process (TS1–TS3 are the three tertiary studies)

Some papers originally included in EPTS1 were excluded in this stage, on the basis that they had inadequate findings or recommendations. Figure 1 provides an overview of the overall process and resulting numbers. We have referred to the reviews selected from Source-set1 as *Dataset1*, and those from Source-set2 as *Dataset2*.

Because identification of the findings and recommendations from the systematic reviews was often a complex process (elements of these were apt to be spread around the final sections of a paper), we performed a further check upon the reliability of our interpretation. For each systematic review we tried to contact the designated corresponding author by e-mail and asked them to comment on our interpretation of the outcomes. Where this was no longer a valid address, we then tried contacting any of the authors for whom we could find a suitable e-mail address. In only two cases were we unable to trace any of the authors. We received 27 responses, all of which were generally in agreement with our interpretation, with 16 of them suggesting changes of wording, with all of these being minor.

The final set used in this study (EPTS2) contained 49 reports of systematic reviews, listed in Appendix A. Because the data for one systematic review was used for two analyses (#54 and #118), both of which met the inclusion criteria, there were actually 48 sets of primary studies.

5.3. Quality assessment

An assessment against the DARE criteria, using the interpretations provided in Table 3 was per-

formed as part of the process of data extraction, and using the same randomly-allocated pairings of reviewers.

5.4. Categorisation against the SEEK

To categorise the systematic reviews against the SEEK two of the reviewers (DB and PB) performed another analysis of the reviews after all of the data extractions had been completed. Again, our argument for doing this as a separate analysis, as against performing it as part of data extraction, was largely a matter of ensuring greater consistency of interpretation. It was considered that this would be more easily achieved if the whole set of studies was categorised in a single process.

For each study we determined both the most appropriate *Knowledge Area* (summarised in Table 1) and also what we considered to be a suitable assignment to the more detailed *Knowledge Unit* within this. While for some studies the most appropriate KA and KU values were relatively obvious, many did require quite extensive discussion to determine an appropriate allocation as inevitably, the topic of a systematic review and its findings may well span more than one KA or KU. Indeed, the nature of the findings may be more important than the topic of the review in terms of determining how it should most appropriately be categorised.

5.5. Further data extraction

As noted in Section 1, we have performed further analyses of the 49 systematic reviews. These are reported in [10] and [11] and involved some addi-

Table 7. Additional data extraction from systematic reviews

Item	Description
9.	Whether and how any quality scores derived for the primary studies were used (if at all).
10.	The form(s) of synthesis used in the study, and whether these classifications were made by the authors of the systematic review or by us. Categories used were: meta-analysis, narrative synthesis, meta-ethnography, grounded theory, cross-case analysis, thematic analysis, vote counting, and “other”. The definitions of these were taken from [16].
11.	The forms of primary studies used, where the primary studies were performed; who conducted these, and who formed the participants (students or industry practitioners) or what sources of data were used (industry or artificial).

tional data extractions. These were performed by two of us (DB and PB) and are summarised in Table 7. Some of this supplemental information is included in Appendix B. We should note that for both of these, the process of study selection was as reported here. So, while they investigated further questions about reporting and provenance, their analyses were limited to providing answers related to studies about software engineering practice.

6. The findings – What knowledge is available?

To present the outcomes from the process described in the previous sections, we begin by providing an overview of all of the studies. We also look at some of the supplementary information about these, with particularly regard to

such aspects as *provenance*. We then look at the studies in more detail, and in particular, present the *findings* and *recommendations* that were extracted for each one. These are grouped under the different SEEK headings, enabling us to also comment on the extent of the available knowledge for each heading.

6.1. Summary of the systematic reviews

As the total number of studies is quite large, we have presented the summary of the findings for the two datasets separately in Tables 8 and 9. This is largely a convenience for presentation, although it also helps distinguish the reviews that were undertaken when the practices for systematic reviews in software engineering were less well established. Both are described using the same format. Each entry is described in terms of its index number (#1–#276) as used in this

Table 8. Details of the systematic reviews included in this study: Dataset1 (2004–2009)

No.	Period covered	Topic and Citation	SEEK KA	DARE	Primary Studies	
					ind.	total
52	unclear	Motivations for adopting CMM-based SPI [37]	FND	2.5	49	49
54	1980–6/2006	Motivation in software engineering [38]	PRF	5.0		79
118	(as 54)	Models of motivation [39]	PRF	(as 54)		(79)
15	1992–2002	Capture-recapture in s/w inspections [40]	VAV	1.5	1	25
66	1996–2007	Search-based non-functional testing [41]	VAV	4.5	17	35
82	1969–2006	Regression test selection techniques [42]	VAV	4.5	4	36
8	to 2006	Estimation of s/w development work effort [43]	PRO	1.0	14	16
22	unclear	Assessment of development cost uncertainty [44]	PRO	2.5		40
39	1994–2005	Benefits of software reuse [45]	PRO	3.5	11	11
50	1996–3/2006	SPI in small and medium s/w enterprises [46]	PRO	4.0	45	45
84	to 2007	Effectiveness of pair programming [47]	PRO	4.0	5	19
102	1995–2005	Managing risks in distributed s/w projects [48]	PRO	2.5	72	72

Table 9. Details of the systematic reviews included in this study: Dataset2 (2010–2015)

No.	Period covered	Topic and Citation	SEEK KA	DARE	Primary Studies	
					ind.	total
135	1980–2008	Antecedents to personnel’s intention to leave [49]	PRF	3.0	72	72
246	2003–4/13	Newcomers on OSS projects [50]	PRF	3.5	20	20
167	2006–2011	Evaluating cloud services [51]	VAV	4.0	82	82
197	to 10/2011	Software fault prediction metrics [52]	VAV	4.5	81	106
205	2000–2011	Test-Driven Development [53]	VAV	4.5	22	41
252	2002–2013	Metrics in Agile/Lean development. [54]	VAV	3.5	30	30
124	1970–2007	Characterising s/w architecture changes [55]	DES	3.5		130
130	1997–2008	Aspect-oriented programming [56]	DES	4.5	6	22
154	1995–2009	Software design patterns [57]	DES	2.0	11	18
123	unclear	Domain analysis tools [58]	MAA	3.5	7	19
126	1989–2006	Does the TAM predict actual use? [59]	MAA	5.0		79
146	2000–2010	Dependency analysis solutions [60]	MAA	2.5	38	65
155	2000–2010	Fault prediction performance [61]	MAA	4.5	35	36
134	to 3/2005	Elicitation techniques [22]	REQ	5.0	7	32
161	1993–2011	Stakeholders for requirements elicitation [62]	REQ	4.5	42	42
259	1992–2/14	Use case specifications research [63]	REQ	4.0	27	119
219	to 2012	OO measures and quality [64]	QUA	4.5	33	99
121	2000–2007	Global software engineering [21]	PRO	3.0	37	56
138	to 2009	Measuring and predicting software productivity [65]	PRO	4.5	25	38
150	to 6/2010	Agile product line engineering [66]	PRO	3.5	14	39
157	to 2/2011	Test-Driven Development [67]	PRO	4.0	10	37
160	to 4/2009	Reconciling software development methods [68]	PRO	2.5	42	42
174	unclear	Industrial use of software process simulation [69]	PRO	3.5	87	87
175	to mid-2008	Selecting outsourcing vendors [70]	PRO	3.5	77	77
193	to 7/2010	Social software for global software dev. [71]	PRO	4.0	61	84
215	to 12/2013	Software development in start-ups [72]	PRO	4.5	30	43
217	1997–2011	Influence of user participation [20]	PRO	3.5	82	82
222	1990–2012	The Kanban approach [73]	PRO	4.0	37	37
228	1997–1/08	Software process assessment [74]	PRO	2.5	22	22
236	2001–2013	Global team dispersion [75]	PRO	4.5	40	43
239	to 2011	Using CMMI with Agile [76]	PRO	4.5	59	60
241	1980–2012	User-involvement and success [77]	PRO	4.5	87	87
244	1990–2012	Development effort estimation [78]	PRO	5.0	61	61
249	2002–10/12	User-centred agile development [79]	PRO	4.5	26	83
260	to 5/15	Use of SE practices in science [80]	PRO	2.5		43
268	1996–2/08	Product derivation support [81]	PRO	2.0		118
276	1996–10/13	Adopting SPL [82]	PRO	3.0	31	31

study, the period covered by the search in the systematic review, its topic, and reference. The tables also provide some of the key information about each review: the SEEK Knowledge Area (KA) it has been assigned to (the keys we use were provided in Table 1); the DARE score we derived for the study; the number of primary studies that we could identify as being either explicitly or implicitly conducted in an industry setting or making use of industry participants; and the total number of primary studies.

The issue of the provenance of the primary studies is discussed in more detail in [11], where we have categorised the context for the primary studies used in each systematic review as far as we were able, based upon the available information. Two key points from this are worth repeating here. The first is that for those systematic reviews where we could not determine whether some of the primary studies were explicitly or implicitly conducted in an industrial setting, it is highly probable that many of these were actually

Table 10. Counts of reviews for each Knowledge Area

KA	Topic	Dataset 1 (up to 2009)	Dataset2 (2010–2015)	Total
FND	Fundamentals	1	–	1
PRF	Professional Practice	2	2	4
VAV	Verification and Validation	3	4	7
DES	Design	–	3	3
MAA	Modelling and Analysis	–	4	4
REQ	Requirements Analysis/Specification	–	3	3
QUA	Software Quality	–	1	1
PRO	Software Process	6	20	26
Totals		12	37	49

Table 11. Forms of guidance provided by reviews

Type	Systematic Reviews								
	FND	PRF	VAV	DES	MAA	REQ	QUA	PRO	
Experience	–	–	#167, #252	#124, #154	#123, #146	#259	–	#39, #50, #174, #193, #222, #239, #260, #102	
Lists of Factors	#52	#54, #118, #135	–	–	#155	–	–	#160, #276, #84, #215, #236, #241, #121, #244	
Comparisons	–	–	#15, #66, #197	#130	–	#134	#219	#8	

industry-related, but the lack of detail meant that we simply could not tell. The second point is that what was considered to be an acceptable primary study in terms of the inclusion/exclusion criteria used in the review, and the way that these were interpreted, did vary quite considerably. Some reviews included a number of non-empirical reports among the primary studies, as well as papers that were classified as “opinion”, “experience” and even “theory”. So while other characteristics such as DARE scores might usefully be compared across a set of systematic reviews, it is definitely not appropriate to make comparisons between the numbers for each type of primary study as reported by different systematic reviews.

The answer to RQ1 (which systematic reviews produced findings relevant to teaching about practice?) is provided by the entries in Tables 8 and 9. Overall, as indicated, we were able to identify 49 systematic reviews (from 276) that contained findings considered to be of use in teaching about software engineering. In the tables, the systematic reviews have been grouped under the SEEK Knowledge Areas, which also highlights

the uneven distribution of reviews across the KAs. Table 10 gives the counts of the reviews categorised under each KA. The large proportion categorised as PRO arises in part because much of what we do in software engineering involves processes. Many of the systematic reviews can be described as investigating “best practice”, where this may relate to testing, design, etc., and these ended up being categorised as PRO wherever we concluded that the emphasis was more upon practice rather than the technology involved.

The answer to RQ2 (what guidance did each systematic review provide?) is contained in the fuller descriptions of the findings and recommendations, together with their context, provided in Appendix B. As discussed earlier, we observed that systematic reviews provide guidance in a number of forms, largely depending upon the research question being addressed by the review. In Table 11 we identify those reviews providing each of the three types of guidance (experience, lists of factors, and comparisons). Inevitably, the findings of reviews do not always fall exactly into one of these categories, and so

we have only included the 35 systematic reviews where we collectively felt that the findings mainly fitted one category. What Table 11 does show though is that few reviews provided comparative findings. VAV was the only KA for which there was more than one systematic review (3 from 5) providing comparative findings, largely because these were comparing testing practices that produced deterministic outcomes concerned with whether or not a test was successful. (Many aspects of software engineering, such as analysis and design, address “ill-structured” problems [83], and so rarely provide true-false results when comparisons are being made.)

6.2. The findings and recommendations for each review

In this subsection we present the material that helps answer RQ2 (guidance “that could help a student (or practitioner) to understand how to make an effective choice or use of a technology or practice”). For each review, we excluded any findings that were related to research issues or future developments. (Almost every systematic review identifies a need for more and better primary studies.) Where possible, we have taken the wording for the findings and recommendations directly from the systematic reviews. One consequence of this is that the findings from different systematic reviews are apt to be formulated at different levels of granularity. However, given the heterogeneity of the reviews, we considered that it was impractical to present the findings in a uniform matter.

In reporting the findings, we have also provided information about their *provenance*, wherever this was available. This information is provided to aid the reader to make some assessment of the confidence that they might choose to place in the findings. However, as noted earlier, the variation in different reviews between the way that primary study types were interpreted, as well as in the inclusion/exclusion criteria used, means that this information should only be treated as indicative.

The details for each systematic review are provided in Appendix B. For each review we provide the following information (where available)

1. The main SEEK knowledge area and knowledge unit identified as appropriate.
2. The title of the systematic review.
3. Citation details.
4. The DARE score, reported on a scale of 0–5.
5. Any information available that might provide an assessment of the *strength of evidence* for the findings. Where possible, we report this for each finding.
6. The number of *primary studies*. Where possible, we included the following additional information:
 - The count of primary studies that we could *explicitly* identify as being conducted in an industry setting.
 - The count of primary studies that were *implicitly* conducted in an industry setting, based upon comments in the text.
 - The count of primary studies conducted in an “academic” setting (such as experiments that used student participants).
7. The form(s) of *synthesis* used in the study, noting that some did use more than one form to answer different research questions. (We did not attempt to classify the forms of synthesis used in the earlier studies (*Dataset1*).
8. The *findings* from the study.
9. The *recommendations* from the study.
10. Information about any response from the authors to our request for them to check the accuracy of our extraction of the findings and recommendations.

We have grouped the reviews according to their assignment to SEEK Knowledge Areas. For each review, we suggest the most relevant Knowledge Area and Knowledge Unit, accepting that many reviews do not fit neatly into the SEEK model. We have also noted where there are Knowledge Units (other than those dealing with issues such as “concepts”) for which there are no systematic reviews, in order to help illustrate the overall degree of coverage.

6.2.1. Findings – fundamentals (FND)

Perhaps not surprisingly, there is only one systematic review categorised under this heading. The key details for this are provided in Table B1. The reason for including this review under FND was that we felt it best fitted the Knowledge Unit *engineering economics for software*. (This was the only heading for this KA that did not address “foundations”.)

In this review, the conclusions about the reasons for adopting SPI (Software Process Improvement) largely reinforce the claims made in the literature.

6.2.2. Findings – professional practice (PRF)

We classified four systematic reviews under this heading, described in Tables B2, B3, B4 and B5. Two of these (#54 and #118) used the same dataset, but performed quite different analyses of the material. We were also unable to determine a specific Knowledge Unit for those two analyses, due to the wide span of issues that they address. There were no systematic reviews directly addressing the KUs *communications skills* or *professionalism*, although some other systematic reviews did indirectly address issues related to team and group communication.

The first two reviews (which share a dataset) address issues around what motivates software engineers and provide details of factors considered relevant. Study #135 is also related to staff (de)motivation, providing a set of related recommendations. The remaining study addresses the role that group dynamics plays when participating in open source development.

6.2.3. Findings – software verification and validation (VAV)

There were seven reviews included under this heading. These are summarised in Tables B6–B12. These reviews provide a set of findings that span three of the four Knowledge Units making up the VAV Knowledge Area. We have no reviews for one KU, *problem analysis and reporting*.

These reviews span a range of issues. Most are concerned with techniques for selecting or evaluating tests (such as those used for regression testing) and provide rankings of different approaches that are likely to be directly applicable to practice.

6.2.4. Findings – software design (DES)

Software engineering can be considered as very much a “design” discipline, with “design thinking” permeating many activities, including of course, software design. However, the creative element involved in designing also means that this Knowledge Area forms a significant challenge for empirical studies. There are only three systematic reviews in this group, summarised in Tables B13–B15, although they do address three separate Knowledge Units. KUs with no contributions are *design concepts*, *human-computer interaction design* and *design evaluation*.

None of the reviews offer very strong conclusions, and in the only one that offered more specific guidance about design choices (#130), it was noted that these were based upon a low strength of evidence.

6.2.5. Findings – modelling and analysis (MAA)

The four systematic reviews under this heading are all classified as belonging to the same Knowledge Unit (*types of models*). Given that the other two KUs address *foundations* and *fundamentals*, this is perhaps not surprising. Tables B16–B19 provide a summary of these reviews.

The reviews span a range of issues including model reliability (#126) and observations about fault prediction (#155). Collectively they do provide helpful guidance about some specific models that are used by software engineers.

6.2.6. Findings – requirements analysis and specification (REQ)

The three systematic reviews addressing requirements, described in Tables B20–B22, cover two of the four Knowledge Units making up the REQ Knowledge Area. In particular, we have no re-

views that address the KU *requirements validation*.

All the reviews provide useful insight into the approaches used in requirements engineering. Review #134 particularly provides a useful rating of different elicitation techniques, and all three offer useful insight.

6.2.7. Findings – software quality (QUA)

There is only one systematic review categorised under this heading. The key details for this are provided in Table B23. This was categorised against the KU *product assurance*, and there were no reviews covering the KU *process assurance*. The review does offer useful insight into the relative merits of a range of object-oriented measures.

6.2.8. Findings – software process (PRO)

By far the largest set of reviews fall into this Knowledge Area (which does form something of a “catch-all”). We have grouped these by Knowledge Unit (KU), although we should note that only three of the five knowledge units were covered. There were no reviews classified as *configuration management* (PRO.cm) or *evolution processes and activities* (PRO.evo). Table B24 onward provide the details for this set of reviews.

With so many reviews being classified as belonging to this KA, it is difficult to provide a concise and general summary of what is useful for practice and teaching. Many of the reviews classified against *project planning and tracking* offer quite specific and detailed advice that is highly relevant to both teaching and practice. In contrast, for *process concepts* many of the findings tend to be more in the nature of observations, with *process implementation* coming somewhere between these. Overall though, this set of reviews do provide a fertile source of experience for others to draw upon.

7. Discussion

We first consider what the outcomes from our study tell us about the knowledge available from

this set of systematic reviews, and what the limitations on this knowledge are. We go on to consider how this knowledge might be used to inform teaching (and hence in the longer term, practice) by looking at how such knowledge is used in other disciplines, and hence what lessons might be learned. We then consider the threats to validity for this study, since we need to determine how trustworthy our findings are, both in terms of the selection of the systematic reviews, and also the findings and recommendations that we extracted from them. Finally, we consider how such knowledge can be gathered more effectively and completely in the future, and in particular, how it might be possible to avoid having to do this retrospectively (and laboriously), as in this tertiary study.

7.1. How good is the knowledge available from systematic reviews?

In answering RQ1, we can identify 49 (out of 276) systematic reviews that provide knowledge about software engineering practice and hence might be used to support teaching about software engineering. The systematic reviews that we identified also span a range of topics when matched against the SEEK, although they are not evenly distributed between the Knowledge Areas. The extent, quality, and form of the knowledge is also unevenly distributed, with some reviews providing findings that provide quite useful information about practice, while others are rather less specific.

In addition, few reviews provided any indication of the *strength of evidence* available to support their findings from the primary studies. Examination of the 49 reviews shows that only two of them (#130 and #239) made use of the GRADE approach to assess the strength of evidence for their findings [84], as recommended in [23]. A few of the others (#008, #022, #039, #197, #215) did also make assessments through unspecified means. Where provided, such assessments tend to indicate a strength of evidence for recommendations as being “low” (#130 and #239) or “modest” (#008). However, as noted in the revised guidelines on conducting systematic reviews in software engineering [29], empirical software engineers “must often make do with much weaker forms of study” (than those working in other disciplines).

It is also worth noting that some of the more qualitative reviews, such as those identifying “factors relevant to the adoption of X”, are unsuited to the use of an approach such as GRADE. A number of these did provide tables that listed and enumerated the primary studies that identified a particular factor as being significant, with examples of this occurring in #54, #161 and #205.

To address this question, we have identified the set of systematic reviews which we consider offer both useful and usable guidance about practice. To select these, each of the authors was asked to rate each review, using the information presented in Appendix B, and assigning one of the following values to it.

“**y**” if the review was one that *could* be readily used as an example when teaching;

“**p**” for reviews that *might* be used;

“**x**” if the review should *not* be used as an example.

In performing the rating, each author was asked to consider the following three factors.

1. The *usefulness* of the review: such that its outcomes relate to a reasonably “mainstream” topic that might be included in an introductory course on software engineering.
2. The *usability* of the review’s findings, whereby these can provide some element of guidance about what a software engineer might be advised to do in practice.
3. The *quality* of the review, largely based upon the DARE score. It was suggested that a score of ≥ 3.5 would be acceptable, while also bearing in mind that earlier reviews often had less conventional reporting structures.

Each review was considered on its own merit, and there was no constraint upon how many

Table 12. Findings considered most useful and usable

Score	Review	KA.KU	Table	Knowledge	Summary
4.0	#134	REQ.er	B20	comparison	Examines knowledge elicitation techniques. Unstructured interviews generally perform as well as or better than other forms such as introspective techniques when considering effectiveness, efficiency and completeness.
	#236	PRO.imp	B40	list of factors	Assesses impact of global dispersion for development process and product quality. Lists key effects and recommends issues to consider for such projects.
3.5	#197	VAV.fnd	B10	comparison	Assesses different metrics for their usefulness in fault prediction. Recommendations relate to project characteristics.
	#84	PRO.imp	B34	list of factors	Provides guidance on when to employ pair programming.
	#241	PRO.imp	B42	list of factors	Analyses how far system success is related to user involvement in development and the forms that this takes.
3.0	#82	VAV.tst	B8		Assesses the effectiveness of test selection techniques for regression testing.
	#205	VAV.fnd	B11		Examines studies of test-driven development (TDD).
	#39	PRO.con	B24	experience	Identifies the benefits of software reuse based upon its use in industrial studies.
	#217	PRO.imp	B38		Looks at the consequences of user participation and involvement (UPI) in terms of project success in industry projects.

reviews could be given a particular rating. (The number of “y” ratings employed ranged between 10 and 17.)

Since our teaching experiences stemmed from teaching different courses and we had different interests within software engineering, we did not expect to obtain close agreement from this process. So a “score” for each review was computed by assigning each “y” to a value of 1.0, a “p” as 0.5, and an “x” as 0.0, and then summing the four values.

Table 12 shows the index values for the top-scoring reviews that emerged from this process. We have also indicated the type of knowledge provided by these reviews, where a single value was available, and provided a summary of their findings together with a reference to the Table in the appendix where further details can be found. While not too much weight should be placed upon this relatively informal exercise, it is interesting to note the predominance of reviews categorised as VAV and PRO, as well as of reviews with more “structured” findings in the form of lists or comparisons. It does also indicate that, while all 49 reviews were considered relevant enough to be included in the tertiary study, few of them achieved this quite basic quality threshold for the three criteria, with “good” studies available for only a few Knowledge Areas.

So, to answer the question posed in the heading for the subsection (and RQ2), we can conclude that while suitable evidence-based material is becoming available for use by teachers, only a rather disappointingly small proportion of systematic reviews appear to have findings that can readily be used.

However, there is one quite important caveat that should be mentioned here. In the above exercise we only considered *direct* use of this material in teaching on introductory courses, enhancing what is covered in the textbooks. There are however other ways of using this knowledge, such as in course design (for example, using the findings on the unsuitability of design patterns for use by novices to determine how this topic would be covered in a course). There is also scope to use the findings differently on more advanced courses, including postgraduate ones, or with

individual student projects. All of the 49 reviews are viewed as having findings that are *potentially* useful, but these may need to be used in different ways. We address this issue further in the next sub-section.

7.2. Using the findings to support teaching and practice

Having identified a set of systematic reviews that contain knowledge that is useful for teaching and practice, this raises the question of how to *use* this material? To help answer this we looked at how other disciplines make use of such material.

Studies in education and healthcare have investigated how students and practitioners understand and engage with the findings from empirical research. This is relevant for software engineering, since using the material from this study would require familiarity with evaluation practices and empirical studies.

In education, a ‘rapid evidence review’ investigated what is known about effective approaches to school and teacher engagement with evidence [85]. The report points out that *knowledge mobilisation*, the process of making research findings more accessible and usable requires a supportive infrastructure, including collaborations between researchers and teaching professionals, intermediaries to translate evidence into tools and professional bodies that provide leadership on the use of evidence in education. Also, the review suggests that evidence needs to be contextualised and presented in clear and structured summaries of effective approaches. This point is also made by Goldacre³, who emphasises the need for better support for the dissemination of research findings, as well as by others, in relation to evidence based healthcare, where structured abstracts and plain language summaries are advocated [86, 87].

As well as learning the skills necessary to acquire, appraise and apply evidence, students and trainees can also benefit from acquiring an awareness of ways to use this knowledge to bring about change at the organisational level [88]. A discussion of this is beyond the scope of this paper, however, desirable skills might include be-

³<https://www.gov.uk/government/news/building-evidence-into-education>

ing able to identify where changes to guidelines or to established practice are needed and where change would be worthwhile.

The importance of leadership in enhancing engagement with, and use of, research findings is also a key message from a recent study on evidence-informed teaching practice, published by the UK's Department of Education [89].

Viewed overall though, there seems to be little guidance available on how to provide advice for teachers about using empirical material such as the knowledge-set from these systematic reviews to support the way that software engineering is taught. Clearly, as such knowledge accumulates, this will present an increasingly important pedagogical research question to be pursued.

7.3. Limitations of this study

We can identify a number of limitations upon the outcomes from our tertiary study that stem from the way that we performed the various elements of the study. We discuss these here, together with any factors that may help to alleviate their effects.

1. One limitation is the way that we selected the secondary studies (*Dataset1* + *Dataset2*). Since we were performing a mapping study, we did not attempt to find all of the systematic reviews that were published during the period covered by our tertiary study, and confined ourselves to those reviews identified in the three broad tertiary studies and then the five software engineering journals, while explicitly excluding any studies published as conference papers. We did however conduct a broad electronic search as a check that we were not missing any significant source of systematic reviews, and we should observe that eight of the 11 reviews included in *Dataset1* were published in the five journals that we used in the later part of the search. Since we were investigating the use of systematic reviews in teaching, there was the possibility that relevant reviews could be found in educational journals related to software topics. A check of the papers published in ACM Transactions on Computer Education

(TOCE) and IEEE Transactions on Education (ToE) for the period 2004–2018 inclusive, identified only five systematic reviews. All of these were addressing pedagogical knowledge rather than “topic” knowledge and we could not identify any papers related to the use of evidence-based material in teaching.

2. In our original research protocol we selected a cut-off date for inclusion as the end of 2015. Because the processes of inclusion/exclusion and data extraction were complicated by the heterogenous nature of the selected set of systematic reviews, and as changes in circumstances also meant that two members of the team would not be available for this task, we felt that we could not ensure that any extension would be consistent with the original study, particularly regarding the interpretation for Inc-2 and Inc-3. As explained in §1 we also performed and published two other analyses on this dataset, further delaying the production of this paper. There is therefore the possibility that in the time following our cut-off date and submitting this paper, there may have been some changes in the way that systematic reviews have been reported, and obviously, new topics will have been covered. Informally, based upon our experiences over this period reviewing systematic reviews as well as performing some informal monitoring of journal contents, we have not observed any developments that would have significantly affected our findings. It is also possible that the balance of systematic reviews across the SEEK KAs might have changed. However, topics such as design and requirements elicitation still continue to present some real challenges to conducting rigorous primary studies [90], limiting the scope to perform systematic reviews for those KAs.
3. When calculating the DARE score for a review, our definition in Table 3 does not address the question of whether or not the search conducted by the reviewers was *adequate* for the purpose of the systematic review. While it would be desirable to make such an assessment, we did not feel our knowledge about the research areas related to the review

topics would allow us to do this in a consistent manner.

4. The selection process that we used to identify relevant systematic reviews did require an element of human interpretation, including for inclusion criterion Inc-2 (relevance to introductory teaching). We drew upon our experience of teaching software engineering to determine whether a review addressed a suitable topic, as well as using randomly allocated pairs of team members for all aspects of this part of the process, and discussing our decisions.
5. Further interpretation was required for the purpose of identifying the findings and recommendations embodied in a review (criterion Inc-3). The quality of reporting did not always assist with this [10], so as a check, we did seek to consult the original authors wherever possible. We received responses from approximately half of these, with no-one suggesting other than minor rewording or clarification, which suggests that we managed to perform this task fairly well.
6. Our supplementary data extractions were performed by two of the authors, partly to ensure consistency of interpretation. For our interpretation of the *synthesis* methods adopted in the 49 systematic reviews, we were able to check a proportion of our decisions against a baseline study [16].
7. For categorisation of the studies against the SEEK we again used two members of the team, to provide consistency in our allocations. Since many systematic reviews span different knowledge areas, this is very much an issue of interpretation, and we would certainly advise anyone seeking knowledge about a topic to check that whether it appears as an element in other studies (particularly those categorised as PRO).
8. Our informal assessment of how “useful” reviews were (summarised in Table 12) used a simple ranking procedure as described in Section 7.1. However, our individual assessments, as indicated by the different numbers of “y” rankings used by each assessor, were inevitably influenced both by our own teach-

ing experience (as we note) and also possibly by our familiarity with the topics of specific reviews.

9. We were unable to obtain assessments of the strength of evidence for the findings from most of the reviews. Where an assessment was available, the findings were generally rated as being based on low or moderate strength of evidence.

Hence there may be some variation in consistency between different elements of our overall dataset, particularly as regards the confidence that we can place in the findings from each systematic review.

7.4. The way ahead?

Conducting a tertiary study of this form requires quite extensive interpretation of the reported findings from a heterogeneous set of systematic reviews. So an obvious question is whether this knowledge, assuming it is considered to be useful to the community, can be extracted from the reports of systematic reviews by other (and better) means in the future. In particular it would be better if this avoided the need to perform studies such as this one that involve retrospective analysis, both because the distance from the original study means that much of the knowledge about how it was done may not be available, and also because the original systematic reviewers can be expected to possess greater expertise about the topic of a review, as well as being better able to assess the *quality* of the evidence [90].

A relatively simple and efficacious mechanism for enabling this does exist, and is already used in other disciplines [91]. In healthcare research where the needs of policy-making may go alongside those of practice, this consists of requiring that a systematic review and its findings are reported as a set of documents with different lengths and levels of abstraction, in order to meet different needs. The Canadian Health Services Research Foundation describe this as a *1-3-25* format, consisting of: a one-page summary of “take-home” messages; a three-page executive summary; and a more detailed report. Like any such mechanism this is not infallible

of course, and as Oliver and Dickson comment: “some teams were better than others at producing a policy-friendly report” [92].

Adapting this model to the needs of software engineering appears to be quite feasible. At its most simple, it would consist of requiring, as a condition of publication, that authors also provide a one-page summary of their findings, worded in a form that made them readily accessible to practitioners and students, and including an estimate of the strength of the supporting evidence. Appendix A provides two examples of a one-page summary to illustrate this concept. The first is a summary of this tertiary study, while the second is a summary of a systematic review from our set of 49, for which one of us was an author (#154). When used for healthcare reviews, the single page often consists of a brief summary of the purpose of a study followed by a set of bullets that summarise the key findings, and we have largely adopted this model. However, for teaching purposes this may need to be supplemented by a more effective visual structure such as the one proposed for *evidence briefings* [93], and our choice of layout has also been influenced by that model.

There are obviously a number of practical issues to address in creating such a mechanism (including obtaining the cooperation of journal editors). It would require reporting guidelines for authors (we already have a set of these in [10]); a means of checking that the summary was appropriate; and (preferably) some central means of indexing the summaries. But in exchange, adopting such a system has the potential to make it likely that future reviews had findings that were translated for practice by the people most familiar with the material. Prospective reviewers would also be able to check more easily if there was an existing systematic review addressing their planned topic.

8. Reflections and conclusions

Our tertiary mapping study identified 49 systematic reviews, published in the period 2004–2015, that contained findings and recommendations

considered to be useful for teaching about software engineering (RQ1). Within these, we were able to identify a smaller number that did provide guidance and information that could be used to help make “effective choices” (RQ2).

However, it is evident that useful findings are available from only a small proportion of the published systematic reviews that we surveyed. There may be many reasons why this is so: one of which may simply be that in software engineering the role of the systematic review has so far been mainly to be used as a tool to aid research and to provide a useful training exercise and preparation for PhD students.

This underlying emphasis upon research may also explain many of the quality issues that have been identified regarding the conduct and reporting of systematic reviews in software engineering. Some may well arise because there is therefore no requirement to report to an external sponsor, others because the reviews are sometimes conducted by relatively inexperienced researchers. In contrast, other disciplines tend to use information specialists to undertake much of the work involved in searching and selecting material [15].

Following on from these conclusions, empirical researchers and others might wish to consider how researchers can better provide information about the outcomes from systematic reviews, so that this is of greater use to others. From this study, and from the other analyses we have performed upon our data, we can suggest three mechanisms that could contribute towards achieving this aim.

1. Providing better reporting of the *conduct* of a systematic review. In our analysis of reporting quality [10] we identify 12 lessons about reporting, and suggest a checklist that should be used by reviewers (and authors). Better reporting can help to establish the *provenance* for the findings from a review, and so help justify its publication.
2. Facilitating better reporting of the *findings* from a study. In part this overlaps with item 1 above, in that the reporting of a review should make its findings clear. This was only the case for fewer than one in five of the 276 systematic reviews that we examined, and even

for the 49 included in the final set, we often found it difficult to extract the findings and recommendations, as these were sometimes spread over different sections of a paper. In addition, as discussed in the previous section, making the provision of a summary of findings a pre-requisite for publication will also help to make the findings more widely and readily available to others. This is clearly a concern that is also shared by other disciplines, hence the emphasis upon such mechanisms as the 1-3-25 model.

3. Creating the means to provide effective *currency* of the knowledge about and from systematic reviews, particularly as the number of these increases. As a newcomer to the use of systematic reviews, software engineering has so far not embraced the idea of creating anything equivalent to the Cochrane and Campbell collaborations that oversee and facilitate the conduct of systematic reviews in clinical medicine and social science respectively. These bodies play a number of roles, including providing public information about relevant findings from systematic reviews.

We see the first two mechanisms as needing to be adopted in collaboration with those journals that publish systematic reviews. And all three may need the involvement of the professional bodies. What is clear from our findings though, is that without such interventions, systematic reviews in software engineering will very likely remain a tool used mainly for academic research rather than, as in other disciplines, forming a valuable (and valued) source of knowledge for software developers, teachers, and researchers.

Acknowledgements

Our thanks to Professor Barbara Kitchenham for advice and observations, as well as for conducting independent electronic searching for publications that we might have missed. We would like to thank the authors of the systematic reviews we studied, especially those who were good enough to check the accuracy of our extracted conclusions and to pass comment on these. We are also

grateful to Professor Steve Higgins for informing us about current ideas about evidence-based teaching in education. We also thank the anonymous reviewers for their comments and added insight.

References

- [1] P. Naur and B. Randell, Eds., *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee*. NATO, 1968.
- [2] P. Bourque and R.E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd ed. IEEE Computer Society Press, 2014.
- [3] M. Ardis, D. Budgen, G.W. Hislop, J. Offutt, M. Sebern, and W. Visser, "SE2014: Curriculum Guidelines for undergraduate degree programs in software engineering," *IEEE Computer*, November 2015, pp. 106–109.
- [4] B. Kitchenham, D. Budgen, P. Brereton, M. Turner, S. Charters, and S. Linkman, "Large-Scale Software Engineering Questions – expert opinion or empirical evidence?" *IET Software*, Vol. 1, No. 5, 2007, pp. 161–171.
- [5] P. Devanbu, T. Zimmermann, and C. Bird, "Belief and evidence in empirical software engineering," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. ACM Press, 2016, pp. 108–119.
- [6] E.M. Rogers, *Diffusion of Innovations*, 5th ed. Free Press, New York, 2003.
- [7] C. Theisen, M. Dunaiski, L. Williams, and W. Visser, "Software engineering research at the international conference on software engineering in 2016," *ACM Software Engineering Notes*, Vol. 42, No. 4, 2017, pp. 1–10.
- [8] B. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-based software engineering," in *Proceedings of the 26th International Conference on Software Engineering (ICSE)*. IEEE Computer Society Press, 2004, pp. 273–281.
- [9] D. Budgen, S. Drummond, P. Brereton, and N. Holland, "What scope is there for adopting evidence-informed teaching in software engineering?" in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*. IEEE Computer Society Press, 2012, pp. 1205–1214.
- [10] D. Budgen, P. Brereton, S. Drummond, and N. Williams, "Reporting systematic reviews: Some lessons from a tertiary study," *Information and Software Technology*, Vol. 95, 2018,

- pp. 62–74. [Online]. <http://www.sciencedirect.com/science/article/pii/S0950584916303548>
- [11] D. Budgen, P. Brereton, N. Williams, and S. Drummond, “The contribution that empirical studies performed in industry make to the findings of systematic reviews: A tertiary study,” *Information and Software Technology*, Vol. 94, 2018, pp. 234–244. [Online]. <http://www.sciencedirect.com/science/article/pii/S0950584917303798>
 - [12] J. Gurevitch, J. Koricheva, S. Nakagawa, and G. Stewart, “Meta-analysis and the science of research synthesis,” *Nature*, Vol. 555, 2018, pp. 175–182.
 - [13] E. Barends and D.M. Rousseau, *Evidence-Based Management: How to use evidence to make better organizational decisions*. Kogan Page, 2018.
 - [14] M. Petticrew and H. Roberts, *Systematic Reviews in the Social Sciences A Practical Guide*. Blackwell Publishing, 2006.
 - [15] A. Booth, D. Papaioannou, and A. Sutton, *Systematic Approaches to a Successful Literature Review*. Sage Publications, Ltd., 2012.
 - [16] D.S. Cruzes and T. Dybå, “Research synthesis in software engineering: A tertiary study,” *Information and Software Technology*, Vol. 53, No. 5, 2011, pp. 440–455.
 - [17] S. Martinez-Fernandez, P.S.M. dos Santos, G.P. Ayala, X. Franch, and G.H. Travassos, “Aggregating empirical evidence about the benefits and drawbacks of software reference architectures,” in *Proceedings of 2015 the Conference on Empirical Software Engineering and Measurement*, 2015, pp. 154–163.
 - [18] D. Budgen, J. Bailey, M. Turner, B. Kitchenham, P. Brereton, and S. Charters, “Cross-domain investigation of empirical practices,” *IET Software*, Vol. 3, No. 5, 2009, pp. 410–421, eASE special section.
 - [19] T.V. Ribeiro, J. Massollar, and G.H. Travassos, “Challenges and pitfalls on surveying evidence in the software engineering technical literature: an exploratory study with novices,” *Empirical Software Engineering*, Vol. 23, 2018, pp. 1594–1663.
 - [20] U. Abelein and B. Paech, “Understanding the influence of user participation and involvement on system success – A systematic mapping study,” *Empirical Software Engineering*, Vol. 20, 2015, pp. 28–81.
 - [21] D. Smite, C. Wohlin, T. Gorschek, and R. Feldt, “Empirical evidence in global software engineering: a systematic review,” *Empirical Software Engineering*, Vol. 15, 2010, pp. 91–118.
 - [22] O. Dieste and N. Juristo, “Systematic review and aggregation of empirical studies on elicitation techniques,” *IEEE Transactions on Software Engineering*, Vol. 37, No. 2, 2011, pp. 283–304.
 - [23] T. Dybå and T. Dingsøy, “Strength of evidence in systematic reviews in software engineering,” in *Proceedings of International Symposium on Empirical Software Engineering and Metrics (ESEM)*, 2008, pp. 178–187.
 - [24] M. Ivarsson and T. Gorschek, “A method for evaluating rigor and industrial relevance of technology evaluations,” *Empirical Software Engineering*, Vol. 16, 2011, pp. 365–395.
 - [25] B. Kitchenham, P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, “Systematic literature reviews in software engineering – a systematic literature review,” *Information and Software Technology*, Vol. 51, No. 1, 2009, pp. 7–15.
 - [26] B. Kitchenham, R. Pretorius, D. Budgen, P. Brereton, M. Turner, M. Niazi, and S. Linkman, “Systematic literature reviews in software engineering – a tertiary study,” *Information and Software Technology*, Vol. 52, 2010, pp. 792–805.
 - [27] F.Q. da Silva, A.L. Santos, S. Soares, A.C.C. França, C.V. Monteiro, and F.F. Maciel, “Six years of systematic literature reviews in software engineering: An updated tertiary study,” *Information and Software Technology*, Vol. 53, No. 9, 2011, pp. 899–913.
 - [28] M. Leitner and S. Rinderle-Ma, “A systematic review on security in process-aware information systems,” *Information and Software Technology*, Vol. 56, No. 3, 2014, pp. 273–293.
 - [29] B.A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-Based Software Engineering and Systematic Reviews*, Innovations in Software Engineering and Software Development. CRC Press, 2015.
 - [30] T. Dybå, B. Kitchenham, and M. Jørgensen, “Evidence-based software engineering for practitioners,” *IEEE Software*, Vol. 22, No. 1, 2005, pp. 58–65.
 - [31] B. Kitchenham, “Procedures for undertaking systematic reviews,” Joint Technical Report Keele and Durham Universities, Tech. Rep., 2004.
 - [32] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” Keele University and Durham University Joint Report, Tech. Rep., 2007.
 - [33] P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, “Lessons from applying the systematic literature review process within the software engineering domain,” *Journal of Systems and Software*, Vol. 80, No. 4, 2007, pp. 571–583.

- [34] J. Biolchini, P. Mian, A. Natali, and G. Travassos, "Systematic review in software engineering," COPPE/UFRJ, Tech. Rep. ES679/05, 2005.
- [35] J.L. Fleiss, "Measuring nominal scale agreement among many raters," *Psychological Bulletin*, Vol. 76, 1971, pp. 378–382.
- [36] M. Banerjee, M. Capozzoli, L. McSweeney, and D. Sinha, "Beyond kappa: A review of interrater agreement measures," *Canadian Journal of Statistics*, Vol. 27, No. 1, 1999, pp. 3–23.
- [37] M. Staples and M. Niazi, "Systematic review of organizational motivations for adopting CMM-based SPI," *Information and Software Technology*, Vol. 50, 2008, pp. 605–620.
- [38] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in software engineering: A systematic literature review," *Information and Software Technology*, Vol. 50, No. 9–10, 2008, pp. 860–878.
- [39] H. Sharp, N. Baddoo, S. Beecham, T. Hall, and H. Robinson, "Models of motivation in software engineering," *Information and Software Technology*, Vol. 51, 2009, pp. 219–233.
- [40] H. Petersson, T. Thelin, P. Runeson, and C. Wohlin, "Capture-recapture in software inspections after 10 years research – theory, evaluation and application," *Journal of Systems and Software*, Vol. 72, 2004, pp. 249–264.
- [41] W. Azfal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Information and Software Technology*, Vol. 51, 2009, pp. 957–976.
- [42] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, Vol. 52, 2010, pp. 14–30.
- [43] M. Jørgensen, "Forecasting of software development work effort: Evidence on expert judgement and formal models," *Int. Journal of Forecasting*, Vol. 23, No. 3, 2007, pp. 449–462.
- [44] M. Jørgensen, "Evidence-based guidelines for assessment of software development cost uncertainty," *IEEE Transactions on Software Engineering*, Vol. 31, No. 11, 2005, pp. 942–954.
- [45] P. Mohagheghi and R. Conradi, "Quality, productivity and economic benefits of software reuse: A review of industrial studies," *Empirical Software Engineering*, Vol. 12, 2007, pp. 471–516.
- [46] F.J. Pino, F. Garcia, and M. Piattini, "Software process improvement in small and medium software enterprises: A systematic review," *Software Quality Journal*, Vol. 16, 2008, pp. 237–261.
- [47] J. Hannay, T. Dybå, E. Arisholm, and D. Sjøberg, "The effectiveness of pair programming. A meta analysis," *Information and Software Technology*, Vol. 51, No. 7, 2009, pp. 1110–1122.
- [48] J.S. Persson, L. Mathiassen, J. Boeg, T.S. Madson, and F. Steinson, "Managing risks in distributed software projects: An integrative framework," *IEEE Transactions on Engineering Management*, Vol. 56, No. 3, 2009, pp. 508–532.
- [49] A.H. Ghapanchi and A. Aurum, "Antecedents to IT personnel's intentions to leave: A systematic literature review," *Journal of Systems and Software*, Vol. 84, 2011, pp. 238–249.
- [50] I. Steinmacher, M.A.G. Silva, M.A. Gerosa, and D.F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Information and Software Technology*, Vol. 59, No. 67–85, 2015.
- [51] Z. Li, H. Zhang, L. O'Brien, R. Cai, and S. Flint, "On evaluating commercial cloud services: A systematic review," *Journal of Systems and Software*, Vol. 86, 2013, pp. 2371–2393.
- [52] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, Vol. 55, 2013, pp. 1397–1418.
- [53] H. Munir, M. Moayyed, and K. Peterson, "Considering rigor and relevance when evaluating test driven development: A systematic review," *Information and Software Technology*, Vol. 56, 2014, pp. 375–394.
- [54] E. Kupiainen, M.V. Mäntylä, and J. Itkonen, "Using metrics in agile and lean software development – a systematic literature review of industrial studies," *Information and Software Technology*, Vol. 62, 2015, pp. 143–163.
- [55] B.J. Williams and J.C. Carver, "Characterizing software architecture changes: A systematic review," *Information and Software Technology*, Vol. 52, No. 1, 2010, pp. 31–51.
- [56] M.S. Ali, M.A. Babar, L. Chen, and K.J. Stol, "A systematic review of comparative evidence of aspect-oriented programming," *Information and Software Technology*, Vol. 52, No. 9, 2010, pp. 871–887.
- [57] C. Zhang and D. Budgen, "What do we know about the effectiveness of software design patterns?" *IEEE Transactions on Software Engineering*, Vol. 38, No. 5, 2012, pp. 1213–1231.
- [58] L.B. Lisboa, V.C. Garcia, D. Lucrédio, E.S. de Almeida, S.R. de Lemos Meira, and R.P. de Mattos Fortes, "A systematic review of domain analysis tools," *Information and Software Technology*, Vol. 52, No. 1, 2010, pp. 1–13.

