Information and Software Technology xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Obsolete software requirements

Krzysztof Wnuk [a,*], Tony Gorschek [b,1], Showayb Zahda [c,2]

[a] Department of Computer Science, Lund University, Ole Römers väg 3, SE-223 63 Lund, Sweden
[b] School of Computing Software Engineering Research Lab, Blekinge Institute of Technology, SE-371 79 Karlskrona, Sweden
[c] Axis Communications AB, Emdalavägen 14, SE-223 69 Lund, Sweden

ARTICLE INFO

Article history:
Received 6 January 2012
Received in revised form 9 October 2012
Accepted 3 December 2012
Available online xxxx

Keywords:
Requirements management
Obsolete requirements
Survey
Empirical study

ABSTRACT

Context: Coping with rapid requirements change is crucial for staying competitive in the software business. Frequently changing customer needs and fierce competition are typical drivers of rapid requirements evolution resulting in requirements obsolescence even before project completion.
Objective: Although the obsolete requirements phenomenon and the implications of not addressing them are known, there is a lack of empirical research dedicated to understanding the nature of obsolete software requirements and their role in requirements management.
Method: In this paper, we report results from an empirical investigation with 219 respondents aimed at investigating the phenomenon of obsolete software requirements.
Results: Our results contain, but are not limited to, defining the phenomenon of obsolete software requirements, investigating how they are handled in industry today and their potential impact.
Conclusion: We conclude that obsolete software requirements constitute a significant challenge for companies developing software intensive products, in particular in large projects, and that companies rarely have processes for handling obsolete software requirements. Further, our results call for future research in creating automated methods for obsolete software requirements identification and management, methods that could enable efficient obsolete software requirements management in large projects.

1. Introduction

Software, as a business, is a demanding environment where a growing number of users, rapid introduction of new technologies, and fierce competition are inevitable [1–3]. This rapidly changing business environment is challenging traditional Requirements Engineering (RE) approaches [4–6]. The major challenges in this environment are high volatility and quick evolution of requirements, requirements that often tend to become obsolete even before project completion [1,7–9]. At the same time the product release time is crucial [10–12] for the success of the software products, especially in emerging or rapidly changing markets [10].

Coping with rapid requirements change is crucial as time-to-market pressures often make early pre-defined requirements specifications inappropriate almost immediately after their creation [7]. In Market-Driven Requirements Engineering (MDRE), the pace of incoming requirements [2] and requirements change is high. Software companies have to identify which requirements are obsolete or outdated. The rapid identification and handling of potentially obsolete requirements is important as large volumes of degrading requirements threatens effective requirements management. In extreme cases, obsolete requirements could dramatically extend project timelines, increase the total cost of the project or even cause project failure; and even the successful identification of the obsolete requirements without handling adds little or no product value [13–15]. Thus, the identification, handling, and removal of obsolete requirements is crucial.

The phenomenon of obsolete requirements and the implications of not handling them are known [16,13,14,17–20]. At the same time, several researchers focused on topics related to the phenomenon of obsolete requirements, e.g. requirements volatility and scope creep [21–27]. However, very little research has been performed into obsolete requirements management or guidelines, see e.g. [28–33]. Standards [34,35] do not explicitly mention the phenomenon of Obsolete Software Requirements (OSRs). The term itself is only partly defined and empirically anchored [17].

In this paper, we present the results from an empirical study, based on a survey with 219 respondents from different companies. The survey investigated the phenomenon of obsolete requirements and included, an effort to define the phenomenon based on the perceptions of industry practitioners. The study also aimed to collect data on how obsolete requirements are perceived, how they impact industry, and how they are handled in industry today.

* Corresponding author. Tel.: +46 46 222 45 17; fax: +46 46 13 10 21.
E-mail addresses: Krzysztof.Wnuk@cs.lth.se (K. Wnuk), Tony.Gorschek@bth.se (T. Gorschek), Showayb.Zahda@axis.com (S. Zahda).
[1] Tel.: +46 455 38 58 17; fax: +46 455 38 50 57.
[2] Tel.: +46 46 272 18 00; fax: +46 46 13 61 30.

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.12.001

This paper is structured as follows: Section 2 provides the background and related work, Section 3 describes the research methodology, Section 4 describes and discusses the results of the study, and Section 5 concludes the paper.

2. Background and related work

Requirements management, as an integral part of requirements engineering [9,31], manages the data created in the requirements elicitation and development phases of the project. Requirements management integrates this data into the overall project flow [9] and supports the later lifecycle modification of the requirements [9]. As changes occur during the entire software project lifetime [36], managing changes to the requirements is a major concern of requirements management [29,30] for large software systems. Moreover, in contexts like MDRE, a constant stream of new requirements and change requests is inevitable [2]. Uncontrolled changes to software may cause the cost of the regression testing to exceed 100,000 dollars [9]. Further, the absence of requirements management may sooner or later cause outdated requirements specifications as the information about changes to original requirements is not fed back to the requirements engineers [9]. Finally, the requirements management process descriptions in literature seldom consider managing obsolete requirements [29,28].

Scope creep, requirements creep and requirements leakage (also referred to as uncontrolled requirements creep) [21,22] are related to OSRs. DeMarco and Lister identified scope creep as one of the five core risks during the requirements phase and state that the risk is a direct indictment of how requirements were gathered in the first place [23]. Scope creep has also been mentioned as having a significant impact on risk and risk management in enterprise data warehouse projects [37]. Houston et al. [24] studied software development risk factors and 60% of 458 respondents perceived that requirements creep was a problem in their projects. Anthes [38] reported that the top reason for requirements creep in 44% of the cases is a poor definition of initial requirements. Scope creep can lead to significant scope reductions as overcommitment challenges are addressed. This, in turn, postpones the implementation of the planned functionality and can cause requirements to become obsolete [8] or project failure [22].

Despite its importance as a concept, in relation to managing requirements for software products, the phenomenon of OSRs seems to be underrepresented in literature. To the best of our knowledge, only a handful of articles and books mention the terms obsolete requirements or/and obsolete features. Among the existing evidence, Loesch and Ploederoeder [18] claim that the explosion of the number of variable features and variants in a software product line context is partially caused by the fact that obsolete variable features are not removed. Murphy and Rooney [13] stress that requirements have 'a shelf life' and suggest that the longer it takes from defining requirements to implementation, the higher the risk of change (this inflexibility is also mentioned by Ruel et al. [39]). Moreover, they state that change makes requirements obsolete, and that obsolete requirements can dramatically extend project timelines and increase the total cost of the project. Similarly, Stephen et al. [14] list obsolete requirements as one of the symptoms of failure of IT project for the UK government. While the report does not define obsolete requirements *per se*, the symptom of failure is ascribed to obsolete requirements caused by the inability to unlock the potential of new technologies by timely adoption.

The phenomenon of OSRs has not yet been mentioned by standardization bodies in software engineering. Neither the IEEE 830 standard [34] nor CMMI (v.1.3) [35] mention obsolete software requirements as a phenomenon. Actions, processes and techniques are also not suggested in relation to handling the complexity. On the other hand, Savolainen et al. [17] propose a classification of atomic product line requirements into these categories: non-reusable, mandatory, variable and obsolete. Moreover they propose a short definition of obsolete requirements and the process of managing these requirements for software product lines "by marking them obsolete and hence not available for selection into subsequent systems". Mannion et al. [19] propose a category of variable requirements called obsolete and suggest dealing with them as described by Savolainen et al. [17].

OSRs are related to the concept of requirements volatility. SWEBOOK classifies requirements into a number of dimensions and one of the them is volatility and stability. SWEBOK mentions that some volatile requirements may become obsolete [40]. Kulk and Verhoef [25] reported that the maximum requirements volatility rates depend on size and duration of a project. They proposed a model that calculates the "maximum healthy volatility ratios" for projects. Loconsole and Börstler [27] analyzed requirements volatility by looking at the changes to use case models while Takahashi and Kamayachi [41] investigated the relationship between requirements volatility and defect density. On the other hand, Zowghi and Nurmuliani [26] proposed a taxonomy of requirement changes where one of the reasons for requirements changes is obsolete functionality, defined as "functionality that is no longer required for the current release or has no value for the potential users". For this paper, we understand requirements volatility as a factor that influences requirements change but different from requirements obsolescence. OSRs are, according to our understanding, any type of requirement (stable, small, large, changing) that is not realized or dismissed, but which accumulates in the companies' databases and repositories. Requirements obsolescence is defined as a situation where volatility becomes outdated and remains in the requirements databases [42,43].

Looking at previous work, software artifact obsolescence has been mentioned in the context of obsolete hardware and electronics in, for example, military, avionics or other industries. Among others, Herald et al. proposed an obsolescence management framework for system components (in this case hardware, software, and constraints) that is mainly concerned with system design and evolution phases [20]. While, the framework contains a technology roadmapping component, it does not explicitly mention OSRs. Merola [15] described the software obsolescence problem in today's defense systems of systems (the COTS software components level). He stressed that even though the issue has been recognized as being of equal gravity to the hardware obsolescence issue, it has not reached the same visibility level. Merola outlines some options for managing software obsolescence, such as negotiating with the vendor to downgrade the software license, using wrappers and software application programming interfaces, or performing market analysis and surveys of software vendors.

Due to the limited number of studies in the literature dedicated to the OSR phenomenon, we decided to investigate the concept utilizing a survey research strategy. We investigated the extent to which obsolete software requirements are perceived as a real phenomenon and as a real problem in industry. Moreover, we investigated how OSRs are identified and managed in practice, and what contextual factors influence OSRs.

3. Research methodology

This section covers the research questions, the research methodology, and the data collection methods used in the study.

Table 1
Research questions.

Research question	Aim	Example answer
RQ1: Based on empirical data, what would be an appropriate definition of Obsolete Software Requirements (OSR)?	Instead of defining the phenomenon ourselves we base the definition on how the phenomenon is perceived in industry	"An obsolete software requirements is a requirement that has not been included into the scope of the project for the last 5 projects"
RQ2: What is the impact of the phenomenon of obsolete software requirements on the industry practice?	To investigate to what degree is OSR a serious **concern**	"Yes it is somehow serious"
RQ3: Does requirement type affect the likelihood of a software requirement becoming obsolete?	Are there certain types of requirements that become obsolete more often than others? Can these types be identified?	"A market requirement will become obsolete much faster than a legal requirement"
RQ4: What methods exist, in industry practice, that help to identify obsolete software requirements?	To enact a process to detect, identify or find obsolete software requirements or nominate requirements that risk becoming obsolete	"To read the requirements specification carefully and check if any requirements are obsolete"
RQ5: When OSRs are identified, how are they typically handled in industry?	In order to identify possible alternatives for OSR handling, we first need to understand how they are handled today	"We should mark found obsolete requirements as obsolete but keep them in the requirements database"
RQ6: What context factors, such as project size or domain, influence OSRs?	As a step in understanding and devising solutions for handling OSRs, it is important to identify contextual factors that have an influence on the obsolete requirements phenomenon	"OSRs are more common in large projects and for products that are sold to an open market (MDRE context)"
RQ7: Where in the requirements life cycle should OSRs be handled?	To position requirements obsolescence in the requirements engineering life cycle	"They should be a part of the requirements traceability task"

3.1. Research questions

Due to the limited number of related empirical studies identified in relation to OSRs, we decided to focus on understanding the OSR phenomenon and its place in the requirements engineering landscape. Thus, most of the research questions outlined in Table 1 are existence, descriptive, as well as classification questions [44]. Throughout the research questions, we have used the following definition of OSRs, based on the literature study and the survey:

"An obsolete software requirement is a software requirement, implemented or not, that is no longer required for the current release or future releases, and which has no value or business goals for the potential customers or users of a software artifact for various reasons.[3]"

3.2. Research design

A survey was chosen as the main tool to collect empirical data, enabling us to reach a larger number of respondents from geographically diverse locations [45]. Automation of data collection and analysis ensured flexibility and convenience to both researchers and participants [44,46,47].

The goal of the survey was to elicit as much information from industry practitioners as possible in relation to OSRs. Therefore, we opted for an inclusive approach to catch as many answers as possible. This prompted the use of convenience sampling [47]. The details in relation to survey design and data collection are outlined below.

3.2.1. Survey design

The questionnaire was created based on a literature review of relevant topics, such as requirements management, volatility, and requirements traceability (see Section 2). The questions were iteratively developed. Each version of the questionnaire was discussed among the authors and evaluated in relation to how well the questions reflected the research questions and the research goals.

[3] For reader convenience we present the definition in this section, rather than after presentation of the results. The description of how the definition was derived is available in Section 4.

The questionnaire contained 15 open and close-ended questions of different formats, e.g. single choice questions and multiple choice questions. In open-ended questions, respondents could provide their own answers as well as select a pre-defined answer from the list. The answers were analyzed using the open coding method [48]. The data analysis was started without a preconceived theory in mind. We read all the answers and coded interesting answers by assigning them to a category with similar meaning. For close-ended questions, we used a Likert scale from 1 to 5, where 1 corresponds to *Not likely* and 5 to *Very likely* [49].

The questionnaire was divided into two parts: one related to OSRs (9 questions), and one related to demographics (6 questions). Table 2 shows the survey questions, with a short description of their purpose (2nd column), the list of relevant references (3rd column), and a link to the addressed research question (4th column). It should be observed that an OSR is defined in this work in the context of the current release in order to keep the question fairly simple and avoid introducing other complicating aspects. For reasons of brevity, we do not present the entire survey in the paper. However, the complete questionnaire, including the references that were used to construct the categories for the answers is available online [50].

3.2.2. Operation (execution of the survey)

The survey was conducted using a web-survey support website called SurveyMonkey [52]. Invitations to participate in the questionnaire were sent to the potential audience via:

- Personal emails—utilizing the contact networks of the authors
- Social network websites [53]—placing the link to the questionnaire on the board of SE and RE groups and contacting individuals from the groups based on their designated titles such as senior software engineer, requirements engineer, system analyst, and project manager to name a few
- Mailing lists—requirements engineering and software engineering discussion groups [54]
- Software companies and requirements management tool vendors [55]

Masters and undergraduate students were excluded as potential respondents because their experience was judged insufficient to answer the questionnaire. The questionnaire was published online on the 3rd of April, 2011 and the data collection phase

Table 2
Mapping between the questionnaire questions and the research questions.

