Poznań University of Technology
Institute of Computing Science

SUPPORTING NON-FUNCTIONAL REQUIREMENTS ELICITATION WITH TEMPLATES

Sylwia Kopczyńska

A dissertation submitted to
the Council of the Faculty of Computing
in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Supervisor
Jerzy Nawrocki, PhD, Dr Habil.
Auxiliary supervisor
Mirosław Ochodek, PhD

Poznań, Poland
2018

ABSTRACT

Non-functional requirements (NFRs) state conditions under which functionality is useful (they concern performance, security, availability, etc.). Unfortunately, they are frequently neglected, especially those NFRs that are difficult to write or seem ostensibly obvious. Such behavior is an important risk factor in software projects as, in many cases, improper management of NFRs is one of the root causes of project failures.

One of the approaches to support elicitation of NFRs is using a catalog of *templates*. Templates are natural language statements with some parameters (gaps) to fill in and optional parts to select during elicitation.

Many authors say that templates improve consistency and testability of requirements, and that they reduce ambiguity. Although experts formulate these claims, some recent studies show that practitioners are afraid of using NFR templates in their projects. It is not clear for them what are the benefits and costs of using NFR templates.

In the traditional approaches to software development, the necessity to elicit NFRs seemed rather obvious. Recently, agile approaches have gained popularity but it would be vain to look for agile practices that explicitly refer to NFRs. Therefore, a question arises whether NFRs are still important.

Another issue is user feedback left in online app stores. This feedback has been found a promising source of functional requirements. Thus, it would be valuable to check, if user feedback can also be useful for elicitation of NFRs.

The aim of the thesis is to provide knowledge that would allow managers of software projects to make an informed decision whether to use or not to use a catalog of NFR templates. The focus is on the following research questions:

RQ1. *How important is it for agile practitioners to have NFRs specified in their software projects?*

We have conducted a survey to get to know how agile practitioners perceive the importance of NFRs. It has proved that ❶ having NFRs specified in a software project is perceived by 77% of agile practitioners as an important development practice. Moreover, ❷ the more experienced the practitioners, the higher the opinion on the importance of NFRs.

RQ2. *What is the usefulness of catalog of NFR templates?*

Usefulness of NFR templates was evaluated in two studies: (1) a case study with 7 software projects, and (2) a controlled experiment with 107 people eliciting NFRs for a fictitious e-commerce application.

In the case study ❸ the percentage of initially elicited NFRs that remained valid till the end of each project was above 80% for all but one project, ❹ the percentage of final NFRs that were elicited at the beginning of each project was for almost all the projects above 80%, ❺ the percentage of initially elicited NFRs that have been derived from the templates was for almost all the projects above 60%.

From the controlled experiment it follows that ❻ using template-supported elicitation inexperienced elicitors provided NFRs that were less ambiguous, more detailed, more testable and more complete than those elicited with the *ad hoc* approach for both individuals and teams.

In both studies ❼ over 80% of respondents who used template-supported elicitation regarded NFR templates as useful.

RQ3. *How the benefits and costs of using catalog change during catalog evolution?*

When an organization runs a sequence of projects, their catalog should be updated at the end of each project whenever missing or 'too narrow' templates are identified. We analyzed how catalog characteristics concerning its benefits and costs change in the evolution driven by the mentioned updates. In the study 41 requirements specifications with the total of 2,231 NFRs were used, coming from industry software projects. One of the main observations is that after "learning" templates from about 40 projects one can expect ❽ at least 75% of NFRs of the next project to be derived from the templates of such catalog, and ❾ up to 10% of the templates may need to be modified or added.

RQ4. *To what extent user feedback in app stores is a good source of NFRs and their templates?*

User reviews of software applications stored in three app stores (Apple App Store, Google Play, and Amazon Appstore) were analyzed. ❿ Over 45% of the user reviews left in the app stores concerned non-functional aspects and they could be used as a basis for formulating NFRs and their templates.

From the presented results it follows that NFRs are important even in agile projects. Their elicitation is a difficult task, especially for inexperienced elicitors. In this context, a catalog of NFR templates can be really useful. Creating such a catalog according to the evolutionary paradigm seems very practical. However, only mature catalog can bring full benefits, i.e., high coverage of requirements by the templates, and little maintenance effort. It seems that about 40 projects are needed to make a catalog of NFR templates mature. Moreover, when working on new releases of a software product, it could be beneficial to take into consideration user feedback left in online app stores.

Politechnika Poznańska
Instytut Informatyki

WSPOMAGANIE POZYSKIWANIA WYMAGAŃ POZAFUNKCJONALNYCH Z WYKORZYSTANIEM SZABLONÓW WYMAGAŃ

Sylwia Kopczyńska

Rozprawa doktorska

Przedłożono Radzie Wydziału Informatyki
Politechniki Poznańskiej

Promotor
dr hab. inż. Jerzy Nawrocki, prof. nadzw.
Promotor pomocniczy
dr inż. Mirosław Ochodek

Poznań 2018

STRESZCZENIE

Wymaganie pozafunkcjonalne (ang. *non-functional requirement*, w skrócie NFR) opisuje warunek, który ma być spełniony, aby funkcje realizowane przez system były użyteczne (może dotyczyć wydajności, bezpieczeństwa, dostępności itp.). Niestety NFRy są często lekceważone, w szczególności te, które są trudne do wyspecyfikowania lub wydają się oczywiste. Takie zachowanie to istotny czynnik ryzyka, ponieważ nieprawidłowe zarządzanie NFRami wielokrotnie okazało się jedną z głównych przyczyn niepowodzeń projektów informatycznych.
Jednym z podejść, które wspiera pozyskiwanie NFRów jest katalog *szablonów*. Szablon NFRa jest to wzorzec zdania w języku naturalnym, który zawiera parametry (puste miejsca) do wypełnienia i części opcjonalne, czyli fragmenty do wyboru.
Wielu ekspertów twierdzi, że szablony poprawiają spójność wymagań, zwiększają łatwość ich testowania oraz zmniejszają ich wieloznaczność. Mimo to—jak wynika z ostatnich badań—praktycy obawiają się stosowania szablonów NFRów. Nie jest dla nich jasne jakie są korzyści i koszty związane z korzystaniem z szablonów.
W tradycyjnych podejściach do wytwarzania oprogramowania konieczność zbierania NFRów w zasadzie nie podlegała dyskusji. Jednakże ostatnio zwinne podejścia zyskują na popularności i próżno byłoby szukać wśród zalecanych przez nie praktyk tych, które bezpośrednio odnoszą się do NFRów. Powstaje zatem pytanie, czy NFRy są nadal istotne.
Innym ważnym zgadnieniem są opinie zostawiane przez użytkowników sklepów *on–line* z aplikacjami. Okazały się one obiecującym źródłem wymagań funkcjonalnych. Warto by było dowiedzieć się, czy te opinie mogą być także pomocne przy formułowaniu NFRów.

Celem pracy jest dostarczenie wiedzy, która pozwoliłaby kadrze zarządzającej podejmować świadome decyzje dotyczące stosowania (lub nie stosowania) katalogu szablonów NFRów w projektach informatycznych. W pracy skupiono się na następujących pytaniach badawczych:

RQ1. *Jak ważne dla praktyków jest specyfikowanie NFRów w zwinnych projektach informatycznych?*
Przeprowadziliśmy ankietę, aby dowiedzieć się, jak praktycy postrzegają znaczenie NFRów w zwinnych projektach informatycznych. Dowiedzieliśmy się, że ❶ specyfikowanie NFRów w zwinnym projekcie informatycznym jest postrzegane jako ważne przez 77% praktyków . Co więcej, ❷ im bardziej doświadczeni praktycy, tym bardziej pozytywną mieli opinię na temat przydatności tej praktyki.

RQ2. *Jaka jest użyteczność katalogu szablonów NFRów?*
Użyteczność szablonów NFRów została oceniona w: (1) studium przypadku z 7 projektami informatycznymi i (2) eksperymencie kontrolowanym ze 107 osobami z małym doświadczeniem, które pozyskiwały wymagania dla systemu typu e-commerce.
W naszym studium przypadku ❸ procent początkowo pozyskanych wymagań, które pozostały ważne do końca projektu był na poziomie 80%, ❹ procent końcowych wymagań, które pojawiły się na początku projektu, był powyżej 80% w prawie wszystkich projektach, ❺ procent początkowo pozyskanych NFRów, które zostały wywiedzione z szablonów był powyżej 60% w prawie wszystkich projektach.
Z naszego eksperymentu wynika, że ❻ przy użyciu szablonów osoby dostarczyły wymagania, które są mniej niejednoznaczne, bardziej szczegółowe, lepsze z punktu widzenia testowalności i bardziej kompletne niż te pozyskane *ad hoc* zarówno jeśli pracowały one same, jak i w zespole.
W obu badaniach ❼ ponad 80% ankietowanych osób, które pozyskiwało NFRy ze wsparciem szablonów postrzegało je jako użyteczne.

RQ3. *Jak się zmieniają zyski i koszty związane z katalogiem NFRów w trakcie jego ewolucji?*
Gdy organizacja realizuje sekwencję projektów, ich katalog szablonów NFRów powinien być aktualizowany na końcu każdego projektu wtedy, gdy zostaną wykryte brakujące lub "zbyt wąskie" szablony. Przeanalizowaliśmy, jak zmieniają się cechy katalogu dotyczące związanych z nim korzyści i kosztów w czasie ewolucji wynikającej z wymienionych aktualizacji. W badaniu wykorzystaliśmy 41 specyfikacji wymagań z 2.231 wymaganiami pochodzącymi z projektów realizowanych przez firmy informatyczne. Po katalogu, który powstanie w wyniku ewolucji z użyciem około 40 projektów można się spodziewać, że ❽ co najmniej 75% NFRów kolejnego projektu będzie można sformułować przy pomocy szablonów zgromadzonych w tym katalogu i ❾ do 10% szablonów będzie należało zmodyfikować lub dodać na podstawie tego projektu.

RQ4. *W jakim stopniu opinie klientów sklepów z aplikacjami są dobrym źródłem NFRów i ich szablonów?*
Przeprowadziliśmy analizę opinii użytkowników zamieszczonych w trzech sklepach z aplikacjami (Apple App Store, Google Play i Amazon Appstore). ❿ Ponad 45% opinii użytkowników pozostawionych w sklepach z aplikacjami dotyczyło aspektów pozafunkcjonalnych, które mogą być wykorzystane jako podstawa do formułowania NFRów i ich szablonów.

Z przedstawionych wyników badań można wywnioskować, że specyfikowanie NFRów w projektach informatycznych jest istotne, nawet w tych realizowanych w zwinnych podejściach. Niestety, pozyskiwanie NFRów nie jest łatwym zadaniem, szczególnie dla niedoświadczonych osób. W tym kontekście, katalog szablonów NFRów może okazać się bardzo użyteczny. Wspomaganie pozyskiwania NFRów szablonami pozytywnie wpływa na jakość pojedynczych wymagań oraz kompletność i stabilność zbioru wymagań. Tworzenie katalogu szablonów NFRów zgodnie z paradygmatem ewolucyjnym wydaje się bardzo praktyczne. Jednakże tylko dojrzały katalog może dać pełne korzyści, czyli bardzo dużą część NFRów wywiedzionych z szablonów i niewielkie koszty utrzymania katalogu. Wydaje się, że ok. 40 projektów jest potrzebnych, aby katalog szablonów NFRów stał się dojrzały. Ponadto, przy pozyskiwaniu NFRów warto korzystać z wiedzy zawartej w opiniach użytkowników sklepów z aplikacjami.

List of papers

Symbols:

- (J) or (C) = type of the paper: J=Journal or C = Conference
- IF5 = the 5-year impact factor according to Journals Citations Report (JCR) 2017
- Rank = the rank in the ISI ranking of the Software Engineering category according to JCR 2017
- SNIP = measures actual citations received relative to citations expected for the serial's subject field (Computer Science) according to Scopus, year 2017
- CORE = the score in the ranking created by Australian deans and the Australian Computing Research and Education Association of Australasia
- Qualis = the score in the conference ranking published by the Brazilian ministry of education and based on the H-index as performance measure for conferences
- Ministry = the points assigned by Polish Ministry of Science and Higher Educations as of 2017
- GS = citations according to Google Scholar, visted on 25/07/2018
- ** = in the multidisciplinary category
- () = if the data describing the journals has changed since the publication of a given paper first the value of the measure is given for the publication year than in () the value for 2017 is provided

List of papers by the author that directly concern the PhD thesis

1. (J) S. Kopczyńska, J. Nawrocki, M. Ochodek, "*An Empirical Study on Catalog of Non-functional Requirement Templates: Usefulness and Maintenance Issues*", Information and Software Technology, 103, 75-91, 2018
   IF5 = 2.768, Rank=16/104, SNIP=2.582, Ministry=35, GS=0
2. (J) M. Ochodek, S. Kopczyńska, "*Perceived importance of agile requirements engineering practices–A survey*", Journal of Systems and Software, 143, 29-43, 2018
   IF5 = 2.401, Rank=19/104, SNIP=1.868, Ministry=35, GS=2
3. (J) A. Trendowicz, S. Kopczyńska, "*Adapting Multi-Criteria Decision Analysis for Assessing the Quality of Software Products. Current Approaches and Future Perspectives*", Advances in Computers, 93, 153-226, 2014
   IF5=0.636(0.889), Rank=99/104(44/104), SNIP=1.574 (na), Ministry=15, GS=8
4. (J) S. Kopczyńska, M. Maćkowiak, J. Nawrocki, "*Structured meetings for non-functional requirements elicitation*", Foundations of Computing and Decision Sciences 36 (1), 35-56, 2011
   IF5=na, Rank=na, SNIP=0.523 (na), Ministry=7, GS=3
5. (C) E.C. Groen, S. Kopczyńska, M.P. Hauer, T.D. Krafft, J. Doerr, "*Users—The Hidden Software Product Quality Experts?: A Study on How App Users Report Quality Aspects in Online Reviews*", IEEE 25th International Requirements Engineering Conference (RE), 80-89, 2017
   CORE=A, Qualis=A1, Ministry=10, GS=5
6. (C) S. Kopczyńska, J. Nawrocki, "*Using non-functional requirements templates for elicitation: A case study*", IEEE 4th International Workshop on Requirements Patterns (RePa), 47-54, 2014
   Ministry=10, GS=12
7. (C) S. Kopczyńska, M. Maćkowiak, J. Nawrocki, "*Non-functional Requirements Elicitation Based on ISO 9126*", Software Engineering Techniques in Progress, ed. by Huzar Z., Nawrocki J., Szpyrka M., AGH University of Science and Technology Press, 169-179, 2009
   CORE=na, Qualis=na, Ministry=10, GS=0

Other papers by the author

8. (J) S. Kopczyńska, "*Relating reflection workshop results with team goals*", Computational Methods in Science and Technology 20 (4), 129-138, 2014
   IF5=na, Rank=na, SNIP=na, Ministry=9, GS=1
9. (J) J. Nawrocki, W. Complak, J. Błażewicz, S. Kopczyńska, M. Maćkowiak, "*The Knapsack-Lightening problem and its application to scheduling HRT tasks*", Bulletin of the Polish Academy of Sciences, Technical Sciences 57 (1), 71-77, 2009
   IF5=na(1.323), Rank**= 45/79(39/86), SNIP**=1.005(na), Ministry=30, GS=6
10. (C) M. Ochodek, K. Koronowski, A. Matysiak, P. Miklosik, S. Kopczyńska, "*Sketching Use-Case Scenarios Based on Use-Case Goals and Patterns*", Software Engineering: Challenges and Solutions, 17-30, 2017
    CORE=na, Qualis=na, Ministry=10, GS=2
11. (C) S. Kopczyńska, J. Nawrocki, "*HAZOP-based Approach to Pattern Identification for Non-functional Requirements*", IEEE 5th International Workshop on Requirements Patterns (RePa), 39-46, 2015,
    Ministry=10, GS=0
12. (C) J. Nawrocki, M. Ochodek, J. Jurkiewicz, S. Kopczyńska, B. Alchimowicz, "*Agile requirements engineering: A research perspective*", International Conference on Current Trends in Theory and Practice of Informatics, LNCS 8327, 40-51, 2014
    CORE=B, Qualis=B1, Ministry=10, GS=6
13. (C) S. Kopczyńska, J. Nawrocki, M. Ochodek, "*Software development studio: bringing industrial environment to a classroom*", Proceedings of the First International Workshop on Software Engineering Education Based on Real-World Experiences, 13-16, IEEE Press, 2012
    Ministry=10, GS=10

Acknowledgments

This research project would not have been a success without help from the wonderful people I would like to mention in here.

First, I want to sincerely thank my supervisor, Professor Jerzy Nawrocki. You gave me the challenge to take part in the "scientific adventure", as you usually call it. During this adventure, it was great pleasure to learn from you how to solve scientific problems. Thank you for your support. But, I would like to specially thank you for teaching me how to identify problems from the huge amount of information that flows to us every day and specify them later on, not only in the scientific world.

Next, I would like to thank my co-supervisor Mirosław Ochodek, PhD. Your feedback and guidance are invaluable; they allowed me to greatly improve this work. Thank you for insightful discussions on the faced problems and answering my questions even when I was keeping asking about the same over and over again or disturbing your work when I unexpectedly got stuck.

The work described in the thesis would not be possible without the participants of our studies. Many thanks to the students of the Poznan University of Technology who took part in our studies. I would like to thank also the companies that shared the data with us, especially: ATREM S.A., Roche Sp. z o.o., TALEX S.A., Currency One S.A., Consdata Sp. z o.o., IT Department of Poznan City Hall. Next, I would like to thank the participants of our surveys for their time and eagerness in expressing their viewpoints—your answers helped us in building our theories and improving our methods.

Many thanks go to my colleagues and friends for scientific discussions and valuable advice. Among them, particularly Michał Maćkowiak – thanks for a good joint start of our research projects and discussions, especially those on using different tools and technologies. Next, my thanks go to Konrad Siek for practical hints and explaining the complexities of the English language. I would like to thank also Jakub Jurkiewicz for valuable insights from the industry field. Moreover, I would like to mention Bartosz Alchimowicz, Wojciech Complak, Bartosz Walter, and Adam Wojciechowski – I appreciate your help in research and the time we have spent together at the Poznan University of Technology.

I am grateful to the Authorities of the Institute of Computing Science at the Poznan University of Technology for creating the environment in which I could study, broaden my knowledge and skills, become a researcher.

Finally, but most importantly, I would like to dedicate this work to my parents. They have been supporting me and I cannot thank them enough for that. Especially my mother who always believed in the success of my research project.

Sylwia Kopczyńska, Poznań, 2018

Contents

List of Figures

List of Tables

List of Abbreviations

A	
Avg.	Average
C	
cs	Computing Science
E	
Estim.	Estimation
I	
IEEE	Institute of Electrical and Electronics Engineers
IndCom	Individual Completeness ratio
Indiv.	Individuals
Interp.	Interpretation
ISO	International Organization for Standardization
K	
K	Catalog of NFR templates
M	
ME	Maintenance Effort
mgmt	Management Engineering
Min.	Minimum or minutes (depends on the context)
Max.	Maximum
N	
NFR	Non-functional Requirement
NoRT	Non-functional Requirement Template (name of notation)
P	
P	Project
R	
r	Requirement
RE	Requirements Engineering
S	
SetCom	Set Completeness ratio
SD	Standard deviation
SENoR	Structured Elicitation of Non-functional Requirements
T	
t	Template
Templ	Template-supported
SD	Standard deviation
SENoR	Structured Elicitation of Non-functional Requirements
U	
Unam	Unambiguity ratio
U	Utilization
V	
Veri	Verifiability ratio
Y	
Y	Yield

Chapter 1

Introduction

Software requirements are usually categorized into functional and non-functional ones. The former describe so-called user-valued transactions (i.e., functionality that supports users), and the latter state conditions under which the provided functionality is really useful (e.g., maximum response time).

Numerous cases of software development projects and products provide evidence how important software requirements are. For example, according to the Standish Group [223] that analyzed software projects worldwide, in the top ten factors that challenge projects are incomplete requirements and changing requirements.

Non-functional requirements (NFRs) in turn are too often neglected, especially those that are difficult to write or ostensibly obvious. That is an important risk factor, as in many cases inappropriate management of them was one the root causes of the project failure (see e.g., [22], [145], [173]). Such failures might even threaten human health and life, like in the case of the London Ambulance Service System. The system meant to substitute the manual procedures of dispatching ambulances to emergencies. Although the sum of a series of minor problems is held responsible for its downfall, NFRs played there a significant role. The system was not designed for performance, reliability, usability, and integrity. As a result "*one ambulance arrived to find the patient dead and taken away by undertakers, another ambulance answered a stroke call after 11 hours*" [25]. More frequently inappropriate management of NFRs drives to the rejection of a project or contractual penalties. For example, in the project developing Airborne Self-Protection Jammer for the U.S. Navy requirements concerning both reliability and effectiveness were poorly defined and arbitrarily chosen at the very end of the project. As a result, the project was canceled after spending $1.5 billion [13]. Another example is Archive of Electronic Documents, a system developed for the Poznan City Hall. Missing requirements concerning access control resulted in the prolongation of the project duration by 1 year and the contractual penalties of over 60% of the project budget [189].

One of the approaches aiming at improving the practice of elicitation and specification of NFRs is to use *templates*. Templates, also called *boilerplates* or *blueprints*, are expressed in natural language as a text with some gaps (parameters) to be filled in and optional parts to select while formulating a requirement. Consequently, they are to preserve correct requirements syntax (e.g., EARS [154]). Moreover, they can encompass some semantics of NFRs ([107], [129], [204]). During specification of NFRs one can select templates from a catalog and provide the values of the parameters, which is called template-supported or template-based elicitation.

According to some authors, using templates improves consistency and testability of requirements, reduces ambiguity [20, 154, 240], makes elicitation and specification easier [220, 240], and saves the effort of specification [191]. Therefore, they have been incor-

porated into some tools (e.g., [202]). But those opinions, although formulated by experts, are not convincing to everyone. Palomares et al. [182] have found that practitioners are afraid of using templates. They perceive them as a complex method and do not know what would be the return on investment. Thus, more evidence is needed about *benefits* and *costs* associated with NFR templates.

Another issue is importance (or unimportance) of specifying NFRs in the context of agile projects. Before the agile era, it was natural to perform full requirements analysis including elicitation, specification, and analysis of NFRs upfront (see e.g., Somerville and Sawyer model of organization [220] or descriptions of spiral or waterfall software development models [219]). Agile approaches shifted the focus to communication and interaction over documentation [15]. They introduced many practices, but among them there is no one that explicitly mentions NFRs. Thus, a question arises *how agile practitioners perceive specifying NFRs.* Do they consider it as something useful or just 'hype' and an unnecessary thing? It is an important issue as currently ca. 71% of organizations report using various agile methodologies [195].

Moreover, the common practice to elicit requirements has been through interviews, workshops, observation of users, etc. On the other hand, the known technique to identify NFR templates is to analyze the requirements from past projects. However, recently, some researchers have noticed that online app stores (e.g., Google Play, Apple App Store, Amazon Appstore) can be an interesting new source of features [88, 91, 179, 185]. Thus, it would be worth to check if *user feedback left in app stores could be also a source of non-functional requirements and their templates.*

1.1 Aim and scope

The aim of the thesis is to provide knowledge that would allow managers of software projects to make an informed decision whether to use or not to use a catalog of NFR templates. The focus will be on the following research questions:

- **RQ1.** *How important is it for agile practitioners to have NFRs specified in their software projects?*
  If NFRs are not an issue then research on NFR templates has no meaning.
- **RQ2.** *What is the usefulness of catalog of NFR templates?*
  Assuming NFRs are important, it is worth to study how useful their templates can be. As *usefulness* is a general and fuzzy notion, the above question should be decomposed into subquestions for which it will be easier to propose appropriate measures (quality indicators).
- **RQ3.** *How the benefits and costs of using catalog change during catalog evolution?*
  An evolutionary approach to the development of catalog of NFR templates seems quite reasonable. Project after project new templates can be added to the catalog and some others can be modified. It would be worth to know how the benefits of using a catalog and costs of maintaining it change over time (here time would be measured in numbers of considered projects).
- **RQ4.** *To what extent user feedback in app stores is a good source of NFRs and their templates?*
  Recently it has been discovered that users leave their opinions on software products in social networks and app stores. Thus, it makes sense to check to what extent this information is interesting from the point of view of NFRs and their templates.

The above questions will drive the discussion presented in the next chapters of the thesis. The thesis is organized as follows. Before the readers start to get acquainted with the results of the research, they will be presented the basic knowledge about NFRs and their templates (Chapter 2).

The main part of the thesis starts with Chapter 3 where two studies are presented that aim to determine the importance of NFRs from the personal perspective of software project stakeholders (**RQ1**). One study investigated the importance of the 31 agile Requirements Engineering practices identified in a literature study [176]. The second one focused on NFRs and was not published yet.

Next, in Chapter 4, usefulness of catalog of NFR templates is analyzed (**RQ2**). Two studies are reported. One was a case study with 7 real-life software projects (Section 4.3), the other one was a controlled experiment with 107 elicitors identifying NFRs for a fictitious e-commerce application (Section 4.4). In both studies only one version of the catalog was used, i.e., catalog evolution was neglected.

Then, in Chapter 5, evolution of catalog of NFR templates is studied from the point of view of its usefulness and maintenance cost (**RQ3**). The presented research was based on 41 real-life projects coming from various organizations. First, the evolution is directed by the order in which the author of the thesis has acquired the projects. Next, to check the influence of this particular evolution on the results of the study, 10,000 random evolutions of the same final catalog (i.e., permutations of the initial sequence of the projects) are considered (the second study and its results have not been published yet).

Moreover, while working on the research problem, a new trend in requirements engineering has been identified as promising: it is analyzing user feedback from online stores of software products. This trend inspired the author to investigate if app stores would also be useful for elicitation of NFRs (**RQ4**). The study is described in Chapter 6.

The last chapter, i.e., Chapter 7, recapitulates the results and indicates a possible direction for the future research concerning catalogs of NFR templates.

The materials used in the studies on NFR templates reported in the thesis and the up-to-date status of the work in this area carried out at Poznan University of Technology can be found at `http://norts.cs.put.poznan.pl`.

Many parts of this thesis are based on the journal or conference papers co-authored by the author of the thesis. Each chapter that is based on such material (Chapters 3-6) has a preface containing a reference to the paper(s), a structured abstract, and a description of the responsibilities/contribution of the author of the thesis.

1.2 Typographical conventions

Several typographical conventions are used in the thesis. The meaning of particular mathematical symbols and variables is explained in the place where they are used for the first time.

KEY TERM Margin notes about key terms and concepts appearing in the corresponding paragraph.

[13] Reference to books, articles, and other sources of information with bibliographic details described in the Reference section.

Each chapter contains research questions that refine the main research questions stated in Section 1.1. These subquestions are labeled and referenced to as "RQz.x", where z is the number of the main research questions and x is the number given the subquestion.

Conducting empirical studies involves many people who contribute to the final results. To emphasize this fact and appreciate their contribution in the thesis I am using the pronoun “we”. However, despite the help and inspiration I have received, I am taking the full responsibility for the research and results presented in the thesis.

Chapter 2

Non-functional requirements (NFRs): Elicitation and templates

2.1 Quality characteristics, NFRs, and their equivalence

2.1.1 Lack of consensus about definition of NFR

First, let us discuss what a requirement is. There exist many definitions, and the following paragraphs bring closer some of them.

ISO/IEC/IEEE Standard 24765:2010 [115] contains the following definition of *requirement* : REQUIREMENT

> *(1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents; (3) a documented representation of a condition or capability as in (1) or (2); (4) a condition or capability that must be met or possessed by a system, product, service, result, or component to satisfy a contract, standard, specification, or other formally imposed document. Requirements include the quantified and documented needs, wants, and expectations of the sponsor, customer, and other stakeholders.*

Sommerville and Sawyer [220] claim that "*requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.*" The practitioner and consultant Brian Lawrence says that "*a requirement is anything that drives design choices*" [236]. Wiegers and Beatty [238] on the other hand claim that requirements "*describe the observable behaviors the system will exhibit under certain conditions and the actions the system will let user take*".

To better understand what a requirement is, it is worth to mention types of requirements. Requirements are frequently divided into functional requirements (FRs) and non-functional requirements (NFRs) [238].

There exist some consensus among the definitions of functional requirements. One of the software engineering professors Ian Somerville [219] states that they are "*statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also state what the system should not do.*"

Ebert et al. [67] and Cleland-Huang et al. [42] refer to one of the most popular definitions for functional and non-functional requirements by stating that a requirement that

describes not *what* the software will do, but *how* (or sometimes *how well*) the software will do something is called a non-functional requirement.

Glinz [83] gives a comprehensive overview of NFR definitions in standards and literature which was further extended by Doerr [62], then by Eckhardt [68] and, lastly, by us. The extended version of the overview is given in Table 2.1.

Source	Definition
Antón [10]	describe the non-behavioral aspects of a system, capturing the properties and constraints under which a system must operate.
Burge and Brown [29]	describe desirable overall properties that the system must have.
Cleland Huang [42]	describe important constraints upon the development and behavior of a software system. They specify a such as security, performance, availability, extensibility, and portability. They play a critical role in the architectural design and should, therefore, be considered and specified as early as possible during system analysis.
Davis [57]	the required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability.
Ebert [67]	a requirement that describes not what the software will do, but how the software will do it called a nonfunctional requirement.
Franch [78]	how the system behaves with respect to some observable attributes like performance, reusability, reliability, etc.
Franch and Carvallo [79]	we derive quality requirements as restrictions over the quality model.
Glinz [83]	an attribute of or a constraint on a system.
Jacobson et al. [116]	specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. A requirement that specifies physical constraints on a functional requirement.
Kotonya and Sommerville [135]	requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet.
Landes and Studer [142]	constitute the justifications of design decisions and constrain the way in which the required functionality may be realized.
Lawrence et al. [143]	the characteristics you intend your software to exhibit.
Miller [160]	a specification of how well a software system must function On the other hand, non-functional requirements describe the system environment view or characteristics of the system. They describe the constraints imposed on the design and construction of the system, as well as quality attributes related to the user need for the operation, revision, and ability to handle change or growth.
Mylopoulos et al. [167]	global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like. There is not a formal definition or a complete list of non-functional requirements.
Ncube [171]	the behavioral properties that the specified functions must have, such as performance, usability.
Paech and Kerkow [178]	any requirement describing the quality of the system.
Robertson and Robertson [204]	a property, or quality, that the product must have, such as an appearance, or a speed or accuracy property.

Van Lamsweerde [230]	types of concerns: functional concerns associated with the services to be provided, and non-functional concerns associated with quality of service such as safety, security, accuracy, performance, and so forth.
Wiegers and Beatty [238]	a description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior.
Young [244]	a necessary attribute in a system that specifies how functions are to be performed, often referred to in systems engineering as the -ilities.

Table 2.1: Definitions of NFR.

It follows from Table 2.1 that there is no consensus about what non-functional requirement is. Thus, each publication in this area shall provide a clear statement on the definition used therein.

NFRs play important role while designing the architecture of any system, but functional requirements shall be also considered during this process. Thus, in the literature we might find the term Architecturally Significant Requirements (ASE) that is used to describe those requirements that affect the system architecture (both for NFRs and FRs) [36].

2.1.2 Categorization of NFRs

The definitions of non-functional requirement like those by Cleland-Huang et al.[42], Davis [57], Franch [78], Jacobson et al. [116], Mylopoulos [167], Van Lamsweerde [230] explicitly refer to characteristics like performance, reliability, security etc. Such characteristics are used to explain better what NFRs are and to *categorize* NFRs. Categorization helps to systematize the processes connected with NFRs, e.g., it can be used as an agenda for an elicitation workshop [129, 204], or to organize documentation [110].

There have been proposed multiple categorizations (sets of characteristics). Categorizations are usually based on quality models. Quality models "*provide consistent terminology for specifying, measuring and evaluating system and software product quality. They also provide a set of quality characteristics against which stated quality requirements can be compared for completeness*" [113]. Quality models usually have a hierarchical structure composed of characteristics that are, sometimes, further divided into subcharacteristics.

One of the first models was the Boehm's Quality Model published in 1976 that breaks quality into three top-level characteristics: As-is utility (reliability, efficiency, human engineering), Portability (device independence, self-contentedness), and Maintainability (testability, understandability, modifiability) [23]. Then, in 1977 McCall published his quality model developed for the US Air Force that comprised seven top-level characteristics (correctness, reliability, efficiency, integrity, usability, maintainability, testability, flexibility, portability, reusability, interoperability) [156]. From experience in the industry, namely from Hewlett-Packard, came the FURPS model and its successor FURPS+ (functionality, usability, reliability, performance, supportability) described by Robert Grady [86].

Presumably, the most popular model of software quality is the one contained in the ISO/IEC 9126:2001 standard [114] (abbreviated to ISO 9126) and its successor described in ISO/IEC 25010:2010 [113] (abbr. to ISO 25010). At the time the work on the dissertation started we had been using the ISO 9126 standard but later we based our work on the ISO 25010 standard, and we did it in such a way that all the studies presented in the thesis are consistently based on the latter one. ISO 9126 defines three models: quality-in-use, internal quality, and external quality. The standard makes a distinction between internal and external attributes which influence and determine quality-in-use aspects. It defines 6 characteristics for internal and external product quality which are further divided into 21

subcharacterisitcs (see Figure 2.1). The quality-in-use is defined as a combined effect of 4 characteristics. The standard describes measures for each subcharacteristic, but they are defined vaguely and are difficult to apply. However, the structure (division into characteristics and subcharacteristics) has gained worldwide acceptance.

ISO 25010 defines only two models: quality-in-use and product quality. The quality-in-use model contains 5 characteristics concerning the interaction of a user with the product. It is used to describe the impact the system or software product has on its stakeholders (human-computer interaction). The second model is divided into 8 characteristics each further divided into subcharacteristics. The models are presented in Figure 2.2.

Figure 2.1: Quality characteristics defined in ISO 9126.

Mairiza et al. [149] investigated the literature to find types of NFRs. It resulted in identifying 252 types of NFRs. They found that there are quality attributes (e.g., maintainability, performance, and reliability), development constraints (e.g., timing, cost, and development personnel), external interfaces requirements (e.g., user interface & human factors, look & feel, and system and system interfacing), business rules (e.g. production lifespan),

Figure 2.2: Two quality models defined in ISO 25010.

and others (e.g. cultural, political, and environmental). The top 5 most frequently used types of NFRs are: performance, reliability, usability, security, and maintainability.

2.1.3 Approaches to expressing NFRs

In the thesis, we focus on NFRs that are expressed in natural language as statements. However, it is worth to note that NFRs might be expressed in other forms (notations).

For example one of the approaches to specifying NFRs is based on the goal-oriented modeling. Goal is *"an objective the system under consideration should achieve"* [230]. One of such goals might be an NFR. Goals are represented by nodes, and they are hierarchically decomposed into subgoals represented as "offsprings". In this approach, one can also represent a relation between goals and actors (owners of the goals). One of the most popular implementations of this approach is the NFR Framework by Chung, Mylopulous et al. [167].

There are also other approaches such as GRL based on the i* framework [245], Tropos [32], and KAOS [230].

2.1.4 Notion of NFR used in the thesis

In the thesis we consider requirements that concern software product, which is not only source code, but also includes user guides, technical documentation etc. As defined in ISO 25010, ***software product*** is a "*set of computer programs, procedures, and possibly associated documentation and data*" delivered under a single name for use by others.

SOFTWARE PRODUCT

Let us define ***functional core*** of a software product as a set of (1) business roles the end-users can play, (2) use cases assigned to them that directly support the tasks the end-users are to perform (user-valued transactions), and (3) business objects that are manipulated by them and directly pertain to the business tasks.

FUNCTIONAL CORE

This strong connection between functional core and business tasks is an important distinction between the former as some NFRs (e.g., those concerning non-repudiation or authentication) can be expressed in terms of additional functionality to be used, e.g., by administrator of the system (in other words, NFRs can be 'disguised' as functional requirements).

For instance, an author wanting to submit a paper to a conference would play a business role, his paper would be a business object, and submitting the paper would be part of the functional core. On the other hand, logging all the operations performed by authors to achieve non-repudiation and displaying the log to the administrator would be out of the functional core as it does not directly contribute to the business tasks, i.e., submitting the papers. Thus, functional core strongly depends on the set of business actors.

Our definition of NFR is based on Glinz's one [83]. Thus, we do not consider project nor process requirements. As a reference model for NFRs (e.g., to categorize requirements) we use ISO 25010 subcharacteristics [113] (all but Functional completeness, Functional appropriateness, Capacity, Maturity, and Appropriateness recognizability—the rationale for this exclusion is presented at the end of this subsection). As a ***non-functional requirement*** (NFR) we regard attributes of or constraints imposed on a software product. To be more precise, the definition encompasses:

NON-FUNCTIONAL REQUIREMENT

- a value constraint — a constraint on value of some attribute of the software product (e.g., "*maximum response time* ≤ 1 *second*") or its operational environment (e.g., "*maximum number of users = 10,000*");
- an architectural constraint — a constraint on some architectural decision (e.g., "*the database engine the product is going to use is Oracle 18c*" or "*the data of accepted paper shall be exchanged via API using XML-based format*");
- a quality in use constraint — a constraint on the value of some attribute describing end-users performing certain tasks with the use of the software product (e.g., "*the average time of preparing a report by a dean-office worker ≤ 15 minutes*");

- a metamorphic requirement — a functionality that pertains to quality of the software product but does not belong to the functional core (e.g., "*all technical exceptions shall be logged in the product log*");
- an indirect requirement — a reference to a document where a set of NFRs is specified (e.g., "*the system shall be compliant with the General Data Protection Regulation (EU) 2016/679*");
- a postponed requirement — a demand of a development task (activity or process) to be performed that would allow defining the requirement (e.g., "*Detailed GUI requirements shall be identified during a workshop attended by the staff of Loan Department*").

The neglected ISO 25010 subcharacteristics

As mentioned earlier, we have not included all subcharacteristics of ISO 25010 into our definition of NFR. The reason is our focus on to-be-developed software products. Altogether five subcharacteristics have been omitted, and each of them is discussed below.

- Functional completeness, i.e., *degree to which the set of functions covers all the specified tasks and user objectives*. In the context of a to-be-developed software product, using this subcharacteristic as a basis for formulating an NFR would result rather in a project scoping directive i.g., defining the scope of an increment or release by listing the functions to be implemented.
- Functional appropriateness, i.e., *degree to which the functions facilitate the accomplishment of specified tasks and objectives*. In the context of a to-be-developed software product, using this subcharacteristic as a basis for formulating an NFR would be rather strange and result in defining some relaxation on some functions (e.g., to save some development effort or to shorten time-to-market one could relax some requirements, that is to replace them with their new versions).
- Capacity, i.e., *degree to which the maximum limits of a product or system parameter meet requirements*. In the context of a to-be-developed software product, using this subcharacteristic as a basis for formulating an NFR might result in defining both the maximum limits and the real needs. We recommend to express them separately, as in our opinion it might lead to some misunderstandings (e.g., instead of specifying "degree" one could say what shall be the maximum limit of users for a product and the maximum number of user they really expect).
- Maturity, i.e., *degree to which a system, product or component meets needs for reliability under normal operation*. In the context of a to-be-developed software product, using this subcharacteristic as a basis for formulating an NFR might result either in NFRs that fit other reliability categories or to relaxation of previously defined NFRs.
- Appropriateness recognizability, i.e., *degree to which users can recognize whether a product or system is appropriate for their needs*. We have not encountered any requirement directly specifying expectation about appropriateness recognizability. Therefore we decided to exclude it from our analysis.

2.1.5 Equivalence of NFRs

When we have to compare two NFRs, we will say they are ***strongly equivalent*** (equivalent for short) if they have the same solution spaces that is there exists no solution that satisfies one of them but fails to satisfy the other. STRONG EQUIVALENCE

For example, let us assume that there is a requirement in the functional core saying that a user can let the software product generate a monthly sales report and gets the document as a result. In such a case, if an NFR r is *Maximum time for generating a monthly*

sales report shall not exceed 60 minutes then the following requirements will be considered strongly equivalent to r:
– r'_1*: Maximum time for generating a monthly sales report shall not exceed 1 hour*;
– r'_2*: Monthly sales report is to be generated at maximum in 60 minutes.*
However, the following NFR:
– r'_3*: Less than half an hour is needed for the system to generate a sales report concerning the last month.*
will not be strongly equivalent to r since r'_3 imposes more restrictive time constraint (if the operation takes 1 hour it would satisfy r'_1 and r'_2 but not r'_3).

2.2 Templates of NFRs

2.2.1 Definition and types of templates

The idea of using templates was discovered a long time ago. In the famous book on Requirements Engineering from 1997 Somerville and Sawyer [220] stated that *"a **template** is some structure for the description of each requirement"*. Later on, in 2007, Steven Withall, a practitioner and software engineering expert, described templates as *"a starting point for writing a requirement of this type or more than one if there are distinct alternative ways"* [240]. TEMPLATE

A template (also known as a blueprint or a boilerplate), according to Somerville and Sawyer, should have *"fields to note the information that you wish to associate with the requirements"* [220]. The idea behind using templates is to preserve knowledge so that it can be reused later on. Thus, each template shall have *"named fields which remind the analyst which information is to be collected and recorded"* [220]. These fields in templates are also known as slots, gaps, or parameters.

From the analysis of different templates proposed in the literature it follows that there are the following types of templates used to specify NFRs in natural language (see some examples in Figure 2.3):

- Syntax Templates — they preserve a correct and common syntax of a statement to express a requirement, e.g., Rupp's [192] and the EARS templates [155].
- Statement Templates — they preserve small statements (statement parts) that can be combined to build the full statement expressing a requirement, they focus on syntax, e.g., Denger et al. [59] with sentence parts such as events, conditions, exceptions.
- Syntax and Semantic Templates — they preserve a correct and common syntax of a statement that can express a requirement and contain knowledge how to express specific requirements, e.g., statements/words used to correctly state the maximum time limit a user can wait for the response of the application. A set of these templates might contain templates that concern different concerns such as performance, security, interoperability, etc. Such templates are used by Hull et al. [106], Kopczyńska et al. [131, 132], and they are also parts of the solutions proposed in the PABRE approach [201] and in the approach by Withall [240]. A set of such templates might contain templates that concern only specific category of templates, e.g., like the security-related templates studied by Riaz et al. [203] or the templates on performance proposed by Eckhardt et al. [69].
- Structure Templates — they preserve the attributes that need to get assigned values to specify a requirement, frequently such templates are used to document requirements in the form of a table, e.g., Volere Snow Card [204], use case template [3], Planguage [82].

It shall also be noted that in a software development project there could exist templates at different levels. There might be templates for a single requirement as already

Syntax Templates

Rupp's Template

[<When?><under what conditions?>] the system <system name> shall <process> <object> [<additional details about the object>]

EARS Templates

If <optional preconditions> <trigger>, then the <system name> shall <system response>

While <in a specific state> the <system name> shall <system response>

Statement Templates

Denger et al. Templates

<When | If> (conjunction) noun phrase (VARIABLE) verb (VALUE CHANGE) {numeral adjective (VARIABLE VALUE) | noun phrase (VARIABLE)}

until {EP1 | EP2 | EP3}

Syntax and Semantics Templates

Hull's Template

The <system> shall be able to <function> <object> not less than <performance> times per <units>

Kopczyńska & Nawrocki Template (NoRT notation)

(System | <system part>) shall be available <time amount> per (week | month | year | <time slot>) [between <start hour> and <end hour> <timezone>] [for <user>] [in <geographical location>].

Structure Templates

Planguage

GIST	A short description to help understanding
PLAN	The level at which success can be claimed
SCALE	The scale of measurement used to quantify the statement
METER	The process or device used to measure using the SCALE
MUST	The minimum level required to avoid failure
STRETCH	The best if everything goes perfectly
WISH	A desirable level of achievement
RECORD	The best-known achievement
PAST	Previous results that may be used for comparison
TREND	A set of historical data or extrapolation of this
STAKEHOLDER	A person or organisation materially affected
AUTHORITY	The person, group or level of authorization allocated
DEFINED	The official definition of a term

Volere Snow Card

Requirement#: **Unique Id** Requirement Type: **Template selection** Event/usecase #: **Origin of the requirement**
Description: **A one-sentence statement of the intention of the requirements**
Rationale: **Why is the requirement considered important or necessary?**
Source: **Who realized this requirement?**
Fit Criterion: **A quantification of the requirement used to determine whether the solution meets the requirement.**
Customer Satisfaction: **Measures the desire to have the requirement implemented** Customer Dissatisfaction: **Unhappiness if it is not implemented**
Dependencies: **Other requirements with a change effect** Conflicts: **Requirements that contradict this one**
Supporting Materials: **Pointer to supporting information**
History: **Origin and changes to the requirement**

Figure 2.3: Examples of diffrent types of templates.

mentioned, but there might be templates for documents of requirements, e.g., for a software requirements specification as in the IEEE 830 Standard [109] or the Volere template for specification [204].

2.2.2 Patterns

Another approach is to preserve knowledge for future use is called patterns. The idea came to software engineering from the design and architecture domain [5]. Generally, patterns structurally describe the problem they solve, the context in which they are to be used, and the solution itself. The solution is frequently provided in the form of template(s) if the patterns are targeted at the requirements to be documented in natural language. A pattern usually is used to guide specifying a single requirement at a time. Patterns were proposed for both functional requirements (e.g., [3, 175]) and non-functional requirements. For NFRs there exist a few catalogs of patterns for natural language requirements. The following paragraphs present two well-known solutions.

Withall's patterns. Withall proposed 37 NFRs patterns [240] divided into 8 categories: fundamental, information, data entity, user function, performance, flexibility, access control, commercial. Each pattern is described in a structured manner and well discussed. There are (1) basic details (name, related patterns, anticipated frequency of use, pattern classifications); (2) applicability; (3) discussion; (4) content; (5) template(s); (6) examples; (7) extra requirements; (8) considerations for development; (9) considerations for testing. The templates used to specify the solution (requirements) might contain some parameters and some optional parts. In order to indicate a parameter the '«' and '»' characters are used, and for optional parts the '[' and ']'. For example:

> **5.3 Technology Requirement Pattern → Template:**
> The system shall use «Technology description» for «Technology usage». [«Technology version statement».] «Motivation statement».

PABRE approach. The next important set of patterns dedicated to both functional and non-functional requirements was proposed in the PABRE approach [201, 183]. The approach aims at facilitating requirements elicitation using patterns (a catalog of patterns) with the goal of saving time and reducing errors during this activity. Initially, it was dedicated for off-the-shelf selection projects driven by call for tenders processes [201]. Then, some extensions were also proposed, e.g., for content management domain [184], and a tool was developed [181]. Each pattern consist of: name, authors, context (RE activity, pattern type, business domains, organizational environment factors, stakeholders), problem forces, application, known uses, cataloging (classification, related patterns), and solution. Solution consists of: goal, description, keywords and requirement form. The latter consists of description, comments, versions, author, sources, fixed part and extended parts. Fixed parts and extended parts take the form of templates with some parameters. For example:

> **Faliure alters → Requirement Form → Fixed Part:**
> The solution shall give an alert in case of failure.
> **Faliure alters → Requirement Form → Extended Part:**
> Alerts provided by the solution shall be: AL.
> AL is a non-empty set of altert types.
> AL:Set(AlertType)
> AlertType:{E-mail, SMS, Page, Fax, Skype, IM, ...}

Patterns—Conclusion. Templates are encapsulated into patterns as a mean to express the solution the patterns propose. The advantage of using patterns over pure templates is that they contain more information, e.g., in which context a given template shall be used. Such information might not be obvious when reading just a template. However,

on the other hand, such information needs first to be recorded and later on maintained, which incurs more costs compared to templates.

2.2.3 Notion of NFR template used in the thesis

As it follows from the definitions mentioned in Section 2.2.1, the existing definitions of templates are quite general. Thus, in this Section, we provide the definitions we use in th thesis.

We defined an ***NFR template*** as a regular expression over literals and parameters that allows to *derive* a sentence which constitutes an NFR for some software product. NFR TEMPLATE

The process of ***directly deriving*** (deriving for short) an NFR from a template comprises the following steps: DIRECT DERIVITATION

1. Derive a sequence of literals and parameters from the regular expression, e.g., decide which literal best suits the current context, decide on the multiplicity of a parameter, adjust sentence structure.
2. Replace all the parameters with their actual values, e.g., provide concrete numbers and names.

Our definition of an NFR template allows the flexibility of choosing the notation to document templates that suits current needs. For example one might use the NoRT notation [129], VOLERE Snow Card [204], QUPER's approach [200] or combine them.

In the thesis we use the *NoRT notation* (proposed in [129]) which aims at supporting the NFRs documentation in the form of natural language statements. NORT NOTATION

A finite set of of NFR templates is called a ***catalog of NFR templates*** ('catalog' for short) , in the thesis denoted as K. CATALOG OF NFR TEMPLATES

In the thesis the considered catalogs are organized into categories, each is one subcharacteristic of the ISO 25010 standard [113].

An example from our catalog of an NFR template in the NoRT notation is presented in Figure 2.4. The part of an NFR template that remains unchanged during derivation (not taking into account inflexion) is called *core*; items in angle brackets represent *parameters* (e.g., <number>); alternatives (*options*) are presented in brackets and are separated with bar characters (e.g., (milliseconds | seconds | minutes)). Each option can be either a parameter, a *static* (a statement that remains unchanged) or a combination of the two. To allow choosing more than one option, the '{ }' option modifiers are used instead of brackets and, also, one might indicate the options multiplicity that is a following '+' means 1 or more options can be chosen during derivation, while '*' means 0 or more options. These characters can modify multiplicity of parameters, and, additionally, the '?' following a parameter allows indicating that it can be used 0 or 1 times.