- [59] M. Turner, B. Kitchenham, P. Brereton, S. Charters, and D. Budgen, "Does the technology acceptance model predict actual use? A systematic literature review," *Information and Software Technology*, Vol. 52, No. 5, 2010, pp. 463–479.
- [60] T.B.C. Arias, P. van der Spek, and P. Avgeriou, "A practice-driven systematic review of dependency analysis solutions," *Empirical Software Engineering*, Vol. 16, 2011, pp. 544–586.
- [61] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, Vol. 38, No. 6, 2012, pp. 1276–1304.
- [62] C. Pacheco and I. Garcia, "A systematic literature review of stakeholder identification methods in requirements elicitation," *Journal of Systems and Software*, Vol. 85, 2012, pp. 2171–2181.
- [63] S. Tiwari and A. Gupta, "A systematic literature review of use case specifications research," *Information and Software Technology*, Vol. 67, 2015, pp. 128–158.
- [64] R. Jabangwe, J. Borstler, D. Smite, and C. Wohlin, "Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review," *Empirical Software Engineering*, Vol. 20, 2015, pp. 640–693.
- [65] K. Peterson, "Measuring and predicting software productivity: A systematic map and review," *Information and Software Technology*, Vol. 53, 2011, pp. 317–343.
- [66] J. Díaz, J. Pérez, P.P. Alarcón, and J. Garbajosa, "Agile product line engineering – A systematic literature review," *Software – Practice and Experience*, Vol. 41, 2011, pp. 921–941.
- [67] Y. Rafique and V. Mistic, "The effects of test-driven development on external quality and productivity: A meta-analysis," *IEEE Transactions on Software Engineering*, Vol. 39, No. 6, 2013.
- [68] A.M. Magdaleno, C.M.L. Werner, and R.M. de Araujo, "Reconciling software development models: A quasi-systematic review," *Journal of Systems and Software*, Vol. 85, 2012, pp. 351–369.
- [69] N.B. Ali, K. Peterson, and C. Wohlin, "A systematic literature review on the industrial use of software process simulation," *Journal of Systems and Software*, Vol. 97, 2014, pp. 65–85.
- [70] S.U. Khan, M. Niazi, and R. Ahmad, "Barriers in the selection of offshore software development outsourcing vendors: An exploratory study using a systematic literature review," *Information and Software Technology*, Vol. 53, 2011, pp. 693–706.
- [71] R. Giuffrida and Y. Dittrich, "Empirical studies on the use of social software in global software development – A systematic mapping study," *Information and Software Technology*, Vol. 55, 2013, pp. 1143–1164.
- [72] N. Paternoster, C. Giardino, M. Unterkalmsteiner, and T. Gorschek, "Software development in startup companies: A systematic mapping study," *Information and Software Technology*, Vol. 56, 2014, pp. 1200–1218.
- [73] O. Al-Baik and J. Miller, "The Kanban approach between agility and leanness: a systematic review," *Empirical Software Engineering*, Vol. 20, 2015, pp. 1861–1897.
- [74] M. Zarour, A. Abran, J.M. Desharnais, and A. Alarifi, "An investigation into the best practices for the successful design and implementation of lightweight software process assessment methods: A systematic literature review," *Journal of Systems and Software*, Vol. 101, 2015, pp. 180–192.
- [75] A. Nguyen-Duc, D.S. Cruzes, and R. Conradi, "The impact of global dispersion on coordination, team performance and software quality – A systematic literature review," *Information and Software Technology*, Vol. 57, 2015, pp. 277–294.
- [76] F.S. Silva, F.S.F. Soares, A.L. Peres, I.M. de Azevedo, A.P.L.F. Vasconcelos, F.K. Kamei, and S.R. de Lemos Meira, "Using CMMI together with agile software development: A systematic review," *Information and Software Technology*, Vol. 58, No. 20–43, 2015.
- [77] M. Bano and D. Zowghi, "A systematic review on the relationship between user involvement and system success," *Information and Software Technology*, Vol. 58, No. 148–169, 2015.
- [78] A. Idri, F.A. Amazal, and A. Abran, "Analogy-based software development effort estimation: A systematic mapping and review," *Information and Software Technology*, Vol. 58, 2015, pp. 206–230.
- [79] M. Brhel, H. Meth, A. Maedche, and K. Werder, "Exploring principles of user-centered agile software development: A literature review," *Information and Software Technology*, Vol. 61, 2015, pp. 163–181.
- [80] D. Heaton and J.C. Carver, "Claims about the use of software engineering practices in science: A systematic literature review," *Information and Software Technology*, Vol. 67, 2015, pp. 207–219.
- [81] R. Rabiser, P. Grunbacher, and D. Dhungana, "Requirements for product derivation support: Results from a systematic literature review and an expert survey," *Information and Software Technology*, Vol. 52, 2010, pp. 324–346.
- [82] E. Tüzün, B. Tekinerdogan, M.E. Kalender, and S. Bilgen, "Empirical evaluation of a decision support model for adopting software product line

- engineering,” *Information and Software Technology*, Vol. 60, 2015, pp. 77–101.
- [83] H.A. Simon, “The structure of ill-structured problems,” *Artificial Intelligence*, Vol. 4, 1973, pp. 181–201.
- [84] G.H. Guyatt, A.D. Oxman, G.E. Vist, R. Kunz, Y. Falck-Ytter, P. Alonso-Coello, and H.J. Schünemann, “GRADE: an emerging consensus on rating quality of evidence and strength of recommendations,” *British Medical Journal*, Vol. 336, 2008, pp. 924–926.
- [85] J. Nelson and C. O’Beirne, “Using evidence in the classroom: What works and why?” National Foundation for Educational Research (NFER), Tech. Rep., 2014.
- [86] S. Hopewell, A. Aisinga, and M. Clarke, “Better reporting of randomized trials in biomedical journal and conference abstracts,” *Journal of Information Science*, Vol. 34, No. 2, 2008, pp. 162–173.
- [87] S.E. Rosenbaum, C. Glenton, and A.D. Oxman, “Summary-of-findings tables in Cochrane reviews improved understanding and rapid retrieval of key information,” *Journal of Clinical Epidemiology*, Vol. 63, 2010, pp. 620–626.
- [88] S. Malick, K. Das, and K.S. Khan, “Tips for teaching evidence-based medicine in a clinical setting: Lessons from adult learning theory,” *Journal of the Royal Society of Medicine*, Vol. 101, No. 11, 2008, pp. 536–543.
- [89] M. Coldwell, T. Greany, S. Higgins, C. Brown, B. Maxwell, B. Stiell, L. Stoll, B. Willis, and H. Burns, “Evidence-informed teaching: an evaluation of progress in England,” Department for Education, Tech. Rep., 2017.
- [90] C.L. Goues, C. Jaspan, I. Ozkaya, M. Shaw, and K.T. Stolee, “Bridging the Gap: From research to practical advice,” *IEEE Software*, Vol. 35, No. 5, 2018, pp. 50–57.
- [91] J. Lavis, G. Permanand, A. Oxman, S. Lewin, and A. Fredheim, “SUPPORT tools for evidence-informed health policy-making (STP) 13: Preparing and using policy briefs to support evidence-informed policymaking,” *Health Research Policy and Systems*, Vol. 7, 2009, p. S13.
- [92] S. Oliver and K. Dickson, “Policy-relevant systematic reviews to strengthen health systems: models and mechanisms to support their production,” *Evidence and Policy*, Vol. 12, No. 2, 2016, pp. 235–259.
- [93] B. Cartaxo, G. Pinto, E. Vieira, and S. Soares, “Evidence Briefings: Towards a medium to transfer knowledge from systematic reviews to practitioners,” in *Proceedings of the 2016 Conference on Empirical Software Engineering and Measurement (ESEM)*, 2016, pp. 1–10.

Appendix A. Examples of a one-page summary

<p>What support do systematic reviews provide for evidence-informed teaching about software engineering practice? – Implications & Messages</p> <p>Implications</p> <p>Systematic reviews provide a rigorous way of gathering together evidence obtained from empirical studies. Since 2004 systematic reviews have been used quite extensively by software engineering researchers to examine a range of software engineering practices and the use of different technologies.</p> <p>The findings from a systematic review provide objective and unbiased knowledge about using a practice, that can underpin advice to practitioners, teachers and students, and which can help them assess the likely benefits of adopting it in a particular context.</p> <p>Key Messages</p> <ul style="list-style-type: none"> • Systematic reviews can provide useful guidance for practice and for teaching about practice that can take a range of forms, including: <ul style="list-style-type: none"> ▪ a <i>digest</i> of the experiences of others (for example, related to adopting a new practice such as agile development); ▪ a checklist of the <i>factors</i> that should be considered when thinking of adopting a new practice or technique; ▪ <i>comparisons</i> between different options, such as occur when identifying the most dependably effective practice to use for requirements elicitation. • Much of the guidance and knowledge provided by the systematic reviews was derived from primary studies that involved observing how practising software engineers performed tasks 'in the field'. • Researchers need to provide their findings in a more 'end-user-friendly' form (such as by using a one-page summary like this one) that also explains what the implications of the findings are. This will help teachers, students and practitioners to identify those messages that are useful to them. • A characteristic of software engineering is that, unlike other disciplines, topics for study using a systematic review are chosen by researchers themselves, rather than being selected to meet the needs of practitioners, policy-makers or funding agencies. • There is a need to provide readily-available indexing of the findings from systematic reviews to assist end-users with finding material that they need. This would also help researchers to identify where new systematic reviews, or updating of existing ones, would be useful. We suggest that this is a role that the professional bodies such as ACM could assist with, working in collaboration with journal editors. 	<p>Characteristics of our systematic review</p> <p>From 276 candidate systematic reviews published up to the end of 2015 we selected 49 that provide knowledge that we considered useful for teaching and practice. For each of these we describe:</p> <ul style="list-style-type: none"> • the topic; • The number of primary studies used (and the types of these, when known); • how the outcomes from the primary studies were synthesised; • key findings relevant to teaching and practice.
---	--

Figure A1. Example 1, summarising this tertiary study

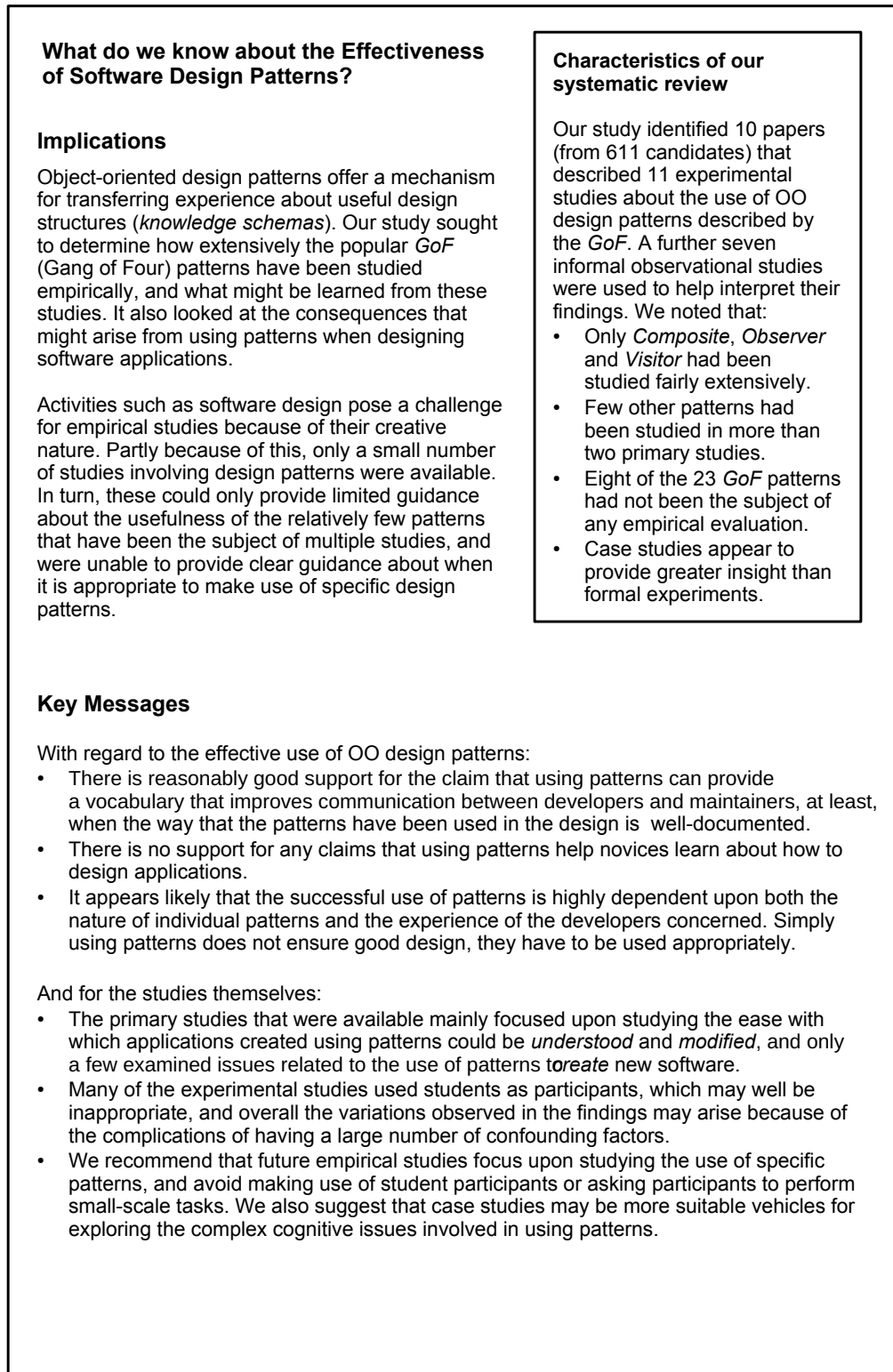


Figure A2. Example 2, summarising paper #154

Appendix B. The findings and recommendations from the reviews

Table B1. Details for Review categorised as FND: #52

Characteristic	Values
1. Knowledge Unit	FND.ec (Engineering Economics for software)
2. Title	Systematic Review of Organizational Motivations for Adopting CMM-based SPI
3. Citation	[37]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made. (Gives counts of studies that identify different reasons.)
6. #Primary Studies	49 (all explicit industry)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	<ol style="list-style-type: none"> 1. Organisations adopted CMM based SPI mainly to improve product quality and project performance but also to improve process management. 2. Satisfying customers was not a common reason for adopting CMM-based SPI. 3. The two most common process related reasons for adopting SPI were to make processes more visible and measurable.
9. Recommendations	None
10. Author Response	The authors observed that meeting “customer demands” in the form of contractual requirement was a fairly major reason for adoption, rather than “customer satisfaction”. They also observed that providing <i>assurance</i> for customers through high ratings was a legitimate reason for recommending the adoption of CMM(I).

Table B2. Details for Reviews categorised as PRF: #54

Characteristic	Values
1. Knowledge Unit	(no specific KU)
2. Title	Motivation in Software Engineering: A systematic literature review
3. Citation	[38]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made. (Lists studies identifying specific motivators.)
6. #Primary Studies	79 (not described)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	<ol style="list-style-type: none"> 1. The most frequently cited motivators relate to the “need to identify with the task” (clear goals, personal interest, understanding the purpose of a task, how it fits with the whole, job satisfaction, and working on an identifiable piece of quality work). Having a clear career path and a variety of tasks is also found motivating. 2. Learning, exploring new techniques and problem solving appear to be motivating aspects of SE. 3. Indicators of demotivation were mainly turnover and absenteeism. 4. Key de-motivators are poor working conditions and lack of resources
9. Recommendations	None
10. Author Response	None

Table B3. Details for Reviews categorised as PRF: #118

Characteristic	Values
1. Knowledge Unit	(no specific KU)
2. Title	Models of motivation in software engineering
3. Citation	[39]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made. (See note for #54.)
6. #Primary Studies	Same as #54 above.
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. A list of 21 motivators is provided in the paper. 2. A new model of motivation in SE (the MOCC model) is presented using the results from the review reported in detail in paper #54)
9. Recommendations	None
10. Author Response	None

Table B4. Details for Reviews categorised as PRF: #135

Characteristic	Values
1. Knowledge Unit	PRF.psy (Group dynamics and psychology)
2. Title	Antecedents to IT personnel's intentions to leave: A systematic literature review
3. Citation	[49]
4. DARE Score	3.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	72 (all implicitly using industry participants)
7. Synthesis used	Thematic Analysis
8. Findings	"Publications reviewed suggest that male IT workers are more likely to leave an organisation than their female counterparts. Younger employees also appear more inclined to leave (mainly due to lower job satisfaction) compared to their older counterparts). Importantly, higher educated IT professionals are more likely to leave a company because of low job satisfaction. Additionally, married IT practitioners as well as those with a lower organisational tenure have a lower tendency to leave an organisation. IT managers can use these insights to assist with their recruitment decisions and employee retention initiatives."
9. Recommendations	1. To overcome role ambiguity and role conflict, managers should: a. communicate clearly and provide clear and precise information about what they expect from their IT professionals. b. make sure that their personnel have the required training and knowledge to carry out their jobs well. c. allow their IT professionals to know the intent of and reasons for doing a specific task. d. better design and define tasks so that the start and end of each task is clear. e. clearly define the sequence in which sub-tasks are carried out. 6. determine task priorities associated with the job. 2. To overcome perceived workload demands managers should maintain an awareness of the workloads of their high valued IT professionals. Direct face-to-face communications has been reported as the most effective means of overcoming this problem. 3. IT managers should be conscious of the benefits of enhanced employee autonomy because lack of autonomy can lead to turnover decision through work exhaustion. Managers should provide IT professionals with enough autonomy and flexibility to reduce exhaustion they might feel because of the structure of their work and should design IT roles that offer enough freedom for IT professional to be innovative and pursue their own thoughts and ideas.

Table B4 continued

Characteristic	Values
	<i>(plus five other recommendations, omitted for reasons of space)</i>
10. Author Response	None

Table B5. Details for Reviews categorised as PRF: #246

Characteristic	Values
1. Knowledge Unit	PRF.psy (Group dynamics and psychology)
2. Title	A systematic literature review on the barriers faced by newcomers to open source software projects
3. Citation	[50]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made. (Lists studies identifying specific barriers.)
6. #Primary Studies	20 (implicitly drawn from industry)
7. Synthesis used	Grounded Theory
8. Findings	1. For projects, improvement in community receptivity and more appropriate collaborative environments for OSS development can result in better support for newcomers. 2. "Keeping the code simple and the documentation organized and up-to-date could potentially increase the odds of receiving contributions from newcomers."
9. Recommendations	"..newcomers that wish to contribute must have a blend of domain knowledge, technical skills, and social interaction, which can increase the odds of a successful joining. The interactions are driven by artifacts that reflect the technical and domain expertise. It is the result of these interactions that will allow both newcomers and developers to perceive the level and possibly lack of background that hinders effective contributions to the project."
10. Author Response	None

Table B6. Details for Reviews categorised as VAV: #15

Characteristic	Values
1. Knowledge Unit	VAV.rev (Reviews and static analysis)
2. Title	Capture-recapture in software inspections after 10 years research – theory, evaluation and application
3. Citation	[40]
4. DARE Score	1.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	25 (1 explicitly from industry, the others not identified)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. Most estimators underestimate. 2. Mh-JK (Jackknife) is the best estimator for software inspections. 3. Mh-JK is appropriate to use for 4 reviewers and more. 4. DPM is the best curve fitting method. 5. Capture-recapture estimators can be used together with perspective-based reasoning (PBR).
9. Recommendations	None
10. Author Response	None

Table B7. Details for Reviews categorised as VAV: #66

Characteristic	Values
1. Knowledge Unit	VAV.tst (Testing)
2. Title	A systematic review of search-based testing for non-functional system properties
3. Citation	[41]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	35 (17 explicitly from industry; 18 academic)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. For performance, genetic algorithms (GAs) consistently outperform random and statistical testing in a wide variety of situations, producing comparatively longer execution times faster and also finding new bounds on best case execution times. 2. GAs were also able to perform better than human testers, and where this failed to occur, it could be attributed to the complexity of the test objects inhibiting evolutionary testability.
9. Recommendations	None
10. Author Response	None

Table B8. Details for Reviews categorised as VAV: #82

Characteristic	Values
1. Knowledge Unit	VAV.tst (Testing)
2. Title	A systematic review on regression test selection techniques
3. Citation	[42]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	36 (4 explicitly from industry; 32 not stated)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. The minimization technique is the most efficient in reducing time and/or number of test cases to run. However this is an unsafe technique and all but one of six studies report significant losses in fault detection. 2. DejaVu (Rothermel and Harrold) is the most efficient safe technique for reducing test cases. However, analysis time for this is shown to be too long (exceeding the time for rerunning all test cases) in early experiments, although in later ones (using different subject programs) it is shown to be good. 3. Regression test selection techniques have to be tailored to specific situations e.g. initially based on the classification of techniques.
9. Recommendations	None
10. Author Response	None

Table B9. Details for Reviews categorised as VAV: #167

Characteristic	Values
1. Knowledge Unit	VAV.fnd (VAV terminology and foundations)
2. Title	On evaluating commercial Cloud services: A systematic review
3. Citation	[51]
4. DARE Score	4.0

Table B9 continued

Characteristic	Values
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	82 (all implicitly from industry)
7. Synthesis used	Could not be identified
8. Findings	<ol style="list-style-type: none"> Existing evaluations have used a large number of metrics to measure performance as well as cost. There is still a lack of metrics for evaluating Cloud elasticity. There are still no metrics that can be used to assess security.
9. Recommendations	None
10. Author Response	<p>The authors suggest a further finding is:</p> <ol style="list-style-type: none"> Various traditional benchmarks have been employed to evaluate performance of Cloud services.

Table B10. Details for Reviews categorised as VAV: #197

Characteristic	Values
1. Knowledge Unit	VAV.fnd (VAV terminology and foundations)
2. Title	Software fault prediction metrics: A systematic literature review
3. Citation	[52]
4. DARE Score	4.5
5. Strength of Evidence	An informal assessment was provided.
6. #Primary Studies	106 (81 explicitly from industry; 25 academic studies)
7. Synthesis used	Vote Counting
8. Findings	<ol style="list-style-type: none"> Cyclomatic complexity was fairly effective in large and OO environments. Although not effective in all categories, the overall effectiveness was estimated as moderate. Halstead's metrics were ineffective when compared with other metrics and were estimated as inappropriate for software fault prediction. The most frequently used and most successful among the OO metrics were the CK metrics. From these, COB, WMC and RFC were effective across all groups. LCOM is not not very successful at finding faults, DIT and NOC were reported as untrustworthy. OO and process metrics are more successful at fault prediction than traditional size and complexity metrics. Source code metrics do not perform well in finding post-release faults. Process metrics were found to be successful at finding post-release faults. Size measures like LOC metrics are simple and easy to extract; but as with complexity metrics, have only limited predictive capabilities. They are partly successful at ranking the most fault prone modules, not the most reliable or successful metrics.
9. Recommendations	<ol style="list-style-type: none"> Industry practitioners looking for effective and reliable process metrics should consider code churn, the number of changes, the age of a module and the change set size metrics. Not all OO metrics are good predictors of faults. NOC and DIT are unreliable and should not be used in fault prediction models. Industry practitioners looking for effective and reliable process metrics (in large post-release systems) could also try static code metrics (e.g. CBO, RFC and WMC) but should keep in mind that they have some limitations in highly iterative and agile development environments.
10. Author Response	None

Table B11. Details for Reviews categorised as VAV: #205

Characteristic	Values
1. Knowledge Unit	VAV.fnd (VAV terminology and foundations)
2. Title	Considering rigor and relevance when evaluating test driven development: A systematic review
3. Citation	[53]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	41 (22 explicitly from industry; 19 academic studies)
7. Synthesis used	Vote Counting
8. Findings	<p>1. Studies with high rigour and relevance indicate that practitioners wanting to adopt TDD will improve their code quality and at the same time maintain or reduce their productivity levels.</p> <p>2. Studies with high rigour and relevance indicate that practitioners wanting to adopt TDD will reduce complexity and at the same time maintain or reduce their productivity levels.</p> <p>3. Studies with high relevance and low rigour suggest that there is a potential to increase external quality, but at the expense of development time and productivity.</p>
9. Recommendations	None
10. Author Response	None

Table B12. Details for Reviews categorised as VAV: #252

Characteristic	Values
1. Knowledge Unit	VAV.fnd (VAV terminology and foundations)
2. Title	Using metrics in Agile and Lean Software Development – A systematic literature review of industrial studies
3. Citation	[54]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	30 (all explicitly from industry)
7. Synthesis used	Thematic Analysis
8. Findings	<p>1. The targets of measurement are the product and the process, but not the people.</p> <p>2. Documentation is not measured, instead the focus is on the actual product and features.</p> <p>3. The use of metrics can motivate people and change the way that people behave in terms of which issues they pay attention to.</p> <p>4. Industrial agile teams use situative metrics based on need.</p> <p>5. Defect counts and customer satisfaction are two of the four high influence metrics (after velocity and effort estimate), although not directly recommended by Lean or Agile methods.</p> <p>6. Areas where metrics are used are sprint and project planning, sprint and project progress tracking, understanding and improving quality, fixing software process problems and motivating people - so not dissimilar to use in plan driven.</p>
9. Recommendations	None
10. Author Response	None

Table B13. Details for Review categorised as DES: #124

Characteristic	Values
1. Knowledge Unit	DES.ar (Architectural design)
2. Title	Characterising software architecture changes: A systematic review
3. Citation	[55]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	130 (not stated)
7. Synthesis used	Thematic Analysis
8. Findings	The Software Architecture Change Characterization Scheme (SACCS) was developed as a result of the review and can (could?) be used to assist developers and maintainers in assessing the potential impact of a proposed change and deciding whether it is feasible to implement the change. Where the change is crucial the scheme will (could?) help generate consensus on how to approach change implementation and provide an indication of the difficulty.
9. Recommendations	None
10. Author Response	The authors reviewed our analysis and agreed with it.

Table B14. Details for Review categorised as DES: #130

Characteristic	Values
1. Knowledge Unit	DES.str (Design strategies)
2. Title	A systematic review of comparative evidence of aspect-oriented programming
3. Citation	[56]
4. DARE Score	4.5
5. Strength of Evidence	The GRADE system was used. Overall the current strength of evidence about benefits and limitations of AOP approaches compared to non-AOP approaches is <i>low</i> .
6. #Primary Studies	22 (6 implicitly industry studies, 16 academic studies)
7. Synthesis used	Narrative synthesis + Vote counting
8. Findings	<ol style="list-style-type: none"> 1. Overall AOP provides improvement over non-AOP based solutions. 2. AOP has a positive effect on performance (within contexts similar to those used in the evaluations). 3. In larger systems where concern scattering and tangling is expected to be widespread, introducing aspects is likely to significantly reduce the number of lines of code. 4. AOP has a positive effect on modularity (but context of use should be carefully assessed). 5. AOP has the potential to develop evolvable and maintainable software.
9. Recommendations	None
10. Author Response	None

Table B15. Details for Review categorised as DES: #154

Characteristic	Values
1. Knowledge Unit	DES.dd (Detailed design)
2. Title	What do we know about the Effectiveness of Software Design Patterns?
3. Citation	[57]

Table B15 continued

Characteristic	Values
4. DARE Score	2.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	18 (11 industrial and 7 academic studies)
7. Synthesis used	Narrative Synthesis
8. Findings	Patterns do not appear to help novices learn about design.
9. Recommendations	None
10. Author Response	The authors reviewed our analysis and agreed with it.

Table B16. Details for Review categorised as MAA: #123

Characteristic	Values
1. Knowledge Unit	MAA.tm (Types of models)
2. Title	A Systematic review of domain analysis tools
3. Citation	[58]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	19 (7 are implicitly industry studies, 12 are academic studies)
7. Synthesis used	Could not be determined.
8. Findings	<ol style="list-style-type: none"> 1. No tools support all functionalities of a specific process. 2. The majority of the analysed tools have similar functionalities. 3. The majority of tools are still being developed and used in an academic environment. 4. The documentation function is being explored more in the more recent tools investigated. 5. The domain analysis process without tool support can lead to an unsuccessful result but the use of any tool will not necessarily lead to an effective result.
9. Recommendations	None
10. Author Response	The authors observe that a finding could be to categorize functionalities as essential, important and low.

Table B17. Details for Review categorised as MAA: #126

Characteristic	Values
1. Knowledge Unit	MAA.tm (Types of models)
2. Title	Does the technology acceptance model predict actual use?
3. Citation	[59]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	79 (type not reported)
7. Synthesis used	Vote counting
8. Findings	Perceived usefulness (PU) and perceived ease of use (PEU) are less likely than behavioural intention (BI) to be correlated with actual use.
9. Recommendations	None
10. Author Response	None

Table B18. Details for Review categorised as MAA: #146

Characteristic	Values
1. Knowledge Unit	MAA.tm (Types of models)
2. Title	A practice-driven systematic review of dependency analysis solutions
3. Citation	[60]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	65 (38 explicit industry, 27 not specified)
7. Synthesis used	Narrative Synthesis
8. Findings	<p>1. Source-code based solutions identify dependencies through code constructs such as function calls and shared variables. Approaches that use this concrete evidence have a high degree of accuracy when it comes to the dependencies they identify, which makes them very reliable and very attractive for practitioners as the resulting information is very tangible. However, they are less suited to analyzing runtime system behaviour.</p> <p>2. Solutions using diagrammatic and semi-formal descriptions are more appealing for practitioners following architecture-driven approaches. Practitioners find these solutions useful to describe dependency information at an architecture level. However, for an efficient application of these solutions, it is necessary to keep up-to-date and synchronize the system requirements, design, and implementation.</p> <p>3. Solutions using run-time and configuration information are applicable in practice due to two main characteristics. First, these solutions are non-intrusive with respect to the development activities. Often, in a research setting, the overhead and maintenance cost of an infrastructure to collect data for dependency analysis is overlooked, whereas practitioners are more concerned about the cost and overhead of maintaining a reliable and up-to-date instrumentation of their system. This is even more important, in heterogeneous situations where multi-vendor components are used and instrumentation cannot be inserted into the system because of security, licensing, lack of knowledge, or other technical constraints. Second, although these solutions are limited by their coverage and links to the system source code, practitioners consider these solutions valid approximations, especially for problem-driven approaches.</p>
9. Recommendations	None
10. Author Response	None

Table B19. Details for Review categorised as MAA: #155

Characteristic	Values
1. Knowledge Unit	MAA.tm (Types of models)
2. Title	A Systematic Literature Review on Fault Prediction Performance in Software Engineering
3. Citation	[61]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	36 (35 explicit industry, 1 academic study)
7. Synthesis used	Thematic Analysis
8. Findings	<p>1. Models that work well tend to be built in a context where the systems are large.</p> <p>2. In terms of context, there is no evidence to suggest that the maturity of systems or language used is related to predictive performance</p>

Table B19 continued

Characteristic	Values
	3. It may be more difficult to build reliable prediction models for some application domains (e.g. embedded systems).
	4. The independent variables used by predictive models that work well seem to be sets of metrics.
	5. Models that use KLOC perform no worse than where only single sets of other static code metrics are used.
	6. The spam filtering technique based on source code performs relatively well.
	7. Models that perform well tend to use simple, easy to use modeling techniques such as Naïve Bayes or Logical Regression. More complex modeling techniques such as SVM tend to be used by models which perform relatively less well.
	8. Successful models tend to be trained on large datasets which have a relatively high proportion of faulty units.
	9. Successful models tend to use a large range of metrics on which feature selection was implied.
	10. For successful models, default parameters for the modelling technique were adjusted to ensure the technique would perform effectively.
9. Recommendations	None
10. Author Response	Agreed with our extracted findings.

Table B20. Details for Reviews categorised as REQ: #134

Characteristic	Values
1. Knowledge Unit	REQ.er (Eliciting requirements)
2. Title	Systematic Review and Aggregation of Empirical Studies on Elicitation Techniques
3. Citation	[22]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	32 (7 explicit industry; 7 academic; 18 unclear)
7. Synthesis used	Vote counting
8. Findings	1. Unstructured interviews (although it is reasonable to assume that the same applies to structured interviews) are equally as or more effective than introspective technique (such as protocol analysis) and sorting techniques. 2. Unstructured interviews (although it is reasonable to assume that the same applies to structured interviews) output more complete information than introspective technique (such as protocol analysis), sorting techniques and Laddering. 3. Unstructured interviews (although it is reasonable to assume that the same applies to structured interviews) are less efficient than sorting techniques and Laddering but as efficient as introspective techniques (such as protocol analysis). 4. The introspective techniques (such as protocol analysis) are the worst of all the tested techniques in all the dimensions (effectiveness, efficiency, completeness) and are outperformed by unstructured interviews (although it is reasonable to assume that the same applies to structured interviews), and sorting techniques and Laddering. 5. Laddering is preferable to sorting techniques (as well as introspection techniques).
9. Recommendations	None
10. Author Response	None

Table B21. Details for Reviews categorised as REQ: #161

Characteristic	Values
1. Knowledge Unit	REQ.er (Eliciting requirements)
2. Title	A systematic literature review of stakeholder identification methods in requirements elicitation
3. Citation	[62]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Lists papers addressing specific practices and issues.)
6. #Primary Studies	42 (all implicitly industry)
7. Synthesis used	Thematic Analysis
8. Findings	None
9. Recommendations	1. Assign appropriate roles to stakeholders through analysis of skills, behaviors in group dynamics and personality tests. 2. Establish constructive interaction between all stakeholders and between stakeholders and the system. 3. Classify requirements elicited from stakeholders according to an evaluation of their priorities in the project
10. Author Response	The authors agreed with our interpretation.

Table B22. Details for Reviews categorised as REQ: #259

Characteristic	Values
1. Knowledge Unit	REQ.rsd (Requirements specification and documentation)
2. Title	A systematic literature review of use case specifications research
3. Citation	[63]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made. (Lists papers addressing specific issues.)
6. #Primary Studies	119 (27 explicit industry; 11 academic; 81 unclear)
7. Synthesis used	Could not be identified.
8. Findings	1. Use case specifications were typically employed in two perspectives: documenting the functional requirements (typically using informal tabular or paragraph style formats); and for generating the lower-level software artifacts by using greater formalism to support a model-transformation process. 2. Use cases have evolved from paragraph format textual descriptions to a more formal keyword-oriented format for facilitating automated information retrieval. 3. Use cases have been applied and used in almost all the software development life cycle activities. However, knowledge about their applicability in planning and estimation and maintenance phases is limited, due to the limited number of published studies.
9. Recommendations	None
10. Author Response	The authors suggested some rewording of the third conclusion (incorporated).

Table B23. Details for Review categorised as QUA: #219

Characteristic	Values
1. Knowledge Unit	QUA.pda (Product assurance)
2. Title	Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review

Table B23 continued

Characteristic	Values
3. Citation	[64]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	99 (33 are implicitly from industry; 5 academic; and 61 could not be classified)
7. Synthesis used	Vote counting
8. Findings	1. Measures for complexity, cohesion, coupling and size show better consistency in their relationship with reliability and maintainability attributes across the primary studies than inheritance. 2. Measures that quantify inheritance properties show poor links to reliability and maintainability.
9. Recommendations	None
10. Author Response	The authors observed that for the second conclusion, the poor showing of the inheritance measures might have stemmed from confounding factors in the primary studies.

Table B24. Details for Review categorised as PRO: #39

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Quality, productivity and economic benefits of software reuse: a review of industrial studies
3. Citation	[45]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	11 (all explicit industry)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. Defect, error or fault density is significantly reduced. 2. Rework effort is significantly reduced. 3. Apparent productivity improves significantly.
9. Recommendations	None
10. Author Response	None

Table B25. Details for Review categorised as PRO: #50

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Software process improvement in small and medium software enterprises: a systematic review
3. Citation	[46]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	45 (all explicit industry)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	It is difficult to successfully apply formal SPI programmes which use models such as CMM to SMEs.
9. Recommendations	None
10. Author Response	None

Table B26. Details for Review categorised as PRO: #138

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Measuring and predicting software productivity: a systematic map and review
3. Citation	[65]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	38 (25 explicit industry, 13 academic)
7. Synthesis used	Narrative Synthesis
8. Findings	<ol style="list-style-type: none"> 1. The variety of model forms means that strong recommendations cannot be provided. However, the studies did not come to conclusions that contradicted each other. 2. Simulation overall provided promising results. 3. Time-series analysis/statistical process control also shows good results in identifying sharp shifts in process performance as well as shifts due to changes in the process. 4. To be able to give a recommendation on the predictive accuracy of regression for software productivity, the model should be built on a sub-set of data points and then used to predict the remaining data points. Thereafter the difference between prediction and actual values should be observed and measured.
9. Recommendations	<ol style="list-style-type: none"> 1. When using univariate models it is important to be aware of high variances and difficulties when comparing productivities. Hence it is important to carefully document the context to be able to compare between products. Comparison should not be on productivity value alone and it is recommended that a scatter diagram be produced based on inputs and outputs to assure comparability of projects with respect to size. 2. When comparing projects it should be made clear what output and input consists of, for example, which lines are included in LOC measures. 3. When possible, use multivariate analysis when data is available, as throughout the software process many outputs are produced. Otherwise, productivity is biased towards one measure (eg LOC). 4. Managers need to be aware of validity threats present in the measures when conducting a comparison. Data should be interpreted with care and awareness of possible bias and noise in the data arising from measurement error. 5. No generic prediction model can be recommended as studies do not clearly agree on what are the predictors for software productivity. In fact, the predictors might differ between contexts. Hence companies need to identify and test predictors relevant to their context.
10. Author Response	<p>The authors identify the following additional finding.</p> <ol style="list-style-type: none"> 5. Data envelopment analysis is promising as it supports multivariate productivity measures, and allows identification of reference projects to which inefficient projects should be compared. This helps with identifying projects from which one can learn, and that are similar, so that evidence may be transferable.

Table B27. Details for Review categorised as PRO: #157

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis

Table B27 continued

Characteristic	Values
3. Citation	[67]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	37 (10 explicit industry; 23 academic; 4 unclear)
7. Synthesis used	Meta-analysis
8. Findings	Use of TDD can result in a small improvement in quality (implicit).
9. Recommendations	None
10. Author Response	None

Table B28. Details for Review categorised as PRO: #160

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Reconciling software development models: A quasi-systematic review
3. Citation	[68]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	42 (all implicit industry studies)
7. Synthesis used	Thematic analysis
8. Findings	1. Three different levels of reconciliation are found: organisational, group and process. 2. Main opportunities for reconciliation are derived from collaboration and code availability. 3. There is a diversity of challenges – most salient is overcoming barriers to culture change.
9. Recommendations	None
10. Author Response	Our findings were confirmed by the authors.

Table B29. Details for Review categorised as PRO: #174

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	A systematic literature review on the industrial use of software process simulation
3. Citation	[69]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	87 (all implicit industry studies)
7. Synthesis used	Could not be determined
8. Findings	No evidence of widespread adoption and impact of SPSM research on industry.
9. Recommendations	When using software process simulation models for scientific purposes, need to be sure that the appropriate steps with respect to model validity checking have been conducted, and do not rely upon a single simulation run.
10. Author Response	We have used a slight rewording of the recommendation suggested by the authors.

Table B30. Details for Review categorised as PRO: #228

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	An investigation into the best practices for the successful design and implementation of lightweight software process assessment methods: A systematic literature review
3. Citation	[74]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made. (Lists papers identifying successful practices.)
6. #Primary Studies	22 (all explicit industry studies)
7. Synthesis used	Thematic analysis
8. Findings	A set of 38 best practices has been collected and classified into five main areas: method, supportive tool, procedure, documentation and user best practices.
9. Recommendations	The paper has identified a set of best practices to support and inform designers and assessors for software process assessment.
10. Author Response	We have used some rewording suggested by the authors.

Table B31. Details for Review categorised as PRO: #249

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Exploring principles of user-centred agile software development: A literature review
3. Citation	[79]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Provides counts of papers identifying issues related to principles.)
6. #Primary Studies	83 (26 implicit industry studies, 57 that could not be classified)
7. Synthesis used	Content analysis
8. Findings	None
9. Recommendations	<ol style="list-style-type: none"> 1. User-centered agile software development should be based on separated product discovery and product creation phases. 2. In user-centered agile approaches, design and development should proceed in parallel interwoven tracks. 3. In user-centered agile approaches, tangible and up-to-date artifacts should be used to document and communicate product and design concepts, and should be accessible to all involved stakeholders.
10. Author Response	None

Table B32. Details for Review categorised as PRO: #268

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Requirements for product derivation support: Results from a systematic literature review and an expert survey
3. Citation	[81]
4. DARE Score	2.0
5. Strength of Evidence	No assessment was made.

Table B32 continued

Characteristic	Values
6. #Primary Studies	118 (unclassified as no details provided)
7. Synthesis used	Narrative synthesis
8. Findings	Systematic Review followed by expert survey identified the following six requirements for product derivation support: <ol style="list-style-type: none"> 1. automated and interactive variability resolution 2. adaptability and extensibility 3. application requirements management support 4. flexible and user-specific visualisations of variability 5. end-user guidance 6. project management support
9. Recommendations	None
10. Author Response	The authors agreed with our extracted findings.

Table B33. Details for Review categorised as PRO: #276

Characteristic	Values
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Empirical evaluation of a decision support model for adopting software product line engineering
3. Citation	[82]
4. DARE Score	3.0
5. Strength of Evidence	No assessment was made. (Lists papers identifying relevant factors.)
6. #Primary Studies	31 (all implicit industry)
7. Synthesis used	Thematic Analysis + Vote Counting
8. Findings	1. The study identifies 25 factors that should be considered when investigating adoption of SPLE (e.g. business motivation, market potential, software architecture competence). In all, 39 questions that might be asked and 312 rules that could be applied are developed. Rules include recommendations and strategies (but only one example provided)
9. Recommendations	None
10. Author Response	The authors agreed with our extracted findings.

Table B34. Details for Review categorised as PRO: #84

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	The effectiveness of pair programming: A meta-analysis
3. Citation	[47]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	19 (5 explicit industry and 14 academic studies)
7. Synthesis used	Meta-analysis
8. Findings	None

Table B34 continued

Characteristic	Values
9. Recommendations	If you do not know the seniority or skill levels of your programmers, but do have a feel for task complexity, then employ PP either when task complexity is low and time is of the essence, or when task complexity is high and correctness is important.
10. Author Response	The authors agreed with our extracted recommendations.

Table B35. Details for Review categorised as PRO: #150

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Agile product line engineering – a systematic literature review
3. Citation	[66]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	39 (14 explicit industry and 25 not specified)
7. Synthesis used	Narrative synthesis
8. Findings	<ol style="list-style-type: none"> 1. If software product line (SPL) developers do not have enough knowledge to completely perform the domain engineering (DE), agile software development (ASD) may facilitate the elicitation of further knowledge. 2. Trade-offs between SPLE and ASD provide the opportunity to apply the agile product line engineering (APLE) approach to a wider variety of projects than those served by only applying ASD or SPL methods. 3. When anticipated changes cannot be predicted and the product life cycle is not known, it would be advantageous to use an incremental approach such as APLE. 4. Agile processes may facilitate fast feedback cycles between requirements engineering (RE), development and field trial in innovative business.
9. Recommendations	None
10. Author Response	The authors agreed with our extracted findings and observed that Table VII in their paper does implicitly provide some recommendations for practice.

Table B36. Details for Review categorised as PRO: #193

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Empirical studies on the use of social software in global software development – A systematic mapping study
3. Citation	[71]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	84 (61 explicit industry and 23 academic studies)
7. Synthesis used	Narrative synthesis
8. Findings	<ol style="list-style-type: none"> 1. Social Networking sites help identify experts and provide awareness of people's expertise. 2. It is necessary to develop structures, rules, good practices and agreements for using SoSo in a work context and on a project basis.
9. Recommendations	None
10. Author Response	None

Table B37. Details for Review categorised as PRO: #215

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Software development in startup companies: A systematic mapping study
3. Citation	[72]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Lists papers identifying factors.)
6. #Primary Studies	43 (30 implicit industry and 13 that could not be classified)
7. Synthesis used	Thematic analysis
8. Findings	<p>1. Light-weight methodologies to obtain flexibility in choosing tailored practices, and reactivity to change the product according to business strategies is a useful process management practice in startups.</p> <p>2. Fast releases to build a prototype in an evolutionary fashion and quickly learn from the users' feedback to address the uncertainty of the market is a useful process management practice in startups.</p> <p>3. The use of well-known frameworks able to provide fast changeability of the product in its refactoring activities is a useful design and architectural practice in startups.</p> <p>4. The use of existing components, leveraging third party code reinforcing ability to scale the product is a useful design and architectural practice in startups.</p> <p>5. The use of ongoing customer acceptance with the use of focus groups of early adopters, which aims to determine the fitness of the product for the market is a useful quality assurance practice in startups.</p> <p>(plus six further conclusions)</p>
9. Recommendations	None
10. Author Response	The authors agreed with our interpretation.

Table B38. Details for Review categorised as PRO: #217

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Understanding the Influence of User Participation and Involvement on System Success – A Systematic Mapping Study
3. Citation	[20]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	82 (all implicit industry)
7. Synthesis used	Meta-analysis
8. Findings	<p>1. "Given the vast amount of positive correlations, we can conclude that, even though the results are not completely consistent, the amount of studies with positive correlations of the various aspects of UPI on system success provides evidence of a robust and transferable effect."</p> <p>2. Most studies with negative correlations from aspects of UPI on system success were published more than 10 years ago.</p> <p>3. UPI has a positive effect on user satisfaction and system use.</p>
9. Recommendations	None
10. Author Response	The authors suggested some revisions which were partly adopted.

Table B39. Details for Review categorised as PRO: #222

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	The Kanban approach, between agility and leanness: a systematic review
3. Citation	[73]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made. (Lists papers identifying relevant benefits.)
6. #Primary Studies	37 (all implicit industry)
7. Synthesis used	Case survey
8. Findings	1. There is a lack of details and guidelines on how the Kanban approach can be used by IT organisations. 2. The Kanban board is an efficient visualisation tool.
9. Recommendations	Discuss Kanban elements together, based on the five pillars of the lean approach, to minimize the risk of evolving contradictory elements and to facilitate establishing guidelines and instructions on how to set up the Kanban approach, to give practitioners an overall framework that increases the likelihood of successfully implementing the Kanban approach in IT organisations.
10. Author Response	None

Table B40. Details for Review categorised as PRO: #236

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	The impact of global dispersion on coordination, team performance and software quality – A systematic literature review
3. Citation	[75]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Provides counts of studies addressing factors.)
6. #Primary Studies	43 (40 explicit industry and 3 academic)
7. Synthesis used	Thematic analysis + Vote counting
8. Findings	1. The impact of each dispersion dimension on project outcomes is mediated by a different set of coordination issues in GSD. 2. A distributed task takes a longer time to communicate and resolve than a co-located task does in GSD. 3. Temporal dispersion has a positive impact on objective team performance while it has a negative impact on perceived team performance in GSD. 4. Geographical dispersion has a negative impact on software quality, at both file and project level in GSD. 5. Temporal dispersion has a negative impact on software quality, at both file and project level in GSD.
9. Recommendations	1. Managers should be aware of the influence of dispersion dimension at different organisational levels. At the individual level, lack of face-to-face interaction and working in different time zones affects directly and negatively a developer's work. At the team and project level, the negative influences of these dispersion dimensions might be underestimated in considering different goals and priorities. This issue must be taken into account when aligning the objective of individuals and teams with organizational goals.