Question	Purpose	Relevant references	RQ
Q1	To derive the definition of Obsolete Software Requirements	[17,20,26,15]	RQ1
Q2	To investigate the impact of the OSRs on industry practice	[13,14]	RQ2
Q3	To investigate how likely the various types of requirements would become obsolete	The list of requirements types was derived from analyzing several requirements classifications [42,43]	RQ3
Q4	To investigate the possible ways of identifying OSR in the requirements documents	[18,20]	RQ4
Q5	To investigate the possible actions to be taken against obsolete requirements after they are discovered	[18,20]	RQ5
Q6	To investigate whether there is a correlation between project size and the effects of OSRs	The classification of different sizes of requirements engineering was adopted from Regnell et al. [51]	RQ6
Q7	To investigate if OSRs are related to the software context	[14]	RQ6
Q8	To understand where in the requirements life cycle OSRs should be handled	Current standards for requirements engineering and process models [34,35] do not consider obsolete requirements	RQ5, partly RQ7
Q9	To investigate if industry has processes for managing OSR	[18,17]	RQ5

ended on the 3rd of May, 2011. In total, approximately 1700 individual invitations were sent out with 219 completed responses collected. The response rate, around 8%, is an expected level [44,45]. The results of the survey are presented in Section 4.

3.3. Validity

In this section, we discuss the threats to validity in relation to the research design and data collection phases. The four perspectives of validity discussed in this section are based on the classification proposed by Wohlin et al. [56].

3.3.1. Construct validity

Construct validity concerns the relationship between the observations from the study and the theories behind the research. The phrasing of questions is a threat to construct validity. The authors of this paper and an independent native English speaker and writer–reviewer revised the questionnaire to alleviate this threat. To minimize the risk of misunderstanding or misinterpreting the survey questions, a pilot study was conducted on master students in software engineering. The pilot study clearly indicated that a shorter list of categories is more preferable than a more extensive one. No participant in the pilot study indicated that the requirements categories in question 3 [50] were hard to understand or vague. However, the choice of the categories used in the paper for eliciting information from practitioners remains a threat to construct validity. There is always a threat that the categories are too simple, too few, too complex or too many. The choices we made, see Section 4.4, are based on keeping it as simple as possible and were derived after reviewing several classifications and a pilot study.

The reader should keep in mind that the data given by respondents is not based on any objective measurements and thus its subjectivity affects the interpretation of the results. The mono-operational bias [56] threat to construct validity is addressed by collecting data from more than 200 respondents from 45 countries. Finally, the mono-method bias [56] threat to construct validity was partly addressed by analyzing related publications. While several related publications have been identified (see Section 2), this threat is not fully alleviated and requires further studies. Finally, considering social threats to construct validity it is important to mention the evaluation apprehension threat [56]. The respondents' anonymity was guaranteed.

Some may argue that using the same questionnaire to define the term and to investigate it threatens construct validity. However, the fact that the presented OSR definition is based on over 50% of the answers and that the definition turned out to be independent of the respondents' roles, the size of the organizations, the length of the typical project, the domain and the development methodologies used gives us the basis to state that the understanding of the measured phenomenon was rather homogeneous among the respondents (Section 4.2). In addition, we do gain one aspect by combining the two, namely the subject's interpretation/understanding of what an obsolete requirement is. We are able to identify if respondents disagree, a fact which is essential for combining results (several respondents answers) for analysis.

3.3.2. Conclusion validity

Conclusion validity is concerned with the ability to draw correct conclusions from the study. To address the measures reliability threat, the questions used in the study were reviewed by the authors of this paper and one external reviewer, a native English speaker. The low statistical power threat [56] was addressed by using as suitable statistical tests as was possible on the given type of data. Before running the tests, we tested if assumptions of the statistical tests were not violated. However, since multiple tests were conducted on the same data, the risk of type-I error increases and using, for example, the Bonferroni correction should be discussed here. Since the correction was criticized by a number of authors [57,58] it remains an open question if it should be used. Therefore, we report the *p*-values of all performed tests in case the readers want to evaluate the results using the Bonferroni correction or other adjustment techniques [57]. Finally, the random heterogeneity of subjects [56] threat should be mentioned here as this aspect was only partly controlled. However, low heterogeneity of subjects allows us to state conclusions of a greater external validity.

3.3.3. Internal validity

Internal validity threats are related to factors that affect the causal relationship between the treatment and the outcome. Reviews of the questionnaire and the pilot study addressed the instrumentation threat [56] to internal validity. The maturation threat to internal validity was alleviated by measuring the time needed to participate in the survey in the pilot study (15 min). The selection bias threat to internal validity is relevant as non-random sampling was used. Since the respondents were volunteers, their performance may vary from the performance of the whole population [56]. However, the fact that 219 participants from 45 countries with different experience and industrial roles answered the survey minimizes the effect of this threat. Finally, the level of education in development processes and methodologies may have impacted the results from the survey. It remains future work to investigate whether this factor impacts the results. However, as the survey participants are professionals (many of whom work in large successful companies) their education might not be the main

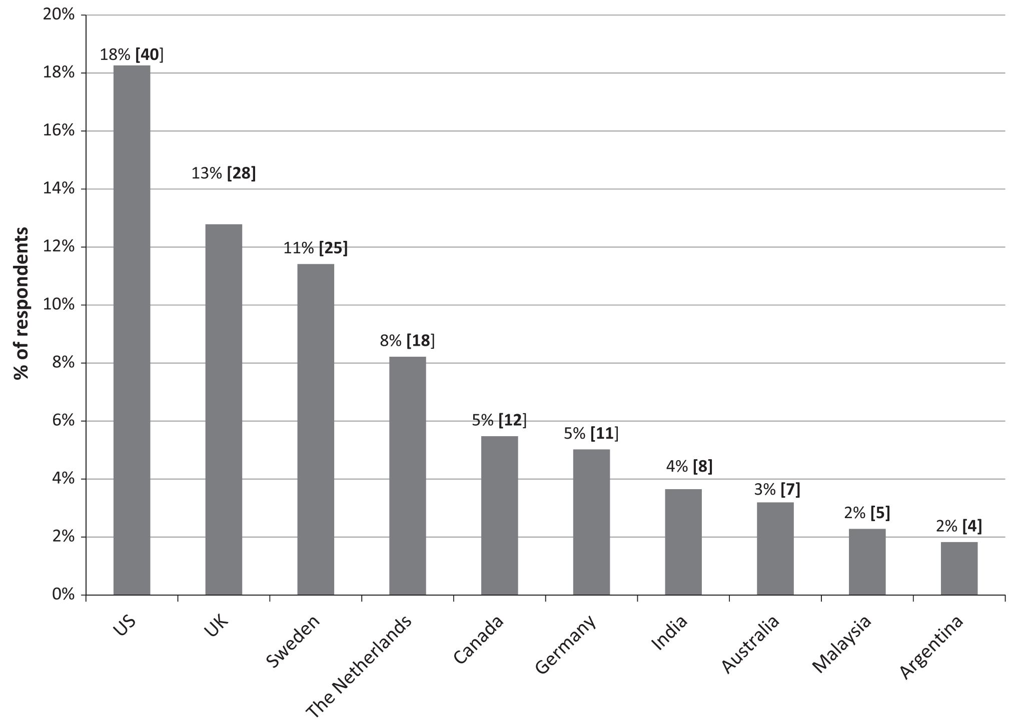

Fig. 1. Top 10 countries among respondents.

issue for discussion. What is interesting, however, is that we are investigating the state of current industry practice and not how it might be. Education is a powerful tool, but not the focus of this paper.

3.3.4. External validity

External validity threats concern the ability to generalize the result of research efforts to industrial practice [56]. The survey research method was selected to assure as many responses as possible, generating more general results [44,59,47] than a qualitative interview study. Moreover, the large number of respondents from various countries, contexts, and professions contributes to the generalization of results.

4. Results and analysis

The survey was answered by 219 respondents. When questions allowed multiple answers, we calculated the results over the total number of answers, not respondents. For questions that used a Likert scale, we present the results using average rating and the percentage received by each answer on the scale. All results are presented in percentage form and complemented by the number of answers or respondents when relevant. The answers given to the open questions were analyzed using the open coding method [48] (Section 3.2.1). Statistical analysis, when relevant, was performed using the chi-square test [60], and the complete results from the analysis, including contingency tables for some of the answers [61], are listed online.

4.1. Demographics

Fig. 1 depicts the top 10 respondent countries (out of 45).[4] The full list of the countries is available in [62]. The US and the UK constitute about 30% of the total respondents and 54% of the respondents came from Europe.

Fig. 2 depicts the main roles of the respondents in their organizations. About one quarter of the respondents (24.9% or 54 respondents) described their role as requirements engineers, analysts or coordinators. The second largest category, *Other* (with 30 answers), include roles such as *System Engineers, Software Quality Assurance, Process Engineers, and Business Analysts*. The third largest category was *Researchers* or *Academics* (11.5% of all answers). *Software Project Managers* and *Software Architect or Designer* roles had the same number of respondents (22 each). Twelve respondents declared their main role as *Software Product Manager*, a relatively high number since product managers are generally few in an organization. This would seem to indicate that middle and senior managers overall represented a substantial part of the respondents.

Fig. 3 gives an overview of the business domain of the respondents. A total of 32.8% stated the *IT or Computer and Software Services*. The second largest group (12.5%) is *Engineering* (automotive, aerospace and energy). These were followed by *Telecommunication* (10.7%) and *Consultancy* (9.3%).

Fig. 4 depicts the sizes of the respondents' organizations. We can see that more than half of the respondents work in large companies (>501 employees).

Fig. 5 looks at the average duration of a typical project in the respondents' organizations. About half of the respondents (∼45%) were involved in projects that lasted for less than a year, one quarter in projects that lasted between 1 and 2 years and one quarter in projects typically lasting more than 2 years.

Fig. 6 investigates the development methodologies and processes used by the respondents. Since this question allowed for the possibility of providing multiple answers, the results are based on the number of responses. Agile development tops the list of answers with approximately a quarter (23.6%). Incremental and evolutionary methodology (18.8%) is in second place. Surprisingly, waterfall is still common and widely used (17.7%). In the *Other* category, the respondents reported that they mixed several methodologies "combination of agile and incremental" or "it is a

[4] The actual category names have been changed for readability purposes. The original names are mentioned using *italics* in the paper and are available in the survey questionnaire [50].

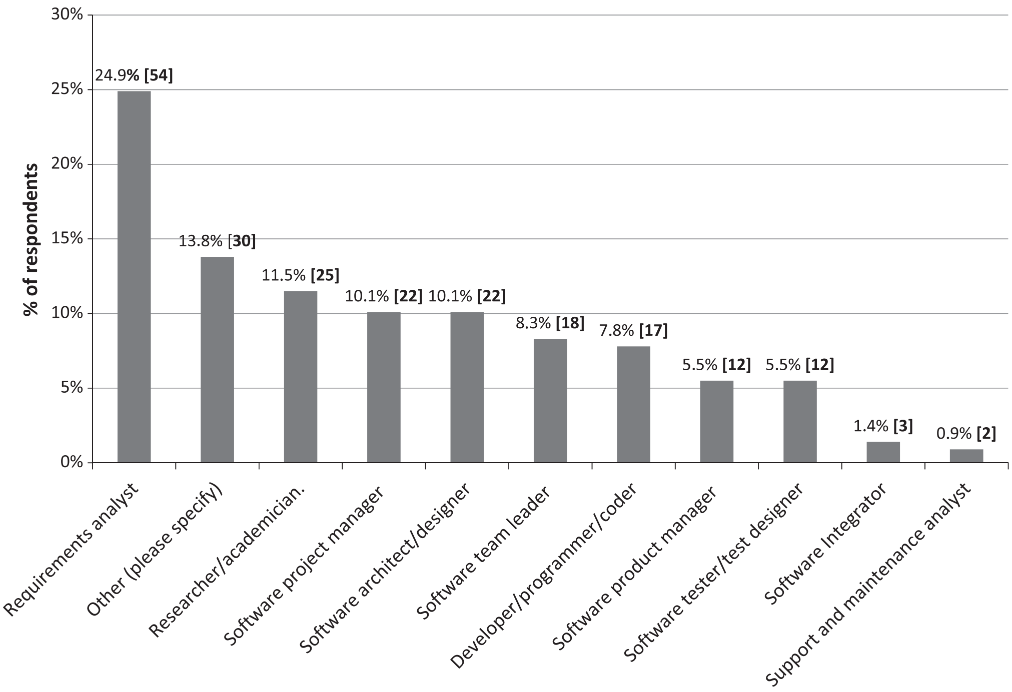

Fig. 2. Main role of respondents.

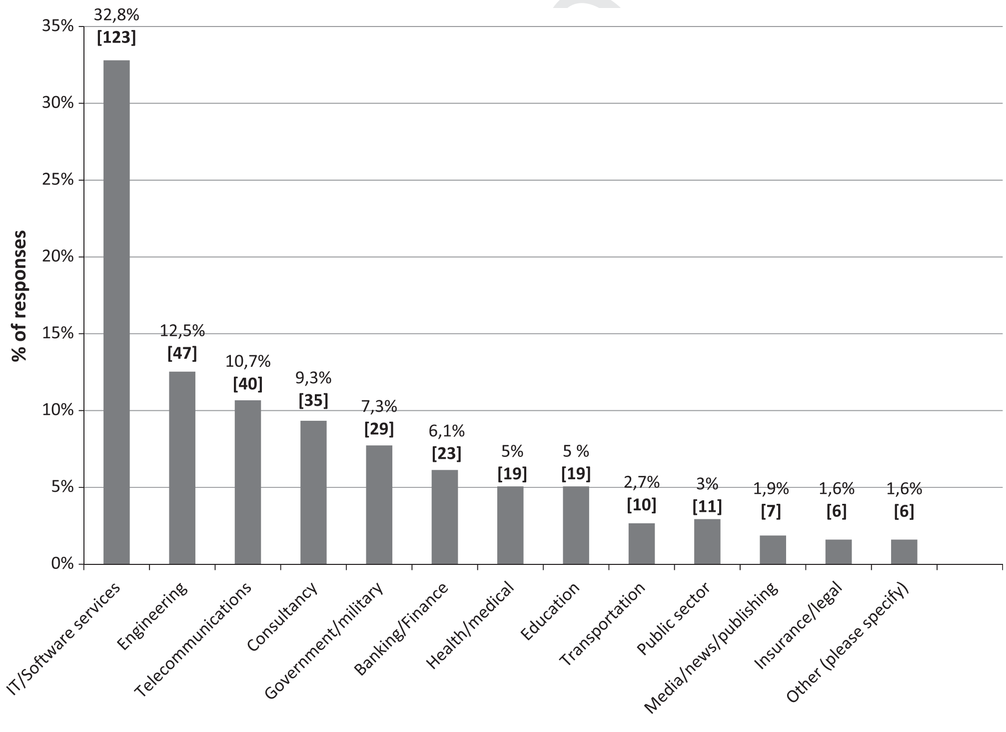

Fig. 3. Types of business domains of respondents.

mix of incremental, agile and others''. Other respondents used ''V-Model'', ''SLIM'', ''CMMI Level 3'', ''CMMI Level 5'' or had their own tailored methodology ''created for each company by blending methods/processes''.