While deriving an NFR from the template presented in Figure 2.4, one would need first to decide which alternative of the three possible suits their needs, e.g., "maximum", and select it. In the second step the two parameters – request and response would be replaced with values, e.g., "starting generation of a monthly sales report" and "obtaining the report." Then, the amount of time and its unit must be provided, e.g., 1 minute.

Since, usually, a catalog of NFR templates is being created based on the lessons learned from past projects, it might not be perfect in all the contexts it gets used. During elicitation, it might happen that one finds that in a given template needs some modifications to document the requirement the person has in mind. Then, ***template extension*** might prove useful. To extend template t' from catalog K one can, e.g., add some parameters and modify the static text or the options of template t' in such a way that new template t is *backward compatible* with the old one, i.e., every requirement r that can be directly derived from old template t' can also be directly derived from new template t. Such derivation is called ***indirect derivation***. TEMPLATE EXTENSION INDIRECT DERIVIATION

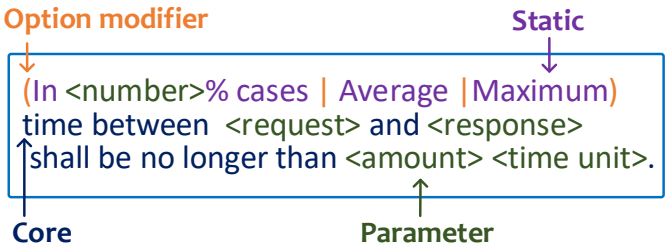


Figure 2.4: An example of a non-functional requirement template expressed using the NoRT notation.

2.3 Elicitation of NFRs

2.3.1 Elicitation and other RE activities

Requirements Engineering is one of the processes in a software development project. It can be divided into the following activities [236]:

REQUIREMENTS ENGINEERING

ELICITATION

- *elicitation* which aims at understanding the users and other stakeholders and discovering their needs;
- *analysis* which is to look at the gathered requirements from different perspectives, organize them, add more detailed information, create models, prototypes, look for priorities, search missing requirements, evaluate risks, etc.;
- *specification or documentation* aims to record the information that will facilitate further communication in the project, create documents, etc.;
- *validation* is to ensure that the requirements are of good quality and satisfy the needs of stakeholders;
- *management* starts when the requirements are agreed that they are basis (baseline) for design and development, according to Pohl [191] management is about maintaining the RE artifacts throughout the project (e.g., creating artifacts, prioritizing requirements, change management), about organizing and controlling RE activities, and about observing the system context to identify any changes that require RE activities.

The first four steps are called requirements development by some authors [236]. The four mentioned activities seem to happen once and sequentially, but in practice there are multiple recurrences. For example, when during specification it appears that there is some information missing, then one has to elicit the information from project stakeholders again. Moreover, during a project, these activities might be executed upfront for all requirements as in the waterfall approach, or multiple times for smaller chunks of requirements when a project is realized in iterations.

In the thesis, we focused on elicitation by many regarded as *"the most difficult, most critical, most error-prone, and most communication intensive aspect of software development. (...) the heart of requirements engineering"* [238]. Hoffmann and Lehner found in 2001 that successful project teams allocate ca. 28% of the project effort to RE, and ca. 11% is spent on elicitation [103].

Pohl in his book on RE [190] emphasizes that elicitation is not only about gathering the existing requirements people already have in their minds, but also about developing new and innovative requirements.

One of the first tasks during elicitation is the identification of sources of requirements. There exist various sources such as stakeholders who have some idea about the system under development in their minds, documentation which for example describes some con-

straints on the system, video recordings of for example how the process which is to be supported by the system under development is executed, etc.

2.3.2 Approaches to elicitation of NFRs

Although it is claimed that frequently requirements elicitation happens in an *ad hoc* manner [27], in the literature, we may find numerous works on the methods that aim at supporting this process.

There are general guidelines that apply to both functional and non-functional requirements, for example, interviews, observations, focus group meetings, etc. [219, 220, 238]. After the huge success of Joint Application Design workshops at IBM, Gottensdiener argues that workshops are one of the most powerful technique for requirements elicitation. She investigated the factors that influence the effectiveness of workshops and published a book containing guidelines on how to conduct requirements workshops [85].

On the basis of the mentioned general approaches, there are developed some methods and techniques that are dedicated for non-functional requirements. In the Tables 2.2 and 2.3 we have compared the selected elicitation methods, including our method—SENoR, which is described more precisely in Chapter 4. It can be seen that the majority of approaches is based on interviews and workshops that are methods that involve interaction with stakeholders. Then, in some approaches, e.g., Luo et al. [147], documents are also used during the elicitation process. There are also some methods that we might call supporting. After the elicitation activity involving stakeholders, one can apply them to analyze the available recordings and documents to suggest NFRs automatically (e.g., [42]).

Different forms of support are added to the general elicitation approaches. In our method, there are templates of NFRs and characteristics [129]. In the NFR Method by Doerr there are also templates, but also checklists, quality models and an algorithm to analyze functional requirements [62]. Moreover, the approach by Cysneiros and Leite proposes to use Language Extended Lexicon as a prompt list [53].

Of course, when one wants to select the approach that would suit their context, they need to take into account also the constraints. One of them is suitability to a specific context, e.g., there are general methods like ours [129], Doerr's [62] etc., but also such that are dedicated to the certain area, e.g., agent-oriented methods [147].

Moreover, choice of the notation used to document NFRs is highly important. Except for natural language, there could be methods using goal-based modeling, e.g., [53].

In Table 2.4 we have presented the activities needed to execute to elicit NFRs for the identified elicitation approaches. Some authors describe their proposals generally, some more precisely, and, as a result, they become less or more approachable for potential elicitors.

One of the most precisely described approaches is the one proposed by Doerr et al. [62]. It starts from functional requirements as the basis and then utilizes quality model, checklists, requirements templates, privatization questionnaire to derive NFRs. Also the method by Herrmann et al. called MOQARE [97] seems to have accurately defined steps. The method also uses functional requirements as a starting point, and then, the NFRs are being discovered by identifying misuses (events that might threaten the system). Then, quality requirements shall be the solution of how not to allow to damage the system.

The existing elicitation approaches also include those that allow eliciting NFRs in other than natural language form, e.g., NFR Framework [167], KAOS [230].

Reference	Elicitation process	Description
Alexander [6]	**Pre-activities**	—
	Activities	Identify negative agents that might threaten the system, then brainstorm a list of misuse cases for each agent, then for each use case search ways in which misuse cases could threaten them, which might lead to creating more use cases or misuse cases.
	Post activities	Analyze requirements looking for relationships, e.g., conflicts.
Cleland-Huang et al. [42]	**Pre-activities**	Conduct interviews, workshops trying to gather/elicit as much information from stakeholders as possible.
	Activities	Run the software tool on the existing documentation.
	Post activities	—
Clements and Bass [43]	**Pre-activities**	—
	Activities	After 3 presentations on: the business case, business drivers, architecture drivers presentation, business goals are elicited and expressed as scenarios. Next, NFRs are identified by participants from the business goals. Then, there are relationships (traces) identified between business goals, NFRs and architectural drivers, and the detail values of requirements are elicited.
	Post activities	—
Cysneiros and Leite [53]	**Pre-activities**	Ensure LEL is present (build one or take exiting one), build functional perspective of the system.
	Activities	Go through LEL asking yourself (analyst) or customers which element of LEL is important, decompose softgoals in top-down or bottom-up manner trying to find the operationalizations—behavioral responses—trying to match NFRs (first in LEL, then NFR framework to graphs), apply heuristics and find interdependencies, solve conflicts, integrate functional and non-functional perspective (check if NFRs are implemented or included in other diagrams).
	Post activities	—
Doerr [62]	**Pre-activities**	—
	Activities	After prioritization of NFR areas, quality models get tailored within workshops by determining which aspects are important, and the templates are used to derive NFRs.
	Post activities	—
Hemann and Peach [98]	**Pre-activities**	—
	Activities	(1) Find the quality goals (based on business goals); (2) Describe Misuse Cases; (3) Define countermeasures (NFRs); (4) With countermeasures restart the cycle.
	Post activities	—
Luo et al. [147]	**Pre-activities**	Identify information sources (customers, marketing, end users, development team, documents).
	Activities	Collect information from requirements sources through interviews and workshops, and based on that information, build goal models.
	Post activities	Analyze the goal model using the ROADMAP approach.
Kopczyńska and Nawrocki [129]	**Pre-activities**	Optionally send out the agenda and a list of subcharacteristics (category) which will be discussed during the workshop, ensure a catalog of NFR templates is present.
	Activities	After a presentation of business drivers, problem idea, go through a catalog of NFR templates, characteristic(category) by characteristic, discuss which templates apply to the idea, select them and derive requirements.
	Post activities	—

Renault et al. [201]	**Pre-activities**	—
	Activities	IT consultant browses a catalog of patterns and suggests to Client those patterns that might apply. Client is to decide if the patterns apply to their problem. For the selected patterns IT Consultant together with Client specify requirements based on the templates that are given in the patterns.
	Post activities	Any lessons learned from the project/elicitation shall be reflected in the pattern catalog.
Song et al. [221]	**Pre-activities**	Prepare questionnaires.
	Activities	First stakeholders fill in the prepared questionnaires after some discussions. Then, the terms used by stakeholders get unified, and the statements they formulated are normalized (converted to the same scale, e.g., number of alarms per second). Then, from their statements the NFRs are formulated by requirements engineers for the interfaces the stakeholders interact with. Next, the NFRs for the entire platform are formulated. Next, the NFRs are checked if they are testable and any missing constrains get added. Lastly, the NFRs are adjusted for feasibility.
	Post activities	—

Table 2.4: Comparison of Activities executed during elicitation processes

Reference	Alexander [6]	Cleland-Huang et al. [42]	Clements and Bass [43]	Cysneiros and Leite [53]	Doerr [62]
Name	Misuse Cases	—	Pedigreed Attribute eLicitation Method (PALM)	—	NFR Method
Output requirements	NFRs and FRs	NFRs	NFRs	NFRs and FRs	NFRs
General elicitation type	workshop	analysis of documentation	interview or workshop	individual elicitation, interview	workshops
Type of support	negative thinking (of misuses)	software tool	links between business goals, architectural drivers, and NFRs	Language Extended Lexicon (LEL), NFR Framework	elicitation algorithm, quality model, templates
Tool	none	tool developed at DePaul University	—	lexicon: OORNF Tool, tool for goal models	Microsoft Office Plugin
Type of method	elicit requirements form stakeholders	support analysts to extract NFRs from existing documentation and recordings	elicit NFRs from stakeholders	from requirements to models of the system	elicit NFRs from stakeholders
Requirements source	stakeholders	documentation, recordings (e.g., of interview, workshop)	stakeholders	stakeholders	stakeholders
Roles		requirements analyst	—	—	—
Constraints		requires existing documentation and/or recordings of interviews or workshops to be done at least partially	—	—	—
Formalism to document NFRs	natural language	natural language	natural language	goal model	hierarchy with natural language requirements
Input	use cases	meeting notes, other documents	overview presentation, buisiness drivers presentation, architectural drivers presentation	—	documents, goal models, functional requirements
Output	misuse cases and use cases	set of sentences classified to NFRs categories - candidate requirements	List of NFRs connected to business goals	goal model	model of important quality aspects with NFRs

Table 2.2: Comparison of Elicitation Methods – Part 1

ID	Hemann and Peach [98]	Luo et al. [147]	Kopczynska and Nawrocki [129]	Renault et al. [201]	Song et al. [221]
Name	MOQARE	Hybrid Agent-Oriented Methodology	SENoR (Structured Elicitation of Non-functional Requirements)	PABRE	Platform NFR Development (PND)
Output requirements	NFRs	NFRs and FRs	NFRs	NFRs	NFRs
General elicitation type	interviews, document analysis, software analysis (legacy systems)	interview and workshop	workshop	workshop	discussion, survey
Type of support	checklist, thinking in terms of misuses	no	templates, standard ISO 25010	catalog of patterns	questionnaire
Tool	—	REBEL	Meeting Assistant	PABRE-Man	Microsoft Excel
Type of method	elicit NFRs	elicit FRs and NFRs from stakeholders and analyze them	elicit NFRs from stakeholders	elicit NFRs from stakeholders	elicit NFRs from users
Requirements source	stakeholders, documents, existing systems	customers, marketing people, development team, domain experts, end users, documents	stakeholders	stakeholders	users
Roles	requirements engineer	—	stakeholder, presenter, moderator, recorder	Requirements Expert, IT Consultant, Customer, Supplier	stakeholder, requirements engineer
Constraints	—	dedicated to agent-oriented systems	no	designed for the call for tender process	service oriented systems, platforms
Formalism to document NFRs	as part of Misuse Tree	ROADMAP: diagram, goal models	natural language	natural language	natural language
Input	FR (usually incomplete)	initial information, documents	presentation contacting description of the business case, overview of the solution; catalog of NFR templates	catalog of patterns	spreadsheet, initial information
Output	Misuse Tree	goal models with FRs and NFRs	list of NFRs	list of NFRs	list of NFRs

Table 2.3: Comparison of Elicitation Methods – Part 2

Chapter 3

Importance of NFRs

Preface
This chapter is based on the following papers:

1. M. Ochodek, S. Kopczyńska, "*Perceived importance of agile requirements engineering practices–A survey*", Journal of Systems and Software 143, 29-43 (Section 3.1);
2. S. Kopczyńska, M. Ochodek, J. Nawrocki, "*On importance of non-functional requirements in Agile software projects*", submitted for publication (Section 3.2).

Context.
The analyses of many projects that did not apply agile approaches (e.g., London Ambulance Service System, Airborne Self-Protection Jammer, Airine 5, Electronic Archive System) show evidence that a project failure can be traced to inappropriate management of NFRs (missing requirements, incorrectly specified, etc.). Over the last 15 years, agile approaches have changed the practices used in software projects. They shifted focus to interaction and communication among project stakeholders over processes, tools, and documentation. Among numerous agile practices, over 30 concern Requirements Engineering (RE). None of them mentions explicitly specifying NFRs. It might seem that it is not important anymore. Since a considerable number of organizations claim to work according to the agile principles (more than 70% according to the recent studies), it would be valuable to check whether NFRs are really unimportant nowadays or it is just a misleading impression.
Aim. The aim of the chapter is to investigate what is the perception of the importance of specifying NFRs in agile software projects.
Method. To achieve the stated goal, we performed the following two steps. First, we conducted a survey asking agile software development practitioners how they perceive the importance of the 31 agile RE practices that we had identified in a literature study (Section 3.1). Secondly, we asked those practitioners how they perceive the importance of having NFRs specified in their projects and juxtaposed the answers with their opinions on the perceived importance of the 31 agile RE practices (Section 3.2).
Results. The opinions of 136 respondents from a wide range of countries around the globe allowed us to determine the perceived importance of the agile RE practices and create a seven-tier ranking of the practices. We proposed a ranking method based on the PROMETHEE family methods.
118 of our respondents shared their opinion on how important it is to have NFRs defined in a software project. 77% of the respondents perceive having NFRs specified in an agile software project as at least important (this includes 30% of those for whom it appeared critical).

Moreover, the analysis concerning demographic data let us identify the relationship between the experience of the respondents and their view on the importance of specifying

NFRs. The more experience the practitioners have, the more important NFRs appear to them.
Conclusion. It makes sense to investigate methods of elicitation and management of NFRs.
My contribution. First, I have designed the surveys together with Mirosław Ochodek. Then, we conducted a literature study to identify agile RE practices. I was responsible for reviewing part of the primary studies, creating the survey questionnaire and sending it out to over half of the participants. Next, we validated and analyzed the data. In case of the first study, I was responsible for summarizing the data, open text analysis, and I discussed the ranking method. In case of the second study, I analyzed the data and juxtaposed them with the results of the first study.

3.1 Importance of agile requirements engineering practices

3.1.1 Introduction

Software development (SD) is a complex process, executed in the contexts quite far from the certainty with changing technology and requirements [222]. The agile SD methodologies have been proposed to help software development teams addressing many of the problems that arise in such environments. For the last almost 30 years, they have gained worldwide popularity.

One of the key processes in software development is Requirements Engineering (RE). The RE practices employed by agile teams differ from those used in traditional RE [30, 112]. The holistic nature of agile SD processes and focus on face-to-face communication makes it hard to distinguish between RE and non-RE agile practices. The practices such as collaboratively planned short iterations, or organizing daily team meetings that traditionally would not be perceived as being part of RE play an enabling role for lightweight requirements-related practices such as writing user stories.

As every process, also RE should be continuously improved. Agile software development methods recommend using a step by step approach to improving the SD processes and adapting them to changing conditions (e.g., focusing on introducing or improving single practices) rather than changing everything at once. However, still, many project teams are faced with a challenge to select the right practices to improve, and various strategies to approach this problem could be taken [217].

A decision on selection is vital and should take into account both the cost of mastering the practice (e.g., the effort) and how important the practice is for a project. Thus, a question arises *RQ1.1: which agile RE practices are the most critical for agile projects and shall be adopted/improved at first?*

A commonly used approach to learn about agile practices is to ask practitioners about their experience. For instance, there have been few survey research studies aiming at identifying critical success factors (CSFs) for agile projects [37, 77, 224]. Unfortunately, only some of these factors consider practices; more often they relate to environmental or organizational aspects of software development. Another approach to identifying valuable practices is to look for the most popular ones. Over the years, there have been many survey studies conducted that asked about the adoption level of agile practices [7, 12, 28, 33, 64, 108, 138, 170, 186, 205, 211]—among them a well-known "annual state of agile report" [231]. There were even few such studies on agile RE practices [30, 123, 124, 125, 235]. However, the question emerges *whether the popularity of an agile RE practice is a good predictor of its importance for an agile project.*

The focus of this chapter is on agile RE practices. We describe the findings from a survey research study conducted among 136 practitioners whom we asked about their view

on the importance of these practices for agile projects. Based on the collected responses we:

- Create a ranking of relative importance of agile RE practices that could help practitioners deciding on which practices they should focus at first.
- We analyze the impact of respondents demographic profiles (e.g., countries, responsibilities, agile SD methodologies, experience, domains) on how they perceive the importance of the agile RE practices.
- We compare the rankings of relative importance and popularity of agile RE practices to investigate if they correlate.

This section is organized as follows. Section 3.1.2 discusses the similarities and differences between the previous studies on agile RE practices and this study. In Section 3.1.3, we present the catalog of agile RE practices considered in our study. Section 3.1.4 explains the design of the survey questionnaire and presents the methods used to analyze the collected responses. The results and findings are presented and discussed in Section 3.1.5. Then, in Section 3.1.6, we discuss the threats to the validity of our study. Finally, we summarize our findings in Section 3.1.7.

3.1.2 Related Work

Survey research studies are common in the context of agile software development. There have been numerous studies investigating adoption levels and popularity of agile methods and practices. Some of these studies characterize usage of agile methods in general (e.g., [64, 138, 186]) other focus on particular countries (e.g., [7, 28, 170, 205]) or concern applicability of agile to particular domains, types of applications, or activities in projects (e.g., [12, 33, 108, 211]). There are also few studies focusing on the popularity of RE techniques and tools in agile projects [123, 124, 125, 235]. In general, the results of descriptive surveys seems attractive for practitioners since there are even surveys conducted annually by software vendors such as Version One [231]. This particular survey summarizes the current trends in the agile community when it comes to methods, practices, and tools. It also aims at identifying the benefits and challenges of adopting agile methods. However, despite its popularity, the survey received some criticism for lacking scientific rigor that decreases its trustworthiness [225]. The level of adoption and popularity of agile practices may to some extent be considered as indicators of the importance of practices. However, only a few of the mentioned studies aimed at investigating the influence of the agile practices on software development project or processes in IT organizations (e.g., [12, 64, 108, 186, 205, 211]). None of them focus solely on agile RE practices; however, few of them provide some insights into the importance of the selected agile RE practices (see Section 3.1.5).

Another group of survey research studies investigated critical success factors (CSF) for agile projects. For instance, Chow and Cao [37] surveyed 109 agile software projects from 25 countries trying to confirm the relevance of the CSFs previously identified in the literature. Later, the study was replicated by Stankovic et al. [224] in the context of 23 IT companies in former Yugoslavia and recently by França et al. [77] who studied the critical success factors in 11 Scrum projects in Brazil. The considered CSFs cover different aspects of software development and are usually quite general (e.g., delivery strategy, team capability, project management process). However, there is also a small intersection between the set of CSFs and the agile RE practices considered in our study (e.g., customer involvement and regular delivery of software).

Another relevant survey research study is the one by Cao and Ramesh [30]. They surveyed 16 organizations about the degrees to which they follow 7 agile RE practices and

what are benefits and challenges of using them. In this context, our study seems complementary. However, the main differences are that we investigate a larger set of agile RE practices and evaluate the relative importance of the practices. Finally, we study the relationship between the perceived importance of the practices and the experience of the respondents (e.g., domains, application types).

Williams [239] reported the results of a survey aiming at investigating which of the agile practices are essential for a team to be considered agile. Although the study covers a large set of 44 agile practices, it only partially overlaps with the set of agile RE practices considered in our study. Secondly, the question asked in our survey is a little bit different since we ask whether a practice is important for an agile project rather than if it makes the team agile.

Apart from the survey research studies, some insights on the importance of agile practices are given by studies comparing agile and non-agile approaches to software development (e.g., Elshandidy and Mazen [71]), or by systematic literature reviews (SLRs) discussing the benefits and challenges of agile RE (e.g., Inayat et al. [112]). However, according to the authors best knowledge, none of the works have discussed the importance of agile RE practices for a project.

Another relevant source of information about the value of agile RE practices are maturity models, i.e., the general ones (e.g., CMMI[45]), those dedicated to specific areas like RE (e.g., REPM[84]), or those dedicated to agile software development (e.g., AMM [187]). Recently, Fontana et al. [76] conducted a survey research study aiming at clustering agile practices depending on the maturity level of software development companies. Their results show the usage of which practices are more characteristic for mature agile organizations. However, it does not imply that such practices are less or more critical for agile projects in general. Thus, the results of our study provide a little bit different view on the agile practices.

3.1.3 Agile RE Practices

Many works unanimously use the term *practice* to refer to certain activities; however, the results of the study by Paivarinta and Smolander [180] show that the meaning of the term differs between studies, e.g., a practice can be a set of thoroughly predefined procedures executed by humans or a set of less organized activities. In this chapter, we propose the following definition of practice:

Definition 1. A ***practice*** is a pattern of actions or behavior that is recurrently executed by humans in order to achieve certain goals. PRACTICE

To create a catalog of agile RE practices we performed a literature review. We searched for secondary studies that could provide us with candidate agile practices. We found two systematic literature studies and several less formal literature reviews: a systematic literature review by Inayat et al. [112], a systematic mapping study by Diebold [60] with the base analysis by Abrahamsson et al. [2], a comparison of agile and traditional SD by Elshandidy and Mazen [71], a paper investigating the usage of agile practices at Microsoft [16], and a clustering analysis of agile practices by Fontana et al. [76]. In addition, we searched for definitions of practices in the books regarding agile software development and added practices found there if they were missing the previously searched studies. The more detailed analysis can be found on our website[1].

We have classified the practices into the following four categories:

[1] http://lio.cs.put.poznan.pl/AgileRESurvey

1. Practices characteristic for traditional RE, though they are also present in agile methods, e.g., Let customer prioritize requirements (P25) or Define project/ product constraints (P05).
2. Practices related to requirements that were proposed by the agile community and are not present in traditional RE, e.g., Available/on-site customer (P01) or Prepare acceptance tests before coding (P15).
3. Agile practices that have a broader scope than RE, but support agile RE playing an enabling role for lightweight requirements-related practices. For example, the practices: Organize everyday team meetings (P06) and Provide and maintain informative workspace (P09) aim at ensuring that every team member has information about the current status of the project (including the done requirements, changes in the requirements) and is able to discuss issues with requirements. Without them, for instance, the practice of writing short requirements like user stories or having short iterations ending with working software would involve the risk of having not enough knowledge to implement the software system. Another good example is the practice Perform the 'elevator test' (P12), which assures that team members have deep understating of the requirements imposed on the product they are working on as they shall be able to explain the idea of a product in the time it takes to ride up in an elevator.
4. Agile practices that do not have a direct influence on requirements, e.g., test driven development. Moreover, we decided to exclude the *code refactoring* practice, despite its inclusion in agile RE by Inayat et al. [112]. The practice seems to *loosely* contribute to RE activities, mainly by making further software modifications easier. In that sense, it is similar to many other code-related practices, such as *collective code ownership* or *simple design*. Therefore, in our opinion incorporating this practice is unjustified, and its inclusion in the practices catalog would imply the necessity of adding those practices as well. Similarly, we decided to exclude the *metaphor* practice proposed by XP, as it relates to system architecture rather than requirements [207].

While identifying candidate practices we encountered several issues. The first difficulty was that the naming of the practices was often inconsistent. Thus, we decided to select one of the alternative names for each practice arbitrarily. We also noticed that some of the practices mentioned by Fontana et al. [76] did not propose any patterns of actions or behavior, and as such, they were not convergent with our definition of practice (e.g., regulatory compliance—"*dealing easily with regulatory compliance*"). Therefore, we decided to exclude these candidate practices from further analysis.

Next, we observed that there is no consensus with respect to the definitions and scope of the practices. Thus, we applied the following "strategies" to capture their essence while creating our catalog of agile RE practices:

- **Look for "root definition" and enhance it with recent findings**—since there were practices proposed some time ago, some more recent work appeared that changed the way the practices are perceived. For example, the practice of *on-site customer* was introduced by Kent Beck [14] who stated that a "*real customer must sit with the team, available to answer questions, resolve disputes, and set small-scale priorities. By 'real customer' I mean someone who will really use the system when it is in production*". Rosenberg and Stephens [207] points out that Beck's definition does not directly state that the customer has to remain in the same physical location as the development team. Although some recent research [102] shows that distance factor can influence customer's involvement, there have also been examples of globally distributed agile projects that were able to achieve an acceptable level of communication despite the team members dispersal [133]. In addition, Abrahamsson and

Koskela [1] reported that in the project they analyzed, the customer was present on-site on average over 80% of the total work time, but only 21% of his work effort was required to assist developers in the development. Therefore, rather than the physical location of the customer, it seems more important to consider the effectiveness of the communication. And thus, we defined the practice of "*available/on-site customer*" as the availability and ability to answer questions regarding requirements in such a way that the response time does not negatively impact the work of the development team.

- **Define specific practices**—when we found that practice is defined at different levels of abstraction in different positions, we looked for the most specific one. For instance, the practice of *on-site customer* sometimes appeared as a standalone practice [14, 76], and, on other occasions, it was included in a more general one—*customer involvement* [112]. Thus, finally, we investigated the definitions of practices to determine if more fine-grained practices can be specified.
- **Make practices implementation neutral**—when a definition of practice was tight to a specific implementation rather than focusing on the general idea, we tried to make it more abstract. For instance, rather than keeping the practice *user stories*, we decided to decompose it into a set of practices derived from the INVEST checklist [233]. Thus, we focused on how a requirement shall be expressed rather than a specific documentation technique.

In Tables 3.1 and 3.2 we presented short definitions of the 31 identified practices considered in the study. It also contains the references to the literature positions based on which the definitions were developed.

3.1.4 Research Methodology

The goal of our study was to identify the perceived importance of agile RE practices in the context of agile software development projects. To achieve the goal we designed a cross-sectional survey according to the guidelines by Kitchenham and Pfleeger [127].

Survey instrument

To collect opinions of a wide variety of respondents (with respect to country, domain, experience) we designed an on-line questionnaire divided into three parts.

The first part included 31 questions asking about the *perceived* importance of the previously identified agile RE practices (see Section 3.1.3). The perceived importance of practice was evaluated using a three-point ordinal scale. Respondents were supposed to recall the projects they had participated in and judge whether the practice was "critical" for these projects, "important", or "additional" (helpful but also supplementary—could have been rejected without doing any harm to the projects). Alternatively, the respondents could choose the option "other" to provide their own, descriptive answers instead of using our ordinal scale. Finally, we allowed skipping questions by selecting the option "don't have an opinion".

The second part contained 11 questions asking about demographic information. These questions played two roles. The first one was to characterize the sample of respondents. The second one was to identify potential relationships between the demographic information and perceived importance of the practices to determine the degree to which the results are transferable between different contexts. In particular, we asked about the participants' professional experience: years of experience, activities performed in projects, experience with respect to applications types, domains, methodologies, and size of the projects they participated in. These questions were formulated either using a ratio scale (the years

ID	Description with references to sources
P01	**Available / On-site customer.** The customer/user is available and able to answer questions regarding requirements in such a way that the response time does not negatively impact the work of the development team (in particular, works on-site with developers) [1, 14, 60, 76, 102, 112, 133, 151, 161, 162, 207].
P02	**Involve different stakeholders.** The stakeholders in the project are identified and have their roles and responsibilities defined. The analysis of requirements is performed by taking into account similarities (overlaps), inconsistencies and contradictions between different stakeholders' viewpoints, functional areas, and quality expectations [56, 161, 162, 191, 208, 237].
P03	**Establish project's shared vision.** The problem to be solved (goal of the project) and the proposed solution are well defined and kept up-to-date [2, 50, 60, 93, 208, 232].
P04	**Create prototypes to better understand requirements.** A prototype is created for functional requirements (e.g., wireframes, mockup, storyboard, etc.). The customer validates the created prototypes and provides feedback [2, 11, 47, 71, 112, 157, 166].
P05	**Define project / product constraints.** Project and product constraints are defined and kept up-to-date[208, 237].
P06	**Organize everyday team meetings.** The team discusses the progress, evaluate the feasibility of iteration, plan and adjust the plan (if it is required) (e.g., each team member states what he/she was able to accomplish on the previous day, what he/she is going to work on that day and informs about impediments) [2, 16, 60, 213].
P07	**Organize review meetings.** A meeting is conducted after an iteration during which the team presents its goal/scope, the work completed, the key decisions and a demo of the completed work. The feedback from stakeholders shall be received [2, 71, 76, 60, 112, 213].
P08	**Organize retrospective meetings.** After each finished iteration, a meeting is organized to discuss all issues that appeared during the iteration and to define improvement actions [2, 60, 112, 213].
P09	**Provide and maintain informative workspace.** Along with face-to-face communication, team members have constant access (either in the workplace or through a software system) to information such as project status. e.g., a burndown chart, completed tasks, currently developed tasks, awaiting tasks, requirements, reference materials such as law regulations, business documents etc. [37, 49, 60].
P10	**Provide easy access to requirements.** The requirements are stored in a place agreed upon by all project stakeholders, e.g., in a software system, that allows easy access for the development team, customer, and all authorized stakeholders[35].
P11	**Maintain information about 'bad smells' and best practices related to requirements.** The information about problems related to defects (or misunderstandings) in requirements and to requirement process (elicitation, analysis, documentation, verification) and their impact on a project is stored, kept up-to-date and available to team members [47, 193].
P12	**Perform the 'elevator test'.** All team members are able to pass the 'elevator test', i.e., to explain the idea of a product in the time it takes to ride up in an elevator [99, 163, 164, 188].
P13	**Let customer define acceptance tests.** Acceptance criteria and the corresponding acceptance test scenarios are defined by the customer, or at least the customer accepts them [14, 47, 58, 94, 96, 112, 165, 208].
P14	**Prepare and maintain automatic acceptance tests.** An automatic version of each acceptance test is implemented and kept up-to-date [14, 47, 49, 58, 112, 165, 208].
P15	**Prepare acceptance tests before coding.** Acceptance criteria and acceptance tests for each requirement are defined before a requirement is included into the scope of the iteration [2, 16, 71, 76, 80, 112, 134, 152, 153, 218].
P16	**Perform regression acceptance testing.** Acceptance tests for previously implemented functions are executed in the following iteration to verify if they are still meeting requirements [2, 146, 243].
P17	**Cover requirements with acceptance tests.** Acceptance criteria and acceptance test scenarios are defined for requirements (including non-functional requirements) [47, 112, 208].
P18	**Make requirements independent.** The requirements identified as dependent are grouped together into one or more independent requirements [48, 71, 76, 112, 233].
P19	**Write short, negotiable requirements.** Each requirement's description includes the most important information from the user's perspective, any additional information (extra precision) is separated — attached to the core requirement as annotations [47, 48, 71, 76, 112, 208, 233].
P20	**Make complex requirements divisible.** All sub-requirements of a compound requirement are identified in order to ease its decomposition, and the requirements identified as difficult to implement are augmented with the description of issues, concerns to help the development team extract some spike solution tasks [1, 18, 47, 48, 71, 76, 112].

Table 3.1: Agile RE Practices with their descriptions – Part 1.

ID	Description with references to sources
P21	**Requirements should be valuable to purchasers or users.** Each requirement is defined by the customer or at least the customer accepts the requirement proposed by the development team [24, 71, 76, 112, 233].
P22	**Make requirements estimable.** Size of each requirement is small enough to enable its effort estimation by the development team, and the domain vocabulary used for its description is clear to them [47, 71, 76, 112].
P23	**Make requirements testable.** It is possible to verify that the software meets the acceptance criteria in a reasonable time and using a reasonable amount of resources [47, 71, 76, 112, 233].
P24	**Follow the user role modeling approach.** An identification of the user roles is performed (e.g., brainstorming session), the dependencies between user roles and user classes are identified, and attributes, properties of each user role and associated user classes are defined (e.g., the frequency of system usage by user class members; users' level of expertise in domain) [47, 51, 61].
P25	**Let customer prioritize requirements.** The priority of each requirement is provided by the customer or at least accepted by him [2, 48, 76, 112, 213].
P26	**Define requirements using notation and language that can easily be understood by all stakeholders.** The notation used to document a requirement can easily be understood by all stakeholders [19, 101, 112, 121, 159, 207].
P27	**Assess implementation risks for requirements.** The implementation risks of each requirement are analyzed and documented [49, 71, 112].
P28	**Negotiate iteration scope with customer.** The scope of each iteration is negotiated between the customer and the development team until a consensus is achieved. The customer understands justifications for estimates of requirements included in the scope of the iteration and the development team understands the business rationale behind these requirements [2, 76, 112].
P29	**Avoid changing increment scope after it is agreed upon.** The scope of each iteration does not get changed after it has been started unless there is a mutual agreement between the development team and the customer (the scope is renegotiated) [112, 208, 213].
P30	**Keep iteration length short to continually collect feedback.** The length of iteration is no longer than 4 weeks [2, 18, 71, 112, 208, 213].
P31	**Define a fixed iteration length.** The length of each iteration is defined, justified, and does not change (excluding external causes like national holidays, etc.) [48, 112, 208].

Table 3.2: Agile RE Practices with their descriptions – Part 2.

of experience and number of projects) or as multiple-response true/false questions (more than one choice allowed). For the questions regarding domains and application types, we have adopted the classification scheme used by the ISBSG database [100].

The last, third part of the survey asked for any comments, questions, and remarks. Additionally, respondents could provide their email addresses to obtain the summary of results. The questionnaire was sent in two runs, so in the second one, the participants had to declare if they had already participated in the survey.

The initial version of the questionnaire underwent proofreading by a professional linguist and a pilot study among three members of our laboratory. Their feedback allowed us to introduce minor corrections in the wording of some questions (no major problems were identified).

The questionnaire was distributed using Google Docs Forms[2]. The printout version of the questionnaire is available on our website[3].

Population and sample

We defined our target population as agile software development projects' participants. We did not limit our focus to any specific types of applications or domains. However, we assumed that an individual belonging to the population did in fact participated in ag-

[2] http://docs.google.com

[3] http://lio.cs.put.poznan.pl/AgileRESurvey

ile projects (has at least one year of experience in agile SD). This is a wide group; however, it seems well-suited for the purpose — obtaining a general view on the importance of agile RE practices. Unfortunately, there is no obvious 'place' in which we could identify and access the population's representatives. Therefore, we could only rely on non-systematic sampling methods (convenience sampling). As Punter et al. [196] states, the non-systematic approach of drawing a sample implies that the accuracy cannot be determined (on the error between population and sample). However, we can determine the characteristics of the respondents afterward to characterize the sample.

We decided to use two sampling methods in parallel the balance the strong and weak points of each method (we discuss them briefly in Section 3.1.6). The main sampling method was self-recruiting [196]. We published invitations in social network groups related to agile and RE (LinkedIn, Facebook, Yahoo mailing groups). The secondary sampling method was to send direct invitations to people we knew to have experience in agile, the members of Scrum Alliance using the communication tool available on their website[4], and to people who published their curricula vitae on the Internet (mainly on LinkedIn) indicating that they have experience in agile software development.

Data analysis and validation

We decided to validate the responses using the following criteria: (1) a participant has to answer more than 75% of questions by providing responses other than "I don't have an opinion", and (2) has at least one year of experience in agile software development.

We decided to use several techniques to visualize and analyze the responses. The simplest was a frequency analysis, which is used to obtain a general overview of the importance of the practices. We also used a ranking algorithm and statistical methods to investigate relationships between the responses to demographic questions and perceived importance of the agile RE practices.

Relative importance ranking method The idea of a relative importance ranking is to investigate which of the practices are preferred over other practices by a respondent and to aggregate this information for the whole sample. If the practice P_A has a higher position in the ranking than the practice P_B, it means that P_A is more often evaluated higher than the remaining practices than the practice P_B against the same set of practices.

The ranking method is inspired by the PROMETHEE family of methods [17] and consists of the following steps:

(1) For each pair of practices P_i and P_j (i $\neq$ j) and respondents r_1, ..., r_n calculate the number of times P_i is evaluated by the respondent r_k (k = 1, ..., n) higher than P_j, and denote it as $d(P_i, P_j)$ ("critical" > "important" > "additional").

(2) For each pair of practices determine the preference degree $\pi(P_i, P_j)$ by performing one-tail Exact Binomial Test for equal probability (the alternative hypothesis: Prob$[d(P_i, P_j)]$ > Prob$[d(P_j, P_i)]$) with the significance level α set to 0.05. If the null hypothesis is rejected, it means that it is highly probable that, in the population, the practice P_i is more often preferred over P_j than the opposite. In such a case, assign $\pi(P_i, P_j) = 1$, otherwise $\pi(P_i, P_j) = 0$.

(3) Calculate the ranking score $R(P_i)$ of each practice by summing up the $\pi(P_i, P_j)$.

(4) Sort the practices by $R(P_i)$ in descending order.

It is worthwhile to mention that the ranking is created based on direct comparisons between practices. We called it "relative" because we investigate how a single respondent evaluates each pair of practices. As a result, we can say that some practice P_A (a higher

[4] https://www.scrumalliance.org

position in the ranking) is more often indicated as more important than some other practice P_B in the eyes of a single respondent.

Bivariate analysis of responses We used two approaches to investigate relationships between the perceived importance of the practices and the responses given on demographic questions depending on the type of measurement scales used:

- Multiple response categorical variable (MRCV) — we used a test for multiple marginal independence (MMI) between one single response categorical variable (SRCV) and one MRCV with Bonferroni correction available in MRCV R package [136] (the significance level α set to 0.05).
- Ratio scales — we used a non-parametric Kendall's τ correlation coefficient because the perceived importance of the practices was measured on an ordinal scale. We also used a test for association between paired samples for Kendall's τ [104] available in R statistical package [197] (the significance level α set to 0.05).

The nature of the questions asked in the survey does not allow explaining the existence of particular relationships. Therefore, we decided to search for further explanations in the available systematic literature reviews (SLRs) and mapping studies on agile software development and testing [4, 26, 31, 54, 55, 60, 66, 81, 105, 112, 117, 118, 199, 210, 215, 216].

Analysis of open-text questions We used the guidelines of Charmaz [34] to analyzed the open-text responses. First, each researcher individually performed initial coding incident-by-incident of the statements or short sentences provided by participants. As the initial codes, we used the list of agile RE practices to which the researchers added the new codes that emerged. Then, in vivo codes underwent constant comparison. Next, we discussed the results together, proposed the final schema of codes and performed axial coding. Secondly, we performed individually coding with the final codes, and at the end of the process, we discussed the results.

Popularity/adoption-level ranking We created an alternative ranking of agile practices based on the results of five, recently conducted survey research studies (2014-2017) regarding the level of popularity of agile practices [12, 64, 124, 186, 231]. We sorted the practices in descending order based on their average reported frequency of use.

We used Kendall's τ coefficient to investigate correlations between the rankings of relative importance and popularity/adoption levels of the agile RE practices.

3.1.5 Results and discussion

Characteristics of the respondents

The invitations to participate in our survey were sent in two runs: between the 9th of October and 20th of December 2015 (the responses were collected until the end of June 2016), and between 21 December 2017 and 11 January 2018 (the responses were collected until 21 February 2018). We received 138 responses (95 in the first run, 43 in the second run). Two of them did not meet our validation criteria (see Section 3.1.4). We received the following numbers of responses from different sources:

- Social media interests groups related to agile or Requirements Engineering = 74 (72 after rejecting two responses)
- Direct invitations sent via Scrum Alliance community page = 12
- Direct invitations to known persons = 23
- Direct invitations based on CVs = 29

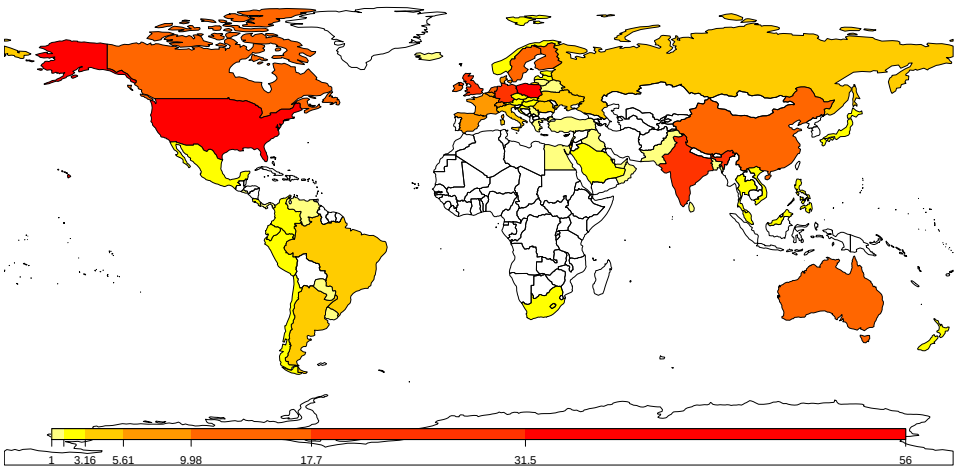


Figure 3.1: Countries in which the agile projects that respondents participated in were conducted (multiple choices allowed). (Figure taken from [176].)

We obtained a wide coverage of countries around the globe in which the projects of our respondents were conducted (see Figure 3.1). The biggest number of projects were conducted in the US, Poland, UK, India, and Germany.

The coverage of domains and types of software applications developed by participants was also high. As shown in Table 3.3, the four most dominating domains were services, financial, telecommunication, and banking. The most frequently developed types of applications were web and mobile applications, financial applications, and management information systems.

The most popular agile methods were Scrum and Kanban. Slightly less popular were eXtreme Programming (XP) and Scrumban. Only a few respondents mentioned other methods such as DSDM, SAFe, or Crystal Clear.

The working experience also spread nearly through all of the activities we asked for in the survey. Although the dominating responsibilities were Scrum Master (or equivalent), project manager, and requirements analyst, we also had many participants with experience in quality assurance (QA), testing, programming, and software designing. Additionally, the participants were responsible for: coaching, help desk, product management, research, client relationship management, and UX design. The participants had on average 16 years of experience in SD and more than 7 years in agile projects. The average number of agile projects was 22; however, the meaning of the term project could be understood differently by the participants.

The participants had experience in agile projects of different sizes, starting from small projects having up to five members of a team and total development effort smaller than one man-month, up to very big projects with more than 50 people involved and total development effort greater than 80 man-months.

Perceived importance of the practices

The summary of the responses on the perceived importance of the agile RE practice is presented in Figure 3.2.

The results show that all of the agile RE practices seemed important to the participants. However, none of the practices was unanimously perceived as critical. Nevertheless, there were at least four practices that were perceived as really important having at least 85% of responses “important” and “critical”: P01: Available / on-site customer (91%),

Domains (N=136, multiple choices allowed)			
Services	66	Insurance	37
Financial	53	Government	34
Telecommunication	50	Entertainment	33
Banking	48	Trading	26
Electronics&Computers	41	Manufacturing	20
Medical&Health Care	41	Other	30

(a) Domains in which the respondents' agile projects were conducted

Types of applications (N=134, multiple choices allowed)			
Web or e-business	87	Document management	47
Mobile apps	65	Billing	42
Financial	59	Sales&Markteting	39
Mgmt Inform. Systems	59	Personnel	29
Electronic data interchange	57	Logistics	29
Transaction or production systems	47	Trading	27
		Others	25

(b) Types of applications developed in the respondents' agile projects

Methodologies (N=83, multiple choices allowed)			
Scrum	78	Scumban	22
Kanban	50	DSDM	6
XP	33	Others	21

(c) Methodologies used in the respondents' agile projects

Responsibilities (N=136, multiple choices allowed)			
Scrum Master	104	Programmer	61
Project manager	96	Soft. Designer	66
Req. Analyst	91	Coach	13
TestingQA	72	Others	12

(d) Responsibilities of the respondents in agile projects

Size of teams (N=118, multiple choices allowed)			
<=5	57	20-30	17
5-10	89	30-50	20
10-15	52	>50	26
15-20	30		

(e) Sizes of teams in the respondents' agile projects

Total development effort [man-month] (N=126, multiple choices allowed)			
<=1	22	8-20	65
1-2	35	20-40	49
2-4	56	40-80	41
4-8	61	>80	49

(f) Total development effort in the respondents' agile projects

Table 3.3: The results with respect ot the agile projects in which the respondents participated.

P03: Establish project shared vision (91%), P07: Organize demo meetings (85%), P23: Make requirements testable (88%).

Contrary, only the practice P12: perform the Elevator Test seemed to be perceived as rather optional (still, around 37% respondents evaluated it as at least "important"). The other practices that were both perceived as at most "additional" and at least "important"

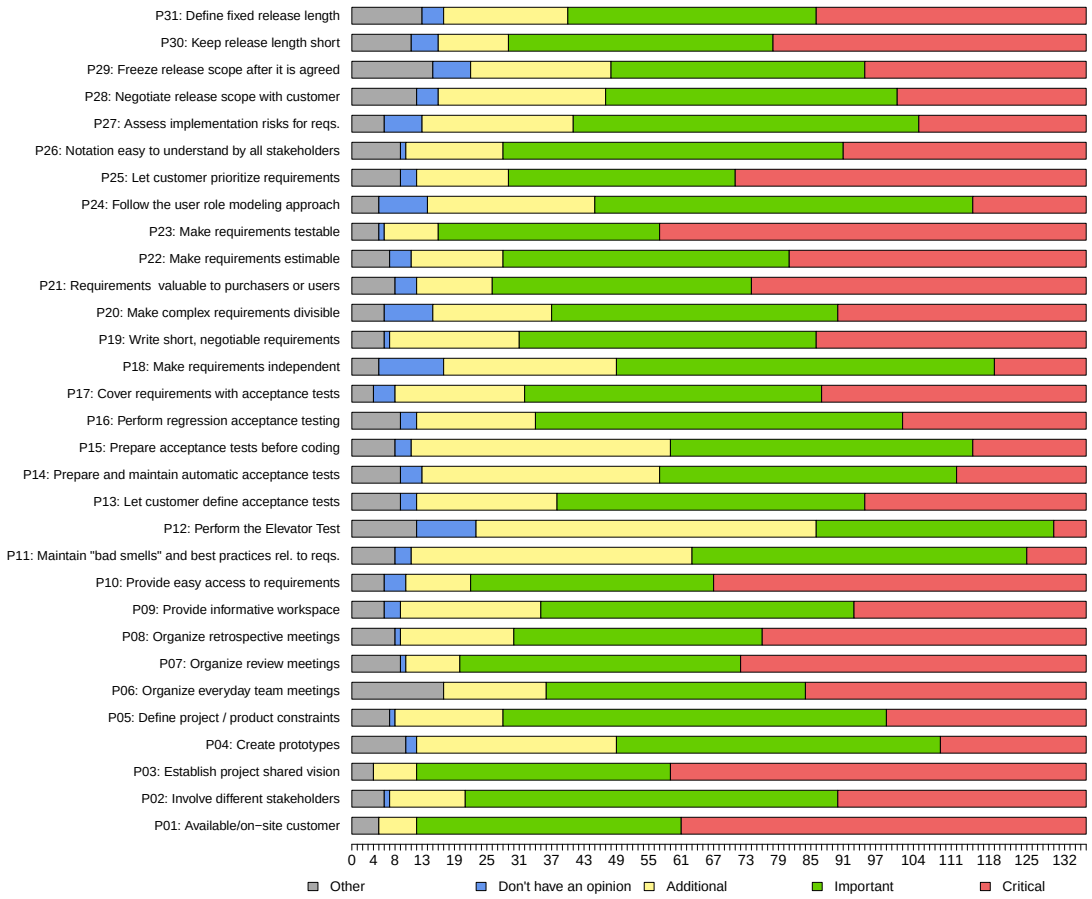


Figure 3.2: Perceived importance of the agile RE practices. (Figure taken from [176].)

by a similar number of participants were: P11: Maintain "bad smells" and best practices related to requirements (38% at most “additional” and 54% at least “important”), P14: Prepare and maintain automatic acceptance tests (32% at most “additional” and 58% at least “important”), and P15: Prepare acceptance tests before coding (35% at most “additional” and 57% at least “important”).

There was also a group of practices for which we recorded a similar, large numbers of extreme responses (the difference between the number of responses “additional” and “critical” was at most 10% and together they contributed at least 40% of all responses): P16: Perform regression acceptance testing (16% “additional”, 25% “critical”), P28: Negotiate iteration scope with customer (23% “additional”, 26% “critical”), P27: Assess implementation risks for requirements (21% “additional”, 23% “critical”), P04: Create prototypes to better understand requirements (20% “additional”, 27% “critical”).