Table B40 continued

Characteristic	Values
	2. Decisions on which coordination mechanisms to use should depend on the current dispersion context setting, the current team coordination technology and practices, and prioritized type of interdependencies. Our summary shows that communication and shared artifacts should be used together as needed and a defined process should be adopted at the team and organizational levels.
10. Author Response	The authors agreed with our extracted findings.

Table B41. Details for Review categorised as PRO: #239

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Using CMMI together with agile software development: A systematic review
3. Citation	[76]
4. DARE Score	4.5
5. Strength of Evidence	Use of GRADE. Strength of evidence considered to be <i>low</i> for all findings.
6. #Primary Studies	60 (59 explicit industry and 1 academic)
7. Synthesis used	Thematic analysis
8. Findings	<ol style="list-style-type: none"> 1. Agile methodologies have been used by companies to enhance their efforts to reach levels 2 and 3 of CMMI, with reports of applying agile practices to achieve level 5. 2. Agile methodologies alone are not sufficient to achieve the level required, it being necessary to resort to additional practices. 3. Organisations should seek to ensure that how CMMI and agile can be combined is understood and undertaken by those involved.
9. Recommendations	None
10. Author Response	None

Table B42. Details for Review categorised as PRO: #241

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	A systematic review on the relationship between user involvement and system success
3. Citation	[77]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Lists papers identifying specific benefits.)
6. #Primary Studies	87 (all implicit industry)
7. Synthesis used	Thematic analysis + Vote counting
8. Findings	<ol style="list-style-type: none"> 1. Identification of the right type of users who will be involved, and who will participate, are important factors according to the literature, but the review did not find enough empirical evidence about this to confirm it. 2. The perspective of user involvement is one of the most important factors. Analysis identified five major perspectives for user involvement: psychological, managerial, methodological, political, and cultural. 3. User involvement takes different forms for development of different types of system. 4. User satisfaction leads to system success (the top cited factor).

Table B42 continued

Characteristic	Values
	5. To achieve benefits in methodological and psychological perspectives, user involvement in the requirements phases seems to be most effective.
	6. To achieve benefits for political and cultural perspectives, users need to be involved in the design and implementation phases.
9. Recommendations	None
10. Author Response	The authors provided some comments which we have used to modify the findings.

Table B43. Details for Review categorised as PRO: #260

Characteristic	Values
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Claims about the use of software engineering practices in science: A systematic literature review
3. Citation	[80]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made. (Lists papers addressing specific issues.)
6. #Primary Studies	43 (all academic)
7. Synthesis used	Thematic analysis
8. Findings	1. Scientific software developers benefit from using a wide range of testing practices from software engineering. 2. Open-source is especially useful to scientific software developers. 3. Documentation is a necessary enabler of software quality. 4. Version control software is necessary for research groups with more than one developer. (Note: These were the conclusions with strongest supporting evidence.)
9. Recommendations	None
10. Author Response	The authors provided some comments which we have used to modify the findings.

Table B44. Details for Review categorised as PRO: #8

Characteristic	Values
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Forecasting of software development work effort: Evidence on expert judgement and formal models
3. Citation	[43]
4. DARE Score	1.0
5. Strength of Evidence	Informally estimated as “modest”.
6. #Primary Studies	16 (14 explicit industry and 2 academic studies)
7. Synthesis used	(Synthesis was not assessed for Dataset1)
8. Findings	1. The review does not support the view that we should replace expert judgement with models. 2. The review does not support the view that software estimation models are useless.. 3. Models failed to systematically perform better than the experts when estimating.

Table B44 continued

Characteristic	Values
	4. Two conditions for producing more accurate expert judgement-based effort seem to be that the models are not calibrated to the organization using them, and that the experts possess important contextual information not included in the formal models and apply it efficiently.
	5. The use of models, either alone or in combination with expert judgement, may be particularly useful when i) there are situational biases that are believed to lead to a strong bias towards overoptimism; ii) the amount of contextual information possessed by experts is low; and iii) the models are calibrated to the organization using them.
9. Recommendations	It is best to use a combination of models and experts when estimating the level of effort required to complete software development tasks.
10. Author Response	None

Table B45. Details for Review categorised as PRO: #22

Characteristic	Values
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Evidence-Based Guidelines for Assessment of Software Development Cost Uncertainty
3. Citation	[44]
4. DARE Score	2.5
5. Strength of Evidence	An assessment was made for each guideline, identifying supporting papers.
6. #Primary Studies	40 (none could be classified)
7. Synthesis used	(Synthesis was not assessed for Dataset1)
8. Findings	None
9. Recommendations	1. Do not rely solely on unaided, Intuition-based processes. (<i>Strong</i> evidence.) 2. Do not replace expert judgement with formal models. (<i>Medium</i> evidence.) 3. Apply structured and explicit judgement-based processes. (<i>Strong</i> evidence.) 4. Apply strategies based on an outside view of the project. (<i>Medium</i> evidence.) 5. Use motivational mechanisms with care and only if it is likely that more effort leads to improved assessments. (<i>Medium</i> evidence.) 6. Frame the assessment problem to fit the structure of the uncertainty relevant information and the assessment process. (<i>Medium</i> evidence.)
10. Author Response	None

Table B46. Details for Review categorised as PRO: #102

Characteristic	Values
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Managing risks in distributed software projects: An integrative framework
3. Citation	[48]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	72 (implicit industry studies)
7. Synthesis used	(Synthesis was not assessed for Dataset1)

Table B46 continued

Characteristic	Values
8. Findings	<ol style="list-style-type: none"> 1. Built framework demonstrating complex nature of risks in GDSP and offers concepts and heuristics that practitioners can use to assess and control the risks they face in specific projects. Can be used by project managers. 2. Provides a useful vocabulary.
9. Recommendations	<ol style="list-style-type: none"> 1. Revisit risk management regularly during project lifetime. 2. Practitioners are advised to go through the steps of risk assessment, risk control and risk management planning.
10. Author Response	None

Table B47. Details for Review categorised as PRO: #121

Characteristic	Values
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Empirical evidence in global software engineering: A systematic review
3. Citation	[21]
4. DARE Score	3.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	56 (37 explicit industry studies, 16 academic, 3 not stated)
7. Synthesis used	Narrative synthesis
8. Findings	<ol style="list-style-type: none"> 1. Trust, cohesiveness and effective teamwork can be achieved through F2F meetings, temporal colocation and exchange visits - but entail extra costs. 2. Greater awareness and process transparency can be achieved through the use of a centralised repository and common configuration management tool support - but requires overcoming heterogeneity. 3. Trust and cohesiveness can be improved through effective and frequent synchronous communications – but entail extra costs. 4. Effective communications can be achieved if infrastructure is reliable and communications media are rich. 5. Effective teamwork can be achieved through synchronous interaction – but requires temporal proximity . 6. Effective teamwork can be achieved through task distribution based on architectural decoupling and low dependencies across remote locations – but requires full transition of parts of the work. 7. Early feedback and capability evaluation can be achieved through the use of incremental short-cycle development – but requires frequent and transparent communications. <p>There is still no recipe for successful and efficient performance in globally distributed software engineering.</p>
9. Recommendations	None
10. Author Response	The authors observe that since key practices that help minimise risk require additional investments, global collaboration might not be suitable for companies that enter global projects to reduce costs.

Table B48. Details for Review categorised as PRO: #175

Characteristic	Values
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Barriers in the selection of offshore software development outsourcing vendors: An exploratory study using a systematic literature review
3. Citation	[70]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made. (Provides counts of papers identifying specific barriers.)
6. #Primary Studies	77 (All explicit industry studies)
7. Synthesis used	Thematic analysis
8. Findings	<ol style="list-style-type: none"> 1. Barriers vary with organisation size. These are summarised in Table 7 of their paper. The one common barrier is “language and cultural barriers”. 2. Viewed over two decades, different barriers have “risen” and “fallen” in importance.
9. Recommendations	<ol style="list-style-type: none"> 1. Outsourcing vendors should focus on the identified barriers in order to have a positive impact on outsourcing clients and to win outsourcing contracts: language and cultural barriers. 2. Vendors should focus on the barriers identified in order to have a positive impact on outsourcing clients and to win outsourcing contracts: country instability. 3. Vendors should focus on the barrier identified in order to have a positive impact on outsourcing clients and to win outsourcing contracts: lack of project management. 4. Vendors should focus on the barriers identified in order to have a positive impact on outsourcing clients and to win outsourcing contracts: lack of protection for IPR. 5. Vendors should focus on the barriers identified in order to have a positive impact on outsourcing clients and to win outsourcing contracts: lack of technical capability.
10. Author Response	The authors agreed with our extracted data.

Table B49. Details for Review categorised as PRO: #244

Characteristic	Values
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Analogy-based software development effort estimation: A systematic mapping and review
3. Citation	[78]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made. (Lists studies identifying specific factors.)
6. #Primary Studies	61 (all were explicitly industrial)
7. Synthesis used	Narrative synthesis
8. Findings	<ol style="list-style-type: none"> 1. ASEE methods tend to yield acceptable estimates. 2. ASEE methods outperform regression based methods. 3. ASEE methods outperform ANN based methods. 4. ASEE methods outperform DT based methods. 5. One ASEE technique alone may not be the best estimation method in all contexts. However, in any context, an appropriate effort estimation model can be built by combining an ASEE technique with other techniques to overcome the weaknesses.”

Table B49 continued

Characteristic	Values
	<p>6. The results suggest overall that the estimation accuracy of ASEE methods is improved when used in combination with other techniques, especially FL and GA. As has been found, SM improves the accuracy of ASEE techniques much less than the other techniques. This suggests that using ML rather than non ML techniques in combination with analogy would be preferable, in particular, fuzzy logic, genetic algorithms, the model tree, and the collaborative filtering. The limited number of studies on ASEE methods combined with these techniques may account for these inconclusive results.</p> <p>7. Taking into consideration the number of evaluations and based on the median of the MMRE, MT is the technique that improves the accuracy of ASEE methods the most (59.42% improvement), followed by CF combined with RSA (51.85%) and LSR (41.03%). Based on the median of the MdmRE, MT has the greatest impact (67.75%), followed by FL combined with GRA (40.80%) and GA (37.93%). Based on the arithmetic median of Pred(25), ASEE techniques are improved the most by MT (129.01% improvement), followed by CF (108.33%) and GA (100.00%).</p> <p>8. Results suggest overall that all the techniques listed in Section 3.5 improve the estimation accuracy of ASEE methods, especially GA and FL, which are supported by 4 studies each. There is much less improvement in the accuracy of ASEE techniques when combined with SM.</p>
9. Recommendations	None
10. Author Response	The authors agreed with our extracted findings.

Technical Debt Aware Estimations in Software Engineering: A Systematic Mapping Study

Paweł Klimczyk*, Lech Madeyski**

*GEMOTIAL

**Faculty of Computer Science and Management, Wrocław University of Science and Technology,
Wyb. Wyspińskiego 27, 50-370 Wrocław, Poland

pawel@klimczyk.pl, lech.madeyski@pwr.edu.pl

Abstract

Context: The Technical Debt metaphor has grown in popularity. More software is being created and has to be maintained. Agile methodologies, in particular Scrum, are widely used by development teams around the world. Estimation is an often practised step in sprint planning. The subject matter of this paper is the impact technical debt has on estimations.

Objective: The goal of this research is to identify estimation problems and their solutions due to previously introduced technical debt in software projects.

Method: The Systematic mapping study (SMS) method was applied in the research. Papers were selected from the popular digital databases (IEEE, ACM, Scopus, etc.) using defined search criteria. Afterwards, a snowballing procedure was performed and the final publication set was filtered using inclusion/exclusion criteria.

Results: 42 studies were selected and evaluated. Five categories of problems and seven proposed solutions to the problems have been extracted from the papers. Problems include items related to business perspective (delivery pressure or lack of technical debt understanding by business decision-makers) and technical perspective (difficulties in forecasting architectural technical debt impact or limits of source code analysis). Solutions were categorized in: more sophisticated decision-making tools for business managers, better tools for estimation support and technical debt management tools on an architectural-level, portfolio approach to technical debt, code audit and technical debt reduction routine conducted every sprint.

Conclusion: The results of this mapping study can help taking the appropriate approach in technical debt mitigation in organizations. However, the outcome of the conducted research shows that the problem of measuring technical debt impact on estimations has not yet been solved. We propose several directions for further investigation. In particular, we would focus on more sophisticated decision-making tools.

Keywords: Software estimation, technical debt, project management, decision making, change impact

1. Introduction

Today software is present in all industries worldwide. The Industry 4.0 [1, 2]¹ or Internet of Things [3] concepts are based on software to operate and provide solutions. Agile methods were proposed to *better handle inevitable changes* [4].

A number of practices have become popular, e.g., Continuous Integration, TDD, Pair Programming, to ensure sufficient production code and tests quality (e.g., [5–7]) and software development productivity (e.g., [8]).

Cunningham [10] introduced the *technical debt* term to describe shortcuts taken by soft-

¹Note that two reference lists are included at the end of this paper: the first one includes papers found during our systematic mapping, the second one is the main reference list.

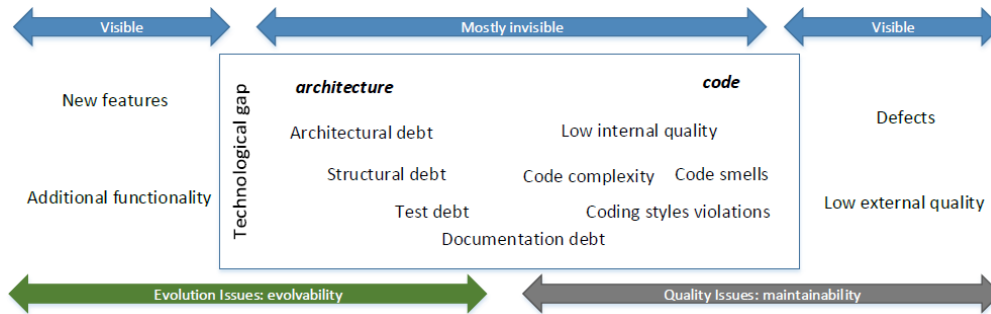


Figure 1. Technical Debt Landscape (inspired by [9])

ware engineers in order to deliver value on time. “A little debt speeds development so long as it is paid back promptly with a rewrite... The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.” [10]. The number of software developers increases every year. That implies creating more code and more technical debt in the result. According to Google Trends² technical debt metaphor has been growing in popularity.

Software project features may be delivered faster to users, but the effects of taking technical debt (e.g., storing application data in a file instead of a database) will have to be addressed in the future. As stated by Fowler [11], technical debt can be taken intentionally or unintentionally. Along with technical debt there is a *interests* concept. *Interests* can be considered as “the extra maintenance cost spent for not achieving the ideal quality level.” [12]. It is a metaphor for unpaid technical debt becoming more expensive to repay over time. Technical debt grows during the software development process as stated in [13].

Technical Debt Landscape (Figure 1) was introduced by Kruchten et al. [9]. The landscape identifies *mostly invisible* area where potential problems affecting estimations exist. *Mostly invisible* items are hidden to everybody apart from software engineers. Other members of the project team are aware of them, but might not know the details. The authors state: *Technical debt should not be treated in isolation from adding new functionality or fixing defects and The challenge is in expressing all software develop-*

ment activities in terms of sequences of changes associated with a cost and a value [9]. Software development teams should communicate the “technological gap” in effort estimation so *mostly invisible* parts are known to the managers and stakeholders.

Estimation is a process of rough calculation of how much time is needed to deliver business value related to the estimated task or feature. There is a number of techniques helping developers to provide more accurate estimation [14–16] (e.g., poker planning, smart use cases or bucket system). Some of them use the developer’s experience in a project to consider technical debt impact on estimation accuracy.

Estimations are straightforward in well-specified projects. Development teams start from scratch and will introduce technical debt. As new features are implemented or as existing features are extended, the project’s complexity increases. The problem with estimations becomes visible after the technical debt has been taken and has to be addressed. It may be expected that forecasting technical debt impact on a new or changed feature is more difficult in later development stages. Estimations are becoming inaccurate and one of the reasons is improper technical debt measurement. The problem has to be addressed.

The goal of this research was to conduct a systematic mapping study on technical debt in the context of estimations. A number of publications were collected, examined and categorized giving several directions for further research.

The paper is organized as follows: Section 2 presents related work. Section 3 defines research

²<https://trends.google.com/trends/explore?date=all&q=technicaldebt>

questions for this systematic mapping study (SMS). Research methodology and crucial details of the SMS protocol are described in Section 4. Section 5 shows study results with a detailed description. In Section 6 we interpret responses to the posed study research questions. Section 7 presents threats to validity, while in Section 8 we conclude the work and show directions of further research. A list of primary sources found in our SMS is presented before references.

2. Related work

The amount of produced software worldwide increases every year which in turn affects technical debt. A number of studies have been conducted to address the problem of increasing technical debt from various perspectives.

Fernández-Sánchez et al. [17] searched for elements required in the technical debt management. They came up with a list of 12 items that will support decision making in managing technical debt. Items are divided into three types: *(T1) Basic decision-making factors*, *(T2) Cost estimation techniques* and *(T3) Practices and techniques for decision-making*. The result of this article is a framework introduced to aid decision making in technical debt management.

Another research by Fernández-Sánchez et al. [18] covers available techniques and methods for technical debt management from a software architecture perspective. In their systematic mapping study authors discovered the impact of various technical debt types, like code technical debt, documentation technical debt etc. on architectural technical debt. The conclusion is that further studies on architectural debt from a more holistic approach are needed.

Ribeiro et al. [19] provides a list of 14 decision criteria on which technical debt repayment can be prioritized. Authors conclude that none of the researched studies has performed an empirical evaluation. In the authors' opinion, this may indicate a low level of maturity in decision-making criteria itself.

Li et al. [20] in their mapping study on technical debt and its management identify a list of ten

technical debt types and 29 tools used as technical debt management systems. They indicate, however, that only four tools are dedicated to technical debt management. The rest is adapted in various ways from other software development areas. They conclude that there is a need for more sophisticated and dedicated technical debt management tools and further research on technical debt management. More high-level studies should be conducted by the software engineering community.

In another related work, Behutiye et al. [21] analyse the concept of technical debt in Agile Software Development (ASD). A list of ten causes and five consequences of incurring technical debt in ASD was identified in the research. Authors also classified a list of technical debt management strategies in ASD. The research indicates *the need for more tools, models and guidelines that support management of technical debt in ASD* [21] and the role of architecture in ASD.

The financial aspect is considered by Ampatzoglou et al. [22]. Authors introduced a glossary of financial terminology and classification schema of financial approaches used in technical debt management. The publication also states that it is easier for developers to communicate with non-technical managers.

Systematic mapping study on identification and management of technical debt was conducted in [23]. Research enumerates strategies that have been proposed to identify or manage technical debt in software projects. The conclusion is that most of the strategies are new but they lack studies to evaluate their real applicability.

None of the mentioned publications addressed the problem of technical debt impact on estimations. The goal of our work differs from the other secondary studies in terms of the research perspective and scope. Our study focuses on understanding how task delivery estimation is affected by technical debt and what software development teams do to develop software according to plan.

3. Research objectives

The objectives of this study were to identify problems in estimations due to existing technical debt

in software projects and collect ideas on how development teams try to overcome the problems. Following research questions were stated:

RQ1: What are the problems for the development team during task estimation due to technical debt?

The purpose of this question is to confirm problem existence. Potentially it could be possible to identify groups of similar problems.

RQ2: What kind of solutions are proposed to mitigate the impact of technical debt on task estimation?

The purpose of this question is to collect the actions taken by development teams to reduce technical debt factor in estimations.

4. Research methodology

In software engineering, guidelines developed by Kitchenham et al. [24] and Petersen et al. [25] provide comprehensive instructions on how to conduct systematic literature reviews (SLR) and systematic mapping studies. They share some commonalities (e.g., related to searching and study selection). However, the difference between both approaches is that systematic literature reviews focus on synthesising the evidence and gaining a new knowledge, while systematic mapping studies [25] are focused on structuring the research area and creating an overview. Systematic mapping study was chosen as a framework for this research to answer the questions posed in Section 3.

4.1. Systematic Mapping Study (SMS) protocol

Our protocol defines the procedures we intended to use for SMS including the following steps:

1. Define study objectives and research questions.
2. Define search query and digital source databases.
3. Define publication selection criteria.
4. Define inclusion and exclusion criteria.
5. Conduct data extraction and assessment.
6. Conduct data synthesis.

After trialling the specified processes, the final version of the protocol was agreed by both authors. The following sections are based on the processes defined in the protocol. However, it is worth mentioning that we have added an additional exclusion criteria (short summary reports) that was not mentioned in the protocol.

4.2. Search query

We performed a series of trial queries against electronic databases. In result the following search query was formulated:

("software") AND ("technical debt" OR "change impact") AND ("estimation" OR "decision making" OR "management")

Such a search query will find publications with a technical debt aspect in various contexts.

4.3. Digital source databases

Publication sources include all popular academic databases. The year 1992 was chosen as the time-frame limit since Cunningham published his paper at that time [10]. Studies from following digital source databases were included:

- IEEE Xplore [26],
- ACM Digital Library [27],
- Springer Link [28],
- Science Direct [29],
- Scopus [30].

4.4. Inclusion/exclusion criteria

Search query defined in Section 4.2 returned a total number of 2003 candidate documents for primary studies set. The distribution of documents per source database is presented in Table 1.

Primary studies set contained many irrelevant publications, due to query search generic nature. Thus, following inclusion/exclusion criteria were applied to select only relevant studies.

Inclusion criteria:

- Publications that describe the problem of technical debt in software development and technical debt management.
- Case studies and surveys based on industrial examples of technical debt management.

Table 1. Distribution of publications per source

Source	No. of publications returned by search query	No. of publications included in our paper
IEEE Xplore	275	14
ACM Digital Library	652	10
Springer Link	341	5
Science Direct	369	5
Scopus	366	7
Snowballing	n/a	1

- Technical debt management technique proposals.
- Papers written since 1992 when Cunningham [10] introduced the *technical debt* term³.
- Papers written in English – English is a common language used by researchers.

Exclusion criteria:

- Publications that only mention technical debt as an issue, but do not focus on deeper elaboration/description of the problem.
- Short summary reports about what workshop participants discussed instead of real research contributions – short summaries do not provide enough information.
- Duplicate publications.
- Publications with only abstract available – we were interested in the details of a particular research.
- Papers not written in English.

4.4.1. Snowballing

The importance of the snowballing step in SMS is described in [31]. Backward snowballing was performed for this study. Papers found in snowballing were checked using the same inclusion/exclusion criteria list as primary papers. The snowballing technique found one additional publication.

4.5. Data extraction and assessment

Data extraction and assessment process focused on collecting evidence that can formulate an an-

swer to RQs. All filtered publications were read in full. Microsoft Excel was used to record and organize the following data: title, source, citation eligible for RQ1 or RQ2 and publication type. The assessment was based on whether a study provides evidence to answer one of the RQs.

4.6. Initial research set

Initial research set consisted of 45 articles. After applying inclusion/exclusion criteria papers [S1], [S2] and [S3] was excluded. Decisions were discussed by both authors.

4.7. Rigor and relevance

We applied a checklist proposed by Ivarsson and Gorschek [32] to access rigor and relevance of the final dataset. The rating model consists of two perspectives to measure: rigor and relevance. Rigor refers to how an evaluation is performed and how is it reported. Relevance measures the industrial applicability in the usage context, used research method, subjects/users and scalability. Each item is scored by 0, 0.5 and 1 in rigor perspective and 0 or 1 in relevance perspective.

The first author rated the studies for quality assurance. The rigor and relevance scores distribution in our SMS is presented in Figure 2.

In order to review the selection agreement among the authors, a Kappa analysis [33] was performed. Seven randomly selected⁴ publications were examined by the second author. Based on the selected sample Kappa value was calcu-

³The technical debt knowledge, along with programming languages, has evolved over last 30 years, and we do not expect that problems and solutions discussed in papers written before 1992, and not cited after that year, would add value to the paper.

⁴<https://www.random.org/sequences/?min=1&max=42&col=1&format=html&rnd=new>

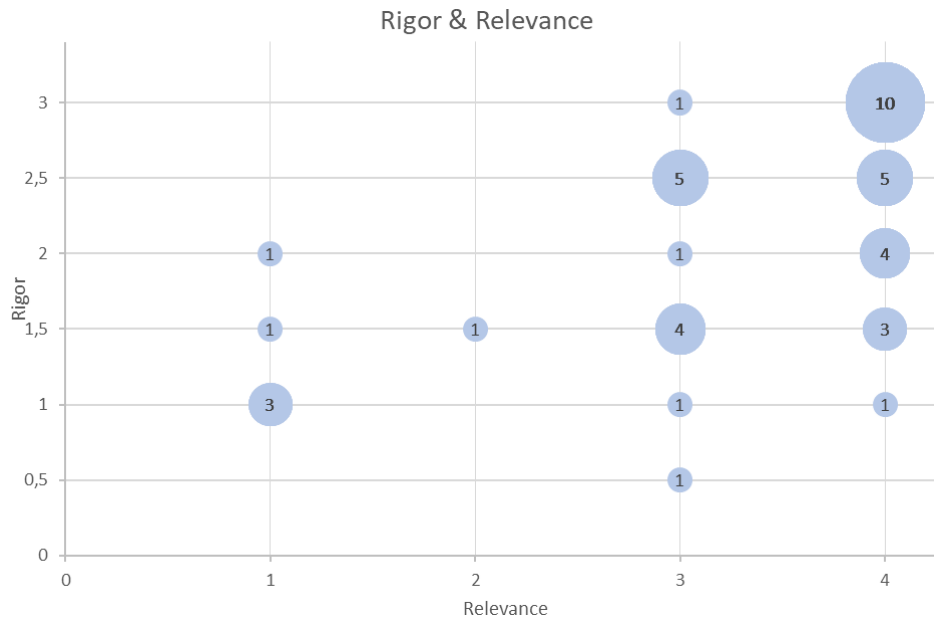


Figure 2. Mapping of selected papers with respect to rigor and relevance

lated – the strength of agreement was very good ($\kappa = 1.0$).

4.8. Final set of papers

We selected 42 publications, see Table 2 and the list of primary studies found in our systematic mapping, presented before references. 41 of the papers were filtered through digital source databases using search query presented in Section 4.2. An additional one was found during the snowballing process. Table 1 presents a distribution of publications per digital source databases and snowballing procedure. It is worth mentioning that case studies were the most popular publication types among the accepted primary studies.

At this point we assessed all evidence for eligibility and divided into two groups: **Identified problem categories (G1 – addressing RQ1)** and **Identified potential solution categories (G2 – addressing RQ2)**. Groups would later provide potential answers to RQs accordingly. The next step was to synthesise the data.

4.9. Data synthesis

The purpose of data processing is to synthesize extracted data in order to answer RQs from Sec-

tion 3. Data extracted in Section 4.5 was divided into two groups. Each group contains a number of categories that emerged from examined publications. Category names were deduced from clustering items in each group.

Each category has its description and several papers addressing a particular subject. Results of data synthesis are available in Table 2.

5. Study results

We conducted a systematic mapping study according to the procedure described in Section 4. In total 42 publications were examined. During data extraction and synthesis, five categories of problems (corresponding to RQ1) and seven categories of proposed solutions (corresponding to RQ2) to the problems were identified for the selected studies. RQs findings are discussed in Sections 5.1 and 5.2.

5.1. Problems in estimation due to technical debt (RQ1)

We gathered five categories of problems in user-story estimation due to technical debt:

- **Business pressure on delivery** – 11 papers (i.e., 44% of publications that identified prob-

Table 2. Data synthesis results

	Category	Description	No. of studies	Sources
Identified problem categories for development teams during estimations (G1)	Business pressure on delivery	Studies showing business pressure of any kind on the project delivery (e.g., release project ahead of competition, new regulations introduced by public administration, raising company market value)	11	[S4], [S5], [S6], [S7], [S8], [S9], [S10], [S11], [S12], [S13], [S14]
	Lack of technical debt awareness in company	Studies showing that non-technical stakeholders are now aware of technical debt impact on estimations	5	[S11], [S15], [S16], [S17], [S18]
	No procedures for technical debt management	Studies stating a lack of any technical debt management techniques incorporated in software engineering process	3	[S16], [S19], [S20]
	Architectural technical debt impact	Studies providing samples where architecture technical debt had impact on software estimation and delivery	9	[S8], [S9], [S15], [S16], [S21], [S22], [S23], [S24], [S25]
	Source code analysis is not sufficient	Studies claiming that sole code analysis measurements are not enough in task estimation improvements	7	[S9], [S15], [S16], [S21], [S26], [S27], [S28]
	Total distinct studies			25
Identified proposed solution categories to mitigate technical debt in estimations problem (G2)	Tools for decision support	Studies indicating need of high level tools that will help business people to take development decision with technical debt consideration (e.g., which parts of the system will be affected by implementing particular feature, how much human resources needs to be involved)	14	[S7], [S8], [S9], [S11], [S29], [S30], [S31], [S32], [S23], [S33], [S34], [S24], [S18], [S35]
	Tools for estimation support	Studies stating the need of technical debt estimation tool for development team. Such tool would improve estimation accuracy	9	[S17], [S15], [S27], [S36], [S37], [S38], [S39], [S40], [S20]
	Portfolio approach (technical debt Items)	Studies proposing various catalogues of technical debt items managed by development team in structured manner. Newly introduced technical debt should be added to catalog	11	[S5], [S41], [S22], [S28], [S29], [S32], [S42], [S43], [S18], [S35], [S44]
	Architecture level technical debt visualization tool	Studies stating the need of managing technical debt on architectural level. Overview tool of a complex system that would show a map of potentially affected areas by new changes	7	[S21], [S15], [S27], [S9], [S16], [S22], [S24]
	Technical debt reduction in every sprint	Studies suggesting that a certain amount of time should be devoted to reducing technical debt by the development team	8	[S6], [S8], [S5], [S19], [S32], [S45], [S13], [S44]
	Code audit activity	Studies advising to conduct structured code audit periodically	3	[S6], [S19], [S12]
	Extra resources	Studies claiming that more resources such as people, infrastructure or budget are needed	2	[S45], [S22]
	Total distinct studies			37

lems) emphasised that business pressure was the key factor in estimations and therefore technical debt introduction. Hence, we think that this problem is widespread. In one of the studies, authors say: *The participants commonly acknowledged that technical debt is essentially a balance between software quality and business reality* [S6]. Authors list a number of reasons behind that statement: (1) *being contractually obligated to deliver the system under a tight deadline*, (2) *meeting deadlines to integrate with a partner product before release*, (3) *delivering in time for an upcoming trade show that presented food marketing opportunities*, (4) *developing a working prototype to secure investors funding* [S6].

- **Lack of technical debt awareness in company** – Five studies notice that non-technical stakeholders were unaware of technical debt impact on the project. In one study authors write: *From developer’s perspective, management remains largely unaware of technical debt and the value of managing it* [S15].
- **No procedures for technical debt management** – Authors of three publications inform about lack of any methodology in projects they have investigated. In one study we can find a statement: *Neither of the product lines had any specific approach for dealing with technical debt management and reduction* [S19].
- **Architectural technical debt impact** – Nine studies conclude that complex code architecture structure and its technical debt has an impact on estimations. Authors of one of the studies stated: *Architectural issues are the greatest source of technical debt... Architectural issues are difficult to deal with, since they were often caused many years previously* [S15].
- **Source code analysis is not sufficient** – This problem is brought by seven studies. Software engineers see that source code analysis does not show the whole picture of the system. This has an impact on estimations. One of the studies stated: *... technical debt is not only about code and code quality. Code analysis tools will identify a small*

number of black elements. Therefore, code analysis tools aren’t sufficient for identifying technical debt... [S26]⁵. In another study, authors write: *Tools do not capture the key areas of accumulating problems in technical debt* [S15].

5.2. Proposed solutions to mitigate the impact of technical debt on task estimation (RQ2)

Conducted research provides seven techniques for mitigating the impact of technical debt on estimations:

- **Tools for decision support** – This finding uncovers a communication gap between organisation units in an organisation. What was not expected was how widespread is the opinion that non-technical management should have a tool for better decision support in the project. As much as 14 of 37 papers (i.e., 38% of papers that identified solutions) emphasised such need.
- **Tools for estimation support** – Nine papers propose introducing estimation support tools for development teams. Authors of one study say *... by the later stages of the project the algorithm is more reliable than manual Planning Poker estimates and thus suitable as a tool for augmenting human effort estimation* [S36].
- **Portfolio approach (technical debt Items)** – As much as 11 of 37 papers that identified solutions propose managing technical debt in a structured way. Developers should fill “technical debt Item” cards so the team is aware of how much technical debt there is in the system. In one of the papers, authors write: *... managers expressed that the backlog would be used in the future... to reduce technical debt in small iterations* [S22].
- **Architectural-level technical debt visualisation tool** – Seven publications indicate the need for a high-level technical debt monitoring tool. A tool that will have the knowledge about technical debt not only in separate

⁵The black element refers to technical debt which was visually presented in Figure 2 on page 20 [S26].

system components but also between them and the system as a whole. Authors of one study stated: *Making the architectural debt visible provides the necessary information for making informed decisions for managing the potential impact of rework over time* [S21]. This issue is also mentioned by others: *The lack of tool support for accurately managing and tracking architectural sources of debt is a key issue...* [S15].

- **Technical debt reduction in every sprint** – Eight publications propose continuous technical debt reduction during every sprint. A related excerpt in one of the papers is as follows: *one participant described a policy of allocating 5 to 10 per cent of each release cycle to addressing technical debt* [S6].
- **Code audit activity** – Three papers ([S6], [S19], [S12]) propose periodical and systematic code audit actions conducted by the development team. Authors of one of the studies conclude: *... conduct audits with the entire development team to make technical debt visible and explicit; track it using a Wiki, backlog, or task board* [S6]
- **Extra resources** – Two papers propose adding extra resources such as people [S22], infrastructure or budget [S45] to the project. Such solutions may indicate a tight project schedule or an attempt to reaching the project deadline.

6. Discussion

The overall goal of this research was to identify problems, as well as proposed solutions occurring in estimations due to previously introduced technical debt. In this section we will present our interpretation of systematic mapping study results and their implications for academia and industry.

6.1. Problems in estimation due to technical debt

Business pressure on delivery and lack of technical debt awareness in management are related to the business perspective in a par-

ticular software project. The main purpose of building software is to support other processes. Managers and business officers are focused on growing the organization. Software support can give them a competitive advantage and that is why they force pressure on short software release cycles.

No procedures for technical debt management mentioned in three research papers indicate immature development process. This may be due to various reasons. Company owners may not be aware of the technical debt problem or may consider a particular project as a prototype where technical debt is not considered as a problem. On the other side, the project can be so big that introducing new development procedures is too cumbersome or too expensive. Finally, the development team may not know how to introduce such procedures.

Results such as **architectural technical debt impact** and **source code analysis is not sufficient**, can be interpreted differently. Those problems are more related to technical aspects. The **architectural technical debt impact** item is strongly bound to project evolution. For instance, the mainstream in web development is moving to cloud-based solutions and application containers providing better scalability and flexibility. Adjusting old software can be difficult and can be considered as a sample of architectural technical debt. **Source code analysis is also not sufficient** because engineers would adjust their code in such a way that it will pass the code analysis, but remind a poor quality.

Depending from which perspective we consider the situation different problems are present. In the worst-case scenario, all of them can occur in the organization and will slow down the development process even further.

6.2. Solutions to mitigate technical debt in estimations

Only one proposed solution focuses on non-technical stakeholders (**tools for decision support**). However, 38% of examined studies (14 of 37) state that this is the desired solution. This indicates the complicated nature of modern software solutions. Managers and decision-makers have difficulties

understanding the technical implications of their business decisions. Especially in competitive markets, where the software should be adjusted quickly, managers should see the results fast and be able to respond to them. Worth mentioning here are automatic code generators where solutions can be created without software engineers.

Another interesting interpretation arises from **portfolio approach (technical debt items), technical debt reduction in every sprint and code audit activity**. All of those solutions can be concluded as a need for deeper software development processes standardization and/or regulations. In other industries like medicine, maritime, aviation or automotive rules and regulations according to which certain procedures have to be conducted do exist. In IT there is ISO 25010 standard, but it is not mandatory to implement it.

The findings indicate that “Time To Market” has the biggest impact on schedule and the decision to repay or not the technical debt. The software solutions are too complicated and cannot be adapted fast enough in a rapidly changing world. An interesting fact the study uncovers is that source code analysis tools are not sufficient to cope with technical debt in estimations.

Based on the information from the performed SMS, we recommend focus future research on various decision-support levels. The complexity of software solutions grows and it is more difficult to get an overview from both business and technical perspectives. We propose that such decision-support research should take in consideration software maintenance and evolution.

7. Threats to validity

A systematic mapping study is conducted by people and thus an inevitable risk is related to the bias that may come from the choice of search engines/digital libraries and of search terms that may favour finding some studies and perhaps missing others. Hence, an important threat to the validity of this SMS is related to the search strategy employed and the possibility that we have not identified all relevant papers. The completeness of the search depends on the search string used, the scope of the search in terms

of selected search engines, as well as their limitations Brereton et al. [34]. For example, it is possible to extend the search query even further by adding additional words like “managing”. We do not think this is a significant threat. Nevertheless, it is still possible that after such extension the result set of papers would be a different, but (in our opinion) to a minor extent. To reduce this threat we selected a range of digital libraries and thus widened its scope. We also used a known set of references to validate the search terms before undertaking the mapping study and the search terms were amended where necessary (e.g., we included “change impact” that we initially missed).

The time window chosen by us (since 1992 till now) can be seen as a threat. That said, we think that the knowledge about technical debt, software development and programming languages has evolved to such extent that we probably do not lose anything crucial excluding papers before 1992.

We also conducted snowballing to limit the possibility of missing relevant papers. Only one additional paper was identified by searching the references of included studies.

A closely related threat is that “grey literature” may not be found due to the nature of digital libraries used. Snowballing can be seen as a partial solution to limit this threat as references of the papers found in digital libraries may include “grey literature” as well.

It is also worth mentioning that categories synthesised from publications data extraction emerged from our best understanding of the topic. We proposed category names presented in Table 2 based on our experience in software engineering.

We limited the scope of our search to articles written in English. Thus the presented results can be biased by omitting publications written in other languages (e.g., Chinese). However, we based our research on the most popular language among software engineering researchers and practitioners.

A search-related limitation of this mapping study is that the search only covers publications that were included in the chosen digital libraries before January 2019. This date is related to the moment when the mapping study was performed.

Table 3. Evaluation of our mapping process (see [25])

Rubric	Score	Description
Need for review	1	Partial evaluation – motivations and questions are provided.
Choosing the search strategy	1	Minimal evaluation – two search strategies (automated database search and snowballing) have been used.
Evaluation of the search	2	Partial evaluation – at least one action has been taken to improve the reliability of the search and the inclusion/exclusion.
Extraction and classification	2	Partial evaluation – at least one action has been taken to increase the reliability of the extraction process, and research type and method have been classified.
Study validity	1	Full evaluation – threats and limitations are described.

It is therefore probable (due to the fact that technical debt is perceived as an interesting topic) that a number of other relevant papers will have been published since this date that we have not included in this mapping study. However, this limitation is difficult to avoid and the common solution is to conduct a new search and/or snowballing to update the results of the mapping study.

Additionally, Table 3 presents an evaluation of our mapping process on a basis of the quality checklist rubric criteria (defined by Petersen et al. [25]) including: identifying the need for SMS, study identification, data extraction and classification, as well as validity discussion.

8. Conclusions

In this systematic mapping study, 42 out of 2003 relevant publications were selected. 41 from query search in five digital databases and one additional from the snowballing procedure. The contribution of this study is a categorisation of technical debt related issues in task estimations and proposed solutions to the issues presented in Section 5. Five problems and seven solutions identified in literature have been categorised. Furthermore, the majority of identified categories of problems and solutions include real-life examples describing industry cases.

The technical debt impact on task estimation is an important issue to address. Our SMS shows seven approaches to extend the current state of technical debt management. We conclude that the task estimation accuracy can

be further improved in one of the following directions:

- business direction – research on how the managers can gain more insight into the software system that is supporting their business. Understand the system’s current limitations and the impact of new business decisions on it. That implies research on how software engineers can improve communication with “the business.”
- operational direction – research on software systems maintainability and development routines. That includes new ways of formalizing and structuring software components, data flows, integrations and others so that it would be easy to analyse new requirements impact on the software project.

The problem of business pressure on features delivery has appeared in our findings on several occasions. Our further research will focus on decision-making tools. In our opinion, there is a room for improvement that will potentially help development teams to measure the impact of technical debt on estimations with more accurate precision.

References found during our systematic mapping study

- [S1] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, “Managing technical debt in software-reliant systems,” in *Proceedings of the FSE/SDP*

- Workshop on Future of Software Engineering Research*, FoSER '10. ACM, 2010, pp. 47–52.
- [S2] C. Izurieta, I. Ozkaya, C. Seaman, P. Kruchten, R.L. Nord, W. Snipes, and P. Avgeriou, “Perspectives on managing technical debt: A transition point and roadmap from dagstuhl,” in *Joint Proceedings of the 4th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2016) and 1st International Workshop on Technical Debt Analytics (TDA 2016)*, 2016, pp. 84–87. [Online]. <http://ceur-ws.org/Vol-1771/paper15.pdf>
- [S3] P. Kruchten, R.L. Nord, I. Ozkaya, and D. Flessi, “Technical debt: Towards a crisper definition report on the 4th international workshop on managing technical debt,” *ACM SIGSOFT Software Engineering Notes*, Vol. 38, No. 5, 2013, pp. 51–54.
- [S4] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F.Q.B. Da Silva, A.L.M. Santos, and C. Siebra, “Tracking technical debt – An exploratory case study,” in *Proceedings of the 27th IEEE International Conference on Software Maintenance*, ICSM '11. IEEE Computer Society, 2011, pp. 528–531.
- [S5] K. Power, “Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options,” in *Proceedings of the 4th International Workshop on Managing Technical Debt*, 2013, pp. 28–31.
- [S6] E. Lim, N. Taksande, and C. Seaman, “A balancing act: What software practitioners have to say about technical debt,” *IEEE Software*, Vol. 29, No. 6, 2012, pp. 22–27.
- [S7] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, “An enterprise perspective on technical debt,” in *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11. ACM, 2011, pp. 35–38.
- [S8] Z. Codabux and B. Williams, “Managing technical debt: An industrial case study,” in *Proceedings of the 4th International Workshop on Managing Technical Debt*, MTD '13. IEEE Press, 2013, pp. 8–15.
- [S9] A. Martini, J. Bosch, and M. Chaudron, “Investigating architectural technical debt accumulation and refactoring over time,” *Information and Software Technology*, Vol. 67, No. C, 2015, pp. 237–253.
- [S10] N. Ramasubbu, C.F. Kemerer, and C.J. Woodard, “Managing technical debt: Insights from recent empirical evidence,” *IEEE Software*, Vol. 32, No. 2, 2015, pp. 22–25.
- [S11] J. Bohnet and J. Döllner, “Monitoring code quality and development activity by software maps,” in *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11. ACM, 2011, pp. 9–16.
- [S12] J.C. Rocha, V. Zapalowski, and I. Nunes, “Understanding technical debt at the code level from the perspective of software developers,” in *Proceedings of the 31st Brazilian Symposium on Software Engineering, SBES*, 2017, pp. 64–73.
- [S13] R.K. Gupta, P. Manikreddy, and K.C. Arya, “Pragmatic scrum transformation: Challenges, practices and impacts during the journey A case study in a multi-location legacy software product development team,” in *Proceedings of the 10th Innovations in Software Engineering Conference, ISEC*, 2017, pp. 147–156.
- [S14] N. Rios, R.O. Spínola, M.G. de Mendonça Neto, and C.B. Seaman, “A study of factors that lead development teams to incur technical debt in software projects,” in *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA*, 2018, pp. 429–436.
- [S15] N.A. Ernst, S. Bellomo, I. Ozkaya, R.L. Nord, and I. Gorton, “Measure it? Manage it? Ignore it? Software practitioners and technical debt,” in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015*. ACM, 2015, pp. 50–60.
- [S16] J. Yli-Huumo, A. Maglyas, and K. Smolander, “How do software development teams manage technical debt? – An empirical study,” *Journal of Systems and Software*, Vol. 120, 2016, pp. 195–218.
- [S17] C.Y. Chen, C.W. She, and J.D. Tang, “An object-based, attribute-oriented approach for software change impact analysis,” in *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management*, 2007, pp. 577–581.
- [S18] R.R. de Almeida, U. Kulesza, C. Treude, D.C. Feitosa, and A.H.G. Lima, “Aligning technical debt prioritization with business objectives: A multiple-case study,” in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution, ICSME*, 2018, pp. 655–664.
- [S19] J. Yli-Huumo, A. Maglyas, and K. Smolander, “The sources and approaches to management

- of technical debt: A case study of two product lines in a middle-size finnish software company,” in *Proceedings of the 15th International Conference Product-Focused Software Process Improvement*, 2014, pp. 93–107.
- [S20] T. Besker, A. Martini, J. Bosch, and M. Tichy, “An investigation of technical debt in automatic production systems,” in *Proceedings of the XP2017 Scientific Workshops*, 2017, pp. 6:1–6:7.
- [S21] R.L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, “In search of a metric for managing architectural technical debt,” in *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA-ECSA ’12*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 91–100.
- [S22] J. Yli-Huumo, A. Maglyas, K. Smolander, J. Haller, and H. Törnroos, “Developing processes to increase technical debt visibility and manageability – An action research study in industry,” in *Proceedings of the International Conference on Product-Focused Software Process Improvement*, Lecture Notes in Computer Science, Vol. 10027. Springer International Publishing, 2016, pp. 368–378.
- [S23] C. de Souza and D. Redmiles, “An empirical study of software developers’ management of dependencies and changes,” in *Proceedings of the 30th International Conference on Software Engineering*, 2008, pp. 241–250.
- [S24] A. Martini, E. Sikander, and N. Madlani, “A semi-automated framework for the identification and estimation of architectural technical debt: A comparative case-study on the modularization of a software component,” *Information and Software Technology*, Vol. 93, 2018, pp. 264–279.
- [S25] T. Besker, A. Martini, and J. Bosch, “The pricey bill of technical debt: When and by whom will it be paid?” in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution, ICSME*, 2017, pp. 13–23.
- [S26] P. Kruchten, R.L. Nord, and I. Ozkaya, “Technical debt: From metaphor to theory and practice,” *IEEE Software*, Vol. 29, No. 6, 2012, pp. 18–21.
- [S27] Z. Li, P. Liang, and P. Avgeriou, “Architectural technical debt identification based on architecture decisions and change scenarios,” in *Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture*, 2015, pp. 65–74.
- [S28] N. Zazworka, R.O. Spínola, A. Vetro, F. Shull, and C. Seaman, “A case study on effectively identifying technical debt,” in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, EASE ’13*. ACM, 2013, pp. 42–47.
- [S29] C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull, and A. Vetro, “Using technical debt data in decision making: Potential decision approaches,” in *Proceedings of the Third International Workshop on Managing Technical Debt, MTD ’12*. IEEE Press, 2012, pp. 45–48.
- [S30] C. Fernández-Sánchez, J. Garbajosa, and A. Yagüe, “A framework to aid in decision making for technical debt management,” in *Proceedings of the 7th IEEE International Workshop on Managing Technical Debt, MTD@ICSME*, 2015, pp. 69–76.
- [S31] H. Jason and R. Günther, “When-to-release decisions in consideration of technical debt,” in *Proceedings of the Sixth International Workshop on Managing Technical Debt, MTD@ICSME*, 2014, pp. 31–34.
- [S32] R.K. Gupta, P. Manikreddy, S. Naik, and K. Arya, “Pragmatic approach for managing technical debt in legacy software project,” in *Proceedings of the 9th India Software Engineering Conference, ISEC ’16*. ACM, 2016, pp. 170–176.
- [S33] A. Pacheco, G. Marín-Raventós, and G. López, “Designing a technical debt visualization tool to improve stakeholder communication in the decision-making process: A case study,” in *Proceedings of the 12th IFIP WG 8.9 Working Conference on Research and Practical Issues of Enterprise Information Systems*, 2018, pp. 15–26.
- [S34] T. Amanatidis, A. Chatzigeorgiou, and A. Ampatzoglou, “The relation between technical debt and corrective maintenance in PHP web applications,” *Information and Software Technology*, Vol. 90, 2017, pp. 70–74.
- [S35] M. M. Bomfim and V. A. Santos, “Strategies for reducing technical debt in agile teams,” in *Proceedings of the Brazilian Workshop on Agile Methods*. Springer International Publishing, 2017, pp. 60–71.
- [S36] K. Moharreri, A.V. Sapre, J. Ramanathan, and R. Ramnath, “Cost-effective supervised learning models for software effort estimation in agile environments,” in *Proceedings*

- of the Computer Software and Applications Conference (COMPSAC), 2016, pp. 135–140.
- [S37] A. Nugroho, J. Visser, and T. Kuipers, “An empirical model of technical debt and interest,” in *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD ’11. ACM, 2011, pp. 1–8.
- [S38] B. Tanveer, “Guidelines for utilizing change impact analysis when estimating effort in agile software development,” in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE*, 2017, pp. 252–257.
- [S39] S.J. Kabeer, M. Nayebi, G. Ruhe, C. Carlson, and F. Chew, “Predicting the vector impact of change – an industrial case study at bright-squid,” in *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM*, 2017, pp. 131–140.
- [S40] B. Tanveer, L. Guzmán, and U.M. Engel, “Effort estimation in agile software development: Case study and improvement framework,” *Journal of Software: Evolution and Process*, Vol. 29, No. 11, 2017.
- [S41] K. Schmid, “A formal approach to technical debt decision making,” in *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures, QoSA ’13*. ACM, 2013, pp. 153–162.
- [S42] Y. Guo and C. Seaman, “A portfolio approach to technical debt management,” in *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD ’11. ACM, 2011, pp. 31–34.
- [S43] Y. Guo, R.O. Spínola, and C. Seaman, “Exploring the costs of technical debt management – A case study,” *Empirical Software Engineering*, Vol. 21, No. 1, 2016, pp. 159–182.
- [S44] P. Mohagheghi and M.E. Aparicio, “An industry experience report on managing product quality requirements in a large organization,” *Information and Software Technology*, Vol. 88, 2017, pp. 96–109.
- [S45] Z.S. Hossein Abad, R. Karimpour, J. Ho, S.M. Didar-Al-Alam, G. Ruhe, E. Tse, K. Barabash, and I. Hargreaves, “Understanding the impact of technical debt in coding and testing: An exploratory case study,” in *Proceedings of the 3rd International Workshop on Software Engineering Research and Industrial Practice, SER& IP ’16*. ACM, 2016, pp. 25–31.