Fig. 7 investigates the type of requirements engineering the respondents are involved in. Since this question also allowed multiple answers, the results are calculated based on the total number of responses. *Bespoke or Contract driven requirements engineering* received 44.2% of all the answers. *Market-driven requirements engineering* received 29.5%, while *Open source* only 5.1%. *Outsourced projects* appeared in 19.9% of the answers. Six answers were given to the *Other* category. Two respondents suggested none of the following. One was working with ''normal flow, requirements from product owner or developers'', one with ''builds'' one mainly with infrastructure projects, and one with ''large SAP implementation projects in a client organization''.

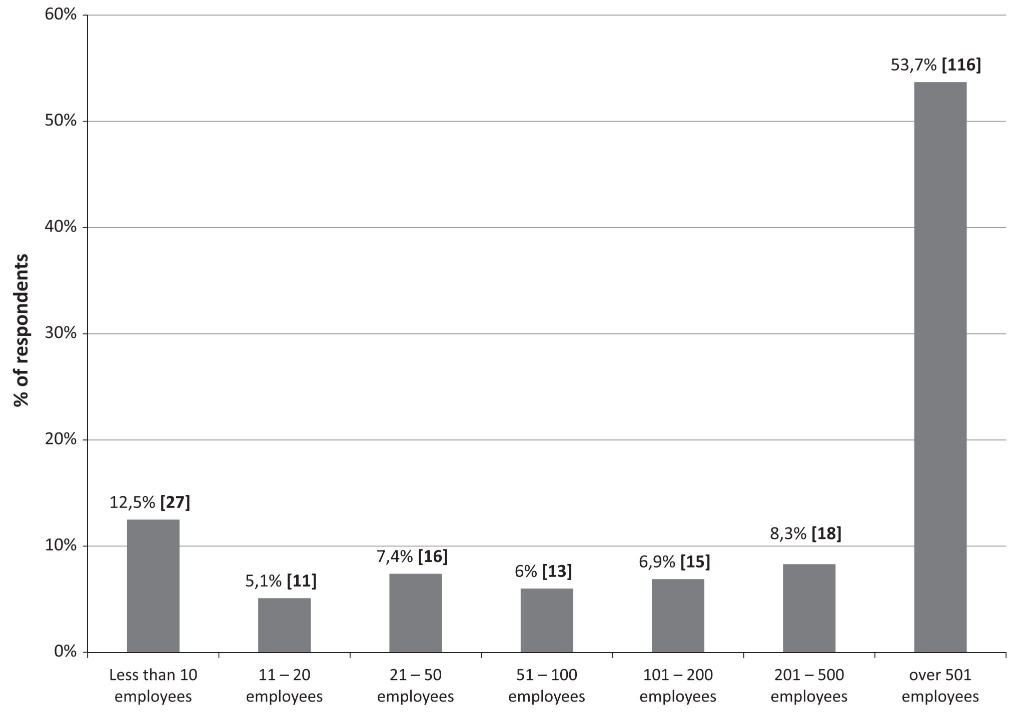

Fig. 4. Size of respondents' organization.

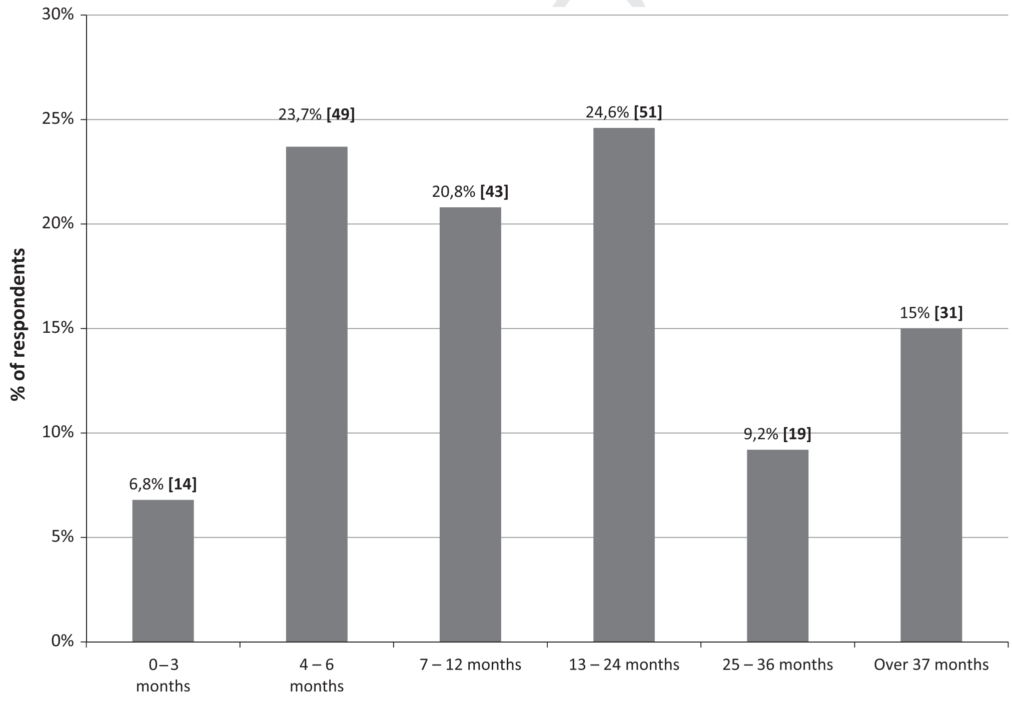

Fig. 5. Average duration of typical projects from our respondents.

4.2. Defining obsolete requirements (RQ1)

Defining the term Obsolete Software Requirement (OSR) is central to the understanding of the phenomenon. The categories used in this question were inspired by the definitions of OSR found in literature (Section 2), and are defined in the context of the current release (Section 3.2.1). Fig. 8 depicts the answers from all respondents. Since the question allows multiple answers, the results are calculated for all the answers, not the respondents. The primary answer selected (29.9%) defines OSR as "*no longer required for the current release for various reasons*". This result is in line with the definition of obsolete functionality provided by Zowghi and Nurmuliani [26]. The definition of an OSR as a requirement that: "*has no value for the potential users in the current release*" received 21% if the responses. This category is similar to the definition of obsolete software applications provided by Merola [15], as applications are taken off the market due to decrease in product popularity or other market factors.

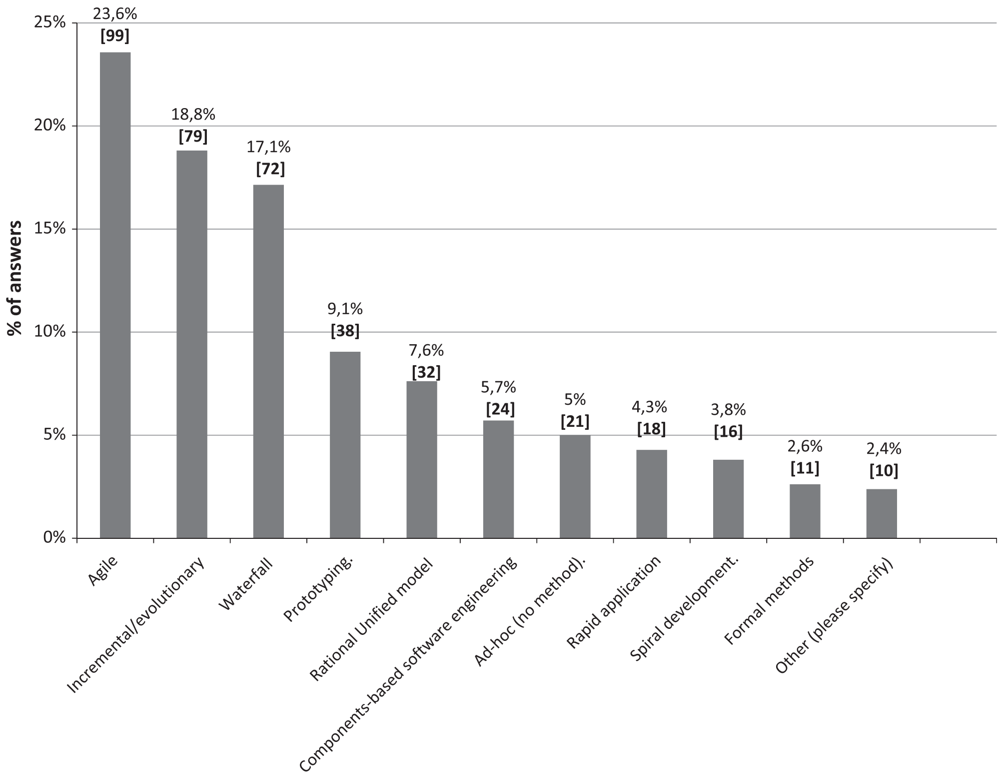

Fig. 6. Development processes and methodologies.

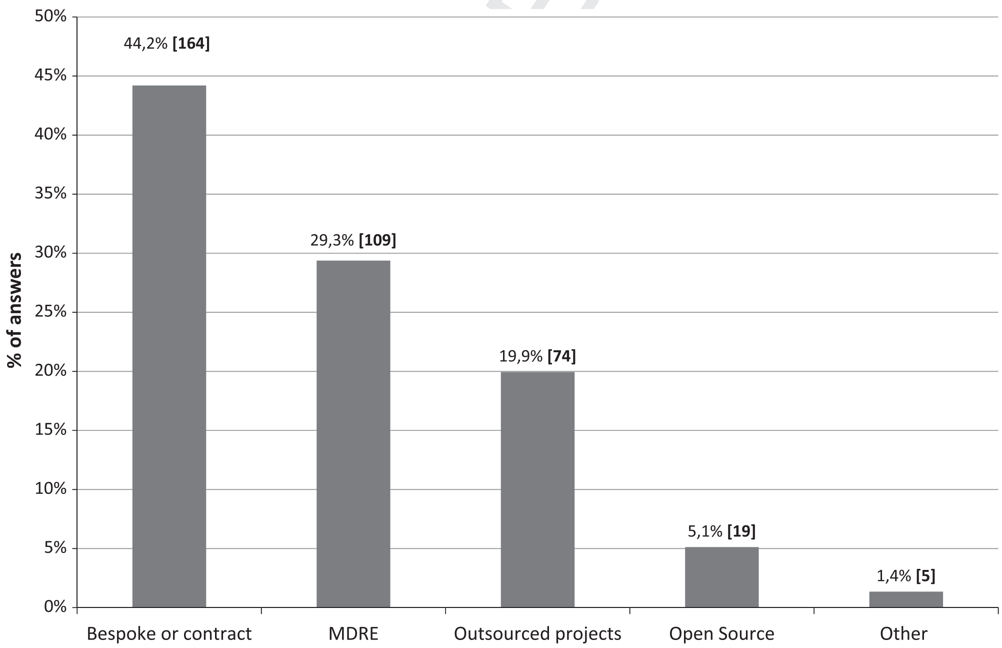

Fig. 7. Types of requirements engineering.

A total of 33 responses (7.7%) were in the *Other* category. Of these, 8 respondents (∼25%) suggested that an OSR is not necessarily confined to the current release, but it also goes to future releases. Respondents stressed that an OSR is a requirement that has lost its business goal or value. Other interesting definitions included: "an OSR is a requirement that evolved in concept but not in documentation", "an OSR will be implemented but will not be tested", and "carved by the IT to showcase technical capabilities to the end user".

As a result, the following definition of an OSR was formulated:

"An obsolete software requirement is a software requirement (implemented or not) that is no longer required for the current

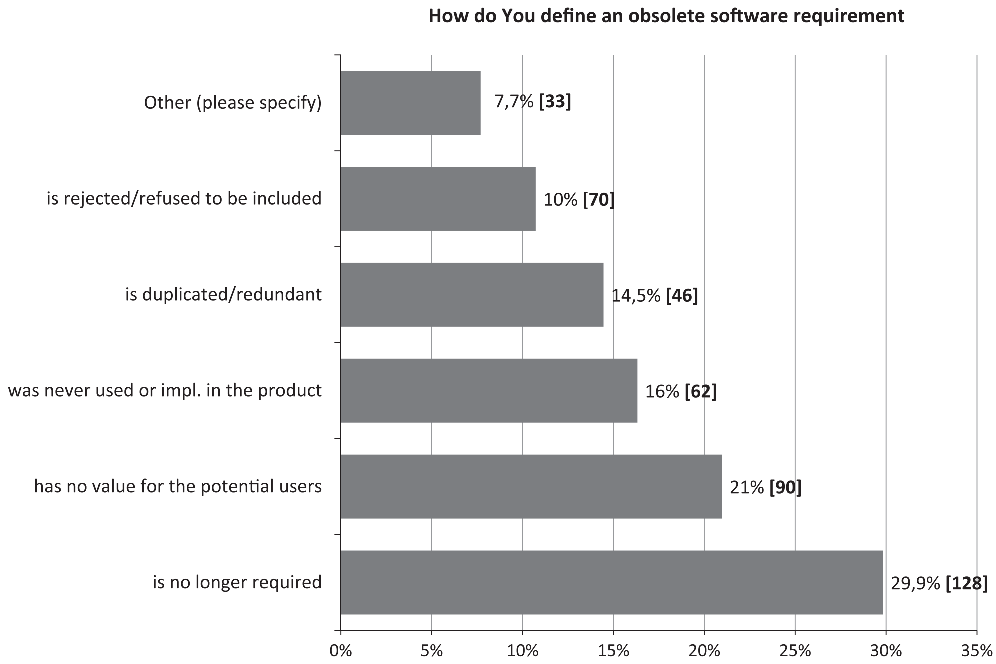

Fig. 8. Respondents' definition of an OSR.

release or future releases and, for various reasons, has little or no business value for the potential customers or users of a software product."