The analysis of the open-text responses resulted in a hierarchy of codes for each practice describing the causes of why the respondents could not decide on the importance of certain practice.

The first reason was that the respondents claimed that the importance of ten practices (P01, P03, P04, P05, P14, P18, P19, P22, P24, P27) depends on the context. Most frequently they just left general statements such as “*Context dependent. May be good in some cases and not so useful in other cases*”. However, one person pointed out that writing short, negotiable requirements (P19) depends on the team members. Then, other two persons spotted that the importance of assessing implementation risks for requirements (P27)

depends on two factors: "*on how good your team are at dealing with risks as they surface*" and "*on the criticality of system you are building*".

Secondly, the respondents identified six practices (P04, P06, P07, P08, P09, P10) as implementation-depended. They indicated that the way a practice is executed (implemented) influences how important it is. For example, four persons pointed out that it is crucial how prototypes are created (P04); one of them used such justification: "*very lightweight prototypes can be really useful, but sometimes teams spend way too much time on prototyping*". One person remarked that having an environment in which it is possible to deploy new features really fast makes this practice useless. Another four respondents claimed that meetings could be spoiled when improperly organized. One of them claimed: "*meetings waste time, especially teleconferences, everybody is getting boring, and nothing follows from them, better is to discuss everything electronically*".

Demographic factors and practice importance

We performed a bivariate analysis to investigate if the perceived usefulness of the practices depends on the experience of the survey respondents. We only considered those relationships between practices and demographic questions for which we were able to reject the null hypotheses of the statistical tests (see Section 3.1.4 for details). We also excluded categories of demographic questions that had ten or fewer responses. Otherwise, we would finish with sparse contingency tables that would negatively affect the reliability of the statistical tests.

The identified relationships are presented in Figure 3.3. Since the considered sample is relatively small comparing to the potential size of the population, it is not justified to make strong conclusions about the nature of the relationships. However, we hope that the brief description of the relationships presented in this section will stimulate discussion and trigger future research directions.

Countries. We identified ten relationships between the countries in which the projects were run and perceived importance of practices. Usually, there were two or three countries for which we observe a visible difference in the distribution of the importance evaluations (P01: CN, PL; P08: US, PL; P12: IN, PL; P15: UK, PL; P19: UK, IN, CN, PL; P20: CN, PL; P23: CA, PL; P24: IN, PL; P26: CN, DE; P28: CA, CN).

We reviewed the available SLRs and mapping studies (see Section 3.1.4 for references) to look for empirical evidence that would allow us explaining the observed differences in perceived importance of the practices shown above. Although some of the reviews include primary studies concerning human and social factors related to agile software development [66, 216] or globally distributed software development [54, 105], they do not identify any country-specific factors that could justify different views on the importance of the agile RE practices. However, the survey research studies discussed in Section 3.1.2 show that the adoption level of agile methods differs between countries. It could be one of the factors influencing the perceived importance of the practices. Other potential reasons could be the difference in cultural factors (e.g., the way people communicate, the structure of society), legal regulations, or the structure of IT market (e.g., a dominance of outsourcing software development).

Responsibilities. We indicated five relationships between performing certain activities and the perceived importance of practices. The unexpected observation was that the practices that are supposed to support developers, like organizing everyday team meetings (P06), providing informative workspace (P09) were perceived less important by them than by people responsible for requirements or Scrum Masters/project managers. Also, the developers valued the availability of customer (P01) less than the people having other responsibilities (it was mostly "critical" for people performing Scrum Master's or equiva-

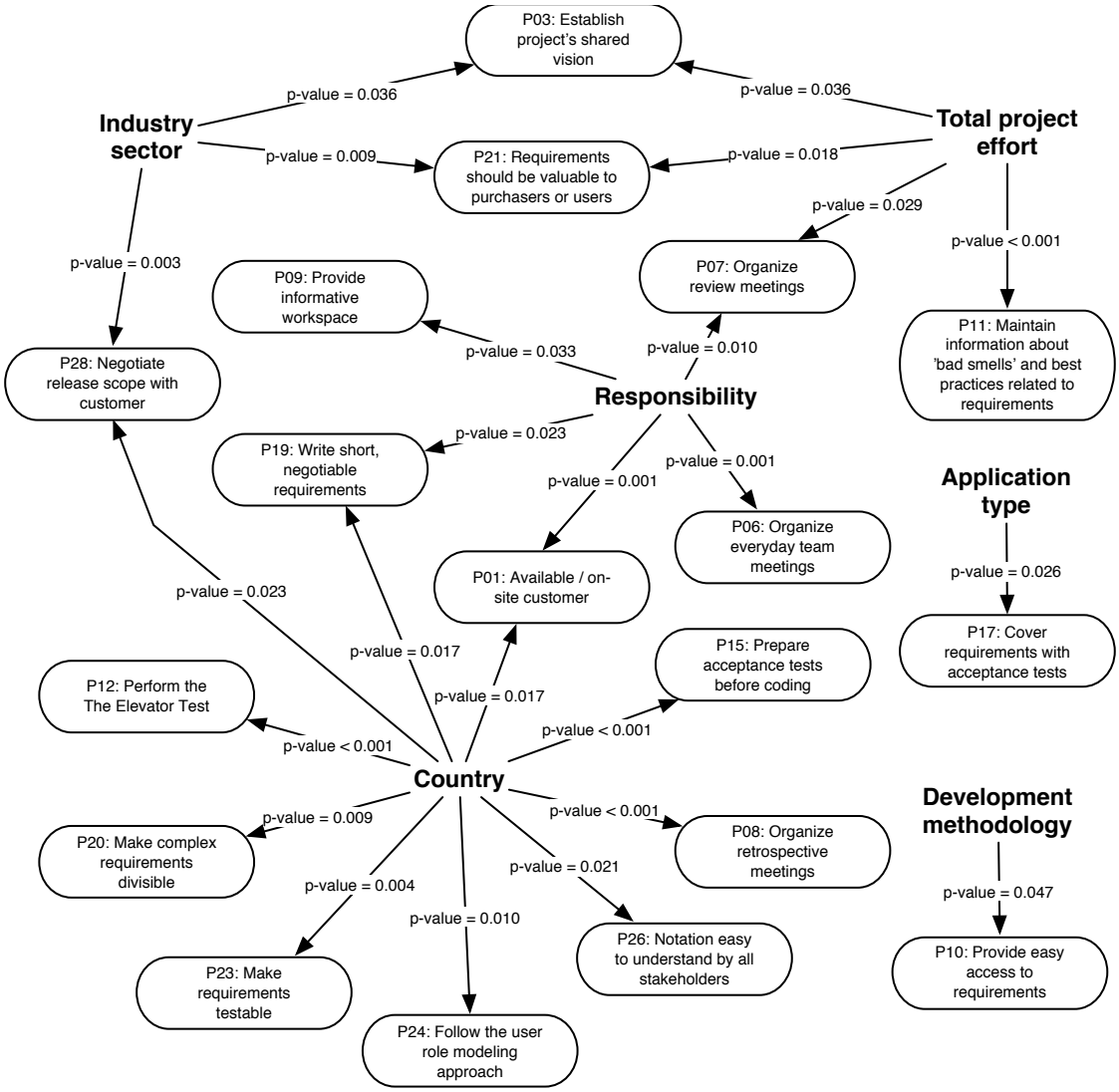


Figure 3.3: The relationships between the perceived importance of the agile RE practices and demographic information. (Figure taken from [176].)

lent tasks). Finally, the developers/designers perceived organizing review meeting (P07) as slightly less important than for the people performing tasks related to RE, management, or testing.

We investigated the same set of SLRs and mapping studies as before to look for the explanations for the observed relationships. Some of the studies reported that the daily stand-up as valuable to developers and customers [66, 111, 150]. These observations are convergent with some of the opinions we received as open-text responses. Some developers perceived everyday team meetings as a waste of time, especially when they were badly organized. Other claimed that such meetings are not necessary to correctly organize the work of development teams. These observations only partially explain some of the observed relationships. Also, we were not able to find any evidence that would explain why developers perceive the practice of preparing an informative workspace less important than the Scrum Masters/project managers, QA, or people responsible for requirements.

Software development methodology. We observed that providing easy access to requirements (P10) was more important to respondents having experience in Kanban. This

observation seems rational because of the dynamics of task assignment in Kanban. Since the priorities of the tasks may change very quickly, it seems important to be able to switch contexts easily (which is difficult if you do not have access to requirements).

Total project effort. We observed that with the increase in the size of projects with respect to the total project effort people tend to perceive the need of organizing review meetings (P07) as more critical. Contrary, people who participated in smaller projects valued more practices such as establishing project's shared vision (P03), having requirements valuable to purchasers or users (P21), and maintaining information about "bad smells" and best practices related to requirements (P11). Interestingly, most of these practices regard delivering software that is really valuable to stakeholders.

Domain. We observed that negotiating the scope of release/iteration with a customer (P28) is perceived as more critical by the respondents working in banking, electronics, manufacturing, medical & health care, and financial domains. Also, the results suggest that in manufacturing or trading sectors the practice of ensuring that requirements are valuable to purchasers or users (P21) is less critical. Contrary, people who worked in medical & health care perceived this practice as critical more often the those working in other domains. Having the well-established vision of the project (P03) was perceived as "critical" by nearly all of the respondents who worked in trading domain.

Application type.We identified only one relationship. Covering requirements with acceptance tests (P17) was indicated as more important for respondents developing management information systems (MIS).

Years of experience and number of agile projects. The overall experience of the respondents seems to have an important role in how they perceive the importance of the agile RE practices. We observed that with the increase of experience in agile, the respondents value more the access to the customer (P01), team retrospectives (P08), having acceptance tests (P17) and making them automatic (P14), and negotiating the scope of releases/iterations with a customer (P28). Interestingly, we also observed a negative correlation between the years of experience in agile projects and the practice of maintaining information about "bad smells" and best practices related to requirements (P11).

There was also a group of practices for which the perceived importance correlated only with the overall number of years of experience, but not with the number of years of experience in agile, i.e., acceptance tests defined by a customer (P13), regression testing (P16), testable requirements (P23), and having requirements valuable to purchasers or users (P21). It seems that people value those practices more if they have longer experience in non-agile projects because those practices could have been either missing in those projects or so important that affected their opinion and, as a result, propagated it on agile projects as well.

Relative importance of the practices

We used the ranking method (see Section 3.1.4) to aggregate responses and create the ranking of relative importance. The resulting ranking is presented in Figure 3.4.

Based on the constructed graph of relative importance, the practices could be divided into *tiers*. The boundary of a tier is determined by practice P_A which has the lowest score and still is dominated only by practices from the higher tiers (tiers with lower index numbers).

The most important practices (Tier 1) could be characterized as those that seem fundamental for propelling a fast pacing, iterative, agile project without a fully predefined scope and emergent requirements. First of all, the agile methods emphasize delivering working software. Thus, the team should be able to verify if a released increment satisfies the requirements (P23). It is of similar importance that a project team needs to share a

Table 3.4: Correlation between the participants' years of experience, number of projects and evaluation of the practices.

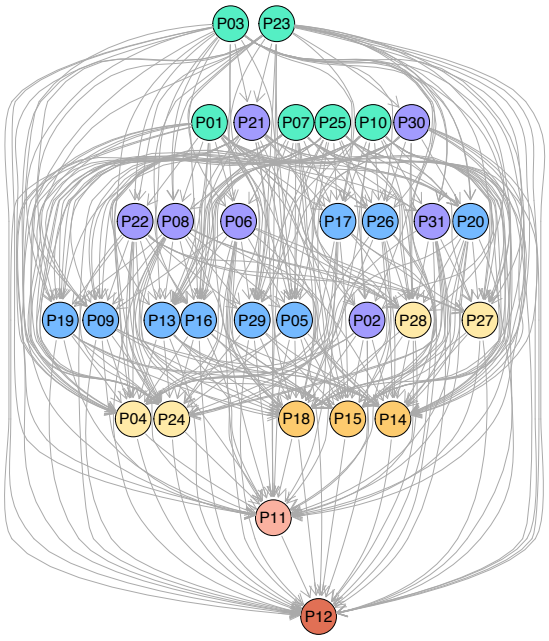
	Years of exp.		Years of exp. agile		No. of agile projects	
	τ	p-value	τ	p-value	τ	p-value
P01	**0.19**	0.009	**0.19**	0.009	**0.15**	0.044
P02	0.10	0.168	-0.01	0.898	0.05	0.444
P03	0.12	0.084	0.11	0.116	**0.25**	0.001
P04	-0.01	0.875	-0.09	0.224	-0.12	0.104
P05	0.02	0.738	0.00	0.987	-0.01	0.841
P06	-0.03	0.641	-0.08	0.273	-0.02	0.777
P07	0.06	0.396	0.06	0.419	0.14	0.060
P08	**0.22**	0.002	**0.18**	0.015	0.12	0.100
P09	0.06	0.424	0.07	0.363	**0.15**	0.031
P10	0.00	0.964	0.00	0.987	0.01	0.881
P11	-0.09	0.221	**-0.24**	0.001	**-0.20**	0.007
P12	0.13	0.091	0.08	0.328	0.15	0.054
P13	**0.17**	0.017	0.05	0.522	0.05	0.487
P14	**0.19**	0.007	**0.18**	0.013	0.08	0.243
P15	0.10	0.164	0.02	0.789	-0.02	0.731
P16	**0.16**	0.028	0.12	0.108	0.06	0.439
P17	**0.31**	0.000	**0.22**	0.002	0.11	0.124
P18	0.08	0.274	0.01	0.877	0.06	0.397
P19	0.01	0.907	-0.02	0.750	0.03	0.717
P20	0.11	0.142	0.09	0.214	0.12	0.115
P21	**0.18**	0.013	0.07	0.316	0.13	0.068
P22	0.05	0.485	-0.06	0.434	-0.04	0.544
P23	**0.16**	0.028	0.11	0.140	0.07	0.335
P24	-0.11	0.122	-0.09	0.205	-0.08	0.247
P25	0.06	0.435	-0.02	0.752	0.00	0.971
P26	**0.14**	0.045	0.04	0.575	0.05	0.502
P27	-0.10	0.168	-0.05	0.466	-0.07	0.305
P28	**0.15**	0.037	**0.16**	0.027	**0.20**	0.006
P29	0.06	0.440	-0.01	0.908	-0.01	0.927
P30	0.05	0.490	0.02	0.743	0.12	0.094
P31	-0.01	0.905	0.04	0.587	0.12	0.119

*) Kendall's rank correlation τ is not equal to zero with $\alpha = 0.05$

clearly defined vision (P03) to push the development in a right direction. There is also a need for close collaboration with the customer (P01: Available/on-site customer, P10: Provide easy access to requirements, P25: Let customer prioritize requirements), and receiving frequent feedback (P07: Organize review meetings).

The practices in Tier 2 also focus on iterative delivery of valuable software product, e.g., requirements valuable to purchasers or users (P21,)keeping releases short and predictable (P22, P30, P31), and getting feedback on a product, plan, and how the team operates (P06, P08). Altogether it is the way how agile teams deal with requirements validation [112, 95].

The set of most important practices seems convergent with the findings of other studies. Nine out of twelve practices in Tier 1 and Tier 2 were identified as critical success factors (CSFs) [37, 77, 224]: continuous delivery of valuable, working software in short time scale (P21, P30), sustaining a constant pace (P31, partially P22), face-to-face conversation within team (partially P06, P07, P08), customer involvement (P01), delivering most important features first (P25). Also, two first tiers of our ranking include five out of seven agile



	Practice	Score	Rank Sensitivity* mean	Rank Sensitivity* SD	Tier Stability**
Tier 1	P23: Make requirements testable	25	0.0	0.0	100%
	P03: Establish project shared vision	24	0.0	0.0	100%
	P01: Available/on-site customer	23	0.0	0.0	100%
	P07: Organize review meetings	22	0.0	0.0	97%
	P10: Provide easy access to requirements	20	-0.9	2.1	100%
	P25: Let customer prioritize requirements	19	0.0	0.0	56%
Tier 2	P21: Requirements valuable to purchasers or users	16	0.0	0.0	98%
	P30: Keep release length short	16	0.0	0.5	77%
	P08: Organize retrospective meetings	11	-0.3	0.5	98%
	P22: Make requirements estimable	11	0.0	0.1	100%
	P06: Organize everyday team meetings	9	0.3	1.1	98%
	P31: Define fixed release length	4	0.0	0.0	80%
	P02: Involve different stakeholders	1	-0.7	0.9	79%
Tier 3	P17: Cover requirements with acceptance tests	1	0.0	0.0	96%
	P20: Make complex requirements divisible	1	-0.2	0.9	96%
	P26: Notation easy to understand by all stakeholders	0	-0.1	2.1	91%
	P05: Define project / product constraints	-3	0.0	0.0	92%
	P16: Perform regression acceptance testing	-3	1.9	3.8	95%
	P19: Write short, negotiable requirements	-3	0.0	0.0	99%
	P09: Provide informative workspace	-4	0.0	0.0	96%
	P13: Let customer define acceptance tests	-4	0.0	0.0	99%
	P29: Freeze release scope after it is agreed	-5	0.0	0.0	94%
Tier 4	P28: Negotiate release scope with customer	-9	0.1	0.5	71%
	P27: Assess implementation risks for reqs.	-10	0.0	0.0	90%
	P04: Create prototypes	-19	-0.2	1.6	86%
	P24: Follow the user role modeling approach	-20	0.0	0.0	88%
Tier 5	P18: Make requirements independent	-21	0.3	1.5	64%
	P14: Prepare and maintain automatic acceptance tests	-23	0.2	1.1	94%
	P15: Prepare acceptance tests before coding	-23	0.0	0.0	61%
Tier 6	P11: Maintain "bad smells" and best practices rel. to reqs.	-26	0.0	0.0	71%
Tier 7	P12: Perform the Elevator Test	-30	-3.3	4.3	74%

** The mean difference in the ranking positions (and standard deviation); 100 reps. and 90% subsample*

*** The percentage of times a given practice was assigned into the tier; 100 reps. and 90% subsample*

Figure 3.4: The ranking of relative importance of the agile RE practices. The graph shows the preference degree $\pi(P_i, P_j)$ between the practices. The existence of an arc between P_i and P_j means that $\pi(P_i, P_j) = 1$. (Figure taken from [176].)

RE practices identified by Cao and Ramesh [30]. Also, all six of the agile RE practices from tiers 1 and 2 that appeared in the study by Fontana et al. [76] were indicated as practices characteristic for a high level of maturity. Although neither of these studies considered our top-ranked practice—establishing a shared vision (P03), the "unclear vision of goals" was considered as one of the causes of communication gaps within the project teams [21, 112]. Also, a similarly named practice—"whole" multidisciplinary *team with one goal*—appears in the list of essential practices for a team to be considered agile composed by Williams [239].

Tier 3 consists of more specific practices related to different aspects of software development. Most of them concern the way the requirements, constraints, and acceptance tests should be defined (P05, P13, P16, P17, P19, P20, P26). The second group relates to communication between stakeholders (P09, P29).

The remaining tiers include many practices that were perceived as supplementary, e.g., P12: Perform the Elevator Test and P11: Maintain "bad smells" and best practices related to requirements. However, still few respondents indicated them as critical. Surprisingly, these tiers also include some practices that have gained lots of attention in recent years, e.g., automated acceptance testing (P14), test-driven development (P15), or prototyping (P04). The question arises, why are these practices perceived as less important for an agile project?

Among the many possible hypotheses, the one that seems to have the strongest support in the collected data is that these practices could be beneficial but only in specific project contexts. Otherwise, they are either hard to implement correctly or do not provide the expected benefits. Several arguments are supporting this hypothesis. The practices related to prototyping and automated acceptance testing received 15–25% of responses indicating them as critical (P04–20%, P14–18%, P15–15%, P16–25%). Therefore, there are certain contexts in which they play major roles. At the same time, they seem to be the least frequently adopted by agile teams (see Figure 3.5). Similar observations could be found in the literature. For instance, Inayat et al. [112] point out that despite its benefits, prototyping can be expensive and problematic, especially in the case of an exceptional increase in client requirements. There are also problems in synchronizing the creation of prototypes and development [55, 139, 144]. Similarly, the automatic and test-driven development seems to be one of the most critical enabling practice for quality in the agile software development [215]. However, there also false expectations regarding automatic testing and many challenges related to the maintenance of such tests [111, 199].

Finally, we calculated two diagnostic measures to investigate how much the ranking could be affected by a sample under study (calculated based on 100 subsamples of the sample under study by randomly selecting 90% of the main sample):

- Rank Sensitivity — the average difference between the rank of the practice in the reference ranking and the ranking created based on 90% subsample (and standard deviation).
- Tier Stability — the percentage of rankings in which a given practice was assigned to the same tier as in the reference ranking.

The calculated measures are presented in Figure 3.4. Based on the results, we could tell that in overall ranking is stable with the exception of a few practices that usually lies on near to the tiers boundaries.

Perceived importance vs. popularity of the practices

The created ranking of popularity/adoption level of agile RE practices is presented in Figure 3.5 (see Section 3.1.4). It includes twelve of the considered agile RE practices since the selected survey research studies directly covered only these practices. Few other agile

Score	Practice	Rank		Rank	Practice	Avg. Usage
23	P01: Available/on-site customer	3		1	P08: Organize retrospective meetings	83%
22	P07: Organize review meetings	4		2	P07: Organize review meetings	81%
16	P30: Keep release length short	8		3	P30: Keep release length short	72%
11	P08: Organize retrospective meetings	9		4	P06: Organize everyday team meetings	71%
9	P06: Organize everyday team meetings	11		5	P18: Make requirements independent	63%
4	P31: Define fixed release length	12		6	P31: Define fixed release length	59%
-3	P19: Write short, negotiable requirements	19		7	P19: Write short, negotiable requirements	56%
-4	P09: Provide informative workspace	20		8	P01: Available/on-site customer	50%
-19	P04: Create prototypes	25		9	P09: Provide informative workspace	46%
-21	P18: Make requirements independent	27		10	P04: Create prototypes	45%
-23	P14: Prepare and maintain automatic acceptance tests	28.5		11	P14: Prepare and maintain automatic acceptance tests	42%
-23	P15: Prepare acceptance tests before coding	28.5		12	P15: Prepare acceptance tests before coding	28%

Figure 3.5: Correlation between popularity/adoption-level and perceived importance rankings of agile RE practices.

RE practices were also partially covered by these studies in that sense that they concern more specific aspects of the twelve practices included in these studies (e.g., INVEST practices cover specific aspects of the practice of writing user stories). However, we decided to consider only these practices for which we had direct evidence.

We observed that both rankings are highly correlated (Kendall's τ = 0.595, p-value = 0.005). Therefore, the popularity of practices could be considered a good predictor of how people perceive their importance. It can also mean that agile teams most frequently adopt practices that are recognized for their importance by agile community. However, the differences between both rankings suggest that the other factor driving the popularity of the practices could be their ease of implementation. For instance, one of the most challenging practices to adopt P01: Available/on-site customer [66, 112] was ranked visibly lowered for its popularity than for its importance. On the contrary, practices like P08: Organize retrospective meetings, P18: Make requirements independent, or P06: Organize everyday team meetings were indicated as more popular than important.

3.1.6 Threats to validity

We performed the analysis of validity threats based on the guidelines provided by Wohlin et al. [241].

Conclusion validity. There are at least two threats to validity related to the bivariate analysis of the practice-importance and demographic questions. These threats, if materialized, lowered the chance of rejecting the null hypotheses of the MMI statistical tests. The first one regards the fact that MMI test assumes that variables are expressed on nominal scales while in our study only answers for demographic questions used nominal scales while the importance of the practices was evaluated on an ordinal scale. Consequently, the additional information about the order of items was ignored by the tests lowering the chance of detecting relationships between variables. The second threat regards using Bonferroni correction, which is known to be very conservative.

There is also a threat regarding the influence of the scale used to evaluate the importance of practices on the method of constructing the ranking. If a participant responded "don't have an opinion" or "other" to practice, we would not be able to use this information to evaluate the relative importance of the practice. The worst-case scenario would be that none of the respondents provided importance evaluation for a pair of practices or

the number of the responses is too small to have reasonable chances for rejecting the null hypothesis if it is false. To investigate if this threat materialized in our study, we checked the number of responses for each pair of the practices. The min. number was 50, the median 69, and the max. 93. Even if we considered the min. number of responses (50) and a small effect size (0.20 – as suggested by Cohen [46]), the power of the binomial test would be very high = 0.86 (for 69 responses = 0.96, and for 93 = 0.99).

Another threat relates to the reliability of the scale used to evaluate the importance of the practices for agile projects. Our three-point scale ("additional", "important", "critical") is still subjective and could be interpreted differently by respondents depending on their culture, knowledge, experience, etc. We also allowed providing opened-text answers or resigning from providing any response to avoid biasing the results by forcing the respondents to answer about the practices they had not had any experience with or their experience does not allow them to answer using the proposed scale. The second issue is that we only considered positive (or neutral) influence of the practices on the projects. We assumed that if a respondent had a negative experience with some practices, he/she could express that using the "other" option (none of the respondents shared such opinions).

Finally, we used a qualitative, coding technique to analyze open-text responses. Because the technique is subjective by its nature, we could have introduced some bias to the conclusions made based on the analysis.

Construct validity. Agile practices are not formally defined. Also, most of them have evolved over time and could be understood and performed differently by different people and organizations. To mitigate the impact of this issue on our study, we identified and selected practices based on available studies and tried to capture their essence while creating our catalog of agile RE practices. Still, the provided definitions might differ from those found in some literature positions. And, secondly, the catalog of might miss some practices that were not mentioned in the studies that form the basis for our chapter.

Internal validity. We decided to send invitations to members of agile community forums (LinkedIn, Facebook, etc.) to increase the reach of our survey. This approach limited our control of the response-collection process. As a result, we are not able to reliably determine how many people received and read our invitation (e.g., invitation messages could be classified as spam, some of the members could be inactive). Also, there is a question of trustworthiness of the respondents. The respondents could intentionally try to influence the results of the survey. However, the results of the sensitivity analysis show that it would not have a visible impact on the outcomes of the study.

We sent the invitation to participate in our study twice–in two runs–to receive more responses. To ensure that we ask each participant only once, we added a question directly asking about the participation in the first run (the positive answer eliminated the participant's response from the further analysis).

Another threat to internal validity regards the technical skills required to respond to the survey. We assumed that the members of the target population would not have problems in responding to an on-line survey and are fluent enough in English to understand the questions. We also conducted a pilot study to ensure the questionnaire correctness and ease of understanding. A professional linguist also proofread the descriptions of the practices. We believe that the assumption was reasonable because English is a lingua franca of computer science. Besides, it turned out that ca. 86% of respondents participated in projects conducted in countries having English as an official language.

When it comes to the commitment of the respondents, we did not observe any suspicious responses that would suggest the lack of commitment or intent to sabotage the study. In addition, we believe that the topic was attractive to the participants because 72% of them voluntary shared their e-mail addresses to receive the summary of results.

The last two threats to internal validity regard the way the importance of the practices was evaluated. To collect strong evidence we asked the participants about their overall opinion regarding the importance of agile RE practices. However, firstly, we do not know what particular events in the respondents' projects determined how they perceived the importance of practices (e.g., was it a single project that failed because of lack of certain practices, or was a practice essential for all the projects?). Secondly, many of our respondents had long-term experience in agile projects, and thus their opinions could change in time (e.g., some facts from the earlier projects might have been blurred when they were filling in the questionnaire).

External validity. We intended to conduct our study among people who have experience in agile software development. We collected responses from participants taking part in projects conducted in various countries and domains, also performing different project roles. Still, there are some threats related to the representativeness of the sample that we should take into account.

Firstly, the sample of 136 respondents is still small comparing to the expected size of the population (however, this cannot be determined accurately). It may happen that the sample is too small and the obtained results could differ when a different sample is studied. On the other hand, the results of the sensitivity analysis of the ranking show that the number of respondents seems sufficient to create a stable ranking of the practices.

The second threat relates to the coverage of different projects' contexts. It seems that the respondents (as a group) had experience in various domains, types of applications, performing different project roles, and participated in projects all over the world. However, the number of observations for specific contexts differ visibly. Unfortunately, we cannot determine if the observed proportions reflect the ones in the population. Therefore, we cannot make strong conclusions regarding the relationships between the perceived importance of the practices and context factors. This part of the study should be taken as an exploratory analysis and needs further investigation.

The third threat regards using the non-systematic sampling methods. By sending invitations to social groups regarding agile and RE, we could reach people that are particularly interested in these topics (e.g., "agile enthusiasts"). Therefore, their view on agile practices might be somehow biased in comparison to other individuals in the population. We tried to mitigate this problem by sending direct invitations to people participating in agile projects (balancing the sample with respect to the sources).

Finally, w asked the respondents how important are the agile RE practices for agile projects. Consequently, the responses we received reflect their subjective view on the subject. Without triangulation, we cannot firmly state whether the *real* importance of the practices is equivalent to the *perceived* one. However, we do not concern this threat as significant since the results of the study seem, in general, convergent with the results of the studies regarding critical success factors in agile projects [37, 77, 224] and correlated with the adoption level of agile practices [12, 64, 124, 186, 231].

3.1.7 Conclusion

The main goal of this study was to investigate how people (in general) perceive the importance of agile RE practices for agile projects. To achieve this goal, we conducted a survey and asked 136 agile software development practitioners how they perceive the importance of the 31 agile RE practices that we had identified in a literature study.

Implications for agile practitioners

Based on the responses, we created a ranking of relative importance of the agile RE practices. It groups the practices into seven tiers. The practices from higher-ranked tiers are more often perceived as more important (by the same respondent) than the practices from the lower-ranked ones. The ranking complements the results of available survey studies on the adoption of agile practices as it evaluates them from a different perspective. Also, the method of constructing the ranking is unique in this sense that it allows for a direct, one-to-one comparison between the practices based on actual preferences of respondents rather than on the frequency of the responses.

The ranking includes some agile RE practices that have not been investigated by other survey studies. Interestingly, all of the considered practices were perceived as useful, and only one them (i.e., P12. Perform elevator test) received more responses indicating it as a supplementary rather than as an important or critical practice.

The top-ranked agile RE practices are:

- P01. Available/On-site customer
- P03. Establish the project's shared vision
- P06. Organize everyday team meetings
- P07. Organize demo meetings
- P10. Provide easy access to requirements
- P23. Make requirements testable

They can be summarized to a triple: **shared vision**, **customer collaboration**, and **continuous feedback / verification**.

We have also identified some relationships between the importance of the practices and the respondents' background in agile software development. Among them, an interesting observation is that the agile RE practices that are supposed to support developers (i.e., P09. Provide and maintain informative workspace, P06. Organize daily team meetings, P28. Negotiate iteration scope with customer) were perceived less important by them than by people responsible for requirements or Scrum Masters/project managers. The collected data also suggest that there exist some factors describing the context of a software development project, such as country, size, domain, etc., that might influence the way people perceive the importance of the agile RE practices. However, taking into account the sample considered in our study, we were not able to firmly characterize/explain the nature of these relationships.

Finally, we investigated the correlation between perceived importance and the popularity/adoption level of some of the agile RE practices to learn that the frequency of using a practice could be a good predictor of its perceived importance. The exception from this rule are practices that are difficult to adopt but very important (e.g., available/on-site customer). However, taking into account the lesser popularity of the surveys studies on the importance of the agile RE practices, it seems justified for practitioners to rely on the more available studies on the frequency of usage of the practices while making their decision on practice adoption and improvement.

Implications for researchers

Although the body of knowledge about agile SD has increased over the years, there is still much to learn about the applicability and usefulness of certain agile practices in specific project contexts. We have analyzed sixteen secondary studies (and the primary studies they referred to) trying to find information that would allow us to get a deeper understanding of the relationships between the perceived importance of the agile RE practices and respondents' demographic profiles that we observed in our study. Unfortunately, we

could not find any strong empirical evidence allowing us to characterize or explain any of them. Therefore, more research in this area is needed to understand how the importance of agile RE practices depends on project contexts.

3.2 Importance of NFRs in agile software projects

3.2.1 Introduction

Quite frequently project failures can be traced to inappropriate management of NFRs (see e.g., [22], [25], [145], [173]).

In spite of this, as reported by Cao and Ramesh [30], in many projects based on agile principles NFRs are neglected. It is also somehow confirmed by what was presented in Section 3.1: none of the identified agile RE practices explicitly mentions NFRs. Therefore, the following question arises:

- **RQ1.2.** *Is it important to specify non-functional requirements in agile software projects?*

To answer this question we asked software development practitioners with experience in working with agile approaches about their perceived importance of specifying NFRs (see Section 3.2.2 for the design of the survey). Then, we analyzed the answers and compared them with the perceived importance of the 31 agile RE practices identified in Section 3.1 (see Section 3.2.3, and for validity threats refer to Section 3.2.4). We also showed how other researchers and practitioners assess the importance of practices in agile software development and how they investigate the importance of NFRs from different perspectives (Section 3.2.5). The conclusions from our work were formulated in Section 3.2.6.

3.2.2 Survey Design

The goal of the study was to identify the perceived importance of NFRs in the context of agile software development projects. To achieve the goal we designed a survey using the guidelines by Kitchenham and Pfleeger [127].

Survey instrument

To collect the opinions of a wide variety of respondents, we asked the participants of the Agile RE practices survey [176] to take part also in our survey. In an online questionnaire we asked about the perceived importance of the practice: "*How important it is to specify NFRs for an agile project?*". Respondents were supposed to recall the project they had participated in and judge whether the practice was "critical", "important", or "additional" (helpful but also supplementary–could have been rejected without doing any harm to the projects). The respondents could choose the answer "other" and provide their own descriptive answer or could skip the question by selecting the "don't have an opinion" answer.

Since it was required to participate in the Agile RE practices survey [176] first, we took advantage of this. In the analysis, we used the answers (1) on the perceived importance on the 31 Agile RE practices; and (2) the demographic information provided by the respondents who also answered the question concerning NFRs.

The short online one-question survey underwent proofreading by a professional linguist and a pilot study with three members of our laboratory. No problems were identified. The questionnaire was distributed using Google Docs Forms with the Agile RE survey.

Population

Our target population can be defined as agile software development project's participants. We did not limit our focus to any specific types of applications or domains. However, we assumed that an individual belonging to the population needs to have at least one year of experience in agile software development. Our aim was to get a general view of the perceived importance of NFRs, which required to cover a wide group of people. Unfortunately, there is no obvious place in which we could identify and access the representatives of the defined population. Thus, we relied on non-systematic sampling methods (convenience sampling). Then, the accuracy concerning the error between the population and sample cannot be determined [196]. However, we determined the characteristics of the respondents afterward to describe the sample.

The survey inherited from the Agile RE survey the two sampling approaches used in parallel to balance the strong and weak points of each method. First, we used self-recruiting ([196]). We placed posts inviting to the Agile RE survey the members of social network groups related to agile software development and RE on LinkedIn, Facebook, Yahoo. Secondly, we sent direct invitations to people we knew to have experience in agile projects, the members of Scrum Alliance using the communication tool available on their website, and to people who published their curricula vitae on the Internet (mainly on LinkedIn) indicating that they have experience in agile software development.

Data validation and analysis

We decided to validate the responses using the following criteria: (1) a participant had to answer more than 75% of questions from the Agile RE survey providing responses other than "I don't have an opinion", and (2) had at least one year of experience in agile software development. To analyze the responses we used the following methods and techniques:

- *frequency analysis* – to get a general overview of the opinions of the respondents;
- *relationship identification* – to investigate the relationships between the perceived importance of the practices and the responses given on the demographic questions in the Agile RE survey; we used a non-parametric Kendall's τ correlation coefficient as the perceived importance of the practice was measured on an ordinal scale. Moreover, we used a test for association between paired samples for Kendall's τ [104] from the R statistical package with the significance level α set to 0.05.
- *ranking comparison* – we analyzed the perceived importance of the NFRs practice against the ranking proposed by the authors of the Agile RE survey. The aim was to see the relative importance of the NFRs practice that is over which Agile RE practices it is preferred.
- *open-text* – we used guidelines of Charmaz [34] to analyze the responses left by participants if they expressed their opinion on the perceived importance of the practice textually rather than choosing one of the given answers.

3.2.3 Results and discussion

Respondents

Our survey followed the Agile RE survey [176] the invitations to which were sent in two runs: 09/10-20/12/2015 (the responses were collected until the end of June 2016), and 21/12/2017-11/01/2018 (the responses were collected until 28 February 2018). There were 147 responses to the Agile RE survey (2 of them did not meet the validation criteria), and 118 of respondents answered our survey concerning NFRs (all of them met the validation criteria). We received the following numbers of responses from different sources:

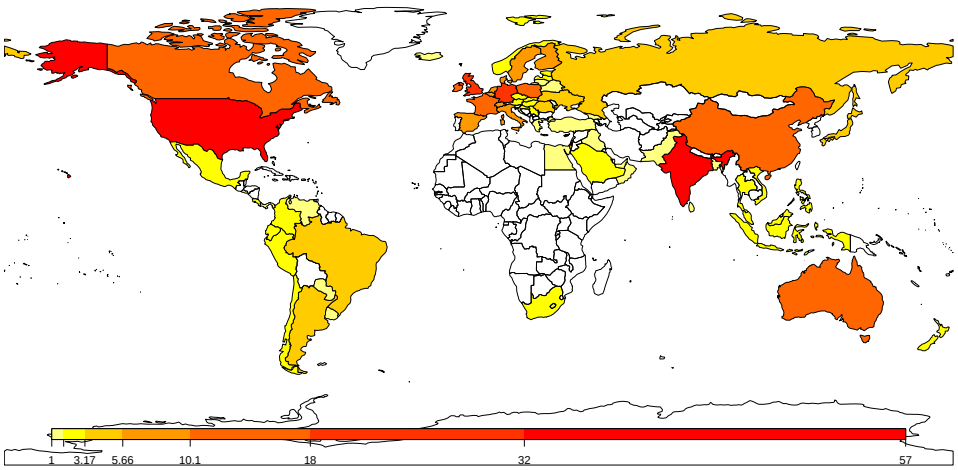


Figure 3.6: Countries in which the agile projects that respondents participated in were conducted (multiple choices allowed).

- Social media interests groups related to agile software development approaches or Requirements Engineering = 76
- Direct invitations sent via Scrum Alliance community page = 12
- Direct invitation to known persons = 1
- Direct incitations based on CVs = 29

We obtained a wide coverage of countries around the globe in which the projects of our respondents were conducted (see Figure 3.6). In total the respondents had experience in 71 different countries with the top five of the United States of America (US), India (IN), United Kingdom of Great Britain and Northern Ireland (GB), Germany (DE), and Poland (PL). Compared to the Agile RE survey (see Subsection 3.1.5), the total number of countries is the same, but the majority of those respondents who did not answer the survey concerning NFRs had experience in software projects run in Poland (PL) (26 people) and Ireland (IE) (5 people); the difference for other countries is of maximum 2 people.

The coverage of domains and types of software applications developed by participants was also high. As shown in Table 3.5, the four most dominating domains were services, financial, telecommunication, and banking. The most frequently developed types of applications were web and mobile applications, financial applications, and management information systems.

The most popular agile methods were Scrum and Kanban. Slightly less popular were eXtreme Programming (XP) and Scrumban. Only a few respondents mentioned other methods such as DSDM, SAFe, or Crystal Clear.

The working experience also spread nearly through all of the activities we asked for in the survey. Although the dominating responsibilities were Scrum Master (or equivalent), project manager, and requirements analyst, we also had many participants with experience in quality assurance (QA), testing, programming, and software designing. The participants had on average 18 years of experience in SD and more than 8 years in agile projects. The average number of agile projects was 24; however, the meaning of the term project could be understood differently by the participants.

The participants had experience in agile projects of different sizes, starting from small projects having up to five members of a team and total development effort smaller than one man-month, up to very big projects with more than 50 people involved and total development effort greater than 80 man-months.

Domains (N=116, multiple choices allowed)			
Services	53	Insurance	35
Financial	52	Entertainment	30
Banking	43	Government	27
Telecommunication	42	Trading	20
Electronics&Computers	40	Manufacturing	17
Medical&Health Care	39	Other	32

(a) Domains in which the respondents' agile projects were conducted

Types of applications (N=117, multiple choices allowed)			
Web or e-business	82	Document management	43
Mobile apps	61	Billing	33
Financial	56	Sales&Marketing	32
Mgmt Inform. Systems	48	Logistics	28
Electronic data interchange	47	Personnel	23
Transaction or production systems	42	Trading	20
		Others	21

(b) Types of applications developed in the respondents' agile projects

Methodologies (N=90, multiple choices allowed)			
Scrum	86	Scumban	23
Kanban	52	DSDM	6
XP	34	Others	21

(c) Methodologies used in the respondents' agile projects

Responsibilities (N=118, multiple choices allowed)			
Scrum Master	91	Soft. Designer	58
Project manager	88	Programmer	53
Req. Analyst	80	Coach	13
TestingQA	60	Others	16

(d) Responsibilities of the respondents in agile projects

Size of teams (N=118, multiple choices allowed)			
<=5	57	20-30	19
5-10	92	30-50	20
10-15	53	>50	20
15-20	32		

(e) Sizes of teams in the respondents' agile projects

Total development effort [man-month] (N=118, multiple choices allowed)			
<=1	21	8-20	66
1-2	33	20-40	49
2-4	54	40-80	42
4-8	59	>80	47

(f) Total development effort in the respondents' agile projects

Table 3.5: The results with respect ot the agile projects in which the respondents participated.

Perceived importance of NFRs practice

The summary of the responses on the perceived importance of the NFRs practice is presented in Figure 3.7.

It follows from the results that 47.46% of respondents (56 people) considers specifying NFRs as important. Interestingly, 30.51% respondents (36) claim that the practice is

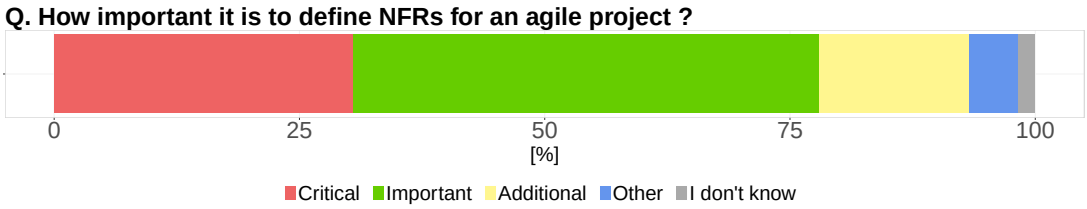


Figure 3.7: Perceived importance of the practice concerning NFRs.

critical. Thus, the following observation can be formulated:

Observation 3.1: *Over 77% of agile practitioners who took part in our study find specifying NFRs as at least important practice for a software development project.*

Importance of NFRs versus other practices

The ranking of relative importance of the 31 Agile RE and the practice of specifying NFRs is presented in Figure 3.8.

Based on the constructed graph of relative importance, the practices could be divided into *tiers*. The boundary of a tier is determined by practice P_A which has the lowest score and still is dominated only by practices from the higher tiers (tiers with lower index numbers).

The most critical practices (Tier 1) could be characterized as those that seem fundamental for propelling a fast pacing, iterative, agile project without a fully predefined scope and emergent requirements. The practice of specifying NFRs is located at Tier 4. Similarly, its neighbor and a very similar practice P05: Define project and product constraints. First, it shows that consistency of our respondents as NFRs can be viewed as constraints. Secondly, it follows from the results, that among essential practices for agile software development, although there are some practices perceived as more important, specifying NFRs is still quite high – the fourth tier.

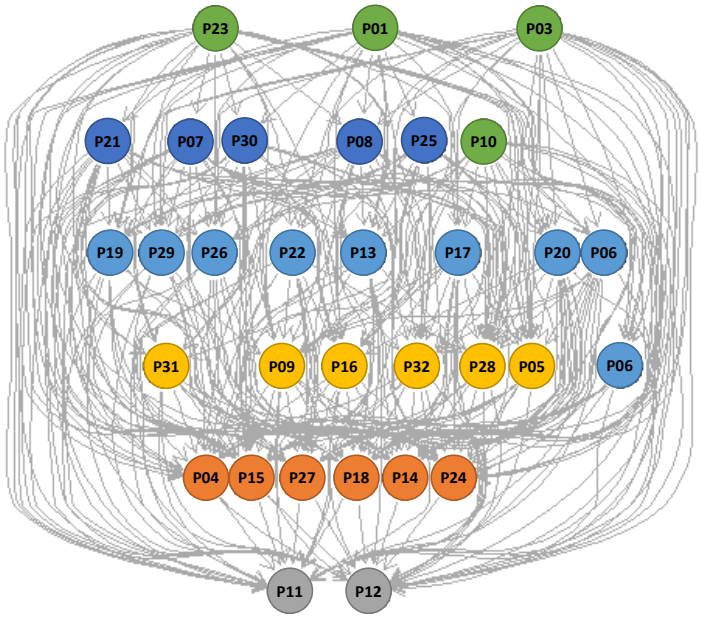
Compared to the ranking constructed based on the results of the initial survey (see Subsection 3.1.5) few practices were classified to lower tiers (e.g., P25, P07), one P31 (Define fixed iteration length) dropped down by two tiers, and practices P11 and P12 were classified at the same tier 6.

Such ranking might be helpful when, for example, one wants to know which practices are the most important and introduce them first, or what might be the order of practices to improve.

Perceived importance of NFRs and demographic data

Following the method described in Section 3.2.2 we investigated whether the perceived importance of specifying NFRs depends on the experience of respondents. The overall experience of respondents seems to have a significant role in how they perceive the importance of NFRs. We observed a correlation between specifying NFRs and two factors (1) the number of years of experience in software development projects ($\tau = 0.21$, p-value = 0.005) and (2) the number years of experience in agile software development projects ($\tau = 0.23$, p-value = 0.004). Thus, the following observation can be formulated:

Observation 3.2.: *The more experience our respondents have both generally in software projects and specifically in agile software projects, the more important for them the practice of specifying NFR is.*



Tier	Practices
Tier 1	P01: Available/On-site Customer P03: Establish project shared vision P23: Make requirements testable P10: Provide easy access to requirements
Tier 2	P21: Requirements valuable to purchasers or users P07: Organize review meetings P30: Keep release length short P08: Organize retrospective meetings P25: Let customer prioritize requirements P22: Make requirements estimable
Tier 3	P19: Write short, negotiable requirements P29: Freeze release scope after it is agreed P26: Notation easy to understand by all stakeholders P13: Let customer define acceptance tests P17: Cover requirements with acceptance tests P20: Make complex requirements divisible P06: Organize everyday team meetings P02: Involve different stakeholders
Tier 4	P31: Define fixed release length P09: Provide informative workspace P16: Perform regression acceptance testing P32: Define quality requirements/non-functional requirements P28: Negotiate rerelease scope with customer P05: Define project/product constrains
Tier 5	P04: Create prototypes P15: Prepare acceptance tests before coding P27: Assess implementation risks for requrements P18: Make requirements independent P14: Prepare and maintain automatic acceptance tests P24: Follow the user role modeling approach
Tier 6	P11: Maintain “bad smells” and best practices rel. to requirements P12: Perform the Elevator Test

Figure 3.8: The ranking of relative importance of the Agile RE practices enhanced with the practice of specifying NFRs. The graph shows the preference degree $\pi(P_i, P_j)$ between the practices. The existence of an arc between P_i and P_j means that $\pi(P_i, P_j) = 1$.

Analysis of open-text questions

The analysis of open-text answers provided some interesting insights concerning NFRs. First, about the time when NFRs are important:

- after functional requirements are written – "*Important, **after functional requirements** have been written.*"
- early in the project – "*We need to know the NFRs **eventually**; in some contexts we need to know them **very early in the project***"

Similar insights had the participants of the survey concerning elicitation of NFRs [132], but they also added that in agile projects NFRs shall be elicited and specified in the iterative approach throughout the project.

Secondly, two respondents pointed out that there are different forms of defining NFRs:

- as definition of done – "*should be agreed upon - in Scrum that is **the definition of 'done'***"
- as acceptance criteria – "*They should always be part of **user story acceptance criteria.***"

The same comments left the participants of the survey concerning elicitation of NFRs [132], but they added also that NFRs are present also in the definition of ready. The participants of the other survey also pointed out that those requirements are difficult to specify and to write, especially when discussing with user/client but two things could help:

- talking about consequences – that is either "*how much it would cost to implement a requirement*".
- using abstract but easy to grasp terms by a user and/or alternatives – for example asking "*would it be ok if the report refreshes once a day or shall it be live*".

3.2.4 Validity Threats

The following analysis of the threats to the validity of our study is based on the guidelines by Wohlin et al. [241].

Conclusion validity. There are two threats about the scale used to assess the perceived importance of the practice concerning NFRs and the same scale used in the Agile RE survey. First, if a respondent chose "don't have an opinion" or "other", then we were not able to use the information to evaluate the relative importance of the practice. Fortunately, there were only eight such respondents, which is negligibly small amount comparing to the number of all responses.

Secondly, the scale is subjective and might be interpreted differently by respondents depending on their background, culture, etc. We allowed providing open-text answers or resigning from giving any response to avoid biasing the results by forcing the respondents to answer about the practice they had not had any experience with or does not allow them to respond using the proposed scale. Thirdly, the scale is built around only positive (or neutral) influence of the practice on a project. We assumed that if a respondent had a negative experience with it, they would express that using the "other" option. One of the respondents shared such opinion stating that specifying NFRs is pointless. Finally, we used a qualitative coding technique to analyze open-text responses. This approach is subjective by its nature, and we could have introduced some bias to the conclusions made based on the analysis.

Construct validity. Neither the practice of specifying the NFRs nor the 31 Agile RE practices used as the basis for the relative comparison, do not have formal and widely accepted definitions. Thus, they could have been understood by the respondents in a different way. Although there were definitions provided for each practice, still the understanding might have been different.

In the survey, we used the term NFRs (NFRs) which also has no one agreed definition. By some practitioners and researchers, the term is used as the synonym to quality requirements (QRs), by some QRs are a subtype of NFRs [83]. Thus, the term could have been understood by the respondents in a different way.