References

- [1] E. Mueller, X.L. Chen, and R. Riedel, “Challenges and requirements for the application of Industry 4.0: A special insight with the usage of cyber-physical system,” *Chinese Journal of Mechanical Engineering*, Vol. 30, No. 5, 2017, pp. 1050–1057.
- [2] M. Hermann, T. Pentek, and B. Otto, “Design principles for Industrie 4.0 scenarios,” in *Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 3928–3937.
- [3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, Vol. 29, No. 7, 2013, pp. 1645–1660.
- [4] J. Highsmith and A. Cockburn, “Agile software development: The business of innovation,” *Computer*, Vol. 34, No. 9, 2001, pp. 120–122.
- [5] L. Madeyski, “On the effects of pair programming on thoroughness and fault-finding effectiveness of unit tests,” in *Product-Focused Software Process Improvement*, Lecture Notes in Computer Science, J. Münch and P. Abrahamsson, Eds. Springer Berlin Heidelberg, 2007, Vol. 4589, pp. 207–221.
- [6] L. Madeyski, “The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment,” *Information and Software Technology*, Vol. 52, No. 2, 2010, pp. 169–184.
- [7] L. Madeyski, *Test-Driven Development: An Empirical Evaluation of Agile Practice*. (Heidelberg, London, New York): Springer, 2010.
- [8] L. Madeyski and Ł. Szała, “The impact of test-driven development on software development productivity – An empirical study,” in *Software Process Improvement*, Lecture Notes in Computer Science, P. Abrahamsson, N. Baddoo, T. Margaria, and R. Messnarz, Eds. Springer Berlin Heidelberg, 2007, Vol. 4764, pp. 200–211.

- [9] P. Kruchten, R.L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, Vol. 29, No. 6, 2012, pp. 18–21.
- [10] W. Cunningham, "The WyCash portfolio management system," in *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '92. New York, NY, USA: ACM, 1992, pp. 29–30.
- [11] M. Fowler, "Technical debt quadrant," 2009. [Online]. <https://martinfowler.com/bliki/TechnicalDebtQuadrant.html>
- [12] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11. ACM, 2011, pp. 1–8.
- [13] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, Vol. 86, No. 6, 2013, pp. 1498–1516.
- [14] M. Cohn, *Agile Estimating and Planning*. Pearson Education, 2005.
- [15] S. Hoogendoorn, *This is Agile: Beyond the Basics. Beyond the Hype. Beyond Scrum*. Dymaxicon, 2014.
- [16] "The bucket system," 2017. [Online]. <http://www.agileadvice.com/wp-content/uploads/2013/07/H10-Estimation-The-Bucket-System.pdf>
- [17] C. Fernández-Sánchez, J. Garbajosa, and A. Yagüe, "A framework to aid in decision making for technical debt management," *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, 2015, pp. 69–76.
- [18] C. Fernández-Sánchez, J. Garbajosa, C. Vidal, and A. Yagüe, "An analysis of techniques and methods for technical debt management: A reflection from the architecture perspective," in *Proceedings of the Second International Workshop on Software Architecture and Metrics*, SAM '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 22–28.
- [19] L.F. Ribeiro, M.A.d.F. Farias, M. Mendonça, and R.O. Spínola, "Decision criteria for the payment of technical debt in software projects: A systematic mapping study," in *Proceedings of the 18th International Conference on Enterprise Information Systems*, ICEIS 2016. SCITEPRESS - Science and Technology Publications, Lda, 2016, pp. 572–579.
- [20] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, Vol. 101, 2015, pp. 193–220.
- [21] W.N. Behutiye, P. Rodríguez, M. Oivo, and A. Tosun, "Analyzing the concept of technical debt in the context of agile software development: A systematic literature review," *Information and Software Technology*, Vol. 82, 2017, pp. 139–158.
- [22] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review," *Information and Software Technology*, Vol. 64, 2015, pp. 52–73.
- [23] N.S. Alves, T.S. Mendes, M.G. de Mendonça, R.O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt," *Information and Software Technology*, Vol. 70, No. C, 2016, pp. 100–121.
- [24] B.A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-Based Software Engineering and Systematic Reviews*. Chapman and Hall/CRC, 2016.
- [25] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, Vol. 64, 2015, pp. 1 – 18.
- [26] "IEEE Xplore Digital Library," 2017. [Online]. <http://ieeexplore.ieee.org>
- [27] "ACM digital library," 2017. [Online]. <http://dl.acm.org>
- [28] "Springer Link," 2017. [Online]. <https://link.springer.com>
- [29] "Science Direct," 2017. [Online]. <http://www.sciencedirect.com>
- [30] "Scopus Preview," 2017. [Online]. <https://www.scopus.com>
- [31] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Pro-*

- ceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14. ACM, 2014, pp. 38:1–38:10.
- [32] M. Ivarsson and T. Gorschek, “A method for evaluating rigor and industrial relevance of technology evaluations,” *Empirical Software Engineering*, Vol. 16, No. 3, 2011, pp. 365–395.
- [33] J. Cohen, “A coefficient of agreement for nominal scales,” *Educational and Psychological Measurement*, Vol. 20, No. 1, 1960, pp. 37–46.
- [34] P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, “Lessons from applying the systematic literature review process within the software engineering domain,” *Journal of Systems and Software*, Vol. 80, No. 4, 2007, pp. 571–583.

SIoT Framework: Towards an Approach for Early Identification of Security Requirements for Internet-of-things Applications

Ronald Jabangwe*, Anh Nguyen-Duc**

**The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Software Engineering, Denmark / Software Improvement Group, SIG Nordics.*

***School of Business, University of South Eastern Norway, Norway, Department of Business and IT*
rja@mmmi.sdu.dk / r.jabangwe@sig.eu, anh.nguyen.duc@usn.no

Abstract

Background: Security has become more of a concern with the wide deployment of Internet-of-things (IoT) devices. The importance of addressing security risks early in the development lifecycle before pushing to market cannot be over emphasized. Aim: To this end, we propose a conceptual framework to help with identifying security concerns early in the product development lifecycle for Internet-of-things, that we refer to as SIoT (Security for Internet-of-Things). Method: The framework adopts well known security engineering approaches and best practices, and systematically builds on existing research work on IoT architecture. Results: Practitioners at a Norwegian start-up company evaluated the framework and found it useful as a foundation for addressing critical security concerns for IoT applications early in the development lifecycle. The output from using the framework can be a checklist that can be used as input during security requirements engineering activities for IoT applications. Conclusions: However, security is a multi-faced concept; therefore, users of the SIoT framework should not view the framework as a panacea to all security threats. The framework may need to be refined in the future, particularly to improve its completeness to cover various IoT contexts.

Keywords: security requirement; Internet-of-things; Software Engineering; Requirement Engineering; Security Framework

1. Introduction

Within the past decade, we have witnessed the rapid growth of commercial systems that deeply integrate software, hardware and the contextual environment. The most notable are Internet-of-Things (IoT), Industry 4.0, cyber-physical systems, and smart wearable devices. The number of (IoT) devices being introduced in the market has been increasing drastically with the number of connected devices approaching 15 billion [1]. This trend is expected to continue, with an estimate of 26 billion network connected devices by the year 2020 [1].

Security has become even more important as the number of “things” connected increases

through the vulnerable internet and other networks. The border between software and hardware parts is less visible when it comes to providing customer value. Internet-of-things integrate both sensors, connectivity infrastructure and processors with a software platform. The consideration of security, therefore, needs to be in a holistic view that combines both software and hardware parts of the system. Security issues are not new and have been a concern for years to manufacturers. However, security in software-intensive products is often neglected or treated as an afterthought. Business pressure, time-to-market and reduction of development costs are among factors that drive the treatment of security as an add-on feature.

Software Engineering (SE) researchers are looking for a way to address security concerns as early as possible in the development and operation of software-intensive products [2, 3]. In our study, we refer to “security concerns” for a given system as vulnerabilities, risks or threats that can negatively impact the security properties of the system, specifically, confidentiality, integrity and availability. The aim is to promote security-by-design, which leads to having a proactive rather than a reactive approach for addressing security. The goal, which is also the same for threat modeling [4, 5], is to help with identifying security concerns for a given system. These security concerns can potentially be mapped to security requirements, which in-turn can help with designing secure systems.

Security requirements affect all aspects of the design, development, deployment, and maintenance of complex systems that provide customer value. To address security early in the development cycle, security aspects should be considered from the planning and requirements phase, and throughout all the other phases. In response to the urgent need to deal with security in software-intensive product development [6–8], we aim at proposing a comprehensive approach to identify security issues in the context of cyber-physical systems, specifically focusing on Internet-of-things. More importantly, the approach will handle data security issues as an input for both product development and operation. Last but not least, the approach should be lightweight and easy to adopt in various sizes of organizations, particularly start-up companies. This is due to the emerging number of Internet-of-things developed by start-ups.

A plethora of research work on software engineering exists in relation to software security and requirements engineering, and there is also a growing interest in Internet-of-things. Internet-of-things development is challenging due to the multiple cross-cutting concerns, such as connectivity, security and the lack of high-level abstractions to address both the large-scale and heterogeneity [9]. The heterogeneousness of Internet-of-things introduces additional complexity to software layer development, in particularly com-

plex data flow and architectural cross-cutting concerns [6, 10]. Consequently, securing the Internet-of-things application development would require joint knowledge from data security, requirements engineering and Internet-of-things architecture. Security should be addressed early in the development process by ensuring that requirements are clearly defined that when implemented, prevent or mitigate security issues. It is noted that we do not aim at generating specific security requirements through our framework. As a preliminary result from a qualitative survey of quality concerns and practices in Internet-of-things startups, we identified the need for early consideration of security requirements and mapping them into actual implementation. Addressing the issues early avoids costly rework late in the development process. To this end, we take a software engineering approach for addressing data security concerns early through a lightweight framework, SIoT (Security for Internet-of-things Applications). For Internet-of-things end-users, this can reduce safety risks and potentially improves privacy and data protection.

Designing secure systems requires understanding the complex interaction between different parts of architecture and the security threats for those parts. The SIoT framework, which takes a layered view of the architecture of Internet-of-things applications, provides a foundation for promoting that thought process. The aim is not to generate specific security requirements through our framework. However, the output from using the framework can be a checklist that can be used to help with identifying security requirements for IoT applications.

The remainder of the paper is organized as follows: Section 2 introduces basic understanding about Internet-of-things products and security identification approaches. Section 3 describes the need for a lightweight and early-stage framework for Internet-of-things development via a preliminary industrial survey. Section 4 describes our framework. Section 4 presents the case company for which the framework was developed and would be evaluated. The discussion and conclusion are in Section 5.

2. Background and related work

2.1. Existing IoT frameworks

The framework that is more similar to our framework is work of Meridji et al. [11]. The framework is intended to help developers identifying, specifying and measuring security requirements. The design of the overall framework is based on the use of the interdependency graphs (SIG) and the CIA triad, i.e., confidentiality integrity, availability, confidentiality. Whilst the proposed framework was systematically developed, it is, however based on generic models and generic view of security. As a result, the framework takes a broad and generic view of system engineering. In contrast, our framework is intended for a specific type of system, i.e., IoT systems. Another aspect that differs between our framework and the framework proposed by Meridji et al. [11] is in how security aspects are derived. The framework by Meridji et al. [11] relies on three international standards (ECSS, IEEE and ISO) for deriving security requirements. Whereas our framework emphasizes the need to focus on the architecture design in order to derive relevant security concerns for the specific system. Our motivation for going with this approach is that the architecture may differ from system to system, and how a system is designed is crucial for understanding how best to strengthen the security of the overall system. Because our framework focuses more on the architecture of IoT systems, it allows for more flexibility in terms of adoptability and adaptability to various IoT systems. Ammar et al. [12] report on a survey of existing IoT platforms that offer cloud-based services such as AWS IoT. In the report they make an assessment of eight platforms focusing on the features offered by the platforms for developing IoT applications, including hardware and security features. Our framework takes a software engineering and process approach for developing IoT applications. The overarching aim is to provide developers with an approach that can help them with identifying security concerns of their specific IoT applications, irrespective of the platform that they use.

2.2. Security requirement identification in software development

Security requirements have traditionally been considered to be non-functional or quality requirements [2, 13]. Like other non-functional requirements, security requirements need to be described in the way that they can be implemented later. Carnegie Mellon University was among the first to propose a methodology (SQUARE) to help organizations build security into the early stages of the production life cycle [14]. The SQUARE approach includes nine steps that require formal participation of requirements engineers and other stakeholders of an IT project. The team starts with outlining security goals, threats identification and risk assessment based on a full understanding of the relevant system. After that, the team decides on the best method for eliciting initial security requirements from stakeholders, and to elicit an initial set of security requirements. In the final step, security requirements are inspected to ensure consistency and accuracy. However, the methodology is at a high level of abstraction and is not specific to a particular domain.

Several researchers have focused on tools and methods for identification of security requirements, for instance, misuse cases [15], goal and anti-goal analysis [16], and patterns of security goals [17]. These approaches are proposed regardless of the context of software development and operation. Security concerns should be considered not only in the early stage of product development, but also as a continuous integral element of product development. Despite the benefits that Agile software development promises, there are security challenges faced within the paradigm that can manifest into vulnerable software products [18]. In turn, this can significantly impact the longevity of the software product on the market. There are studies that adopt and adapt agile approaches in order to ensure that security initiatives are addressed, e.g., Beznosov's work [19] and Ghani's work [20]. However, it is also critical to have a framework that is specific to Internet-of-things contexts that not only helps address security concerns but also can be adopted into agile software development processes.

2.3. Non-functional requirement modelling

Modelling and documentation techniques that can also be used when implementing approaches for collecting, categorizing and prioritizing security requirements are attack trees, abuse cases, abuser stories, misuse cases and fault trees [21]. An attack tree is a tree-like representation of the different ways that an identified asset can be attacked based on attack goals. Abuse cases are descriptions of how a user of a system or the system can be attacked or abused. Abuser stories help capture and describe likely goals of an attacker. Unlike user stories that are written from the perspective of a user of a system, abuser stories are written from the perspective of an attacker. Misuse cases are based on use cases, but they describe, using, for example, UML use case diagrams, and how malicious activities can be carried out on the system. A fault tree is a deduction approach for analyzing system failures and security concerns using graphical Boolean logic. These approaches can also be used to support the SQUARE method or any similar approaches. Nevertheless, modelling of security requirements is out of the scope of this paper. We only focus on providing a framework to help with the process of identifying security requirements in Internet-of-things applications.

2.4. IoT product development

From a technological perspective, the implementation of Internet-of-things typically requires the combination of hardware, software and middleware components collaborating with each other [22]. Hardware used for Internet-of-things include sensors, actuators, and processors that can be added to existing core hardware components, and integrated to manage and operate the functionality of connected things. Communication protocols such as MQTT, AMQP, XMPP, and Zigbee enable the communication between the sensor devices and the cloud [23]. A typical Internet-of-things product will have a “cloud” part including an application platform that provides fundamental operating en-

vironments for Internet-of-things applications. Internet-of-things applications, which employs web or mobile interfaces, provide functionalities to store, process and analyze a vast amount of time series-based machine data. There exist various architectural views on Internet-of-things systems depending on research goals. Based on existing classifications [22, 24], we adopted a 4-layer view on Internet-of-things systems with the purpose of differentiating security concerns and resolution techniques among layers. The layers are Application tier, Network tier, Sensor tier, and Data processing tier, which will be described in the SIIoT framework (Section 4).

Internet-of-things development is challenging due to the multiple cross-cutting concerns, such as connectivity, security and the lack of high-level abstractions to address both the large-scale and heterogeneity [9]. Patel et al. proposed a development framework that separates Internet-of-things into four concerns: architecture, domain, platform and deployment concerns [9]. However, the authors do not explicitly explore the elicitation and implementation of security concerns.

2.5. Identification and modelling security requirements in IoT development

There are other research articles that address security requirements for IoT applications [25–29]. Babar et al. proposed a framework that separates security concerns for software and hardware parts of embedded systems [25]. Although the need for built-in security framework is emphasized, the model was not validated. Jacobsson et al. proposed a risk analysis for smart home systems based on architectural views [26]. Gan et al. suggested several security requirements for Internet-of-things in their analysis [27]. Ahmad et al. proposed a model to capture security and privacy properties in Internet-of-things [28]. They point out common security challenges, but they are not categorized into system architectural dimensions. Kim et al. discussed the security concerns according to system tiers and system development phases [29]. The proposal is however subjective without validation.

Apart from the industry evaluation, our framework differs in that it adopts the well known

SQUARE methodology [14], and best practices for security engineering (e.g., [5, 32]) and adapts them for Internet-of-things contexts

3. Industrial demand on a security modelling framework

In order to motivate the need of a security modelling framework for industry, in this section, we provide preliminary results of a qualitative survey that we are currently performing on Internet-of-things startups. Early results support the notion as they suggest that startups need assistance with a framework for identifying security concerns early in the software development process. The preliminary results are shown in Table 1. In the on-going survey, we are surveying the state-of-practice of Internet-of-things application development, focusing specifically on exploring Internet-of-things development practices among IT startups. We aim at collecting as many participants as possible. There is no limitation on the type of companies in our survey as we would like to have a variety of the sample. The survey was designed in 2015 and is an on-going effort. Participants are being searched from three channels (1) our professional network, (2) regional incubators and accelerator programs, (3) and startup portal, i.e., Startup Norway and Crunchbase. Participants who accept our invitation are also invited to participate in a one-hour interview.

We used semi-structured interviews to enable open-end answers from participants. Our interview process has four parts (1) background information about business and product, (2) prototyping and production development practices and challenges (3) quality concerns and testing, and (4) final reflection. The full interview question list is shown in Appendix A. We found eleven Internet-of-things startups that are relevant to the scope of this study. This is a subset from

our IoT survey, from which we can extract information about security requirements. It is possible that the other companies also have similar concerns, but this is not explicitly mentioned in the survey.

Our previous work reveals some of the prototyping and development challenges of such startups, i.e., insufficient testing, technical debt, balancing agility and quality, etc. [33, 34]. Table 1 summarizes how security concerns are considered and managed in the eleven Internet-of-things startups from our survey. In the table, “foundation year” is the year the company was officially formed. Regardless of the startups’ active time, a startup can be in an idealization, a prototyping or in a production state [35]. Agile approaches are clearly common across the cases. Many companies report that they adopt certain ways of Agile in developing (part of) their products. Some report waterfall and adhoc approaches as their preferred approaches when dealing with the production of the hardware parts. In the right-most column in Table 1 we also reveal as a part of the product quality assurance practices, how security concerns were considered and managed.

Table 1 reveals that 80% of our cases emphasize the importance of security for their business, regardless of the startup stage. Security goals are often established in both startups at prototyping and production phases. Security is considered a significant concern, and in some cases as an essential value proposition in the companies’ business models. The consideration occurs at different levels, organizational, managerial and technical levels, for instance:

Security is the main infrastructure of Internet-of-things applications. As it is everywhere and one cannot think of compromising the everyday equipment they use (C03).

Table 1. Security goals

Goals	Description
Confidentiality	Ensure that data is not revealed to or accessed by unauthorized individuals [30, 31]
Availability	Ensure that authorized users can access and use data on demand [30, 31]
Integrity	Prevent unauthorized tampering of data when it is being processed, or in transit or when at rest [30, 31]

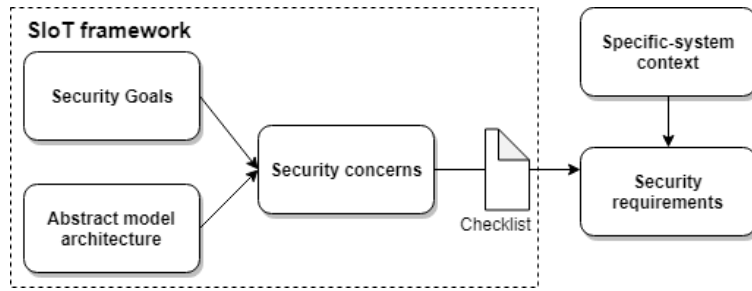


Figure 1. The scope of SIoT framework

As shown in Figure 1, the SIoT framework consists of the following three main components:

1. Security goals.
2. Internet-of-things abstract model architecture.
3. Internet-of-things security concerns.

Devices in Internet-of-things applications communicate through Internet and share their data over the network. So, there are huge chances of vulnerability (C05).

However, there is a limited action on implementing security concerns. 30% of the surveyed companies did not implement any security-related features. 40% of the cases have their security dependent on external vendors or open source components. There are only three cases that implement security as a part of their competitive strategy, as illustrated below:

Our data can be traced and exposed over the network in case of lack of proper security measures. Security features play a key role in Internet-of-things applications (C05).

The two pillars of simple access and security must work in unison. The first provides a simple way to securely connect devices to the Wi-Fi network, while the second ensures only the IoT application can traverse the Wi-Fi network from the IoT device to the server. This can help prevent malicious attacks (C07).

There is a lack of clarity of a systematic approach for mapping security goals to actual actions for addressing security concerns. For instance, it is not clear how startups cooperate security concerns into the product architecture, at different times of consideration (i.e., prototyping

or production). We also recognize some startups (40% of the total number of cases) that perceive security as a dependent concern on open source community or third-party providers. All in all, the preliminary observations of the 10 Internet-of-things startups suggest a methodological need of a systematic approach for considering security during Internet-of-things product development life cycle and practices.

Overall, we found only one case in which the company was taking steps to implement a methodological approach for security assurance. The company representatives explained that they do it because of market demands in their domain. Hence it helps them gain a competitive advantage over its competitors. The company was willing to participate in the evaluation of the framework because of their interest in methods for effectively addressing security concerns. More details of the company are presented in Section 5 of the paper.

4. The proposed conceptual SIoT framework

The SIoT (Security for Internet-of-things Applications) framework adopts the well known SQUARE methodology [14], and best practices for security engineering (e.g., [5, 32] and adapts them for Internet-of-things contexts. The framework also builds on existing work on Internet-of-things applications [22, 24, 36–38].

4.1. Security goals

Maintaining data confidentiality, integrity, availability are the primary goals for data security initiatives [39–41]. The three goals are also re-

Table 2. Internet-of-things application and Security consideration in startups

ID	Found year	Startup stage	Startup product	Development method	Thoughts	Actions
C01	2009	Prototyping	A underwater camera	Adhoc	Security was not considered at this stage	Quality perceived at open source module
C02	2013	Prototyping	A tracking device for shipments	Agile	Quality consideration i.e., robustness and security at software tier	Outsourced: Quality testing was done by subcontractors
C03	2011	Production	A mobile muscle trainer	Agile	After thought on security at software tier	Outsourced: Quality testing of the hardware tier was outsourced
C04	2015	Production	connected smart home solution	Waterfall and Agile	Importance of security at software and cloud tier	Depending on security of third party modules
C05	2015	Production	Home electricity usage management system	Agile for IoTs related development	Security concerns at three components: circuits, mobile apps and cloud	Implementing security features at various tiers
C06	2016	Prototyping	A navigating device for visually impaired individuals	Agile	Security is the most prominent feature	No
C07	2016	Prototyping	A car remote controller	Agile at start, Waterfall afterward	Security as a main concern	Implementing security features at various tiers
C08	2014	Production	A predictive analytic platform for vehicles	Agile	Security is as important as usability.	Limited
C09	2012	Prototyping	A body index tracking	Agile	Security concerns at methodological, organizational and technical level	Experimenting at methodological level
C10	2015	Prototyping	A water farming management system	Adhoc	Security concerns at organizational level	No
C11	2013	Prototyping	Glucose monitoring device	Adhoc	Security concerns at organizational and technical level	No

ferred to as the CIA triad [40]. The definitions for the security goals are in Table 2. Our proposal is to break down security into the three goals and then identifying security concerns that need to be addressed in order to realize each goal. This approach will help with addressing security from different but critical perspectives for protecting data for Internet-of-things applications.

There are other security attributes, such as authorization, authentication, and non-repudiation that can be perceived as independent categories. However, in line with Bass et al. [41], we also believe these attributes support the security goals outlined in Table 2. For example, authorization is intended to ensure that access to data is based on user privileges. This can be traced to confidentiality. Authentication is about verifying users to ensure confidentiality and integrity. Non repudiation can be traced to confidentiality and integrity as it relates to ensuring that users do not deny accessing, editing or deleting data.

4.2. Internet-of-things abstract model architecture

Decomposing the architecture of Internet-of-things applications into distinct layers provides an overview of the idiosyncrasies associated with the systems. This helps to better understand how to tackle data security concerns of such complex systems. Based on existing classification Internet-of-things architecture, for example in [22, 24, 36–38], we have identified the following abstract layers as being the foundation of an Internet-of-things system:

- Application tier [36, 38] This layer provides users access to the Internet-of-things through, for example, a mobile device. The control of the application and intelligent decision-making is performed through this tier. It provides the typical functions of the whole system, including the APIs to consumers, decision-making, task analysis, task schedule and so on. In this tier, a number of services are deployed and interact with each other.
- Network tier [36–38] This layer is responsible for data transmission. The transmission

can be through, for example, a local area network or a mobile cellular data network. Hassanalieragh et al. refer to this layer as data transmission [36].

- Sensor tier [36–38]: This layer is responsible for collecting data from an object of interest through sensors. The data can come from a human-being, environment, or any object of interest. Basically, the function of this layer is to provide environment or situational awareness. It is mainly achieved by sensors that may or may not perform a preliminary data pre-processing, which then transmit the data, through the network layer, to the application and eventually to the support layer. WSN (Wireless Sensor Network) is one of the basic techniques of this sensor tier. This layer can also be referred to as the data acquisition layer [36], perceptual layer [38], and sensation layer [37].
- Data processing tier [36]: This layer consists of the computational devices and storage devices, that provide heterogeneous data processing such as normalization, noise reduction, data storage and other similar functions. This tier is the bridge between Producer and Service. Hassanalieragh et al. refer to this layer as the data cloud processing layer [36].

4.3. Identification of security concerns

4.3.1. General Internet-of-things security considerations

Techniques to compromise data security keep evolving just as fast as the countermeasures to address them do. Thus ensuring data confidentiality, integrity and availability is challenging for Internet-of-things systems. The basic security needs that should be taken into consideration in each of the layers are listed in Tables 3–6. The checklist provided in the tables, which includes particular vulnerable areas for each tier, will help to ensure that security requirements are formulated to address security from different angles using the CIA triad. It also helps with capturing well known issues that can compromise data security, for example, eavesdropping and

unauthorized gathering of data, as well as causing data availability issues through distributed denial-of-service attacks [22, 42].

4.3.2. Domain-specific Internet-of-things security needs

The checklist listed in Tables 3–6 are measures that should be taken into consideration to ensure data security. However, it is important to note that the checklist is not a comprehensive list. This is because depending on the configuration of the Internet-of-things the architecture tier, and the types of security risks can differ across contexts. For example, distributed denial-of-service is a common security issue across networks. But

because not all networks are based on the same communication protocol, it is important to assess each type of network that is used in the Internet-of-things application for any additional relevant threats. For this reason, a context-specific security modeling approach is needed. The application tier involves integrated or individual specific application business, such as smart grid, intelligent transportation, smart security, smart home, wearable devices, and smart city. There are certain security concerns that cannot be solved in other tiers of Internet-of-things, such as privacy protection issue, which does not occur in sensor layer and network layer but can become a concern in certain contexts of the application layer. Moreover, different applications might have a different

Table 3. Security Concerns for the Application tier

Goals Checklist	Description
Confidentiality	Access control for authorized users (i.e., user authentication) Authorized user roles types Least privilege (least functionality) for each user role/type Verification of authorized users
Availability	Third party data and service integration Access to the system on demand by authorized user Access to data on demand by authorized users Data input validation
Integrity	Verification of data source Audit trail of user access into the system Audit trail of user access to data Audit trail of changes to data Principle of separation of duties Attribution of user access to application

Table 4. Security Concerns for the Network tier

Goals Checklist	Description
Confidentiality/ Integrity	Access control for authorized users (i.e., user authentication) Authorized user roles/types Least privilege (least functionality) for each user role/type Verification of authorized users
Availability	Third party data and service integration Prevent distributed denial of service attack Maintain connectivity Detection and prevention of common network attacks (e.g., denial of service)
Integrity	Verification of data source Audit trail of user access into the system Audit trail of user access to data Audit trail of changes to data Principle of separation of duties Attribution of user access to application

Table 5. Security Concerns for the Sensor tier

Goals Checklist	Description
Confidentiality	Data anonymization Encryption algorithm and protocol when data is in transit from sensors Access control (user and device authentication) Encryption key management
Availability	Node and device authentication RFID protocol security Access to data on demand by authorized users and devices Detection and prevention of common sensor attacks (e.g., denial of service) Audit of data collected Audit of data sources

Table 6. Security Concerns for the Data tier

Goals Checklist	Description
Confidentiality	Encryption method when data is in transit from sensors Access control (user and device authentication) Encryption key management
Availability	Access to data on demand by authorized users and devices Malware and virus detection Data recovery mechanism (in case of disaster or failure) Mitigation strategy for disaster and data recovery
Integrity	Verification of data source Audit trail of user access to data Audit trail of changes to data

priority on security requirements. For example, data privacy would be of great importance for Transportation and Healthcare sector, but, on the other hand, data authenticity may be more important for a Smart city.

In addition, in regulated domains, there are security requirements that need to be implemented for data protection. A good example is medical device software, which needs to comply with specific data protection guidelines. In the United States of America, for example, medical device software needs to implement security measures outlined in the Security Rule that is in the Health Insurance Portability and Accountability Act (HIPAA) [43]. Therefore, it is also essential to consider the domain in which the Internet-of-things application will be used and understand the unique data security demands and regulations.

4.3.3. Assessment of security concerns

Identifying and assessing security threats can be performed by following a threat modeling approach [32]. We propose following the approach

shown in Figure 2, which adopts well known threat modeling techniques in security engineering [5] that are used as a basis for deriving security requirements [4, 32].

The approach shown in Figure 2 considers the context of the system, the likelihood of threats occurring and the impact that they have on the system. This is essential for an effective threat modeling approach. The decomposition of the Internet-of-things application in Section 4.3.2 can be used as Step 1 in Figure 3. Step 2 is a critical step because in order to identify relevant security concerns it is important to understand the complexities and mechanisms used to collect, transmit and store data. Table 7 shows information that should be defined during Step 2 in order to characterize each tier and describe contextual factors that are unique to a particular Internet-of-things application. The aim is to help identify the contextual setting of product development. Data flow diagrams can also help model and understand how data flows through each tier, which is useful information for understanding data security concerns in the Internet-of-things application.

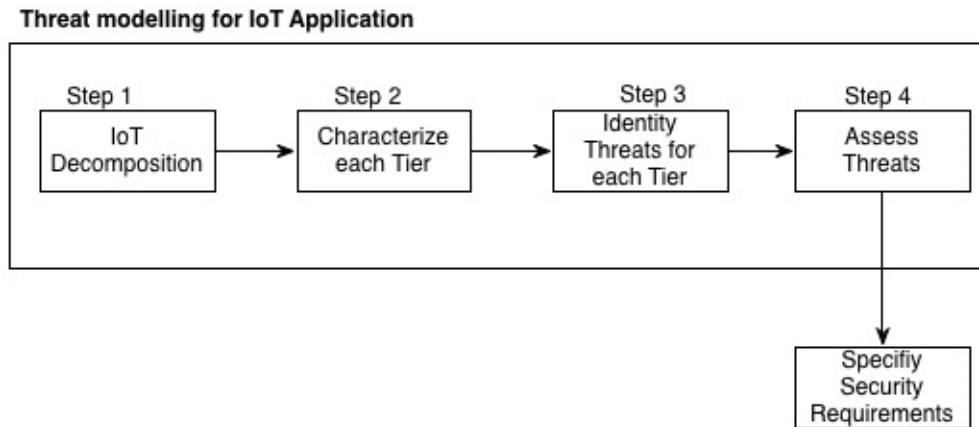


Figure 2. Threat modelling for Internet-of-things application

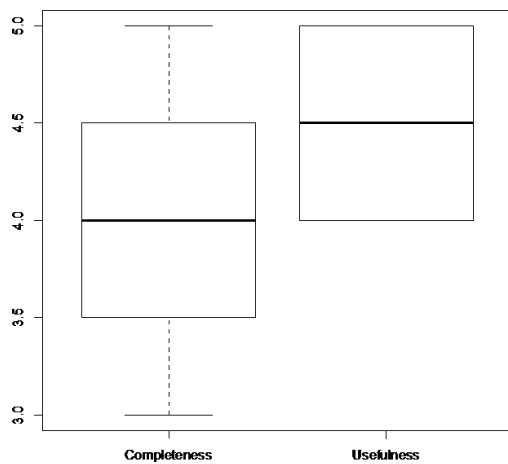


Figure 3. The evaluation of the SIoT Framework at ABC company

The characterization done in Step 2 should then be used as input in Step 3, which involves activities of identifying threats within each tier. Assessment of the relevance to the Internet-of-things application being analyzed will then be done in Step 4. The threats that are found to be relevant can also be assessed on their likelihood of occurrence and severity or extent of negative impact on data security. The assessment can be used for prioritization during the implementation of security requirements.

Table 7. Security Concerns for the Data tier

Goals Checklist	Description
Application Tier	Types/roles of authorized users Number of authorized users Criticality of data that can be accessed (e.g., private and sensitive data)
Sensor tier	Devices used (e.g., hand-held mobile devices and laptops) Number of authorized users to connect to sensors Number of authorized devices to connect to sensors Criticality of data that can collected (e.g., private and sensitive data)
Network tier	Data transmission method from the sensor layer to the application layer (wireless personal networks [44], e.g., Bluetooth and Zigbee [45]) Data transmission method from the application layer to the data processing layer, e.g., mobile cellular network, wireless local area networks [44]
Data processing tier	Data transmission method from the data processing layer to the application layer, e.g., mobile cellular network, Wireless Fidelity (i.e., WiFi). Communication protocols Use of local device storage system Cloud storage model (e.g., the use of either private, public, community or hybrid cloud [46])

5. Evaluation of the SIoT framework

5.1. The company context

To evaluate the SIoT framework, we apply it in a company that we will refer to as ABC to preserve its anonymity. ABC is a spin-off startup from an international enterprise that provides real-time industrial IT integration, automation and manufacturing solutions. ABC has approximately 20 employees, developing a system for estimating glucose (blood sugar) based on the combination of several non-invasive measurement principles. The company adopts several engineering methodologies:

- Process-driven development: the company was approved according to ISO 13485 (quality management system for the development and manufacturing of medical devices). Product development involves a significant amount of documentation for internal use and external communication. The company adopts a tailored version of Agile with long-term iterations. A lot of physical tests are performed on hardware components, e.g., strap test, temperature test and load test. Automated testing and continuous integration are done for software components.
- Quality-driven development: the developed product is classified as a Class IIa Medical Device product, which highlights the criticality of several quality attributes, such as performance, safety and most importantly, security.
- Software platform development: product development involves the implementation of an embedded platform using C++/ RTOS, Java, noSQL and secured REST-API.

ABC was used to evaluate the SIoT framework as a follow-up research activity after the industrial survey (case C11 in the survey in Section 3). Security has been recognized as a vital quality attribute at ABC. The company also expressed the need for a framework for addressing the security concerns of their product.

With the permission of the CTO we formed a focus group consisting of developers from ABC to evaluate the SIoT framework. The focus group

aimed at evaluating the SIoT framework on its usefulness in practice, and to assess if SIoT can help identify any additional security requirements apart from those that they already knew and had documented. During the evaluation, the focus group used the security requirements for the current glucose estimator prototype. Some of the security concerns that the company was keen on addressing are data confidentiality and integrity.

The glucose estimator device will be body mounted in the form of a wearable device that communicates with a mobile app for displaying and monitoring data. The system is similar to the Internet-of-things health monitoring system presented by Hassanali et al. that is also based on WBAN and cloud-based processing [36]. The prototyping development was finished in the winter of 2017 and the product is currently under European regulator evaluation. The development team includes five people with competences in electronic engineering, software engineering and medical expertise. The prototyping process started in Spring 2015. The development approach is research-based with long iteration. All R & D activities occurred in-house.

5.2. Focus group and the evaluation process

A focus group is a popular research method in social science, that involves carefully planned discussions to obtain the perceptions of group members on a defined topic [47]. Typically, there are 3 to 10 participants in a focus group, that are facilitated by a moderator, who guides a structured discussion on a specific topic. The approach has been used in requirements engineering to elicit and to analyze requirements [48]. In our case, we aim at discovering the security concerns of the current prototypes by using the SIoT framework as a proxy object. We followed the focus group meeting guidelines proposed by Edmunds [49], specifically:

- Defining the research problem: the group aims at evaluating the current security concerns for the prototype.

- Selecting participants: we include all stakeholders who are involved in the development of the prototype. In total four engineers participated in the focus group
- Planning and conducting the focus group: we hold a 120-minute discussion with participants. Each session started with an overview of the objectives of the study and with a discussion on how participants should discuss and act during the session. The first topic was to discuss the current security level of the prototype. A researcher, who acted as the moderator, went through the four checklists of security requirements. The second topic was to discuss the usefulness and completeness of our SIoT. Each participant would give evaluation scores for the completeness and usefulness of SIoT at the end of the activity.
- Analysis: the discussion was noted and summarized into points. Each point was mapped to (1) requirements that are already implemented in the current prototype and (2) requirements should be implemented in the next version of the prototype.

SIoT helped the focus group identify security requirements that they had missed when developing the prototype. The results of the focus group were summarized in Table 8.

5.3. Evaluation results and lessons learned

Figure 3 presents the boxplot (mean, min, max and outliers) of the evaluation scores from focus group participants. As mentioned previously, each participant gave a score for the completeness of the framework (if SIoT covers all relevant aspects to them) and the usefulness of the framework (if SIoT helps them in identifying security requirements). The scores range from one to five, with five being the highest degree. Figure 3 shows that participants perceived SIoT positively and very useful. They appreciate the framework in facilitating the discussion on security concerns for the product as well as helping with identifying security requirements for future releases.

Nevertheless, the participants pointed out three main issues when using the framework:

- Requirements of multiple expertise: it is remarked that the framework goes beyond a singular tier of Internet-of-things system, hence, the adoption of the framework needs to involve software developers, hardware engineers, business analysts, etc. to reflect the comprehensive set of requirements.
- Abstraction of some security concerns: for instance, common attacking approaches or

Table 8. Security requirements in ABC: “had” vs “should have” lists

Goals	Requirements Implemented	Requirements to be implemented
Application Tier		
Having: Confidentiality, Availability	Access control for authorized users (i.e., user authentication)	Authorized user roles/types
Should have: Confidentiality	Access to the system on demand by authorized user	Least privilege (least functionality) for each user role/type
	Data input validation	Verification of authorized users
		Third party data and service integration
Sensor Tier		
Having: Confidentiality	Encryption algorithm and protocol when data is in transit from sensors	Node and device authentication
Should have: Confidentiality, Availability		RFID protocol security
		Access to data on demand by authorized users and devices
Network Tier		
Having: Confidentiality	Identity authentication	N/A
Should have: Confidentiality, Availability		
Data processing tiers		
Having: Confidentiality	Access control (user and device authentication)	Verification of data source
Should have: Integrity		

detection of a virus are elements that locate in a high abstract level. These elements are not directly transferred into implementable requirements. Further discussions would be required to identify specific security requirements.

- Domain-specific focus: the evaluation was done on a prototype from the healthcare domain. In such an application domain with a lot of regulations, there are specific demands on adhering to security standards and laws. While the framework is useful as a starting point to help address critical security concerns, domain-specific concerns linked to regulations and standards are not straightforward.

6. Discussions

In this section, we discuss and summarize the SIoT framework (Section 6.1), and also discuss the application of SIoT in iterative development (Section 6.2) and opportunities for improvement (Section 6.3).

6.1. Comprehensive vs. domain specific security consideration

Our SIoT framework looks at security concerns from the product architecture perspective. We emphasize the security concerns of the entire system, rather than just the security of a single piece of software or a single Internet-of-things layer. This incorporates the fact that Internet-of-things application includes multiple tiers of software, middleware and hardware. A multi-perspective view has also been considered in existing models [25, 27]. Our model has been revised and validated by practitioners. Nevertheless, the real-life scenario of Internet-of-things applications could become an ecosystem, in which security is not only considered at a technical level, but also at organizational level and ecosystem level. Therefore, one might consider in future work, a comprehensive framework that captures security concerns at both technical, organizational and ecosystem levels.

The SIoT framework consists of three main components (See Section 4). We propose that the first step in applying the framework should be the “security goals”, which results in the characterization of data security for the system. This is necessary for performing actions within the other two components, i.e., “IoT abstract model” and “IoT security considerations”. These other two components can be performed in parallel, and their output is a list of security needs that are then translated into security requirements. The security requirements can be documented, for example, using natural language [50].

The consensus within the focus group during the evaluation in the industry was that SIoT is useful and helps address security from different key angles of Internet-of-things systems. As the focus group participants pointed out the framework goes beyond a singular tier of Internet-of-things system, hence promoting the notion of having a holistic view of security for the system. Developing and marking secure Internet-of-things systems requires that not only software engineers are security-aware but all stakeholders within the company. There are hardware concerns and market considerations. Therefore, in order to get a comprehensive set of security requirements, we encourage that the adoption of the SIoT framework should involve software and hardware engineers, testers, business analysts, architects, and any other personnel that is involved directly or indirectly in the development of the Internet-of-things system.

Overall SIoT addresses data security from multiple perspectives in order to ensure that essential security requirements are identified. This includes, taking into account the idiosyncrasies of the Internet-of-things application as well as the regulatory requirements of the domain in which it will be used. This provides a more comprehensive view of security concerns for the Internet-of-things application. However, there will be some overlap in terms of security requirements in particular during the activities for the Internet-of-things security considerations. Thus, the security requirements should be aggregated, analysed, and duplicates should be removed.

6.2. Fitting SIoT into iterative development

The proposed SIoT framework can be used in an iteration/ sprint plan to ensure security concerns are addressed and captured by the security requirements [51]. The output from the framework can then be used as a starting point to develop metaphors or user stories which then could be added to the backlog. Schwaber provides a list of questions to guide a retrospective meeting [51]. We propose to add the following to the list in order to bring the topic of security into the meetings: (1) which security concerns have been addressed, (2) which security concerns might have been missed in the backlog, and (3) which security concerns need to be added to the backlog. In addition, an agile approach to continuous changes and integration may introduce security vulnerabilities, which results in the development of insecure software [18]. Hence, we suggest adopting an agile and continuous assessment of security requirements to ensure that they are appropriate and in line with the best practices for addressing the most current and sophisticated security risks that are relevant to the Internet-of-things system. This can be facilitated by adopting threat modeling within the development process, specifically during the requirements and design phases [18]. Furthermore, having a role within the process dedicated to overseeing software security related aspects should also be considered, for example, Ghani et al. suggest adding a “security master” role in Extreme Programming (XP) [20].