We performed statistical analyses to investigate whether there were relationships between the selected definition of OSRs and the respondents' roles, the size of organizations and the development methodologies used. Overall, the relationships were statistically insignificant due to violations of the chi-square test assumptions (some alternative answers had too few respondents, see Table A.2 in [61]). However, significant results could be observed (using the chi-square test) between the top five methodologies (Fig. 6) and and the results for choice of OSR definition (*p*-value 0.011, Table A.2a in [61]). Respondents that reported using a *Rational Unified Process* (RUP) methodology less frequently selected the definition of OSRs as no longer required for the current release (31.3% of all answers compared to over 50%) or never implemented in the product (34.4% of all answers compared to over 40%) than respondents that reported utilizing any of the remaining four methodologies. Moreover, the *RUP* respondents provided more answers in the *Other* category and indicated that OSRs can be "a requirement that evolved in concept but not in documentation" or "an abstract requirement to showcase the technical capability to the end user". Finally, only three *RUP* respondents defined OSR as a requirement that is rejected for inclusion in the current release, while about 20% of the respondents that selected the other top four methodologies selected this answer. This would seem to indicate that the perceived definition of an OSR for respondents using the *RUP* methodology is more stable than that for respondents using other methodologies.

Since the *RUP* methodology considers iterative development with continuous risk analysis as a core component of the method [63], we can assume that the risk of keeping never used or implemented requirements in the projects is lower. Moreover, the majority of the *RUP* respondents also reported working on bespoke or contract-driven projects, where the number of changes after the contract is signed is limited and usually extensively negotiated. Thus it appears to be possible that the *RUP* respondents could avoid rejected or refused requirements and could manage to achieve more precise and stable agreements with their customers [63] which in turn could result in fewer OSRs.

Reviewing the top five methodologies, the most popular answer was no longer required for the current release. Interestingly, among the respondents working with agile, incremental or evolutionary methodologies, the fourth most popular answer was *never used or implemented in the product.*

In contrast, respondents who worked with waterfall, prototyping or RUP methodologies have the same order of popularity of answers. The definition of an OSR as a *was never used or implemented in the product* requirement was the second most popular answer while the option *is duplicated/redundant in the current release* was the third most popular answer. The possible interpretation of these results is that agile and incremental methodologies less frequently experience OSRs as never used or implemented but experience more OSRs as duplicated requirements and requirements with no value for the potential users.

Further analysis reveals that the definition of OSRs is not significantly related to the size of the companies, the length of the typical project, or the domain (*p*-values in all cases greater than 0.05). Domain and project length could be seen as qualifiers of OSRs. For example, projects running over long periods could suffer increased requirements creep [8]. However, this would most probably not be visible in the definition of OSRs, but rather in the impact of OSRs, which is investigated in the next section.

4.3. The potential impact of OSRs (RQ2)

When queried about the potential impact of OSRs on their product development efforts a total of 84.3% of all respondents considered OSR to be *Serious* or *Somehow serious* (Fig. 9). This indicates that among the majority of our respondents OSRs seems to have a substantial impact on product development. Our result confirms previous experiences. (See, e.g., Murphy and Rooney [13], Stephen et al. [14] and Loesch and Ploederoeder [18].) For 6% of the respondents OSRs are a *Very serious* issue, while 10% (21 respondents) deemed OSR a *Trivial* matter.

To further decompose and test context variables, e.g., company size, respondents' roles and development methodologies, we

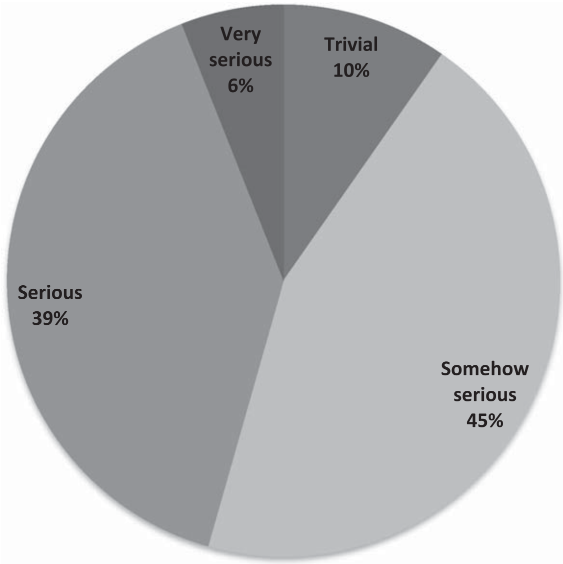

Fig. 9. Impact of OSRs on industry practice.

performed chi-square tests (Table A.1 in [61]) between the context variables and the degree to which OSRs were considered having a substantial impact. The tests resulted in *p*-values greater than 0.05, which indicates that no statistically significant relationships between the analyzed factors could be seen. We can, however, ascertain that a clear majority of the respondents deemed the phenomenon of OSRs a relevant factor to be taken into consideration in development efforts.

Of the 21 (10%) respondents who considered OSRs to be *Trivial*, approximately 58% worked with requirements or in project management roles. This would seem to indicate that those respondents, contrary to those in software development roles, have less difficulty in managing OSRs. An analysis of the answers to questionnaire question 9 ([61] and Section 4.9) revealed that 10 respondents who considered OSRs to be *Trivial* also confirmed having a process for managing OSRs. Thus, it appears to be a logical conclusion that the negative influence of OSRs on product development could be alleviated by designing and introducing an appropriate process of managing OSRs. More about the current processes discovered among our respondents can be found in Section 4.9.

Further analysis of the respondents who considered OSRs as *Trivial* indicated that more than 80% of them worked for large companies with >101 employees. Since large companies often use more complex process models [64], in contrast to small companies which might have budget constraints to prevent hiring highly quality professionals and whose processes are typically informal and rather immature [65], we could assume that the issue of managing OSRs could have been already addressed in these cases.

Further analysis of the *Trivial* group indicated that almost half of them (47.6%) worked in the *IT or computer and software services* domain, In the service domain, the main focus of requirements engineering is to identify the services that match system requirements [66]. In the case of insufficient alignment of new requirements with the current system, product development may simply select a new, more suitable, service. This, in turn, might imply that the OSRs are discarded by replacing the old service with the new one. Further, the typical product lifetime for IT systems is usually shorter than for engineering-focused long-lead time products [67] (such as those in the aerospace industry), which in turn could minimize the number of old and legacy requirements that have to be managed. The possible interpretation of our analysis is that OSRs are less critical in IT and service oriented domains. Although this is a possible and plausible explanation, further investigation is needed to reach a conclusion.

Among the respondents who considered OSRs *Very serious* (13 respondents), the majority (53.8%) worked in large companies and used agile, ad hoc, or incremental methodologies (61.6%). This result seems to indicate that OSRs are also relevant for agile development and not reserved for only more traditional approaches like waterfall. Ramesh et al. [4] pointed out after Boehm [68] that quickly evolving requirements that often become obsolete even before project completion significantly challenge traditional (waterfall) requirements engineering processes. Murphy and Rooney [13] stressed that the traditional requirements process seriously contributes to the creation of obsolete requirements by creating a "latency between the time the requirements are captured and implemented". This latency should be lower in agile projects, characterized by shorter iterations and greater delivery frequency. This might indicate that either the latency is present in agile projects as well, or that latency is not the primary determinant of OSRs. It should be observed that 69.2% of the respondents who considered OSRs as *Very serious* reported having no process for handling OSRs. This could indicate why OSRs were considered a *Very serious* problem.

The cumulative cross tabulation analysis of the respondents who considered OSRs *Somehow serious*, *Serious* or *Very serious* (total 196 respondents, 89%) confirmed the severe impact of OSRs on large market-driven and outsourced projects (Section 4.7.2). Moreover, 76.8% of those respondents reported that they had no process, method, or tool for handling OSRs. In addition, 72.3% of respondents who considered OSRs *Somehow serious*, *Serious* or *Very serious* used manual methods to identify OSRs. It is also interesting to observe that the were only small differences between answers fron respondents who declared the following: *Agile software development* or *Incremental or evolutionary development* methodologies, and *Waterfall development*. Respondents using *Waterfall development* (and considered OSRs *Serious* or *Somehow serious* or *Very serious*) were somewhat more prone to dismiss the impact of OSRs compared to respondents using *Agile software development* or *Incremental or evolutionary development* methodologies. This would seem to indicate that, because waterfall-like processes usually restrict late or unanticipated changes and focus on extensive documentation [69,7,70], the impact of OSRs in those processes could be minimized. However, it says nothing about the extent that the realized features were useful or usable for the customers. Some waterfall projects may not have perceived OSRs to be a major issue for the project, but they might be for the product *per se*. That is, implementing an outdated feature might not be a perceived as a problem in a project. At the product level, where the overall value of the product for the customer should be maximized through the selection of the right features to implement and best alternative investment should be considered, another feature could be implemented instead of the outdated one. This is a classical case of perspective being a part of the value consideration as described by Gorschek and Davis [71].

The type of requirements engineering context factor (Fig. 7) only minimally influenced the overall results for this questionnaire question. Respondents who reported to work with *Bespoke or contract driven requirements engineering* graded OSRs slightly less serious than respondents who reported working with *MDRE*. This seems to indicate that OSRs are a problem in both contract driven (where renegotiation is possible [2]) and market-driven (where time to market is dominant [2]) projects. However, the difference could also indicate that there is a somewhat alleviating factor in contract-based development. That is, contract based development aims at delivering features and quality in relation to stated contract, thus getting paid for a requirement even if it is out of date

Please cite this article in press as: K. Wnuk et al., Obsolete software requirements, Inform. Softw. Technol. (2012), http://dx.doi.org/10.1016/j.infsof.2012.12.001

at delivery time. In an MDRE context, however, the product might fail to sell if the requirements are not fulfilled and the features out of date [2].

4.4. Requirements types and OSRs (RQ3)

The respondents were asked to choose what types of requirements were most likely to become obsolete (Likert scale, 1 = *Not likely*, and 5 = *Very likely*). We reviewed several classification schemes before choosing the categories. The classification proposed by Aurum and Wohlin [72] and SWEBOOK [73] inspired us to have functional and quality requirements types as well as to include sources of requirements into categories. The examples of requirements related to government legislations in banking provided by SWEBOOK [73] and the *change to government policy or regulation* trigger mentioned by McGee and Greer [43] inspired us to include requirements related to standards, laws and regulations. The scope of the requirement dimension suggested by SWEBOOK inspired us to include *requirements related to third party components e.g. COTS* and *requirements related to design and architecture* categories.

The analysis of the reasons of requirements changes presented by Nurmuliani et al. [74] inspired us to add incorrect requirements (mentioned as one of the reasons for requirements changes by Nurmuliani et al.), ambiguous and inconsistent requirements categories. Both Harker et al. [42] and McGee and Greer [43] focused on the changing nature of software requirements. The *enduring requirements* type suggested by Harker et al. [42] inspired us to include *requirements about the company's organization and policies* category. The *changing requirements* category inspired us to include *functional requirements originated from customers*, *functional requirements originated from end users* and *functional requirements originated from developers* categories. The *customer hardware change* trigger of requirements changes listed by McGee and Greer [43] inspired us to include *hardware related requirements* category. The classification of software project requirements knowledge presented by Shan et al. [75] was reviewed but not used while creating the categories.

According to the results depicted in Fig. 10, OSRs seem to belong to the categories of *Incorrect or misunderstood requirements* (mean 3.88), *Inconsistent requirements* (mean 3.74), or *Ambiguous requirements* (mean 3.72). While several studies focused on the problem of inconsistencies between requirements, e.g., by proposing techniques to identify and remove inconsistencies [76], decomposing a requirements specification into a structure of "viewpoints" [77], or distributing development of specifications from multiple views [78], they did not study inconsistent requirements as a potential source of OSRs. From a becoming obsolete standpoint, the level and quality of specification should not matter *per se*. However, if the lack of quality of a requirement's specification is seen as an indicator of a lack of investment in the analysis and specification of the requirement, several possible scenarios could emerge. For example, practitioners in industry might have a *gut feeling* that certain requirements will become OSRs and thus, are not worth the effort. Another possibility is that OSRs are harder (require more effort and knowledge) to specify than other requirements types, although, it could just as well indicate that most requirements are specified badly and thus are also OSRs. Further investigation is needed to investigate the potential reasons for the results achieved. The only thing we can say for certain is that requirements becoming obsolete seem to suffer from inadequacies in terms of correctness, consistency, and ambiguous specification.

Interestingly, requirements from domain experts were considered less likely to become obsolete than requirements from customers, end users, and developers respectively. One explanation could be that domain experts possess the knowledge and experience of the domain, and thus their requirements may be less likely to change [79]. On the other hand, since the customers are the main source of software requirements and the main source of economic benefits to the company, their requirements are crucial to the success of any software project [80]. This implies that this category must be kept up to date and thus be less likely to become obsolete. Another possible explanation could be that customer requirements are not as well or unambiguously specified as internal requirements [80,29], resulting in a tendency of those requirements to become obsolete faster or more frequently.