To assure trust, transparency and repeatability, when designing and describing our study, we discussed and then reported as many details of the survey as possible. We followed the guidelines by Kitchenham and Pfleeger [127]. We also conducted a self-assessment using the Stavru's criteria [225] to verify if we reported all essential elements. In the assessment, we obtained the score equal to 0.89 (range 0-1). The result is higher than the highest score in the agile surveys examined by Stavru [225], which seems to be the sufficient level of detail.

Internal validity. Our control over collecting responses was limited. We are not able to reliably determine the number of people who received or read our invitation. Secondly, there is a question of trustworthiness of the respondents. They might have intentionally tried to influence the results of the surveys. However, the results of the sensitivity analysis run for the first survey [176] showed that it would not have a visible impact on the outcomes. Moreover, we could not find any reasonable rationale to explain the bad intentions of the respondents.

The surveys were sent in two runs to receive more responses. To ensure that we ask each participant only once we added a question directly asking about the participation in the first run (the positive answer eliminated the participant's response from the further analysis).

Another threat to internal validity regards the technical skills required to respond to the survey. We assumed that the members of the target population would not have problems in responding to an online survey and are fluent enough in English to understand the questions. We also conducted a pilot study to ensure correctness and ease of understating of the questionnaire correctness. A professional linguist also proofread the survey. We believe that the assumption was reasonable because English is a lingua franca of computer science. Besides, it turned out that ca. 86% of respondents of the Agile RE survey participated in projects conducted in the countries having English as an official language.

When it comes to the commitment of the respondents, we did not observe any suspicious responses that would suggest the lack of commitment or intent to sabotage the study. Besides, we believe that the topic was attractive to the participants because 72% of the Agile RE survey respondents voluntary shared their e-mail addresses to receive the summary of results.

The last two threats to internal validity regard the assessment of the importance of the practice. First, we do not know what particular events in the respondents' projects determined how they perceived the importance (e.g., was it a single project that failed because of lack of certain practices, or was a practice essential for all the projects?). Secondly, many of our respondents had long-term experience in agile projects, and thus their opinions could change in time (e.g., some facts from the earlier projects might have been blurred when they were filling in the questionnaire).

External validity. We intended to conduct our study among people who have experience in agile software development. We collected responses from participants taking part in projects undertaken in various countries and domains, also with experience in various project roles. Still, there are some threats related to the representativeness of the sample that we should consider. First, the sample of 118 respondents is still small comparing to the expected size of the population (however, this cannot be determined accurately). The sample may be too small and the obtained results could be different for a different sample. On the other hand, the results of the sensitivity analysis of the ranking conducted when analyzing the Agile RE survey show that the number of respondents seems sufficient to create a stable ranking of the practices.

The second threat regards using the non-systematic sampling methods. By sending invitations to social groups regarding agile software development approaches and RE, we

could reach people that are particularly interested in these topics (e.g., "agile enthusiasts"). Therefore, their view of agile practices might be somehow biased in comparison to other individuals in the population. We tried to mitigate this problem by sending direct invitations to people participating in agile projects (balancing the sample with respect to the sources).

Finally, the responses of the participants reflect their subjective view of the subject. Without triangulation, we cannot firmly state whether the real importance is equivalent to the perceived one. However, we do not concern this threat as significant since the results of the Agile RE study seem, in general, convergent with the results of the studies regarding critical success factors in agile projects (e.g., [37]).

3.2.5 Related Work

Survey research studies are common in the context of agile software development. There have been carried out many studies investigating adoption levels and popularity of agile methods and practices. Some of these studies characterize usage of agile methods in general (e.g., [138, 186]) other focus on particular countries (e.g., [28]) or concern applicability of agile to particular domains, types of applications, or activities in projects (e.g., [108, 211]). There are also few studies focusing on the popularity of RE techniques and tools in agile projects, e.g., [123, 235]. In general, the results of descriptive surveys seems attractive to practitioners since there are even surveys conducted annually by software vendors such as Version One [231]. This particular survey summarizes the current trends in the agile community when it comes to methods, practices, and tools. It also aims at identifying the benefits and challenges of adopting agile methods. However, despite its popularity, the survey received some criticism for lacking scientific rigor that decreases its trustworthiness [225].

Williams [239] reported the results of a survey aiming at investigating which of the agile practices are essential for a team to be considered agile. Some insights into the importance of agile practices are given by studies comparing agile and non-agile approaches to software development (e.g., Elshandidy and Mazen [71]), by systematic literature reviews (SLRs) or mapping studies discussing the benefits and challenges of agile RE (e.g., [60, 112]), or by a clustering analysis of agile practices by Fontana et al. [76].

Another relevant survey research study is the one by Cao and Ramesh [30]. They surveyed 16 organizations about the degrees to which they follow agile RE practices and what are benefits and challenges of using them. They also found that especially in agile projects there are difficulties with eliciting NFRs.

On the other hand, there are also surveys trying to investigate the problems in software projects, e.g., [122] from which we can learn on which practices to focus.

In the area of NFRs, the researchers try to understand better the actions taken by project stakeholders connected with NFRs, e.g., [9, 70, 182]. There are also research initiatives that aim to understand the relationship between agile software development and NFRs' awareness, e.g., [92]. Additionally, few works investigate project failures to provide evidence of what might happen when NFRs are treated as not important, e.g., [74]. Groen et al. [87] also showed that opinions of software product users might show which particular NFRs are essential.

3.2.6 Summary

The goal of our study was to investigate the importance of non-functional requirements (NFRs) in agile software projects. To achieve the goal the respondents of the Agile RE survey [176] were asked to provide their opinion concerning NFRs. Based on the answers

of 118 participants having experience in agile software development projects in a wide variety of countries across the globe we formulated the following observations.

- (*Observation 3.1*) For over 77% of the respondents the practice of specifying NFRs is at least important (for 30% of the respondents it is even critical) for an agile project.
- (*Observation 3.2*) The longer the experience either in software projects generally or in agile software projects our respondents have, the more important the practice of specifying NFRs they find.

Some agile practices are considered more important than specifying NFRs, e.g. establishing the vision of the project, having an available customer, ensuring that requirements are testable, etc. (see Figure 3.4 for all of them).

Some small fraction of respondents claimed that specifying NFRs is an additional practice. The body of knowledge about agile software development indicates that specific practices might be more important in some contexts. Unfortunately, we were not able to identify any strong relationship between the perceived importance of the NFRs practice and data on projects they participated in. Therefore, more studies are needed to understand how the perceived importance of NFRs depends on project context.

The participants also indicated that NFRs might be in some cases necessary early in the project life cycle. Moreover, they mentioned that NFRs are used in different forms such as definition of done or acceptance tests. Thus, more research would be needed to investigate 'when', 'where', 'in which form', and 'how' NFRs are used in software development projects.

Chapter 4

Usefulness of catalog of NFR templates

Preface
This chapter is based on the following already published papers:

1. S. Kopczyńska, J. Nawrocki, "*Using non-functional requirements templates for elicitation: A case study*", 4th IEEE International Workshop on Requirements Patterns (RePa), 2014, 47-54 (Section 4.3);
2. S. Kopczyńska, J. Nawrocki, M. Ochodek, "*An Empirical Study on Catalog of Non-functional Requirement Templates: Usefulness and Maintenance Issues*", Information and Software Technology, 103, 75-91, 2018 (Section 4.4)

The second paper tackles two aspects of catalog of NFRs templates: usefulness and dynamics of catalog characteristics associated with catalog evolution. In the PhD thesis the two aspects are analyzed separately. In this chapter (Chapter 4) the first part of the paper is presented which concerns usefulness of catalog of NFR templates. The second part, dealing with the dynamics of catalog characteristics, is discussed in the next chapter (Chapter 5).

Context. Non-functional requirements (NFRs) are not easy to elicit nor to formulate. Therefore, some experts advocate using templates, i.e., statement patterns with parameters and optional parts. However, practitioners are afraid of using templates as they do not know what would be the return on investment of using this approach. Unfortunately, there is still scarcity of evidence showing the benefits of this approach.
Aim. The aim of this chapter is to evaluate the usefulness of catalog of NFR templates.
Method. First, we carried out an exploratory case study with 7 software projects, each lasting about a year, that delivered tailor-made web applications (Section 4.3). In each project, NFRs were elicited using a catalog of templates and after closing the projects, we studied the modifications the NFRs underwent. The focus was on *firmness* of NFRs, *effectiveness* of elicitation process, *helpfulness* of the used catalog, and *personal impression* of elicitors.

Second, we conducted an experiment with 107 participants (Section 4.4). The participants, individually or in teams, elicited NFRs based on a business case concerning an e-commerce system. They used either the template-supported approach or the *ad hoc* one. Then, we evaluated the quality of the NFRs taking into account their *unambiguity*, *individual completeness*, *verifiability*, *set completeness*, and *yield* of the elicitation. We were also interested in the impression about usefulness of NFR templates.
Results. First, our exploratory case study showed that the firmness of NFRs (i.e., the percentage of initially elicited NFRs that remained valid till the end of a project) was at the level of 80%. The effectiveness of template-supported elicitation of NFRs (i.e., the percentage of final NFRs that were elicited at the beginning of each project) was also at the level of 80%. Participants of template-supported elicitation of NFRs regarded NFR templates as useful (about 85% of them positively rated the usefulness of templates). The effort to elicit

NFRs with the use of templates was about 3 hours per person (two ca. 1.5-hour elicitation workshops).

The participants of the second study (the experiment) who used NFR templates provided NFRs that were more complete, less ambiguous, more detailed, and better from the point of view of verifiability than their counterparts using *ad hoc* approach. However, the catalog of templates did not speed up the elicitation process. Over 85% of the respondents who used NFR templates perceived them as useful.

Conclusions. Catalog of NFR templates seems a useful tool. It does not hinder the firmness of NFRs and is perceived as useful by elicitors. Moreover, it increases the quality of NFRs but does not speed up the elicitation process.

My contribution. Inspired by Jerzy Nawrocki, I proposed and documented a systematic approach to elicitation of NFRs, called SENoR, and the measures to evaluate the usefulness of catalog of NFR templates. I designed the case study described in Section 4.1 (supported by Jerzy Nawrocki and Michał Maćkowiak), conducted the study, and analyzed the obtained data. Together with Michał Maćkowiak, we developed a software tool that supported elicitation of NFRs.

I designed the experiment and proposed the measures used to evaluate the usefulness of catalog of NFR templates (supported by Jerzy Nawrocki and Mirosław Ochodek). I was also responsible for conducting the study and analyzing the obtained data. I developed software tools that supported the analysis.

4.1 Introduction

Software requirements are usually categorized into functional and non-functional ones. The former describe so-called user-valued transactions (i.e., functionality that supports the users), and the latter state conditions under which the provided functionality is really useful (e.g., maximum response time).

Non-functional requirements (NFRs) are too often neglected, especially those that are difficult to write or ostensibly obvious (e.g., [30]). That is an important risk factor, as in many cases a project failure can be traced, amongst others, to an inappropriate management of them (see e.g., [22, 25, 145, 173]).

One of the approaches aiming at improving the practice of specifying NFRs is to use *templates*. Templates, also called *boilerplates* or *blueprints*, are expressed in natural language as a text with some gaps (parameters) to be filled in and optional parts to select while formulating a requirement. They preserve correct requirements syntax (e.g., EARS [154]) as well as they can encompass some semantics of NFRs ([107, 129, 204]). During NFRs specification one can select templates from a catalog and provide the values of the parameters they define, which is called template-supported or template-based elicitation.

According to some authors, using templates improves consistency and testability of requirements, reduces ambiguity [20, 154, 240], makes elicitation and specification easier [240], and reduces the effort of specification [191]. Therefore, they have been incorporated into some tools (e.g., [202]). But those opinions, although formulated by experts, are not convincing to everyone. Palomares et al. [182] have found that practitioners are afraid of using templates. They perceive them as a complex method and do not know what would be the return on investment. Thus, more evidence is needed about benefits and costs associated with NFR templates.

In this chapter, two empirical studies are presented concerning the usefulness of using catalog of NFR templates:

- **a case study** (see Section 4.3) in which we aimed at characterizing template-supported elicitation of NFRs focusing on: stability of elicited NFRs, helpfulness of the catalog during elicitation process, and personal impression of elicitors in the context of seven projects that developed tailor-made software applications;
- **an experiment** (see Section 4.4) in which we compared template-supported elicitation of NFRs with *ad hoc* one focusing on: quality of elicited NFRs, and speed of elicitation in the context of inexperienced elicitors working both individually and in teams.

This chapter is structured as follows. In Section 4.2 we discussed the related work. In Section 4.3 we reported the design, and the results of our case study. Then, in Section 4.4 we described our experiment, i.e., we introduced some terminology, described the study design, presented results. The findings of both studies are summarized in Section 4.5.

4.2 Related work

Patterns, templates, and boilerplates. There are few methods and approaches that utilize the knowledge from previous projects to assure the quality of requirements proposed by the requirements engineering community. One of them is using patterns. Patterns structurally describe the problem they solve, the context they are to be used in, and the solution itself. They were proposed for both functional requirements (e.g., [3, 175]) and non-functional requirements. For NFRs there exist few catalogs of patterns, e.g., for natural language requirements: the Withall's 37 NFRs patterns [240], PABRE patterns [201], or for the goal-based requirements notation: the NFR Pattern approach [227] an extension to NFR Framework [39].

Another approach is to use templates or boilerplates ([106]) of requirements, like our NFR templates. They are sentences or statements with some parameters. Frequently, they are parts of patterns (they constitute the solutions that the patterns propose), but they also exist solely as catalogs. Templates might capture knowledge on different levels of abstraction. For example, those by Denger et al. [59] are sentence parts such as events, conditions, exceptions, etc. which could be combined with sentence patterns to describe complete requirement statements. Similarly the boilerplates are constructed: EARS [154] and Rupp's [192]. The mentioned approaches concern also functional requirements and are reported to be effective in practice. Then, there were proposed templates that capture not only the sentence structure but also some knowledge of particular area such as security [75, 203] or performance [69].

Several researchers conducted studies to understand the experience of using templates and patterns in the industry. Palomares et al. [182] found that NFRs are those requirements that are more likely to be reused in companies, especially those expressed using natural language. Although 11% of the respondents of their survey used patterns, participants were afraid of too high investment, high complexity of methods, and had doubts about the return on investment. Naish and Zhao [168] found that the choice of the format for requirements reuse has a significant impact on the effectiveness and efficiency of the patterns and templates management, i.e., creation, maintenance, and also usage. To support selection decisions whether to use or not to use templates of non-functional requirements, with our studies, we broaden the existing knowledge on the NFR templates that cover more than one quality aspect and capture both syntactic structure and knowledge on NFRs.

Difficulties in selecting proper RE technique due to scarcity of empirical knowledge. Although it is claimed that frequently requirements elicitation happens in an *ad hoc* man-

ner [27], in the literature, we may find numerous works on the methods that aim at supporting this process—from general guidelines, e.g., EasyWinWin [89], Joint Application Development workshops [242], Gottensdiener's guidelines on requirements workshops [85] to the ones more specific for NFRs, e.g., Herrmann et al. [97] or Doerr et al. [62]. In the thesis, we focus on NFRs expressed in natural language, but there are also some other approaches like expressing NFRs as goals, which are also studied by researchers, e.g., NFR Framework [167], KAOS [230].

Unfortunately, we still need more empirical research and empirical evidence on using the methods and techniques supporting elicitation of requirements (and in particular NFRs) [65, 228]. Among the available studies, some researchers compare and discuss existing approaches like Herrmann [97]. Others, as we do in this chapter, deliver empirical evidence regarding requirement elicitation. For example, based on the experience Strohmaier et al. [226] reported that the existing methods are costly taking into account effort and complexity, the experience of Cleland-Huang et al. [40] (they identified the existing methods as too heavy-weight especially for the agile context) driven them to propose their own light-weight approach.

Worth attention and similar to our experiment, is also the work by Riaz et al. [203], they conducted a series of experiments that focused on an automated suggestion of security requirements templates implied from the existing functional requirements. They used students as the study subjects, as we did. They evaluated the quality of resulted NFRs by asking experts to provide their opinion using a 5-point Likert-scale. They used a checklist consisting of 5 questions as the evaluation supporting material. Similarly to us, they considered the level of detail and the interpretation aspect (logical inconsistency, specificity). Moreover, they evaluated coverage of the requirements comparing the results with their oracle and efficiency of elicitation. Their results for security and our results for other non-functional characteristics (categories) are aligned. Both studies show statistically significant results regarding the increase of completeness, and in some areas, better quality of NFRs elicited with the use of the template-supported approach. Doerr et al. [62] investigated the IESE NFR method that utilizes templates in a structured elicitation approach. They looked at completeness from another perspective. They asked project stakeholders postmortem about new requirements. Their conclusion on the improvement of completeness is aligned with our conjecture.

Several researchers investigated techniques of assuring the quality of NFRs or identifying errors made while specifying NFRs. Templates are one of such techniques. Denger et al. [59] proposed to generate templates from a meta-model for embedded systems. Their finding was that such approach may allow eliminating imprecisions (e.g., ambiguity) but may require some help with usage. Eckart et al. [69] showed that well-grounded templates could improve the completeness of performance requirements meant as the completeness of all necessary information in a single requirement. This requirement quality characteristic was also investigated in our studies, not only for performance requirements but for nearly all categories of the ISO 25010 standard. The results of both studies are aligned.

4.3 Case study

4.3.1 Introduction

We decided to carry out a case study to better understand the role of catalog of NFR templates for elicitation of NFRs. This type of approach is useful to deeply explore a phenomenon in real-life settings [209]. The goal of the case study can be formulated as follows:

Goal. Characterize template-supported elicitation of NFRs focusing on (1) stability of elicited NFRs, (2) helpfulness of the catalog during the elicitation process, and (3) personal impression of the elicitors. The considered context is: projects aiming to deliver tailor-made software applications.

Two major factors influence Stability of NFRs: some requirements can be removed (the more of them, the lower the *firmness*) and some others can be added (the more of them, the lower the *effectiveness* of elicitation). Taking this into account we refined our goal into the following set of research questions:

RQ 4.1. (*Firmness*): How many of the NFRs initially elicited with the help of catalog of NFR templates remained valid till the end of a given project?
RQ 4.2. (*Effectiveness*): How many of the final NFRs have been elicited with the help of catalog of NFR templates at the beginning a given project?
RQ 4.3. (*Helpfulness*): What is the coverage of elicited NFRs by NFR templates contained in the catalog?
RQ 4.4. (*Impression*): What is the impression of those who have used catalog of NFR templates?

4.3.2 Study design

Indicators

To define indicators (some people call them measures) for the above quality characteristics the following sets will be used:

- P^{Ini} is a set of all NFRs elicited at the beginning of a given project;
- P^{Fin} is a set of all NFRs valid at the end of a given project;
- $Perfect^{Ini}$ is a set of all NFRs directly derived from the NFR templates at the beginning of a given project;
- $Perfect^{Fin}$ is a set of all NFRs directly derived from the NFR templates that are valid at the end of a given project and modified versions thereof (here modifications of NFRs are limited to syntactic improvements);
- $Modified^{Ini}$ is a set of NFRs indirectly derived from the NFR templates at the beginning of a given project;
- $Modified^{Fin}$ is a set of NFRs indirectly derived from the NFR templates that are valid at the end of a given project and modified versions thereof (NFRs can be only syntactically improved);

Those sets and relations between them are depicted in Figure 4.1.

We proposed the following indicators of *Firmness*, F, and of *Effectiveness*, E ($|S|$ denotes the number of items belonging to set S):

$$F = \frac{|Perfect^{Fin} \cup Modified^{Fin}|}{|Perfect^{Ini} \cup Modified^{Ini}|} \quad (4.1) \qquad E = \frac{|Perfect^{Fin} \cup Modified^{Fin}|}{|P^{Fin}|} \quad (4.2)$$

To have a better insight into the elicitation processes supported by catalog of templates, it seems worth to consider *strong* versions of these indicators, i.e., limiting derivations only to direct ones (this is derivation in the strong sense). Then, one can define strong versions of the *firmness* and *effectiveness* indicators:

$$sF = \frac{|Perfect^{Fin}|}{|Perfect^{Ini}|} \quad (4.3) \qquad sE = \frac{|Perfect^{Fin}|}{|P^{Fin}|} \quad (4.4)$$

To measure the helpfulness of catalog we will use *Catalog Value* and its strong version as defined below:

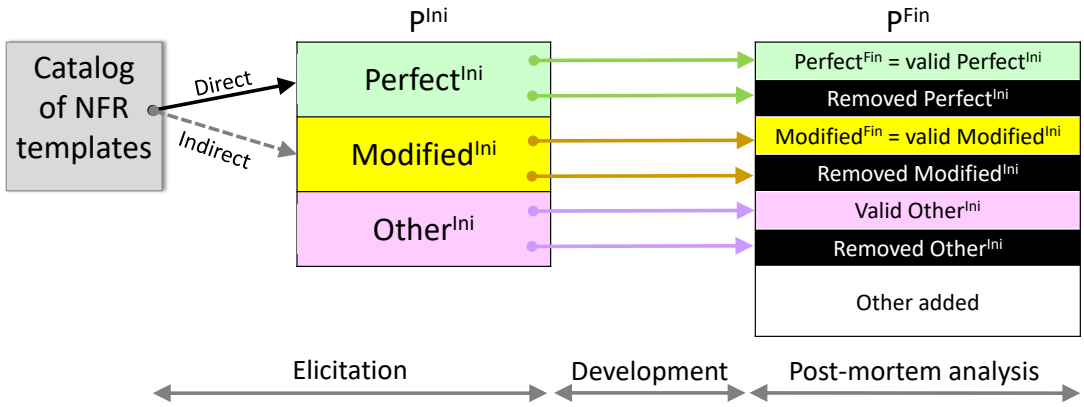


Figure 4.1: Partition of NFRs elicited at the beginning of a given project (P^{Ini}) and NFRs valid till the end of the project (P^{Fin}) into subsets that are used to define performance indicators.

$$Value = \frac{|Perfect^{Ini} \cup Modified^{Ini}|}{|P^{Ini}|} \quad (4.5) \qquad sValue = \frac{|Perfect^{Ini}|}{|P^{Ini}|} \quad (4.6)$$

As regards *Impression* we have decided to decompose the main question into the following sub-questions:

Q1. Have NFR templates been useful?
Q2. Have NFR templates been of good quality?
Q3. Are the elicited NFRs of good-enough quality for architecture design?

We asked the above questions to the participants of the elicitation workshops, and each of them had to select one of the following answers (they form well-known 5-level Likert scale): *Strongly agree*, *Rather agree*, *Hard to say*, *Rather disagree*, *Strongly disagree*. For each question, we computed the percentage of votes for each of those options.

Case and participants

We decided to conduct our study in an organization that runs several projects and develops tailor-made software. Such setting we found in Software Department at Poznan University of Technology (PUT). Software Department is responsible for delivering software to various units of PUT, e.g., for students and for tutors to manage the grades, for dean office employees to manage some documents. Software Department cooperates with Infrastructure Department responsible for the management of hardware and network infrastructure (e.g., servers), and Department of Service Development responsible for the quality of software services provided at the University (e.g., Service Desk). Moreover, Software Department closely works with Software Development Studio (SDS), which allows increasing its software development capacity. Each year within SDS ca. 5 software projects are run in which students are to learn good software engineering practices by participating in a real-life project.

In each project there are the following stakeholders: (1) employees of Software Department (responsible for product quality, suppliers of some data, and future software maintainers), (2) one person from Software Engineering Laboratory (SEL) (responsible for process quality, so-called quality assurance), (3) customer representative(s) (University employee(s) responsible for business value, and for management of project budget), (4) representatives of end-users (employees of PUT, students or partners of PUT who are to use

the software product), (5) a mentor (an employee of PUT, who takes care of the team of developers), and (6) the participants of the SDS (two or three Master students who play roles of project manager, analyst and architect, and four Bachelor students who are software developers and testers).

Each team is composed of 9 to 14 people; as in other software development organizations they are senior and junior technical- and process experts. In each project there are at least representatives of the voices of customer, user, and maintainers. Each project is run according to the XPrince methodology [169] and lasts for about a year. There are two or three releases, and each one is finished with the delivery of the software to the real environment. According to XPrince, requirements are gathered after defining a business case and analyzed before architecture design, and then just before each release the requirements are being improved and made more precise during GUI workshops and planning games. Functional requirements are documented in use cases and frequently refined to user stories. Completion of FR elicitation was a prerequisite for teams to take part in the study. The teams work in the rooms assigned to them with hardware and software equipment. Moreover, each project's budget allocates salaries which are at the level of salaries that students may earn working part-time in industry. The projects take ca. 1000-1200 person-hours. More information about the environment can be found in the paper by Kopczynska et al. [131].

The case study was carried out in 2011–2012 (1 project) and in 2012–2013 (6 projects). The projects had the following goals:

- Project **A** – the aim of the project was to develop a module for the Moodle platform (http://moodle.com) that allows conducting surveys among students and graduates.
- Project **B** – the goal of the project was to develop a system for management of teaching duties so that a dean office employee can plan the assignment of tutors to the courses.
- Project **C** – the aim of the project was to build a system for collecting and analyzing data about University units (a kind of management information system).
- Project **D** – the goal of the project was to develop a system for management of organizational duties of University employees and for tracking employees effort.
- Project **E** – the goal of the project was to develop a system for curricula construction, their verification, and evaluation with respect to the requirements of the Ministry of Higher Education.
- Project **F** – the aim of the project was to develop a system to support Bachelor and Master examination jury, so they can provide grades, results of the theses reviews, and generate correct documents.
- Project **G** — the aim of the project was to enhance the existing system for management of grades with a new module dedicated to an administrator, and for management of data from the previous examination sessions.

We asked each project to elicit NFRs at the beginning of their project, just after the business case and the overview of functional requirements get accepted. They were to use the template-supported elicitation method called SENoR (see the description of the process in Section 4.3.2). They conducted elicitation workshops with the representatives of investor (client), users, and those suppliers who would maintain the software in the future. We encouraged them to conduct, if possible, a separate workshop with the maintainers. Then, after the development was finished, we asked each team for the final sets of NFRs, i.e., the one with all valid NFRs at the end of the project.

Elicitation process

In the study we asked each team to use *SENoR*, i.e., a method for eliciting NFRs that consists of 3 steps: Preparation, Workshop, and Follow-up. Workshop is the cornerstone of the method, and is executed as a series of short brainstorming sessions. Each session is dedicated to one ISO25010's quality subcharacteristic [113], and is supported by a catalog of NFR templates. In the following sections, we describe how the method was applied in our study, more information about the approach can be found in the published papers [128, 129]. SENoR

Roles and responsibilities. SENoR defines the roles of Moderator, Presenter, Recorder, and Experts. The first three roles were played by the project stakeholders who were analysts, project managers, and architects. They had the following responsibilities:

- **Moderator** – facilitated the whole process, e.g., was to take care about the proper pace, conformance with agenda, enforce the rules, so the objectives could be met; was responsible also for completing administrative and logistical tasks.
- **Presenter** – at the beginning of Workshop introduced the participants into the idea of the project for which NFRs would be elicited. Presenter had to describe the goal of the project, business drivers, and the main functions.
- **Recorder** – recorded the work of participants—the elicited NFRs and action items, the person who played the role of analyst in each project was asked to be Recorder.

The other project stakeholders who participate in Workshop—**Experts**—were people who had the content expertise for defining requirements. The representatives of investors (clients), users, future maintainers, process quality assurance were asked to actively take part in brainstorming about the NFRs during Workshops to express their expectations, and share domain knowledge.

Steps. To assure that each team will execute the template-supported process, those who were to play the Moderator and Recorder roles participated in a tutorial on NFRs. The first part of the tutorial (1h 30 min.) was on general knowledge of NFRs. During the second part (1h 30 min.) participants solved some tasks regarding NFRs, familiarize themselves with guidelines on organizing Workshops, and played with the software tool used to record Workshops (see Section 4.3.2).

Preparation is the initial step of SENoR and aims at planning and preparing the next step—Workshop. During Preparation each Moderator ensured that resources and facilities were available, invited other project stakeholders, assigned roles, scheduled and arranged Workshop.

The goal of the next step, i.e., *Workshop*, is to elicit NFRs going through agenda items one by one (e.g., the agenda is presented in Figure 4.2). We encouraged, where possible, to organize two separate Workshops: (1) for the representatives of investors and users, (2) for the representatives of future maintainers. As a result, some agenda items could be gone through faster, e.g., future maintainers were not interested in the operability or learnability of the system, which was, on the other hand, very important for users.

At the beginning of each Workshop, each Moderator presented the agenda, and Presenter gave a short reminder of the project business case (the overview presentation). Moderator also distributed a catalog of NFR templates in a pen-and-paper form to each participant. Then, participants discussed each ISO25010's subcharacterisitc one by one in a three-step approach:

(1) Definition – a quick clarification what a given subcharacteritic is about by Moderator,

(2) Individual Work – participants had some time to think over the given subcharacterisitc answering in their minds the question: "What NFRs that can be classified in this category do I expect from the system?",
(3) Discussion – a short brainstorming session, participants proposed and discussed NFRs.

Recorder was responsible for recording Workshop using a camera (audio and video), and for noting down the elicited NFRs, and any other notes, action items, etc.
Workshops ended with Voting. It was not only to make an informed decision about the priorities of each NFR but also some doubts, and inconsistencies were clarified. Moreover, Voting served as a presentation of the results, which was to grant Participants for their work during Workshop. The total time of Workshop should not have exceeded 1.5 hours, which was to be controlled by Moderator.

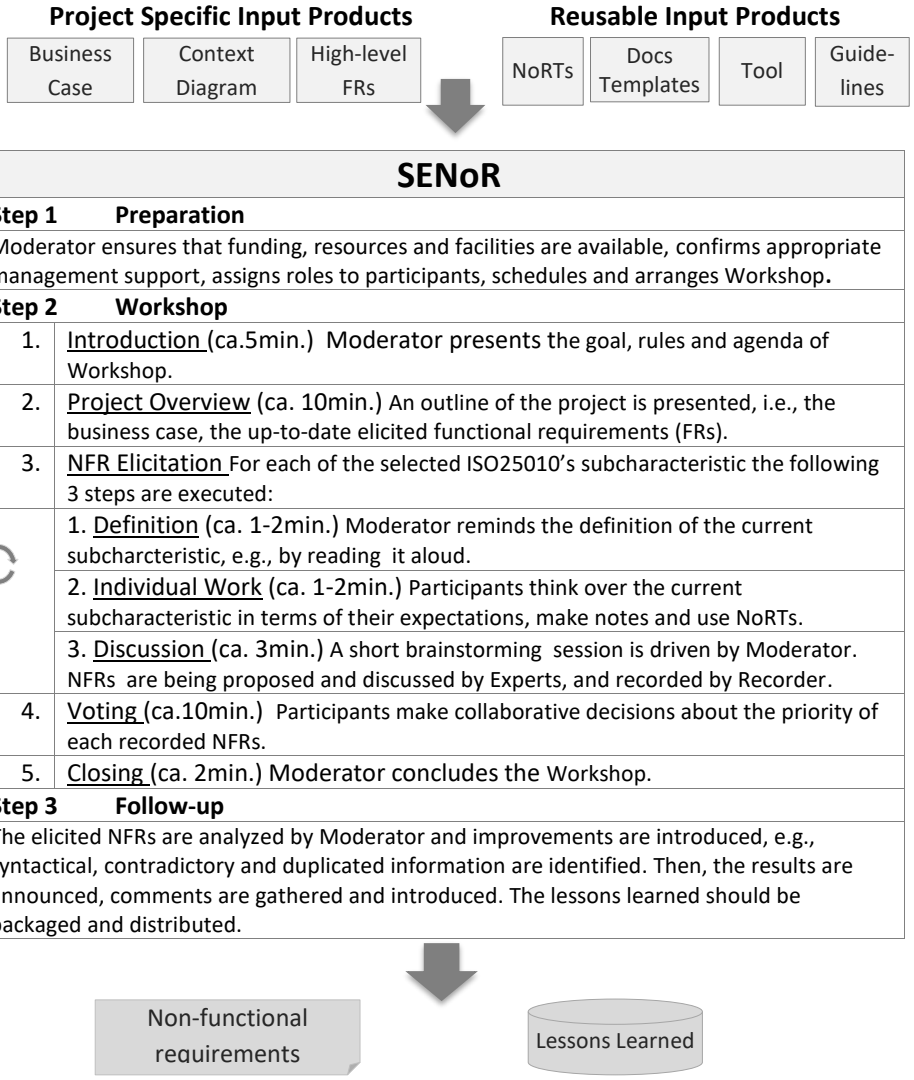


Figure 4.2: Template-supported elicitation of NFRs — SENoR — the process, the input and output products. (Figure taken from [129].)

The final step of SENoR is called *Follow-up*. Then, Recorders were to analyze the elicited NFRs for contradictory, requiring trade-offs and duplicated information, and, if necessary, to improve the construction of the sentences (syntactic analysis). Next, the results were announced, e.g., distributed by mail to the participants, with the notes, and action items.

Input items. The structured elicitation workshop requires the use of the following input items (they can be downloaded from the website of the project [140]), all of them the participants were to use for elicitation.

- *Project overview presentation* – a short presentation given at the beginning of Workshop by Presenter which covers: description of the project business case (a problem statement, impact of the problem, solution of the problem, description of the investor), time constraints of the project, scope of the project, and a description of the main functions.
- *Agenda items presentation* – a presentation that is composed of a set of definitions of the ISO25010's subcharacteristics selected for discussion and simultaneously is used to navigate Workshop.
- *Catalog of NFR templates* – a set of templates of NFRs in the NoRTs notation grouped by the ISO25010 standard's quality subcharacteristics and some space for notes. Such list was distributed, in a pen-and-paper form, among the Workshop's participants.

Support of NFR templates. At the beginning of Workshop, each participant received a catalog of NFR templates divided into subcharacteristics that constitute the Workshop's agenda. The NFR templates were expressed in the NoRTs notation. Some examples of NFR templates are presented in Figure 4.3, for details refer to Subsection 2.2.3.

During Workshop, participants could select templates from the provided catalog of NFR templates and specify their expectations regarding the template's parameters and optional parts—derive NFRs from the selected templates.

The catalog we used in the case study follows from our experience and some pilot studies described in [128].

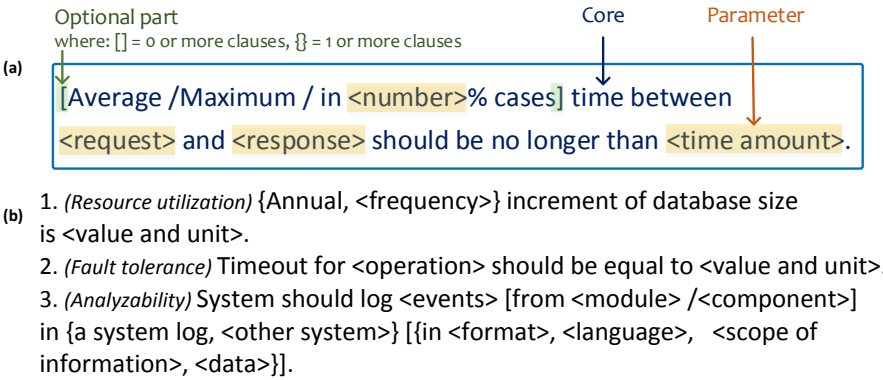


Figure 4.3: (a) An example of NFR templates with the elements of the NoRTs notation; (b) other examples of NFR templates. (Figure taken from [129].)

Tool support. We asked Recorders to use a software tool that aims to help with capturing the work of group during Workshop—MeetingAssistant. Most importantly, it allows recording a set of NFRs just by completing the appropriate NFR templates, supports voting, and exporting data to a document. A demo version can be downloaded from the website of the project [140].

Data collection and analysis procedures

In this case study we considered as feasible the following procedures for data collection:

- *Surveys* – short questionnaires distributed right after each Workshop asking the participants about their impression on the activities and on the quality of requirements.
- *Input and output products* – we collected the input products of SENoR and the resulting list of the elicited NFRs.
- *Recordings* – we recorded each Workshop with Polycom CX5000, which is a device allowing to create a video recording showing the participant who is speaking in a certain moment and is specially designed for video- and teleconferences.
- *Project artefacts* – we collected documents with requirements that contained NFRs and task management reports.

The collected data were analyzed with descriptive statistics. Moreover, to determine the duration of each workshop's agenda item, video recordings were broken into chunks, each concerned one agenda item. Then, to analyze the contents of recordings, we used constant comparison method with a set of preformed open codes, which we intended to extend with postformed ones [214].

4.3.3 Results

In the following paragraphs, we discuss the results of our case study concerning the 7 projects described earlier. The team members organized 13 SENoR workshops in which 50 project stakeholders participated. One team organized just one workshop in which one person played the role of user, investor, and supplier representative.

The obtained values of *Firmness*, *Effectiveness*, and *Catalog Value* (in both regular and strong version) depicted in Figures 4.4, 4.5, and 4.6 and they are also presented in Table 4.1. Looking at those data one can make the following three observations (Observation 4.1., 4.2., 4.3.):

Observation 4.1. *For all the considered projects Firmness of NFRs (i.e., the percentage of initially elicited NFRs that remained valid till the end of each project) resulting from template-supported elicitation was above 70% and for all the projects but one it was even above 80%. Strong firmness was similarly high.*

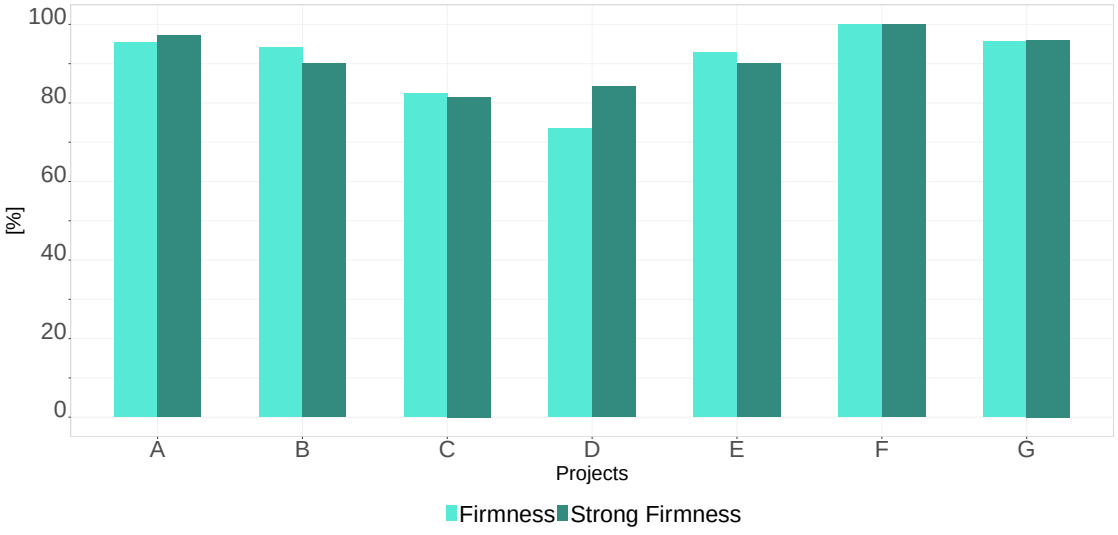


Figure 4.4: Firmness of elicited NFRs (regular and strong) for the considered projects.

	A	B	C	D	E	F	G
Firmness [%]	95.35	94.12	82.36	73.53	92.86	100.00	95.65
Strong Firmness [%]	97.14	90.00	81.48	84.21	90.00	100.00	96.00
Effectiveness [%]	100.00	94.12	82.35	89.29	59.09	100.00	100.00
Strong Effectiveness [%]	100.00	90.00	84.62	100.00	60.00	100.00	100.00
Catalog Value [%]	62.32	65.39	64.15	75.56	45.16	73.33	83.33
Strong Catalog Value [%]	50.73	38.46	51.79	42.20	32.26	53.33	76.67

Table 4.1: Firmness, Effectiveness and Catalog Value (regular and strong) for the 7 projects (A, ..., G).

Observation 4.2. *For all the considered projects but one Effectiveness of template-supported elicitation of NFRs (i.e., the percentage of final NFRs that were elicited at the beginning of each project) was above 80% (in one case it was almost 60%). Strong effectiveness was similarly high.*

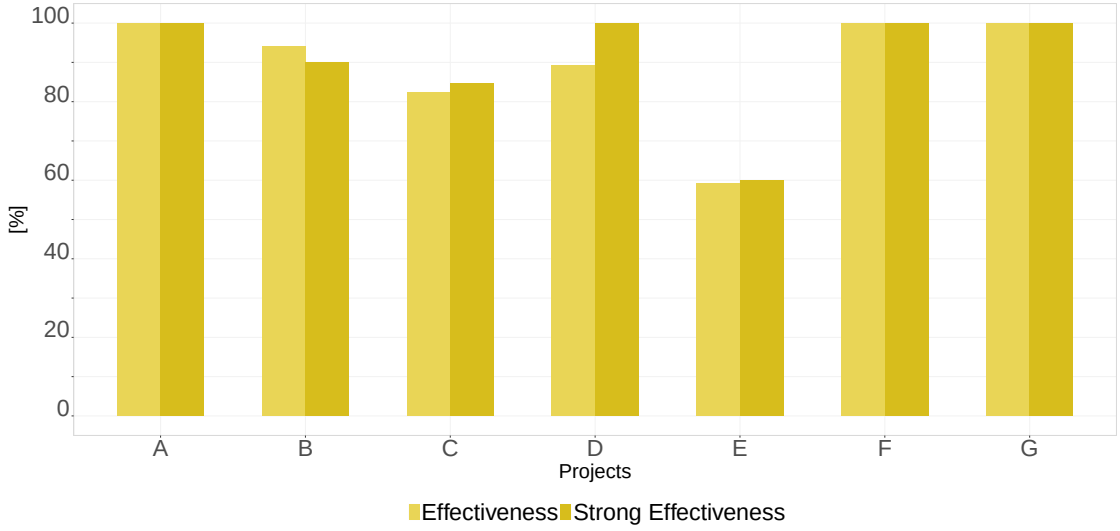


Figure 4.5: Effectiveness of template-supported elicitation (regular and strong) for the considered projects.

Observation 4.3. *For all the considered projects but one Catalog Value (i.e., the percentage of initially elicited NFRs that were derived from the templates) was above 60% (in one case it was 45%). In all the cases but one strong Catalog Value was lower than the regular one by at least 10% (in one case it was about 7%).*

From Observation 4.3. it follows that:

- the catalog used in the case study was rather far from complete (not *mature*),
- the elicitors did not blindly relied on the catalog but also modified the existing NFR templates and identified NFRs not suggested by it (a catalog of NFR templates is an advanced form of a prompt list, and some people warn against prompt lists as they can limit 'creativity' of the elicitors).

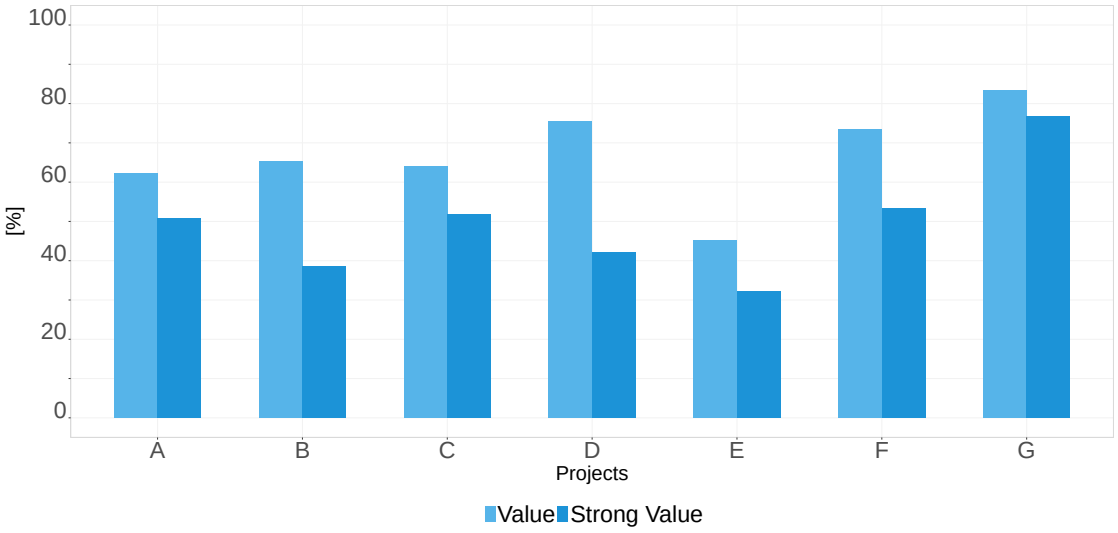


Figure 4.6: Catalog Value (regular and strong) for the considered projects.

In Subsection 4.3.2 impression of the participants about the usefulness of the catalog has been decomposed into three questions, and the summary of the received answers is depicted in Figure 4.7. From the data the following observation follows:

Observation 4.4. *84% of participants of the elicitation workshops regarded the catalog of NFR templates as useful.*

Discussion. The three questions about impression have been asked right after the elicitation workshops. There were 50 participants, and 38 of them responded. 84% of the respondents (32 people) regarded NFR templates as useful (the answers *Strongly yes* or *Rather yes*) and 82% of them (31 people) have had a positive opinion on the quality of the templates. However, they were a bit less optimistic about the quality of the NFRs elicited by them in the context of architecture design—only 74% of the respondents (28 people) answered this question selecting the *Strongly yes* or *Rather yes* options.

□

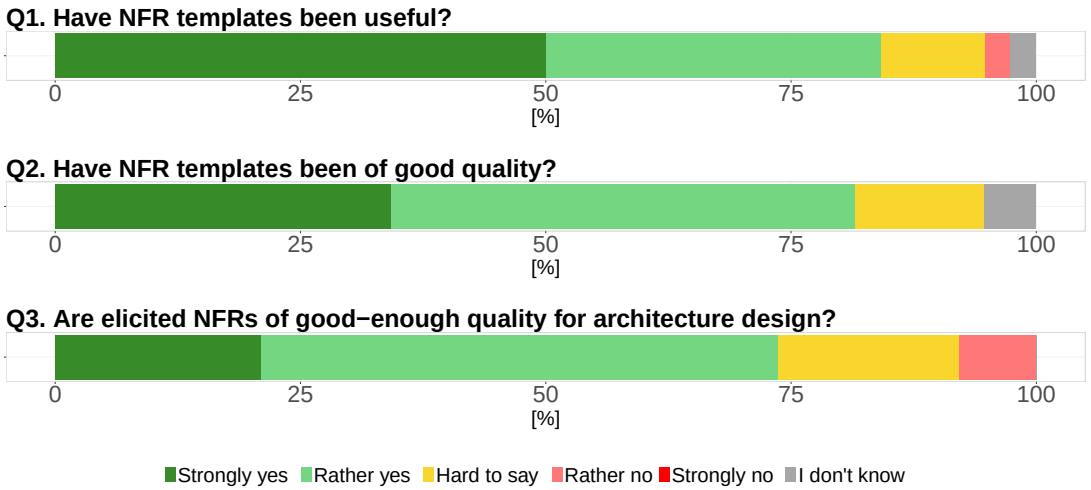


Figure 4.7: The results of the survey carried out at the end of workshops.

Observation 4.5. *Two 90-minute elicitation workshops supported with a catalog of NFR templates seem sufficient to obtain a good-enough set of NFRs.*

Discussion. The average total time of workshop was ca. 1h 16min. (minimum 55min., maximum 2h 5 min.), which is the time to execute the planned agenda items. The time for Project overview, Elicitation of NFRs, and Voting is presented in Figure 4.8. A short brainstorming session on a single quality subcharacteristic took ca. 3-4 minutes. During two workshops of project B, ca. 4min. and ca. 6min. were spent on clarifying the functional scope and the project aim which has not been properly presented (participants had mixed feelings about the quality of the presentation, which was confirmed by the results of the survey). During one workshop of project F, one function was added while presenting the overview of the project, which took ca. 6min. The presentations took on average ca. 4min. (minimum 1.5min., maximum 7.5min.), and 92.11% (35) survey respondents agreed that it is worth to present business drivers before elicitation of NFRs.

□

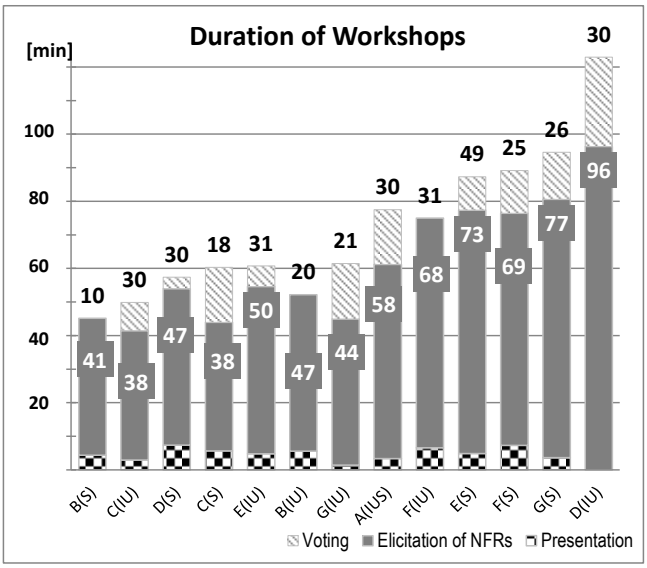


Figure 4.8: Duration Workshops. The white font is used for the duration of the Elicitation of NFRs , and the black font for the number of elicited NFRs. (Figure taken from [129].)

4.3.4 Threats to validity

External validity. Our case study was not designed to provide statistically valid conclusions for all software development projects or organizations. Its aim was to thoroughly describe and deeply understand the proposed method regarding the context. We tried to provide a detailed description of the context in Section 4.3.2.