Security risks and threats will keep evolving as mitigation strategies and new technologies

emerge. Therefore, it is crucial to keep monitoring and managing mechanisms that are intended to protect data. Therefore, a continuous approach of updating security requirements is paramount for ensuring that appropriate risks are taken into account as the software evolves. This can be done by adopting an iterative continuous process for addressing security requirements as captured in Figure 4. Security goals need to be identified at the beginning of the project, i.e., aligning with business strategy. The first iteration (Sprint 1) will identify the architectural model at the abstract level, following by the identification of general security concerns. Consequent iterations refine the architectural model and re-evaluate the list of security concerns. At some point in time (Sprint N), the domain-specific security concerns are identified and reflected in the architectural model.

6.3. Limitation of the framework

The framework provides a conceptual approach that can be applied in any company context. The security consideration (described in Table 1) presents the need for such a framework. The completeness and usefulness of the framework is evaluated using quantitative forms. However, the evaluation of the framework is preliminary and only bases on a focus group. We are aware of the limitation and plan for future work with thorough validation of the model. Our main future plan is to continue evaluating the framework more cases industry, particularly in various domains. At the moment we mitigate this issue by carefully ensuring that SIoT is based on existing best practices for security engineering,

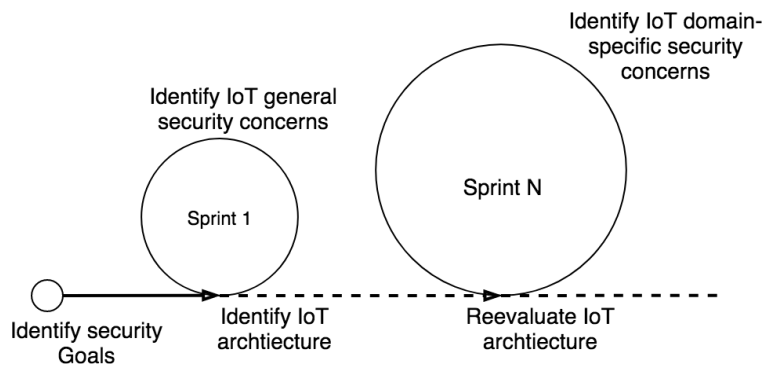


Figure 4. Continuously considering security concerns

e.g., [5, 14, 32] and research on Internet-of-things security, e.g., [16, 22, 24–29, 36–38].

While SIoT was perceived positively with regards to its usefulness, there was an issue regarding its completeness. This was pointed out by practitioners in the focus group meeting. This pertains to addressing security outlined for regulatory environments, demanded by law or specified in technical standards such as those in a certain domain like the medical device industry. This is a critical issue to note. Therefore, users of the framework should also reflect on any other requirements demanded in their own regulated environment. Possible future work is to provide a guideline for mapping the SIoT framework to specific regulatory requirements in a specific domain.

7. Conclusions

In this paper, we proposed a systematic engineering approach for identifying security requirements in Internet-of-things systems. The gaps in requirements and system design were found in Internet-of-thing startups. Considering an abstract architecture of an Internet-of-things application, we are able to come up with a SIoT framework, that offers a systematic way to identify, maintain and to evaluate security aspects in Internet-of-thing applications. The framework has been used in a Norwegian startup with initial positive feedback. However, it is worth mentioning that Internet-of-things security issues are application specific, so the approach needs to be adapted in a specific application domain, which might introduce specific architectural elements. Hence, this SIoT framework may need to be refined in the future.

Our case company suggested that the framework provides a good basis to help address critical security concerns for Internet-of-things applications. However, security is a multi-faced concept; therefore users of the framework should not view the framework as a panacea to all security threats. In addition, security threats will keep evolving as technology evolves. Therefore, there is a need to update SIoT accordingly

over time, as well as to conduct further validation with practitioners and improving the framework based on feedback. Furthermore, we suggest complementing the framework by adopting supporting security activities, e.g., continuous security considerations (e.g., shown in Figure 4), penetration testing and keeping up-to-date with emerging security threats from resources like OWASP foundation¹. Designing secure systems requires understanding the complex interaction between different parts of architecture and the security threats for those parts. The SIoT framework, which takes a layered view of the architecture of Internet-of-things applications, provides a foundation for promoting that thought process.

Acknowledgments

We appreciate Prof. Tor Stalhane from NTNU and Dr. Indira Nurdiani (University of Southern Denmark) for their constructive review and feedback on the SIoT framework.

References

- [1] S. Lucero, “IoT platforms: Enabling the Internet of Things,” IHS Technology, Whitepaper, 2016. [Online]. <https://www.esparkinfo.com/wp-content/uploads/2018/11/enabling-IOT.pdf>
- [2] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, International Series in Software Engineering. Springer, 2000. [Online]. <https://www.springer.com/gp/book/9780792386667>
- [3] A. Olmsted, “Secure software development through non-functional requirements modeling,” in *International Conference on Information Society (i-Society)*, 2016, pp. 22–27.
- [4] S. Myagmar, A.J. Lee, and W. Yurcik, “Threat modeling as a basis for security requirements,” in *Proceedings of the IEEE Symposium on Requirements Engineering for Information Security*, 2005.
- [5] F. Swiderski and W. Snyder, *Threat Modeling*. Microsoft Press, 2004.
- [6] A.N. Duc, R. Jabangwe, P. Paul, and P. Abrahamsson, “Security challenges in IoT development: A software engineering perspective,” in

¹The OWASP foundation can be found at this link: www.owasp.org

- Proceedings of the XP2017 Scientific Workshops, XP '17*. ACM, 2017, pp. 11:1–11:5.
- [7] A.S. Sani, D. Yuan, J. Jin, L. Gao, S. Yu, and Z.Y. Dong, “Cyber security framework for internet of things-based energy internet,” *Future Generation Computer Systems*, Vol. 93, No. 4, 2019, pp. 849–859.
- [8] I. Jacobson, I. Spence, and P.W. Ng, “Is there a single method for the internet of things?” *Queue*, Vol. 60, No. 11, 2017.
- [9] P. Patel and D. Cassou, “Enabling high-level application development for the Internet of Things,” *Journal of Systems and Software*, Vol. 103, 2015, pp. 62–84.
- [10] B. Morin, N. Harrant, and F. Fleurey, “Model-based software engineering to tame the IoT jungle,” *IEEE Software*, Vol. 34, No. 1, 2017, pp. 30–36.
- [11] K. Meridji, K.T. Al-Sarayreh, A. Abran, and S. Trudel, “System security requirements: A framework for early identification, specification and measurement of related software requirements,” *Computer Standards and Interfaces*, Vol. 66, 2019, p. 103346.
- [12] M. Ammar, G. Russello, and B. Crispo, “Internet of things: A survey on the security of IoT frameworks,” *Journal of Information Security and Applications*, Vol. 38, 2018, pp. 8–27. [Online]. <http://www.sciencedirect.com/science/article/pii/S2214212617302934>
- [13] P. Devanbu and S. Stubblebine, “Software engineering for security: A roadmap,” in *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, 2000. [Online]. https://www.researchgate.net/publication/2393383_Software_Engineering_for_Security_a_Roadmap
- [14] N. Mead, “Security quality requirements engineering (SQUARE),” Software Engineering Institute, Tech. Rep., 2011.
- [15] G. Sindre and A.L. Opdahl, “Eliciting security requirements with misuse cases,” *Requirements Engineering*, Vol. 10, No. 1, 2005, pp. 34–44.
- [16] A. van Lamsweerde, “Elaborating security requirements by construction of intentional anti-models,” in *Proceedings. 26th International Conference on Software Engineering*, 2004, pp. 148–157.
- [17] Y. Yu, H. Kaiya, H. Washizaki, Y. Xiong, Z. Hu, and N. Yoshioka, “Enforcing a security pattern in stakeholder goal models,” in *Proceedings of the 4th ACM workshop on Quality of protection*, 2008, pp. 9–14.
- [18] S.H. Adelyar and A. Norta, “Towards a secure agile software development process,” in *10th International Conference on the Quality of Information and Communications Technology (QUATIC)*, 2016, pp. 101–106.
- [19] K. Beznosov, “eXtreme security engineering: On employing XP practices to achieve “good enough security” without defining it,” in *First ACM Workshop on Business Driven Security Engineering (BizSec)*, 2005.
- [20] I. Ghani and N.I.A. Firdaus, “Role-based extreme programming (XP) for secure software development,” in *Special Issue – Agile Symposium*, 2013.
- [21] M.R.R. Ramesh and A. Tadepalligudem, “A survey on security requirement elicitation methods: classification, merits and demerits,” *International Journal of Applied Engineering Research*, 2016.
- [22] Q. Jing, A.V. Vasilakos, J. Wan, J. Lu, and D. Qiu, “Security of the Internet of Things: Perspectives and challenges,” *Wireless Networks*, 2014.
- [23] F. Wortmann and K. Fluchter, “Internet of Things,” *Business and Information Systems Engineering*, Vol. 57, No. 3, 2015, pp. 221–224.
- [24] H.J. La and S.D. Kim, “A service-based approach to designing cyber physical systems,” in *IEEE/ACIS 9th International Conference on Computer and Information Science*, 2010, pp. 895–900.
- [25] S. Babar, A. Stango, N. Prasad, J. Sen, and R. Prasad, “Proposed embedded security framework for internet of things (IoT),” in *2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE)*, 2011, pp. 1–5.
- [26] A. Jacobsson, M. Boldt, and B. Carlsson, “A risk analysis of a smart home automation system,” *Future Generation Computer Systems*, Vol. 56, 2016, pp. 719–733.
- [27] G. Gan, Z. Lu, and J. Jiang, “Internet of things security analysis,” in *International Conference on Internet Technology and Applications*, 2011, pp. 1–4.
- [28] A.W. Atamli and A. Martin, “Threat-based security analysis for the internet of things,” in *International Workshop on Secure Internet of Things*, 2014, pp. 35–43.
- [29] D.H. Kim, J.Y. Cho, S. Kim, and J. Lim, *A Study of Developing Security Requirements for Internet of Things (IoT)*, 2015. [Online].

- <https://www.semanticscholar.org/paper/A-Study-of-Developing-Security-Requirements-for-of-Kim-Cho/>
- [30] R.L. Kissel, Ed., *Glossary of Key Information Security Terms*. National Institute of Standards and Technology, 2013. [Online]. <https://www.nist.gov/publications/glossary-key-information-security-terms-1>
- [31] G. Stoneburner, "Underlying technical models for information technology security," National Institute of Standards and Technology, Tech. Rep. 800-33, 2001.
- [32] A. Shostack, *Threat Modeling: Designing for Security*. Wiley, 2014.
- [33] A. Nguyen Duc, K. Khalid, T. Lønnestad, S. Bajwa Shahid, X. Wang, and P. Abrahamsson, "How do startups develop internet-of-things systems – A multiple exploratory case study," in *IEEE/ACM International Conference on Software and System Processes (ICSSP)*, 2019, pp. 74–83.
- [34] A. Nguyen-Duc, X. Weng, and P. Abrahamsson, "A preliminary study of agility in business and production: Cases of early-stage hardware startups," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '18*. ACM, 2018, pp. 51:1–51:4.
- [35] A. Nguyen-Duc, S.M.A. Shah, and P. Abrahamsson, "Towards an early stage software startups evolution model," in *42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2016, pp. 120–127.
- [36] M. Hassanaliheragh, A. Page, T. Soyata, G. Sharma, M. Aktas, G. Mateos, B. Kantarci, and S. Andreescu, "Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges," in *IEEE International Conference on Services Computing*, 2015, pp. 285–292.
- [37] X. Sun and C. Wang, "The research of security technology in the internet of things," in *Advances in Computer Science, Intelligent System and Environment*, Advances in Intelligent and Soft Computing, D. Jin and S. Lin, Eds. Springer, 2011, pp. 113–119.
- [38] H. Suo, J. Wan, C. Zou, and J. Liu, "Security in the internet of things: A review," in *International Conference on Computer Science and Electronics Engineering*, Vol. 3, 2012, pp. 648–651.
- [39] National Institute of Standards and Technology, "Standards for security categorization of federal information and information systems," U.S. Department of Commerce, Tech. Rep. Federal Information Processing Standard (FIPS) 199, 2004.
- [40] F.Y. Sattarova and T.H. Kim, "IT security review: Privacy, protection, access control, assurance and system security," *International Journal of Multimedia and Ubiquitous Engineering*, Vol. 2, No. 2, 2007, pp. 17–31.
- [41] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley, 2003.
- [42] D. Fischer, B. Markscheffel, S. Frosch, and D. Buettner, "A survey of threats and security measures for data transmission over GSM/ UMTS networks," in *International Conference for Internet Technology and Secured Transactions*, 2012, pp. 477–482.
- [43] M. Scholl, K. Stine, J. Hash, P. Bowen, L. Johnson, C. Smith, and D. Steinberg, "An introductory resource guide for implementing the health insurance portability and accountability act (HIPAA) security rule," National Institute of Standards and Technology, Tech. Rep. 800-66, 2008. [Online]. <https://csrc.nist.gov/publications/detail/sp/800-66/rev-1/final>
- [44] K. Scarfone, D. Dicoi, M. Sexton, K. Scarfone, D. Dicoi, M. Sexton, C. Tibbs, and C.M. Gutierrez, "Guide to securing legacy IEEE 802.11 wireless networks recommendations of the national," NIST, Tech. Rep. 800-48 Rev 1, 2008.
- [45] D. Gislason, *Zigbee Wireless Networking*. Newnes, 2008.
- [46] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Tech. Rep. 800-145, 2011.
- [47] S. Caplan, "Using focus group methodology for ergonomic design," *Ergonomics*, Vol. 33, No. 5, 1990, pp. 527–533.
- [48] K. Garmer, J. Ylven, and M. Karlsson, "User participation in requirements elicitation comparing focus group interviews and usability tests for eliciting usability requirements for medical equipment: A case study," *International Journal of Industrial Ergonomics*, Vol. 33, No. 2, 2004, pp. 85–98. [Online]. <http://www.sciencedirect.com/science/article/pii/S0169814103001318>
- [49] H. Edmunds, *Focus Group Research Handbook*. McGraw-Hill, 2000.
- [50] P. Salini and S. Kanmani, "Survey and analysis on security requirements engineering," *Computers and Electrical Engineering*, Vol. 38, No. 6, 2012, pp. 1785–1797. [Online]. <http://www.sciencedirect.com/science/article/pii/S0045790612001644>
- [51] M. Sliger, *Agile project management with Scrum*. Project Management Institute, 2011.

Appendix A

– **Part 1: General information**

Q1a. Describe your product

Q1b. Describe your company, i.e history, current head count

Q1c. What are the key software development methods, processes, environments and tools?

– **Part 2: Production development practices**

Q2a. How did you build the first prototype?

Q2b. What were the reasons behind the first prototype?

Q2c. How did you make other prototypes?

Q2d. What have you learnt from the prototyping process?

Q2e. When the actual development started?

Q2f. How does the final product different from prototypes?

Q2g. Please name three most important challenges during product development

Q2h. How many significant pivots have you encountered?

– **Part 3: Quality concerns and testing**

Q3a. What are quality attributes important for your products? Q3b. How do you manage your product quality? Q3c. How do you do testing? Q3d. When did you last refactor your codebase? Q3e. How do you consider Security in your final product?

– **Part 4 – final reflection**

Q4. Any final interesting comment ?

Extending UML Use Case Diagrams to Represent Non-Interactive Functional Requirements

Saqib Iqbal*, Issam Al-Azzoni*, Gary Allen**, Hikmat Ullah Khan***

**Department of Software Engineering and Computer Science, Al Ain University, Al Ain, UAE*

***Department of Computer Science, University of Huddersfield, UK*

****Department of Computer Science, COMSATS University Islamabad, Wah Campus, Pakistan*

saqib.iqbal@aaau.ac.ae, issam.alazzoni@aaau.ac.ae, g.allen@hud.ac.uk,
hikmat.ullah@ciitwah.edu.pk

Abstract

Background: The comprehensive representation of functional requirements is a crucial activity in the analysis phase of the software development life cycle. Representation of a complete set of functional requirements helps in tracing business goals effectively throughout the development life cycle. Use case modelling is one of the most widely-used methods to represent and document functional requirements of the system. Practitioners exploit use case modelling to represent interactive functional requirements of the system while overlooking some of the non-interactive functional requirements. The non-interactive functional requirements are the ones which are performed by the system without an initiation by the user, for instance, notifying something to the user or creating an internal backup.

Aim: This paper addresses the representation of non-interactive requirements along with interactive ones (use cases) in one model. This paper calls such requirements ‘operation cases’ and proposes a new set of graphical and textual notations to represent them.

Method: The proposed notations have been applied on a case study and have also been empirically evaluated to demonstrate the effectiveness of the new notations in capturing non-interactive functional requirements.

Results and Conclusion: The results of the evaluation indicate that the representation of operation cases helps in documenting a complete set of functional requirements, which ultimately results in a comprehensive translation of requirements into design.

Keywords: Use Case modeling UML Requirements Engineering Functional Requirements

1. Introduction

Software Engineering is concerned with developing software as per the stakeholders’ expectations (requirements). Gathering, eliciting and documenting these requirements is the most crucial phase of the software engineering process. During this phase, detailed software requirements specification (SRS) documents are developed to specify the system requirements. One of the most widely used requirements specification tools is use case modelling, which represents the system

users (actors) and their interactive requirements (use cases). Use case modelling was proposed by Jacobson [1] and was later adopted by the Unified Modelling Language (UML) [2]. The purpose of use case modelling is to represent requirements in such a way that all stakeholders (from a technical or non-technical background) could easily understand and review them [3]. Use case modelling has been considered as an effective tool by the academic research community [4, 5] and industry [6, 7] to specify and model functional requirements. There are, however, some functional

requirements which often are not represented as use cases as they are not initiated by an actor. Examples of such requirements in a simple ATM banking system would be ‘Check Cash Dispenser’, ‘Notify Bank About an Empty Cash Dispenser’ or ‘Display Promotional or Informational Messages’. These functions are triggered either in response to an interactive requirement (a use case), an event, or at a specified time according to the internal system clock. We may call these requirements non-interactive requirements as they are triggered by the system without users’ initiation. System specification is not complete without the representation of these requirements along with use cases.

A use case, as the name suggests, has always been considered as a case of usage of the system, a usage scenario in other words. The representation of a requirement that does not represent a usage scenario (a non-interactive requirement) as a use case would only lead to confusions. There is a need for a separate representation for such requirements which is graphically and textually different from a use case. Both types of requirements, however, are needed to be represented in one model because they are both functional requirements and their representation in one model would provide a single point of reference for a complete list of functional requirements.

To cater for this need, we propose a new construct called ‘*Operation Case*’. The Operation Case is a system function which is not initiated by an actor, rather is triggered either by a use case or is initiated in response to an event or system clock. Operation cases are modelled along with the use cases in the same subject (system or module) to represent a full set of functional requirements of the subject. The construct has been added to the use case models in addition to other constructs by introducing a new profile to UML. The proposed constructs have been applied on a case study to show the comprehensiveness of the approach in representing functional requirements. In addition, an empirical evaluation has been conducted. The evaluation focuses on addressing the hypothesis that the proposed constructs and method represent a comprehensive list of functional requirements which eventually

leads to a complete and consistent design. The empirical evaluation demonstrates that use case modelling without operation cases can lead to overlooking of key functional requirements.

The rest of the paper is organized as follows: Section 2 provides a review of the related literature. Section 3 outlines the problem and motivations behind the research. Section 4 describes operation cases in detail. Section 5 provides details of the implementation of the proposed concepts. Section 6 illustrates the application of the new notation via a worked case study. Section 7 reports on the results of a controlled experiment. To address possible internal validity threats to the conclusion of the controlled experiment, a second experiment was conducted and it is presented in Section 8. Section 9 concludes the paper with a discussion on the future work.

2. Related work

In [8], Glinz identifies and demonstrates several deficiencies of UML, with emphasis on use case models and system decomposition. Our work attempts to address two of the deficiencies mentioned. The first deficiency is the omission of active objects in UML use case diagrams. Inclusion of active objects in use case diagrams is needed to specify interaction requirements where the system itself initiates an interaction between the system and an external actor. The use of observers in our new notation fills this gap. The second deficiency is that UML use case models cannot express state-dependent system behaviour adequately. We address this issue by introducing operation cases, which can be specified to capture the system state.

Several papers have also presented problems and limitations of use cases. The paper by Génova *et al.* [9] identifies sources of ambiguity that exist in use case models. The paper by Metz *et al.* [10] looks at the problem of use case interleaving present in UML 1.3. In [11], the authors highlight major problems associated with the semantics of extension points and rejoin points, which are used as branching and return locations for a use case’s alternative interaction

courses. The paper by Simons [12] traces the unstable semantics of use cases from Jacobson [13] to UML 1.3.

The paper by Tiwari and Gupta [14] presents a systematic literature review that examines the evolution of use cases, their applications, quality assessments, open issues and the future directions. In addition, the paper identifies a total of twenty existing templates that are used to specify use cases. In [15], the authors investigate via empirical studies the comprehension and learnability aspects of these templates.

The paper by Misbhauddin and Alshayeb [16] proposes an extension to the UML use case metamodel. The extended metamodel captures both the structural and behavioural views of use cases. The aim is to exploit the extended metamodel for model composition, model evaluation, and model interchange.

In [17], the authors propose an extension to the UML metamodel for presenting a refinement relationship between two use cases. The authors discuss the differences between include and refine relationships. The refinement of a use case results in more detailed use cases. Refinement can be defined by decomposing a use case according to the parts that compose the object of that use case, or according to the activities that compose the use case being refined. In our new notation, we do not attempt to represent use case refinement, but rather we introduce the concept of operation case to model non-interactive requirements. In [17], both the refining and refined use cases represent external functionality of the system and thus they agree with the UML definition of use cases. In our work, an operation case is not a type of use case. Other work that builds on use case refinement is [18]. There the authors present an approach to decompose a use case model into models at several levels of abstraction. The authors extend the UML use case metamodel with a refine relationship between a use case and a Use-CaseModel. For each abstraction level, several use case diagrams are used to capture the use cases at that abstraction level.

Several authors have attempted to formalize use case notations. In [19], the control-flow semantics of use cases is described in terms of

control-flow graphs. The technical report by Hurlbut [20] presents a survey of approaches for describing and formalizing use cases. The paper by Metz *et al.* [21] provides definitions for different types of alternative interaction courses in the context of goal-driven requirements engineering. Stevens [22] explores how UML use case notations can be formalized. Savic *et al.* in [23] propose the idea of use case specification at different levels of abstraction: interaction, behaviour, and user interface levels. Each abstraction level extends the previous level. The interactions in a use case are specified using the SilabReq language, which is a textual domain specific language.

The paper of Al-alshuhai and Siewe [24] proposes an extension to the UML use case diagram with new notations to model context-aware applications. The proposed extension, called a use context diagram, allows the modelling of context-aware requirements in addition to the functional requirements of a software application. The new notations include new metamodel elements such as Context Sources and Use Contexts as well as a new utilise relationship between Use Contexts. This extension is useful to cater for the modelling and analysis of the requirements of context-aware applications. We note that our new notation can also be used to model context-aware requirements: an Operation Case can be used as a Use Context to model sequences of actions a system performs to acquire, aggregate, or infer context information. A Context Source can be represented as an Observer that measures context information, and a trigger relationship replaces the utilise relationship.

A systematic literature review on producing high quality use case models is presented by El-Attar and Miller [25]. In their work, twenty six anti-patterns are suggested. A use case anti-pattern explains a repeated pattern in use case models that may initially appear beneficial but ultimately may cause deficiencies [25]. Modellers can exploit these anti-patterns to improve the quality of their use case models.

Identifying use cases can be very useful for the subsequent phases in software development. For instance, Yue *et al.* have created a tool to automatically generate a UML analysis model

comprising class, sequence, and activity diagrams from a use case model [26]. The tool also supports the automatic generation of traceability links between model elements of the use case model and the generated analysis model. Wang *et al.* have proposed an approach for automatically generating executable test cases by exploiting the behavioural information described in use case specifications [27]. The use cases are assumed to be specified in a restricted form of use case specification called Restricted Use Case Modelling (RUCM) [28]. Kesserwan *et al.* present an approach for generating test artefacts from scenario models through model transformation [29]. In the proposed approach, the scenarios are deduced from use case specifications written in the Cockburn use case notation [30]. It is of interest to apply such work on operation cases as well.

In the book by Smialek and Nowakowski [31], the authors present a language specific for requirements modelling, called the Requirements Specification Language (RSL). RSL forms the basis for the framework of model transformation and code generation presented in the book. Functional requirements in RSL are defined mostly through use case models. Use cases in RSL are derived from UML, but several new and changed features exist. These changes are due to the ambiguous semantics of the use case models, as defined in the UML specification [31]. RSL is designed to be a comprehensive language to model use case scenarios while linking those scenarios to their respective domain model elements. We note that the authors define use cases in relation to outside actors. In their definition, a use case starts with the interaction of an outside actor with the system. Hence, RSL seems to capture interactive functional requirements only. The ability of RSL to model non-interactive requirements requires further investigation.

Use cases can also be useful for effort estimation in use case driven projects. Qi and Boehm [32] have proposed an effort estimation model based on a use case model that can be used to estimate project effort during the early iterations in system development. In their work, the size metrics are defined based on the artefacts of a use case model. For example, the Early

Use Case Points (EUCP) metric is a size metric that weights each use case with the number of scenarios identified from the use case description. We believe that operation cases can be dealt with in a manner similar to use cases and they can be useful in effort estimation as well. Use Case Points (UCP) is a software effort estimation technique based on the use case model. A review of effort estimation frameworks and tools based on UCP is provided in [33].

3. Problems and motivation

There have been a number of efforts to extend use case models for representing non-functional requirements [8, 34] along with use cases but there is no evidence in the literature of addressing representation of non-interactive requirements. These requirements are the functional requirements, which are not initiated by an actor rather are triggered by a use case, event or the system clock. For instance, ‘turn on power saving mode’ in a mobile phone operating system is triggered in response to the battery level dropping to a certain level. Similarly, messages to the user, such as ‘battery fully charged’, ‘new SMS received’, or ‘application update available’ are also triggered without the user’s involvement. We may call these requirements non-interactive requirement as they are performed by the system without interaction with the user.

To illustrate these non-interactive requirements in more detail, let us consider Library Management System with the use case diagram in Figure 1. The functional requirements of the system would be:

- R1:** The librarian shall be able to add items such as books, journals and magazines to the system.
- R2:** The librarian shall be able to issue a library item to a user.
- R3:** The librarian shall be able to return an issued item.
- R4:** The librarian shall be able to send a request for a new library item to a vendor.
- R5:** The user shall be able to search for a library item.

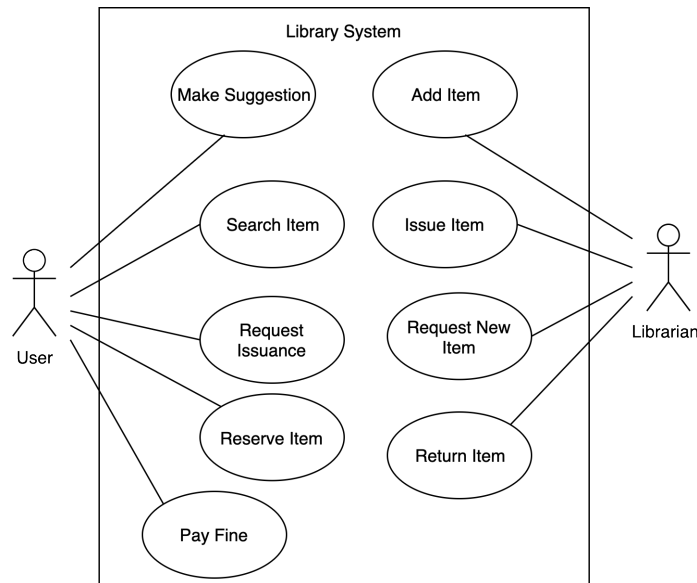


Figure 1. The use case diagram of Library Management System

- R6:** The user shall be able to request issuance of an item that is available.
- R7:** The user shall be able to reserve an item if the item is already issued to someone else.
- R8:** The user shall be able to pay fines where these have been incurred.
- R9:** The user shall be able to make suggestions for new library items.
- R10:** The system shall notify the user when a reserved book becomes available.
- R11:** The system shall calculate fine for late returns, with fines accruing each day after 15 days of issuance of an item.
- R12:** The system shall notify the user of any fines every 3 days.
- R13:** The system shall notify the suggestions provided by the users to the librarian.
- R14:** The system shall make backups at specified times.

The use case diagram of these functional requirements, shown in Figure 1, captures the interactive requirements, but is unable to capture the system requirements, R10 to R14. These requirements are as important for the complete functionality of the system as any other requirement, but they cannot be captured/represented by a use case model as they are not initiated by an actor.

The design artefacts extracted from the use case model would not include these requirements,

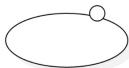

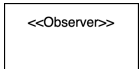
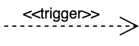
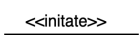
which would eventually lead to incomplete implementation. Representation of these requirements at use case modelling level would make the requirement specification complete and compliant with the system goals. More detailed discussion of this example is provided in Section 7.

4. Operation cases

We propose a new construct for the representation of non-interactive requirements, called ‘operation cases’. An operation case is an internal function which is initiated within the system either by a system timer, an event observer, or a use case. An operation case cannot initiate other operation cases, but can interact with the user, for instance, in case of a dialogue input or a message display to the user. The rationale behind the name of the operation case is that an operation case will represent a complete scenario of an internal operation. It has a separate notational representation to distinguish it from a use case. A complete use case model would include both use cases and operation cases representing all functional requirements identified during the analysis phase, which later will be translated into design mechanisms and design constructs.

A complete list of new notations and related associations is given in Table 1.

Table 1. Notations and Descriptions of New Constructs

Construct	Notation	Description
Operation Case		An <i>Operation Case</i> specifies a set of actions performed by its subjects, which may or may not yield an observable result that is of value for one or more <i>Actors</i> or other stakeholders of each subject. The actions in an <i>Operation Case</i> can only be triggered by an action in a <i>Use Case</i> or initiated by an <i>Observer</i> or a <i>Timer</i> .
Timer		A <i>Timer</i> represents an internal clock of the system or a specific interval of time represented in the implementing software.
Observer		An <i>Observer</i> represent a system component that initiates <i>Operation Cases</i> in response to an internal or external event.
Trigger		<i>Trigger</i> relationship defines that a <i>Use Case</i> triggers an <i>Operation Case</i> .
Initiate		<i>Initiate</i> relationship defines that an <i>Observer</i> or a <i>Timer</i> initiates an <i>Operation Case</i> .

5. Extension to use case modelling

The UML [1] is a widely used modelling language for representing and designing structural and behavioural properties of a system. It provides graphical models and notations that help in modelling internal and external behaviour of a system and representing the structural organization of system modules. Although UML is the most popular visual modelling language in software design, it only supports one paradigm of software design, which is object-oriented design. To counter this problem, the Object Management Group (OMG), the proprietary owner of UML, has proposed UML 2.0, which offers flexibility of extending UML diagrams and design notations. The extension is achieved through the introduction of profiles. A UML profile is an element of the UML; it is defined inside the UML metamodel [2]. Profiles are used to extend classes of the UML metamodel with additional stereotypes, tagged values, and constraints. The stereotypes are used to distinguish similar design notations representing different concepts; the tagged values are new attributes attached to a design construct; whereas constraints are used to introduce invariants and semantic-related limitations on a design diagram or a notation.

5.1. Operation cases profile definition

Since the operation cases introduce a new notational concept in the use case model, we intro-

duce a new profile, named *OperationCasesProfile*, to extend UML metaclasses. The new profile defines several stereotypes which extend standard UML metaclasses. Figure 2 shows the new operation cases profile. The new stereotypes are: *OperationCase*, *Trigger*, *Initiate*, *Observer*, and *Timer*. Their corresponding icons are shown in Table 1.

An *OperationCase* extends the UML metaclass *UseCase*. An *OperationCase* specifies some behaviour that a *subject* can perform. An *OperationCase* defines an offered behaviour of the *subject* with possible reference to its internal structure. Similar to a *UseCase*, an *OperationCase* may apply to any number of *subjects*.

An *OperationCase* may include or extend any number of other *OperationCases*, but may not include or extend any other *BehaviouredClassifiers* (i.e. *UseCases* or *Actors*). In addition, an *OperationCase* cannot be included or extended by a *UseCase*. The definitions of the *Include* and *Extend* relationships between *OperationCases* is the same as those between *UseCases*.

A new relationship added by the profile is the *Trigger* relationship. It is a relationship from a *UseCase* to an *OperationCase*. It specifies that a *UseCase* triggers an *OperationCase*. *Trigger* extends both *Include* and *Extend* metaclasses, such that the source is the triggering *UseCase* and the target is the triggered *OperationCase*. This indicates that the behaviour of the *OperationCase* is triggered while the behaviour of the *UseCase* is being executed. In UML profiles, if a stereotype

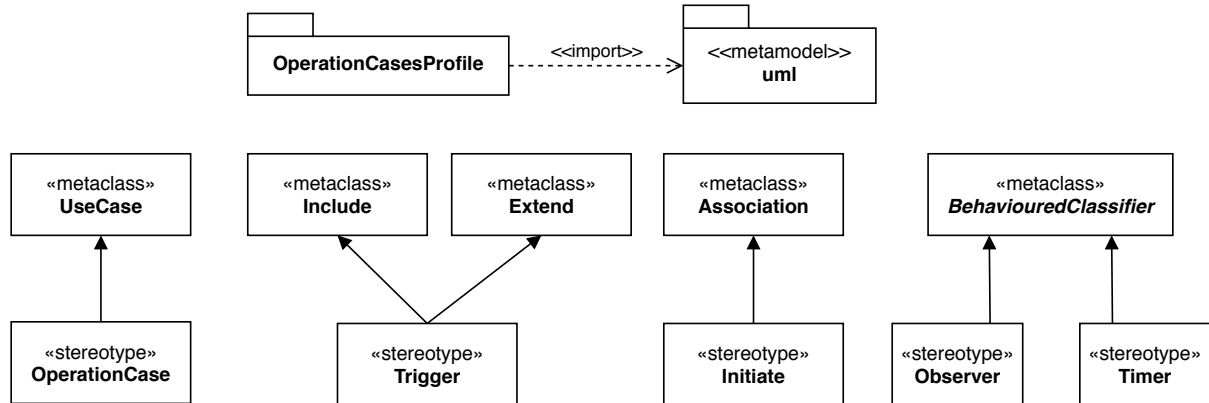


Figure 2. The operation cases profile

extends several metaclasses, it can only be applied to exactly one instance of one of those metaclasses at any point of time. The rationale for extending both *Include* and *Extend* metaclasses is that the behaviour of a triggered operation case can be inserted into the behaviour of the including operation case (in the case of *Include*), or can be added, possibly conditionally, to the behaviour of the extended operation case (in the case of *Extend*).

When an *OperationCase* applies to a *subject*, it specifies a set of behaviours performed by the *subject*. These behaviours can be triggered by an action in a *UseCase* or initiated by an *Observer* or a *Timer*. An *Observer* extends the UML metaclass *BehaviouredClassifier*. Part of a subject, an *Observer* observes events and when an observed event occurs it causes (initiates) the execution of the behaviour of an associated *OperationCase*. A *Timer* also extends *BehaviouredClassifier*. Part of a subject, a *Timer* initiates an operation case when its time interval expires. *Observers* are useful to model behaviours that are initiated by internal or external events. *Timers* are useful to model behaviours that are initiated at specified times according to an internal system clock. The *OperationCasesProfile* defines an *Initiate* stereotype which extends the UML metaclass *Association*. An *Initiate* association is between a *Timer* or an *Observer* on one end of the association and an *OperationCase* on the other end.

An *OperationCase* cannot be associated with *Actors*. Rather, it can only be associated with *Timers* or *Observers*. An *Actor* interacts with a subject through its associated *UseCases* which can indirectly trigger *OperationCases*.

We also add the following constraints to the operation cases profile:

- *OperationCases* can only be involved in binary associations.

```
context OperationCase
inv: Association.allInstances() ->
  forAll(a | a.memberEnd.type ->
    includes(self) implies
      a.memberEnd->size() = 2)
```

- An *OperationCase* cannot include *OperationCases* that directly or indirectly include it.
- ```
context OperationCase
inv: not allIncludedOperationCases()
 -> includes(self)
```

Here, the operation *allIncludedOperationCases()* returns the transitive closure of all *OperationCases* included by this *OperationCase*.

- An *OperationCase* must have a name.
- ```
context OperationCase
inv: name -> notEmpty ()
```
- An *Observer* must have a name. The same is true for a *Timer*.

```
context Observer
inv: name -> notEmpty ()
```

- An *Observer* can only have Associations to *OperationCases*. Furthermore, these Associations must be binary. The same is true for *Timers*

```
context Observer
inv: Association.allInstances() ->
  forAll(a | a.memberEnd ->
    collect(type) -> includes(self)
    implies
      (
        a.memberEnd -> size() = 2 and
```

```

let
  observerEnd : Property =
  a.memberEnd -> any(type = self)
in
  observerEnd.opposite.class
  oclIsKindOf(OperationCase)
)
)

```

5.2. Operation case template

Jacobson [1] introduced a use case template to represent and document the description of a use case. Due to the complexity and unneeded formalism within the template, several variations have been introduced [35–39]. Operation cases are represented in a similar textual template, as shown in Table 2. The template contains a description of constituent items of an operation case, its associations, and related details. The template also mentions the requirements which are represented by the operation case. This helps to improve documentation and traceability.

Table 2. Operation Case Template

Operation Case ID:	
Operation Case Name:	
Requirement ID:	
Created By:	Last Updated By:
Date Created:	Date Last Updated:
Description:	
Pre-conditions:	
Post-conditions:	
Priority:	
Frequency of Use:	
Normal Course of Events:	
Alternative Courses:	
Exceptions:	
Includes:	
Triggered/Initiated By:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

6. Application of new notations

We have selected a subset of functional requirements of a simple mobile phone system for the sake of simplicity. The selected subset of requirements is summarised according to their classification below:

- **Interactive Functional Requirements:**
 - Make a phone call,
 - Receive a phone call,
 - Send a message,
 - Add a contact,
 - Set an alarm.
- **Non-Interactive Functional Requirements:**
 - Transmit data to the service provider,
 - Manage Contact Book (This requirement is concerned with system adding new contacts and placing them in alphabetical order),
 - Receive Push Notifications,
 - Turn on Power Saving Mode in the case that the battery is lower than a threshold,
 - Notify user of updates,
 - Make the phone ring on receipt of an incoming call,
 - Sound an alarm at the required time.

Figure 3 shows the traditional representation of these requirements in a use case diagram. The non-interactive requirements are missing from this model as they do not represent any usage scenario. This model is supposed to be translated into design artefacts and models, but if the model is taken as a complete set of functional requirements, a number of critical requirements (non-interactive requirements) may be overlooked. Figure 4, on the other hand, shows a representation of a complete set of requirements. The requirements, such as ‘Turn on power saving mode’ or ‘Receive push notification’, are represented along with other interactive functional requirements in the same sub-system boundary.

As can be seen, the basic use cases remain the same, showing how the user will interact with the system. However, we can also see that:

- The “Receive Call”, “Make Call”, and “Send Message” use cases each trigger a “Transmit Data” operation case. In traditional use case modelling this could be modelled as a step in the primary path of each of the three separate use cases, but would not be shown on the diagram. By triggering an operation case the shared nature of this functionality becomes explicit, thus both simplifying the descriptions of the individual use cases and capturing

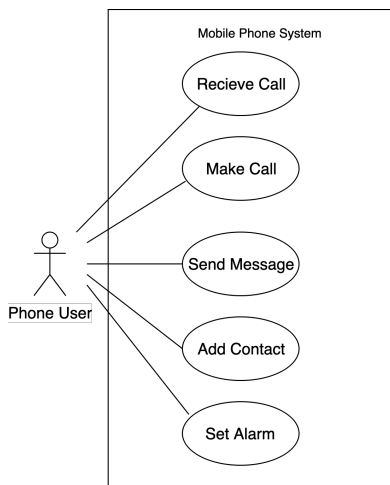


Figure 3. Use case diagram of a mobile phone system

ing the relationships between these functional concerns.

- The “Add Contact” use case triggers the “Manage ContactBook” operation case. It could be argued that the latter is simply a step in the primary path of the former, however we would argue that our model is clearer and it allows for reuse of the “Update Contact Book” operation case. Additionally, the implementation of the operation case would need to deal with issues such as poor network connectivity and failure to connect to the server. This functionality would sit more sensibly in the “Update Contact Book” operation case than in the “Add Contact” use case.
- Three observers have been implemented, two of which monitor incoming connections (incoming calls and incoming push notifications), and one of which monitors the battery and turns on power saving mode when required. This latter example is a classic case of functionality that is difficult to represent clearly using a traditional use case model. There is no actor to drive the functionality, as it is not an interactive use case, instead being event driven. The use of the observer makes this internal functionality explicit.
- One timer driven event has also been implemented. This is the “Sound Alarm” operation case, which is initiated by a timer. Again,

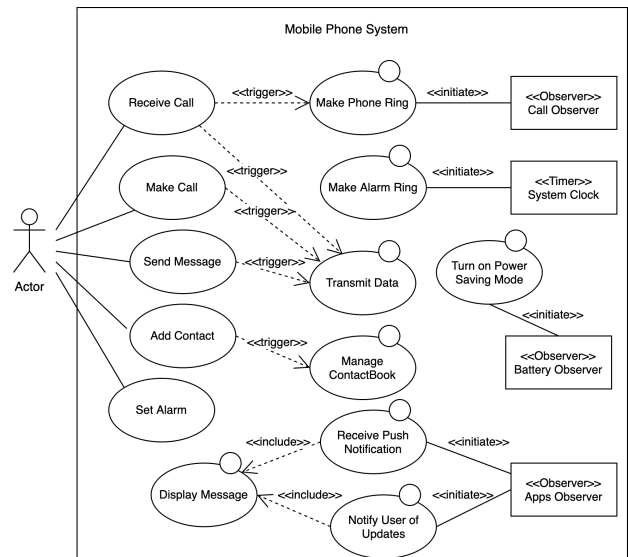


Figure 4. Revised use case diagram of a mobile phone system with operation cases

standard use case models do not allow for the representation of such functionality.

Note that, while a use case can trigger an operation case, the reverse is not true. Take as an example the “Make Phone Ring” operation case. It may, at first sight, appear that this could in turn trigger the “Receive Call” use case. However, making the phone ring does not necessarily cause the user to answer it. The user may be away from the phone, the phone may be on silent, or the user may simply choose to ignore the incoming call. It is not, therefore, possible to assume that the operation case will cause the user to react. Similarly, the “Display Message” operation case may cause the user to read that message, but this is not certain, so the operation case cannot trigger a “Read Message” use case, nor can the operation case deliver data to the user via a directed association. In essence, an operation case may be triggered by use cases, but not vice-versa.

Having modelled the use cases and operation cases in a diagrammatic form, the next step is to write up detailed use case and operation case descriptions. Traditional use case modelling always includes this step, and several standards have been suggested for the layout of use case descriptions. These standards tend to be very similar, and the one chosen for use here is one of the selection that can be found at [40]. One exam-

Table 3. Transmit Data Operation Case Description

Operation Case ID:	MP_OC1		
Operation Case Name:	Transmit Data		
Requirement ID:	SR15, SR20, SR35		
Created By:	GA	Last Updated By:	GA
Date Created:	25 May, 2018	Date Last Updated:	29 May, 2018
Description:	This is a background process running on the phone which receives data from apps such as the dialler app and the messaging app and uploads those data to the mobile network.		
Pre-conditions:	The phone must be switched on and connected to the mobile network.		
Post-conditions:	None.		
Priority:	High – this is a core piece of functionality.		
Frequency of Use:	Variable, depending on the usage patterns of the user.		
Normal Course of Events:	<ol style="list-style-type: none"> 1. The operation case receives a request to upload data from another use case or operation case. 2. A connection to the mobile network is opened. 3. The data are uploaded. 4. The connection to the mobile network is closed. 		
Alternative Courses:	<ol style="list-style-type: none"> 2.1 A connection cannot be established. Try again. 3.1 The data does not upload correctly. Try again. 		
Exceptions:	If at any time the connection is lost and cannot be re-established within 1 second, then the operation case will return an error to the calling use case or operation case.		
Includes:	None.		
Triggered/Initiated By:	Triggered by use cases MP1a Make Call; MP2a Receive Call; MP3a Send Message.		
Special Requirements:	None.		
Assumptions:	None.		
Notes and Issues:	None at present.		

ple operation case description, for the Transmit Data operation case, is given in Table 3.

As can be seen, the primary, alternative, and exception paths of several of the use cases can be simplified by the use of operation cases, as their use helps to partition the system behaviour, and to identify and abstract out shared or common functionality. As an example, the Send Message use case in the traditional model contains the following steps in the primary path:

5. The phone handset connects to the mobile network and attempts to send the SMS message.
6. The message is sent successfully.

and the following alternatives or exceptions:

- 5.1. If the mobile network is unreachable then the phone will retry at intervals until successful.
 - 5.2. If the receiver's phone is unreachable (*e.g.* a wrong number or the phone is switched off) then an error is displayed to the phone user.
- All of this behaviour can be abstracted out to the Transmit Data operation case, thus simplifying the use case description.

In order to integrate Operation Cases fully into the requirements model a small number of amendments have been made to the use case description template. The template has also been

adapted for the description of Operation Cases. The changes proposed are:

- A new section, “Triggers:” has been added to the Use Case Description template. This section lists the Operation Cases that can (optionally) be triggered by the use case.
- When describing Operation Cases, the “Use Case ID” and “Use Case Name” have been changed to “Operation Case ID” and “Operation Case Name”.
- When describing Operation Cases, the “Actor” section has been removed.
- When describing Operation Cases, a new section, “Triggered/Initiated By:” has been added to the template. This section is used to list the use cases, operation cases, observers, or timers that can trigger or initiate the operation case.

With these modifications and additions to the template, we have provided a clear mechanism for the description of all use cases and operation cases within the system model.

We can see that the operation case “Transmit Data” is triggered by the “Make Call”, “Receive Call”, and “Send Message” use cases. The clarity of this information aids understanding of the structure of the requirements, and helps software engineers identify shared and core functionality, in a way that traditional use case modelling is unable to support. This should in turn help software architects with the design of the software, and help project managers to prioritise the development of the system components.

7. A controlled-experiment based evaluation

This section reports on a controlled experiment that was conducted to test whether using use case diagrams extended with operation cases results in a comprehensive system design that incorporates both interactive and non-interactive functional requirements. Section 7.1 presents the research question and hypothesis. Section 7.2 presents the research design, and Section 7.3 presents and discusses the results.

7.1. Research question and hypothesis

Our main proposition is that the UML use case diagrams fail to represent non-interactive functional requirements. Therefore, a systems analyst following an object-oriented development methodology that uses these diagrams to design the system, *i.e.*, construct the class diagram, will likely fail to include the necessary methods to implement the system’s non-interactive functional requirements. The end result is a design and an implementation of a system that do not implement all functional requirements and for which late changes are likely to be costly.