Obsolescence of customer requirements, rather than internal requirements from domain experts, is confirmed by Wnuk et al. [8]. They reported that stakeholder priority dictates removal and postponement of the requirements realization, and domain experts are often part of the prioritization of all requirements. On the other hand, Kabbedijk et al. [81] reported that change requests from external customers are more likely to be accepted than change requests from internal customers. This might imply that some customer requirements are handled as change requests instead of as requirements input to development projects. In both cases, the

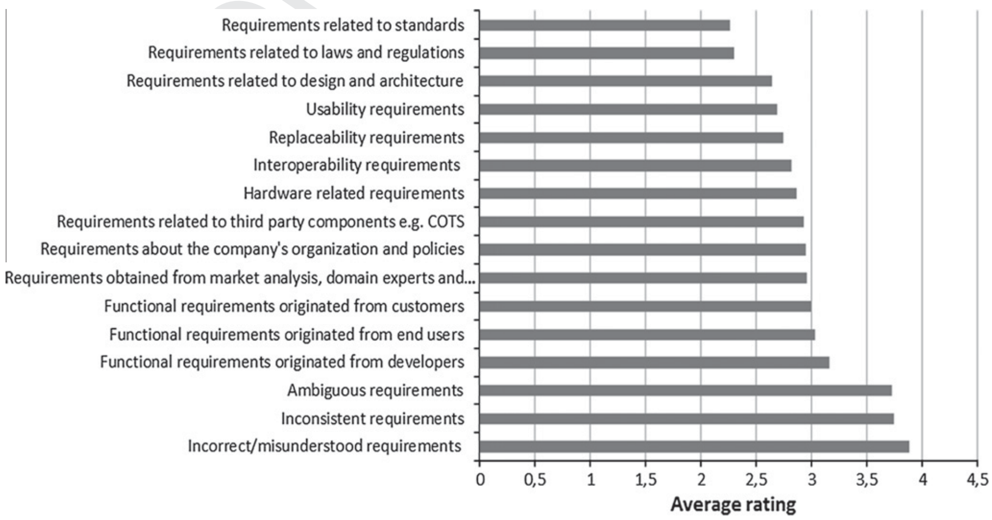

Fig. 10. Types of OSRs likely to become obsolete.

authors reported high requirements volatility, which is in line with the study by Zowghi and Nurmuliani [26] who related obsolete requirements related to requirements volatility.

According to our respondents, requirements related to standards, laws and regulations are the least likely to become obsolete, which seems logical, as the lifetime of legislation and standards is often long in comparison to customer requirements. Furthermore, the low average score for the *Requirements related to third party components e.g. COTS* (even lower than for the requirements related to the company's organization and policies) also seems to be logical, especially in relation to the results for RQ2 (Section 4.3) where almost half of the respondents who considered OSRs to be *Trivial* worked with *IT or Computer and software services domain*. We assume, after Bano and Ikram [66], that COTS are used in the software service domain. The results for the respondents who worked with *Outsourced projects* (question 15 in [50]) are in accordance with the overall results.

The differences between the respondents who worked with *Outsourced*, *MDRE* and *Bespoke or contract driven requirements engineering* projects in relation to the degree of obsolescence of COTS requirements are subtle. This may suggest that other aspects not investigated in this study could influence the results. Although OSRs do not appear to be related to the main challenges of COTS systems, i.e., the mismatch between the set of capabilities offered by COTS products and the system requirements [82], the nature of the COTS selection process, (e.g. many possible systems to consider and possible frequent changes of the entire COTS solution), may help to avoid OSRs.

Further analysis of the influence of the context factors indicates that the respondents' domains, company size, and methodologies have minimal impact on the results. Not surprising, more respondents who worked with projects running over longer time spans graded *Functional requirements originated from end users* as *Very likely* to become obsolete than respondents who worked with short projects (8.7% of respondents who worked with projects <1 year and 25.7% respondents who worked with projects >1 year). One explanation could be that long projects, if deprived of direct and frequent communication with their customers and exposed to rapidly changing market situations, can face the risk of working on requirements that are obsolete from the users' point of view. This interpretation is to some extent supported by the results from RQ7 (Table 4) where the respondents graded *MDRE* contexts (characterized by limited possibilities to directly contact the end users and continuously arriving requirements [2]) or *Outsourced projects* (where communication is often done across time zones and large distances [83]) as more affected by OSRs than bespoke contexts. The success of *Market-driven projects* primarily depends on the market response to the proposed products [2], which if released with obsolete functionality, may simply be required by customers. Thus, we believe that it is important to further investigate additional factors that could render *Functional requirements originated from end users* obsolete.

4.5. Methods to identify OSRs (RQ4)

More than 50% of the answers pointed out that manual ways of discovering OSRs are currently the primary method (Fig. 11). At the same time, the context factors such as the different methodologies, types of RE, length of the projects, roles of respondents and the domain that respondents worked in did not significantly affect the top answer for this question. A total of 13.29% of all answers indicated the presence of a predefined "obsolete" status. Furthermore, 11.19% of all answers (32 answers) were given to the category *I never found them or I never thought of finding them*. Finally, less than 10% of all answers (24 answers) indicated the existence of any sort of automation to identify OSRs.

In the *Other* category, seven respondents mentioned that OSRs could be identified "by execution of test cases based on requirements" or "during regression testing cycles". Further, three answers suggested "using requirements traceability matrix while testing the software" while three answers suggested improved communication "by discussion of user stories with stakeholders". Finally, one respondent suggested that goal-oriented requirements engineering makes "finding OSRs trivial".

The answers from respondents who indicated using automated ways of discovering OSRs provided some names for the automated techniques, e.g., "customized system based on JIRA that takes OSRs into account by using special view filters", "traceability using DOORs to analyze for orphan and to track and status obsolete requirements", or "a tool called Aligned Elements to detect any inconsistencies including not implemented requirements". This would indicate that some tool support is present. However, tool efficiency and effectiveness was not part of this study.

Further analysis indicated that the majority of respondents using tools of some sort worked with companies with >501 employees (62%). This seems reasonable as large companies usually have more money for tool support [65], and can even request especially tailored software from the requirements management tool vendors. The fact that automated methods to identify OSRs are rare among the smaller companies calls for further research into lightweight and inexpensive methods of OSR identification that can more easily be adapted in those companies. Furthermore, as both smaller and larger companies fall short in automation and assuming that larger companies can invest more money into education, this is probably not due to education either.

More than half (15) of the respondents from the automated group also indicated that they identify OSRs manually. One explanation could be that automated methods are used together with manual methods, e.g., after the respondents manually mark requirements as obsolete or perform other preliminary analysis that enables automated sorting. Searching, tagging or filtering capabilities in their requirements management tools are most likely dominant and seen as *automated* in relation to OSRs, but this task is done in an ad hoc manner and not integrated with their requirements management process. Thus the "level of automation" needs further investigation.

The reasonably high number of answers given to the category *I never found them or I never thought of finding them* is intriguing and needs further investigation. Thirty respondents from this group (93.8%) also indicated having no process for managing OSRs. This seems logical as the inability to find OSRs could be related to the lack of processes for managing OSRs. Further, the majority of the respondents that indicated never finding OSRs worked with projects shorter than 12 months, and one fourth of them indicated having an ad hoc process for managing requirements. The relatively short project times were not an indication of OSRs not being an issue as >80% of these same respondents indicated OSRs as being a *Serious* or *Very serious* issue. The absence of a defined and repeatable process might be a better indicator for not identifying OSRs in this case. In addition, waterfall was represented in more than 11% of the cases, while only about 6% worked in an agile manner.

Neither organizational size nor development methodology were statistically significant factors in terms of how OSRs were discovered or identified (Table A.5 in [61]). However, a statistically significant relationship was identified in relation to the top five methodologies and how OSRs were identified (chi-square test $p < 0.004$, Table A.5a in [61]). This result could be explained by the following: (1) respondents who worked with waterfall methodology admitted more often to never finding OSRs (11%) than respondents who worked with agile methodologies (3.8%), (2) more respondents who worked with *RUP* methodology (34%)

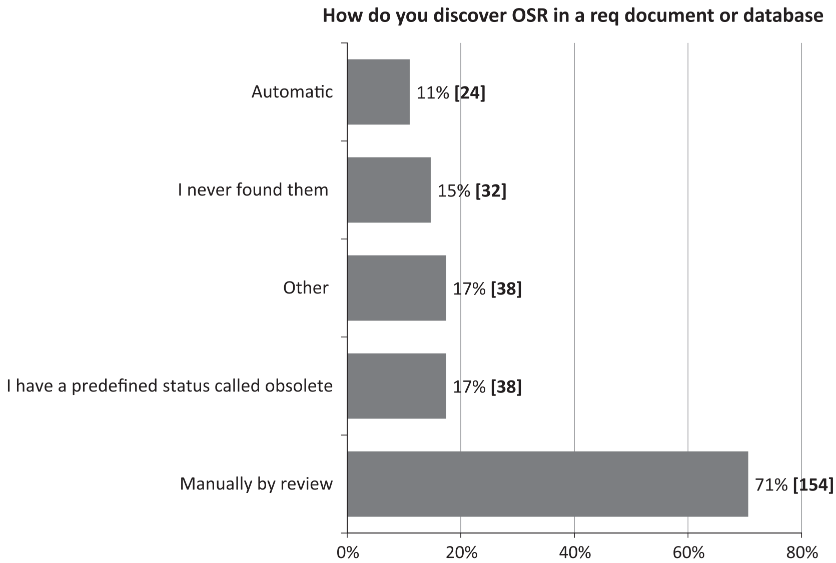

Fig. 11. Methods used to identify OSRs.

selected the option *I have a predefined status called obsolete* than respondents who worked with agile methodology (10%). Looking further, we could also see that the majority of the respondents who worked with *RUP* or *Prototyping* methodologies also worked with companies with >201 employees. This would seem to indicate that within the two mentioned methodologies it is possible to implement tool support for identification of OSRs. It is worth mentioning that a statistically significant relationship was also achieved between the top five methodologies and the results for choice of OSR definition (*p*-value 0.011, Table A.2a in [61]) and Section 4.3. The results suggest that the respondents who worked with the RUP methodology may have a different opinion about the definition of OSRs and more frequently use a predefined status called obsolete to identify OSRs.

Looking at the types of requirements engineering used, the results showed that the respondents who work with *Bespoke or contract driven requirements engineering* did not use predefined categories for OSRs; it was not part of their standard procedure to sort out OSRs. This seems to be logical as the majority of the respondent who admitted to never finding OSRs worked with bespoke or contract-driven projects. Finally, only one respondent mentioned automatic methods of finding OSRs.

For the context factor of project length, longer projects have more automated ways of identifying OSRs (the difference is about 5%) than shorter projects. This seems reasonable as longer projects usually invest more into project infrastructure and project management tools and processes. However, a large part of the longer projects respondents also indicated manual methods of identifying OSRs (about 60% for projects >1 year). In comparison, subjects typically working in shorter projects used more tool supported automated methods (about 52% for projects <1 year). Thus the respondents working in longer projects did see the point of, and did try to, identify OSRs to a larger extent than the ones working in shorter duration projects, although manual methods dominated.

The analysis of the influence of the respondents' roles on the results revealed only minimal differences. Among the interesting differences, project and product managers respondents gave no answers in the I never found them category. This may indicate that they always find OSRs. Further, the management roles had the highest score for manual identification of OSRs. This result might indicate that management is, to some extent, more aware of the need for finding OSRs which may severely impede the project efforts. However, tool support is often lacking.

4.6. Handling of identified obsolete software requirements (RQ5)

More than 60% of the answers (results for multiple answer questions are calculated based on all the answers) indicated that the respondents kept the OSRs but assigned them a status called "obsolete" (see Fig. 12). This might indicate that OSRs are a useful source of information about the history of the software product for both requirements analyst and software development roles. Moreover, 21.9% of all answers (66) suggested moving OSRs into a separated section in requirements documents. These views were the most popular among the respondents regardless of their role, methodology, domain, size, project length and context. One could interpret this response as indicating that the most suitable way to manage identified OSRs is to classify them as obsolete, supplying rationale, and move them into a separated section or document or SRS. However, maintaining traceability links between OSRs and other requirements could prove work intensive, especially if end-to-end traceability is required [64]. Regnell et al. [51] discuss scalable methods for managing requirements information where effective grouping of requirements e.g., placing semantically similar requirements in the same module, could enable more efficient maintenance of large structures of requirements (although OSRs were not mentioned specifically).

Looking at the answers given the *Other* category, two answers suggested informing the stakeholders about assigning a requirement an obsolete status. Furthermore, two respondents suggested to "hide and tag requirements that are obsolete using requirements management tools". Interestingly, one respondent questioned "why would you spend time in on dealing with not needed things". Since this person worked with a very small company with about 20 employees, we assume that the problem of overloaded database with legacy requirements is not known to this person. Finally, the other answers in this category mostly suggested keeping OSRs and optionally writing the justification.

Most of the answers in the *Other* category (~6%, 20 answers) suggested either removing OSRs, or keeping them, but moving them to a separated section or module in the database. Only ~9% of answers (26) suggested deleting the OSRs from the requirements database or document. This suggests that most respondents think OSRs should be stored for reference and traceability reasons. However, keeping OSRs appears to be inconsistent with recommended practice for reducing the complexity of large and very large projects [51,84], and handling information overload as high-

Please cite this article in press as: K. Wnuk et al., Obsolete software requirements, Inform. Softw. Technol. (2012), http://dx.doi.org/10.1016/j.infsof.2012.12.001

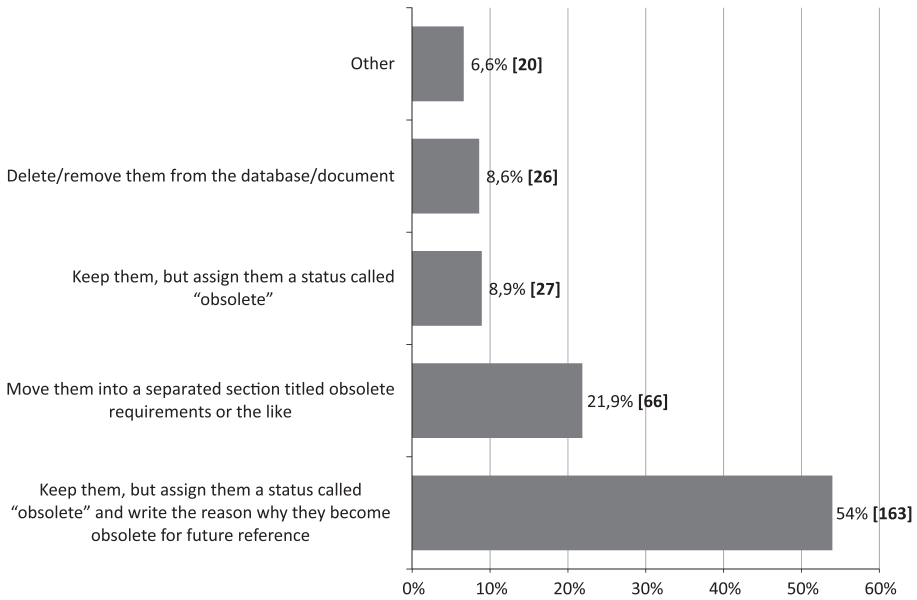

Fig. 12. Methods used to manage identified OSRs.

lighted by Regnell et al. [51]. The desired behavior in large and very large projects would seem to indicate the removal of unnecessary requirements to decrease the complexity of the requirements structure and traceability links. One possible avenue for further investigation is to evaluate the value of keeping OSRs.