The threat of not having the setting representative of industrial practice was addressed by the decision of conducting the study with Software Development Studio at PUT. There are of course different types of Software Studios, e.g.,[206, 234], which meet the goal of conscious application of good software engineering practices in a different manner. We claim that the Software Development Studio at PUT simulates the real environment well because there are real investors, user representatives, dedicated workspaces, the teams work with other employees, and most importantly they sign contracts, may obtain salary equal to an average salary that a student may get working part-time, and participate in the transition of the developed software to the final environment.

Internal validity. Another threat might have been caused by environmental (e.g., light, temperature) or human-related (e.g., a person who keeps interrupting, strongest voices which tend to win discussion) factors that might have affected the workshops. We did not try to eliminate the factors, as our aim was to evaluate the method in real life, and such factors normally appear. Moderator is then responsible for overcoming such obstacles. We asked participants of our study in a questionnaire about any remarks, and we analyzed the recordings of workshops. We did not identify any factor that might have disturbed the elicitation of NFRs. In two workshops Voting was not executed due to time constraints of the participants, and in one project there was a failure with the recording device and we are not able to measure its duration. However, our observations focus on the elicitation of NFRs, so we regard that the mentioned events have a negligible if any influence on the findings.

Construct validity. In order to sufficiently define the constructs, we described the template-supported elicitation process, cases, participants, and the indicators (measures) used during data analysis.

To assure *reliability* of study we linked all the data to project repositories. The measurement of the duration of the agenda items of the workshops was done twice and the precision of half a minute was applied. To assure the understandability of the context of the study—Software Development Studio—we described the case in Section 4.3.2 and provided a reference to the more detailed work about it (see [131]).

4.4 Experiment

4.4.1 Introduction

To investigate the role of catalog of NFR templates as a mean supporting individuals and teams in elicitation of NFRs we decided to conduct an experiment. The goal of the study can be formulated as follows:

Goal. Compare template-supported elicitation of NFRs with *ad hoc* one focusing on (1) quality of elicited NFRs, (2) speed of elicitation. The considered context is: inexperienced elicitors of requirements working both individually and in teams.

We refined our goal into the following set of research questions.

First, since a catalog the templates contains lessons learned from past projects it shall assure that each NFR formulated using the catalog is of high quality (e.g., unambiguous). Moreover, such catalog is also to minimize the risk that some NFR concerns are omitted. Thus, the following research question arises:

RQ4.6. What is the impact of using catalog of NFR templates on *quality* and *completeness* of elicited NFRs?

Secondly, the speed of elicitation can be understood as the number of NFRs elicited per unit of time. When the process is too slow, it can hinder its usefulness (esp. its operability). *Ad hoc* elicitation could be slower than the template-supported one as one has first to brainstorm different NFR concerns, select those that apply, and then formulate statements. Thus, the following research question arises:

RQ4.7. What is the impact of using catalog of NFR templates on *speed* of elicitation?

Moreover, since the opinion of users of any method is very important, we decided to instigate what is the perceived usefulness of NFR templates:

RQ4.8. What is the *impression* of those you have used catalog of NFRs templates?

4.4.2 Terminology

Many quality characteristics can be defined in the form of a question for which there is a binary answer, i.e., *yes* or *no*. This approach is very useful in the context of an experimental evaluation of requirements, and we have applied it in our research.

In this paper we are interested in the following requirement quality characteristics related to a single NFR:

- *Individual Completeness*: Does a given NFR contain all the information necessary to implement it and verify the correctness of the implementation?
- *Verifiability*: Is there a finite, cost-effective process to check if the software product satisfies a given requirement?
- *Unambiguity*: Will a given requirement be understood in the same way by different people (i.e., has it only one interpretation)?

Moreover, we consider one more quality characteristic which concerns a set of requirements:

- *Set Completeness*: Does a given set of NFRs contain everything pertinent to the definition of the software product being specified?

The above characteristics are based on the IEEE 830 [109] and 29148 [110] standards (we have omitted Singularity as we consider it less important). All of them allow to assess the quality of NFRs right after elicitation, and they do not require any more information except a business case and a solution outline.

As mentioned earlier, the quality assessment required from us answering the above-stated questions. For example, in the case of Verifiability, the answer to the requirements "System shall be available from *every* Internet browser" or "System shall be resistant to [*any*] viruses that could be on user computers" would be "no" because the constraints imposed by the NFRs are too wide and extremely costly (or even impossible) to verify (e.g., one would have to verify the resistance of a system to all possible computer viruses). On the contrary, the answer to the requirement "The size of the output file cannot be greater than 1 MB" would be "yes" since the size of the file can be verified. The proposed strategy is consistent with the definition of Verifiability and the examples provided in the IEEE 830 standard.

4.4.3 Study design

We designed our study as a laboratory experiment and followed the guidelines described by Wohlin et al. [241].

Variables

We considered only one **independent variable**, elicitation approach, with the following treatments (values): template-supported or *ad hoc*.

Before we define dependent variables, we need to introduce three auxiliary notions:

- *Reference specification.* It is the set of NFRs for a given business case of the highest possible level of completeness (set completeness) that might be achieved in a study. Initially, we had created it during preparation to the study, when we went through the business case and elicited appropriate NFRs (it was a brainstorming followed by cleaning and filtering). Secondly, after the elicitation sessions with the participants, we added sensible requirements identified by them but missing in our version. For each requirement in the reference specification, there is a template in the catalog from which the requirement was (or could be) derived.

- *Theme-equivalence*. Let r be a requirement belonging to the reference specification and t be a template from the catalog associated with r (i.e., r is derived from t). Requirement r' is *theme-equivalent* to r if it is possible to derive from t requirement r'' strongly equivalent to r' (strong equivalence was defined in Subsection 2.1.5). In other words, r and r' concern the same topic or aspect of the software product—they differ only in values of some parameters.
- *Relevance*. Requirement r' is *relevant* if there is requirement r in the reference specification such that r' is theme-equivalent to r (in other words, r' should have its counterpart in the reference specification which concerns the same topic).

As we wanted to observe *dynamics of the elicitation processes*, we decided to use the ISO 25010 quality subcharacteristics as a prompt list (prompt lists have been in use since many years (see, e.g., [194])), and we asked the subjects to elicit NFRs by going through the list in the order shown in Table 4.2 (each subcharacteristic constitutes a distinct category of NFRs).

Category of NFRs	**ISO 25010 subcharacteristic**
c_1	Functional correctness
c_2	Availability
c_3	Fault tolerance
c_4	Recoverability
c_5	Freedom from risk
c_6	Context coverage
c_7	Operability
c_8	User error protection
c_9	User interface aesthetics
c_{10}	Accessibility
c_{11}	Analysability
c_{12}	Changeability
c_{13}	Testability
c_{14}	Confidentiality
c_{15}	Integrity
c_{16}	Nonrepudiation
c_{17}	Accountability
c_{18}	Authenticity
c_{19}	Time behavior
c_{20}	Resource utilization

Table 4.2: The ordered list of ISO25010 subcharacteristics used as a prompt list during NFRs elicitation in our study.

Let k denote the number of NFRs categories of Table 4.2 a given participant (or team of participants) went through. As the starting point was c_1 and the participants went through the categories sequentially in the order shown in Table 4.2, the k parameter can be considered a counterpart of time.

The **dependent variables** observed in our experiment refer to a set of NFRs produced by a participant (or a team of thereof) and they can be defined as follows:

- Unambiguity ratio, $Unam(k)$, is the percentage of unambiguous NFRs (see Subsection 4.4.2) in a subset of NFRs containing all the NFRs related to any of the categories $c_1,\ldots,c_k$.
- Verifiability ratio, $Veri(k)$, is the percentage of verifiable NFRs (see Subsection 4.4.2) in a subset of NFRs containing all the NFRs related to any of the categories $c_1,\ldots,c_k$.
- Individual completeness ratio, $IndCom(k)$, is the percentage of individually complete NFRs (see Subsection 4.4.2) in a subset of NFRs containing all the NFRs related to any of the categories $c_1,\ldots,c_k$.

Topics	Courses of the cs students	Courses of the mgmt students
Basics of: HTML, SQL, C or Visual Basic	Introduction to Computer Science (15 lec., 15 lab.)	Computer Science in Management Part 1 & Part 2 (30 lec., 30 lab.)
	Databases (15 lec., 15 lab.)	Information Technology (30 proj.)
Project life cycle phases (the waterfall model and the agile model)	Software Engineering (15 lec., 15 lab.)	Project Management (15 lec., 15 lab., 15 proj.)

Table 4.3: Excerpt from the syllabuses. The topics important from the perspective of our study and the courses covering them that the participants attended (lec.=lecture of 1.5h, lab.=laboratory of 1.5h, proj.=project).

- Set completeness ratio, $SetCom(k)$, is the ratio between number of relevant NFRs in a subset of NFRs containing all the NFRs related to any of the categories $c_1,\ldots,c_k$ and the number of NFRs in the reference specification that are related to the same categories $c_1,\ldots,c_k$.
- Yield, Y, is the number of relevant NFRs in a given set of NFRs elicited in one hour (i.e., in this case, all the categories are taken into account).

The *impression* of those who have used a catalog of NFR templates we decided to study based on the responses to the following questions:

Q1. Have NFR templates been useful?
Q2. Have NFR templates been of good quality?
Q3. Are the elicited NFRs of good-enough quality for architecture design?

The answer to each question could be one of the following options (they form well-known 5-level Likert scale): *Strongly agree*, *Rather agree*, *Hard to say*, *Rather disagree*, *Strongly disagree*. For each question, the percentage of responses for each of those options would be calculated.

Participants and their assignment to treatments

Our study investigated the population of inexperienced elicitors of NFRs, and it was based on convenience sampling. The participants were 3rd-year students working towards their Bachelor degree at the Poznan University of Technology. 91 of them studied Computing Science (cs), and 16 studied Management Engineering (mgmt). According to their program syllabuses, they had already acquired some programming skills and basic knowledge on software projects life-cycles (more details can be found in Table 4.3).

In general, there are two *modes of elicitation*: by an individual and by a team (the latter can involve different number of participants). We decided to consider both of them. As the number of mgmt students was considerably smaller than the number of cs students, we have created teams of two types:

- mixed teams: each of them consisted of 3 students of Computing Science and 1 student of Management Engineering,
- cs teams: they consisted of 4 students of Computing Science each.

The experiment was divided into 10 experimental sessions due to the limited size of the available rooms. First, we used the round-robin approach to assign treatments to the sessions, then to assign participants to modes of elicitation within each session. The number of teams or individuals of a particular type can be found in Table 4.4.

		Templ.	*Ad hoc*
Teams	**Mixed**	9	4
	Cs-only	3	3
Individuals	**Cs students**	14	14
	Mgmt students	3	–

Table 4.4: Number of subjects per mode of elicitation (teams vs. individuals) and treatment.

In order to present and discuss results concerning individuals and teams in a similar way, we will use the term **virtual subject** which means either an individual or team who elicited a set of NFRs. For instance, from Table 4.4 it follows that 29 virtual subjects used templates and 21 virtual subjects took the *ad hoc* approach.

Object and instrumentation

The object of the experiment was a project aimed to deliver an e-commerce system, called *StudentDeal*, similar to eBay, Amazon, or Allegro. The system is unique in that sense that only students of a given university can make purchases, and only companies accepted by the authorities of the university can sell products and offer services (see [140] for the business case file).

Since we assumed that a business case and a solution outline constitute the minimum required information about the system under analysis (see Subsection 4.4.2), the following instrumentation was provided to the participants (see [140] for the files):

- **Presentation slides** contained the description of the StudentDeal project—business case (8 slides) and the solution outline (7 slides), and a short reminder about software development project life-cycle (1 slide) and NFRs (17 slides). There was a section (3 slides) about how templates of NFRs should be used—it was presented and made available only to the participants expected to use the template-supported approach. As the participants were expected to elicit NFRs as if they were members of a real project team, there was 1 slide presenting the expected behavior.
- **Results form** was a prepared MS Word file to which the participants wrote down the elicited NFRs (there was one computer per a virtual subject). The file contained names and definitions of the ISO 25010's subcharacteristics. The version of the form given to the template-oriented participants contained a catalog of 83 NFR templates.
- **Questionnaire** was used to collect information about participants' experience in real-life projects, familiarity with Internet shopping, quality of the study instruments, and impressions of the elicitation task. It was a pen-and-paper form.

Experiment execution and analysis procedure

Each session of the experiment consisted of the following steps:

1. **Presentation**. Using the *presentation slides*, one of the authors gave a short presentation (ca. 20 minutes).
2. **Experiment task**. The participants elicited NFRs for the *StudentDeal* software product and documented them in the *Results form*. They had access to the in-print *presentation slides*. The time allotted was 60 minutes.
3. **Survey**. The participants filled in the *questionnaire* (it took about 5 minutes).

When all the sessions had been completed, the experimenters went through all the NFRs elicited by the participants, one by one, and they evaluated their quality by answering the three questions concerning Individual Completeness, Unambiguity, and Verifiability (each time only yes/no answers were possible—see Subsection 4.4.2). Here a kind of

Delphi process was used: first, each experimenter answered the three questions in private (on an in-print answer sheet), then the answers were announced, and if there was a difference, a short discussion was started, after which a final answer was decided. (This evaluation was the most time-consuming part of the experiment as in total over 1000 NFRs had been elicited—see Table 4.5.) Finally, the values of the dependent variables were determined (see Section 4.4.3) and the following steps of the analysis procedure have been performed:

1. *Identification of outlying observations.* Using descriptive statistics and direct analysis of the *results forms* we checked for potentially suspicious subjects (outliers). We identified only one outlier. It was a *results form* containing templates instead of NFRs (parameter values were missing), so the file was removed.
2. *Confounding factor analysis.* In the case of individual students of Management Engineering (see Table 4.4) there was lack of balance between those using templates (3 persons) and their counterparts taking the *ad hoc* approach (zero). This uneven addition of mgmt students to students of Computing Science could be the source of bias if the students of Management Engineering performed, on average, significantly below (or above) average of the cs students. It was necessary to investigate the impact of this situation. It turned out that 66% of the results of the mgmt students were within 1 standard deviation (SD), and 98% within 2 SD with respect to the mean results of the cs students (for the normal distribution, 95% of values lie within 2 SD from the mean). It means that this addition of mgmt students was not dangerous and one can assume that in this case, the impact of this confounding factor is negligible.
3. *Selection of statistical inference tests.* The analysis of the samples using descriptive statistics, Q-Q plots, and Shapiro-Wilk test[1] convinced us that the normality assumption could be violated. Therefore, we decided to use non-parametric Wilcoxon rank-sum test (the significance level α set to 0.05) and the Cliff's δ non-parametric effect-size measure [44]. We used the thresholds by Kitchenham et al. [126] to interpret the values of Cliff's δ (small: $|\delta|$=0.112, medium: $|\delta|$=0.276, and large: $|\delta|$=0.428).
4. *Statistical hypothesis testing.* We formulated a pair of statistical hypotheses concerning equality of medians, M, for all the dependent variables (see Section 4.4.3), all the modes of elicitation μ (individuals/teams), and all the categories of NFRs, c_k, where $k = 1 \ldots 20$). All of them were tested using the non-parametric Wilcoxon rank-sum test. It was followed by post-hoc power analysis and determination of effect-size measure to estimate the chance of rejecting false null hypotheses.

4.4.4 Results

To understand the results of the experiment one needs to realize that the participants worked with different speed in the sense of NFR categories they managed to go through. Table 4.5 presents dependence of the number of virtual subjects on the NFR categories c_k. The higher the NFR category, the smaller the number of virtual subjects (both single participants and teams). Out of 20 NFR categories only the first 5 have been gone through by all the individuals and only 4 by all the teams.

[1] The null hypotheses (stating that the population is normally distributed) were rejected for 24% (37 out of 152) dependent variables and ISO 25010 subcharacteristics. The significance level α was set to 0.05.

	Category	Individuals				Teams			
		#Individuals		#NFRs		#Teams		#NFRs	
		Templ	***Ad hoc***	**Templ**	***Ad hoc***	**Templ**	***Ad hoc***	**Templ**	***Ad hoc***
1	Functional correctness	17	14	37	12	12	7	47	5
2	Availability	17	14	60	19	12	7	42	9
3	Fault tolerance	17	14	50	21	12	7	47	12
4	Recoverability	17	14	40	21	12	7	27	10
5	Freedom from risk	17	14	20	19	11	7	15	9
6	Context coverage	16	14	28	13	11	7	25	11
7	Operability	16	14	58	22	8	7	28	9
8	User error protection	15	13	36	20	7	7	19	9
9	User interface aesthetics	14	13	16	17	5	7	4	9
10	Accessibility	14	13	34	12	5	7	6	6
11	Analysability	10	13	18	16	5	7	8	6
12	Changeability	10	13	11	11	4	7	3	5
13	Testability	8	13	5	12	4	7	3	6
14	Confidentiality	8	13	8	14	4	7	11	3
15	Integrity	8	12	13	19	2	7	6	12
16	Nonrepudiation	6	12	5	13	2	7	2	5
17	Accountability	6	12	5	7	2	7	3	7
18	Authenticity	5	12	12	28	2	7	6	7
19	Time behavior	3	11	4	22				
20	Resource utilization	2	11	2	13				
	Total			**462**	**331**			**302**	**140**

Table 4.5: Number of virtual subjects (individual participants or teams) with the number of NFRs they elicited.

Set Completeness

Observation 4.6.1. *In the context of inexperienced NFRs elicitors, template-supported elicitation, when compared to ad hoc one, results in greater Set Completeness of elicited NFRs for both individuals and teams.*

Justification. The assumed indicator of *Set Completeness* is the *Set Completeness Ratio* variable (see Section 4.4.3). The distribution of this variable over the NFRs categories is presented in Figure 4.9 (see Table 4.6 for more precise values). The figure itself is a good argument in favor of the observation. Nevertheless, we performed also statistical hypothesis testing.

Let $SetCom^{\mu}_{\tau}(k)$ denote value of $SetCom(k)$ for treatment τ (either template-supported (Templ), or *Ad hoc*, (Ad hoc)) and elicitation mode μ (either Individuals or Teams). We formulated a pair of hypotheses concerning equality of medians M for each elicitation mode μ, and for all categories of NFRs (k=1,..., 20):

$$H_0 : M(SetCom^{\mu}_{Templ}(k)) = M(SetCom^{\mu}_{AdHoc}(k))$$

$$H_1 : M(SetCom^{\mu}_{Templ}(k)) > M(SetCom^{\mu}_{AdHoc}(k))$$

As shown in Table 4.7 columns 3-5, the null hypotheses could be rejected with the assumed significance level in all but two cases. The only two exceptions were for individuals that covered the last two categories (k = 19 and 20, p-values equal to 0.100 and 0.068). The calculated post-hoc power for these tests was equal to 0.327 and 0.128. Therefore, the chance of detecting the difference (if it exists in population) was limited.

			Min.	Mean	Median	Max.
Individuals	**templ.**	**Set Complet.**	0.00	0.23	0.23	0.44
		Unambiguity	0.00	0.77	0.79	1.00
		Indiv. Complet.	0.50	0.88	0.92	1.00
		Verifiability	0.00	0.87	0.90	1.00
		Yield	11.00	27.00	26.00	40.00
	ad hoc	**Set Complet.**	0.00	0.11	0.11	0.31
		Unambiguity	0.00	0.47	0.50	1.00
		Indiv. Complet.	0.00	0.63	0.64	1.00
		Verifiability	0.00	0.54	0.50	1.00
		Yield	8.00	24.00	23.50	38.00
Teams	**templ.**	**Set Complet.**	0.00	0.30	0.28	0.57
		Unambiguity	0.60	0.79	0.78	1.00
		Indiv. Complet.	0.50	0.83	0.83	1.00
		Verifiability	0.67	0.88	0.88	1.00
		Yield	11.00	24.00	23.00	34.00
	ad hoc	**Set Complet.**	0.00	0.10	0.11	0.25
		Unambiguity	0.00	0.46	0.78	1.00
		Indiv. Complet.	0.00	0.66	0.83	1.00
		Verifiability	0.00	0.54	0.88	1.00
		Yield	13.00	21.00	21.00	30.00

Table 4.6: Minimum, maximum, average and median values for all independent variables (Set Complet. = Set Completeness, Indiv. Complet. = Individual Completeness).

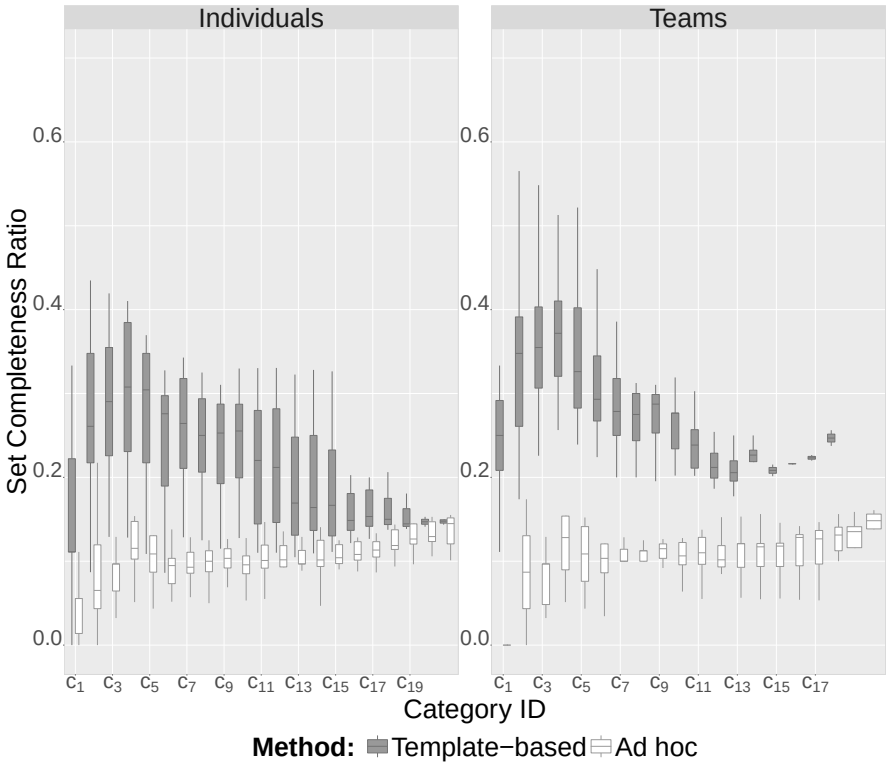


Figure 4.9: Distribution of Set Completeness Ratios over NFR categories $k = 1 \ldots 20$. (Figure taken from [132].)

The values of Cliff's δ for both modes of elicitation and all categories were not less

than 0.455, which can be interpreted as *large* effect size. On average the participants using the template-supported approach formulated around 2.0 and 2.5 times more NFRs than those using the *ad hoc* approach (individuals and teams, respectively).

Note: Some of the participants using the template-supported approach had a strategy to select one or two templates for each category. It allowed them to proceed fast and cover many categories, but at the same time, it resulted in low completeness (the difference between the two treatments decreased with the elicitation progress).

Quality of single NFRs

Observation 4.6.2. *In the context of inexperienced elicitors of NFRs, template-supported elicitation results in NFRs of better quality, with respect to individual completeness, unambiguity, and verifiability, for both individuals and teams.*

Justification. The assumed indicators of quality of NFRs are the following dependent variables: *Unambiguity Ratio, Individual Completeness Ratio*, and *Verifiability Ratio* (see Section 4.4.3). The distributions of those variables over the NFR categories are presented in Figure 4.10. The figure itself seems a good argument in favor of the observation. Nevertheless, we performed also statistical hypothesis testing in a way similar to the one used in the previous section.

Let $\varphi(k)$ denote one of the variables $Unam(k)$, $Veri(k)$, and $IndCom(k)$ (see Sec. 4.4.3). Moreover, let $\varphi^{\mu}_{\tau}(k)$ denote values of $\varphi(k)$ restricted to a treatment τ and a mode of elicitation μ. We formulated a pair of hypotheses concerning equality of medians, M, for each of the variables $\varphi(k)$, each elicitation mode μ, and for all categories of NFRs (k=1 ... 20):

$$H_0 : M(\varphi^{\mu}_{Templ}(k)) = M(\varphi^{\mu}_{AdHoc}(k))$$

$$H_1 : M(\varphi^{\mu}_{Templ}(k)) > M(\varphi^{\mu}_{AdHoc}(k))$$

The results of hypothesis testing (p-values) and magnitudes of effect size (Cliff's δ) with their interpretation are presented in Table 4.7 and Table 4.8 columns 6-14.

The average values of all quality indicators were visibly higher for participants using the template-supported approach. For individuals using templates, these indicators were from 1.42 to 1.67 times higher than for their counterparts working without templates. The same tendency was observed for teams: the quality indicators for the template-supported approach were from 1.31 to 1.68 times higher than for the *ad hoc* approach. We also observed lesser variability of the quality indicators for the template-supported approach than for the *ad hoc* one (see Figure 4.10).

For individuals, the null hypotheses were ***rejected*** for all but two tests (see Table 4.7, row 1 and columns 6-8 and 9-11). The two exceptions were for category index $k = 1$ (corresponding to Functional correctness) with respect to unambiguity and individual completeness. The observed effect size for these cases could be perceived as small (Cliff's δ equal to ca. 0.20). The calculated post-hoc power of these tests was equal to 0.172 and 0.333. For other subcharacteristics, the size could be interpreted as **large** for all but four cases.

For teams, the ***majority*** of null hypotheses was ***rejected*** (17 out of 18 for unambiguity, 16 out of 18 for verifiability, and 9 out of 18 for individual completeness). Only for the first three categories, the Cliff's δ indicated effect size smaller than **large**. Still, the post-hoc power of the tests for which the null hypotheses could not be rejected was low

in most cases and ranged between 0.176 and 0.502. Therefore, the main factor influencing our chances of rejecting the null hypotheses was the limited number of teams in the experiment.

Yield

Observation 4.7. *In the context of inexperienced requirements elicitors, template-supported elicitation does not seem to speed up the elicitation process.*

Justification. To investigate whether the template-supported approach hinders the speed the elicitation process we have introduced variable Y_{τ}^{μ} describing values of the independent variable yield Y restricted to a treatment τ and elicitation mode μ. Then, we formulated a pair of statistical hypotheses regarding equality of medians, M, for both modes of elicitation μ (individuals and teams):

$$H_0 : M(Y^{\mu}_{Templ}) = M(Y^{\mu}_{AdHoc})$$
$$H_1 : M(Y^{\mu}_{Templ}) > M(Y^{\mu}_{AdHoc})$$

None of the null hypotheses could be rejected with the assumed level of significance (p-values were equal to 0.231 for individuals and 0.263 for teams). The observed effect size could be interpreted as ***small*** (Cliff's δ equal to 0.16 and 0.19, respectively) in favor of NFR templates.

Perceived usefulness

Observation 4.8. *In the context of inexperienced requirements elicitors, templates are regarded by 86% of elicitors as useful for elicitation of NFRs.*

Justification. Three questions about impression concerning templates presented in Figure 4.11 have been asked right after the elicitation task. There were 29 virtual subjects and 65 elicitors responded. 86.15% of the respondents (56 people) regarded NFR templates as useful (the answers *Strongly yes* or *Rather yes*) and 72.31% of them (47 people) had a positive opinion on the quality of the templates. However, they were less optimistic about the quality of the NFRs elicited by them in the context of architecture design—only 58.73% of the respondents (39 people) answered this question selecting the *Strongly yes* or *Rather yes* options.

The pattern is visible for both individuals and teams, thus in Figure 4.11 we present the aggregated results, for more details refer to Appendix A.

□

Interestingly the Observation 4.8 is aligned with the Observation 4.4 (see Subsection 4.3.3) that follows from the case study. In both studies over 84% of respondents regard NFR templates as useful.

4.4.5 Threats to validity

In the following paragraphs, using the guidelines provided by Wohlin et al. [241], we discuss the validity threats that we identified as having an impact on our study.

Conclusion validity

Random irrelevancies in experimental setting. Such elements as light, noise, table setting, etc. might influence the results. We did not observe anything that might have

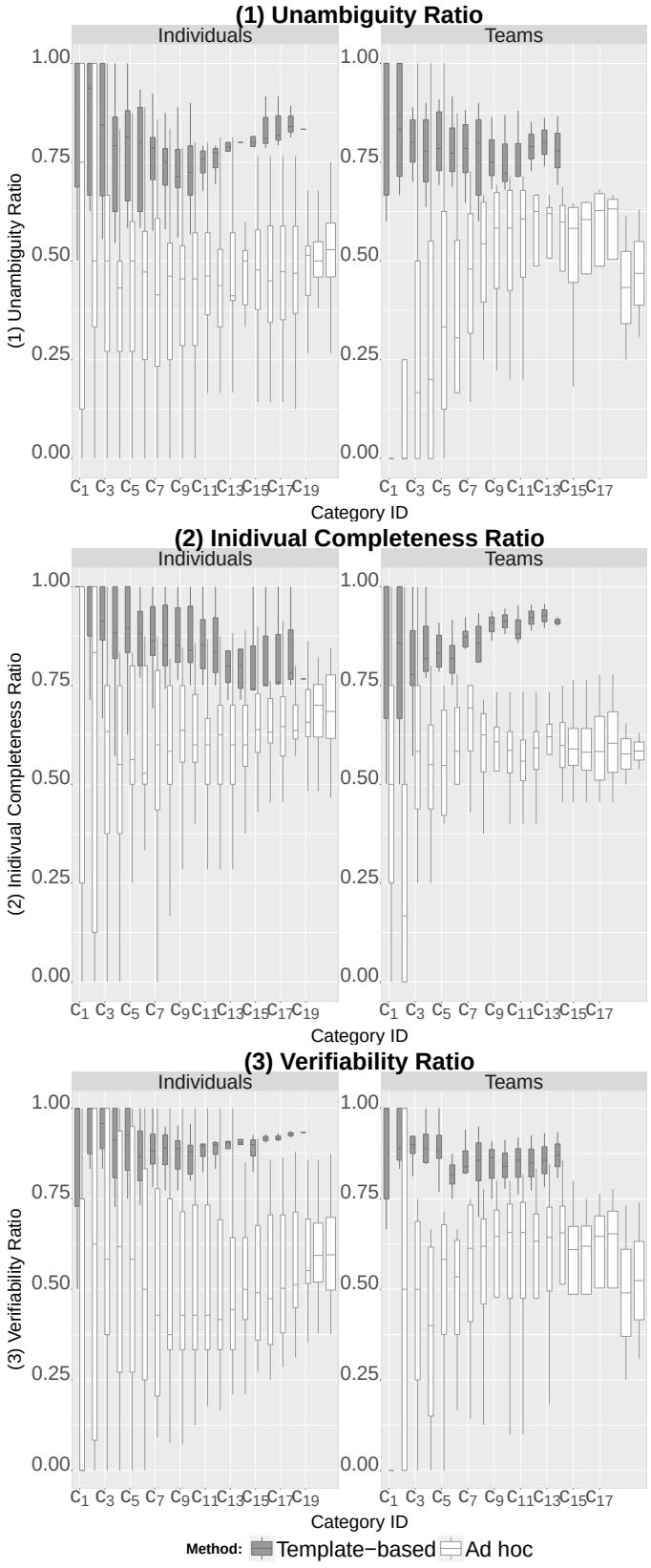


Figure 4.10: Distribution of three dependent variables describing quality of elicited NFRs. (Figure taken from [132].)

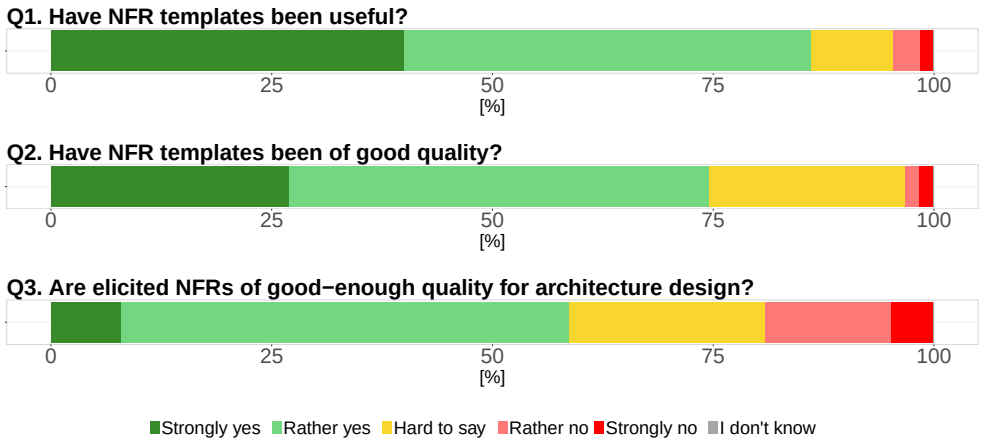


Figure 4.11: The results of the survey carried out at the end of elicitation.

threatened the study. None of the participants suggested the existence of any factors that could disturb their work in the questionnaire distributed at the end of the experiment.

Low statistical power. There are at least two factors that could affect the power of the tests in our experiment. Firstly, we used non-parametric tests, which impose less strict requirements on variable distributions but are also less powerful than their parametric counterparts. Secondly, for some of the ISO 25010 subcharacteristics, the number of subjects was too low to achieve high power for the tests. We used the G*Power tool [73] to estimate the post-hoc power of the tests using the asymptotic relative efficiency (A.R.E.) method.

Multiple testing. In the experiment, we run a separate statistical inference test for each dependent variable and ISO 25010 subcharacteristic. Therefore, a problem of multiple testing could materialize. However, even if some of the null hypotheses were falsely rejected, we did not build our conclusions on the results of particular tests but on a general view of how a specific approach performed for multiple different categories. Therefore, the problem of multiple testing should not threaten the overall study outcomes.

Fishing. The experimenters might have searched for a specific result when analyzing the data, e.g., favor some elicitation approach. Thus, they had access only to the contents of the elicited NFRs while determining the values of the dependent variables.

Reliability of measures. The main issue here is the subjectivity of quality evaluation as it is based on expert judgment. In the experiment, it was performed by two persons with experience in requirements analysis of minimum 4 years. Also, the detailed guidelines on how to perform the task were created beforehand. To minimize the bias, first, both experts evaluated randomly selected 30% of all NFRs. We characterized the reliability of this evaluation using Cohen κ and overall agreement [137]. We obtained the following values of κ: Unambiguity = 0.767, Individual Completeness = 0.731, Verifiability = 0.714, and of the overall agreement: 0.885, 0.891, 0.872, respectively. The results indicate a substantial agreement between the reviewers; thus, we decided to analyze together these assessments with the other assessments performed by a single experimenter. Although this action mitigates the risk, it obviously does not eliminate it.

Random heterogeneity of subjects. Though we assigned participants the approach and mode of elicitation at random, we asked them about their experience that in our opinion could introduce unexpected variation in the results. The first potential problem could be insufficient domain knowledge, i.e., lack of experience in using e-commerce platforms. Fortunately, all of the participants had at least a few times purchased items over the Internet. The second factor might be the experience in participating in software projects. It appeared that nearly all participants took part in 1–3 industry software development

projects, except for 10 persons (2 individuals—one per treatment, 8 team members—four per treatment). The last considered factor was the experience in eliciting NFRs using templates. It turned out that none of the participants had used templates before the study.

Internal validity

Interaction with selection. Each subject was introduced to the experiment and guided through the presentation by the same experimenter. Uncontrolled exchange of information among the participants not belonging to the same team might have threatened the internal validity. To mitigate this problem, we arranged the rooms in such a way that the distance between the subjects allowed them to work comfortably but prevented communication between different subjects.

Instrumentation. We asked our participants about the quality of the provided materials. Fourteen participants (13.2%) found the descriptions of the ISO 25010 subcharacteristics as of poor quality (from remarks we learned that they are difficult to understand and not clear enough), 3.8% (4) of participants suggested improvement of templates. Since all participants received the same description of the categories, it should affect all of them in the same way. After analyzing the suggested improvements for templates, we did not find them important in the context of the study.

Mono-operation bias. We shall take into account that the completeness and the quality of the requirements the participants created with the use of templates depend on the completeness and the quality of the catalog of templates. Although the templates used in the experiment are taken from a catalog that we created based on 16 software projects (see [129]), still there might exist an area for its improvement, and other catalogs (if they exist) could lead to different results.

Diffusion or imitation of treatments. The participants were not informed that there were different approaches and modes of elicitation under study and which they would be using to minimize the possibility that they share their experience or try to imitate the other treatment.

Compensatory rivalry and resentful demoralization. To motivate the participants and simultaneously mitigate the named risk, we organized a lottery with gift cards to one of the most popular shopping centers for all participants who returned a set of NFRs. Participation in the study did not influence the grades of the courses within which the students were invited to participate in the study. We assured that they would have their presence positively verified. Moreover, we were not involved in these courses, nor in any other course, the participants were taking part in at the time of the study, to avoid trying to please the researchers.

Construct validity

Design threats. In the study, we followed the definitions of an NFR and NFR template explained in Section 2.1. Taking other definitions, from the multiple of existing ones, could lead to other conjectures.

Moreover, whenever someone is going to propose requirements, he or she should be familiar with the domain knowledge. From the survey we learned that 61.9% of the participants did shopping in an Internet shop at least a few times, 33.3% do it a few times per month, 3.8% at least once a week. Therefore, the business case used in the experiment which describes a system for Internet shopping appears to be suitable–easy to get familiar with for the participants so that they spent time on elicitation rather than on learning domain knowledge.

Another threat to validity concerns the development of the reference set of NFRs. First, it might not have reflected all the requirements people expect from such software prod-

ucts. It might also have contained all requirements that were possible to be created using templates and, what is more, the catalog might have contained only templates that are needed to create the NFRs. To mitigate these threats, we did not use the golden solution solely as the reference set but extended it with all the relevant requirements proposed by the participants. We also used the catalog that with respect to the golden solution contained more than needed templates and missed some templates.

External validity

Interaction of selection and treatment. The results of some recent studies show that the decision to involve students does not seem to threaten the external validity as they are those who potentially elicit NFRs in the industry. First, according to Yusop at al. [246] in the web development domain developers are those who elicit requirements including NFRs. Secondly, according to Kalinowski et al. [122] many employees of software development companies are not experts in RE, they miss experience and qualification, which is one of the most commonly reported cause of problems in software projects connected to RE.

Thirdly, to provide evidence that using the students who we selected in our experiment as participants is a valid simplification of reality, we carried out a small survey among the employees of the companies based in Poland cooperating with Poznan University of Technology. Since our experiment concerned inexperienced NFRs elicitors, the goal of our survey was to analyze software project participants focusing on their experience. Thus, we asked about participants' first experience with elicitation of non-functional requirements and their level of education at that moment. In total, 65 people filled in our questionnaire distributed online. It appeared that 40% of them (26 persons) participated in NFR elicitation having less than one year of experience in commercial software projects. What is interesting for 29% (19) were eliciting requirements in their first commercial project. Then, 22% (14) participants had 1-3 years of experience; 12% (8) 3-5 years; 18% (12) over 5 years; 5 did not take part at all. 12 of those inexperienced NFR elicitors, i.e., those who had less than 1 year of experience, were right after the bachelor level; 13 of them had Master of science degree, and 1 person was a high school graduate. Moreover, it turned out that, although the majority of those inexperienced NFR elicitors were computer science engineers (8 BSc graduates, 5 MSc graduates), people who graduated from other study programs also take part in elicitation of non-functional requirements. For example, we had participants studying economy and management (3 MSc graduates), electronics and telecommunication (2 BSc graduates), robotics (2 MSc graduates), bioinformatics (1 BSc graduate), diabetology (1 MSc graduate), and three from humanistic fields like philology or library science.

The students we selected had the following experience: 40% (43) less than one year, 22% (23) from 1 to 3 years, 2% (2) over 3 years, and 36%(39) did not participate in any commercial project. They were students of Computer Science or Management Engineering at the end of their Bachelor studies. Moreover, they participated at least in courses delivering the knowledge and skills on how a software project is organized and the basic idea of computing science (e.g., on project life-cycle, HTML, SQL, C, and Visual Basic).

Our survey shows that it is not uncommon that people with little experience as our students elicit NFRs in commercial projects. Moreover, not all NFRs elicitors have an academic background in computer science. Thus, we perceive that using the students we selected for our experiment as participants seems a valid simplification of reality in our laboratory context.

Interaction of setting and treatment. The experiment task shall reflect the situation we might meet in practice. The business case described a system that is similar to other

popular ones (e.g., eBay, Allegro), the ISO 25010 standard is used during elicitation used in industry, and the time constraint (1.5h with 1 hour for elicitation) reflects a good practice for the duration of a requirements workshop (see Gottesdiener [85]).

The participants had never used the templates for non-functional requirements; hence they were learning this approach during the experiment. This factor could possibly negatively influence the results. In the industry context, browsing a catalog of NFR templates and the decision process on the usage of certain templates might be faster. For example, people might prepare for the meeting by at least going through the templates beforehand, or their experience from using the templates in past projects might speed up the process.

4.5 Summary

In the chapter, we investigated the usefulness of catalog of NFR templates for elicitation of NFRs. We conducted an exploratory case study and a controlled experiment.

In the case study seven software development teams at the beginning of their projects elicited NFRs using template-supported process. They followed the SENoR (Structured Elicitation of Non-functional Requirements) approach composed of a series of short brainstorming sessions driven by the ISO25010's quality subcharacteristics. The elicitation was supported by the templates that were expressed in the NoRTs (Non-functional Requirements Templates) notation. At the end of each project, we compared the NFRs valid at that moment with those initially elicited.

We found that in the studied software development projects:

- (*Observation 4.1.*) Firmness of NFRs (i.e., the percentage of initially elicited NFRs that remained valid till the end of a project) was at the level of 80%;
- (*Observation 4.2.*) Effectiveness of template-supported elicitation of NFRs (i.e., the percentage of final NFRs that were elicited at the beginning of each project) was at the level of 80%;
- (*Observation 4.4.*) Participants of template-supported elicitation of NFR regarded NFR templates as useful (about 85% of them positively rated the usefulness of templates);
- (*Observation 4.5.*) The effort to elicit NFRs with the use of templates of two ca. 1.5-hour elicitation workshops seemed sufficient to obtain good-enough NFRs. The teams did it just after general functional requirements definition, but before architecture design and coding.

In our second study, the experiment, inexperienced NFRs elicitors working both individually and in teams elicited NFRs for an e-commerce system. Some of them, the treatment groups, used a catalog of templates documented in the NoRT notation (see Subsection 2.2.3); and the control groups did not have this type of support (the latter is called *ad hoc* approach). The outcome of the study are the following observations concerning inexperienced elicitors of NFRs:

> Template-supported elicitation results in NFRs which are more complete, less ambiguous, more detailed, and better from the point of view of verifiability, for both individuals and teams (*Observation 4.6.1., 4.6.2.*).

Unfortunately, templates do not seem to speed up the elicitation process (*Observation 4.7.*). In our experiment the number of NFRs elicited per hour was between 21 and 27. Template-supported elicitation was slightly faster than *ad hoc* one, but it was not statistically significant.

Moreover, those who used NFR templates regarded them as useful for elicitation of NFRs (86% of respondents positively evaluated the usefulness of NFR templates) (*Observation 4.8.*).

Individuals												
Category	Set Completeness			Unamibiguity			Indiv. Completeness			Verifiability		
	p-value	Cliff's		p-value	Cliff's		p-value	Cliff's		p-value	Cliff's	
		estim.	interp.		estim.	interp.		estim.	interp.		estim.	interp.
1 Functional correctness	<0.0005	0.617	large	0.185	0.200	small	0.117	0.201	small	0.020	0.467	large
2 Availability	<0.0005	0.899	large	0.033	0.378	medium	0.037	0.336	medium	0.033	0.366	medium
3 Fault tolerance	<0.0005	0.911	large	0.005	0.546	large	0.001	0.689	large	0.025	0.403	medium
4 Recoverability	<0.0005	0.853	large	<0.0005	0.735	large	<0.0005	0.811	large	0.008	0.504	large
5 Freedom from risk	<0.0005	0.853	large	0.001	0.702	large	<0.0005	0.794	large	0.008	0.500	large
6 Context coverage	<0.0005	0.808	large	0.001	0.696	large	<0.0005	0.737	large	0.007	0.527	large
7 Operability	<0.0005	0.910	large	0.001	0.696	large	0.001	0.710	large	0.003	0.585	large
8 User error protection	<0.0005	0.923	large	0.001	0.733	large	<0.0005	0.805	large	0.001	0.718	large
9 UI aesthetics	<0.0005	0.901	large	0.001	0.714	large	<0.0005	0.808	large	<0.0005	0.775	large
10Accessibility	<0.0005	0.923	large	0.001	0.720	large	<0.0005	0.857	large	0.001	0.747	large
11Analysability	<0.0005	0.846	large	<0.0005	0.785	large	<0.0005	0.869	large	0.001	0.785	large
12Changeability	<0.0005	0.876	large	0.001	0.785	large	<0.0005	0.846	large	0.001	0.792	large
13Testability	<0.0005	0.827	large	0.003	0.770	large	<0.0005	0.894	large	0.002	0.788	large
14Confidentiality	<0.0005	0.808	large	0.003	0.731	large	0.001	0.846	large	0.002	0.788	large
15Integrity	0.001	0.823	large	0.002	0.813	large	0.001	0.833	large	0.002	0.792	large
16Nonrepudiation	0.004	0.778	large	0.002	0.736	large	0.004	0.806	large	0.002	0.889	large
17Accountability	0.004	0.792	large	0.002	0.736	large	0.007	0.750	large	0.002	0.806	large
18Authenticity	0.009	0.750	large	0.001	0.917	large	0.004	0.850	large	0.001	0.900	large
19Time behaviour	0.100	0.515	large	0.003	0.909	large	0.011	0.818	large	0.006	0.879	large
20Resource utilization	0.180	0.455	large	0.019	0.909	large	0.013	0.909	large	0.026	0.864	large

Table 4.7: The results of the testing procedure for Individuals for the independent variables – p-values and the magnitude of the effect size

	Teams											
Category	**Set Completeness**			**Unamibiguity**			**Indiv. Completeness**			**Verifiability**		
	p-value	**Cliff's**		**p-value**	**Cliff's**		**p-value**	**Cliff's**		**p-value**	**Cliff's**	
		estim.	**interp.**		**estim.**	**interp.**		**estim.**	**interp.**		**estim.**	**interp.**
1 Functional correctness	<0.0005	0.857	large	0.012	0.500	large	0.272	0.292	medium	0.012	0.500	large
2 Availability	<0.0005	0.857	large	0.126	0.321	medium	0.044	0.488	large	0.377	0.095	negligible
3 Fault tolerance	<0.0005	0.857	large	0.008	0.690	large	0.100	0.369	medium	<0.0005	0.857	large
4 Recoverability	<0.0005	0.857	large	0.006	0.714	large	0.028	0.548	large	<0.0005	0.952	large
5 Freedom from risk	<0.0005	0.857	large	0.007	0.714	large	0.025	0.571	large	0.001	0.948	large
6 Context coverage	<0.0005	0.857	large	0.004	0.740	large	0.047	0.494	large	0.002	0.844	large
7 Operability	0.001	0.857	large	0.005	0.804	large	0.047	0.536	large	0.004	0.839	large
8 User error protection	0.001	0.857	large	0.003	0.898	large	0.090	0.449	large	0.004	0.857	large
9 UI aesthetics	0.003	0.857	large	0.007	0.800	large	0.075	0.543	large	0.017	0.771	large
10Accessibility	0.003	0.857	large	0.007	0.800	large	0.053	0.600	large	0.015	0.771	large
11Analysability	0.003	0.857	large	0.007	0.829	large	0.015	0.771	large	0.017	0.771	large
12Changeability	0.005	0.857	large	0.005	0.857	large	0.021	0.743	large	0.012	0.857	large
13Testability	0.005	0.857	large	0.003	0.857	large	0.007	0.857	large	0.012	0.857	large
14Confidentiality	0.005	0.857	large	0.006	0.821	large	0.018	0.821	large	0.012	0.786	large
15Integrity	0.028	0.857	large	0.028	0.857	large	0.111	0.714	large	0.111	0.714	large
16Nonrepudiation	0.028	0.857	large	0.028	0.857	large	0.111	0.714	large	0.028	0.857	large
17Accountability	0.028	0.857	large	0.028	0.857	large	0.250	0.429	large	0.028	0.857	large
18Authenticity	0.028	0.857	large	0.028	0.857	large	0.111	0.714	large	0.028	0.857	large
19Time behaviour												
20Resource utilization												

Table 4.8: The results of the testing procedure for Teams for the independent variables – p-values and the magnitude of the effect size

Chapter 5

Evolution of catalog of NFR templates

Preface
This chapter is an extended version of Section 4 of the following journal paper:

- S. Kopczyńska, J. Nawrocki, M. Ochodek, "*An Empirical Study on Catalog of Non-functional Requirement Templates: Usefulness and Maintenance Issues*", Information and Software Technology, 103, pp. 75-91, 2018.

The main extension is (1) introducing the notion of *multiple evolutions* of a catalog of NFR templates (roughly speaking, it is considering 10,000 permutations of the original evolution reported in the above mentioned paper), and (2) performing analysis of those extra evolutions to get deeper insight into the investigated phenomena (thanks to this, the obtained results are more robust).