We carried out a controlled experiment to investigate the following research question:

Will using extended use case diagrams help systems analysts to not miss incorporating non-interactive functional requirements in system design?

The experiment was structured as follows. We developed a system story and asked the participants to draw the class diagram. We separated the participants into two equal groups: participants in the **UC group** were requested to draw the use case diagram for the system first and use it to draw the class diagram, while participants in the **OC group** were instructed on the use of operation cases and extended use diagrams and subsequently requested to draw the extended use case diagram and use it to draw the class diagram. More details on the participants are provided in Section 7.2.

In relation to our research question, we formulated the following hypothesis:

The number of correctly identified methods and classes related to non-interactive functional requirements will be higher in the OC group compared to the UC group.

The research variables are as follows. The independent variable is the diagram used: the use case diagram in the case of the UC group or the extended use case diagram in the case of the OC group. The dependent variable is the completeness of the class diagram with respect to incorporating the non-interactive functional requirements. This is captured in a score that

ranges from 0 to 9. The scoring system is described in the next section.

7.2. Research design

There were 14 participants in the experiment. The participants were asked to fill a consent form before participating in the experiment. These participants were undergraduate students taking core courses in the program of Software Engineering in the College of Engineering at Al Ain University of Science and Technology. The OC group consisted of 7 students taking the course “Formal Specifications and Design Methods”, and the UC group consisted of 7 students taking the course “Software Measurement and Testing”. These courses were selected because they are advanced courses and the prerequisite course for both courses is “Software Requirements and Specification”. In this prerequisite course the students study capturing and representation of requirements. The students have completed courseworks and projects in which they have gathered and represented requirements using use case modelling.

The experiment was conducted in the form of two voluntary quizzes; one in each course. The quizzes were conducted on separate days, and involved the participation of the first two authors. The students were given an incentive in the form of bonus points, which would be added to their final grade for the course. To encourage the students, the number of bonus points for each student were tied with the student’s score as follows: 3 points to scores of 7 or more, 2 points to scores of 4–6, and 1 point to scores of 3 or less. During the selection of students it was also ensured that the average Cumulative Grade Point Average (CGPA) of both groups was the same (UC group = 2.54/4, OC group = 2.62/4).

The students had been informed of the voluntary quiz and the bonus points in the earlier class. They were simply asked to show up in the same classroom at the regular class time. There were 10 students who attempted the quiz from the course “Formal Specifications and Design Methods”. On the other hand, only 7 students attempted the quiz from the course “Software

Measurement and Testing”. Since the cumulative degree averages of both groups are different, we only scored a subset of the students who attempted the quiz in the “Formal Specifications and Design Methods” class. We ranked the 10 students in terms of their cumulative degree averages, and then we selected a sequence of 7 students such that the group’s cumulative degree average was close to the that of the group of the second course. The attempts by the other students were discarded; these attempts were never evaluated.

The experimental procedure was as follows: the first author gave a half-an-hour tutorial reviewing use case modelling. For both groups, the tutorial included a review of use cases and use case diagrams. An example system story was used in both tutorials. The instructor of the tutorial worked out an exercise developing a use case diagram that modelled the functional requirements presented in the example system story. The instructor presented how to create a class diagram based on the identified use cases. In particular, the instructor reminded the students of the usefulness of use case scenarios in identifying the classes and their methods. The instructor encouraged the students to apply what they had learned in their earlier courses such as the use of sequence diagrams to model the use case scenarios and construct the class diagram. The instructor worked with the students on constructing the class diagram representing the initial design of the example system story.

The contents mentioned earlier were common in both tutorials. However, there were two key differences between the two. In the tutorial instructing the UC group, there was no mention of non-interactive requirements. The students were simply asked to apply what they already knew. On the other hand, operation cases and extended use case diagrams were introduced in the tutorial instructing the OC group. Non-interactive functional requirements were defined and discussed in this tutorial. These were also applied on the example system story. The students were recommended to use the identified operation cases in constructing the class diagram in a similar fashion to what they would do using use cases.

Figure 5 shows the system story. It describes the functional requirements of an online library management system. We selected this system since undergraduate students at this level are typically familiar with the services provided by the University's online library system. Therefore, they are familiar with library concepts such as searching for and reserving library items. The figure includes the instructions handed to the UC group's students; for the OC group, the students were given the same instructions but were asked to first draw the extended use case diagram with operation cases rather than the standard UML use case diagram. The description includes a set of interactive requirements, such as requesting an item for issuance and requesting new items from vendors, in addition to a set of non-interactive requirements, such as the weekly back-up and the fine's calculation and notification requirements.

An expert solution in the form of a class diagram was created by the first author and checked by the second author. The class diagram includes five classes, including the class *Library* which is used as a system class implementing the methods that are necessary to realise some non-interactive requirements. A total of 19 methods were identified, including 6 methods to realise the non-interactive requirements. The form used in the evaluation of each student's work is presented in Figure 6. The six methods realising the non-interactive requirements are shown in bold.

Since the experiment is concerned with non-interactive requirements, we developed a scoring method for these requirements only. A student gets one point for each correctly identified non-interactive method, *i.e.*, one of the six methods realising the non-interactive requirements. If a student places a method in an incorrect class, they are not rewarded the point. The justification for this is that the use case diagram (or its extended diagram variant) should help the systems analyst in building a complete and sound design. The only class that is critical to implement the non-interactive requirements is the system class *Library*. Four out of six non-interactive methods are in this class. Given its relevance, we assigned the weight of three points for identifying the *Library* class. Thus, the maximum score is nine points, including three points for the *Library* class and one point for each correctly identified non-interactive method. The presented scoring method is similar to [41], but it only considers non-interactive requirements.

The first author who presented the tutorials also evaluated the students' class diagrams. These were subsequently checked by the second author. Since the expert solution is not the only correct one, the evaluators were tolerant of class diagram variations as long as the identified classes and methods were in line with the criteria mentioned earlier. For example, a student may use different method and class names and/or place a method in a different, but correct class.

Experiment to Evaluate Operation Cases

System: Online Library Management System

The online library system automates main features of a library. The librarian can add items to the library, which are books, journals and magazines. The librarian can issue and return an item and also can send a request for new items to a vendor. The users who are also members of the library can search an item. If they find the item and the item is available, they can request for the issuance, or else they can reserve it if it is not available. The item is issued for 15 days. If the item is not returned within 15 days, a specified fine amount starts being calculated for each delayed day and an email message is sent to the user after every 3 days to inform them about the late return and the fine. The users can also make suggestions for new books which are displayed to the Librarian automatically. The system also makes a weekly back-up of all the items.

Figure 5. The description of Library Management System used in the evaluation

<u>Evaluation Form</u>	
Student ID:	
Completeness - Identification of Classes	
Library	<input type="checkbox"/>
LibraryItem	<input type="checkbox"/>
User	<input type="checkbox"/>
Librarian	<input type="checkbox"/>
Vendor	<input type="checkbox"/>
Score:	
Completeness – Correct Allocation of Methods to the Classes	
<u>Library</u> notifySuggestion() <input type="checkbox"/> notifyLateReturn() <input type="checkbox"/> notifyFine() <input type="checkbox"/> createBackUp() <input type="checkbox"/> <u>LibraryItem</u> searchItem() <input type="checkbox"/> reserveItem() <input type="checkbox"/> return() <input type="checkbox"/> issue() <input type="checkbox"/> calculateFine() <input type="checkbox"/> notifyAvailability() <input type="checkbox"/> <u>Vendor</u> supplyItem() <input type="checkbox"/>	<u>User</u> reserveItem() <input type="checkbox"/> returnItem() <input type="checkbox"/> makeSuggestion() <input type="checkbox"/> payFine() <input type="checkbox"/> requestIssuance() <input type="checkbox"/> <u>Librarian</u> <ul style="list-style-type: none"> • addItem() <input type="checkbox"/> • orderItem() <input type="checkbox"/> • issueItem() <input type="checkbox"/> • returnItem() <input type="checkbox"/>
Score:	
No. of Methods Identified:	
No. of Non-Interactive Methods Identified:	

Figure 6. The evaluation form used in evaluating a student's work

7.3. Results and discussion

To address our research question presented in Section 7.1, we tested the following hypothesis which is similar to hypotheses in [41–43]:

H_0 : There is no difference between the scores of the UC and OC groups.

We tested the research hypothesis using the Mann–Whitney U test, as in [42, 44]. Mann–Whitney U test is a nonparametric test for the difference in two means [45]. The results of the test were obtained using XLSTAT which is a statistical analysis tool for Microsoft Excel [46]. The findings indicate that the score of the OC group is significantly higher than the score of the UC group (p -value = 0.027), and therefore hypothesis H_0 is rejected. Note that a significance level of $\alpha = 0.05$ is chosen as the level

of significance. On average, participants in the OC group scored significantly higher ($\bar{X} = 4.286$, $SD = 2.498$) than participants in the UC group ($\bar{X} = 1.286$, $SD = 2.215$; $p = 0.027$) (see Figure 7).

Below, we consider the four categories of threats to validity:

1. **Conclusion validity:** A study has conclusion validity if the results are statistically significant using appropriate statistical tests [47, 48]. We used the Mann–Whitney U test to analyse the results. The assumptions of using this test have been checked. In order to increase the reliability of measures [48], all student evaluations performed by the first author were checked by the second author. All solutions were checked against an expert solution that was constructed prior

to the evaluation and checked for correctness and completeness by the authors.

2. **Internal validity:** Internal validity refers to the cause and effect relationship between the independent and dependent variables. One factor affecting this kind of validity is having any prior significant difference between the groups. In the design of our experiment, there was no significant difference between the groups with respect to the cumulative degree average. In addition, we believe that the exercise of identifying operation cases causes the systems analyst to identify non-interactive functional requirements, and thereby not miss incorporating them in system design. This is our rationale for why the independent variable would affect the dependent variable. One could argue that the participants in the OC group received direct training on identifying and modelling non-interactive requirements while the participants in the UC group did not. This could represent a threat to the internal validity of the experiment. To address this potential threat, we conducted a second experiment (see Section 8) that demonstrates that practitioner software engineers who typically create use case diagrams and follow the unified process of constructing use case diagrams first and using them to create the analysis and design level class diagrams are

expected to miss some non-interactive requirements. This is because these non-interactive requirements are not emphasized (in fact, they are neglected) by the standard use case notations.

3. **Construct validity:** Construct validity concerns the use of measures that are relevant to the study. One factor affecting construct validity is how much the experimental setting differed from a real-world setting. The participants were not involved in a real-world system with real clients and users. However, with regards to the limitation of use case diagrams in modelling non-interactive requirements, the experimental setup highly resembles real-world conditions. A systems analyst cannot capture non-interactive requirements using a use case diagram only; and this has a significant impact on the completeness of system design and implementation. Another relevant factor is the use of meaningful measures of the completeness of design models with respect to non-interactive requirements. We used the same measure for completeness in terms of the number of identified methods in the class diagram as in [41]. We believe that this is a relevant measure since missing a method implies a design and an implementation that do not implement all functional requirements.

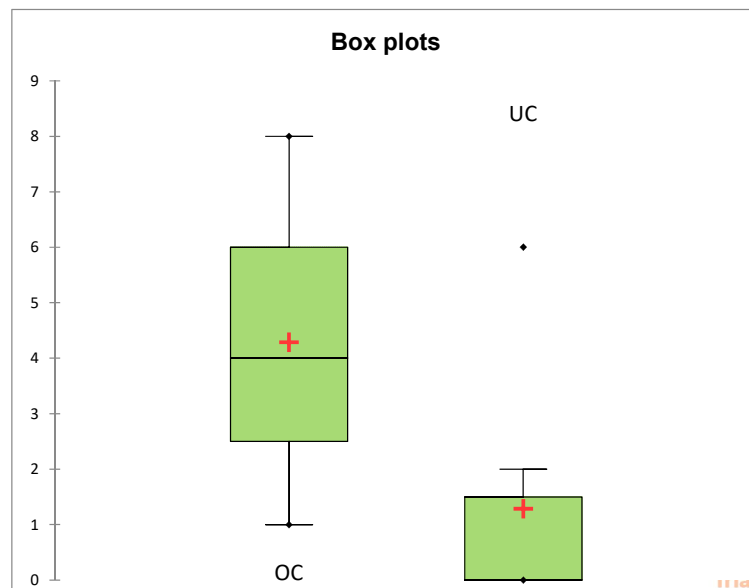


Figure 7. The box plots depicting the scores of the OC and UC groups

4. **External validity:** External validity refers to the generalisability of the results. One factor related to external validity concerns the fact that all participants were students. However, the study in [49], as noted in [42], found that there are minor differences between software engineering students and professional software developers suggesting the use of students instead of professional developers in software engineering experiments is valid under certain conditions. A second factor is related to the number of participants which is relatively small. Given the resources available on hand, this is the best subject population we could find. A third factor is related to the size of the task which is relatively small and does not reflect the typical work by a systems analyst. However, most software engineering experiments use such small tasks due to the inherent difficulty of measuring attributes of large and complex tasks [41–43, 47].

8. An Empirical evaluation using a case study

To address the threats to the internal validity of the controlled experiment presented in Section 7, a second experiment was conducted. The experiment was in the form of a case study analysing the performance of 30 graduate students in the Master of Science in Computer Science program at COMSATS University Islamabad (Wah Campus) on identifying non-interactive requirements of a system. All of the students who participated in the study had a bachelor’s degree and had completed at least one undergraduate-level course on object-oriented analysis and design. The majority of the students were part-time students who were actively working in industry. They were taking the course Advanced Topics in Object-Oriented Software Engineering at the time of the experiment. The experiment was conducted during regular class hours. All students completed the tasks during the regular class hours, although they had been informed that extra time would be given in case it was needed.

This experiment used the same system story as the one in the controlled experiment. The students were handed the description of the Library Management System shown in Figure 5. The task was identical to the task performed by the UC group in the controlled experiment, however no tutorial was provided on use case modelling. Information on each participant was collected with the submission, including the student id number, number of years since graduation, and current employment if any. The class diagrams developed by the students were collected and evaluated following the same scoring procedure as in the controlled experiment.

The same evaluation form was used as in the controlled experiment (shown in Figure 6). With respect to the number of non-interactive methods identified, the average score was 1.1 out of 6. On a 95%-confidence level, the confidence interval for the average score was (0.66, 1.54). The analysis was done using XLSTAT. This shows that many non-interactive requirements were missed and therefore not incorporated in the class diagram. Furthermore, there were only 20 participants who had identified at least one non-interactive method. Of these, 13 participants (*i.e.*, 65%) incorrectly represented the system itself as an actor in the use case diagram. This shows that the standard use case notation as practised might be inadequate in capturing non-interactive requirements.

Since the task done by the participants in this experiment is identical to that of the UC group in the controlled experiment, this experiment is likely to suffer from the same threats to validity as in the controlled experiment. However, the number of participants is much higher in this experiment. In addition, the participants have higher proficiency on use case modelling since they are graduate students with the majority working in industry. This experiment demonstrates that systems analysts who typically create use case diagrams and follow the unified process of constructing use case diagrams first and using them to create the analysis-level class diagrams are expected to miss some non-interactive requirements. This is because these non-interactive requirements are not emphasized (in fact, they are neglected) by the standard use case notations.

9. Conclusion and future work

The representation and documentation of functional requirements at the analysis phase is the most crucial activity in the software development life cycle. Use case modelling solves this problem by providing both textual and graphical methods, which makes it the most widely-used methodology. Use cases are designed into implementable modules in the next phase of the development life cycle. The problem, however, is the exclusion of some of the functional requirements in the use case models. These requirements are often not represented as use cases as they are not initiated by a user, and are thus known as non-interactive requirements. Such requirements are often directly addressed in the design phase without having any backward tracing to the use case models. It is evident from the available literature and existing software development practices that use case models are considered as a complete representation of all functional requirements of the system. Due to this practice, non-interactive requirements are often overlooked and consequently result in implementation of an incomplete system. To represent and document a complete system, non-interactive requirements need to be comprehensively represented and documented along with interactive requirements (use cases) in the analysis phase. This paper addresses this problem and proposes an extension to use case models to accommodate non-interactive requirements. These requirements have been named as Operation Cases and are represented with a new set of graphical notations and textual templates. The paper presents a new profile to extend UML's use case notation with operation cases and their related constructs. The addressing of operation cases at the analysis phase allows analysts and designers to comprehensively document and trace the functional requirements effectively.

We applied operation cases in modelling a (partial) Mobile Phone operating system. For the sake of keeping it concise, only a few relevant functional requirements of the case study are discussed. In the case study, we showed that using use case models alone cannot represent internal non-interactive requirements of the system. Rep-

resentation of operation cases solves this problem and makes use case models a more comprehensive graphical representation of the functional requirements of the system. A controlled experiment was also conducted to investigate the hypothesis that using operation cases results in more comprehensive designs than when using traditional use cases only. The results of the experiment confirmed our hypothesis.

Acknowledgement

This work has not received any funding.

References

- [1] I. Jacobson, "Object-oriented development in an industrial environment," in *Proceedings of the Conference on Object-oriented Programming Systems, Languages and Applications*, 1987, pp. 183–191.
- [2] "Unified modeling language," 2015, [Accessed September 2019]. [Online]. <http://www.omg.org/spec/UML/2.5>
- [3] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.
- [4] B. Anda, K. Hansen, and G. Sand, "An investigation of use case quality in a large safety-critical software development project," *Information and Software Technology*, Vol. 51, No. 12, 2009, pp. 1699–1711.
- [5] S. Tiwari and A. Gupta, "Does increasing formalism in the use case template help?" in *Proceedings of the 7th India Software Engineering Conference*, 2014, pp. 6:1–6:10.
- [6] D. Parachuri, A.S.M. Sajeev, and R. Shukla, "An empirical study of structural defects in industrial use-cases," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 14–23.
- [7] M. Ivarsson and T. Gorschek, "A method for evaluating rigor and industrial relevance of technology evaluations," *Empirical Software Engineering*, Vol. 16, No. 3, 2011, pp. 365–395.
- [8] M. Glinz, "Problems and deficiencies of UML as a requirements specification language," in *Proceedings of the International Workshop on Software Specification and Design*, 2000, pp. 11–22.
- [9] G. Génova, J.L. Morillo, P. Metz, R. Prieto-Díaz, and H. Astudillo, "Open issues in industrial use

- case modeling,” *Journal of Object Technology*, Vol. 4, No. 6, 2005, pp. 7–14.
- [10] P. Metz, J. O’Brien, and W. Weber, “Against use case interleaving,” in *Proceedings of the International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools*, 2001, pp. 472–486.
- [11] P. Metz, J. O’Brien, and W. Weber, “Specifying use case interaction: Clarifying extension points and rejoin points,” *Journal of Object Technology*, Vol. 3, No. 5, 2004, pp. 87–102.
- [12] A.J.H. Simons, “Use cases considered harmful,” in *Proceedings of the International Conference on Technology of Object-Oriented Languages and Systems*, 1999, pp. 194–203.
- [13] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object-oriented software engineering – A use case driven approach*. Addison-Wesley, 1992.
- [14] S. Tiwari and A. Gupta, “A systematic literature review of use case specifications research,” *Information and Software Technology*, Vol. 67, 2015, pp. 128–158.
- [15] S. Tiwari and A. Gupta, “Investigating comprehension and learnability aspects of use cases for software specification problems,” *Information and Software Technology*, Vol. 91, 2017, pp. 22–43.
- [16] M. Misbhauddin and M. Alshayeb, “Extending the UML use case metamodel with behavioral information to facilitate model analysis and interchange,” *Software and Systems Modeling*, Vol. 14, No. 2, 2015, pp. 813–838.
- [17] S. Azevedo, R.J. Machado, A. Bragança, and H. Ribeiro, “The UML «include» relationship and the functional refinement of use cases,” in *Proceedings of the EUROMICRO Conference on Software Engineering and Advanced Applications*, 2010, pp. 156–163.
- [18] E.F. Cruz, R.J. Machado, and M.Y. Santos, “On the decomposition of use cases for the refinement of software requirements,” in *Proceedings of the International Conference on Computational Science and Its Applications*, 2014, pp. 237–240.
- [19] K. van den Berg and A.J.H. Simons, “Control-flow semantics of use cases in UML,” *Information and Software Technology*, Vol. 41, No. 10, 1999, pp. 651–659.
- [20] R.R. Hurlbut, “A survey of approaches for describing and formalizing use cases,” Department of Computer Science, Illinois Institute of Technology, Tech. Rep., 1997.
- [21] P. Metz, J. O’Brien, and W. Weber, “Specifying use case interaction: Types of alternative courses,” *Journal of Object Technology*, Vol. 2, No. 2, 2003, pp. 111–131.
- [22] P. Stevens, “On use cases and their relationships in the unified modelling language,” in *Proceedings of the International Conference on Fundamental Approaches to Software Engineering*, 2001, pp. 140–155.
- [23] D. Savic, A.R. da Silva, S. Vlajic, S. Lazarevic, V. Stanojevic, I. Antovic, and M. Milic, “Use case specification at different levels of abstraction,” in *Proceedings of the International Conference on the Quality of Information and Communications Technology*, 2012, pp. 187–192.
- [24] A. Al-alshuhai and F. Siewe, “An extension of the use case diagram to model context-aware applications,” in *Proceedings of the SAI Intelligent Systems Conference*, 2015, pp. 884–888.
- [25] M. El-Attar and J. Miller, “Constructing high quality use case models: a systematic review of current practices,” *Requirements Engineering*, Vol. 17, No. 3, 2012, pp. 187–201.
- [26] T. Yue, L.C. Briand, and Y. Labiche, “aToucan: an automated framework to derive UML analysis models from use case models,” *ACM Transactions on Software Engineering and Methodology*, Vol. 24, No. 3, 2015, pp. 13:1–13:52.
- [27] C. Wang, F. Pastore, A. Goknil, L.C. Briand, and M.Z.Z. Iqbal, “Automatic generation of system test cases from use case specifications,” in *Proceedings of the International Symposium on Software Testing and Analysis*, 2015, pp. 385–396.
- [28] T. Yue, L.C. Briand, and Y. Labiche, “Facilitating the transition from use case models to analysis models: Approach and experiments,” *ACM Transactions on Software Engineering and Methodology*, Vol. 22, No. 1, 2013, pp. 5:1–5:38.
- [29] N. Kesserwan, R. Dssouli, J. Bentahar, B. Stepien, and P. Labrèche, “From use case maps to executable test procedures: a scenario-based approach,” *Software and Systems Modeling*, 2017.
- [30] S. Adolph, A. Cockburn, and P. Bramble, *Patterns for Effective Use Cases*. Addison-Wesley Longman Publishing Co., 2002.
- [31] M. Smialek and W. Nowakowski, *From Requirements to Java in a Snap – Model-Driven Requirements Engineering in Practice*. Springer, 2015.
- [32] K. Qi and B.W. Boehm, “A light-weight incremental effort estimation model for use case driven projects,” in *Proceedings of the IEEE Software Technology Conference*, 2017.
- [33] M. Saroha and S. Sahu, “Tools and methods for software effort estimation using use case points model – A review,” in *Proceedings of the Inter-*

- national Conference on Computing, Communication and Automation*, 2015, pp. 874–879.
- [34] M. Grossman, J.E. Aronson, and R.V. McCarthy, “Does UML make the grade? insights from the software development community,” *Information and Software Technology*, Vol. 47, No. 6, 2005, pp. 383–397.
- [35] D. Kulak and E. Guiney, *Use Cases: Requirements in Context*. ACM Press, 2000.
- [36] D. Liu, K. Subramaniam, B. Far, and A. Eberlein, “Automating transition from use cases to class model,” in *Proceedings of the Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology*, 2003, pp. 831–834.
- [37] P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd ed. Addison-Wesley, 2003.
- [38] S.S. Somé, “Supporting use case based requirements engineering,” *Information and Software Technology*, Vol. 48, No. 1, 2006, pp. 43–58.
- [39] J. Kettenis, “Getting started with use case modeling: White paper,” Oracle Corporation, Tech. Rep., 2007.
- [40] “40 use case templates and examples,” [Accessed September 2019]. [Online]. <http://templatelab.com/use-case-templates/>
- [41] B. Anda and D.I.K. Sjøberg, “Investigating the role of use cases in the construction of class diagrams,” *Empirical Software Engineering*, Vol. 10, No. 3, 2005, pp. 285–309.
- [42] D. Beimel and E. Kedmi-Shahar, “Improving the identification of functional system requirements when novice analysts create use case diagrams: the benefits of applying conceptual mental models,” *Requirements Engineering*, 2018.
- [43] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, and E. Astesiano, “Assessing the effect of screen mockups on the comprehension of functional requirements,” *ACM Transactions on Software Engineering and Methodology*, Vol. 24, No. 1, 2014.
- [44] M. Dahan, P. Shoval, and A. Sturm, “Comparing the impact of the OO-DFD and the use case methods for modeling functional requirements on comprehension and quality of models: a controlled experiment,” *Requirements Engineering*, Vol. 19, No. 1, 2014, pp. 27–43.
- [45] D.C. Montgomery and G.C. Runger, *Applied Statistics and Probability for Engineers, 6th Edition*. John Wiley and Sons, 2013.
- [46] “XLSTAT,” [Accessed March 2019]. [Online]. <https://www.xlstat.com/en/>
- [47] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach*, 3rd ed. CRC Press, Inc., 2014.
- [48] C. Wohlin, P. Runeson, M. Hst, M.C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer Publishing Company, 2012.
- [49] M. Höst, B. Regnell, and C. Wohlin, “Using students as subjects—a comparative study of students and professionals in lead-time impact assessment,” *Empirical Software Engineering*, Vol. 5, No. 3, 2000, pp. 201–214.

System performance requirements: A standards-based model for early identification, allocation to software functions and size measurement

Khalid T. Al-Sarayreh*, Kenza Meridji**, Alain Abran***, Sylvie Trudel****

* *Department of Software Engineering, Hashemite University*

** *Department of Software Engineering, University of Petra*

*** *Department of Software Engineering and Information Technology, École de technologie supérieure (ETS),
Université du Québec*

**** *Département d'informatique, University of Quebec at Montreal*

KhalidT@hu.edu.jo, kmeridji@uop.edu.jo, alain.abran@etsmtl.ca, trudel.s@uqam.ca

To be or not? To be! – Wally Shakelance

Abstract

Background: In practice, the developers focus is on early identification of the functional requirements (FR) allocated to software, while the system non-functional requirements (NFRs) are left to be specified and detailed much later in the development lifecycle.

Aim: A standards-based model of system performance NFRs for early identification and measurement of FR-related performance of software functions.

Method: 1) Analysis of performance NFR in IEEE and ECSS standards and the modeling of the identified system/software performance functions using Softgoal Interdependency Graphs. 2) Application of the COSMIC-FSM method (e.g., ISO 19761) to measure the functional size of the performance requirements allocated to software functions. 3) Use of the COSMIC-SOA guideline to tailor this framework to service-oriented architecture (SOA) for performance requirements specification and measurement. 4) Illustration of the applicability of the proposed approach for specification and measurement of system performance NFR allocated to the software for an automated teller machine (ATM) in an SOA context.

Result: A standards-based framework for identifying, specifying and measuring NFR system performance of software functions.

Conclusion: Such a standards-based system performance reference framework at the function and service levels can be used early in the lifecycle by software developers to identify, specify and measure performance NFR.

Keywords: Non-functional requirements, (NFR) performance requirements, international standards, Softgoal Interdependency Graphs(SIGs), COSMIC-FSM, COSMIC-SOA

1. Introduction

Over the years, system non-functional requirements (NFRs) from a variety of stakeholders have significantly increased the urgency and effort required to deliver software systems with

very high-quality levels. The large and diverse body of literature on software quality and NFR makes it challenging for practitioners to figure out detailed reference works to use as a baseline for early identification, specification and measurement of any of the large number of NFRs.

Developers must take into consideration both system functional user requirements (FURs) and non-functional requirements (NFRs) early in the system requirements analysis in order to then allocate them at the software/hardware FR level [1–6] (see Figure 1).

The success of a software project depends heavily on its ability to be executed with the required functionalities while under specific constraints. Software functionalities fall under the concept of functional user requirements (FURs) and refer to the set of functions or services required from the system and allocated to the software, while constraints fall under the concept of non-functional requirements (NFRs).

In practice, requirements are usually addressed at the system level [1–4] at the start of the project either as high-level system functional user requirements (system-FURs), or as high-level system-NFRs. The latter must typically be detailed, allocated, and implemented in either hardware or software, or both – see Figure 1.

Software engineers focus on software-FURs for the early development phases, while system-NFRs are typically discussed at later development phases, such as evaluation or testing phases. To distinguish between these types of requirements, the term system-FURs is used to describe the required functions in a system, while system-NFRs is used to describe how the required functions must behave in a system.

In the software requirements engineering phase, system-NFRs are analyzed and detailed, and some may be specified as Software-FURs to allow a software engineer to develop, test, and configure the final software deliverables to system users. It should be noted that a number of such system constraints, while referred to as system-NFRs by some authors, are referred to as quality aspects by other authors.

A number of researchers have investigated issues related to NFR, such as considering them as measurable inputs to effort estimation models [1], which, although based on a different point of view, can be used concurrently with FUR, including their procedures and approaches.

This paper specifically addresses system performance NFR and extends our previous research on three other types of NFR: security [2], portability [3] and maintainability [4]. A key strength of the approach in our previous work is that it is based on the consensus documented in international standards, such as the European Cooperation for Space Standardization (ECSS), the Institute of Electrical and Electronics Engineers (IEEE) and ISO on a number of such NFR, and our proposal for a standards-based reference model for specific types of NFR.

The contribution of this work is a standards-based measurement framework of system performance requirements to be used by de-

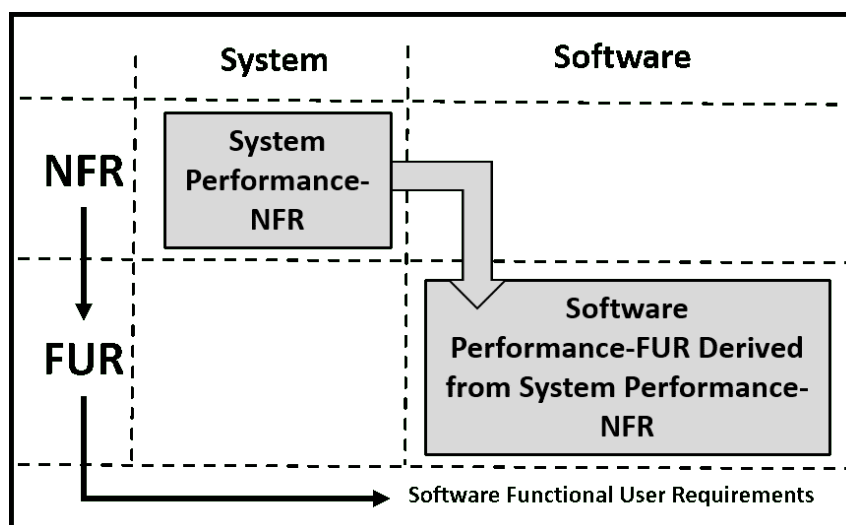


Figure 1. System performance-NFR allocated to software performance-FUR

velopers in the early development stages as a generic model for the identification, specification and measurement of the system performance requirements allocated to software functions.

The proposed framework was developed in four main steps:

1. Identifying, analyzing, and categorizing into an integrated view the system performance requirement functions and services described from different perspectives into ECSS and IEEE standards. Then, modeling the identified system/software performance requirements and clarifying the relations between these requirements using SIGs.
2. Applying the COSMIC-FSM method to identify and measure the data movements derived from the allocated software performance requirements. This leads to handling the system performance requirements allocated to the measured software performance requirements as quantitative requirements.
3. Developing the proposed framework in the context of service-oriented architecture using COSMIC-SOA guidelines to support a distinct business domain.
4. Illustrating the applicability of the proposed approach for the specification and measurement of system performance NFRs allocated to the software for an automated teller machine (ATM) within an SOA context.

The rest of the paper is structured as follows. Section 2 presents related work. Section 3 discusses the system performance requirements identification and related software performance requirements in international standards. Section 4 details the proposed standards-based system performance reference framework at the function and service levels in the context of a service-oriented architecture (SOA). Section 5 presents an illustrative example using the proposed standards-based framework for identifying and specifying ATM banking system performance requirements, allocating them to the software performance functions in an SOA context and measuring them with COSMIC, an ISO-recognized measurement unit. Section 6 presents conclusions and further work.

2. Related work

2.1. Non-functional requirements in the literature and international standards

A number of proposals for identifying and specifying different types of NFR, including different methods, approaches, views and terminologies have been made [1–7].

To help software project teams make the best tradeoff decisions for conflicting NFRs, Zhang and Wang [8] proposed a tradeoff model for conflicting software non-functional requirements (CNFR) using a fuzzy ranking method to express stakeholder assessments of each NFR.

To help developers prioritize such kinds of requirements early in the project cycle, Shah et al. [9] proposed an approach for specifying the NFR conflicts from previous ontological representations of the NFRs.

Daclin et al. [10] analyzed interoperability as a single NFR as a part of a complex NFR domain, linking interoperability and its impacts on the system performance requirements into a collaborative system in a crisis management framework.

Cysneiros et al. [11] highlighted the challenges facing developers of capturing NFR simultaneously with FR at the early phases of software development. They suggested the integration of NFR with FR into conceptual models based on a goal oriented strategy aimed at reducing the cost of software development as well as increasing customer satisfaction.

Various studies have focused on NFR [12–15] within the software product line process. Tawhid and Petriu [16] for example, proposed a UML model transformation framework to determine and reuse the performance requirements for a specific product.

Siegmund et al. [17] proposed a holistic approach, named SPL Conqueror, for the optimization of the specification and measurement of NFR in the SPL domain. They also carried out an analysis of the quality attributes (i.e., NFR) in SPL as well as a verification of product satisfaction of the quality conditions and constraints.

Danylenko and Lowe [18] studied a context-aware recommender system with the objective to defer architectural decisions, thus permitting concentration on the core system functionality design. In early development phases this recommender system helps to ease the difficulties of NFR efficiency.

Kyo and Gil-Haeng [19] proposed a systematic software development process to support successful management and modeling for NFR. This process allows NFR to be systematically managed and efficiently modeled.

Industry, through its participation in international standards organizations, has also contributed by describing and categorizing NFR. For example: performance requirements are one of sixteen NFR types in ECSS [20–25], which have been categorized by IEEE [26] as one of thirteen NFR types, using different terms and views. Although in academia and industry NFR performance requirements are frequently discussed, there is a lack of a performance model that can be used in the early development stages. In the research reported in this paper, we propose a standards-based framework for early identification and measurement of system performance requirements by analyzing all the performance NFR related terms and views dispersed throughout the international standards, such as ECSS and IEEE.

2.2. System performance requirements in the literature

To develop new insights into performance, in this research, we analyzed related works in standards on performance in hardware domains where there is considerable, accumulated expertise. We looked for performance-related concepts and sub-concepts that were also relevant to software.

System performance requirements have been discussed from various viewpoints in the literature. Shang et al. analyzed [27] the VxWorks real time operating system used in the aerospace and medical fields including five significant performance indicators: task switching time, pre-emption time, interrupt latency time, message

communication time and semaphore shuffling time.

Alwadi et al. [28] proposed a framework for the quality of service (QoS) attributes and included performance as one of the prime system NFR, allowing performance requirements to be decomposed and allocated to a set of the system's functionalities.

Zhiwei et al. [29] proposed an approach for improving the concurrent system performance on the dynamic weighted k -out-of- n system (DWKNS). Subsequent to the state possibility and request of system components over time, this approach was combined with the Markov process with the universal generating function (UGF) method and the state probability for the performance of the system components.

Al-Sarayreh [30] considered system performance requirements to be more comprehensive than typical hardware-centric systems and proposed that dynamic system performance requirements be included with maintainability, upgradability, interface interoperability, reliability, safety and security (MUIRSS). MUIRSS should be analyzed and visibly connected to ensure that they are included with development.

The system dynamic performance of Kai and Huamin [31] for control systems indicates a relational variance of the controller design method. Their proposed method is used for the first order plus delay time system.

Krishna and Abraham [32] discussed the importance of the analysis of performance and memory NFR in real time embedded systems. Based on agile using incremental development, their development approach helps system engineers track the system performance requirements and related parameters throughout the development cycle. Their results are taken as a reference for a systematic analysis approach for memory and system performance NFR parameters using the most suitable mathematical methods.

Vila et al. [33] presented an approach for estimating the radio resource requirements for RAN slice admission control in order to describe the interference conditions of resource estimation method influences on system performance

requirements extracted from data analytics collected from management plans.

Ruberg et al. [34] proposed a data processing and cleaning method for a performance and energy consumption estimation approach to manage system performance requirements. Their approach links software component feature measurements (SCFM) and software performance quality indicators (SPQIs) to diagnose the software and functional requirements.

2.3. COSMIC functional size measurement method

There are currently five functional size measurement (FSM) methods adopted by ISO: the COSMIC Function Points method is the only second generation of such FSM methods, and its design has corrected a number of the defects identified in the other four FSM methods of the first generation.

Measuring software functional size is an important factor for managing and estimating the project budget early in the software development lifecycle. The COSMIC functional size measurement (FSM) method conforms to the measurement requirements proposed in ISO 14143-1 [35] and has been adopted as ISO 19761 [36]. This subsection presents the COSMIC generic model for software requirements and how such a model can be used to measure software functionalities with an ISO-recognized measurement unit.

In the COSMIC-FSM method, the functional user requirements are decomposed into one or more functional processes, each of which may be comprised of sub-processes and include a number of data movements.

Figure 2 illustrates the COSMIC generic model of software FR. The front-end direction of the model shows that users access the software through input/output devices (such as mouse and microphone) or engineered devices (such as sensors). The back-end direction shows that the software is accessed by storage hardware (such as RAM memory). Figure 2 also illustrates the following four types of data movement: ENTRIES (E): exchanges data groups from users or engineered devices to software (left-hand side in Figure 2). EXITS (X): exchanges data groups from software to users or engineered devices (left-hand side in Figure 2). READS (R): exchanges data groups from hardware storage to software (right-hand side in Figure 2). WRITES (W): exchanges data groups from software to hardware storage (right-hand side in Figure 2).

The core principle of the COSMIC-FSM method is to measure the size of software FR by identifying the recognized data movements (E, X, R and W). Once the data movements are identified, each type of data movement is assigned the value of one COSMIC Function Point (e.g., 1 CFP). The functional size of the software to be measured is obtained by summing the sizes of all the corresponding data movements. Since the

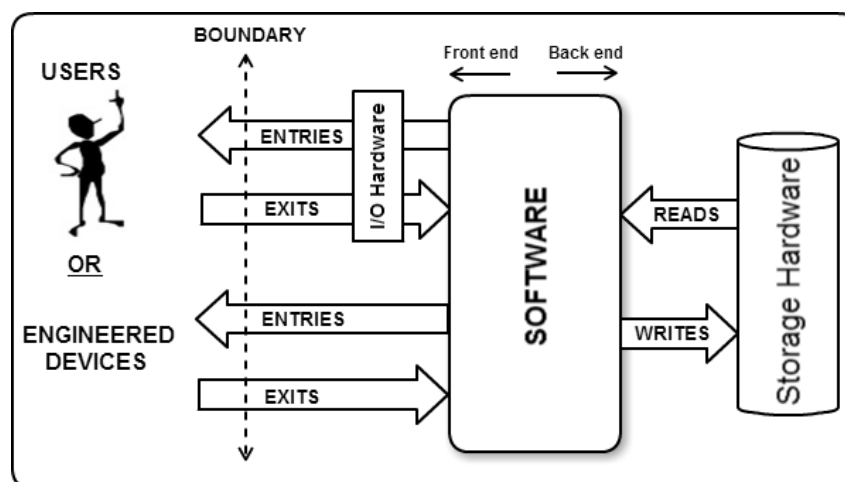


Figure 2. COSMIC generic model for software FR

COSMIC-FSM method aims to measure the size of the software, only the functional user requirements allocated to the software are considered in the measurement procedure.

Moreover, the COSMIC-FSM method is applicable to all the software development phases, from the analysis to implementation phases. Note that the COSMIC generic model in Figure 4 is not specific to any type of software nor to any particular method for describing functional user requirements. In the framework proposed in this paper, the COSMIC-FSM method is applied to measure the size of the software performance functional requirements with an ISO-recognized size unit.

2.4. Service-oriented architecture (SOA) and its COSMIC view

The service-oriented architecture (SOA) approach provides significant benefits to organizations, such as reducing software development and maintenance costs and increasing software quality by reusing services [37]. Various definitions have been introduced to define SOA, but none have been universally adopted. For instance, SOA has been defined as: 1) A process that involves the definition of the architecture, components, modules, interfaces, and

data for a system to satisfy specified requirements [36, 37]; 2) A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides means to offer, discover, interact with, and use capabilities to produce desired effects consistent with measurable preconditions and expectations [36]; 3) Utilization of loosely coupled software services to support business processes requirements and user requirements [37].

The COSMIC-SOA guidelines document illustrates how to measure the size of software services in an SOA context [37]. The term services in the COSMIC guideline refers to a suite of related functions of software FR and also to the separation of functions into distinct units, where these services are connected with each other by exchanging data, shared format or by coordinating activities between two or more services [37].

COSMIC-SOA guidelines offer three types of data movements – exchange services, intermediary services and data exchanges, which are described in more detail in the following sections.

2.4.1. COSMIC-SOA exchange messages

COSMIC-SOA exchanges messages (Figure 3) when an application needs information from a dif-

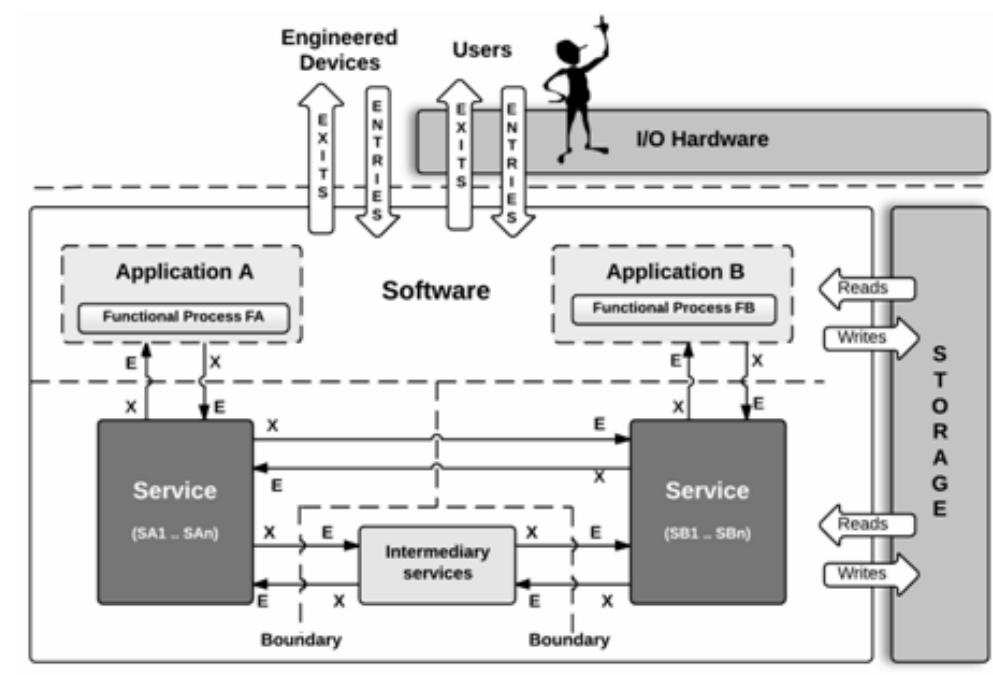


Figure 3. COSMIC-SOA guidelines for modeling data movements

ferent application. For instance, if application A needs to exchange data with application B, the services of application A will be invoked by the functional process of A to communicate with the services of application B to obtain the needed information. These calls between the functional processes of A and its services or between A services and B services are known as messages, where each message may involve one or more data movements [37].

2.4.2. COSMIC-SOA intermediary services

When services (Figure 3) of any application require data from another application in the overall SOA framework, the intermediary service will be used. For instance, if services of application A need data from services of application B, the services of application A will invoke the intermediary services to obtain the required data from the services of application B.

2.4.3. COSMIC-SOA data exchanges

For components in the same layer (e.g., in the application service layer) (Figure 3), two types of data movements can be used: direct and indirect message exchanges. For instance, in direct exchange, if the service of application A requires to exchange a message with a service of application B, it will use an Exit and/or an Entry for exchanging messages with the service of application B. While the indirect exchange occurs using storage, for instance, the service of application A writes the data in storage which is read later by service B [37].

2.5. Softgoal Interdependency Graphs

Softgoal Interdependency Graphs (SIGs) [38] have been proposed for analyzing and demonstrating NFR as softgoals. Each softgoal can be represented as decomposed into one or more specific goals using interdependency relations between the analyzed goals until arriving at solutions that satisfy the assigned NFR.

SIGs [39] illustrate three different types of goals at the high level: 1) Softgoals that satisfy

the NFR with the software, 2) Claim softgoals which enhance the rationale between related softgoals, and 3) Operationalization of system softgoals (including a set of processes, data representations and system behavior).

These SIGs [39] at the low level (i.e., subgoals) provide both positive and negative contributions to the assigned softgoals at the high level.

Softgoals and subgoals can interact with each other using the following relations [39]: 1) AND means that each softgoal is decomposed into more than one related goal and is satisfied if all the related goals are satisfied. 2) OR means that each softgoal is decomposed into one or more related goals and is satisfied if at least one related goal is satisfied. 3) EQUAL means that each softgoal is decomposed into one related goal and is satisfied if the linked goal is satisfied.

The SIGs [38, 39] approach uses the terms goals and subgoals to represent the conditions or criteria that the system should meet (e.g., non-functional requirements or quality attributes) instead of more commonly used terms in software engineering, such as functions and software specifications. In addition, the SIGs approach does not distinguish between the system view and the software view.

This research provides a mapping between some of the SIGs terms to the standards-based terms used in this paper, as presented in Figure 1. Therefore, the expression function to be specified is used instead of a functional goal while both are encoded as is in the SIG approach.

3. Performance requirements identification

This section introduces and discusses performance terms and views for identifying system performance NFR and related software performance FR, which may then be used for specifying and measuring the system performance requirements. Numerous terms and views are found throughout the ECSS and IEEE international standards as well as previous works in academia. These have addressed software performance FR derived from system performance FR and NFR (see Figure 1).

Figure 1 also illustrates system performance requirements expressed as either system performance NFR or system performance FR.

3.1. ECSS concepts for performance requirements

ECSS standards [20–25] mention the importance of establishing performance requirements in detail at both system and software levels during the development phase so as to evaluate the consistency and cohesion of the control system within the required standards. This includes: 1) The objective(s) for each designed control system, which are normally created by the requirements engineering process; 2) The formal mathematical requirements, which are created by the requirements analysis.

Enhancing and regularly improving software applications requires system monitoring and evaluation of system performance. The performance monitor [23] provides information related to the use of processor instruction execution and storage control. For example, to provide information related to the period of time passed between events in a processing system.