Of the group who opted for OSRs deletion upon identification, the majority of the answers came from respondents who worked with large companies (>501 employees, 77%) and long projects (>12 months, 53.9%). Moreover, a majority of these respondents considered OSRs to be *Serious* or *Somehow serious* (Section 4.3). On the contrary, respondents that worked in smaller companies opted to keep OSRs.

Analysis revealed a lack of statistically significant relationships between the answers for this question (Fig. 12) and and the respondents' roles, domains, organizational size and, methodologies used (Table A.6 in [61]). However, some indications could be observed. Respondents working in the engineering domain seemed to prefer the deletion of OSRs compared to respondents from other domains. One possible explanation could be that since the projects in the engineering domain are highly regulated, and often require end-to-end traceability [64], keeping OSRs in the scope could clutter the focus threatening to impede requirements and project management activities.

Type of requirements engineering factor turned out to have a minimal impact on the results regarding this question. However, one observation worth mentioning is that more support was given to the option of removing OSRs among the respondents who worked with *Bespoke or contract driven requirements engineering* (12.3%) than respondents who worked in *MDRE* (9.2% of answers). This appears to be logical as, in bespoke projects, obsolete requirements could be discarded after the contract is fulfilled. In market-driven projects they could be kept and later used during the requirements consolidation task, where new incoming requirements could be examined against already implemented or analyzed requirements which include OSRs [85].

4.7. Context factors and obsolete software requirements (RQ6 and RQ7)

4.7.1. Obsolete software requirements and project size

The respondents were asked to indicate to what extent the phenomenon of OSRs would potentially (negatively) impact a project, and whether project size had anything to do with the likelihood of negative impact. The respondents used a Likert scale from 1 (*Not likely* impact) to 5 (a *Very likely* impact). The results are presented in Tables 3 and 4 below. The size classification is graded in relation to number of requirements and interdependencies, inspired by Regnell et al. [51].

Column 7 in Table 3 presents the average rating for each project size. We see that the larger the project, the more likely there will be a negative effect from OSRs. Looking at Table 3 for *Small-scale requirements* projects, most respondents deemed OSR impact as *Not likely* (35.3%) or *Somewhat likely* (35.8%). However, moving up just one category to *Medium-scale requirements* projects with hundreds of requirements, the respondents indicated the impact as being *Likely* (41.5%). The trend continues with *More than likely* (32.7) for *Large-scale requirements* projects, and *Very likely* for *Very large-scale requirements* projects (38.9%). The results confirm the viewpoint of Herald et al. [20] who listed OSRs as one of the risks in large integrated systems.

One interesting observation is that the results could be seen as potentially contradictory to the results from questionnaire question 2 (Section 4.3) where the respondents who worked in larger companies (over 100 employees) graded the overall impact of OSRs slightly lower than respondents from smaller companies. However, since large companies often have large databases of requirements [51] and often run projects with several thousands of requirements [86], this would suggest that there are other factors that influence the impact of OSRs.

When it comes to the influence of methodology used by our respondents, we report that the respondents who used *Agile software development* methodology primarily graded OSRs as only *Likely* to affect *Large-scale requirements* projects, while respondents who used *Waterfall* methodology primarily graded the impact of OSRs as *More likely*. Interestingly, this result seems to contradict the results for RQ2 (Section 4.3), where the majority of respondents who considered OSRs *Very serious* worked in large companies and used agile or incremental methodologies. This might indicate that the size of the project is not more dominant than the size of the company, and the methodology used. This requires further investigation.

The respondents who worked with bespoke or contract driven requirements engineering primarily graded the effect of OSRs on *Large-scale requirements* projects as *Likely*. On the contrary, the

Table 3
OSRs effect on project size (215/219 respondents).

	(1) Not likely	(2) Some-what likely	(3) Likely	(4) More than likely	(5) Very likely	Rating average
Small-scale (~10 of req.)	35.3% (76)	**35.8% (77)**	13.5% (29)	7.0% (15)	8.4% (18)	2.17
Medium-scale (~100 of req.)	9% (19)	31.6% (67)	**41.5% (88)**	16.0% (34)	1.9% (4)	2.70
Large-scale (~1000 of req.)	3.8% (8)	17.1% (36)	31.3% (66)	**32.7% (69)**	15.2% (32)	3.38
Very large-scale (>10,000 of req.)	8.1% (17)	12.8% (27)	16.6% (35)	23.7% (50)	**38.9% (82)**	3.73

Table 4
How likely OSRs affect various project types (215/219 respondents).

	(1) Not likely	(2) Some-what likely	(3) Likely	(4) More than likely	(5) Very likely	Rating average
Bespoke projects	14.4% (31)	**32.1% (69)**	26% (56)	16.3% (35)	11.2% (24)	2.78
Market-driven projects	6.5% (14)	20% (43)	**35.8% (77)**	23.3% (50)	14.4% (31)	3.19
Outsourced projects	2.3% (5)	16.4% (35)	**35.7% (76)**	27.2% (58)	18.3% (39)	3.43

respondents who worked with *Market-driven projects* primarily graded the impact of OSRs on *Large-scale requirements* projects as *Very Likely*. This result confirms the results for RQ2 (Section 4.3) where OSRs were also graded less serious by respondents who worked in bespoke contexts. Finally, for the *Very large-scale requirements* projects our respondents primarily graded the impact of OSRs as *Very likely* regardless of context factors.

4.7.2. Obsolete software requirements and project types

The respondents were also asked to rate how likely it was that OSRs affected various project types (on a scale from 1 to 5, where 1 is *Not likely*, and 5 is *Very likely*). The results for the average rating (column 7 in Table 4) indicate that *Outsourced projects* are the most likely to be affected by OSRs (average rating 3.43). One possible explanation for this result could be the inherited difficulties of frequent and direct communication with customers and end users in *Outsourced projects*. Moreover, as communication in *Outsourced projects* often needs to be done across time zones and large distances [83,87], the risk of requirements misunderstanding increases, and as we have seen (Section 4.4), inadequately specified requirements run a higher risk of becoming OSRs.

The high average rating for the *Market-driven projects* (average scope 3.19) can be explained by the inherited characteristics of the MDRE context where it is crucial to follow the market and customer needs and the direct communication with the customer may be limited [2]. This in turn can result in frequent scope changes [8] that may render requirements obsolete. Finally, it is worth mentioning that the gap between the *Market-driven projects* and *Bespoke projects* (average score 2.78) is wider than between *Outsourced* (average scope 3.43) and *Market-driven projects* (average score 3.19). One possible explanation could be that both *Market-driven projects* and *Outsourced projects* suffer similar difficulties in directly interacting with the end users or customers [2,83] and thus the risk of requirements becoming obsolete could be higher.

The results for all the categories and scales are presented in columns 2–6 in Table 4. Our respondents primarily graded the impact of OSRs on *Market-driven projects* and *Outsourced projects* as *Likely* and only *Somehow likely* for *Bespoke projects*. Interestingly, the answer *Very likely* did not receive top scores for any of the three types of projects. This would seem to indicate that the "project type" factor is less dominant in relation to OSRs than the "size" of the project discussed earlier in this section.

Since the statistical analysis between the results from the question and the context variables revealed no significant relationships, we performed descriptive analysis of the results. The respondents who indicated having a managerial role (32.7%) primarily graded the impact of OSRs on the *Market-driven projects* as *More than likely*, while the requirements analysts primarily graded this impact as only *Likely*. Similar to this result are the results for RQ2 (Section 4.3) where the managers primarily considered OSRs as *Serious* while requirements analysts predominantly considered it *Somehow serious*. The comparison is, however, not straight forward as in case of RQ2 where respondents were grading all types of requirements projects, not only *Bespoke projects*. Finally, the opinions of software development and management roles are aligned when grading the impact of OSRs on bespoke projects (the majority of the respondents from both roles graded the impact as *Somehow likely*).

In relation to project duration, interestingly, respondents who worked with smaller companies (<200 employees) more often graded the effect of OSRs on *Bespoke projects, Market-driven projects* or *Outsourced projects* as *Likely* or even *Very likely*. The majority of the respondents who worked for companies with >201 employees selected the *Somehow likely* answer for the *Bespoke projects* and *Market-driven projects*. This result confirms the previous analysis (Section 4.7.1) by indicating that size is not the only factor that impacts the seriousness of OSRs. It can also be speculated that the phenomenon of OSRs might be clearer in smaller organizations where less specialization makes outdated requirements more "everybody's concern", while in larger organizations, with high specialization, the view of "not my job" might play a factor [64].

4.8. Where in the requirements life cycle should OSRs be handled (RQ7)

The results for this question are presented in Fig. 13 as percentages of the total number of answers (717) since the question allowed multiple answers. The list of phases (or processes) in the requirements engineering lifecycle was inspired by Nurmuliani and Zowghi [26]. According to our respondents OSRs should first be handled during *Requirements analysis, Requirements validation* and *Requirements changes* phases (each with about 14% of the answers). This result is, to some extent in line with the study by Murphy and Rooney [13], SWEBOK [40], and Nurmuliani and Zowghi [26] who report that change leads to volatility, and volatility in its turn leads to obsolescence. However, less than 5% of the survey respondents indicate that OSRs should be managed as a part of chandling requirements volatility seems to support a distinction between volatility and the phenomenon of OSRs as such. That is, volatility may be related to OSRs; however, it needs to be handled continuously during analysis and validation as a part of change management in general.

The high numbers of answers given to *Requirements analysis* (14.5%) and *Requirements specification* (9.2%) phases confirm the suggestions made by Savolainen et al. [17] to manage OSRs in the requirements analysis phases. The low score in the Require-

Please cite this article in press as: K. Wnuk et al., Obsolete software requirements, Inform. Softw. Technol. (2012), http://dx.doi.org/10.1016/j.infsof.2012.12.001

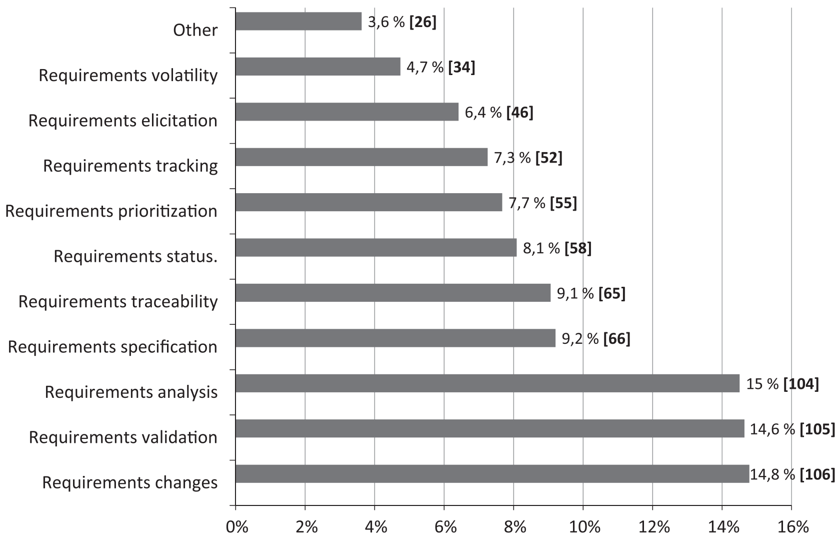

Fig. 13. Requirements lifecycle stages for addressing OSRs.

ments elicitation phase answer (6.42% of all answers) contradicts the viewpoint of Merola [15] who suggested managing obsolete software by continuous and timely market tracking and market trend change identification. This might seem to indicate that our respondents have difficulties understanding how OSRs could be managed, for example by finding and dismissing OSRs faster due to continuous elicitation depending on the accepted definition of OSRs.

Respondents working with *Agile software development* methodologies preferred to handle OSRs as a part of the *Requirements changes* phase, while respondents working in a *Waterfall* manner preferred the *Requirements validation* phase. This seems logical, as a part of agile methodology is to embrace change [4], while waterfall philosophy sees OSRs as something to be "handled" more formally in a development step (focusing on the specification and validation phases) [30].

Type of requirements engineering context (Fig. 7) did not seem to significantly influence answers for this question. Requirements analysis, validation, and changes phases seemed to be dominant for *MDRE* and *Bespoke or contract driven requirements engineering* alike. However, looking at company size and project duration, respondents from larger companies with longer projects focused on handling OSRs in specific phases, i.e., analysis and validation. This result seems reasonable as large projects usually require more extensive requirements analysis due to, e.g., the larger number of stakeholders involved and possible higher complexity of the system to be developed [51,64,84].

Looking at the answers given in the *Other* category, four answers suggested that OSRs should be managed in all phases of software lifecycle: one answer suggested all requirements management phases and one suggested quality assurance. Further investigation is needed.

4.9. Existing processes and practices regarding managing OSRs (RQ5)

When queried about the existing processes and practices for managing OSRs, 73.6% of all respondents (159) indicated that their requirements engineering process does not take OSRs into consideration. This result can be interpreted as clear evidence of a lack of methods regarding OSRs in industry and confirms the need for developing methods for managing OSRs. At the same time, some processes for managing OSRs do exist, as indicated by 26.4% (57) of our respondents. The list of processes and methods used by our respondents include:

- Reviews of requirements and requirements specifications (19 respondents)
- Using tools and "marking requirements as obsolete" (6 respondents)
- Requirements traceability (6 respondents)
- Discussing and prioritizing requirements with customers in an agile context (4 respondents)
- "Mark obsolete requirements as obsolete" (4 respondents)—these respondents did not indicate if using a tool or not.
- During the requirements management process by identifying OSRs (3 respondents)
- Moving OSRs into a separated section in the SRS (3 respondents)
- Through a change management process (2 respondents)
- During the requirements analysis process (1 respondent)
- Having a proprietary process (1 respondent)

The identified "categories" of processes and methods above provide further support for previous results from the survey. For example, the process of managing OSRs by requirements reviews overlaps the most popular way to identify OSRs (Fig. 11, Section 4.5), as indicated by our respondents. This would seem to indicate that manually reviewing requirements is dominant. Whether or not this is sufficient is another question which needs to be investigated further. The results confirm what was reported in Section 4.5, that automated methods for identification and management of OSRs are rare. Therefore, further research on scalable automatic methods for identification and management of OSRs is needed.