Context. As it was already mentioned in the previous chapters, although NFR templates are recommended by experts for elicitation and specification of requirements, practitioners are afraid of using templates as they do not know what would be the return on investment of this technique. In Chapter 4, we investigated the benefits a catalog of NFR templates can provide, but the catalog was static. In many cases, however, a catalog evolves. It is subject to improvement resulting from the lessons learned from the past projects. Thus, its value is expected to be growing, but there are also some maintenance costs that must be paid.
Aim. The aim of the chapter is to investigate how the usefulness of catalog of NFR templates and its maintenance costs change over time. Do they saturate and if so, how fast is this process?
Method. Using 41 industrial projects with 2,231 NFRs, we simulated the evolution of a catalog of NFR templates. The subject of the first study was a single catalog evolution driven by the order in which the projects (i.e., requirements specifications) had been acquired. In the second study, we considered 10,000 different random permutations of the original sequence of projects to study the influence of the order of projects on the distribution of catalog value and maintenance cost over time. In both studies, we investigated how the following characteristics of the catalog change over a sequence of projects (a counterpart of elapsing time): (1) catalog value (i.e., the percentage of the NFRs of a given project that might be elicited with the use of the catalog); (2) maintenance effort (measured in the number of *add-a-template* and *modify-a-template* operations needed to incorporate lessons learned from a single project); (3) catalog utilization (it describes the percentage of templates that are used by a single project).
Results. From the performed analysis it follows that after considering about 40 projects we can expect catalog value of 75% or more and maintenance effort of 10% or less. We call such a catalog *mature*. In our studies, a mature catalog contained about 400 templates but less than 10% of them were used by a single project (on average).

Conclusions. From the perspective of a big or medium size software company the requirement of having 40 projects to get a mature catalog seems not a big problem. The more important issue seems the necessity of searching through 400 templates to get those 20 or 30 NFR templates that are truly needed. A method of identifying useful templates better than the brute–force seems very needed.
My Contribution. I designed the studies and proposed the measures to describe the maintenance process of catalog of NFR templates, I was supported by my supervisor, Jerzy Nawrocki. I prepared and executed the studies which were conducted using a software tool developed by me. Then, I analyzed the results, and next, during several workshop sessions together with Jerzy Nawrocki and Mirosław Ochodek, we formulated the conclusions.

5.1 Introduction

Non-functional requirements (NFRs) are those that state conditions under which the functionality of a system is useful (e.g., they describe how fast a system shall work, the security rules, the environments in which the system is expected to work).

Numerous surveys constantly have been showing that incomplete or changing requirements are one of the top problems challenging software projects, e.g., [223, 229]. Non-functional requirements are often neglected, especially those that are difficult to write or ostensibly obvious. That is an important risk factor, as in many cases a project failure can be traced, amongst others, to inappropriate management of them (see e.g., [22, 25, 145, 173]).

On the other hand, many experts recommend preserving the best practices in the form of templates. Templates are expressed in natural language as statements with some gaps (parameters) to be filled in and optional parts to select while formulating a requirement. They preserve correct requirements syntax (e.g., EARS [154]) as well as they can encompass some semantics of NFRs ([107, 129, 204]). During NFRs specification one can select templates from a catalog and provide the values of the parameters they define, which is called template-supported elicitation. Then, the knowledge from the past projects is reused. Some authors state that using templates improves consistency and testability of requirements, reduces ambiguity [20, 154, 240], makes elicitation and specification easier [240], and saves the effort of specification [191].

On another hand, there exists a problem with applying templates in software projects. We can hear the opinion of practitioners who are afraid of using templates (see e.g., the study by Palomares et al. [182]). They perceive them as a complex method and do not know what would be the return on investment. Thus, more evidence is needed about benefits and costs associated with NFR templates.

In Chapter 3, we investigated the benefits a catalog of NFR templates can provide. The catalogs we used in those studies were static. However, it would be entirely unrealistic to expect that a company can get or buy an ideal catalog that would fit all the projects run by it. Rather such catalog is subject to change resulting from the lessons learned from the past projects. Thus, it is practical to assume that after each project, some templates that have been found missing are added, and others are modified. As a result, the usefulness of such catalog is changing, but some maintenance costs are incurred. Dynamics of catalog characteristics is the process of change of those characteristics over time, and we are interested in the following research questions concerning the subject:

- **RQ3.1.** What is the dynamics of *value* of catalog of NFR templates for a project (i.e., the percentage of NFRs it can help to derive)?

- **RQ3.2.** What is the dynamics of *maintenance effort* catalog of NFR templates needs after a single project (i.e., the percentage of NFR templates that require to be added or modified to incorporate the lessons learned)?
- **RQ3.3.** What is the dynamics of *utilization* of catalog of NFR templates (i.e., the percentage of NFR templates that get used in a single project to derive NFRs)?
- **RQ3.4.** When can one say that a catalog is *mature* (i.e., the number of updates becomes negligible and the catalog value is high)?

The chapter is structured as follows. In Section 5.2 some definitions and terminology are provided to make the problems discussed in the chapter clear. Next, in Section 5.4, the designs of two empirical studies are reported. They are based on industry specifications and aim at the investigation of how the costs and usefulness of catalog of NFR templates change over time. The results are reported in Sections 5.5-5.8 and the validity threats are discussed in Section 5.9. In Section 5.10 we present the related work and the summary of our findings is in Section 5.11.

5.2 Terminology

As we defined in Subsection 2.2.3, a ***catalog of NFR templates*** ('catalog' for short), further denoted as K, is a finite set of NFR templates. Its ***size*** is defined as the number of templates it contains and it is denoted by $|K|$. While an organization executes a sequence of software development projects P_1, P_2, ..., its catalog evolves from K_0 to K_1, K_2, ... More precisely, lessons learned from each consecutive project P_i allow the owner to improve the templates transforming catalog from version K_{i-1} to K_i. For the sake of simplicity we assumed that each project is viewed as a finite set of its non-functional requirements, i.e., $P_i = \{r_i^1, r_i^2, \cdots r_i^{n_i}\}$. Then, the process of considering these requirements, one by one, and modifying or adding templates to the catalog whenever necessary can be viewed as ***catalog maintenance***.

Perfect match

If requirement r_i^j (or its equivalent) can be directly derived (see Subsection 2.1.5) from template t contained in catalog K_{i-1}, no maintenance action is required. One can say that there is a *perfect match* between requirement r_i^j and template t. Every such template will be included into new catalog K_i.

Template extension and indirect derivation

It can happen that no template in old catalog K_{i-1} perfectly matches requirement r_i^j. Then the template extension action might prove useful. As we defined in Subsection 2.2.3, to extend template t' one can, e.g., add some parameters and modify the static text or the options of template t' in such a way that new template t is *backward compatible* with the old one, i.e., every requirement r that perfectly matches old template t' also perfectly matches new template t.

When considering requirement r_i^j, the aim of the extension is to obtain template t such that there is a perfect match between r_i^j and new template t.

New catalog K_i will include new template t but not old template t' (as t is backward compatible with t' there is no need to keep both in K_i).

Missing template

The third situation is when requirement r_i^j (or its equivalent) cannot be directly or indirectly derived from any of the templates contained in old catalog K_{i-1}. Then, the only solution is to add new template to K_i.

Four subsets of templates

To characterize the catalog maintenance, it seems useful to split the templates of new catalog K_i into the following subsets:

- $Perfect_i$: If template t belongs to $Perfect_i$ then t belongs also to old catalog K_{i-1} and in P_i there is at least one requirement r_i^j such that there is perfect match between r_i^j and t. In this case the maintenance effort is negligible.
- $Modified_i$: Every template t of this set is an extension of a template contained in old catalog K_{i-1}, i.e., there is requirement r of project P_i that perfectly matches t and there is no template t' in old catalog K_{i-1} such that r perfectly matches t'.
- $Added_i$: If template t belongs to $Added_i$ then there is requirement r in P_i such that r perfectly matches t and there is no template t' in old catalog K_{i-1} that r could be directly or indirectly derived from t'.
- $Sleeping_i$: It contains all the templates from old catalog K_{i-1} which are not used to directly or indirectly derive any requirement of project P_i. Those templates are unnecessary for the current project, but they can prove useful in the future. The maintenance effort associated with these templates is negligible.

5.3 Description of projects

We collected and analyzed NFRs from 42 sources (see Table 5.1 for a brief description of the projects). 41 of them were software development projects—26 of which were industry projects whose stakeholders shared the data with the authors (denoted as Ind), and 15 came from the projects shared in the Open Science teraPROMISE repository [41] (denoted as Pro). Additionally, we included one literature position claimed to be based on industry projects [204]. The majority of the projects aimed at developing web applications (Web), some of them concerned desktop applications (Desktop), and one project was to deliver an integrated information system (ZSI) containing many web applications and web services. Altogether, we collected 2,231 NFRs. They concerned the development of business applications, i.e., software to perform some business functions, in different domains.

For each project, we analyzed all the statements, in some cases, they contained not only NFRs but also definitions of some terms, schedules of transitions, functional requirements, etc. (see the full list in [140]). Therefore, it was necessary to perform some data cleaning, during which we excluded the statements that did not satisfy the definition of NFR (see Subsection 2.1.4).

5.4 Catalog evolutions

The initial version of the catalog (K_0) was created based on the results of our previous study [129] and consisted of 83 templates.

This chapter contains the study of catalog evolution as presented in our recently published paper [132]. This evolution is referred to as *initial evolution*. It was directed by the order in which the 42 requirements sets have been acquired, and it is depicted in Table 5.1.

Project	#NFRs	Source	Arch.	Application Type	Domain
1	76	Lit	Multiple	Multiple	Multiple
2	55	Ind	Web	Business	Administration
3	1	Ind	Web	Business	Health Care
4	36	Ind	Desktop	Business	Services
5	24	Ind	Web	Business	Administration
6	100	Ind	Web	Business	Health Care
7	10	Ind	Desktop	Business	Services
8	48	Ind	Desktop	Business	Services
9	32	Ind	Desktop	Business	Services
10	33	Ind	Web	Business	Oil and Gas
11	68	Ind	Web	Business	Financial
12	32	Ind	Web	Business	Banking
13	41	Ind	Web	Business	Banking
14	11	Ind	Web	Business	Education
15	60	Ind	Web	Business	Research
16	89	Ind	Web	Business	Banking
17	10	Ind	Web	Business	Banking
18	43	Ind	Web	Business	Banking
19	20	Ind	Web	Business	Banking
20	108	Ind	Web	Business	Banking
21	18	Ind	Web	Business	Banking
22	139	Ind	Web	Business	Banking
23	31	Ind	Web	Business	Banking
24	37	Ind	Web	Business	Banking
25	641	Ind	ZSI	Business	Education & Financial
26	33	Ind	Web	Business	Pharmacy
27	65	Ind	Web	Business	Pharmacy
28	15	Pro	–	Business	Media
29	24	Pro	Mobile	Business	Real Estate
30	25	Pro	Web	Business	Education
31	32	Pro	Web	Business	Trade
32	37	Pro	Web	Business	Insurance
33	47	Pro	Web	Business	Services
34	11	Pro	Desktop	Business	Trade
35	72	Pro	Web	Business	Entertainment
36	11	Pro	–	Business	Services
37	14	Pro	Web	Business	Entertainment
38	13	Pro	Web	Business	Communication
39	19	Pro	–	Business	Software & Hardware
40	19	Pro	Web	Business	Services
41	15	Pro	Web	Business	Sport
42	16	Pro	Web	Business	Financial

Table 5.1: Description of the projects included in the study ("–" means that the data was not available).

Upon reflection, we have noticed that maybe the initial evolution of the catalog (i.e., the order of projects) was somehow special. Thus, the question arose what would be the results of our analysis if the order of projects was different. To answer the question, 10,000 random permutations of the projects listed in Table 5.1 were analyzed. We will refer to this

study as *multiple evolutions*. However, we decided to exclude project P_3 from the analysis since we consider it as an outlier, i.e., it contains only one NFR that can be derived from the NFR template that is used only by this project.

To investigate the *initial evolution* of the catalog, the requirements were uploaded into a web application that allowed their analysis according to the following procedure:
For each project P_i and for each requirement $r \in P_i$:

1. We manually analyzed the contents of catalog K_{i-1} to look for template t that could be used to derive r (perfect match) and if such t was found, it was included into the next version of the catalog, K_i.
2. If such a template was not found, we searched K_{i-1} for a template t that could be modified (extended) to new version t' and t' could be used to directly derive r—then t' was included into K_i.
3. Otherwise, we added a new template to catalog K_i.

All the sleeping templates of K_{i-1} (i.e., not used, directly or indirectly, by any r from P_i) were included in K_i

To study *multiple evolutions*, one can adopt *coarse-grained* or *fine-grained* analysis of the relation between a set of NFRs and a set of NFR templates. *Coarse-grained* analysis is based on the following question (the question plays the role of a characteristic function in the set theory):

> *Is it true that a requirement r (from a considered project) can be derived from template t of the final version of the catalog resulting from the initial evolution?*

The other option, the *fine-grained* analysis, is more precise. Each template is presented as a set of parts (e.g., optional parts, parameters, etc.—see Chapter 2) and each requirement is represented as being composed of those parts. The investigated relationships between the NFRs and the catalog of templates is based on the following question:

> *Is it true that a requirement r (from a considered project) needs a given part of the considered template t of the final version of the catalog resulting from the initial evolution for the derivation of r from t?*

The fine-grained analysis consumes much more effort from a researcher but it allows to observe not only additions of new templates but also their modifications (if a template t consists of parts a, b, c, so far only parts a and b have been used, and a requirement r of project P_i uses part c of the template, then project P_i leads to modification of template t). In the analysis presented in this chapter, the fine-grained analysis has been applied. The procedure was analogous to the one used in the case of the initial evolution (three steps concerning (1) perfect match, (2) if perfect match was impossible then modification, (3) otherwise adding a new template).

To analyze the dynamics of variables in multiple catalog evolutions, we used box plots (e.g., Figure 5.1). Each chart has a set of boxes each with a band inside the box which depicts the median value. The lower and upper "hinges" of the boxes correspond to the first and third quartiles, the whiskers extend from the hinge to the highest and lowest value that is within 1.5*IQR of the hinge, where IQR is the inter-quartile range (roughly speaking, the whiskers correspond to the 5^{th} and the 95^{th} percentiles [198]).

Moreover, to study how the diversity of projects influences the results of our analysis, we performed five additional analyses. For each one, we excluded some projects which we call pseudo-outliers. *Pseudo-outliers* are those projects in the pool that use the highest number of unique (specific) NFR templates (in that sense each real outlier is also a pseudo-outlier but not vice-versa). To determine pseudo-outliers first, we ranked the projects from the most specific to the least specific by determining the number of unique NFR templates (used only by a given project). Base on this ranking (see Table 5.2) we generated 5 mutations of multiple evolutions (each of size 10,000 as it was before). Each mu-

tated multiple evolution $Evol_i$ was base on 41 projects but first i projects from the ranking were replaced by copies of randomly chosen other projects from the pool.

Project	Num. of unique templates	Rank	Num. of excluded projects
6	40	2	3
20	19	2	3
22	17	2	3
2	15	3	5
11	15	3	5
25	13	4	7
15	11	4	7
1	10	5	9
4	10	5	9
24	8	6	11
5	7	9	11
10	6	7	–
13	4	6	–
18	4	8	–
39	4	10	–
31	3	14	–
33	3	9	–
35	3	7	–
8	2	11	–
14	2	16	–
26	2	13	–
27	2	14	–
7	1	19	–
9	1	12	–
12	1	8	–
16	1	10	–
17	1	18	–
23	1	16	–
28	1	15	–
34	1	18	–
37	1	17	–
41	1	12	–
19	0	20	–
21	0	20	–
29	0	20	–
30	0	20	–
36	1	20	–
38	0	20	–
40	0	13	–
42	0	20	–

Table 5.2: The ranking used to identify pseudo-outliers.

5.5 Dynamics of catalog value

The notion of value of a catalog of NFR templates has been already mentioned in Section 5.1. More precisely, value of catalog K_{i-1} is defined as the percentage of NFRs of project P_i that can be directly (perfect match) or indirectly (after extension of some templates) derived from the templates of catalog K_{i-1}. Using the subsets $Perfect_i$ and $Modified_i$ of new catalog K_i, one can define value of catalog K_{i-1} in the following way:

$$Value(i-1) = \frac{|Perfect_i \cup Modified_i|}{|P_i|} * 100\% \tag{5.1}$$

As mentioned earlier, project P_i is treated as a set of NFRs, thus $|P_i|$, i.e., its cardinality, denotes the number of NFRs of that project.

In practical terms, one can treat catalog value as the degree to which the catalog is useful as a prompt list for a given project.

Observation 5.1. *After considering about 40 projects one can expect catalog value of 75% or more.*

Justification. The above mentioned observation is supported by two observations concerning (1) initial evolution and (2) multiple evolutions (see Section 5.4).

The distribution of catalog value over time (i.e., over a sequence of projects) for the initial evolution is presented in Figure 5.1.A. From this chart, it follows that since project P_{30} for all the projects the catalog value was well above 75%. We also investigated the relationships between the time expressed as the number of projects and value of catalog. Simple linear regression models were fitted. It resulted in finding a significant regression equation with the positive slope of 1.11 (p-value = 7.94e-07), the intercept of 58.85 (p-value = 1.79e-15), and R^2 of 0.46. It indicates an increasing tendency in the data.

The distribution of catalog value for multiple evolutions (10,000 permutations of the initial evolution) is depicted in Figure 5.1.B as a box plot (a very brief explanation of the box plot representation of data is presented in Section 5.4). From the chart, it is pretty clear that for the considered set of 40 projects, independently of their order, one can expect 75% NFRs to be "covered" by templates of the catalog. Moreover, we fitted simple linear regression models for each evolution. We found significant regression equations with positive slopes—the mean value of slopes was 0.58, the median equal to 0.57, and min. and max. equal to 0.27 and 1.01, respectively (the p-value was smaller than 0.05). The intercept ranged from 60.00 to 80.00, with the mean value of 71.25 and median value of 71.27. The results indicate the increasing tendency in the data.

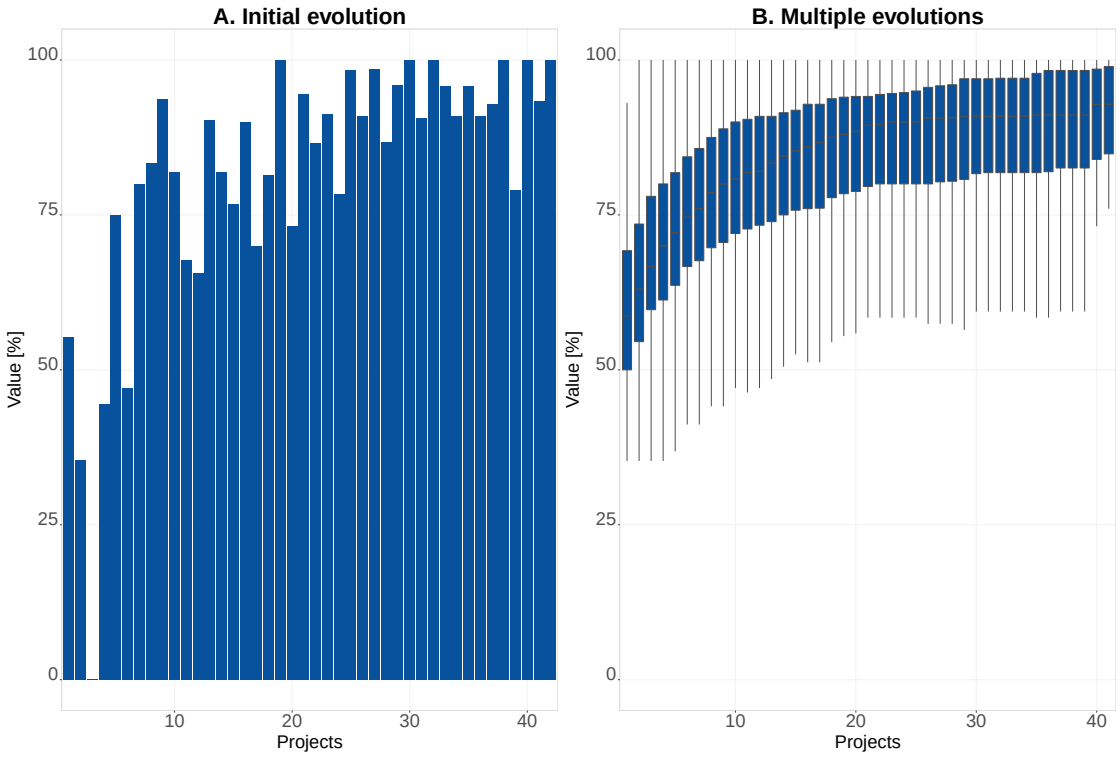


Figure 5.1: Distribution of catalog value for (A) the initial evolution and (B) multiple evolutions.

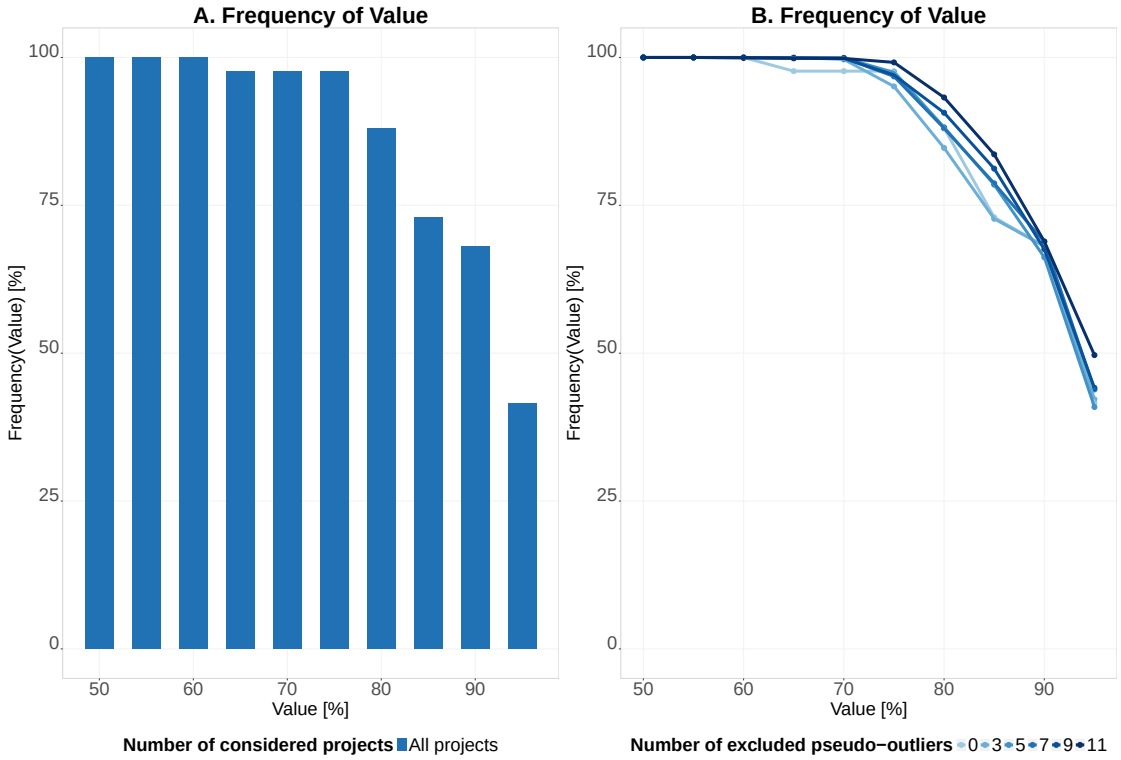


Figure 5.2: Frequency of catalog values observed over 10,000 catalog evolutions for (A) all projects and (B) after exclusion a given number of pseudo–outliers.

Our observation is also supported by another analysis. Let *Frequency*(*Value*) be a function describing the percentage of catalog evolutions for which catalog value is not less than a given *Value*. The function is depicted in Figure 5.2.A. From the chart presented in the figure, it follows that after considering all the 40 projects catalog value of at least 75% was achieved almost always. This chart also shows what is the chance of obtaining other catalog values, e.g., the value of 80% or more has been achieved in about 80% of cases (evolutions).

We have also examined the impact of pseudo-outliers on the catalog value, i.e., on the *Frequency* function (see Section 5.4 for the procedure of identifying pseudo-outliers). In Figure 5.2.B, there are several charts of the *Frequency* function for a given catalog value. Each of them corresponds to a different number of pseudo-outliers excluded from the original set of projects. The general conclusion is that the smaller the number of pseudo-outliers (i.e., projects that importantly differ from the rest of the portfolio of projects) the greater the chance of getting higher catalog value. What is perhaps more interesting, up to the catalog value of at least 75% the impact of pseudo-outliers is almost negligible. In other words, after considering about 40 projects the catalog value of 75% is very probable, even in the presence of pseudo-outlier projects.

□

5.6 Dynamics of maintenance effort

When considering maintenance of a catalog of NFR templates two operations seem the most important and time consuming: (1) adding new templates to the catalog and (2) modifying (extending) the existing ones. The former requires effort more or less proportional to the cardinality of the set $Added_i$ (i.e., the number of added templates), and for

the latter the required effort is proportional to the cardinality of $Modified_i$ which represents the number of modified templates. Thus, one can assume the following indicator of maintenance effort, $ME(i-1)$:

$$ME(i-1) = \frac{|Added_i \cup Modified_i|}{|K_i|} * 100\% \tag{5.2}$$

where K_i denotes a new version of the catalog.

Observations 5.2. *After considering about 40 projects one can expect maintenance effort, ME, to amount up to 10% of catalog size.*

Justification. The observation mentioned above is supported by two observations concerning (1) the initial evolution and (2) the multiple evolutions (see Section 5.4).

The distribution of maintenance effort over time (i.e., over a sequence of projects) for the initial evolution is presented in Figure 5.3.A. It can be observed that since project P_{25} for all the further projects, ME was less than 10%. The visible decreasing tendency of data was confirmed using simple linear regression. We found a significant regression equation with the negative slope of -0.61 (p-value = 2.15e-05), the intercept of 20.03 (p-value = 8.24e-08), and R^2 of 0.37.

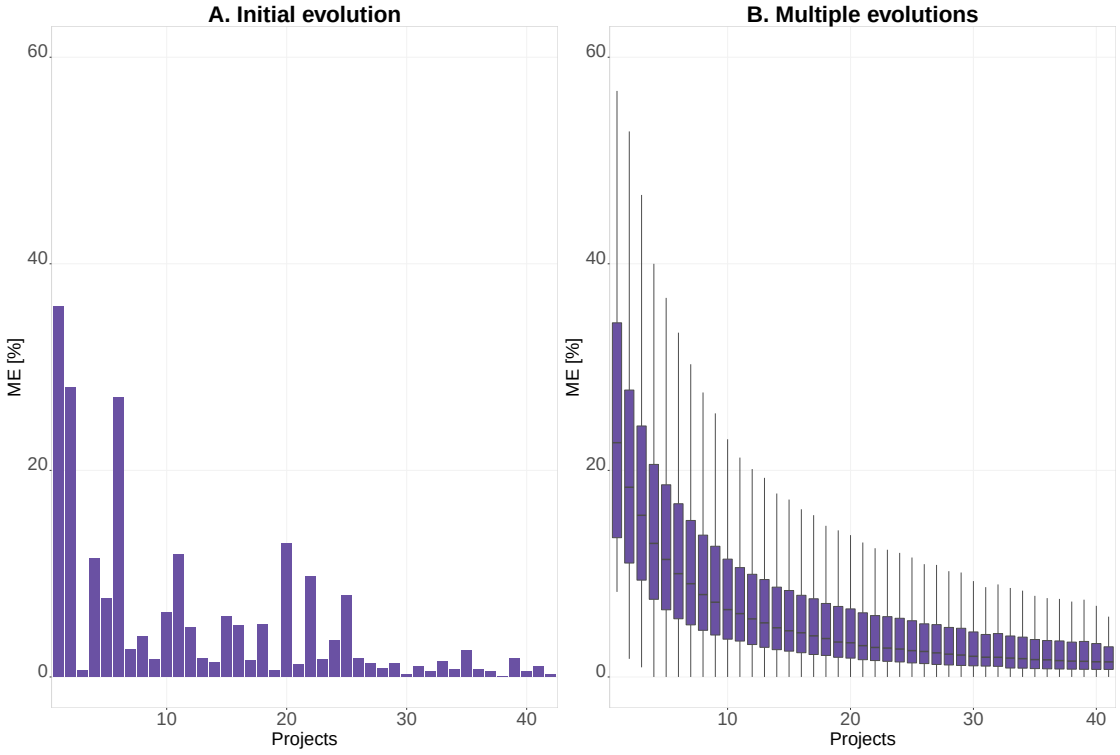


Figure 5.3: Distribution of ME for the initial evolution (A) and multiple evolutions (B).

The distribution of ME for multiple evolutions (10,000 random permutations of the initial evolution) is presented in Figure 5.3.B as a box plot (see Section 5.4 for a description of how to read this box plot). From the figure, it follows that there is a decreasing tendency in the data. The simple regression models were fitted to confirm the observation. We found significant regression equations with the negative slopes with the mean value of -0.45, median equal to -0.44, and min. and max. equal to -0.17 and -0.69, respectively (the p-value was < 0.05). The intercept ranged from 11.63 to 22.13, with the mean value of 16.95 and median value of 16.99. Moreover, from the chart in Figure 5.3.B, it is pretty

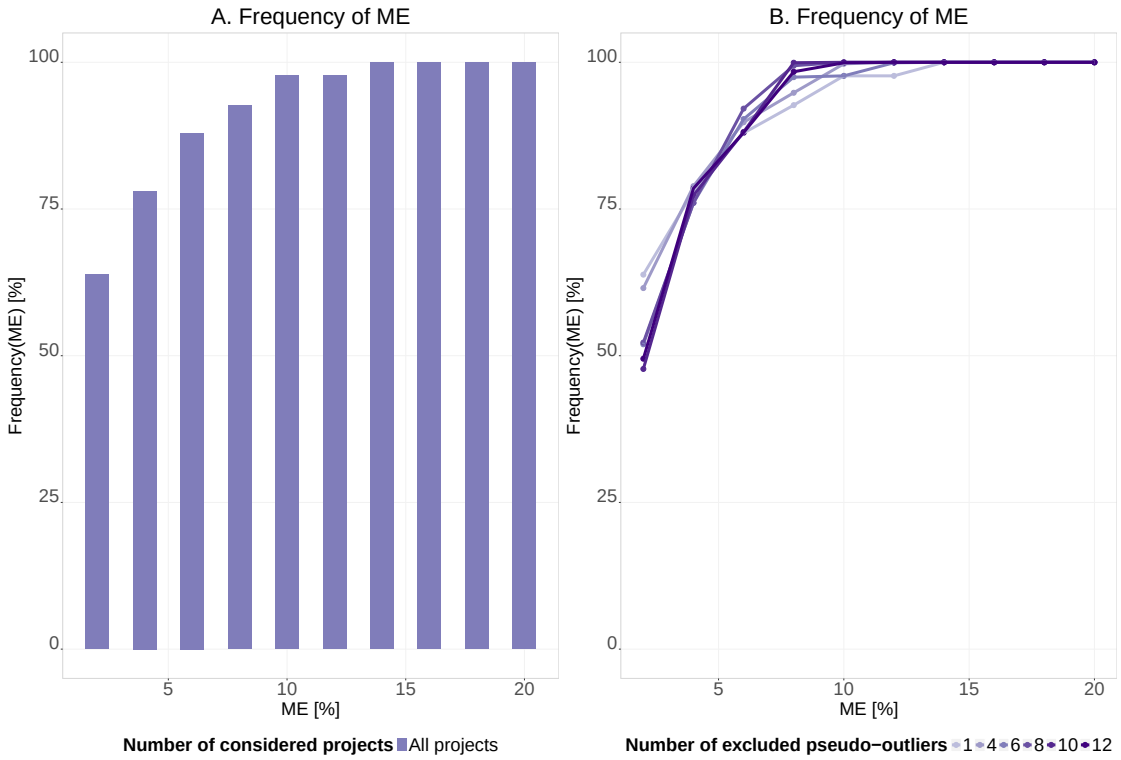


Figure 5.4: Distribution of maintenance effort, ME, for all the projects (A) and when pseudo-outlying projects got excluded from analysis (B).

visible that for the considered set of projects, independent of their order, one can expect that less than 10% of NFR templates require updates during maintenance process after a project.

Observation 5.2 is also supported by frequency analysis. Let *Frequency*(*ME*) be a function returning percentage of catalog evolutions for which maintenance effort was *ME* or less. This function is depicted in Figure 5.4.A. From the chart, it follows that after considering all the projects, maintenance effort of 10% or less was achieved in 97.7% of evolutions. The chart also presents the frequencies for other values, e.g., one might expect that there is 78% chances that the maintenance effort would be up to 5%.

We have also examined the impact of pseudo-outliers on the value of the *Frequency* function (see Section 5.4 for the procedure of identifying pseudo-outliers). In Figure 5.4.B, there are several charts of the *Frequency* function. Each of them corresponds to a different number of pseudo-outliers excluded from the original set of projects. The general conclusion is that the impact of pseudo-outliers on maintenance effort is not very big and after considering about 40 projects the maintenance effort of 10% or less is very probable, even in the presence of pseudo-outlier projects.

□

5.7 Dynamics of catalog utilization

The simplest approach to NFRs elicitation in the presence of a catalog of NFR templates is *brute force*, i.e., going from one template to another and checking if a given template could be used to formulate an NFR for the project at hand. In this context, the following question arises: what is the percentage of considered templates that will be used to specify

NFRs for a project P_i? We will refer to this percentage as *catalog utilization* and its precise definition is presented below:

$$U(i-1) = \frac{|Perfect_i \cup Modified_i|}{|K_{i-1}|} * 100\% \tag{5.3}$$

where K_{i-1} denotes the old (previous) version of the catalog.

Observation 5.3. *After considering about 40 projects one can expect catalog utilization, U, to be below 10%.*

Justification. The observation mentioned above is supported by two observations concerning (1) the initial evolution and (2) the multiple evolutions (see Section 5.4).

The distribution of catalog utilization over time (i.e., over sequence of projects) for the initial evolution is presented in Figure 5.5.A. On average, in each project, ca. 7% of NFR templates were used to derive NFRs, directly or indirectly, with the median of 5%, minimum of 0, and the maximum of 22%. This pattern seems to be constant over time.

The distribution of catalog utilization for multiple evolutions (10,000 random permutations of the initial evolution) is presented in Figure 5.5.B as a box plot (see Section 5.4 for a description of how to read this box plot). From the figure, it follows that the expected value of Utilization (median value) is below 10% for the considered set of 41 projects, independent of their order (average ca. 7%, median ca. 6%, minimum ca. 1%, maximum ca. 38%).

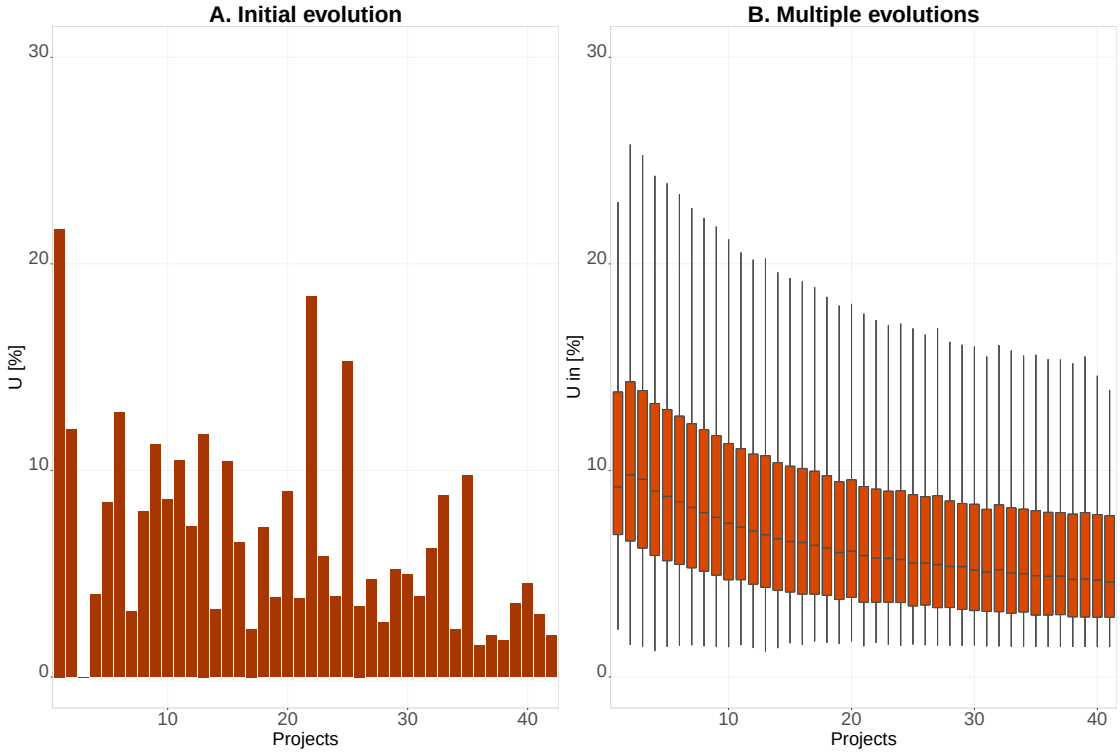


Figure 5.5: Distribution of catalog utilization for the initial evolution (A) and multiple evolutions (B).

Observation 5.3 is also supported by frequency analysis. Let *Frequency*(U) be a function describing percentage of the catalog evolutions for which catalog utilization was less or equal U. The function is depicted in (Figure 5.4.A). From the chart, it follows that after considering all the projects, catalog utilization of 10% or less was achieved in 93.7% of

evolutions. The chart also presents the frequencies for other values, e.g., the utilization of 20% or less was practically in all the observed evolutions of the catalog.

We have also examined the impact of pseudo-outliers on the value of the *Frequency* function (see Section 5.4 for the procedure of identifying pseudo-outliers). In Figure 5.4.B, there are several charts of the *Frequency* function. Each of them corresponds to a different number of pseudo-outliers excluded from the original set of projects. The general conclusion is that the smaller the number of pseudo-outliers (i.e., the projects that differ from each other) the greater the chance of getting higher utilization.

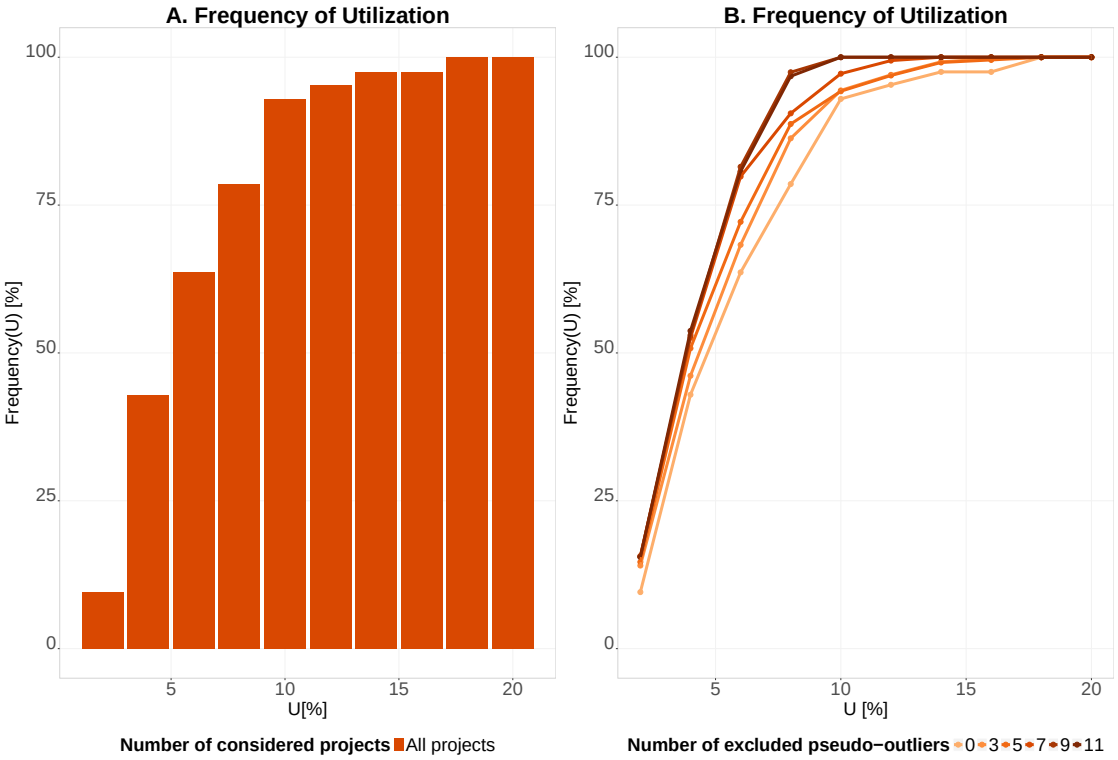


Figure 5.6: The *Frequency*(*U*) function for all the projects (A) and when pseudo-outlying projects got excluded from analysis (B).

Catalog utilization below 10% seems too low from the perspective of elicitors, especially in the context of catalogs containing about 400 templates. From Figure 5.5.B it follows that after considering about 40 projects it is quite probable that the number of templates useful for a particular project will be 20 or even less. It resembles looking for a needle in a haystack. This issue is better illustrated in Figure 5.7 which presents the proportion between the number of NFR templates that got used in each project (dark gray bars) and the catalog size (light gray bars) for the initial evolution. Thus, it would be worth to have a method of searching for NFR templates faster than brute force performed manually.

□

5.8 Maturity of catalog

As it has been observed in Section 5.5, as more and more projects are considered, the catalog value gets saturated. Similar behavior has been observed in Section 5.6 for maintenance effort (but in the opposite direction). Thus, a question arises when a given catalog could be considered *mature*, that is how to characterize a moment in its development

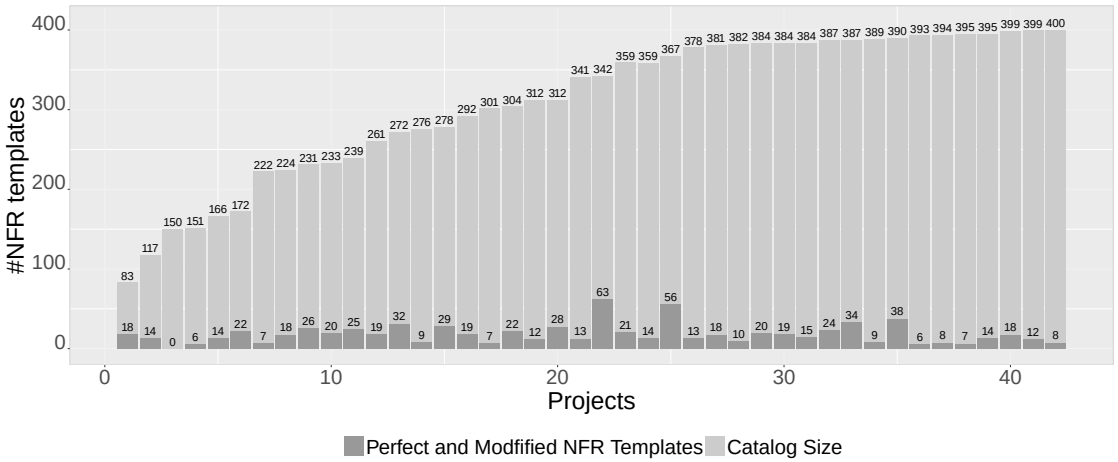


Figure 5.7: The distribution of the number of used NFRs and the catalog size for the initial catalog evolution. (The figure is taken from [132].)

since which (1) its value is high enough, and (2) maintenance effort is low enough (negligible). There are many ways of defining catalog maturity. In our view it is reasonable to assume the following definition:

Definition 5.1 A catalog of NFR templates is mature when its value is 75% or more and its maintenance effort, *ME*, is not greater than 10%.

Observation 5.4. *About 40 projects should suffice to obtain a catalog of NFR templates which is mature.*

Justification. This observation directly follows from Observation 5.1 and Observation 5.2. □

5.9 Validity threats

In the following paragraphs, we discussed the threats to the validity of the study according to the guidelines by Wohlin et al. [241].

5.9.1 Conclusion validity

Reliability of measures. To minimize the influence of inherent ambiguity of natural language that might have been introduced during the identification of the need of improvement of the catalog, the detailed procedure and definitions were created and discussed beforehand.

While conducting the initial catalog evolution as well as while preparing the input data to the fine-grained analysis of multiple catalog evolutions we could have introduced some errors, e.g., while identifying parts of templates or identifying which part of a given template is present in a given requirement. Next, it shall be taken into account that in the multiple evolutions we used computer programs to simulate multiple maintenance processes, which might have contained some errors. To mitigate the threats concerning the multiple evolutions we conducted simulation using the coarse-grained approach that was based only on the relationships between templates and requirements identified in the initial catalog simulation. In Appendix B we compared the results obtained using both the fine- and coarse-grained approaches. Based on these results, we argue, that our observations should not be visibly affected by the mentioned threats.

Fishing. To minimize the threat that the experimenters would search for a specific result when analyzing the data, the researchers did not see any information on a project, the order number of each requirement, and values of the investigated variables.

5.9.2 Internal validity

Selection. The minimum requirement towards all NFRs' sets was that they contain requirements from industrial projects and most likely developing web business applications. Although we selected the source organizations by convenience, they represent a quite broad range of possible cases. Additionally, the NFRs obtained by the authors were combined with the publicly available sets [41].

Another threat relates to the homogeneity of projects (and NFRs). Since they were taken from different organizations, the abstraction levels vary, which might have boosted the size of the catalog, e.g., some people specify only that the Web Content Accessibility Guidelines (WCAG) shall be satisfied, while the others, instead, list concrete guidelines.

5.9.3 Construct validity

Design threat. The researchers did not participate in the projects for which they analyzed NFRs. Although they do have more than 4 years of experience in RE, with a focus on NFRs, they might have misinterpreted some parts of requirements.

Moreover, the analyzed NFRs satisfy the provided NFR definition, and we decided to use NFR templates as defined in Section 2.1. Since one might use other existing definition of an NFR, their results might vary from ours. Moreover, using an approach that requires from an NFR to be documented with more extensive information, e.g., Gilb's Planguage [82] that suggests that each requirement shall have a scale, measure or authority, might also drive to other results.

5.9.4 External validity

Interaction of setting and treatment. In our study, we mimicked the behavior of an organization that creates, uses and maintains its catalog of templates over time. The catalog maintenance procedure was also aligned with the published industry practices that show the steps towards systematic requirements reuse executed in Rolls Royce [141]. Therefore, we perceive the settings as realistic-enough to generalize the conclusions.

The order in which we analyzed the projects in the initial catalog evolution was determined by the time we were able to access the data. As a result, the analysis presents one of the possible evolutions—a sequence of how the catalog could evolve. Then, we conducted a simulation study to investigate different scenarios of how a catalog is maintained over time to limit the impact of the order of the projects on the observations regarding maintenance effort, catalog value, and utilization.

Although our goal was to provide analysis independent of the domain, type of application, we evaluated only 42 sets of requirements. Therefore, we need to accept the threat that the conclusions might be true only for the analyzed domains and types of applications.

5.10 Related work

Patterns, templates, and boilerplates. Few methods and approaches utilize the knowledge from previous projects to assure the quality of requirements proposed by the requirements engineering community. One of them is using patterns. Patterns structurally de-

scribe the problem they solve, the context they are to be used in, and the solution itself. They were proposed for both functional requirements (e.g., [3, 175]) and non-functional requirements. For NFRs there exist few catalogs of patterns. Some of them are published and available (or partly available), e.g., for natural language requirements: the Withall's 37 NFRs patterns [240], PABRE patterns [201], or for the goal-based requirements notation: the NFR Pattern approach [227] an extension to NFRFramework [39].

Another approach is to use templates also known as boilerplates, like our NFR templates. They are sentences or statements with some parameters. Often they constitute part of patterns (they constitute the solutions that the patterns propose), but they also exist solely as catalogs. Templates might capture knowledge on different levels of abstraction. For example, those by Denger et al. [59] are sentence parts such as events, conditions, exceptions, etc. which could be combined with sentence patterns to describe complete requirement statements. Similarly, the EARS boilerplates are constructed that are available in the papers [154] . The mentioned approaches also concern functional requirements and are reported to be effective in practice. Then, there were proposed templates that capture not only the sentence structure but also some knowledge of particular area such as security [75, 203] or performance [69], and those general ones, e.g., [107].

What is a bit surprising, *none* of the found studies investigated the problems concerning evolution of a catalog of NFR templates (or patterns), and change of its value and maintenance costs over time (in the study reported in this chapter, time is represented by a sequence of projects run by a software organization).

5.11 Summary

In this chapter, two empirical studies are reported. Both concerned evolution of a catalog of NFR templates and the focus was on catalog value, maintenance effort required by catalog, and its utilization by a single project. In the first study, a single catalog evolution driven by the order in which the projects had been acquired was analyzed. In the second study, we considered 10,000 different random permutations of the initial sequence of projects to investigate the influence of the order of projects on the distribution of the investigated catalog characteristics. In our analyses, we used NFRs coming from software development projects (26 industry projects whose stakeholders shared the data with us and 15 projects shared in the Open Science tera-PROMISE repository [41]; Section 5.4 contains all the details).

Here are the observations that follow from our study:

- (*Observation 5.1.*) After considering about 40 projects one can expect catalog value of 75% or more.
- (*Observation 5.2.*) After considering about 40 projects one can expect maintenance effort (measured in number of updates) amounting up to 10% of catalog size.
- (*Observation 5.3.*) After considering about 40 projects one can expect catalog utilization to be below 10%.

We expected that after some time (i.e., after considering a big enough number of projects) the maintenance effort will decrease to a small level and remain at it. To characterize this, we introduced the notion of *maturity* of a catalog. We proposed to call a catalog mature when its value is 75% or more and its maintenance effort, *ME*, is not higher than 10%. From the presented observations it follows that about 40 projects are needed to make a catalog mature. When our catalog became mature, it contained about 400 NFR templates.

Since a mature catalog can have 400 templates (or more), the following question arises: *what is the fastest method of searching through a mature catalog of NFR templates?* A naive approach based on going through the templates one-by-one seems too slow and can hin-

der the usability of the catalog. Thus, more advanced approaches to support catalog browsing might prove useful, such as those based on ontology or machine learning techniques (e.g., the one planned by Guzman et. al [92]).