The performance monitor can be used to debug the software application and analyze system faults and errors by defining a machine's state at a specific point in time. The information from the performance monitor helps system engineers to evaluate and improve the performance of a given system, or by developing enhancements of performance requirements in new system design.

ECSS standards [20–25] define the following concepts and views for system performance requirements allocated to software: 1) Frequency domain requirements such as throughput time, which includes: Workload and Bandwidth; 2) Response to reference signals for command profiles, which includes: Response time, Settling time, and Tracking errors; 3) Accuracy and stability errors in the presence of disturbances: Performance errors (absolute and stability errors) for evaluating the accuracy and Knowledge errors (absolute and relative errors) for evaluating accuracy; 4) Processing speed includes: System scalability, and System concurrency; 5) Resource consumptions

include: Processor instruction execution, Main memory time, and Storage device time.

3.2. IEEE concepts for performance requirements

IEEE standard 830-1998 [26] describes the following terms and concepts for system performance requirements allocated to software as dynamic and static numerical requirements: 1) Dynamic numerical requirements, such as workload; 2) Static numerical requirements, such as capacity and concurrency. These two types of system performance requirements should be quantified with a measurable procedural method.

3.3. Describing system performance and related software functions

This section presents a brief description of system performance requirements and their allocated software performance requirements.

3.3.1. Performance dynamic requirements

Performance dynamic numerical requirements may involve the data amount, transaction number and tasks to be processed within a specific period of time for both normal and peak workload conditions [26]. The unified standards-based view includes two types of system requirements for dynamic requirements: the response to reference signals and throughput time.

Response to reference signals (RRS):

Response to reference signals refers to the specific values that change to a new value in a relatively short period of time, including response time or settling time values. Enhancing the response to reference signals is reflected positively on the system performance level. The unified standards-based view includes three types of functions for response to reference signals [25]: response time function, settling time function and tracking error function.

- Response time function (RTF) The response time is widely defined as the period of time that the system takes to respond to the user after receiving the user task. This relation

between response time and performance is an inverse relation, since a decrease in response time leads to an increase in performance level.

- Settling time function (STF) The settling time refers to the time required for the system to recover from an overload and to reach steady state. STF has also been called recovery time or reaction time [40]. It is important to reach steady state in as little time as possible.
- Tracking error function (TEF) Tracking errors includes tracking performance error. The knowledge error (KE) or errors resulting from the central processing unit (CPU) and the main memory are also important to minimize system errors. System performance, therefore, can be enhanced by increasing system accuracy and speed.

Throughput time (TT): Throughput time is the number of event responses carried out by the system in a specific period of time [41]. Thus, maximizing the throughput time leads to increasing performance of the system. The unified standards-based view includes two types of functions for throughput time:

- Bandwidth function (BF) The bandwidth function refers to the maximum amount of data that can be carried over a network or data-transmission medium in a unit of time. The throughput time is limited by the bandwidth function. Large bandwidth leads to more event responses over time [42].
- Workload function (WF) The workload function measures the number of transactions performed by the system within certain periods of time. The performance level is good when the system workload is significantly lower than its capacity [43]. Otherwise system performance will be slow.

3.3.2. Performance static requirements

Performance static numerical requirements are sometimes specified under a separate section as capacity. They may also involve information types, the amount of time handled, the number of simultaneous users and terminals supported [26]. Based on IEEE standards, the unified standards-based view includes three types

of system requirements for static numerical requirements: resource consumption, evaluation of processing speed and evaluation of accuracy.

Resource consumption (RC): The efficiency of system resources (such as CPU, main memory and system storage) significantly affects the system performance. Heavy resource consumption can lead to the system's inability to effectively deal with its processes [44, 45], therefore, slowing down or crashing causing poor system performance. Proper utilization of resources leads to high system performance.

The unified standards-based view includes three types of functions for resource consumption: main memory time function, storage device time function and processor instruction execution function.

- Main memory time function (MMTF) The main memory is also known as the system internal memory or primary memory; it is used to store the data that is in use. When the CPU requires access to specific data from the storage device, the main memory will access the storage device and retrieve the required data to be processed by the CPU [46]. The time spent to access data in the main memory needs to be as small as possible in order to optimize system performance.
- Storage device time function (SDTF) Storage devices have a huge capacity to hold data in a permanent way. Fast storage devices are preferred to slower devices. Storage speed is impacted by two factors: Access time: the average time to locate data on the storage medium, and Data transfer rate: the amount of data transferred to or from the device per second [47].
- Processor instruction execution function (PIEF) Computer instructions are a set of commands executed by the processor to perform specific functions. Increasing the speed of executing such instructions can significantly contribute to improving the system performance level.

Evaluation of accuracy (EA): The developed system should achieve a high level of accuracy (i.e., precision). The definition of accuracy varies from one system to another. For instance: In satel-

lite systems, accuracy refers to the positioning accuracy provided by the system [48]. In radio systems, accuracy refers to how closely the actual output frequency matches the set frequency [48].

System accuracy may be determined through measuring system error. Based on ECSS standards, the unified standards-based view includes two error types: performance error and knowledge error.

– Performance error (PE) is defined as the functions that quantify the difference between the system's desired state and the system's actual state [27, 28]. In the unified standards-based view, two common PE indices are used for measuring performance error:

1. Absolute performance error function (APEF) The absolute performance error is defined as the instantaneous value of the performance error at any given time [25]. Applying a specific mathematical operator on the performance error function determines the APE. In addition, each system has a maximum APE value, which the calculated APE should not exceed.
2. Performance stability error function (PSEF) The system stability is defined as the ability of the system to maintain a particular situation for a given time. The stability error is the peak-to-peak variation of the system attitude during the time period [25]. The PSEF is known as the change of error over a given time [25]. In addition, applying a specific mathematical operator to the performance error function and to the APE determines such a function. Just like the APE, each system has a maximum PSE value and the calculated PSE should not exceed the maximum APE.

It is possible to use other performance error indices if the system so requires.

– Knowledge error (KE) is defined as the functions that quantify the difference between the system's estimated (or known) state and its actual state [27, 28]. In the unified standards-based view, two common knowledge error indices are used for measuring knowledge error:

1. Absolute knowledge error function (AKEF) The absolute knowledge error (AKEF) is defined as the instantaneous value of the knowledge error at any given time [29]. Applying a specific operator to the KE function determines the AKE. Just like the performance error indices, each system has a maximum AKE value and the calculated AKE should not exceed the maximum APE.
2. Relative knowledge error function (RKEF) The relative knowledge error (RKEF) refers to the difference between the instantaneous knowledge error at a specific time and its mean value over a time interval containing that time [29]. It is possible to use two other KE indices types should the system require it.

Evaluation of processing speed (EPS):

The processing speed refers to how quickly the processor handles instructions. The processing speed for a CPU is measured by the CPU clock rate. A CPU with a high clock rate leads to high speed instruction processing. The unified standards-based view includes two types of functions for EPS: system scalability function and concurrency function.

1. System scalability function (SSF) System scalability function (SSF) is the system's ability to process increased workload while maintaining the required system performance level [48]. To make the system scalable, additional hardware is added, such as CPU or memory, without making any changes to the system architecture.
2. Concurrency function (CF) The concurrency function refers to executing several instructions simultaneously, which improves the use of system resources while also reducing the system response time [49, 50].

4. Measurement framework for performance requirements

Figure 4 shows the four main phases used to determine the proposed measurement framework for system performance requirements:

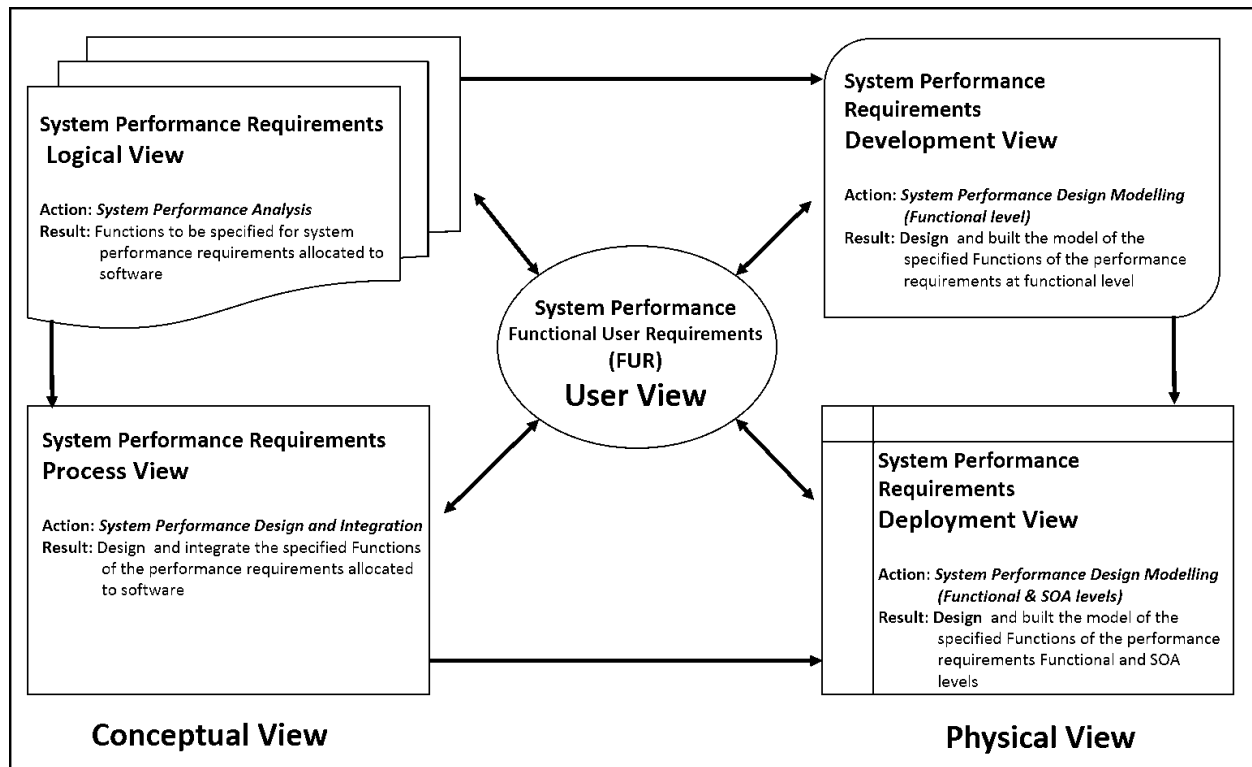


Figure 4. System performance requirements from four different views

Phase 1 (Logical view): Identify and analyze the functions to be specified for system performance requirements. In this phase, the logical views are defined based on the functional user requirements view.

Phase 2 (Process view): Design and integration of the identified system performance requirements. In this phase, the process view is developed which includes default, rationale and component approaches. For more details, see Figure 4.

Phase 3 (Development view): Design a system performance requirements model using SIGs at the functional level. In this phase, the system performance model is designed and built by integrating the logical and process views [51].

Phase 4 (Deployment view): Design a measurement context at functional and service levels with COSMIC-SOA to measure the functional size of the software performance requirements. In this phase, the design measurement strategy is used to develop the proposed generic model of the system performance requirements allocated to software based on functional user requirements

(FUR) views. Next, an architectural measurement context for the service levels is designed by applying the COSMIC-SOA guideline to develop a framework in an SOA context.

“For preliminary design of the performance requirements model, the SIGs tool is used. For the performance measurement model, we used another visualization tool called LibreOffice Draw Tool. This tool extends the preliminary performance model in SIGs by adding the generic COSMIC measurement procedure and adopting the detailed COSMIC-SOA to the proposed performance model.”

4.1. Integration of system performance functions to be allocated to software (Phases 1 and 2)

The terms and views found in ECSS and IEEE to describe the performance NFR in Section 3.3 are combined and integrated using both their logical and process views. This leads to a dynamic view (Figure 5) and a static view (Figure 6) of the system performance functions and related

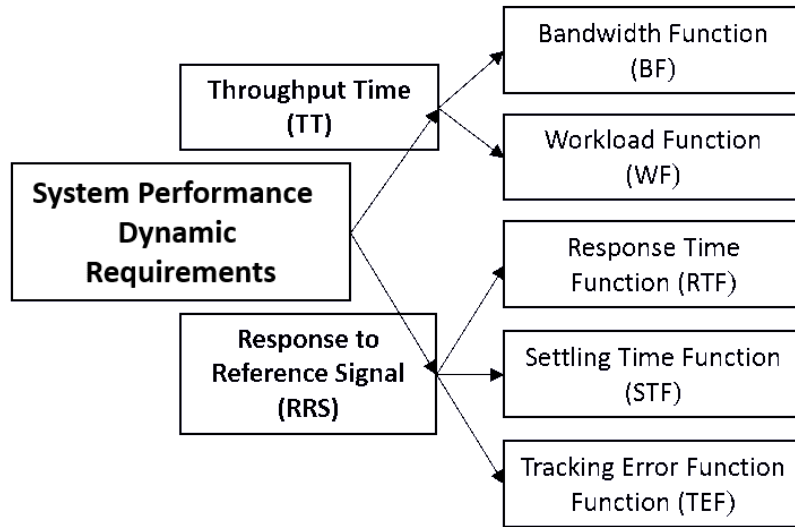


Figure 5. Integrated model of ECSS and IEEE system performance dynamic requirements and related functions

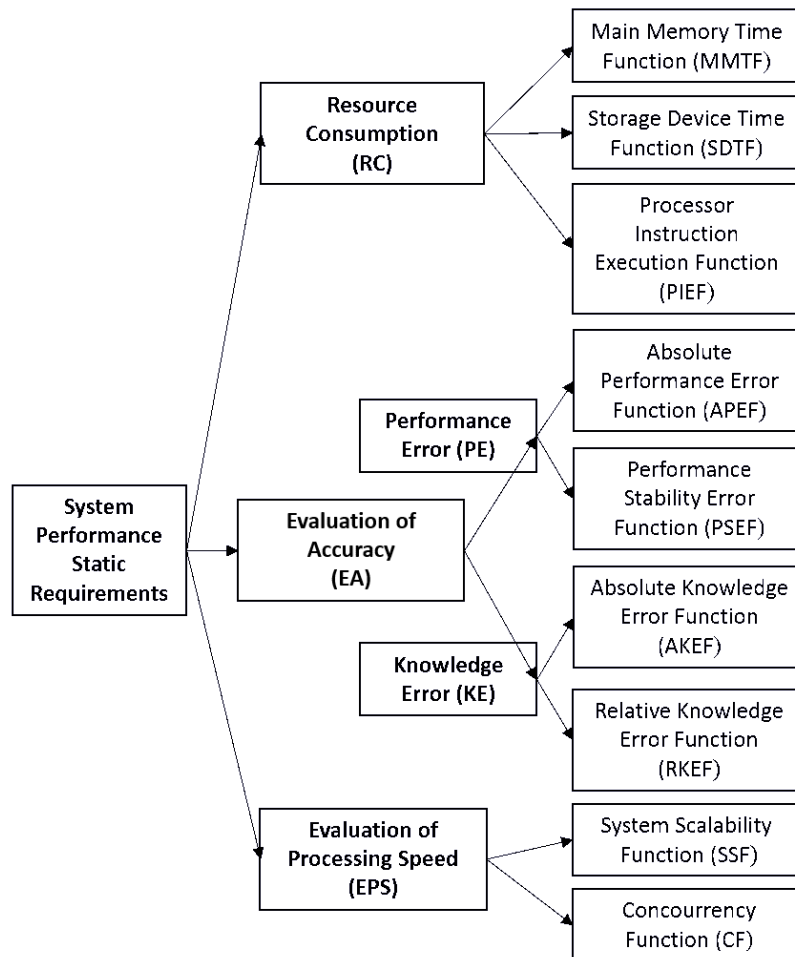


Figure 6. Integrated model of ECSS and IEEE system performance static requirements and functions

software functions, which can then be used to specify and measure them.

4.2. Design of system performance requirements at the functional level (Phase 3)

The proposed framework for system performance requirements is established at two main levels: the functions level and the services level. This section illustrates and describes in detail the framework at the functional level. In this section, the software interdependency goals (SIGs) and ISO 19761 are used to design the framework of the system performance NFR allocated to software at the functional level, divided into four sub-models, see Figure 7.

4.2.1. System performance dynamic requirements (SPDR)

The functions for the system performance dynamic requirements must address:

The throughput time (TT): The throughput time (TT), which involves two functions: the bandwidth function and the workload function. Figure 7 illustrates the interdependency relationships between these functions:

1. The bandwidth (BF) and the workload (WF) functions may exchange data in a direct way with each other, and/or
2. may exchange data in an indirect way through the persistent storage, and as well
3. may require data from any function in the overall performance framework through intermediary services.

The response to reference signals (RRS):

The response to reference signals (RRS) involves three functions: the response time function (RTF), the settling time function (STF) and the tracking error function (TEF). Figure 7 shows the interdependency relations between these functions:

1. The response time, settling time and tracking error functions may exchange data in a direct way with each other;
2. may exchange data in an indirect way through the persistent storage;

3. may require data from any function in the overall performance framework through intermediary services.

4.2.2. System performance static requirements (SPDR)

The system performance static requirements include four function types – see Figure 7.

Resource consumption (RC): Resource consumption (RC) which involves three functions: the main memory time function (MMTF), the storage device time function (SDTF) and the processor instruction execution function (PIEF). Figure 7 illustrates the interdependency relationships between these functions:

1. They may exchange data in a direct way with each other, and/or
2. may exchange data in an indirect way through the persistent storage, and
3. may require data from any function in the overall performance framework through intermediary services.

The evaluation of accuracy (EA): The evaluation of accuracy (EA) which is composed of performance error (PE) and knowledge error (KE). Performance error involves two specified functions: the absolute performance error function (APEF) and the performance stability error function (PSEF). The knowledge error also involves two specified functions: the absolute knowledge error function (AKEF) and the relative knowledge error function (RKEF). Figure 7 illustrates the interdependency relationships between these functions:

1. They may exchange data in a direct way with each other;
2. may exchange data in an indirect way through the persistent storage;
3. may require data from any function in the overall performance framework through intermediary services.

The evaluation of accuracy (EA): The EPS which is composed of two functions: the system scalability function (SSF) and the concurrency function (CF). Figure 7 illustrates the interdependency relationships between these functions:

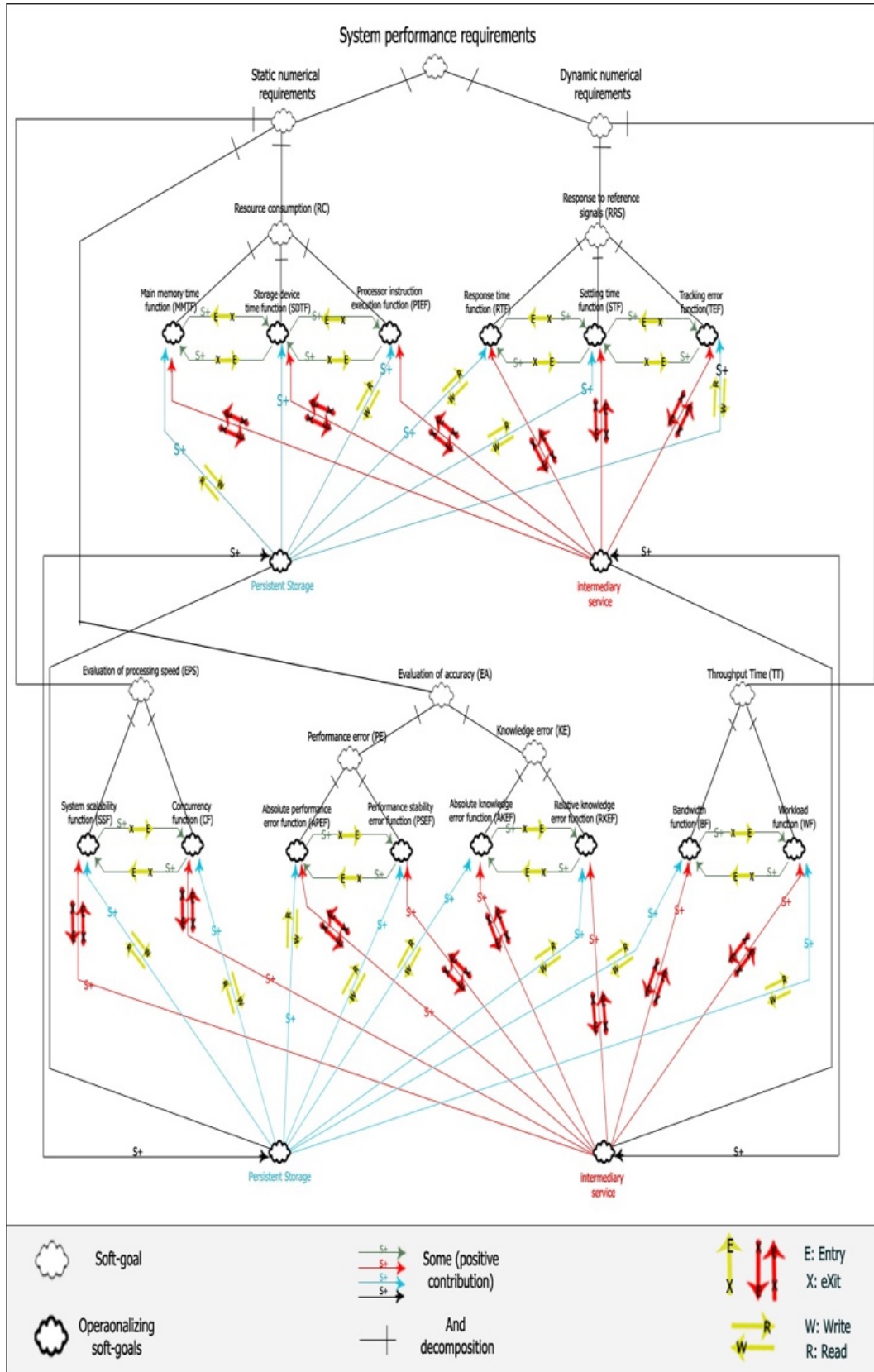


Figure 7. Full view of the system performance NFR at the functional level

1. They may exchange data in a direct way with each other;
2. may exchange data in an indirect way through the persistent storage;
3. may require data from any function in the overall performance framework through intermediary services.

4.3. Design of the measurement framework for system performance NFR (Phase 4)

This section introduces the framework in an SOA context using the COSMIC-SOA guidelines. For clarity, the proposed framework is divided into the seven sub-models illustrated in Figures 8 to 13. Figure 14 shows the full view in the SOA.

Figure 8 shows that all the derived functions from resource consumption have their own services. The interdependency relations between these functions and their services are:

- The main memory time function, the storage device time function and the processor instruction execution function may require data from their services using EXIT and ENTRY data movements.
- The main memory time service may exchange data at a service layer either in a direct way with the storage device time service using COSMIC EXIT and ENTRY data movements. Or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The storage device time service may exchange data at a service layer either in a direct way with the main memory time service and the processor instruction execution service using COSMIC EXIT and ENTRY data movements. Or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The processor instruction execution service may exchange data at a service layer either in a direct way with the storage device time service using COSMIC EXIT and ENTRY data movements. Or it may exchange data in

an indirect way through the persistent storage using COSMIC READ and WRITE data movements.

- The main memory time service, the storage device time service and the processor instruction execution service may require data from any service in the overall performance framework through the intermediary service using COSMIC EXIT and ENTRY data movements.

Figure 9 shows that all the derived functions from the evaluation of processing speed have their own services. The interdependency relations between these functions and their services are:

- The system scalability function and the concurrency function may require data from their services using EXIT and ENTRY data movements.
- The system scalability service may exchange data at a service layer either in a direct way with the concurrency service using COSMIC EXIT and ENTRY data movements. Or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The concurrency service may exchange data at a service layer either in a direct way with the system scalability service using COSMIC EXIT and ENTRY data movements. Or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The system scalability service and the concurrency service may require data from any service in the overall performance framework through the intermediary service using COSMIC EXIT and ENTRY data movements.

As mentioned in Section 4.1, the evaluation of accuracy (EA) is comprised of two error types: performance error (PE) and knowledge error (KE). Figure 10 shows that all the derived functions from performance error have their own services. The interdependency relations between these functions and their services are:

- The absolute performance error function and the performance stability error function may require data from their services using EXIT and ENTRY data movements.

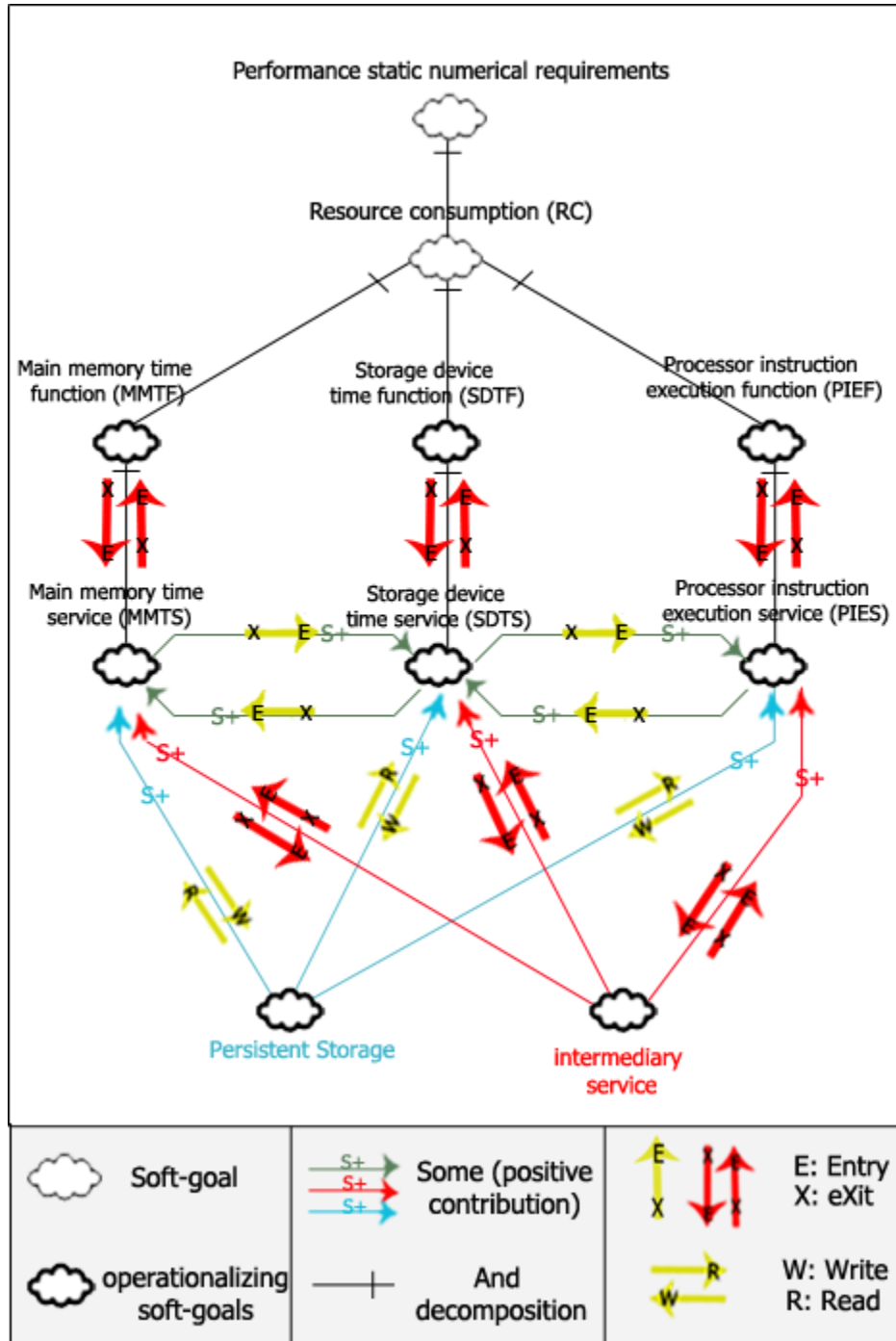


Figure 8. Resource consumption sub-model in an SOA context

- The absolute performance error service may exchange data at a service layer either in a direct way with the performance stability error service using COSMIC EXIT and ENTRY data movements. Or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The performance stability error service may exchange data at a service layer either in a direct way with the absolute performance error service using COSMIC EXIT and ENTRY data movements. Or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.

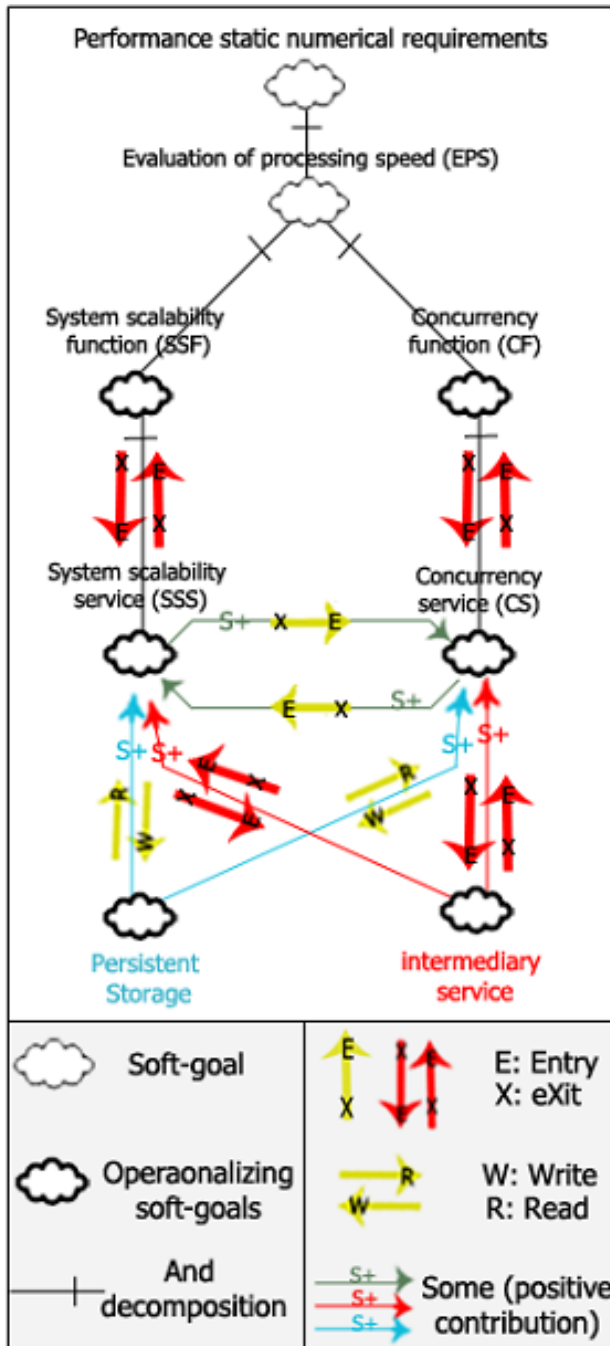


Figure 9. Evaluation of processing speed sub-model in an SOA context

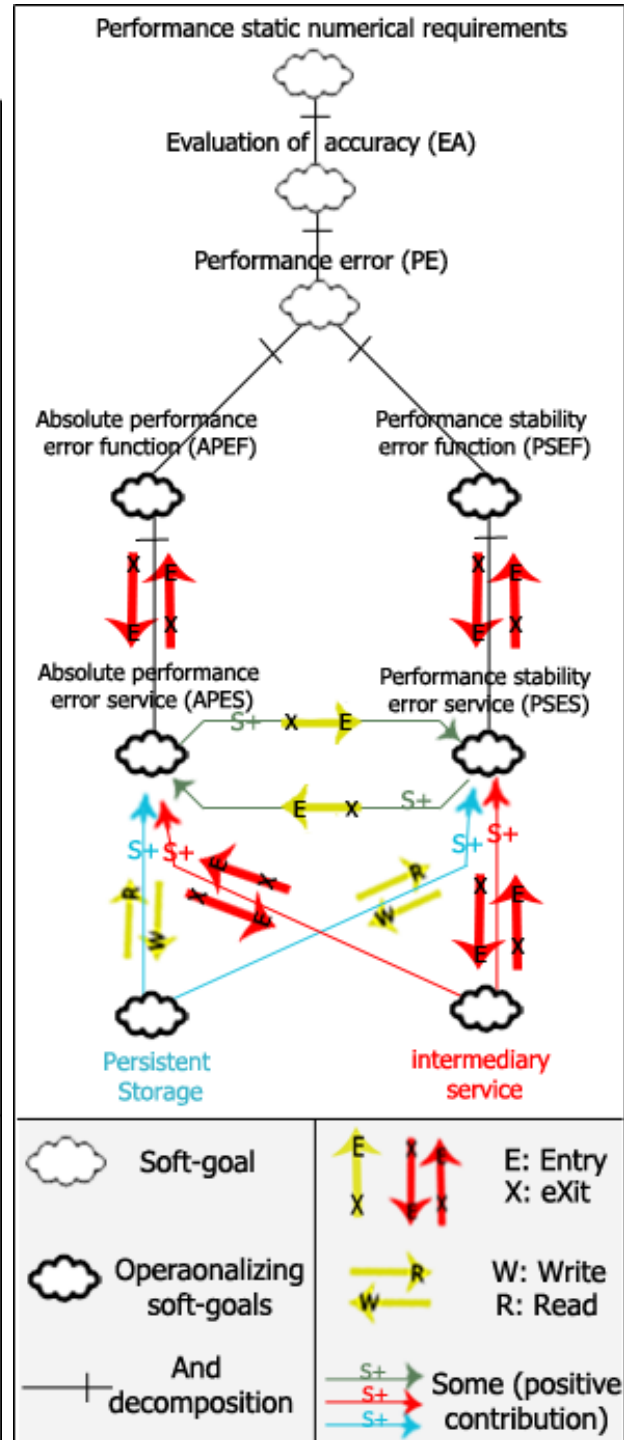


Figure 10. Performance error sub-model in an SOA context

- The absolute performance error service and the performance stability error service may require data from any service in the overall performance framework through the inter-

mediary service using COSMIC EXIT and ENTRY data movements.

Figure 11 shows that all the derived functions from the knowledge error (KE) have their own

services. The interdependency relations between these functions and their services are:

- The absolute knowledge error function and the relative knowledge error function may require data from their services using EXIT and ENTRY data movements.

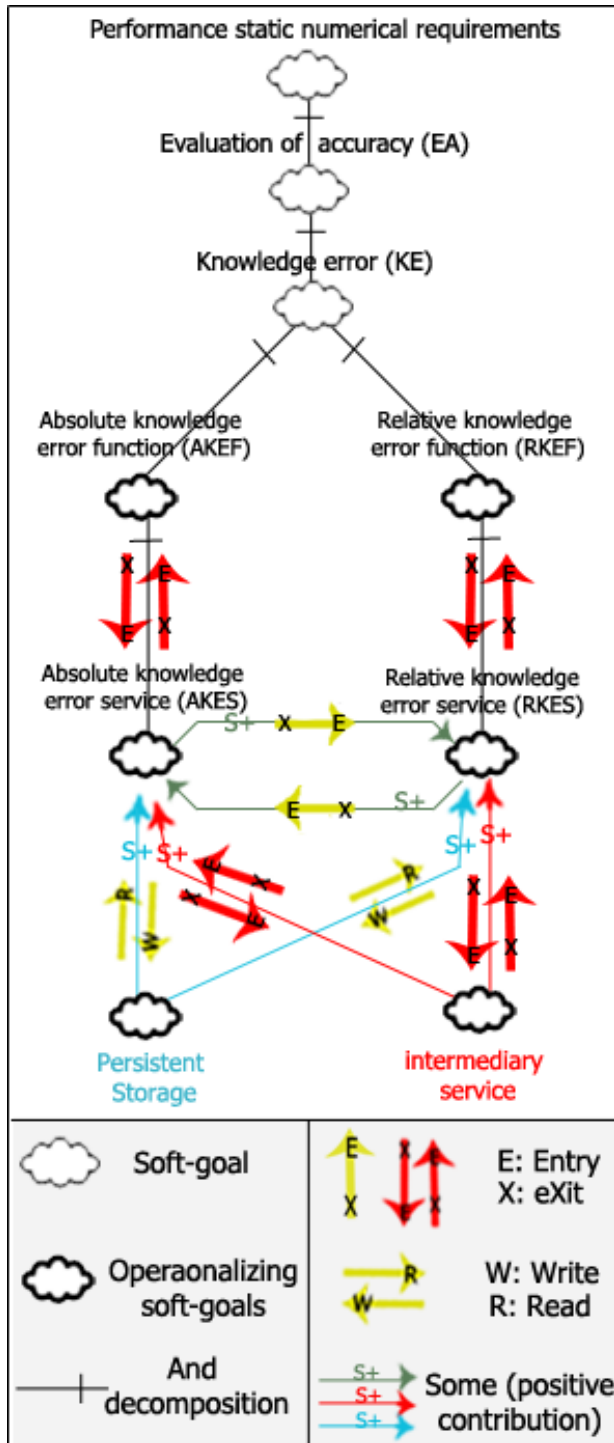


Figure 11. Knowledge error sub-model in an SOA context

- The absolute knowledge error service may exchange data at a service layer either in a direct way with the relative knowledge error service using COSMIC EXIT and ENTRY data movements or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The relative knowledge error service may exchange data at a service layer either in a direct way with the absolute knowledge error service using COSMIC EXIT and ENTRY data movements. Or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The absolute knowledge error service and the relative knowledge error service may require data from any service in the overall performance framework through the intermediary service using COSMIC EXIT and ENTRY data movements.

Figure 12 shows that all the derived functions from the response to reference signals have their own services. The interdependency relations between these functions and their services are:

- The response time function, the settling time function and the tracking error function may require data from their services using EXIT and ENTRY data movements.
- The response time service may exchange data at a service layer either in a direct way with the settling time service using COSMIC EXIT and ENTRY data movements or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The settling time service may exchange data at a service layer either in a direct way with the response time service and the tracking error service using COSMIC EXIT and ENTRY data movements or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The tracking error service may exchange data at a service layer either in a direct way with the settling time service using COSMIC EXIT and ENTRY data movements. Or it may ex-

change data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.

- The response time service, the settling time service and the tracking error service may require data from any service in the overall performance framework through the intermediary service using COSMIC EXIT and ENTRY data movements.

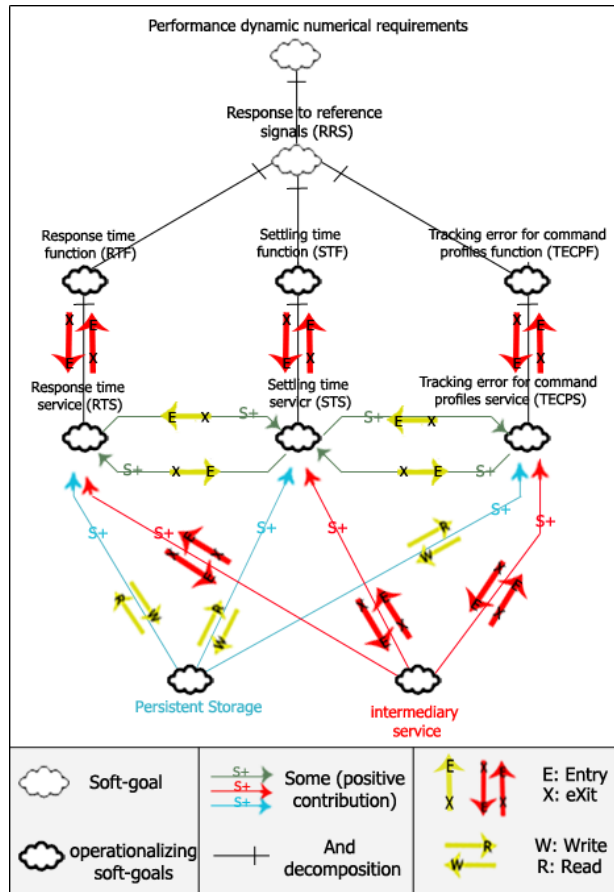


Figure 12. Response to reference signals sub-model in an SOA context

Figure 13 shows that all the derived functions from the throughput time (TT) have their own services. The interdependency relations between these functions and their services are:

- The bandwidth function and the workload function may require data from their services using EXIT and ENTRY data movements.
- The bandwidth service may exchange data at a service layer either in a direct way with the workload service using COSMIC EXIT and ENTRY data movements. Or it may exchange

data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.

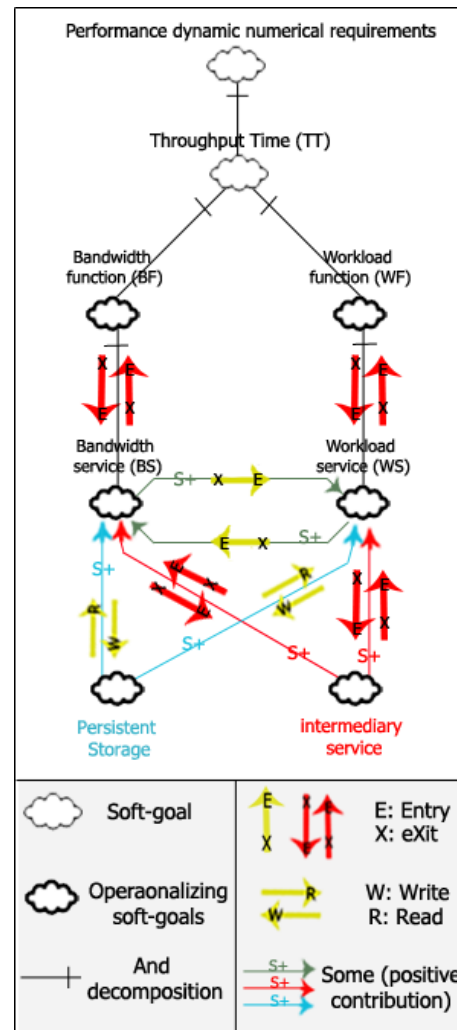


Figure 13. Throughput time sub-model in an SOA context

- The workload service may exchange data at a service layer either in a direct way with the bandwidth service using COSMIC EXIT and ENTRY data movements. Or it may exchange data in an indirect way through the persistent storage using COSMIC READ and WRITE data movements.
- The bandwidth service and the workload service may require data from any service in the overall performance framework through the intermediary service using COSMIC EXIT and ENTRY data movements.

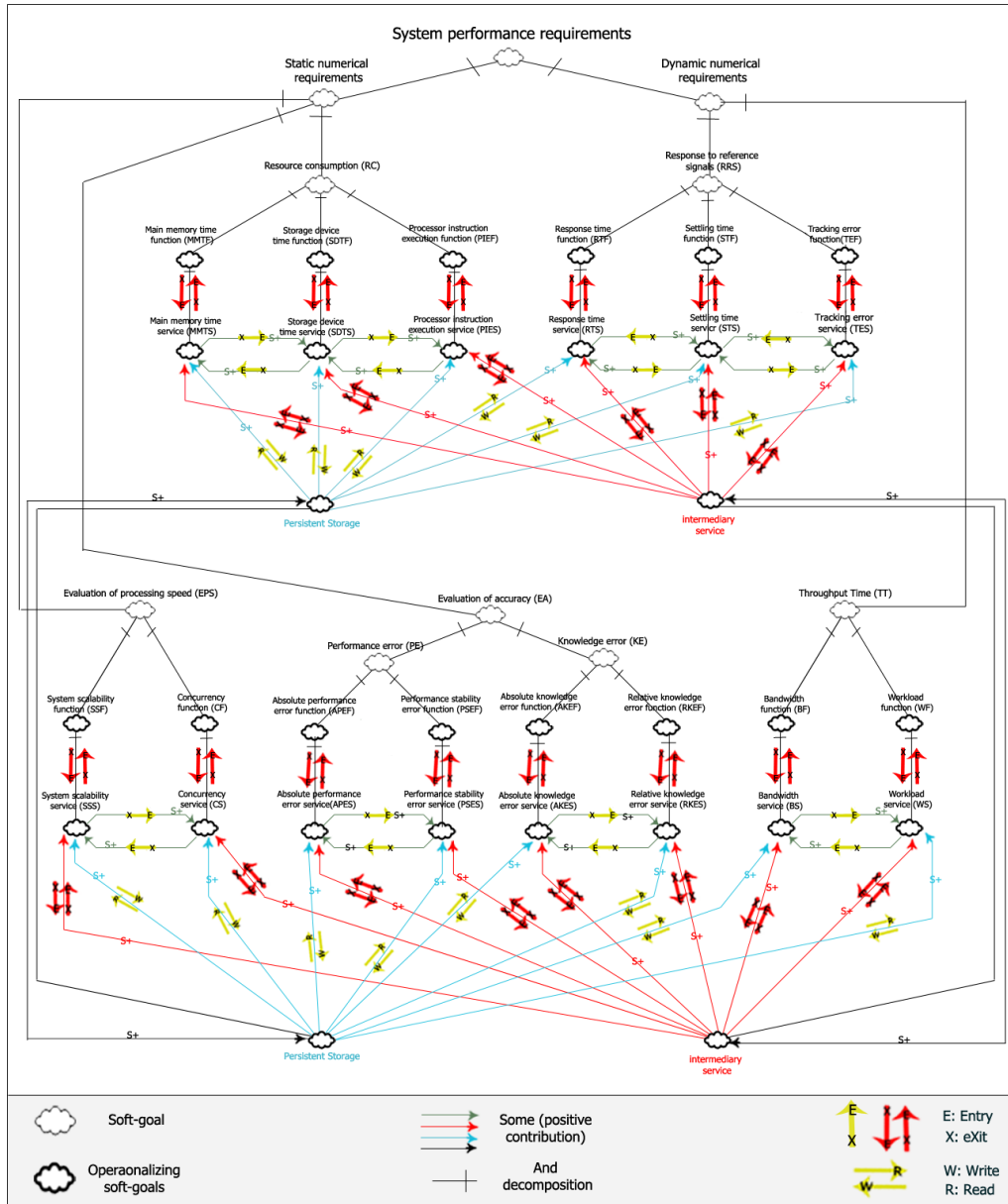


Figure 14. Full view of the system performance NFR model in an SOA context

Figure 14 illustrates the full view for the measurement framework of the system performance requirements on the basis of the previous sub-models in Figures 8–13 at functional level and in an SOA context.

From Figure 14, the following points can be observed for measurement purposes:

- In the direct data exchange situation, each EXIT and ENTRY data movement will be assigned a size of 1 CFP.
- In the indirect data exchange situation, each READ and WRITE data movement will be assigned 1 CFP.

- Data required through intermediary services that requires using 4 EXITS and 4 ENTRIES will be assigned 8 CFP.

5. Illustrative example: ATM banking system

5.1. Overview

Banking systems provide a variety of financial services to individuals, businesses and governments. An automated teller machine (ATM) is

a computerized system found in a public location. Customers are identified by inserting a smartcard that contains a unique number and some information about the customer and their account status. The services typically provided by ATM banking systems include: accepting deposits, cash withdrawal, issuing balance statements, pre-paid mobile charges and money transfers. To achieve high customer satisfaction, such systems must demonstrate high quality levels including: excellent performance, security and reliability.