Some respondents provided names or descriptions of processes and methods used for managing OSRs. Those reported include:

- *Projective analysis through modeling*—Considered as a promising approach to study the complexity pertaining to systems of systems [88], it requires a skilled "process modeler" to seamlessly use the modeling paradigm. If and how the method could be applied for smaller projects, and particularly for identification and management of OSRs remains an open question. Also, the technique is used during the requirements analysis phase which has been considered a good phase for management of OSRs by our respondents (Fig. 13).

Please cite this article in press as: K. Wnuk et al., Obsolete software requirements, Inform. Softw. Technol. (2012), http://dx.doi.org/10.1016/j.infsof.2012.12.001

- *Hierarchical requirements' tables*—Specifying requirements on different abstraction levels is one of the fundamental techniques of requirements engineering that helps various stakeholders to understand requirements better [29]. Considered as one of the requirements specification techniques, this could be promising according to our respondents (Fig. 13). This method could be used to control OSRs to a certain degree as an overview of the requirements can be achieved, to some extent, through abstraction [80]. However, given large numbers of requirements, scalability of the method could be a problem.
- *Project governance*—Support project control activities considering the environment in which project management is performed [89]. By having greater time scope than ordinary project management, project governance could, according to our interpretation, be supportive in the task of continuous identification and management of OSRs.
- *Requirements tracking with risk management*—Although we consider tracking and risk management [29] as separated activities, combining them for the purpose of managing OSRs is an interesting alternative potential. In particular, the role of risk management in identification and management of OSRs should be further investigated, as the software risk management literature does not appear to mention OSRs as one of the software risks [90].
- *Requirements-based test plans*—Aligning requirements with verification, although challenging, could be considered critical in assuring that the developed software fulfills customers' needs. Creating test plans based on requirements that are up-to-date and properly reflect changing customer needs is considered a best practice in software projects [91]. OSRs may create mismatches and problems with alignment between requirements and test cases. The discovery of a test result that was correct, however presently wrong, can indicate that a requirement has become obsolete. We are, however, uncertain to what degree the practice ofwriting test plans based on requirements could help in identification and management of OSRs. The fact that test plans are based on requirements is, to us, independent of the fact that these requirements may simply be obsolete.
- *Commenting out obsolete code and updating requirements documents accordingly*—This technique of managing OSRs could be considered promising and should help to keep the requirements aligned with the newest version of the code. However, the technique seems to only consider implemented requirements that could be directly traced to the code level. Given the fact that many requirements (especially quality requirements) are cross-cutting and require implementation in several places [29] in the source code, an OSR may become even more cross cutting than before. In our opinion, it could be challenging to correctly map changes in the code to changes in requirements. Thus, mapping change in the code to changes in requirements could be part of a solution; however, it lacks the possibility to identify and remove OSRs prior to implementation.
- *Using requirements validation techniques to identify if requirements are no longer needed*—Validating requirements is fundamental for assuring that the customer needs were properly and correctly understood by the development organization [29]. In our opinion, this technique should be used together with customers who can confirm if the requirements are relevant. Our respondents also would like OSRs to be managed during requirements validation phase (Fig. 13). However, if requirements reviews are conducted in isolation from "customers" by e.g., requirements analysts, they could have difficulties in identifying which requirements are, or are about to become, obsolete. This is further aggravated if the development organization operates in a MDRE context.

Looking at the context factors of organizational size, development methodology, and respondent role, although no statistically significant correlations could be observed, some interesting points warrant mentioning. Respondents from smaller companies (<50 employees) to a larger degree had explicit practices for handling OSRs as compared to respondents from larger companies. This seems reasonable when looking at the methods for managing OSRs provided, where manual review methods were most frequent. Quispire et al. [65] mentioned that processes used in small software enterprises are often manually based and less automated.

Respondents who worked with *MDRE* projects (Fig. 7) reported having processes that take OSRs into consideration (34.3%), more often than respondents who worked with *Bespoke or contract driven requirements engineering* (26.5%) or *Outsourced projects* (15.8%) respectively (almost significant results with a *p*-value of 0.059, Table A.8a in [61]). One possible explanation for this could be high and constant requirements influx in MDRE contexts [2,51], combined with frequent changes to requirements dictated by rapidly changing market situations. This in turn is resulting in more requirements becoming obsolete, forcing the use of methods to manage OSRs.

Further statistical tests (Table A.8 in [61]) indicated a statistical significance between the roles of respondents and the existence of processes to manage OSRs ($p = 0.0012$). There was also a moderate association (Cramer's $V = 0.345$) between the respondents' roles and the existence of requirements engineering processes that take OSRs into account. From the cross-tabulation table between the respondents' roles and the existence of OSRs handling process (Table A.9 in [61]) we can see that the respondents who worked in management positions (project and product managers) were more likely to utilize OSRs handling method compared to respondents who worked in software development roles, as developers.

Further, the presence of a method or process that considers OSRs seems to decrease the negative impact of OSRs among our respondents, as 50% of the respondents who deemed OSRs *Trivial* confirmed having a process of managing OSRs (Section 4.3). Moreover, as requirements engineers as well as product and project managers usually work more with requirements engineering related tasks than software development roles, it appears to be logical that more methods of managing OSRs are reported among the management roles.

4.10. Summary of results

The results from the study are summarized in the following points:

- Our respondents defined an OSR (RQ1) as: "a software requirement (implemented or not) that is no longer required for the current release or future releases, and it has no or little business value for the potential customers or users of a software artifact." This definition seems to be homogeneous among our respondents (with a small exception for the respondents who used *RUP* methodologies).
- OSRs constitute a significant challenge for companies developing software intensive products, with the possible exception of companies involved in the service domain. The phenomenon of OSRs is considered serious by 84.3% of our respondents (RQ2). At the same time 73.6% of our respondents reported having no process for handling obsolete software requirements (RQ5).
- Requirements related to standards and laws are the least likely to become obsolete, while inconsistent and ambiguous requirements are the most likely to become obsolete (RQ3). Moreover,

requirements originating from domain experts were less likely to become obsolete than requirements originating from customers or (internal) developers.

- OSRs identification is predominantly a manual activity, and less than 10% of the respondents reported having any automated functionality. They also confirmed that automatic identification of OSR is difficult which suggests research opportunities in creating automated methods for OSR identification and management (RQ4).
- The identified OSRs should, according to more than 60% of the survey answers, be kept in the requirements document or the database, but tagged as obsolete. Deleting OSRs is not a desired behavior (RQ5). Most respondents opted for keeping the OSRs for purposes of reference and traceability, which seems to indicate that the identification of OSRs is not the only action, but a wish to potentially use the OSRs to minimize repeated work (e.g. specifying new requirements that are the same or similar to already identified OSRs). This is especially relevant in the MDRE context where "good ideas" can resurface as proposed by, for example internal developers.
- Although there exist some methods and tool support for the identification and handling of OSRs, a clear majority of the respondents indicated no use of methods or tools to support them. Rather, ad hoc and manual process seemed to dominate (RQ5). Moreover, even when the identification of OSRs was deemed central (e.g., for respondents working in longer duration projects), only some tool support and automation was present (mostly for bespoke projects), but even here manual processes and routines dominated (Section 4.5).
- Project managers and product managers indicate that they always find OSRs in their work (Section 4.5), even if many of the respondents do not actively look for them.
- OSRs are more likely to affect *Large-scale requirements* and *Very large-scale requirements* projects (RQ6). Larger projects (hundreds of requirements) tend to have larger issues related to the presence of OSRs, and there seems to be a correlation between impact severity and project size (amount of requirements). OSRs seem to have a somewhat larger impact on projects in a MDRE context as compared to bespoke or contract driven development (Section 4.7.2). However, for very-large projects (over 10,000 requirements) all respondents, independent of context factors, agree that the potential impact of OSRs was substantial.
- According to the respondents, OSRs should first of all be handled during the *Requirements analysis* and *Requirements validation* phases (RQ7). At the same time, less than 5% of the answers indicate that OSRs should be managed as a part of requirements volatility handling which supports the distinction between volatility and the phenomenon of OSRs as such. Finally, our respondents suggested that *Requirements elicitation* is not the best phase to manage OSRs.
- Latency may not be the main determinant of OSRs becoming a problem. Rather, the results point to the lack of methods and routines for actively handling OSRs as a central determinant. This would imply that claimed low latency development models, like agile, has and can have problems with OSRs.

5. Conclusions and further work

Although the phenomenon of obsolete software requirements and its negative effects on project timelines and the outcomes have been reported in a number of publications [9,13–15,7], little empirical evidence exists that explicitly and exhaustively investigates the phenomenon of OSRs.

In this paper, we report results from a survey conducted among 219 respondents from 45 countries exploring the phenomenon of OSRs by: (1) eliciting a definition of OSRs as seen by practitioners in industry, (2) exploring ways to identify and manage OSRs in requirements documents and databases, (3) investigating the potential impact of OSRs, (4) exploring effects of project context factors on OSRs, and (5) defining what types of requirements are most likely to become obsolete.

Our results clearly indicate that OSRs are a significant challenge for companies developing software systems—OSRs were considered serious by 84.3% of our respondents. Moreover, a clear majority of the respondents indicated no use of methods or tools to support identification and handling OSRs, and only 10% of our respondents reported having automated support. This indicates that there is a need for developing automated methods or tools to support practitioners in the identification and management of OSRs. These proposed methods need to have effective mechanisms for storing requirements tagged as OSRs, enabling the use of the body of OSRs as decision support for future requirements and their analysis. This could potentially enable automated regression analysis of active requirements, continuously identifying candidates for OSRs, and flagging them for analysis.

Although manually managing OSRs is currently the dominant procedure, which could be sufficient in small projects, research effort should be directed towards developing scalable methods for managing OSRs—methods that scale to a reality that is often characterized by large numbers of requirements and a continuous and substantial influx of new requirements. The reality facing many product development organizations developing software intensive systems today is that OSRs are a problem, and as the amount and complexity of software increases so will the impact of OSRs.

References

[1] M. DeBellis, C. Haapala, User-centric software engineering, IEEE Exp. 10 (1) (1995) 34–41, http://dx.doi.org/10.1109/64.391959.
[2] B. Regnell, S. Brinkkemper, Market-driven requirements engineering for software products, in: A. Aurum, C. Wohlin (Eds.), Engineering and Managing Software Requirements, Springer, Berlin Heidelberg, 2005, pp. 287–308.
[3] T. Gorschek, S. Fricker, K. Palm, S. Kunsman, A lightweight innovation process for software-intensive product development, IEEE Softw. 27 (1) (2010) 37–45, http://dx.doi.org/10.1109/MS.2009.164.
[4] B. Ramesh, L. Cao, R. Baskerville, Agile requirements engineering practices and challenges: an empirical study, Inf. Syst. J. 20 (5) (2010) 449–480.
[5] T. Gorschek, M. Svahnberg, A. Borg, A. Loconsole, J. Börstler, K. Sandahl, M. Eriksson, A controlled empirical evaluation of a requirements abstraction model, Inf. Softw. Technol. 49 (2007) 790–805, http://dx.doi.org/10.1016/j.infsof.2006.09.003.
[6] T. Gorschek, P. Garre, S.B.M. Larsson, C. Wohlin, Industry evaluation of the requirements abstraction model, Requir. Eng. 12 (2007) 163–190, http://dx.doi.org/10.1007/s00766-007-0047-z. <http://dl.acm.org/citation.cfm?id=1391227.1391230>.
[7] L. Cao, B. Ramesh, Agile requirements engineering practices: an empirical study, IEEE Softw. 25 (1) (2008) 60–67, http://dx.doi.org/10.1109/MS.2008.1.
[8] K. Wnuk, B. Regnell, L. Karlsson, What happened to our features? Visualization and understanding of scope change dynamics in a large-scale industrial setting, in: 17th IEEE International Requirements Engineering Conference, RE '09, 2009, pp. 89–98. http://dx.doi.org/10.1109/RE.2009.32.
[9] C. Hood, S. Wiedemann, S. Fichtinger, U. Pautz, Requirements Management The Interface Between Requirements Development and All Other Systems Engineering Processes, Springer, Berlin, 2008, http://dx.doi.org/10.1007/978-3-540-68476-3.
[10] J. Chen, R. Reilly, G. Lynn, The impacts of speed-to-market on new product success: the moderating effects of uncertainty, IEEE Trans. Eng. Manag. 52 (2) (2005) 199–212, http://dx.doi.org/10.1109/TEM.2005.844926.
[11] C. Wohlin, M. Xie, M. Ahlgren, Reducing time to market through optimization with respect to soft factors, in: Proceedings of 1995 IEEE Annual International Engineering Management Conference, Global Engineering Management: Emerging Trends in the Asia Pacific, 1995, pp. 116–121. http://dx.doi.org/10.1109/IEMC.1995.523919.
[12] P. Sawyer, Packaged software: challenges for re, in: Proceedings of the Sixth International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ 2000), 2000, pp. 137–142.
[13] D. Murphy, D. Rooney, Investing in agile: aligning agile initiatives with enterprise goals, Cutter IT J. 19 (2) (2006) 6–13.
[14] J. Stephen, J. Page, J. Myers, A. Brown, D. Watson, I. Magee, System Error Fixing the Flaws in Government It, Tech. Rep. 6480524, Institute for Government, London, 2011.