Chapter 6

App stores as a source of NFRs and their templates

Preface
This chapter has been previously published as the following paper:

- E.C. Groen, S. Kopczyńska, M.P. Hauer, T.D. Krafft, J. Doerr, "*Users—The Hidden Software Product Quality Experts?: A Study on How App Users Report Quality Aspects in Online Reviews*", 25th IEEE International Requirements Engineering Conference (RE), 80-89, 2017.

Context. The research presented in the previous chapters was based on rather classical methods of requirements elicitation and NFR templates identification such as workshops (e.g. SENoR), prompt lists (e.g., ISO 25010 list of quality subcharacteristics), or analysis of requirements from past projects. Recently, user feedback from online stores has been identified as a promising source of requirements. As a consequence, user feedback can also be a source of requirement templates that follow from them. The up-to-date research in this area has focused on functional aspects. The question arises to what extent user feedback addresses issues related to the non-functional ones?
Aim. The aim of the chapter is to check whether user reviews in app stores contain information concerning NFRs.
Method. We conducted two studies. First, we carried out a manual tagging of online reviews to determine what information concerning NFRs user reviews contain. We analyzed 1,385 statements from 3 different app stores about 6 applications. That allowed to build an automatic review classifier capable of selecting reviews concerning non-functional aspects. Then, we used the classifier to search through opinions about an extended number of applications.
Results. We found that users wrote mainly about usability, reliability, and portability. There were differences in length and focus of user reviews among different app stores. Usability was the only quality characteristic that users reported on both positive and negative aspects in nearly equal numbers. Some opinions were rather short and general (just an adjective phrase), while others were very detailed.

A small set of 16 language patterns was enough to identify 1,528 statements regarding usability.
Conclusions. Reviews in on-line app stores contain interesting information that could be used as a source of NFRs and their templates. That information can be filtered out automatically.
My Contribution. After proposing the idea, I designed the study together with Eduard Groen. I developed a software tool that would support manual analysis of the reviews.

Then, I analyzed all the reviews together with Eduard Groen (it was manual tagging of the statements). Moreover, I analyzed the results of tagging and participated in the formulation of the conclusions.

6.1 Introduction

The requirements engineering (RE) community increasingly sees user feedback on online platforms as a potential source of user requirements. These platforms include app stores (e.g., those of Samsung, BlackBerry [212], Google, and Apple [91, 179, 185]), social media (e.g., Twitter [90]) or issue tracking systems [158]. Manual tagging studies have found user reviews to provide useful information on product features when classified using content categories such as feature requests [91, 90] and general criticism/praise [179, 148]. However, manual tagging is a tedious and time consuming task. Thus, approaches to automating such analyses are being researched under the name "Crowd-Based RE" [88]. One method, which is practical and powerful at the same time, employs language patterns involving regular expressions to match sentences with the specified pattern through automation.

Typically, requirements are distinguished into three categories: functional requirements (FRs), constraints, and quality requirements or non-functional requirements (NFRs) [191]. With respect to the first category, existing FRs can be refined by analyzing user review on criticism and praise, while new FRs can be derived from feature requests [91]. Requirements of the second category, constraints, affect and limit projects and software solutions, and are often beyond the user's awareness or explanation capabilities. For example, a user will usually not know about the available resources such as budget and manpower, and will not always understand how laws and standards affect a piece of software. Hence, there is little chance that user feedback provides useful information on constrains, if these are addressed at all. Yet user feedback on the third category, NFRs, may be of interest. Since users are directly affected by quality characteristics such as usability, performance efficiency, and security, it is probable that user reviews contain statements about product qualities. However, to the best of our knowledge, no research has yet classified statements according to non-functional aspects.

Research on NFRs is highly relevant, as they are a crucial aspect of specifying and implementing a software. These requirements are the typical architectural drivers [8], and not adequately addressing them will likely lead to project failure and high rework cost [52] (see Section 6.5). They also increasingly form a distinguishing factor (i.e., a unique selling proposition), especially for consumer products. But getting a complete and correct set of NFRs is difficult [38, 63]. Some studies have considered bug reports (e.g., [90, 158, 148]), but their outcomes were used to identify feature shortcomings at the functional level, and at best they discovered keywords such as "crash" to obtain an indication of poor reliability.

Thus, in this research, we set out to determine whether user feedback can also be a useful source of statements that can help with NFRs elicitation or prioritization. We formulated the following research questions:

RQ6.1. What quality aspects are raised by users?

RQ6.2. How should language patterns be defined so that automation can uncover quality aspects formulated by users?

To answer these questions, we first conducted a study where we manually tagged user reviews for statements about product qualities, which is presented in Section 6.2. Then we performed a study where we performed an automated analysis of user reviews, which is presented in Section 6.3. In Section 6.3.3, we discuss the results of these studies, followed

by an analysis of the threats to validity in Section 6.4. Section 6.5 presents related work, and Section 6.6 our conclusions.

6.2 Study I – Investigation of User Reviews

6.2.1 Method

In order to understand what users say about the quality of products, we set out to identify statements on product quality by tagging them through content analysis [172].

Step 1: Select and obtain user reviews. Following the finding of Pagano and Maalej that interactive app categories garner more reviews [179], we looked for products in the five categories they identified, and we added the category "Smart Products", referring to products where the owner of such a "smart" physical product can download a free app to unlock further functionalities. As selection criteria, we only chose apps that are available on three app stores (Apple App Store, Google Play, and Amazon Appstore), and products whose overall rating was neither too positive nor too negative. In each product category, we selected two apps – one paid app and a free app, as user reviews of free and paid apps have also been found to differ [179]. This resulted in a test set from 36 sources ($6\ categories \times 2\ apps \times 3\ app\ stores$), as shown in Table 6.1. The user reviews of these products were then crawled using a customized text-mining tool.

Category	App (Price)	No. crawled reviews		
		Am	Ap	Go
Entertainment	Disney Movies Anywhere	120	2,359	4,217
	Cleverbot ($0.99)	59	1,530	800
Productivity	OneNote	422	6,722	2,959
	Tiny Scan Pro ($4.99)	432	2,140	601
Social Media	Tweet Caster	1,200	7,758	4,050*
	Tweet Caster Pro ($4.00)	857	724	4,023*
Messaging	Viber	889	67,671	4,306*
	IM+ Pro ($2.99)	132	193	2,215
Games	Endless Arcade	698	2,261	4,402*
	Maze Sonic & SEGA All Stars Racing ($2.99)	302	3,548	420
Smart Products	Philips Hue	69	2,469	601
	August Smart Lock	344	290	283
	Total	*5,602*	*97,715*	*28,877*

Table 6.1: List of apps in our dataset with the number of reviews crawled from three app stores. An asterisk (*) indicates that the app store limited the number of retrievable reviews (Am = Amazon Appstore, Ap = Apple App Store, Go = Google Play).

Step 2: Structure user reviews. We took a random sample from the dataset. As each review contains a rating ranging from one star to five stars, we stratified our sample by randomly selecting two reviews for each rating, resulting in ten reviews per product per store, or 360 reviews in total. For Cleverbot on Amazon, our dataset included only one 4-star review, and was compensated by adding a randomly selected review from the most frequent adjacent rating, which was a 5-star review. Using OpenNLP, we then split these reviews into separate sentences (which we call "statements") for a total of 1,385 statements. The

statements were exported to a CSV file, which we loaded into a customized web-based tagging tool to enable the annotators to tag statements independently of one another.

Step 3: Define tagging categories and a tagging schema. Before the annotators could assign "tags" (categorizations) in the tagging tool, we had to determine these tags. As we are interested in statements about product qualities (i.e., non-functional aspects), we decided to use the structure of the quality model for "software product quality" proposed in the ISO 25010 standard[113], a commonly used quality standard. This quality model contains eight software product characteristics, which became our "main tag". These are in turn subdivided into 31 subcharacteristics, which became our "sub tags". For example, the characteristic "compatibility" has the two subcharacteristics "co-existence" and "interoperability". To the "main tag", we added the option "none" to tag statements that are not about quality or which are too unclear. To get a common understanding of the meaning of the tags, we performed two test runs and discussed the process and the results, based on which we iteratively composed a tagging schema with the definitions from ISO 25010, along with examples and signal words. The annotators were additionally trained on the ISO 25010 characteristics and subcharacteristics.

Step 4: Perform tagging. Five annotators tagged the statements separately from each other. This was done by selecting and reading each statement, and then assigning the best fitting main tag. The sub tag was optional, and was only assigned if the statement could clearly be assigned to a particular subcharacteristic. Additionally, the annotators were allowed to provide more than one tag (and their associated sub tags) if a statement could clearly be assigned to more than one characteristic. Cases where this happened included, among others, statements that had independent clauses (e.g., separated by "but" or "and"), that lacked punctuation so that the sentence splitter failed to split them, or that simply covered multiple aspects.

Step 5: Reconcile and process tagging results. After completing the tagging of the entire sample, one annotator exported the tagged statements to an Excel file. If at least two senior annotators agreed, a statement was assigned that tag. Statements that could not be unambiguously assigned one main tag were reconciled by two annotators by analyzing each of these statements, considering the tags provided, and together proposing the definite main and sub tag(s). Based on this process, we built up our ground truth.

Step 6: Analyze data. We analyzed the data resulting from the content analysis by calculating sums and frequencies, making comparisons per app (between app stores), per grouping (between categories), by price (paid vs. free) and by rating (1 to 5).

6.2.2 Results

We analyzed 360 reviews consisting of 1.368 statements, which took each annotator approximately 10 hours to tag. A total of 163 (45.3%) reviews were tagged as containing information on quality characteristics (see Table 6.2). They were made of 263 statements (19.0% of all statements), with two statements on average (i.e., the title and one sentence from the review text), and ranging from only a title to a review with a title and three sentences. Most statements were tagged with only one main characteristic and one subcharacteristic; 32 reviews (36 statements) were assigned two main characteristics, 4 reviews (4 statements) were assigned three main characteristics, and 9 reviews (9 statements) were assigned two subcharacteristics within the same main characteristic.

Frequency of characteristics

Three characteristics were most frequently reported on by users: "usability", "reliability", and "portability" (see Table 6.2). We found 35.0% of all quality-related reviews (R), repre-

senting 28.5% of all quality-related statements (S)[1] to be "usability"-related. Within "usability", the subcharacteristic "operability" was most often discussed (R: 19.6%; S: 16%), followed by UI aesthetics (R: 7.9%; S: 5.3%), and learnability (R: 4.3%; S: 3.0%). We identified 4 reviews (10 statements) on "usability" in general (e.g., "the most intuitive app"), and zero on "appropriate recognizability", "user error protection" or "accessibility".

"Reliability" was identified as the second most frequent characteristic in 20.2% of reviews and 28.9% of sentences. Users predominantly addressed issues regarding "fault tolerance" (R: 19.6%; S: 15.6%), while "availability" (R: 2.4%; S: 1.9%) and "recoverability" (R: 4.2%; S: 3.0%) were addressed much less often, and "maturity" was never addressed. Interestingly, for every product, at least one review addressed "reliability", though there are only 9 reviews that regard the two most frequent characteristics "usability" and "reliability".

The third most frequently found characteristic was "portability" (R: 29.4%; S: 22.8%). The majority of the statements were about "adaptability" (R: 23.3%; S: 14.4%), which gave an idea about the environment (e.g., operating systems, Internet browsers) in which users used or would like to use the app. Fewer statements discussed the apps' "installability" (R: 3.8%; S: 4.2%) or issues with upgrading as a subset of "replaceability" (R: 0.7%; S: 1.1%).

For other characteristics, we will limit the discussion to unusual findings. We concluded that we could not distinguish well between the subcharacteristics of "security" (R: 5.7%; S: 7.2%), because in most cases they already determine the cause or culprit. Without understanding the background of the issue, one would have to guess which subcharacteristic could apply. For example, the statement "Can't even get in" may suggest either an "authentication" or an "authorization" issue. We furthermore did not identify any statement about the characteristic "maintainability" in our test set.

Comparison of app stores

Amazon provided the longest reviews with 480 statements, of which 20.4% (R: 62; S: 98) contained statements on quality aspects. Apple had slightly shorter reviews, resulting in 465 statements, of which 25.8% (R: 66; S: 120) were on quality aspects, though the reviews that address quality aspects were longer on average. Google had the shortest reviews, amounting to a total of 413 statements, of which only 10.9% (R: 35; S: 45) contained quality-related statements (see also Table 6.2). Interestingly though, the reviews from Google provided a nearly even distribution of the reviews across characteristics (avg. 5, median 7, min. 3, max. 11). We also found differences in the emphasis on quality characteristics. The most frequently addressed quality with Apple was "reliability" (R: 30; S: 45) followed by "usability" (R: 28; S: 28) and "portability" (R: 20; S: 29); the reviews on Amazon focused on "usability" (R: 29; S: 36) and "portability" (R: 21; S: 25), and those from Google mainly discussed "reliability" (R: 11; S: 15), followed by "usability" (R: 7; S: 9), "portability" (R: 7; S: 8), and "compatibility" (R:7; S:7).

Comparison of apps

We see that OneNote received a fairly average total number of statements (119 from all three app stores together), but the most quality related reviews with 80.0% (R:24; S:47/119), followed by TweetCaster with 56.7% (R:17; S:20/128) and IM Pro with 53.3% (R:16; S:27/129). August Smart Lock received the longest reviews overall with 209 statements, but only 8.6% of these (R:13; S:18/209) were about quality. CleverBot had the lowest share of quality-related statements with 3.1% (R: 3; S:3/97) (see Figure 6.1). The three most frequently

[1] In the following paragraphs we will report data about both the number and the percentage of reviews and statements. For the sake of brevity, we denote reviews by R, and statements by S.

identified quality characteristics overall are usually also the most often found quality characteristic per app. One exception is IM+ Pro, for which "compatibility" was mentioned most frequently (R:8), as users reported on the advantage of interoperability with many instant messaging services, or some unsuccessful connections between them.

Comparison of app categories

The product category "productivity" (see Table 6.1) had the highest share of the quality-related reviews and statements (R: 39; S: 65/202, 32.2%), by which it accounts for 24.0% of all reviews and 25.0% of statements that are quality-related. The lowest-scoring category was "entertainment" (R: 17; S: 27/181, 14.9%). Although in this category, Disney Movies Anywhere received a higher share of matched statements, CleverBot provided very few matches, causing a low overall score for this category. Paid apps received longer reviews on average, though these but fewer reviews and fewer statements mentioned quality aspects (R: 61, 37.0%; S: 115/724, 15.9%) compared to free apps (R: 102, 63.0%; S: 145/644; 22.5%). The distribution of reviews and statements across subcharacteristics was similar to the patterns observed earlier, except that free apps had a larger relative percentage of reviews and statements on "security" (R: 7.9% vs. 1.2%; S: 7.9% vs. 3.7%), whereas paid apps had a higher share of statements (but not reviews) on "performance efficiency" (13.1% vs. 6.7%) and "functional suitability" (8.0% vs. 2.8%).

Comparison of ratings

Figure 6.2 shows the distribution of the number of reviews per rating. With respect to "reliability", reports on the app crashing resulted in low ratings (1 or 2 stars). Reviews regarding "usability" could provide both positive feedback (e.g., "easy to set up lists") and negative feedback (e.g., "it is difficult to navigate").

6.2.3 Discussion

User feedback vs. quality subcharacteristics

In this study, we were able to ***identify which quality aspects users most often mention in online reviews about apps***. Although the statements of users often contained ambiguities, which made the tagging results not as clear-cut, we did find quite a consistent pattern in the frequency with which product quality are reported, in wich particular product quality characteristics and their subcharacteristics were addressed more frequently than others. This pattern was consistent across apps, app categories, app stores, and price groups. ***"Reliability" and "usability" were found most often***, and they together accounted for over half of all reviews and statements on quality, though users rarely address both characteristics in one review. "Maintainability" received zero statements. Thus, it appears that users will primarily report on qualities by which they are directly affected (either positively or negatively), while on the other hand, they do not, and in many cases cannot, contribute useful input on the maintainability of a product. This means that users are experts in their field, providing reliable firsthand reports of experiences with the app. However, it also means that user feedback will not provide information on other (sub)characteristics, so that mining online reviews may be less suited for helping to determine how well those quality aspects are being fulfilled.

Every characteristic also had at least one subcharacteristic that was either never mentioned, or in the case of "compatibility" and "portability", was mentioned in one and two reviews, respectively. Thus, both between and within characteristics, there are strong and systematic differences between aspects that users focus on. Moreover, ***statements are often***

Characteristics and subcharacteristics	No. of reviews; no. of statements that are quality-related within characteristic			
	Total	**Am**	**Ap**	**Go**
Usability	**57; 75**	**29; 38**	**21; 28**	**7; 9**
Operability	32; 42	15; 22	15; 18	2; 2
UI Aesthetics	13; 14	2; 2	9; 10	2; 2
Learnability	7; 8	7; 8	0; 0	0; 0
General	14; 15	9; 9	1; 1	4; 5
Reliability	**53; 76**	**12; 16**	**30; 45**	**11; 15**
Fault tolerance	32; 41	8; 9	18; 24	6; 8
Recoverability	7; 8	1; 1	4; 4	2; 3
Availability	4; 5	0; 0	3; 4	1;1
General	22; 50	4; 7	15; 17	3; 3
Portability	**48; 60**	**21; 24**	**20; 29**	**7; 7**
Adaptability	38; 38	14; 14	19; 19	5; 5
Installability	10; 10	7; 7	3; 3	0; 0
Replaceability	2; 3	0; 0	2; 3	0; 0
General	9; 9	3; 3	4; 4	2; 2
Compatibility	**27; 31**	**11; 11**	**9; 13**	**7; 9**
Interoperability	22; 26	10; 10	8; 12	4; 4
Co-existence	1; 1	0; 0	0; 0	1; 1
General	4; 4	1; 1	1; 1	2; 2
Performance efficiency	**21; 30**	**7; 10**	**9; 14**	**5; 6**
Time behavior	14; 17	4; 4	5; 7	5; 6
Resource utilization	4; 8	2; 5	2; 3	0; 0
General	5; 5	1; 1	4; 4	0; 0
Security	**15; 19**	**5; 5**	**5; 6**	**5; 8**
Functional suitability	**14; 16**	**1; 1**	**10; 12**	**3; 3**
Functional correctness	9; 9	1; 1	6; 6	2; 2
General	6; 7	0; 0	5; 6	1; 1
	Total	*62; 98*	*66; 120*	*35; 45*

Table 6.2: Number of quality-related reviews and sentences per quality characteristic in total and by app store (Am = Amazon Appstore, Ap = Apple App Store, Go = Google Play). Totals differ from summed values as multiple tags can be assigned to one review.

best considered on a subcharacteristic level, though users do not address all subcharacteristics. Although we only assigned a subcharacteristic if a statement could be clearly assigned to one, every characteristic had ***one predominant subcharacteristic*** that accounted for over half of all tags of that characteristic. The four most common ones we found are "operability", "adaptability", "fault tolerance" and "interoperability". The only exceptions to this finding are that "maintainability" was never addressed at all, and that for "security", it was not possible to assign subcharacteristics without knowing, for example, what caused a particular login problem.

What we can learn from user feedback?

"Usability" is the only quality characteristic that users report on both ***positively and negatively in nearly equal numbers*** (the difference in the percentage of the reviews rating 1–2 vs. 4–5 stars is approx. 10%). Reports on "operability" either confirm or deny ease of use, whereas a lack of such statements might indicate improvement potential. Online reviews

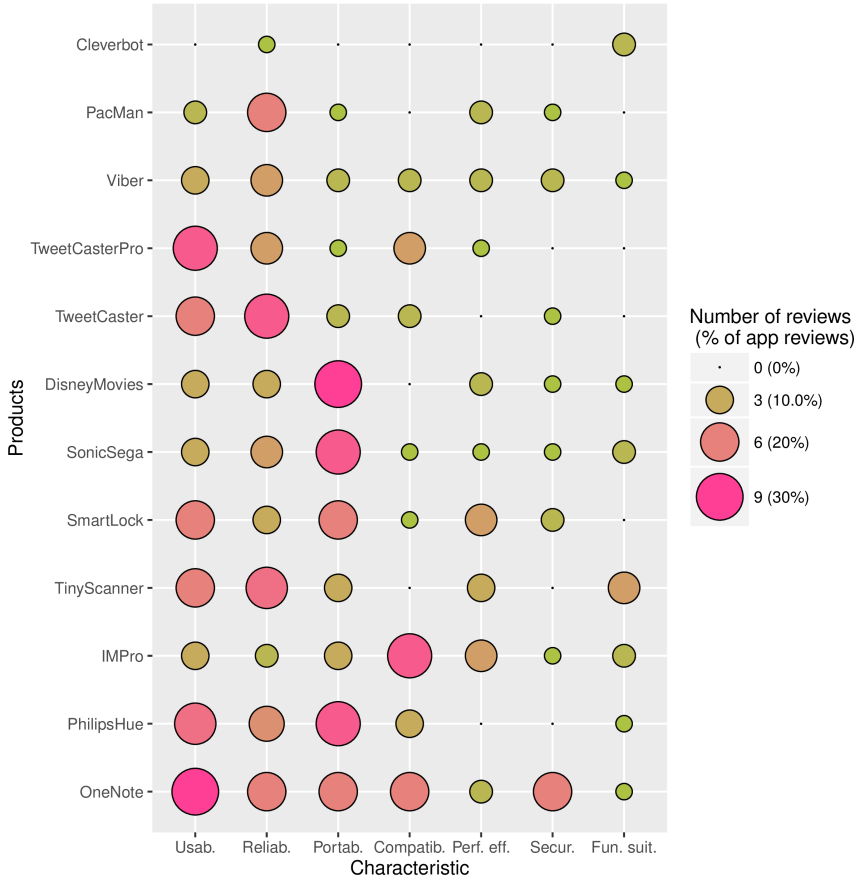


Figure 6.1: Number of quality-related reviews for apps vs. quality characteristics. (Figure taken from [87].)

may additionally inform among others UX designers about the elements that can spoil a positive user experience (e.g., “way to export documents isn’t terribly obvious”, or “Would you please provide us with the ability to have the links inside of the tweet highlighted”).

Statements about “reliability” and its subcharacteristic “fault tolerance” are almost *exclusively negative* and indicative of an error occurring. Such reviews seem to be a good source of test cases (e.g., by providing attributes of the testing environment) and bugs to be improved. This may also associate our findings with research on using bug reports for RE [158]. Most apps appear to have at least one situation in which faults are handled improperly (e.g., crashing without informing the user of what happened; not supporting the user in finding a workaround or in reporting bugs). These kinds of reviews should be monitored with care, as they seem to have the greatest overall influence on an app’s rating, followed by reviews on “performance efficiency”. The latter can help to propose new or refine existing NFRs on the one hand, and, on the other hand, help to create acceptance test cases for functions that are expected to finish faster or smoother. This seems to be especially important for apps that users expect to work efficiently, such as instant messengers or games. Statements on “adaptability” could be useful for deriving the environments (operating systems, devices) in which the app is used or in which users would like to use them. “Interoperability” provides insights into the need for or the ability to transmit data or commands across devices, or to identify which services should cooperate. We also see similar potential for formulating or improving NFRs for other (sub)characteristics, although the information available from user feedback in online reviews will be more limited.

Our results are in line with the findings of Pagano and Maalej [179], whose statistical

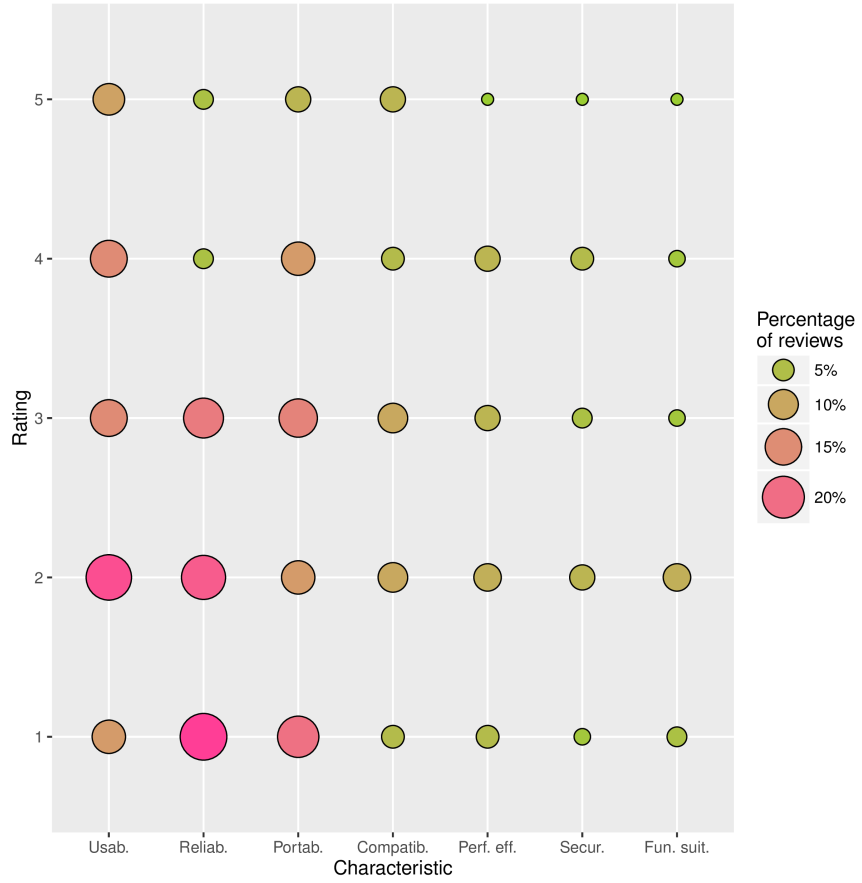


Figure 6.2: Percentage of quality related reviews for ratings vs. quality characteristics. Figure taken from [87].)

analyses revealed that users provide more RE-relevant user feedback for apps with which they interact and build a form of relationship. The deviating pattern of comments on Cleverbot reveals that the awkward conversations users held with this AI bot prevented them from building up such a relationship with this app. Pagano and Maalej also found differences in the user feedback to free vs. paid apps as well as differences related to how many stars the user rated the app. Although we did identify some differences, the pattern with which users report quality aspects persisted.

Beyond software quality

We also found a review that might influence user perceived quality, which reported on the quality of service support. This finding creates a new research direction to investigate reviews from the perspective of requirements to be formulated on a service rather than on a product.

6.3 Study II – Automated Extraction of Quality-Related Sentences

6.3.1 Method

The goal of Study II was to analyze throughly quality-related user feedback sentences to discover language patterns and determine if they can be identified through automated means **(RQ6.2)**. Comparable to the tagging study of Panichella et al. [185] in which 246

recurring linguistic patterns for feature requests were identified and implemented, we focused on "usability"−the characteristic that was most often found in the user reviews in Study I. Our research procedure was executed according to the following steps:

Step 1: Identify linguistic structures and define language patterns. We assessed and compared the 77 individual statements from the 57 reviews that were tagged as "usability" to identify signal words and similar expressions used to describe this characteristic. For statements using distinct words, or for statements with similar structures, we defined language patterns. The notation of these language patterns is an extension of typical regular expressions, where categories of keywords are grouped into word lists (e.g., "EN_Persons" represents any of the words "I", "you", "we", "us", ...). The word list name acts as a placeholder, based on which our analysis tool cycles through all the words from that word list to find a matching pattern in the statements in our data.

The language patterns were furthermore categorized along two dimensions: whether a pattern can be used to find positive, negative, or requesting statements, and whether it matches a functional or quality aspect (in this study, "usability" and its subcharacteristics). For example, the language pattern *"(?i)(EN_Persons)(really |)(need |should)(to |)(be able to)(?<topic>[\w \W]*)"* is a regular expression matching requesting statements that fall under "usability", without addressing a subcharacteristic. Conversely, a language pattern such as *"(?i)(?<topic>[\w \W]*)(?<!EN_Negations)(too |)(complicated)"* matches negative statements on "operability".

Step 2: Evaluate the language pattern syntax Using an online regular expression tester, we were able to determine potential improvements to language patterns based on the following cases:

- A pattern is constructed incorrectly if it does not match any sentence.
- A pattern may be too restrictive if it only matches a few or even only one specific sentence.
- A pattern is too general if it matches sentences not belonging to the (sub)characteristicto which it belongs.
- A pattern is too inaccurate if it cannot correctly differentiate between requirement types.

Since the regular expression tester we used did not support word lists, for the testing we replaced the placeholder with all words in the corresponding word list, separated by "OR" operators.

Step 3: Refine the patterns. We performed a tool-based pattern matching of the language patterns we constructed over the test set of 360 reviews from Study I. Based on this, we were able measure the precision and recall on this test set. After evaluating the language patterns, we improved them in three ways: we resolved (syntax) errors, identified ways to combine similar patterns, and created positive expressions as inverse patterns to negative ones, and vice versa. For example, "A very intuitive, helpful app" and "Not very intuitive" can be recognized by a variant of the pattern *"(?i)(?<!EN_Negation)(EN_Emphasis)(intuitive)"* if the operator *"(?<!EN_Negation)"* is changed to *"(EN_Negation)"*, making it possible to match the negation.

The process of evaluating and refining the language patterns was repeated until they matched (only) the statements tagged as "usability" in our test set (i.e., correct syntax, high precision and recall). We calculated precision as "true positives / matched statements", where the matched statements represented the total of all true positives (TPs) and false positives (FPs).

Step 4: Evaluate the complete dataset. The resulting patterns were used to evaluate the complete dataset of 132,194 statements from our full dataset (see Table 6.1). We then assessed the results to find FPs (wrong subcharacteristic, requirement type, or both). Language patterns with a high FP rate (>50%) were refined in further iterations (if possible)

and tested on the complete dataset again. In the results, we compared the automated results to the tagging results on a per-statement level, as language patterns match individual statements rather than reviews.

6.3.2 Results

In Study I (Section 6.2), 75 statements were tagged as "usability". Of these, 53 (72.6%) were found to follow a pattern for which a language pattern on "usability" or a subcharacteristic could be defined. Of these, 23 (43.4%) were about "operability", 7 (13.2%) about "UI aesthetics", 5 (9.4%) about "learnability", 14 (26.4%) about general "usability", while 3 (5.7%) were tagged as both "operability" and "learnability", and one (1.9%) as "operability" and "UI aesthetics". Based on these, 16 language patterns were composed (see Table 6.3), of which 14 matched statements on "operability" (patterns 1–9, 11, 13–15), one on "learnability" (pattern 10), and one on "UI aesthetics" (pattern 12). The statements on general "usability" did not provide any useful linguistic pattern.

The remaining 20 (28.4%) statements did not provide a pattern for which a language pattern could be constructed, 13 (65.0%) of which referred to "operability", 6 (30.0%) to "UI aesthetics", and 1 (5.0%) to general "usability". These included vaguely formulated or nested statements, such as "One thing that bugs me right away, is all your blocked users show up right in your profile". Other statements provided too little informational content to reliably find statements on "usability" with a language pattern. For example, the pattern in the statement "What a terrible game, look, layout." is "what a terrible...", which would uncover negative statements, but not specifically about "usability". In such cases, searching for individual keywords like "layout" would be possible, but a single word is usually not enough to determine the requirement type.

After using the created language patterns to analyze the complete dataset, 35.940 statements were matched as "usability" (see Table 6.3), of which 34.286 (95,4%) resulted from an erroneous language pattern (9) that did not consider possible negation properly due to a difficult formulation and was therefore excluded from further iterations. The remaining 1,654 statements from 15 language patterns were assessed on their correctness, of which 1,528 (92.4%) were TPs, and 126 (7.6%) FPs. The TPs matched 1,519 statements (99.4%) about "operability", 7 (0.5%) about "learnability", and 2 (0.1%) about "UI aesthetics" . No language patterns assessed general statements about "usability". This shows that there was a much greater emphasis on "operability", compared to the 40 statements we found in our tagged set.

Upon analysis of the FPs, we found that 87 (69.0%) of the statements were relevant statements on "usability", but did not match the valence of the language pattern. For example, the negative statement was matched by a language pattern formulated to find positive statements, because negations were not included as forbidden words (e.g. the language pattern *"(?i)(?<!EN_Negations)(EN_Emphasis)(intuitive)"* should find positive sentences but did not exclude "...settings does not seem very intuitive"). Only 27 (21.4%) of the FPs did not match the characteristic "usability" or the subcharacteristic that should be matched by the language pattern. The remaining 12 FPs (9.5%) addressed multiple (sub)characteristics, including the intended one.

6.3.3 Discussion

Linguistic structure of statements

In this study, we assessed the statements tagged as "usability" and found that ***the majority of the statements could be matched through language patterns*** because their statements

Language Pattern ID	Matched Statements	TPs	FPs	Precision
1	17	13	4	76.5%
2	3	2	1	66.6%
3	1,320	1,301	19	98.6%
4	31	31	0	100.0%
5	55	54	1	98.2%
6	20	14	6	80.0%
7	146	61	85	42.5%
8	10	10	0	100.0%
9	34,268*	–	–	–
10	14	7	7	50.00%
11	1	1	0	100.0%
12	2	2	0	100.0%
13	4	4	0	100.0%
14	1	1	0	100.0%
15	20	17	3	85.00%
16	10	10	0	100.0%
Total	*1,654*	*1,528*	*126*	*92.4%*

Table 6.3: Results from our analysis of the 16 language patterns over the complete dataset. Language patterns with an asterisk (*) were too general and are omitted from further iterations and total counts.

follow a particular pattern **(RQ6.2)**. Although most of the statements for which we were unable to formulate a language pattern were about "operability", still the largest share of statements could address this subcharacteristic.

From analyzing the results of Study I, we found that the majority of statements used variations of word combinations, which included "(not) easy to …", "… (not) very intuitive / user friendly", and "would like to be able to …" to describe their experience with ah app's operability. The first two examples show a clear valence (either positive or negative), while the third shows a request for a quality (for which a negated statement is unlikely). This means that statements with a negative valence should require the combination of a negation and a positive word (e.g., "not easy"), or the use of a negative word (e.g., "hard"), while the same pattern could forbid those words to find statements with a positive valence. Making a distinction in valence is important when using the statements to elicit requirements, because finding either many positive or many negative statements about "usability" have very different implications for RE. Requesting statements about qualities (i.e., "quality requests") will either help to prevent negative statements about this quality in the future or may contain a suggestion for a unique quality property.

Manual extraction of quality-related statements from our full dataset would be very time-consuming. Instead, ***it was possible to construct language patterns to find statements*** following particular types of patterns in a larger dataset. For example, the language pattern addressing the statements that are a variation of "easy to use" read: *"(?i)(?<!EN_Negations)(that |)(simple |simply |easy |easily)(to |)(use |used |install |installed |set up |understand)"*. This way, 16 sentence patterns were composed in total.

Language pattern derivation

The process of ***formulating language patterns was time-consuming*** and several iterations were required. Often because even ***small syntax errors can greatly compromise results***.

However, correct language patterns show promising results, especially as patterns for positive statements can be easily adapted to find negative statements, and vice versa. But sentence constructions reflecting a negative statement were difficult to identify, since in many cases adding a negating word is not enough. Especially the language patterns with IDs 7 and 10 should be further improved with respect to their precision.

Of course it would be also possible to use a query to find particular keywords (such as the combination "easy to use"), which could reveal many relevant statements. However, such a query would not automatically assess for each statement whether it is positive (e.g., "very easy to use"), negative (e.g„ "not easy to use") or requesting (e.g., "please make it more easy to use"). Hence, more intricate patterns are needed.

Language pattern precision

The results already revealed many TPs, though ***we also identified many FPs***, many of which did address "usability" but had the wrong valence or subcharacteristic. Moreover, ***several matched statements contained ambiguities*** (e.g., vague formulations), meaning we could not always identify the result as a TP with certainty. When in doubt, we classified such a statement as a FP.

Although we cannot determine the recall over such a large dataset, we can infer that only about an eighth of the statements were found, but in a very short amount of time and with a limited set of sentence patterns. It is possible that composing additional language patterns may improve the recall. We constructed our language patterns in a bottom-up way by deriving them from identified sentences. One other way to compose patterns may be to use a top-down approach, where possible expressions about "usability" would be collected in an expert workshop through reasoning. However, the ambiguous formulations of users will likely always prevent automation from finding some statements that human annotators would classify as "usability".

One limitation of the 16 language patterns is that compared to the manual tagging, *they covered only a portion of all relevant statements* on "usability". In Study I (see Section 6.2), we found 75 statements on "usability" from 1,368 statements. Assuming linear scalability, our full dataset (consisting of 384,510 statements) theoretically contains 20,518 statements on usability, of which we covered only 7.4%, but using only 16 language patterns. This shows that the recall could be increased overall, especially if more language patterns are used or if these are based on certain keywords aimed at also identifying nested and unclear statements.

General Discussion

From the outcomes of our studies, we gather that the user feedback in online reviews can provide relevant information on software product quality, but only on aspects by which users are affected. By assessing one characteristic, we also found that many such statements follow particular patterns, making it possible to find statements on quality more efficiently through automation. Determining the way users perceive the quality of an app, and the problems or requests they describe, can help in formulating new NFRs or improving existing ones. We therefore we argue that ***online reviews should be considered more seriously as an elicitation source for NFRs***.

6.4 Threats to Validity

We discuss the validity of our studies and results using the guidelines from Wohlin et al. [241].

Conclusion validity. In order to mitigate the 'fishing' threat, we employed a software tool for tagging that presented only the contents of the review to the annotators, we involved five annotators, and we defined a detailed procedure for tagging.. To increase ghe reliability of the results (i.e., the reliability of the measures, the heterogeneity of annotators, and the subjectivity of the tagging), we developed a detailed tagging schema based on the structure of the ISO 25010 standard, performed a pilot study, and discussed any inconsistencies and doubts among the annotators to ensure that each review got the most suitable tag to the best of our knowledge. In our reporting, we used both the number of reviews and the number of statements to provide better insights into our findings from the user feedback.

Construct validity. Another issue that might have threatened our results concerns the way the reviews are written. The language is mostly informal, the knowledge of the users varies greatly and many users write poorly or ambiguously. This also compromised the certainty with which the matched statements in Study II could be marked as TPs, so t in case of doubt, such statements was marked as FPs.

Internal validity. Obviously, there could be some variation in user feedback that is related to a certain (type of) app. Thus, we based our results on a stratified sample (see Table 6.1) and included different apps (product, category, cost) from the three most popular app stores.

External validity. Although our conclusions are based on a sample of reviews for 12 apps (Study I on 360 reviews, Study II on 131,928 reviews), to increase the ability to generalize the results, we used quite a wide variety of apps with respect to category and cost, reviews come from three different app stores and include all possible ratings. Of course, it should be noted that the findings might not be generalizable to other products, as some may show entirely different patterns. With respect to the possibility of automating the extraction of quality-related sentences, the conclusions are restricted to only the characteristic "usability".

6.5 Related Work

It has long been known that NFRs are a crucial aspect of specifying a piece of software [167]. These requirements are the typical architecture drivers [191, 8]. Not addressing them adequately leads to project failure and high rework cost [52]. But getting a complete and correct set of NFRs is difficult [38], and thereby costly. Various approaches exist for eliciting NFRs, among them[129, 39, 8].

One industry-validated approach to eliciting a complete set of NFRs is the NFR method of Fraunhofer IESE [62, 63], which assumes a backlog of experience-based artifacts to identify potentially relevant quality characteristics, a step needed to get a complete set of relevant NFRs. This backlog is modified in several discrete steps to tailor it to the specificities of the project (i.e., the domain in which the NFRs need to be elicited). This process is work- and cost-intensive and requires domain experts who typically have very limited availability. One way to reduce the manual effort required to perform the NFR method is to automate some of its steps. Such an approach using automation would make use of text mining to identify statements about relevant product quality characteristics provided by crowd members. The type of statements indicates relevant characteristics (for the current system development task, but also for the experience-based artifacts). The number

and quality of these statements can then assist in assigning the priority of the characteristics this information is important for focusing the limited NFR elicitation effort.

Analyzing existing textual documents to identify potential requirements has a long tradition in Requirements Engineering. Since the early 2000s, research has been adapting computational linguistic techniques to analyze user feedback for RE [174, 130]. In product-line engineering, document analysis during scoping has proven to contribute to a better understanding of the domain better identification of product features [119]. Analysis of legal texts has been used to extract relevant requirements for software products in regulated environments [177]. In recent years, various works have been published on analyzing app reviews to extract requirements related information, (e.g., by [90, 158, 148]. To the best of our knowledge, no research has been performed on analyzing the specificities of non-functional requirements based on existing reviews. With Crowd-based Requirements Engineering (CrowdRE), text-mining techniques have been adapted to Requirements Engineering to derive statements about requirements in natural language texts [88]. To this end, the checklists and templates used in the NFR-method are replaced by or at least augmented with keywords and queries that make use of those keywords to identify statements from texts (e.g., from crawled product reviews in app stores). But the use of Crowd-based Requirements Engineering to derive relevant product characteristics is not limited to smartphone apps in the above mentioned NFR method. Browsing a catalog of NFR templates [129] could also be supported with knowledge about which qualities are regarded as more important. In addition, some prompts might be provided regarding how to tailor NFR templates to get NFRs that suit potential users. Moreover, goal-oriented approaches like [39] might benefit from CrowdRE as the found characteristics would augment the goal models, typically as soft-goals / quality goals. Proximity of statements could be used to provide hints for relationships in goal models (e.g., [39]).

6.6 Summary

In the chapter we analyzed the contents of on-line reviews from three app stores (Apple App Store, Google Play, Amazon Appstore) looking for statements concerning non-functional aspects.

We have found that more than 40% of user reviews concerned non-functional aspects. Users in their reviews provided NFR-related feedback which might be mapped onto all but one characteristic of the ISO 25010 standard (the exception was the “maintainability” characteristic by which users are not directly affected). Thus, app stores proved a good source of NFRs, and, consequently, of their templates.

Moreover, we used the language patterns approach to automatically identify reviews related to the “usability” characteristic. The approach allowed us to to identify statements about NFR aspects with high precision (more than 90%), though the recall was modest (less than 10%) at the time of the study and the effort needed to construct the patterns was significant.

Chapter 7

Conclusions

7.1 Answers to the research questions

The aim of the thesis was to provide knowledge that would allow managers of software projects to make an informed decision whether to use or not to use a catalog of NFR templates.

According to Endres and Rombach [72], one should distinguish between laws, hypotheses, and conjectures. A law requires strong empirical evidence supporting it. When there are some validity threats concerning, e.g., generalizability, it is better to call such finding 'hypothesis'. Since the presented studies have been performed at one university and have not been reproduced in other contexts, it has been decided to present the findings as 'hypothesis', not 'laws'. Some of our findings are very concrete (they show some numbers). Therefore, they look highly practical but still, they lack generalizability, and the presented numbers may prove very fragile (dependent on the context). Taking these arguments into account we decided to call such findings 'conjectures'.

The hypotheses and conjectures presented below are the answers to the research questions (**RQx**) stated in Section 1.1.

RQ1. *How important it is for agile practitioners to have NFRs specified in their software projects?*

Hypothesis 1. *NFRs are important not only in traditional software projects but also in agile ones.*

Justification. The above hypothesis is based on the results of the survey described in Chapter 3. 47% of 118 agile practitioners considered having NFRs specified as an important aspect for software projects, and for the next 30% this was even critical.
Moreover, in traditional (non-agile) approaches to software development, the importance of NFRs was unquestionable. Also, analyses of the failures of software projects confirm their importance.

RQ2. *What is the usefulness of catalog of NFR templates?*

Hypothesis 2a. *Template-supported elicitation of NFRs does not hinder their firmness.*

Justification. It could happen that given a set of NFR templates people will elicit much more NFRs than really needed. The case study with 7 software development projects (see Section 4.3) showed that this is not the case. The firmness of NFRs elicited with templates (i.e., the percentage of the initial NFRs that remained valid till the end of the project) was in the range of 80–90%. This percentage is very high, and the space left for improvement is rather small. Thus, it is difficult to imagine that elicitation without templates could result in the firmness significantly higher from what we have observed.

Hypothesis 2b. *Template-supported elicitation of NFRs positively impacts their quality.*

Justification. Our experiment with 107 inexperienced elicitors of requirements (see Section 4.4) showed that elicitation of NFRs supported with catalog of NFR templates results in NFRs which are more complete, less ambiguous, more detailed, and better from the point of view of verifiability, for both individual elicitors and teams thereof.

Hypothesis 2c. *Elicitors regard NFR templates as useful for elicitation of NFRs.*

Justification. In both our case study (see Section 4.3) with 7 software development projects and our experiment with 107 inexperienced elicitors of requirements (see Section 4.4) over 80% of respondents had positive impression about NFR templates and regarded them as useful.

Hypothesis 2d. *Using templates to support elicitation of NFRs does not speed up the elicitation process.*

Justification. The above hypothesis is one more result of the study described in Section 4.4, i.e., our experiment with 107 inexperienced elicitors of requirements. We have not noticed a significant difference in the number of NFRs elicited with and without catalog neither for individuals nor for teams.

RQ3. *How the benefits and costs of using catalog change during catalog evolution?*

Hypothesis 3a. *The greater the number of projects used to develop a catalog of NFR templates, the greater the value it provides and the smaller the number of maintenance activities the catalog requires.*

Justification. Our study was based on 41 industrial projects and 2231 NFRs. We investigated the original evolution of the catalog and 10,000 simulated permutations thereof (see Chapter 5). Both examinations support the above hypothesis.

Hypothesis 3b. *The greater the number of projects used to develop a catalog of NFR templates, the greater its size and the smaller its usage.*

Justification. The above hypothesis follows from the study mentioned in Observation 3a and described in detail in Chapter 5. After considering 40 projects our catalog contained about 400 templates. Searching through 400 templates to find less than 40 needed (sometimes even less than 20) could be annoying. Thus, a good catalog of NFR templates should offer a searching method more sophisticated and efficient than brute force.

RQ4. *To what extent user feedback in app stores is a good source of NFRs and their templates?*

Hypothesis 4. *App stores are a good source of NFRs and their templates.*

Justification. The above observation follows from the analysis of user feedback of 1,385 statements left in three app stores: Google Play, Apple App Store, and Amazon Appstore (see Section 6.1). We found that about 20% of statements contain information about non-functional aspects.
We also examined the potential of using language patterns to detect user feedback concerning NFRs automatically. The approach appeared to require a considerable amount of effort and more research seems needed.

From the above-presented results it follows that NFR templates seem a valuable tool to support elicitation of NFRs.

Moreover, based on the studies reported in this thesis one can formulate the following conjectures that can be treated as rules-of-thumb by practitioners:

- **Conjecture 3c.** *After "learning" from about 40 projects one can expect that their catalog of NFR temples gets mature that is at least 75% of NFRs of the next project can be derived from its templates and up to 10% of the templates may need to be modified or added.*
  (Follows from the studies described in Chapter 5.)
- **Conjecture 3d.** *After "learning" from about 40 projects one can expect that less than 10% of the NFRs templates from their catalog would be used in the next project.*
  (Follows from the studies described in Chapter 5.)
- **Conjecture 2e.** *Two 1.5-hour long template-supported workshops seem sufficient to elicit good-enough NFRs for a software project.*
  (Follows from the study described in Section 4.3.)

7.2 Future work

From the research reported in the thesis, it follows that elicitation and management of NFRs is a real issue, even in the era of agile methodologies, and catalog of NFR templates could be helpful in this context. The problem that emerged is the low effectiveness of browsing through a mature catalog. Therefore, investigating other approaches to template–supported elicitation of NFRs seems worth of effort. One of them could be a kind of recommender system that would use functional requirements (e.g., in the form of use cases) as an input. First, those requirements would be analyzed with NLP tools, then some inference would take place to identify needed templates (those templates would be represented in an abstract way), and finally the catalog of templates would be used to generate NFRs in a given natural language (a similar approach to identifying events in use cases has been described in [120]). Such a recommender system could be made polyglot: after translating the templates to, e.g., German one could generate NFRs in both English and German.

Another important stream of future work would be a continuation of experiments and case studies to get a stronger generalization of the announced results. For instance, firmness of NFRs was studied with only 7 projects—further investigation with more projects would be very useful.

Furthermore, since we identified that online user feedback systems like app stores might be useful for NFR elicitation, there are at least two future work directions. First, one might work on an approach on how to use the knowledge gathered in app stores to enrich catalogs of NFR templates. Secondly, since there is a large amount of user feedback in app stores, there is a need to work on methods and tools that could, possibly automatically, extract the non-functional related aspects.

Appendix A

Survey results

In the section we present the summary of the responses of the participants of the study reported in Section 4.4 concerning the impression about NFR templates after using them to elicit NFRs for an e-commerce system.

	Strongly yes	Rather yes	Hard to say	Rather no	Strongly no	I don't know
Q1. Have NFR templates been useful?						
Indv.	41.46% (17)	39.02% (16)	14.63% (6)	2.44% (1)	2.44% (1)	0.00% (0)
Teams	37.50% (9)	58.33% (14)	0.00% (0)	4.17% (1)	0.00% (0)	0.00% (0)
All	40.00% (26)	46.15% (30)	9.23% (6)	3.07% (2)	1.54% (1)	0.00% (0)
Q2. Have NFR templates been of good quality?						
Indv.	31.71% (13)	41.46% (17)	24.40% (10)	0.00% (0)	0.00% (0)	2.44% (1)
Teams	16.67% (4)	54.17% (13)	16.67% (4)	4.17% (1)	4.17% (1)	4.17% (1)
All	26.15% (17)	46.15% (30)	21.54% (14)	1.54% (1)	1.54% (1)	3.08% (2)
Q3. Are elicited NFRs of good-enough quality for architecture design?						
Indv.	12.20% (5)	46.34% (19)	26.83% (11)	9.76% (4)	2.44% (1)	2.44% (1)
Teams	0.00% (0)	54.17% (13)	12.50% (3)	20.83% (5)	8.33% (2)	4.16% (1)
All	7.69% (5)	4.92% (32)	21.54% (14)	13.85% (9)	4.61% (3)	3.07% (2)

Table A.1: The responses of the survey respondents concerning their impression on NFR templates for those who worked individually (Indv.) and in teams (Teams) as the percentage and the number of responses for a given answer option is given in the brackets '()'.