5.2. Purpose and process

The purpose of this example is to present a “proof-of-concept” on a small-scale of the concepts proposed in this paper by illustrating the use of the proposed measurement framework for system performance requirements allocated to software (as in Figure 14). More specifically, to derive the system performance requirements allocated to an ATM system, and to measure the functional size of these allocated requirements using the COSMIC method. This illustrative example was realized by applying the following steps:

- Analyze and specify the main components of the ATM internal structure in a physical view.
- Design the workflow scenario-based application for the customer view.
- Identify the ATM functional user requirements for the customer and system views.
- Specify the ATM system requirements allocated to software.
- Specify the ATM system performance requirements allocated to software as an extended view to step 4.
- Map the allocated system performance requirements with the proposed framework.
- Measure the functional size of the allocated system performance requirements to the specified banking system with the COSMIC ISO-recognized measurement unit.

5.3. ATM internal structure system

Here we detail the internal structure of the bank ATM and the relationships among its various

parts. An ATM typically consists of several devices such as: central processor unit (CPU), crypto processor, memory, customer display, function key buttons (typically situated near the display), smart chip card reader, encrypting PIN pad, customer receipt printer, vault, and modem.

The vault stores all devices and parts that require limited access, such as:

- Cash dispensing mechanism (CDM),
- Deposit mechanism (DM),
- Security sensors (SS) (e.g., magnetic, thermal, seismic, gas),
- Electronic journal system (EJS) to keep system log,
- Cash dispenser (CD) which includes several removable cash cartridges, deposit mechanism and removable deposit cartridges.

The software specifications for the ATM system are: read the ATM card, count currency notes, connect to bank network, take input from user, validate user, dispense cash to user and receive deposit envelopes from the user through deposit slot.

5.4. ATM block diagram

The set of ATM system scenarios includes authentication of the PIN entered with the one encrypted on the card. Once the PIN is confirmed, the customer can access their bank account to make the chosen transaction. Or else the system shows a suitable message to clarify rejection of access. Figure 15 illustrates the first instantiation scenario for customer authentication for the ATM as follows:

- The client inserts his/her smartcard; the card reader processes the smartcard’s data using the card transaction handler, and informs the system that the smartcard is valid.
- The card transaction handler displays a message on the ATM screen asking for the customer PIN number.
- The ATM screen asks the customer to enter the PIN and the customer enters PIN code which is passed on to the card transaction handler.
- The card transaction handler verifies and gives authorization if the PIN is correct; if not,

a message appears on the screen to inform the customer that PIN number is invalid.

- The customer enters PIN again when the message appears on the screen to enter the PIN code number.
- If the customer has not provided the correct PIN in three iterations, the card reader will capture the customer smartcard and the session is terminated.

The second instantiation scenario after customer authentication for the ATM is as follows:

- The main menu that appears on the ATM system screen contains three types of transactions: "get account balance inquiry" (choice 1), "cash withdrawal" (choice 2) and "money deposit" (choice 3). A choice to allow the user to exit the system (choice 4) appears as well.
- The user at that point chooses either to make a transaction by entering one of the three choices or exits the system.
- If the client enters "get account balance inquiry", the ATM retrieves the balance from the bank's database and the screen displays the client's account balance.
- If the client enters "cash withdrawal", the ATM screen displays a menu holding typical withdrawal amounts such as 50, 100, 200.
- The withdrawal menu also displays a choice to permit the customer to cancel the transaction.

- If the withdrawal amount selected is larger than the client's account balance, the screen displays a message telling the client to select a smaller amount. The ATM then returns to the beginning of this scenario.
- If the withdrawal amount selected is less than or equal to the client's account balance, the ATM proceeds and issues the client's requested amount.
- Then the ATM subtracts the withdrawal amount from the client's account in the bank's database.
- The screen displays a message informing the user to take money.

5.5. ATM functional requirements (FR)

The functional requirements (FR) represent the system tasks from the stakeholder perspective and are typically derived from the context of use. The FR perspective can describe customer scenarios, system goals and objectives within the system environment and can connect these perspectives with assigned hardware resources.

Customer requirements are a subset of stakeholder requirements and can be collected in the stakeholder requirements specification document together with other perspective scenarios which have been derived from the system block diagram

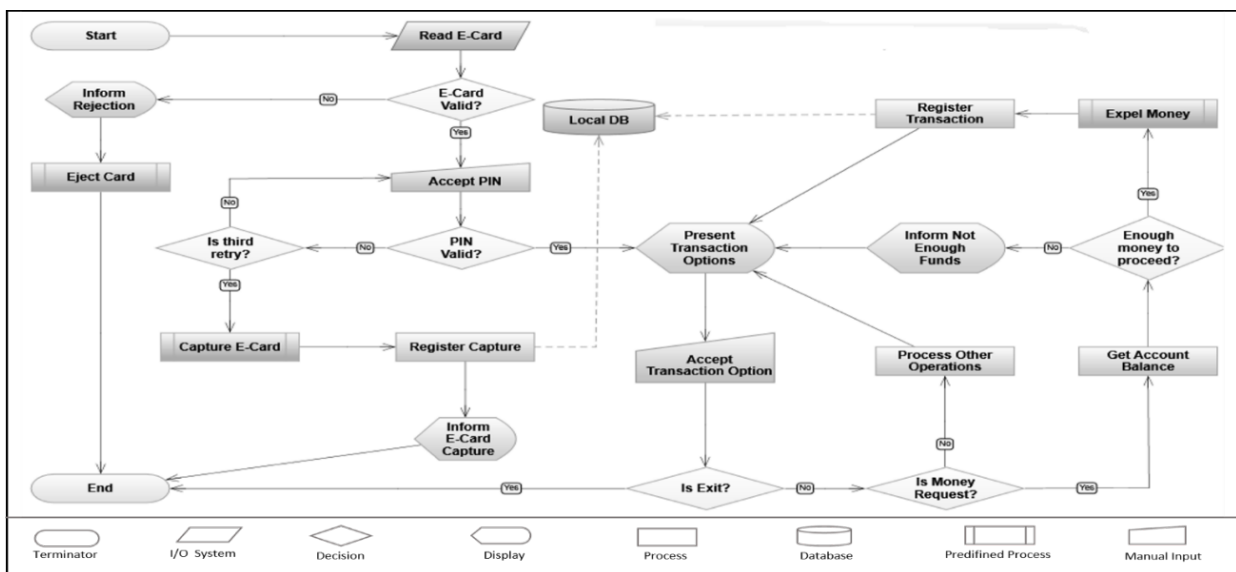


Figure 15. ATM functional requirements at the system level

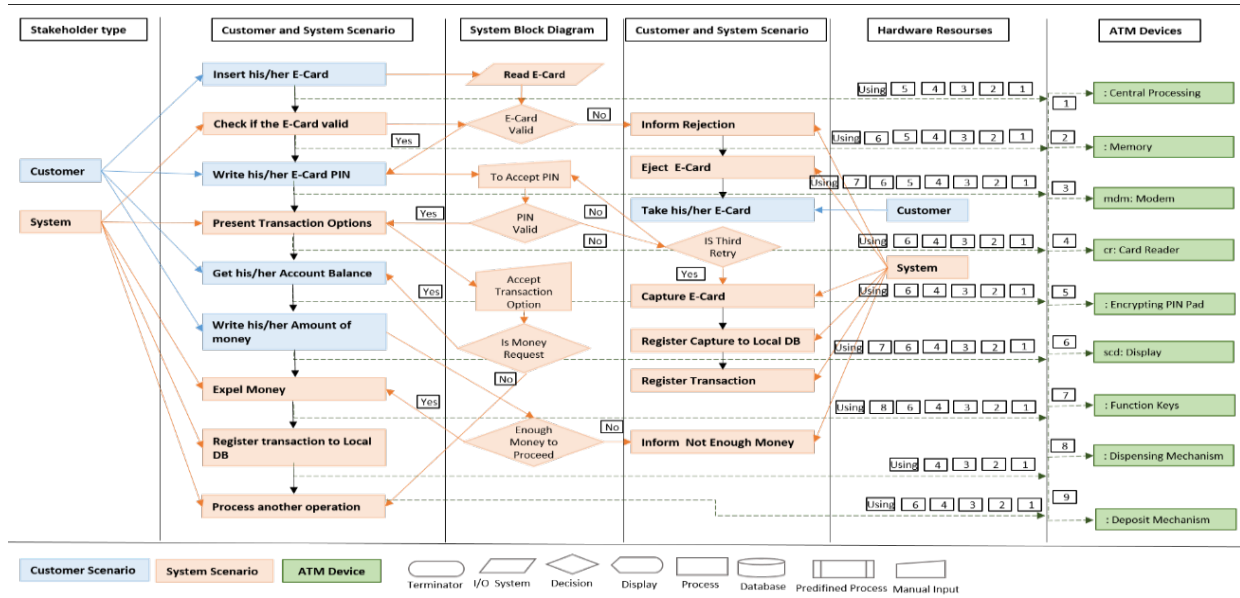


Figure 16. Customers and system scenarios identified from the ATM functional requirements

of Figures 15 and 16, including the identification of related hardware performance.

5.6. ATM system requirements allocated to software

Each system FR should be allocated to some specific software and/or physical component. Allocation should be defined in the early phases of the system development life cycle. At a high level in the system NFR this allocation impacts the design of the system architecture. Figure 17 illustrates the customer FR perspectives connected with software goals and sub software goals to derive the system requirements allocated to software.

5.7. System performance NFR allocated to software functions

This section presents an instantiation of some system performance requirements, and an example of their allocation to software. For this example, the following system performance NFR have been selected for the ATM system:

- Requirement 1: The maximum data transmission over the system network shall be 100 megabits per second.
- Requirement 2: The maximum time to respond to a customer transaction shall be one second.

- Requirement 3: The main memory access time in the system shall be 50 nanoseconds.
- Requirement 4: The system shall be scalable to handle the increased workload while maintaining the system performance level.
- Requirement 5: The system shall support the concurrent execution to execute the customer's transactions in a concurrent way.
- Requirement 6: The maximum time to recover the system from the instability state shall be two minutes.
- Requirement 7: The storage device access time in the system shall be 35 milliseconds and the data transfer rate shall be 156 megabytes per second.
- Requirement 8: The processor of the system shall be able to execute 2000 instructions per second.

For this example, these system performance NFR were allocated to the following software functions, see Figure 18:

- Requirement 1: to the bandwidth function and its services.
- Requirement 2: to the response time function services.
- Requirement 3: to the main memory time function services.
- Requirement 4: to the system scalability function services.

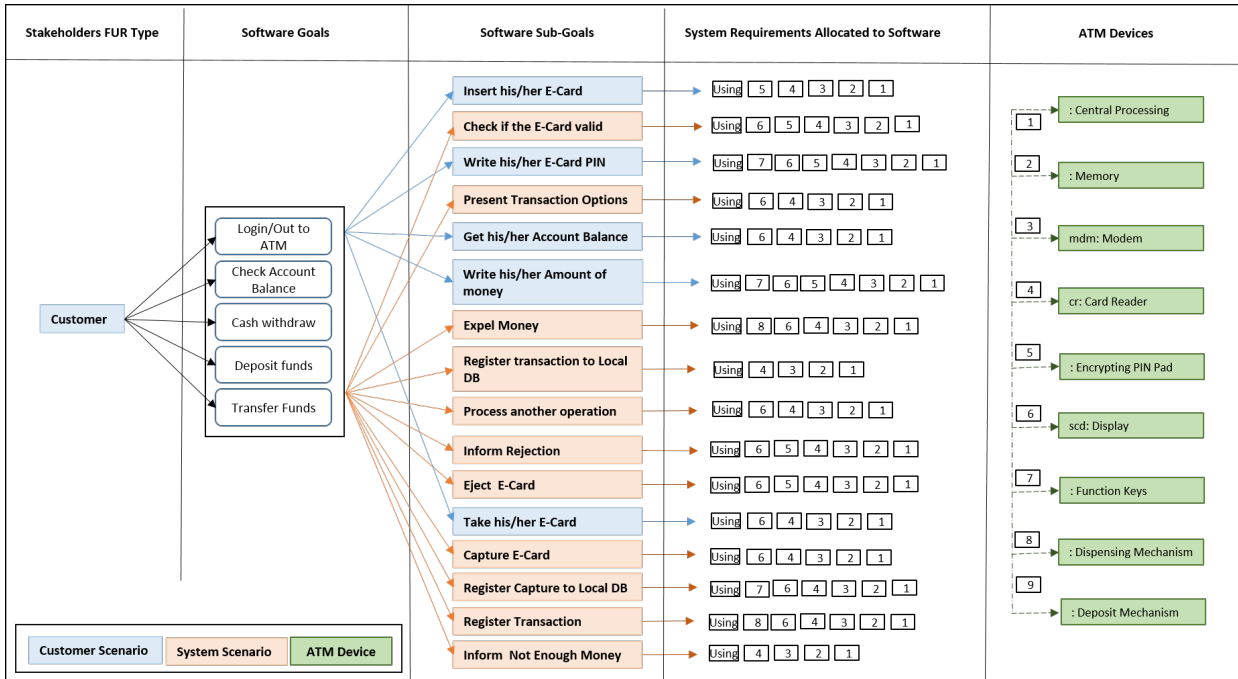


Figure 17. ATM customer and system scenarios allocated to software and ATM devices

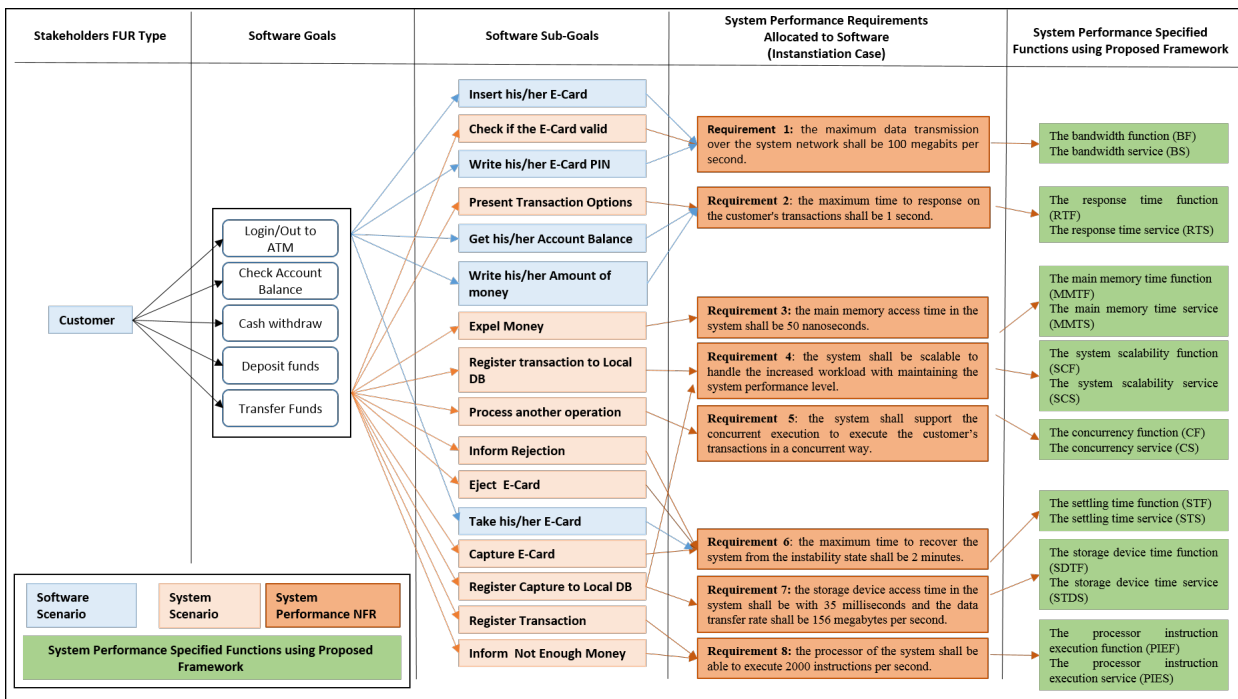


Figure 18. ATM system performance NFR allocated to software functions

- Requirement 5: to the concurrency function services.
- Requirement 6: to the settling time function services.
- Requirement 7: to the storage device time function services.
- Requirement 8: to the processor instruction execution function.

5.8. Mapping system performance NFR to an SOA context

This section maps the specified system performance requirements for the ATM system within an SOA context – see Figure 19. It can then be used to measure the instantiation case of the specified system performance NFR allocated to software for the banking ATM.

5.9. Measuring the specified system banking performance NFR (instantiation case)

This step identifies the detailed data movements for the allocated software performance functions in an SOA context. It is important to note that the performance NFR may require additional resources be added (i.e., hardware) to the system.

In this example, for illustrative purposes, a single data group was selected for each specified performance function, while in an industrial context these performance functions may require more than one data group. Table 1 shows the COSMIC measurement of the system performance NFR allocated to the software requirements at functional and service levels.

This example shows the corresponding COSMIC size for the selected specified requirements at the functional and services level (Figures 18 and 19). For measurement purposes they correspond to COSMIC data movements as follows:

Requirement 1 (R1): is allocated to one identified function (bandwidth function and its services) for the following customer-FR (Insert E-card, check if E-card is valid and Write E-card PIN). It extracts the identification data from the smartcard and invokes its own three services between three customer-FR, and then uses the persistence storage twice (once to check if the card is valid and second to check if the PIN number is correct. Here, we consider that the PIN is entered correctly).

The measurement of the functional size using ISO 19761 (COSMIC) for R1 is calculated based on data movements between groups of processes as follows:

- Three identification functions (3 ENTRY and 3 EXIT).

- Two identification services (4 ENTRY and 4 EXIT).
- Functional and services processes use persistence storage twice to obtain information about the card (2 READ and 2 WRITE).

The total functional size for R1 is 18 CFP.

Requirement 2 (R2): is allocated to one identified function (response time function and its services) for the following customer-FR (Present transaction options, get account balance and Write amount of money).

A. The main menu on the ATM system screen displays three types of transactions options. The measurement of the COSMIC functional size for R2-A is as follows:

- One identification function (1 ENTRY and 1 EXIT).
- One identification service (2 ENTRY and 2 EXIT).
- Functional and services processes use persistence storage once, which appears on the main menu of the ATM application (1 READ and 1 WRITE).

The total functional size for R2-A is 8 CFP.

B. In this instantiation the customer enters “get account balance inquiry”. The ATM retrieves the client’s account balance from the bank’s database which is then displayed on the screen. The measurement of the COSMIC functional size for R2-B is as follows:

- One identification function (1 ENTRY and 1 EXIT).
- One identification service (2 ENTRY and 2 EXIT).
- Functional and service processes use persistence storage once which appears on the main menu for the ATM application (1 READ and 1 WRITE).
- One intermediary service is needed to retrieve the customer account balance from the external database (4 Entry and 4 Exit).

The total functional size for R2-B is 16 CFP.

C. When the customer enters “cash withdrawal”, the ATM screen displays a menu showing typical withdrawal amounts, such as 50, 100, 200. In this

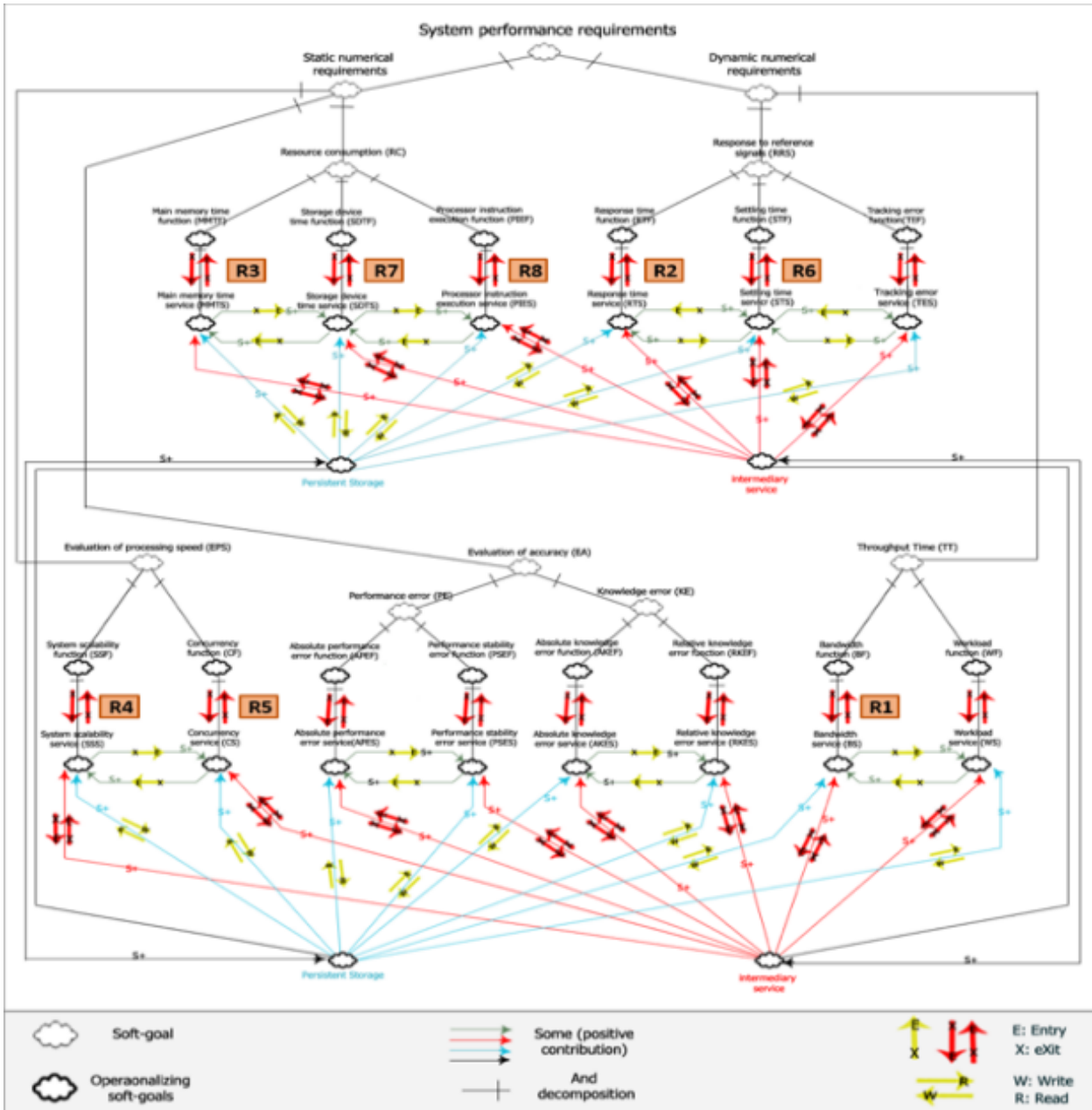


Figure 19. ATM system performance requirements allocated to software functions within an SOA context

instantiation, the customer can enter their own desired amount. The measurement of the COSMIC functional size for R2-C calculated based on data movements between groups of processes is described as follows:

- One identification function (1 ENTRY and 1 EXIT).
- One identification service (2 ENTRY and 2 EXIT).

- One intermediary service is needed to subtract the amount of money from the customer account balance (i.e., the customer has 300, takes 50, the remainder is 250) (4 ENTRY and 4 EXIT).
- Functional and service processes use persistence storage twice to obtain information about the card (1 READ and 1 WRITE).

The total functional size for R2 (A, B, C) is $(8 + 16 + 16) = 40$ CFP.

Requirement 3 (R3): is allocated to one identified function (main memory time function and its services) for the customer-FUR (Expel money).

The measurement of the COSMIC functional size for R3 is calculated based on data movements between groups of processes and is described as follows:

- One identification function (1 ENTRY and 1 EXIT).
- One identification service (2 ENTRY and 2 EXIT).

The total functional size for R3 is 6 CFP.

Requirement 4 (R4): is allocated to one identified function (system scalability function and its services), twice for the customer-FUR (Register transaction to local DB and Register capture to local DB).

The measurement of the COSMIC functional size for R4 is calculated based on data movements between groups of processes and is described as follows:

- One identification function twice (2 ENTRY and 2 EXIT).
- One identification service twice (4 ENTRY and 4 EXIT).
- Twice the intermediary service while the system uses the database (8 ENTRY and 8 EXIT).

The total functional size for R4 is 28 CFP.

Requirement 5 (R5): is allocated to one identified function (settling time function and its services) for the customer-FR (Process another operation).

The measurement of the COSMIC functional size for R5 is calculated based on data movements between groups of processes and is described as follows:

- One identification function (1 ENTRY and 1 EXIT).
- One identification service (2 ENTRY and 2 EXIT).

The total functional size for R5 is 6 CFP.

Requirement 6 (R6): is allocated to one identified function (concurrency function and its services) for the customer-FUR (Inform rejection, Eject E-card, Take E-card and Capture E-card).

The measurement of the COSMIC functional size for R6 is calculated based on data movements between groups of processes and is described as follows:

- One identification function four times (4 ENTRY and 4 EXIT).
- One identification service four times (8 ENTRY and 8 EXIT).

The total functional size for R6 is 24 CFP.

Requirement 7 (R7): is allocated to one identified function (storage device time function and its services) for the customer-FUR (Register capture to local DB).

The measurement of the COSMIC functional size for this process is calculated based on data movements between groups of processes and is described as follows:

- One identification function (1 ENTRY and 1 EXIT).
- One identification service (2 ENTRY and 2 EXIT).
- Two intermediary services, one to use the local ATM database and the second to give order to storage to capture the customer smartcard (8 Entry and 8 Exit).

The total functional size for R7 is 22 CFP.

Requirement 8 (R8): Requirement 8 (R8): is allocated to one identified function (processor instruction execution function and its services) for the customer-FUR (Register transaction and Inform not enough money).

The measurement of the COSMIC functional size for R8 is calculated based on data movements between groups of processes and is described as follows:

- One identification function twice (2 ENTRY and 2 EXIT).
- One identification service twice (4 ENTRY and 4 EXIT).

The total functional size for R8 is 12 CFP.

5.10. Summary of findings

Table 1 lists the 17 software subgoals to be measured from the customer-FUR perspective. These software subgoals call eight specified functions and eight specified services processes of system performance allocated to software in R1, R2 to R8.

These are identified and presented in columns 1, 2 and 3. For each identified functional process, the description of the measured resource represents a data performance group moved by one data movement type, each one measured as 1 CFP (COSMIC function point).

The total functional size for the performance requirements R1, R2 to R8 applied to this software in a functional level is 34 CFP and in SOA context is equal to 122 CFP, independent of the languages and technologies used to implement them. In further applications, this functional size number can be used for effort estimation models and for software benchmarking.

5.11. Limitations of the illustrative example

The ATM example illustrates how the proposed approach is applicable in a relatively simple context. Future research is needed to investigate its scalability to much larger contexts, such as complex systems that perform millions of operations per second. Naturally, for such large complex systems, organizations have more resources and can dedicate much more resources to make use of this approach. In such contexts, additional research on efficiency studies would be welcome.

5.12. Practical Implications

We think that this framework can be easily used by someone knowledgeable in performance standards, COSMIC measurement and SOA architecture. Someone familiar with other environments, such as cloud computing environments, could also use and adapt this framework to a context-specific environment. More specifically, requirement specialists, software architects, performance engineers and project managers can use the proposed standards-based performance framework.

The generic approach proposed in our research program on standards-based identification, specification and measurement of system NFRs allocated to software had already influenced the COSMIC Group when it published its initial strategy on how to handle NFRs and Project Requirements from a project management perspec-

tive [52]. To facilitate industry adoption of this approach, including the performance framework presented here, we recently edited, for the COSMIC Group, a practitioner's guide on Non-Functional Requirements and their Sizing with COSMIC [53]: it includes a set of templates based on the performance framework documented in more detail in this study. Organizations with a very large NFR knowledge base as well as experts in performance issues can also compare their own practices with this framework and identify gaps within their practices; such gaps can be addressed, fully or partially, by using this framework. If their own practices are not structured and explicit, they can use the structure of this framework to structure and document their own related practices.

5.13. Threats to validity

The quality of an experimental case study is important as it can substantiate the limits of the study and identify threats to its validity, which could impact the results. The key types of validity threats as proposed by Wohlin [54] are:

Internal validity is concerned with the reliability of the results and refers to the treatment that caused the outcome [54]. There can be other uncontrollable or not measured factors that influenced the results. The internal validity threat in this example includes the change of the design process for this example. Here, we believe that if different researchers use the same method on the same example, they will get the same results. In contrast, in the absence of a full description of some parts considered in this example, variability in the results would be expected.

External validity [54] here refers to the generalization of the results outside the scope of the example and whether or not the cause and effect established hold in other situations. An external validity threat is expressed at the outcomes level. The proposed measurements framework of system performance NFR was illustrated using only the performance requirements specifications of an ATM system. There is no claim in this study on generalizability and scalability to much larger contexts, such as complex systems that perform

Table 1. ATM measurement of the system performance NFR allocated to software functions

Req. ID	Software sub-goals customer-FR & system-FR	Specified system performance NFR allocated to software functions		Data movements identified				
				E	X	R	W	Size in CFPs
R 1	Insert E-card	Bandwidth function	Bandwidth service	3	3	-	-	6
	Check if E-card valid			4	4	2	2	12
	Enter E-card PIN							
R 2	Present transaction options	Response time function	Response time service	3	3	-	-	6
	Retrieve account balance			14	14	3	3	34
	amount of money Display							
R 3	Expel money	Main memory time function	Main memory time service	1	1	-	-	2
R 4	Register transaction to local DB	System scalability function	System scalability service	2	2	-	-	4
	Register capture to local DB			12	12	-	-	24
R 5	Process another operation	Concurrency function	Concurrency service	1	1	-	-	2
				2	2	-	-	4
R 6	Inform rejection	Settling time function	Settling time service	4	4	-	-	8
	Eject E-card							
	Accept E-card			8	8	-	-	16
	Capture E-card							
R 7	Register capture to local DB	Storage device time function	Storage device time service	1	1	-	-	2
				10	10	-	-	20
R 8	Register transaction	Processor instruction execution function	Processor instruction execution service	2	2	-	-	4
	Inform not enough money			4	4	-	-	8
Total functional size of system performance requirements at functional level				17	17	0	0	34 CFP
Total functional size of system performance requirements at services level				56	56	5	5	122 CFP
Total # functional size measurement at (functional and service levels)				73	73	5	5	156 CFP

millions of operations per second. Scalability is outside the scope of the research presented here. To alleviate the risk of this validity threat additional examples and case studies could be conducted using requirement specifications of different types of software products (e.g., business application software, Telecom software, medical embedded software, etc.).

Construct validity [54] refers to the relation between the theory behind the experiment and the observation(s). The treatment and the results may not correspond to the cause and the effect controlled and measured. A limited number of standards and methods have been selected for this example. Nevertheless, there exist other standards and methods, and future research could investigate the use and relevance of such other standards and methods.

6. Conclusion and further work

This paper proposed a measurement framework of system performance NRF allocated to software functions. This work on system performance NFR extends our previous work on three types of

NFR (security, portability and maintainability) to facilitate the early identification, specification and measurement of such kinds of NFR.

The suggested framework includes some of the consensual performance terms and concepts used by two sets of international standards (ECSS and IEEE) and some related works. They were analyzed and integrated using different design views beginning with the logical view, followed by process view, development view and ending with physical views. Next, the set of ISO 19761 (COSMIC) concepts and views were adopted for describing the framework functionality at a lower level to illustrate that the proposed framework is designed for measurement purposes as well as for capture of the performance concept. The proposed framework was designed using SIGs.

This research considered system performance requirements as both static and dynamic performance types, each with its own set of candidate sub-concepts.

Additionally, for a more complete software view of a complex environment (i.e., functional services in a service-oriented architecture), COSMIC-SOA was applied to the suggested framework.

Finally, an ATM example was presented to guide developers and software engineers to use the measurement of system performance NFR allocated as performance FUR at the software level.

The main contribution of this work is its ability to assist developers, system and software engineers to specify system performance requirements early in the life cycle in order to address the specified performance functions to be allocated to software as functional requirements.

The proposed framework can also be used for identification, specification and measurement of system performance NFR using ISO 19761 independently of any programming languages, and in addition can address software performance FUR early in their implementation.

In this paper, the proposed measurement aspects addressed the system requirements allocated to software. It will be interesting in further work to extend this measurement aspect to consider other types of requirements at the system level containing hardware requirements.

Some related issues were not addressed and additional work is required such as using these functional size measurement results of system performance requirements allocated to software-FURs as new input for estimating models in software engineering projects. Additional empirical work is required to verify that such expanded size can improve the estimation models, including for testing and maintenance effort.

The ATM illustrative example showed that the proposed measurement framework can help to specify and measure the functional size of system performance-NFR allocated to software functions. Consequently, this may improve planning, managing and development of software at different phases of the software development life cycle. Furthermore, the measurement results of the proposed framework may be used in benchmarking studies.

This example was not built to learn but to demonstrate that the framework was usable, and the purpose in building this example was not to evaluate the framework. This again, would require much more empirical work with practi-

tioners to evaluate it in a number of contexts, and with a set of criteria for evaluation.

This proposed measurement framework is limited to measuring the system performance requirements allocated to software at the functional and service levels. It will be interesting in further work to focus on its applicability to different types of software products in order to generalize the results reported in this illustrative example. In addition, further work could focus on automating the measurement of software performance requirements through building an automated measurement tool (or enhancing an existing one).

Future work on the scalability of the framework proposed in this study would be valuable for industrial research where researchers look at such practical scalability issues, with financial resources much larger than the resources available to university researchers.

References

- [1] K.T. Al-Sarayreh, "Model of early specifications of performance requirements at functional levels," *Recent Advances on Electrosience and Computers*, 2015, p. 236.
- [2] K. Meridji, K.T. Al-Sarayreh, A. Abran, and S. Trudel, "System security requirements: A framework for early identification, specification and measurement of related software requirements," *Computer Standards and Interfaces*, Vol. 66, 2019, p. 103346.
- [3] A. Abran, K.T. Al-Sarayreh, and J.J. Cuadrado-Gallego, "A standards-based reference framework for system portability requirements," *Comput. Stand. Interfaces*, Vol. 35, No. 4, Jun. 2013, p. 380–395. [Online]. <https://doi.org/10.1016/j.csi.2012.11.003>
- [4] K.T. Al-Sarayreh, A. Abran, and J.J. Cuadrado-Gallego, "A standards-based model of system maintainability requirements," *journal of software: evolution and process*, Vol. 25, No. 5, 2013, pp. 459–505.
- [5] K.T. Al-Sarayreh, A. Abran, and J.J. Cuadrado-Gallego, "Measurement model of software requirements derived from system portability requirements," in *9th International Conference on Software Engineering Research and Practice (SERP 2010)*, 2010, pp. 553–559.

- [6] K.T. Al-Sarayreh, I. Al-Oqily, and K. Meridji, "A standard based reference framework for system adaptation and installation requirements," in *2012 Sixth International Conference on Next Generation Mobile Applications, Services and Technologies*, 2012, pp. 7–12.
- [7] M. Kassab, O. Ormandjieva, and M. Daneva, "An ontology based approach to non-functional requirements conceptualization," in *2009 Fourth International Conference on Software Engineering Advances*. IEEE, 2009, pp. 299–308.
- [8] X. Zhang and X. Wang, "Tradeoff analysis for conflicting software non-functional requirements," *IEEE Access*, Vol. 7, 2019, pp. 156 463–156 475.
- [9] M. Dewi and Z. Didar, "An ontological framework to manage the relative conflicts between security and usability requirements," in *2010 Third International Workshop on Managing Requirements Knowledge*. IEEE, 2010, pp. 1–6.
- [10] N. Daclin, B. Moradi, and V. Chapurlat, "Analyzing interoperability in a non-functional requirements ecosystem to support crisis management response," *Enterprise Interoperability: Smart Services and Business Impact of Enterprise Interoperability*, 2018, pp. 429–434.
- [11] L.M. Cysneiros, K.K. Breitman, C. Lopez, and H. Astudillo, "Querying software interdependence graphs," in *2008 32nd Annual IEEE Software Engineering Workshop*. IEEE, 2008, pp. 108–112.
- [12] K.T. Al-Sarayreh, "Dependability model for decomposition and allocation of system safety integrity levels of software quality," *International Review on Computers and Software*, Vol. 10, No. 11, 2015.
- [13] S. Al-Qudah, K. Meridji, and K.T. Al-Sarayreh, "A comprehensive survey of software development cost estimation studies," in *Proceedings of the international conference on intelligent information processing, security and advanced communication*, 2015, pp. 1–5.
- [14] R.E. Al-Qutaish and K.T. Al-Sarayreh, "Software process and product ISO standards: a comprehensive survey," *European Journal of Scientific Research*, Vol. 19, No. 2, 2008, pp. 289–303.
- [15] A. Abran and K.T. Al-Sarayreh, "Standards-based model for the specification of system design and implementation constraints," in *Industrial Proceedings, 17th European Systems and Software Process Improvement and Innovation, EuroSPI 2010 Conference*. Publisher: Delta, Denmark Grenoble (France), 2010, pp. 4–7.
- [16] R. Tawhid and D. Petriu, "Automatic derivation of a product performance model from a software product line model," in *2011 15th International Software Product Line Conference*. IEEE, 2011, pp. 80–89.
- [17] M. Noorian, E. Bagheri, and W. Du, "Non-functional properties in software product lines: A axonomy for classification." in *SEKE*, Vol. 12, 2012, pp. 663–667.
- [18] A. Danylenko and W. Löwe, "Context-aware recommender systems for non-functional requirements," in *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*. IEEE, 2012, pp. 80–84.
- [19] H.T. Jung and G.H. Lee, "A systematic software development process for non-functional requirements," in *2010 International conference on information and communication technology convergence (ICTC)*. IEEE, 2010, pp. 431–436.
- [20] *Gyro terminology and performance specification*, European Cooperation for Space Standardization Std. ECSS-E-ST-60-21C, 2017.
- [21] *System engineering general requirements*, European Cooperation for Space Standardization Std. ECSS-E-ST-10C Rev. 1, 2017.
- [22] *Software product assurance*, European Cooperation for Space Standardization Std. ECSS-Q-ST-80C Rev.1, 2017.
- [23] *Space Engineering: Control Performance*, European Cooperation for Space Standardization Std. ECSS-E-ST-60-10C, 2008.
- [24] *Space engineering, Stars sensors terminology and performance specification*, European Cooperation for Space Standardization Std. ECSS-E-ST-60-20C Rev.2 DIR1, 2017.
- [25] *Satellite attitude and orbit control system (AOCS) requirements*, European Cooperation for Space Standardization Std. ECSS-E-ST-60-30C:, 2013.
- [26] *Recommended Practice for Software Requirements Specifications*, Institute of Electrical and Electronics Engineers Std. 830-1998, 1998.
- [27] L. Mo, S. Zhigang, H. Quan, Y. Guizhi, L. Ya, and S. Fengli, "Analysis and testing of key performance indexes of vxworks in real-time system," in *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2018, pp. 369–374.
- [28] A. Ahmad, N. Abdulrahman, B. Sascha, J. Naoum, and T. Klaus, "Toward a performance requirements model for the early design phase of IT systems," in *2018 Sixth International Conference on Enterprise Systems (ES)*. IEEE, 2018, pp. 9–16.
- [29] Z. Chen, T. Zhao, J. Jiao, and H. Wu, "Availability analysis of multi-state weighted k -out-of- n

- systems with component performance requirements,” in *2018 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE, 2018, pp. 1–5.
- [30] K.T. Al-Sarayreh, I. Ibrahim Al-Oqily, and K. Meridji, “A standard-based reference framework for system operations requirements,” *International Journal of Computer Applications in Technology*, Vol. 47, No. 4, 2013, pp. 351–363.
- [31] J. Kai, X. Ling, and Z. Huamin, “A parameter tuning method of proportional integral controller for the first-order plus delay time system based on the desired dynamical performance,” in *Proceeding of the 11th World Congress on Intelligent Control and Automation*. IEEE, 2014, pp. 6110–6115.
- [32] K. Arun and A. Sunil, “Statistical analysis of memory and performance non functional requirements in real time embedded system development for agile methodology,” in *2015 International Conference on Industrial Instrumentation and Control (ICIC)*. IEEE, 2015, pp. 300–305.
- [33] I. Vila, J. Perez-Romero, O. Sallent, A. Umberto, and R. Ferrus, “Performance measurements-based estimation of radio resource requirements for slice admission control,” in *90th Vehicular Technology Conference (VTC2019-Fall)*. IEEE, 2019, pp. 1–6.
- [34] M. Anish, R. Anand, R. Srivaths, and J. Niraj, “Automated energy/performance macromodeling of embedded software,” *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 26, No. 3, 2007, pp. 542–552.
- [35] *Software measurement – Functional size measurement Part 1: Definition of concepts*, ISO/IEC Std. 14143-1, 1998.
- [36] *COSMIC v 3.0 – A Functional Size Measurement Method, I*, ISO/IEC Std. 19761, 2003.
- [37] P. Fagg, A. Lesterhuis, G. Rule, G. Ungerer, K. Galegaonkar, S. and Natarajan, L. Santillo, F. Vogelesang, P. Jain, M. O’Neill, and C. Symons, “Guideline for sizing SOA software,” The Common Software Measurement International Consortium (COSMIC), Tech. Rep., 2010.
- [38] L. Santillo, “Seizing and sizing SOA applications with COSMIC function points,” *Proceedings of SMEF*, 2007.
- [39] E. Marks, *Service-oriented architecture governance for the services driven enterprise*. John Wiley and Sons, 2008.
- [40] L. Chung, B. Nixon, and E. Yu, “Dealing with change: An approach using non-functional requirements,” *Requirements Engineering*, Vol. 1, No. 4, 1996, pp. 238–260.
- [41] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer Science and Business Media, 2012, Vol. 5.
- [42] K. Hemenway, M. Iff, and T. Calishain, *Spidering Hacks: 100 Industrial-Strength Tips and Tools*. " O’Reilly Media, Inc.", 2004.
- [43] I. Lee, J. Leung, and S. Son, *Handbook of real-time and embedded systems*. CRC Press, 2007.
- [44] K.T. Al-Sarayreh, L.A. Hasan, and K. Almakadmeh, “A trade-off model of software requirements for balancing between security and usability issues,” *International Review on Computers and Software*, Vol. 10, No. 12, 2016, pp. 1157–1168.
- [45] C.M. Kozierok, *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference*. No Starch Press, 2005.
- [46] A. Abran and K. Meridji, “Analysis of software engineering from an engineering perspective,” *European Journal for the Informatics Professional*, Vol. 7, No. 1, 2006, pp. 46–52.
- [47] K. Meridji, K.T. Al-Sarayreh, and A. Al-Khasawneh, “A generic model for the specification of software reliability requirements and measurement of their functional size,” *International Journal of Information Quality*, Vol. 3, No. 2, 2013, pp. 139–163.
- [48] J. Carr, *The technician’s radio receiver handbook: wireless and telecommunication technology*. Elsevier, 2001.
- [49] J.J. Parsons and D. Oja, *New Perspectives on Computer Concepts 2014: Comprehensive*. Cengage Learning, 2013.
- [50] B. Parkinson and J. Spilker, “Progress in astronautics and aeronautics: Global positioning system: Theory and applications. american institute of aeronautics/astronautics,” 1996.
- [51] H. El-Rewini and M. Abd-El-Barr, *Advanced computer architecture and parallel processing*. John Wiley and Sons, 2005, Vol. 42.
- [52] K.T. Al-Sarayreh and A. Abran, “Specification and measurement of system configuration non functional requirements,” in *20th International Workshop on Software Measurement and International Conference on Software Measurement, IWSM/Metrikon/Mensura, Stuttgart, Germany*, 2010, pp. 141–156.
- [53] A. Abran and K.T. Al-Sarayreh, “Non-functional requirements and their sizing with cosmic: Practitioner’s guide,” in *COSMIC Group*, 2020, pp. 1–14.
- [54] W. Claes, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in software engineering*. Springer Science and Business Media, 2012.

e-Informatica Software Engineering Journal (eISEJ) is an international, open access, no authorship fees, blind peer-reviewed journal that concerns theoretical and practical issues pertaining development of software systems. Our aim is to focus on experimentation and machine learning in software engineering.

The journal is published under the auspices of the Software Engineering Section of the Committee on Informatics of the Polish Academy of Sciences and Wrocław University of Science and Technology.

Aims and Scope:

The purpose of **e-Informatica Software Engineering Journal** is to publish original and significant results in all areas of software engineering research.

The scope of **e-Informatica Software Engineering Journal** includes methodologies, practices, architectures, technologies and tools used in processes along the software development lifecycle, but particular stress is laid on empirical evaluation.

e-Informatica Software Engineering Journal is published online and in hard copy form. The on-line version is from the beginning published as a gratis, no authorship fees, open access journal, which means it is available at no charge to the public. The printed version of the journal is the primary (reference) one.

Topics of interest include, but are not restricted to:

- Software requirements engineering and modeling
- Software architectures and design
- Software components and reuse
- Software testing, analysis and verification
- Agile software development methodologies and practices
- Model driven development
- Software quality
- Software measurement and metrics
- Reverse engineering and software maintenance
- Empirical and experimental studies in software engineering (incl. replications)
- Evidence based software engineering
- Systematic reviews and mapping studies
- Meta-analyses
- Object-oriented software development
- Aspect-oriented software development
- Software tools, containers, frameworks and development environments
- Formal methods in software engineering.
- Internet software systems development
- Dependability of software systems
- Human-computer interaction
- AI and knowledge based software engineering
- Data mining in software engineering
- Prediction models in software engineering
- Mining software repositories
- Search-based software engineering
- Multiobjective evolutionary algorithms
- Tools for software researchers or practitioners
- Project management
- Software products and process improvement and measurement programs
- Process maturity models

Important information: Papers can be rejected administratively without undergoing review for a variety reasons, such as being out of scope, being badly presented to such an extent as to prevent review, missing some fundamental components of research such as the articulation of a research problem, a clear statement of the contribution and research methods via a **structured abstract** or the evaluation of the proposed solution (empirical evaluation is strongly suggested).

Funding acknowledgements: Authors are requested to identify who provided financial support for the conduct of the research and/or preparation of the article and to briefly describe the role of the sponsor(s), if any, in study design; in the collection, analysis and interpretation of data; in the writing of the paper. If the funding source(s) had no such involvement then this should be stated as well.

The submissions will be accepted for publication on the base of positive reviews done by international Editorial Board (<https://www.e-informatyka.pl/index.php/einformatica/editorial-board/>) and external reviewers. English is the only accepted publication language. To submit an article please enter our online paper submission site (<https://mc.manuscriptcentral.com/e-InformaticaSEJ>).

Subsequent issues of the journal will appear continuously according to the reviewed and accepted submissions.

<http://www.e-informatyka.pl/wiki/e-Informatica>



e-Informatica

ISSN 1897-7979