Please cite this article in press as: K. Wnuk et al., Obsolete software requirements, Inform. Softw. Technol. (2012), http://dx.doi.org/10.1016/j.infsof.2012.12.001

[15] L. Merola, The cots software obsolescence threat, in: Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, 2006, p. 7. http://dx.doi.org/10.1109/ICCBSS.2006.29.
[16] C. Hood, S. Wiedemann, S. Fichtinger, U. Pautz, Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes, Springer-Verlag, Berlin, 2008.
[17] J. Savolainen, I. Oliver, M. Mannion, H. Zuo, Transitioning from product line requirements to product line architecture, in: 29th Annual International Computer Software and Applications Conference, COMPSAC, vols. 1 and 2, 2005, pp. 186–195. http://dx.doi.org/10.1109/COMPSAC.2005.160.
[18] F. Loesch, E. Ploederoeder, Restructuring variability in software product lines using concept analysis of product configurations, in: 11th European Conference on Software Maintenance and Reengineering, CSMR '07, 2007, pp. 159–170. http://dx.doi.org/10.1109/CSMR.2007.40.
[19] M. Mannion, O. Lewis, H. Kaindl, G. Montroni, J. Wheadon, Representing requirements on generic software in an application family model, in: Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability, Springer-Verlag, London, UK, 2000, pp. 153–169. <http://dl.acm.org/citation.cfm?id=645546.656064>.
[20] T. Herald, D. Verma, C. Lubert, R. Cloutier, An obsolescence management framework for system baseline evolution perspectives through the system life cycle, Syst. Eng. 12 (2009) 1–20, http://dx.doi.org/10.1002/sys.v12:1. <http://dl.acm.org/citation.cfm?id=1507335.1507337>.
[21] S. Robertson, J. Robertson, Mastering the Requirements Process, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.
[22] C. Iacovou, A. Dexter, Turning around runaway information technology projects, IEEE Eng. Manag. Rev. 32 (4) (2004) 97–112, http://dx.doi.org/10.1109/EMR.2004.25141.
[23] T. DeMarco, T. Lister, Risk management during requirements, IEEE Softw. 20 (5) (2003) 99–101.
[24] D.X. Houston, G.T. Mackulak, J.S. Collofello, St ochastic simulation of risk factor potential effects for software development risk management, J. Syst. Softw. 59 (3) (2001) 247–257, http://dx.doi.org/10.1016/S0164-1212(01)00066-8. <http://www.sciencedirect.com/science/article/pii/S0164121201000668>.
[25] G.P. Kulk, C. Verhoef, Quantifying requirements volatility effects, Sci. Comput. Program. 72 (3) (2008) 136–175, http://dx.doi.org/10.1016/j.scico.2008.04.003.
[26] D. Zowghi, N. Nurmuliani, A study of the impact of requirements volatility on software project performance, in: Asia-Pacific Software Engineering Conference, vol. 3, 2002. http://doi.ieeecomputersociety.org/10.1109/APSection 2002.1182970.
[27] A. Loconsole, J. Borstler, An industrial case study on requirements volatility measures, in: Asia-Pacific Software Engineering Conference, 2005, pp. 1–8. <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1607159>.
[28] K.E. Wiegers, Software Requirements, second ed., Microsoft Press, Redmond, WA, USA, 2003.
[29] S. Lauesen, Software Requirements – Styles and Techniques, Addison-Wesley, 2002.
[30] G. Kotonya, I. Sommerville, Requirements Engineering, John Wiley & Sons, 1998.
[31] I. Sommerville, P. Sawyer, Requirements Engineering: A Good Practice Guide, John Wiley & Sons, 1997.
[32] A. Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, John Wiley, 2009.
[33] A. Aurum, C. Wohlin, Engineering and Managing Software Requirements, Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 2005.
[34] IEEE, IEEE Recommended Practice for Software Requirements Specifications, 830-1998. <http://standards.ieee.org/findstds/standard/830-1998.html> (September 1997).
[35] S.E. Institute, Capability Maturity Model Integration (CMMI), Version 1.3. <http://www.sei.cmu.edu/cmmi/solutions/info-center.cfm> (last visited, December 2011).
[36] C. Hood, S. Wiedemann, S. Fichtinger, U. Pautz, Change management interface, in: Requirements Management, Springer, Berlin Heidelberg, 2008, pp. 175–191.
[37] I. Legodi, M.-L. Barry, The current challenges and status of risk management in enterprise data warehouse projects in south africa, in: Technology Management for Global Economic Growth (PICMET), Proceedings of PICMET '10, 2010, pp. 1–5.
[38] G. Anthes, No more creeps! Are you a victim of creeping user requirements?, Computerworld 28 (18) (1994) 107–110
[39] H. Ruel, T. Bondarouk, S. Smink, The waterfall approach and requirement uncertainty: an in-depth case study of an enterprise systems implementation at a major airline company, Int. J. Technol. Proj. Manage. (USA) 1 (2) (2010/04/) 43–60 (waterfall approach; requirement uncertainty; enterprise systems implementation; major airline company; project management). http://dx.doi.org/10.4018/jitpm.2010040103.
[40] Software Engineering Body of Knowledge (SWEBOK), Angela Burgess, EUA, 2004. <http://www.swebok.org/>.
[41] M. Takahashi, Y. Kamayachi, An empirical study of a model for program error prediction, IEEE Trans. Softw. Eng. 15 (1989) 82–86. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/32.21729..
[42] S. Harker, K. Eason, J. Dobson, The change and evolution of requirements as a challenge to the practice of software engineering, in: Proceedings of IEEE International Symposium on Requirements Engineering, 1993, pp. 266–272. http://dx.doi.org/10.1109/ISRE.1993.324847.
[43] S. McGee, D. Greer, A software requirements change source taxonomy, in: Fourth International Conference on Software Engineering Advances, ICSEA '09, 2009, pp. 51–58. http://dx.doi.org/10.1109/ICSEA.2009.17.
[44] S. Easterbrook, J. Singer, M.-A. Storey, D. Damian, Selecting empirical methods for software engineering research, in: F. Shull, J. Singer, D.I.K. Sjberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer, London, 2008, pp. 285–311.
[45] J. Singer, S.E. Sim, T.C. Lethbridge, Software engineering data collection for field studies, in: F. Shull, J. Singer, D.I.K. Sjberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer, London, 2008, pp. 9–34.
[46] C. Dawson, Projects in Computing and Information Systems: A Student's Guide, Addison Wesley, 2005.
[47] R.A.P.L.M. Rea, Designing and Conducting Survey Research: A Comprehensive Guide, Jossey-Bass, San Francisco, CA, 94103-1741, 1005.
[48] A. Strauss, J. Corbin, Basics of Qualitative Research: Grounded Theory Procedures and Techniques, Sage Publications, Newbury Park, California, 1990.
[49] Wikipedia, Likert Scale. <http://en.wikipedia.org/wiki/Likert_scale> (last visited, December 2011).
[50] K. Wnuk, The Survey Questionnaire, <http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/AppendixB_SurveyQuestions.pdf> (last visited December 2011).
[51] B. Regnell, R.B. Svensson, K. Wnuk, Can we beat the complexity of very large-scale requirements engineering?, in: Proceedings of the 14th International Conference on Requirements Engineering: Foundation for Software Quality, REFSQ '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp 123–128.
[52] S. Monkey, Survey Monkey Webpage. <http://www.surveymonkey.net> (last visited December 2011).
[53] Linkedin, The Linkedin. <http://www.linkedin.com/> (last visited, December 2011).
[54] K. Wnuk, The Full List of Mailing Lists. <http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/ListOfDiscussionGroups.pdf> (last visited December 2011).
[55] K. Wnuk, The Full List of Tool Vendors. <http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/ListOfVendors.pdf> (last visited December 2011).
[56] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.
[57] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Proceeding of the 33rd International Conference on Software Engineering, ICSE '11, ACM, New York, NY, USA, 2011, pp. 1–10, http://dx.doi.org/10.1145/1985793.1985795.
[58] S. Nakagawa, A farewell to Bonferroni: the problems of low statistical power and publication bias, Behav. Ecol. 15 (6) (2004) 1044–1045.
[59] T.C. Lethbridge, S.E. Sim, J. Singer, Studying software engineers: data collection techniques for software field studies, Empirical Softw. Eng. 10 (2005) 311–341, http://dx.doi.org/10.1007/s10664-005-1290-x.
[60] S. Siegel, N.J. Castellan, Nonparametric Statistics for the Behavioral Sciences, second ed., McGraw-Hill, 1998.
[61] K. Wnuk, The Appendix with Analysis. <http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/AppendixA_Analysis.pdf> (last visited December 2011).
[62] K. Wnuk, The Full List of Countries. <http:// www.fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/COUNTRIES.pdf> (last visited December 2011).
[63] IBM, The Description of the Method. <http://www-01.ibm.com/software/awdtools/rup/> (last visited December 2011).
[64] B. Berenbach, D.J. Paulish, J. Kazmeier, A. Rudorfer, Software & Systems Requirements Engineering: In Practice, Pearson Education Inc., 2009.
[65] A. Quispe, M. Marques, L. Silvestre, S. Ochoa, R. Robbes, Requirements engineering practices in very small software enterprises: a diagnostic study, in: 2010 XXIX International Conference of the Chilean Computer Science Society (SCCC), 2010, pp. 81–87. http://dx.doi.org/10.1109/SCCC.2010.35.
[66] M. Bano, N. Ikram, Issues and challenges of requirement engineering in service oriented software development, in: Fifth International Conference on Software Engineering Advances (ICSEA), 2010, pp. 64–69. http://dx.doi.org/10.1109/ICSEA.2010.17.
[67] M. Kossmann, A. Gillies, M. Odeh, S. Watts, Ontology-driven requirements engineering with reference to the aerospace industry, in: Second International Conference on the Applications of Digital Information and Web Technologies, 2009. ICADIWT '09, pp. 95–103. http://dx.doi.org/10.1109/ICADIWT.2009.5273953.
[68] B. Boehm, Requirements that handle IKIWISI, COTS, and rapid change, Computer 33 (7) (2000) 99–102, http://dx.doi.org/10.1109/2.869384.
[69] I. Sommerville, Software Engineering, Addison–Wesley, 2007.
[70] W. Curtis, H. Krasner, V. Shen, N. Iscoe, On building software process models under the lamppost, in: Proceedings of the 9th International Conference on Software Engineering (ICSE 1987), 1987, pp. 96–103.
[71] T. Gorschek, A.M. Davis, Requirements engineering: in search of the dependent variables, Inf. Softw. Technol. 50 (2008) 67–75, http://dx.doi.org/10.1016/j.infsof.2007.10.003. <http://dl.acm.org/citation.cfm?id=1324618.1324710>.
[72] A. Aurum, C. Wohlin, Requirements engineering: setting the context, in: A. Aurum, C. Wohlin (Eds.), Engineering and Managing Software Requirements, Springer, Berlin Heidelberg, 2005, pp. 1–15, http://dx.doi.org/10.1007/3-540-28244-0_1.
[73] P. Bourque, R. Dupuis, Guide to the Software Engineering Body of Knowledge 2004 Version, SWEBOK.

[74] N. Nurmuliani, D. Zowghi, S. Powell, Analysis of requirements volatility during software development life cycle, in: Proceedings of Australian, Software Engineering Conference, 2004, pp. 28–37. http://dx.doi.org/10.1109/ASWEC.2004.1290455.

[75] X. Shan, G. Jiang, T. Huang, The study on knowledge transfer of software project requirements, in: 2010 International Conference on Biomedical Engineering and Computer Science (ICBECS), 2010, pp. 1–4. http://dx.doi.org/10.1109/ICBECS.2010.5462314.

[76] W.N. Robinson, S.D. Pawlowski, Managing requirements inconsistency with development goal monitors, IEEE Trans. Softw. Eng. 25 (6) (1999) 816–835.

[77] A. Russo, B. Nuseibeh, J. Kramer, Restructuring requirements specifications for inconsistency analysis: a case study, in: Third Intl Conf. Requirements Engineering, IEEE CS Press, Los Alamitos, Calif, IEEE Computer Society Press, 1998, pp. 51–60.

[78] A.C.W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, B. Nuseibeh, Inconsistency handling in multiperspective specifications, IEEE Trans. Softw. Eng. 20 (8) (1994) 569–578, http://dx.doi.org/10.1109/32.310667.

[79] S. Easterbrook, What is requirements engineering? July 2004. <http://www.cs.toronto.edu/sme/papers/2004/FoRE-chapter01-v7.pdf>.

[80] T. Gorschek, C. Wohlin, Requirements abstraction model, Requir. Eng. 11 (2005) 79–101, http://dx.doi.org/10.1007/s00766-005-0020-7. <http://dl.acm.org/citation.cfm?id=1107677.1107682>.

[81] J. Kabbedijk, B.R.K. Wnuk, S. Brinkkemper, What decision characteristics influence decision making in market-driven large-scale software product line development? in: Product Line Requirements Engineering and Quality, 2010, pp. 42–53.

[82] R. Kohl, Changes in the requirements engineering processes for cots-based systems, in: IEEE International Conference on Requirements Engineering, 2001, p. 0271. http://dx.doi.org/10.1109/ISRE.2001.948575.

[83] H. Holmstrom, E.O. Conchuir, P.J. Agerfalk, B. Fitzgerald, Global software development challenges: a case study on temporal, geographical and socio-cultural distance, in: International Conference on Global Software Engineering, ICGSE '06, 2006, pp. 3–11. http://dx.doi.org/10.1109/ICGSE.2006.261210.

[84] S. Buhne, G. Halmans, K. Pohl, M. Weber, H. Kleinwechter, T. Wierczoch, Defining requirements at different levels of abstraction, in: Proceedings of 12th IEEE International, Requirements Engineering Conference, 2004, pp. 346–347. http://dx.doi.org/10.1109/ICRE.2004.1335694.

[85] J. Natt Och Dag, T. Thelin, B. Regnell, An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development, Empirical Softw. Eng. 11 (2006) 303–329, http://dx.doi.org/10.1007/s10664-006-6405-5. <http://dl.acm.org/citation.cfm?id=1120556.1120562>.

[86] S. Konrad, M. Gall, Requirements engineering in the development of large-scale systems, in: Proceedings of the 2008 16th IEEE International Requirements Engineering Conference, RE '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 217–222, http://dx.doi.org/10.1109/RE.2008.31.

[87] D. Šmite, C. Wohlin, T. Gorschek, R. Feldt, Empirical evidence in global software engineering: a systematic review, Empirical Softw. Eng. 15 (2010) 91–118, http://dx.doi.org/10.1007/s10664-009-9123-y.

[88] W. Anderson, P.J. Boxer, L. Brownsword, An examination of a Structural Modeling Risk Probe Technique, Tech. Rep. CMU/SEI-2006-SR-017, Software Engineering Institute, Carnegie Mellon University, 2008.

[89] M. Bekker, H. Steyn, The impact of project governance principles on project performance, in: Management of Engineering Technology, 2008. PICMET 2008. Portland International Conference on, 2008, pp. 1324–1330. http://dx.doi.org/10.1109/PICMET.2008.4599744.

[90] B. Boehm, Software risk management: principles and practices, Software, IEEE 8 (1) (1991) 32–41, http://dx.doi.org/10.1109/52.62930.

[91] K. Pohl, Requirements Engineering: Fundamentals, Principles, and Techniques, first ed., Springer Publishing Company, Incorporated, 2010.