Appendix B

Coarse-grained analysis

B.1 Aim of coarse-grained analysis

In the study described in Chapter 5 we investigated three variables important from the point of view of maintenance process: (1) catalog value (i.e., the percentage of the NFRs of a given project that might be elicited with the use of the catalog); (2) maintenance effort (describes the percentage of the number of *add-a-template* and *modify-a-template* operations needed to incorporate lessons learned from a single project); (3) catalog utilization (i.e., the percentage of templates that are used by a single project). We performed 10,000 different random evolutions of a catalog of NFR templates using 41 sets of NFRs using the fine-grained analysis described in Section 5.4. However, this type of analysis might have introduced some errors. Thus, to mitigate the identified threat, we conducted also the coarse-grained analysis. In the appendix, we compare the investigated variables in both types of analysis: fine-grained and coarse-grained.

The coarse-grained analysis is based only on the information on relationships between NFR templates and requirements. Each relationship was identified in the initial catalog evolution when t was either added to catalog K_{i-1} or was used to derive requirement r from project P_i.

For each project P_i and for each template t that was identified to have a relationship between requirement $r \in P_i$:

1. If catalog K_{i-1} contained t, we also included t in K_i.
2. If such template could not be found, we added new template t to catalog K_i.

All sleeping templates of K_{i-1} (not having any relationship between $r \in P_i$) were included in K_i.

Having only the data on the relationships between templates and requirements does not allow to distinguish between two subsets of templates: Perfect and Modified (it is one set), and, therefore, the value of maintenance effort (ME) cannot be determined. We decided to use an approximation of ME that is the number of added templates–Approx. ME.

B.2 Value

Both Figure B.1 and Table B.1 show that the results concerning catalog Value obtained using the fine-grained approach and the coarse-grained approach do not differ much. The figure illustrates the same increasing tendency in the data. From the figure, it follows that the data ranges seem nearly the same, which can be confirmed by the data summary in Table B.1. The differences are in the value of the mean and standard deviation and are in the second decimal place.

Observation 5.1 is also supported by the coarse-grained approach, i.e., from Figure B.1 it also follows that after concerning about 40 projects one can expect the value of catalog of 75% and more.

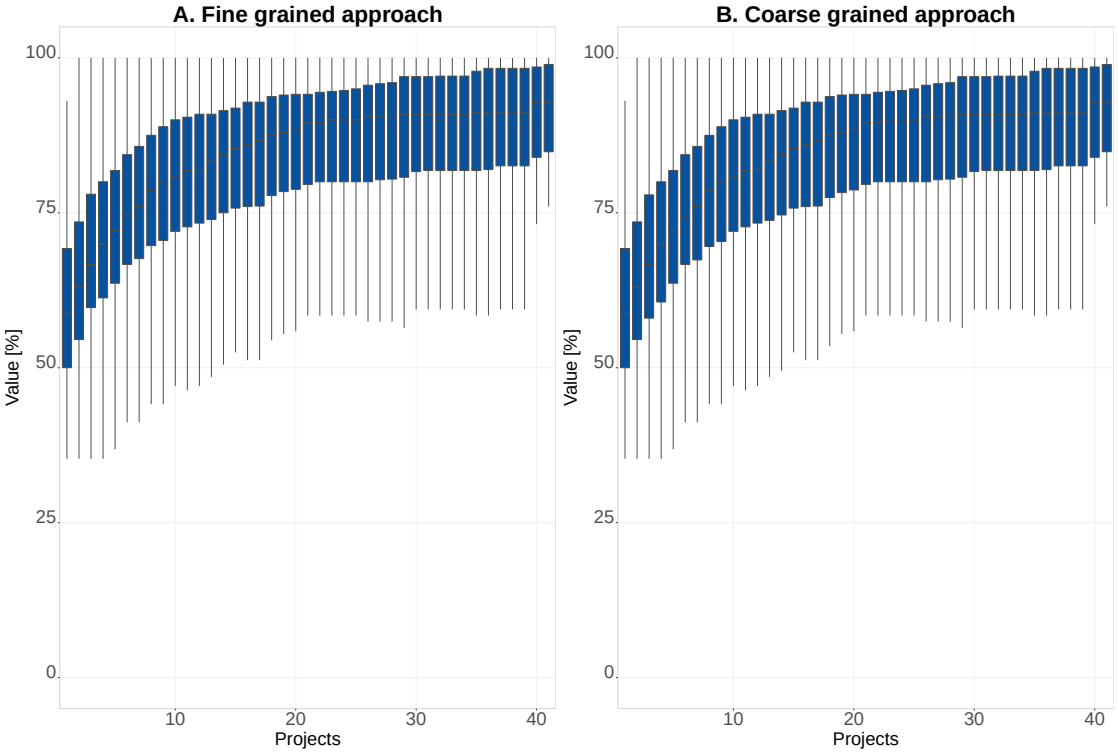


Figure B.1: Distribution of catalog value (Value) for multiple evolution using fine-grained approach (A) and coarse-grained approach (B).

	Catalog Value				
	Min.	**Mean**	**Median**	**Max.**	**SD**
Fine-grained	35.30%	83.50%	86.67%	100.00%	13.24%
Coarse-grained	35.30%	83.47%	86.67%	100.00%	13.28%

Table B.1: Minimum, maximum, average, median and standard deviation (SD) values for catalog value computed using fine-grained and coarse-grained approach to multiple evolution.

B.3 Maintenance Effort

The distribution of ME and approximation of ME is presented in Figure B.2. Since we are not able to calculate the value of ME using the coarse-grained approach, from the figure it follows that using both approaches we can observe the decreasing tendency in the data. Moreover, it is visible that the values of approx. ME are within the values of ME, which is expected.

The obtained data using the coarse-grained approach is not in the contradiction with Observation 5.2, i.e., it seems that less than 10% of templates is added after considering about 40 projects.

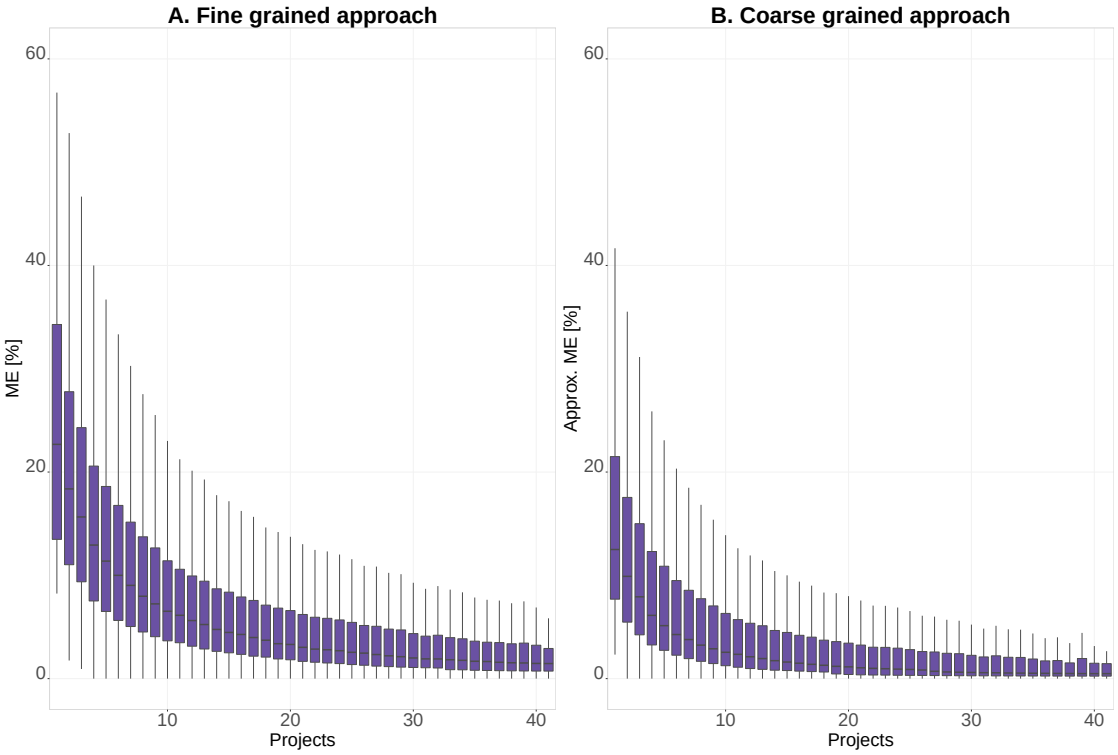


Figure B.2: Distribution of catalog maintenance effort (ME) and its approximation for multiple evolution using fine-grained approach (A) and coarse-grained approach (B).

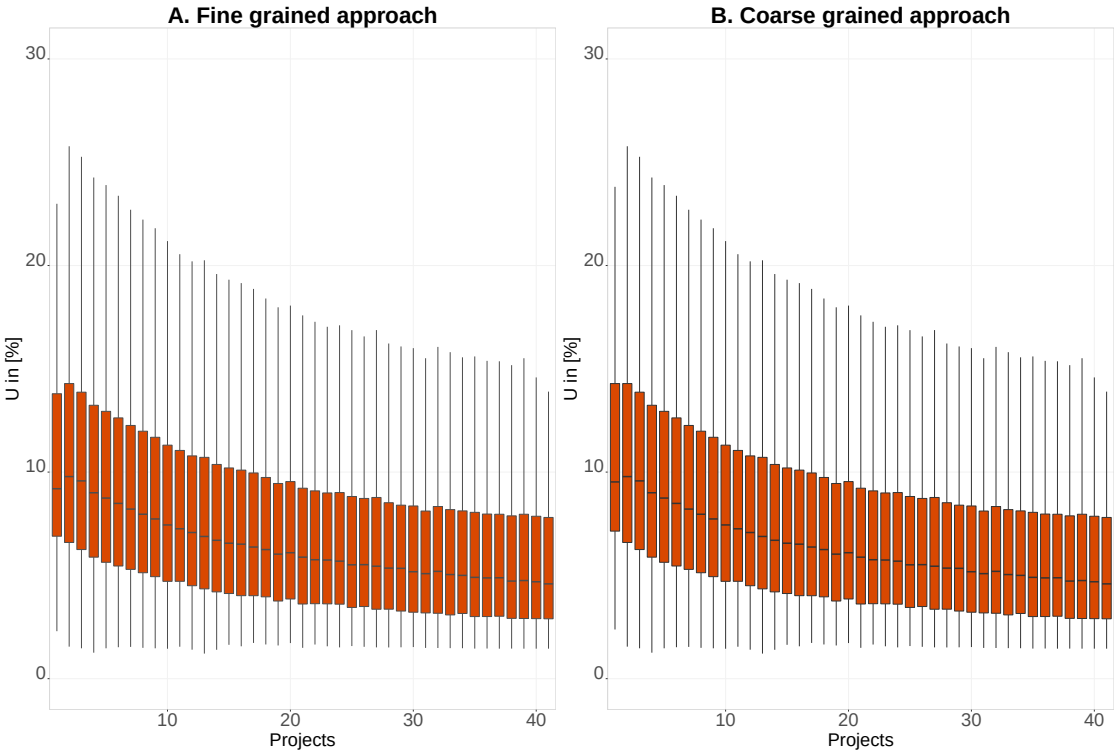


Figure B.3: Distribution of catalog utilization (U) for multiple evolution using fine-grained approach (A) and coarse-grained approach (B).

B.4 Utilization

The distribution of catalog utilization (U) for the fine-grained and the coarse-grained approaches is presented in Figure B.3, and the summary of the data is presented in Table B.2. From both the figure and the table it follows that in both approaches the distribution of data is very similar.

Observation 5.3. is supported by the results of the coarse-grained analysis, i.e., after considering about 40 projects one can expect that catalog utilization would be below 10%.

	Utilization				
	Min.	**Mean**	**Median**	**Max.**	**SD**
Fine-grained	1.21%	7.41%	6.28%	38.05%	4.70%
Coarse-grained	1.21%	7.42%	6.29%	38.05%	4.72%

Table B.2: Minimum, maximum, average, median and standard deviation (SD) values for utilization computed using fine-grained and coarse-grained approach to multiple evolution.

B.5 Conclusions

It follows from the analysis of data distribution using box plots and data summaries that the data obtained using the coarse-grained analysis does not differ importantly from the data obtained using the fine-grained one. Thus, the threat concerning introducing error while performing the fine-grained analysis seems negligible and does not seem to influence the final findings.

References

[1] P. Abrahamsson and J. Koskela. Extreme programming: A survey of empirical data from a controlled case study. In *Proc. of Intl. Symp. on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2004.

[2] P. Abrahamsson, O. Salo, and J. Ronkainen. Agile software development methods: review and analysis. Technical report, VTT Electronics and University of Oulu, 2002.

[3] S. Adolph, P. Bramble, A. Cockburn, and A. Pols. *Patterns for Effective Use Cases*. Addison-Wesley, 2002.

[4] M. O. Ahmad, J. Markkula, and M. Oivo. Kanban in software development: A systematic literature review. In *Proc. of 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2013.

[5] C. Alexander. *A pattern language: towns, buildings, construction*. Oxford university press, 1977.

[6] I. Alexander. Misuse cases: Use cases with hostile intent. *IEEE Software*, 20(1):58–66, 2003.

[7] M. A. Ali. Survey on the state of agile practices implementation in pakistan. *International Journal of Information and Communication Technology Research*, 2(4), 2012.

[8] D. Ameller, C. Ayala, J. Cabot, and X. Franch. Non-functional requirements in architectural decision making. *IEEE Software*, 30(2):61–67, 2013.

[9] D. Ameller and X. Franch. How do software architects consider Non-Functional Requirements: a survey. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2010.

[10] A. Antón. Goal identification and refinement in the specification of information systems. PhD Thesis, Georgia Institute of Technology, 1997.

[11] J. Arnowitz, M. Arent, and N. Berger. *Effective prototyping for software makers*. Elsevier, 2010.

[12] G. Barabino, D. Grechi, D. Tigano, E. Corona, and G. Concas. Agile methodologies in web programming: a survey. In *Proc. of Intl Conference on Agile Software Development*. Springer, 2014.

[13] L. D. Baranek. Victim of friendly fire. Technical report, available online at: `https://www.globalsecurity.org/military/library/report/1993/BD.htm`, last access 06 August 2018, 1993.

[14] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2000.

[15] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. *Manifesto for Agile Software Development, http://www.agilemanifesto.org/*. avilable online, 2001.

[16] A. Begel and N. Nagappan. Usage and perceptions of agile software development in an industrial context: An exploratory study. In *1st Intl Symp. on Empirical Software Engineering and Measurement (ESEM)*, 2007.

[17] M. Behzadian, R. Kazemzadeh, A. Albadvi, and M. Aghdasi. Promethee: A comprehensive literature review on methodologies and applications. *European Journal of Operational Research*, 200(1), 2010.

[18] S. Berczuk. Back to basics: The role of agile principles in success with an distributed scrum team. In *Agile Conference (AGILE)*. IEEE, 2007.

[19] D. M. Berry. Ambiguity in natural language requirements documents. In *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*. Springer, 2008.

[20] D. M. Berry, E. Kamsties, and M. M. Krieger. *From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity. A Handbook. Ver 1.0.* online, available at: `https://cs.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf`, last access: 07/09/2015.

[21] E. Bjarnason, K. Wnuk, and B. Regnell. Requirements are slipping through the gaps – a case study on causes & effects of communication gaps in large-scale software development. In *Proc. of 19th Intl Requirements Engineering Conference (RE)*. IEEE, 2011.

[22] B. Boehm and H. In. Identifying quality-requirement conflicts. *IEEE Software*, 13(2):25–35, 1996.

[23] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd International Conference on Software Engineering*, pages 592–605. IEEE Computer Society Press, 1976.

[24] K. Boness and R. Harrison. Goal sketching: Towards agile requirements engineering. In *Proc. of Intl Conf. on Software Engineering Advances (ICSEA)*. IEEE, 2007.

[25] K. K. Breitman, J. C. S. Leite, and A. Finkelstein. The world sa stage: a survey on requirements engineering using a real-life case study. *Journal of the Brazilian Computer Society*, 6(1):13–37, 1999.

[26] M. Brhel, H. Meth, A. Maedche, and K. Werder. Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology*, 61, 2015.

[27] G. J. Browne and V. Ramesh. Improving information requirements determination: a cognitive perspective. *Information and Management*, 39(8):625–645, 2002.

[28] A. Buchalcevova. Research of the use of agile methodologies in the czech republic. In *Information Systems Development*. Springer, 2009.

[29] J. Burge and D. Brown. NFR's: Fact or Fiction? Computer Science Technical Report WPI-CS-TR-02-01, Worcester Polytechnic Institute.

[30] L. Cao and B. Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 25(1), 2008.

[31] E. S. Cardozo, J. B. F. A. Neto, A. Barza, A. C. C. França, and F. Q. da Silva. SCRUM and Productivity in Software Projects: A Systematic Literature Review. In *Proc. of 14th Intl Conference on Evaluation and Assessment in Software Engineering (EASE)*. BCS Learning & Development Ltd., 2010.

[32] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information Systems*, 27(6):365–389, 2002.

[33] A. Causevic, D. Sundmark, and S. Punnekkat. An industrial survey on contemporary aspects of software testing. In *Proc. of 3rd Intl Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2010.

[34] K. Charmaz. *Constructing Grounded Theory*. SAGE Pulications, 2006.

[35] T. Chau and F. Maurer. Knowledge sharing in agile software teams. In *Logic versus approximation*. Springer, 2004.

[36] L. Chen, M. A. Babar, and B. Nuseibeh. Characterizing architecturally significant requirements. *IEEE Software*, 30(2):38–45, March 2013.

[37] T. Chow and D.-B. Cao. A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, 81(6), 2008.

[38] L. Chung and J. d. P. Leite. On non-functional requirements in software engineering. *Conceptual modeling: Foundations and applications, LNCS*, 2009.

[39] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012.

[40] J. Cleland-Huang, A. Czauderna, and E. Keenan. A persona-based approach for exploring architecturally significant requirements in agile projects. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 18–33. Springer, 2013.

[41] J. Cleland-Huang, S. Mazrouee, H. Liguo, and D. Port. Open-Science teraPROMISE repository. http://openscience.us/repo/requirements/other-requirements/nfr, 2010 (accessed June 26, 2017.

[42] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In *Requirements Engineering, 14th IEEE International Conference*, pages 39–48. IEEE, 2006.

[43] P. Clements and L. Bass. Using business goals to inform a software architecture. In *18th IEEE International Requirements Engineering Conference (RE)*, pages 69–78. IEEE, 2010.

[44] N. Cliff. *Ordinal Methods for Behavioral Data Analysis*. Psychology Press, 1996.

[45] CMMI Product Team. CMMI® for Development, Version 1.3. Technical report, Carnegie Mellon University, November 2010.

[46] J. Cohen. Statistical power analysis. *Current Directions in Psychological Science*, 1(3):98–101, 1992.

[47] M. Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[48] M. Cohn. *Agile estimating and planning*. Pearson Education, 2005.

[49] M. Cohn. *Succeeding with agile: software development using Scrum*. Pearson Education, 2010.

[50] K. Conboy, X. Wang, and B. Fitzgerald. Creativity in agile systems development: A literature review. In *Information Systems–Creativity and Innovation in Small and Medium-Sized Enterprises*. Springer, 2009.

[51] L. L. Constantine and L. A. Lockwood. *Software for use: a practical guide to the models and methods of usage-centered design*. Pearson Education, 1999.

[52] L. M. Cysneiros and J. C. S. do Prado Leite. Using uml to reflect non-functional requirements. In *Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*, page 2. IBM Press, 2001.

[53] L. M. Cysneiros and J. C. S. do Prado Leite. Nonfunctional requirements: From elicitation to conceptual models. *IEEE transactions on Software engineering*, 30(5):328–350, 2004.

[54] F. Q. da Silva, C. Costa, A. C. C. Franca, and R. Prikladinicki. Challenges and solutions in distributed software development project management: A systematic literature review. In *Proc. of 5th Intl Conference on Global Software Engineering (ICGSE)*. IEEE, 2010.

[55] T. S. Da Silva, A. Martin, F. Maurer, and M. Silveira. User-centered design and agile methods: a systematic review. In *Proc. of Agile Conference (AGILE)*. IEEE, 2011.

[56] M. Daneva, E. Van Der Veen, C. Amrit, S. Ghaisas, K. Sikkel, R. Kumar, N. Ajmeri, U. Ramteerthkar, and R. Wieringa. Agile requirements prioritization in large-scale outsourced system projects: An empirical study. *Journal of Systems and Software*, 86(5), 2013.

[57] A. Davis. *Software Requirements: Objects, Functions and States*. Prentice Hall, 1993.

[58] I. Dees, M. Wynne, and A. Hellesoy. *Cucumber Recipes: Automate Anything with BDD Tools and Techniques*. Pragmatic Bookshelf, 2013.

[59] C. Denger, D. M. Berry, and E. Kamsties. Higher quality requirements specifications through natural language patterns. In *IEEE Internat. Conference on Software: Science, Technology and Engineering*, pages 80–90, 2003.

[60] P. Diebold and M. Dahlem. Agile practices in practice: a mapping study. In *Proc. of 18th Intl Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014.

[61] J. P. Djajadiningrat, W. W. Gaver, and J. Fres. Interaction relabelling and extreme characters: methods for exploring aesthetic interactions. In *Proc. of 3rd Conf. on Designing interactive systems: processes, practices, methods, and techniques*. ACM, 2000.

[62] J. Doerr. *Elicitation of a Complete Set of Non-Functional Requirements*. PhD thesis, University of Kaiserslautern, 2011.

[63] J. Doerr, D. Kerkow, T. Koenig, T. Olsson, and T. Suzuki. Non-functional requirements in industry-three case studies adopting an experience-based nfr method. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 373–382. IEEE, 2005.

[64] M. Doyle, L. Williams, M. Cohn, and K. S. Rubin. Agile software development in practice. In *Proc. of Intl Conference on Agile Software Development*. Springer, 2014.

[65] T. Dybå and D. S. Cruzes. Process research in requirements elicitation. In *IEEE Third Int. Workshop Empirical Requir. Eng. (EmpiRE)*, pages 36–39, 2013.

[66] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9), 2008.

[67] C. Ebert. Putting requirement management into praxis: dealing with non- functional requirements. *Information and Software Technology*, 40(3):175–185, 1998.

[68] J. Eckhardt. *Categorizations of Product-related Requirements in Practice*. Phd thesis, Technische Universität München, München, 2017.

[69] J. Eckhardt, A. Vogelsang, H. Femmer, and P. Mager. Challenging incompleteness of performance requirements by sentence patterns. In *International on Requirements Engineering Conference (RE)*, pages 46–55. IEEE, 2016.

[70] J. Eckhardt, A. Vogelsang, and D. M. Fernández. Are" non-functional" requirements really non-functional? an investigation of non-functional requirements in practice. In *International Conference on Software Engineering (ICSE)*, pages 832–842. IEEE/ACM, 2016.

[71] H. Elshandidy and S. Mazen. Agile and traditional requirements engineering: A survey. *Internat. Journal of Scientific & Engineering Research*, 9, 2013.

[72] A. Endres and H. D. Rombach. *A handbook of software and systems engineering: Empirical observations, laws, and theories*. Pearson Education, 2003.

[73] F. Faul, E. Erdfelder, A. Lang, and A. Buchner. G*Power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behavior Research Methods*, 39(2):175–191, 2007.

[74] A. Finkelstein and J. Dowell. A comedy of errors: the London Ambulance Service case study. In *Proc. 8th Int. Work. Softw. Specif. Des.*, page 2, 1996.

[75] D. G. Firesmith. Analyzing and specifying reusable security requirements. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA, 2003.

[76] R. M. Fontana, I. M. Fontana, P. A. da Rosa Garbuio, S. Reinehr, and A. Malucelli. Processes versus people: How should agile software development maturity be defined? *Journal of Systems and Software*, 97, 2014.

[77] A. C. C. França, F. Q. da Silva, and L. M. de Sousa Mariz. An empirical study on the relationship between the use of agile practices and the success of scrum projects. In *Proc. of Intl Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM-IEEE, 2010.

[78] X. Franch. Systematic formulation of non-functional characteristics of software. In *International Conference on Requirements Engineering*, 1998.

[79] X. Franch and J. P. Carvallo. Using quality models in software package selection. *IEEE software*, 20(1):34–41, 2003.

[80] D. C. Gause and G. M. Weinberg. *Exploring requirements: quality before design*. Dorset House New York, 1989.

[81] S. Ghobadi. What drives knowledge sharing in software development teams: A literature review and classification framework. *Information & Management*, 52(1), 2015.

[82] T. Gilb. *Competitive Engineering: A Handbook for Systems and Software Engineering Management using Planguage*. Butterworth-Heinemann, 2005.

[83] M. Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26, 2007.

[84] T. Gorschek. *A Method for Assessing Requirements Engineering PRocess Maturity in Software Projects*. PhD thesis, Depart. of Soft. Engineering and Comp. Science Blekinge Instit. of Technology, 2002.

[85] E. Gottesdiener. *Requirements by Collaboration: Workshops for Defining Needs*. Addison-Wesley Professional, 2002.

[86] R. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice-Hall, 1992.

[87] E. Groen, S. Kopczyńska, M. Hauer, T. D. Krafft, and J. Doerr. Users – the hidden software product quality experts. In *IEEE International Requirements Engineering Conference*, 2017.

[88] E. Groen, N. Seyff, R. Ali, F. Dalpiaz, J. Doerr, E. Guzman, M. Hosseini, J. Marco, M. Oriol, A. Perini, and M. Stade. The crowd in requirements engineering: The landscape and challenges. *IEEE Software*, 34:44–52, 2017.

[89] P. Gruenbacher. Collaborative requirements negotiation with EasyWinWin. In *Database Expert Syst. Appl. Proc. Int. Work.*, pages 954–958, 2000.

[90] E. Guzman, R. Alkadhi, and N. Seyff. A needle in a haystack: What do twitter users say about software? In *24th International Requirements Engineering Conference (RE)*, pages 96–105. IEEE, 2016.

[91] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *22nd International Requirements Engineering Conference (RE)*, pages 153–162. IEEE, 2014.

[92] L. Guzmán, M. Oriol, P. Rodríguez, X. Franch, A. Jedlitschka, and M. Oivo. How can quality awareness support rapid software development? In *REFSQ*, 2017.

[93] J. E. Hannay and H. C. Benestad. Perceived productivity threats in large agile development projects. In *Proc. of Intl Symp. on Empirical Software Engineering and Measurement*. ACM-IEEE, 2010.

[94] N. B. Harrison. A study of extreme programming in a large company. *Avaya Labs*, 2003.

[95] E. Hasnain and T. Hall. Introduction to stand-up meetings in agile methods. In *Proc. of AIP Conference*, volume 1127. AIP, 2009.

[96] B. Haugset and G. K. Hanssen. Automated acceptance testing: A literature review and an industrial case study. In *Proc. of Conference on Agile (AGILE)*. IEEE, 2008.

[97] A. Herrmann, D. Kerkow, and J. Doerr. Exploring the Characteristics of NFR Methods–a Dialogue about two Approaches. In *REFSQ*, pages 320–334. Springer, 2007.

[98] A. Herrmann and B. Paech. Moqare: misuse-oriented quality requirements engineering. *Requirements Engineering*, 13(1):73–86, 2008.

[99] J. Highsmith. *Agile project management: creating innovative products*. Pearson Education, 2009.

[100] P. R. Hill. *Practical Software Project Estimation: A Toolkit for Estimating Software Development Effort & Duration*. McGraw-Hill, 2011.

[101] E. Hochmüller and R. T. Mittermeir. Agile process myths. In *Proc. of Intl workshop on Scrutinizing agile practices or shoot-out at the agile corral*. ACM, 2008.

[102] R. Hoda, J. Noble, and S. Marshall. The impact of inadequate customer collaboration on self-organizing agile teams. *Information and Software Technology*, 53(5), 2011.

[103] H. Hofmann and F. Lehner. Requirements engineering as a success factor in software projects. *IEEE Software*, 18(4):58–66, jul 2001.

[104] M. Hollander and D. A. Wolfe. *Nonparametric statistical methods*. John Wiley & Sons, 1973.

[105] E. Hossain, M. A. Babar, and H.-y. Paik. Using scrum in global software development: a systematic literature review. In *Proc. of 4th Intl Conference on Global Software Engineering (ICGSE)*. IEEE, 2009.

[106] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. Springer, 2005.

[107] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. Springer, 2011.

[108] Z. Hussain, W. Slany, and A. Holzinger. Current state of agile user-centered design: A survey. In *Proc. of 5th Symposium of the Workgroup HCI and Usability Engineering of the Austrian Computer Society on HCI and Usability for e-Inclusion*. Springer, 2009.

[109] IEEE. IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1998*, 1998.

[110] IEEE. IEEE Systems and software engineering – Life cycle processes –Requirements engineering. *IEEE Std 29148-2011*, pages 1–94, 2011.

[111] S. Ilieva, P. Ivanov, and E. Stefanova. Analyses of an agile methodology implementation. In *Proc. of 30th Euromicro Conference*. IEEE, 2004.

[112] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband. A systematic literature review on agile requirements engineering practices and challenges. *Computers in human behavior*, 51, 2015.

[113] International Organization for Standardization (ISO). *ISO/IEC 25010:2011 - Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE)*. International Organization for Standardization, 2011.

[114] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.

[115] ISO/IEC/IEEE. 24765: 2010 systems and software engineering-vocabulary. Technical report, IEEE, 2010.

[116] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1999.

[117] S. Jalali and C. Wohlin. Agile practices in global software engineering-a systematic map. In *Proc. of 5th Intl Conference on Global Software Engineering (ICGSE)*. IEEE, 2010.

[118] S. Jalali and C. Wohlin. Global software engineering and agile practices: a systematic review. *Journal of Software: Evolution and Process*, 24(6), 2012.

[119] I. John. *Pattern-based documentation analysis for software product lines*. University of Kaiserslautern, 2010.

[120] J. Jurkiewicz and J. Nawrocki. Automated events identification in use cases. *Information and Software Technology*, 58:110–122, 2015.

[121] M. Kajko-Mattsson. Problems in agile trenches. In *Proc. of 2nd ACM-IEEE Intl Symp. on Empirical Software Engineering and Measurement*, 2008.

[122] M. Kalinowski, M. Felderer, T. Conte, R. Spinola, R. Prikladnicki, D. Winkler, D. M. Fernández, and S. Wagner. Preventing incomplete/hidden requirements: reflections on survey data from Austria and Brazil. In *Int. Conf. Softw. Qual.*, pages 63–78. Springer, 2016.

[123] M. Kassab. An empirical study on the requirements engineering practices for agile software development. In *Proc. of 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2014.

[124] M. Kassab. The changing landscape of requirements engineering practices over the past decade. In *Proc. of 5th Intl Workshop on Empirical Requirements Engineering (EmpiRE)*. IEEE, 2015.

[125] M. Kassab, C. Neill, and P. Laplante. State of practice in requirements engineering: contemporary data. *Innovations in Systems and Software Engineering*, 10(4), 2014.

[126] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong. Robust statistical methods for empirical software engineering. *Empirical Software Engineering*, 22(2), 2017.

[127] B. A. Kitchenham and S. L. Pfleeger. *in: Guide to advanced empirical software engineering*, volume 93, chapter Personal Opinion Surveys. F. Shull, J. Singer and D. Sjøberg,. Springer, 2008.

[128] S. Kopczyńska, M. Maćkowiak, and J. Nawrocki. Structured meetings for non-functional requirements elicitation. *Foundat. of Computing and Decision Sciences*, 36(1):35–56, 2011.

[129] S. Kopczyńska and J. Nawrocki. Using non-functional requirements templates for elicitation: A case study. In *IEEE Int. Workshop Requirements Patterns*, 2014.

[130] S. Kopczyńska and J. Nawrocki. HAZOP-based Approach to Pattern Identification for Non-functional Requirements. In *Int. Workshop Requir. Patterns (RePa 2015)*, pages 39–46. IEEE, 2015.

[131] S. Kopczyńska, J. Nawrocki, and M. Ochodek. Software development studio - Bringing industrial environment to a classroom. In *Porc. of Software Engineering Education based on Real-World Experiences (EduRex), 2012 First International Workshop on*, pages 13–16. IEEE, 2012.

[132] S. Kopczyńska, J. Nawrocki, and M. Ochodek. An empirical study on catalog of non-functional requirement templates: Usefulness and maintenance issues. *Information and Software Technology*, 2018.

[133] M. Korkala, M. Pikkarainen, and K. Conboy. A case study of customer communication in globally distributed software product development. In *Proc. of 11th Intl Conf. on Product Focused Software*. ACM, 2010.

[134] L. Koskela. *Test Driven: Practical TDD and Acceptance TDD for Java Developers*. Manning Publications Co., 2007.

[135] G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley Publishing, 1st edition, 1998.

[136] N. A. Koziol and C. R. Bilder. MRCV: A Package for Analyzing Categorical Variables with Multiple Response Options. *The R Journal*, 6(1), 2014.

[137] H. L. Kundel and M. Polansky. Measurement of observer agreement 1. *Radiology*, 228(2):303–308, 2003.

[138] N. Kurapati, V. S. C. Manyam, and K. Petersen. Agile software development practice adoption survey. *Agile processes in software engineering and extreme programming*, 2012.

[139] K. Kuusinen, T. Mikkonen, and S. Pakarinen. Agile user experience development in a large software organization: good expertise but limited impact. *Human-Centered Software Engineering*, 2012.

[140] Laboratory of Software Engineering at Institute of Computing Science, Poznan University of Technology . Website of NoRTs, available at: `http://norts.cs.put.poznan.pl/experiment`, last accessed 2018-10-31.

[141] W. Lam, T. McDermid, and A. Vickers. Ten steps towards systematic requirements reuse. In *Intenatl. Symposium on Requirements Engineering*, pages 6–15. IEEE, 1997.

[142] D. Landes and R. Studer. The treatment of non-functional requirements in mike. In *European Software Engineering Conference*, pages 294–306. Springer, 1995.

[143] B. Lawrence, K. Wiegers, and C. Ebert. The top risk of requirements engineering. *IEEE Software*, 18(6):62–63, 2001.

[144] J. C. Lee, D. S. McCrickard, and K. T. Stevens. Examining the foundations of agile usability with extreme scenario-based design. In *Proc. of Agile Conference (AGILE)*. IEEE, 2009.

[145] D. R. Lindstrom. Five ways to destroy a development project. *IEEE Software*, 10(5):55–58, 1993.

[146] C. Lowell and J. Stell-Smith. Successful automation of gui driven acceptance testing. In *Extreme programming and agile processes in software engineering*. Springer, 2003.

[147] Y. Luo, L. Sterling, and K. Taveter. Modelling a smart music player with a hybrid agent-oriented methodology. In *15th International Requirements Engineering Conference*, pages 281–286. IEEE, 2007.

[148] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *23rd International Requirements Engineering Conference (RE)*, pages 116–125. IEEE, 2015.

[149] D. Mairiza, D. Zowghi, and N. Nurmuliani. An investigation into the notion of non-functional requirements. In *ACM Symposium on Applied Computing*. ACM, 2010.

[150] C. Mann and F. Maurer. A case study on the impact of scrum on overtime and customer satisfaction. In *Proc. of Agile Conference*. IEEE, 2005.

[151] A. Martin, J. Noble, and R. Biddle. Being Jane Malkovich: A look into the world of an XP customer. In *Extreme Programming and Agile Processes in Software Engineering*. Springer, 2003.

[152] R. C. Martin and G. Melnik. Tests and requirements, requirements and tests: A möbius strip. *IEEE Software*, 25(1), 2008.

[153] V. Massol and T. Husted. *Junit in action*. Manning Publications Co., 2003.

[154] A. Mavin and P. Wilkinson. Big Ears (The Return of "Easy Approach to Requirements Engineering"). In *Requirements Engineering Conf.*, pages 277–282, 2010.

[155] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. Easy Approach to Requirements Syntax (EARS). In *17th IEEE Int. Requirements Engineering Conference*, pages 317–322, 2009.

[156] J. A. McCall, P. K. Richards, and G. F. Walters. Factors in software quality. volume i. concepts and definitions of software quality. Technical report, GENERAL ELECTRIC CO SUNNYVALE CA, 1977.

[157] T. Memmel, F. Gundelsweiler, and H. Reiterer. Agile human-centered software engineering. In *Proc. of 21st British HCI Group Annual Conf. on People and Computers: HCI... but not as we know it*. British Computer Society, 2007.

[158] T. Merten, M. Falis, P. Hübner, T. Quirchmayr, S. Bürsner, and B. Paech. Software feature request detection in issue tracking systems. In *24th International Requirements Engineering Conference (RE)*, pages 166–175. IEEE, 2016.

[159] B. Meyer. *Agile!: The Good, the Hype and the Ugly*. Springer Science & Business Media, 2014.

[160] R. E. Miller. *The Quest for Software Requirements*. MavenMark Books, 2009.

[161] S. Mohammadi, B. Nikkhahan, and S. Sohrabi. An analytical survey of" on-site customer" practice in extreme programming. In *Proc. of Intl Symp. on Computer Science and its Applications*. IEEE, 2008.

[162] S. Mohammadi, B. Nikkhahan, and S. Sohrabi. Challenges of user involvement in extreme programming projects. *International Journal of Software Engineering and Its Applications*, 3(1), 2009.

[163] G. A. Moore. *Crossing the Chasm: Marketing and selling high-tech goods to mainstream customers*. New York, HarperBusiness, 1991.

[164] L. Morris, M. Ma, and P. Wu. *Agile Innovation: The Revolutionary Approach to Accelerate Success, Inspire Engagement, and Ignite Creativity. 2014*. New York: Wiley, 2014.

[165] R. Mugridge and W. Cunningham. *Fit for developing software: framework for integrated tests*. Pearson Education, 2005.

[166] R. Muñoz and H. H. Miller-Jacobs. In search of the ideal prototype. In *Proc. of SIGCHI conference on Human factors in computing systems*. ACM, 1992.

[167] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on software engineering*, 18(6):483–497, 1992.

[168] J. Naish and L. Zhao. Towards a generalised framework for classifying and retrieving requirements patterns. In *RePa*, pages 42–51, 2011.

[169] J. Nawrocki, L. Olek, M. Jasinski, B. Paliświat, B. Walter, B. Pietrzak, and P. Godek. Balancing agility and discipline with XPrince. In *Proceedings of RISE 2005 Conference (in print), volume 3943 of LNCS*, pages 266–277. Springer Verlag, 2006.

[170] N. Nazir, N. Hasteer, and A. Bansal. A survey on agile practices in the indian it industry. In *Proc. of 6th Intl Conference on Cloud System and Big Data Engineering (Confluence)*. IEEE, 2016.

[171] C. Ncube. *A requirements engineering method for COTS-based systems development*. PhD thesis, City University London, 2000.

[172] K. A. Neuendorf. *The content analysis guidebook*. Sage, 2016.

[173] B. Nuseibeh. Ariane 5: Who dunnit? *IEEE Software*, 14(3):15–16, 1997.

[174] J. N. och Dag, B. Regnell, P. Carlshamre, M. Andersson, and J. Karlsson. A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1):20–33, 2002.

[175] M. Ochodek, B. Alchimowicz, J. Jurkiewicz, and J. Nawrocki. Improving the reliability of transaction identification in use cases. *Information and Software Technology*, 53(8):885–897, 2011.

[176] M. Ochodek and S. Kopczyńska. Perceived importance of agile requirements engineering practices – a survey. *Journal of Systems and Software*, 143, 2018.

[177] P. N. Otto and A. I. Antón. Addressing legal requirements in requirements engineering. In *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, pages 5–14. IEEE, 2007.

[178] B. Paech and D. Kerkow. Non-functional requirements engineering-quality is essential. In *10th International Workshop on Requirments Engineering Foundation for Software Quality*, 2004.

[179] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *21st IEEE International Requirements Engineering Conference (RE)*, pages 125–134. IEEE, 2013.

[180] T. Paivarinta and K. Smolander. Theorizing about software development practices. *Science of Computer Programming*, 101, 2015.

[181] C. Palomares, C. Quer, and X. Franch. Pabre-man: Management of a requirement patterns catalogue. In *Requirements Engineering Conference (RE), 2011 19th IEEE International*, pages 341–342. IEEE, 2011.

[182] C. Palomares, C. Quer, and X. Franch. Requirements reuse and requirement patterns: a state of the practice survey. *Empirical Software Engineering*, pages 1–44, 2015.

[183] C. Palomares, C. Quer, X. Franch, C. Guerlain, and S. Renault. A catalogue of non-technical requirement patterns. In *Requirements Patterns (RePa), 2012 IEEE Second International Workshop on*, pages 1–6. IEEE, 2012.

[184] C. Palomares, C. Quer, X. Franch, S. Renault, and C. Guerlain. A catalogue of functional software requirement patterns for the domain of content management systems. In *Proceedings of the 28th annual acm symposium on applied computing*, pages 1260–1265. ACM, 2013.

[185] S. Panichella, E. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *IEEE international conference on Software maintenance and evolution (ICSME)*, pages 281–290. IEEE, 2015.

[186] E. Papatheocharous and A. S. Andreou. Empirical evidence and state of practice of software agile teams. *Journal of Software: Evolution and Process*, 26(9), 2014.

[187] C. Patel and M. Ramachandran. Agile Maturity Model (AMM): A Software Process Improvement framework for Agile Software Development Practices. *International Journal of Software Engineering*, 2(1), 2009.

[188] R. Pichler. *Agile product management with scrum: Creating products that customers love*. Addison-Wesley Professional, 2010.

[189] A. Piechowiak. Archiwum dokumentów elektronicznych(ADE). available online: `http://www.i3conference.net/online/2009/prezentacje/Archiwum_Dokumentow_Elektronicznych.pdf`, last access 06 August 2018, 2009.

[190] K. Pohl. *Requirements Engineering*. Springer, 1998.

[191] K. Pohl. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[192] K. Pohl and C. Rupp. *Requirements Engineering Fundamentals*. Rocky Nook, 2011.

[193] K. Power. Definition of ready: An experience report from teams at cisco. In *Agile Processes in Software Engineering and Extreme Programming*, pages 312–319. Springer, 2014.

[194] R. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, USA, 7 edition, 2010.

[195] Project Management Institute. Success rates rise. transforming the high cost of low performance, 2017.

[196] T. Punter, M. Ciolkowski, B. Freimut, and I. John. Conducting on-line surveys in software engineering. In *Proc. of Intl Symposium on Empirical Software Engineering, (ISESE)*. IEEE, 2003.

[197] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[198] R Documentation. Box Plots, available online at: `www.rdocumentation.org/packages/graphics/versions/3.5.1/topics/boxplot`, last accessed 28th September 2018.

[199] D. M. Rafi, K. R. K. Moses, K. Petersen, and M. V. Mäntylä. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *Proc. of 7th Intl Workshop on Automation of Software Test*. IEEE, 2012.

[200] B. Regnell, R. B. Svensson, and T. Olsson. Supporting roadmapping of quality requirements. *IEEE Software*, 25(2):42–47, 2008.

[201] S. Renault, Ó. Méndez-Bonilla, X. Franch, and C. Quer. Pabre: pattern-based requirements elicitation. In *Research Challenges in Information Science, 2009. RCIS 2009. Third International Conference on*, pages 81–92. IEEE, 2009.

[202] Reuse Company Products. see `https://www.reusecompany.com/mission-vision`.

[203] M. Riaz, J. King, J. Slankas, L. Williams, F. Massacci, C. Quesada-López, and M. Jenkins. Identifying the implied: Findings from three differentiated replications on the use of security requirements templates. *Empirical Software Engineering*, 2016.

[204] S. Robertson and J. Robertson. *Mastering the Requirements Process: Getting Requirements Right (3rd Edition)*. Addison-Wesley, 2012.

[205] P. Rodríguez, J. Markkula, M. Oivo, and K. Turula. Survey on agile and lean usage in finnish software industry. In *Proc. of Intl Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM-IEEE, 2012.

[206] D. Root, M. Rosso-Llopart, and G. Taran. Proposal based studio projects: How to avoid producing "cookie cutter" software engineers. In *Proc. of the 21st CSEET*, pages 145–151. IEEE, 2008.

[207] D. Rosenberg and M. Stephens. *Extreme programming refactored: the case against XP*. APress, 2003.

[208] K. S. Rubin. *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley, 2012.

[209] P. Runeson, M. Host, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley, 2012.

[210] D. Salah, R. F. Paige, and P. Cairns. A systematic literature review for agile development processes and user centred design integration. In *Proc. of 18th Intl Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014.

[211] O. Salo and P. Abrahamsson. Agile methods in european embedded software development organisations: a survey on the actual use and usefulness of extreme programming and scrum. *IET software*, 2(1), 2008.

[212] F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang. Feature lifecycles as they spread, migrate, remain, and die in app stores. In *23rd International Requirements Engineering Conference*, pages 76–85. IEEE, 2015.

[213] K. Schwaber and J. Sutherland. The Scrum Guide™. The Definitive Guide to Scrum: The Rules of the Game. Scrum.org, 2013.

[214] C. Seaman. Qualitative methods. In Shull, F. et al., editor, *Advanced Empirical Software Engineering*, pages 35–62. Springer, 2008.

[215] P. Sfetsos and I. Stamelos. Empirical studies on quality in agile practices: A systematic literature review. In *Proc. of 7th Intl Conference on Quality of Information and Communications Technology (QUATIC)*. IEEE, 2010.

[216] D. Shahane, P. Jamsandekar, and D. Shahane. Factors influencing the agile methods in practice-literature survey & review. In *Proc. of Intl Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2014.

[217] A. Solinski and K. Petersen. Prioritizing agile benefits and limitations in relation to practice usage. *Software Quality Journal*, 24(2), 2016.

[218] C. Solís and X. Wang. A study of the characteristics of behaviour driven development. In *Proc. of 37th EUROMICRO Conf. on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2011.

[219] I. Sommerville. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[220] I. Sommerville and P. Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, 1997.

[221] X. Song, B. Hwong, and J. Ros. Experiences in developing quantifiable nfrs for the service-oriented software platform. In *17th IEEE International Requirements Engineering Conference*, pages 337–342. IEEE, 2009.

[222] R. Stacey. *Strategic management and organizational dynamics: the challenge of complexity*. Harlow: Prentice Hall, 2002.

[223] Standish Group. Chaos Report. available online at `https://www.projectsmart.co.uk/white-papers/chaos-report.pdf`, last access: 24/07/2018, 1995.

[224] D. Stankovic, V. Nikolic, M. Djordjevic, and D.-B. Cao. A survey study of critical success factors in agile software projects in former yugoslavia it companies. *Journal of Systems and Software*, 86(6), 2013.

[225] S. Stavru. A critical examination of recent industrial surveys on agile method usage. *Journal of Systems and Software*, 94, 2014.

[226] M. Strohmaier, J. Horkoff, and E. Yu. Can Patterns improve i* Modeling? Two Exploratory Studies. In *REFSQ*, pages 153–167. Springer, 2008.

[227] S. Supakkul, T. Hill, L. Chung, T. T. Tun, and J. C. S. do Prado Leite. An nfr pattern approach to dealing with nfrs. In *Requirements Engineering Conference (RE)*, pages 179–188. IEEE, 2010.

[228] R. B. Svensson, M. Host, and B. Regnell. Managing quality requirements: A systematic review. In *Software Engineering and Advanced Applications, EUROMICRO Conference on*, pages 261–268. IEEE, 2010.

[229] N. Team. Naming the pain in requirements engineering - a survey `http://napire.org`, last visited 27 august 2018, 2018.

[230] A. Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 249–262. IEEE, 2001.

[231] VersionOne inc. The 11th annual state of agile report. Technical report, VersionOne inc., 2017.

[232] K. Vlaanderen, S. Jansen, S. Brinkkemper, and E. Jaspers. The agile requirements refinery: Applying scrum principles to software product management. *Information and software technology*, 53, 2011.

[233] B. Wake. INVEST in good stories and SMART tasks. `http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/`, last access: 25/01/2018, 2003.

[234] C. Wang. Software development studio. Technical report, The Univ. of Texas at Austin, 2004.

[235] X. Wang, L. Zhao, Y. Wang, and J. Sun. The role of requirements engineering practices in agile development: An empirical study. In *Proc. of 1st Asia Pacific Requirements Engineering Symposium, APRES*. Springer-Verlag Berlin Heidelberg, 2014.

[236] K. Wiegers. *Requirements Enineering 2nd*. Microsoft Press, 2003.

[237] K. Wiegers. *Software Requirements*. Microsoft Press, 2003.

[238] K. Wiegers and J. Beatty. *Software Requirements*. Microsoft Press, 3rd edition, 2013.

[239] L. Williams. What agile teams think of agile principles. *Communications of the ACM*, 55(4), 2012.

[240] S. Withall. *Software Requirement Patterns (Developer Best Practices)*. Microsoft Press, 2007.

[241] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[242] J. Wood and D. Silver. *Joint application development*. Wiley, 1995.

[243] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2), 2012.

[244] R. R. Young. *The requirements engineering handbook*. Artech House, 2004.

[245] E. S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, pages 226–235. IEEE, 1997.

[246] N. Yusop, D. Zowghi, and D. Lowe. The impacts of non-functional requirements in web system projects. *Int. J. Value Chain Manag.*, 2(1):18–32, 2007.

Index

Institute of Computing Science
Poznań University of Technology, Poznań, Poland

Typeset using LaTeX in Adobe Utopia.

BibTeX entry:

```
@phdthesis{ kopczynska2018PhD,
        author = "Sylwia Kopczyńska",
        title = "{Supporting Non-functional Requirements Elicitation with Templates}",
        school = "Poznań University of Technology",
        address = "Poznań, Poland",
        year = "2018"
}
```