Requirements Engineering: Best Practice

Samuel A. Fricker[1], Rainer Grau[2], and Adrian Zwingli[3]

[1]Software Engineering Research Laboratory, Blekinge Institute of Technology, Sweden
[2]Zühlke Engineering AG, Switzerland
[3]SwissQ Consulting AG, Switzerland
{samuel.fricker@bth.se, rainer.grau@zuehlke.com,
adrian.zwingli@swissq.it}

Abstract. Many software solutions have failed because they did not meet stakeholder needs. In response to this problem a massive amount of techniques were developed to elicit stakeholder needs, to analyze the implications of these needs on the software, to specify proposed software products, and to check acceptance of these proposals. However, many of these techniques did not become industrial practice because they were not practicable or ineffective when used in real-world projects. To obtain an overview of what common practice is and to understand which techniques reflect best practice because they are particularly effective, we have surveyed a large number of industry projects. Based on 419 valid answers, this chapter gives an overview of commonly used requirements engineering techniques. It also shows which of the techniques, when used in a software project, correlate with requirements engineering success. The chapter concludes with recommendations for software projects and future research to improve requirements engineering practice.

1 Introduction

In 1995 the consultancy company Standish Group International published results of an industry survey that showed that only 16% of the software projects were successful, 53% were challenged, and 31% complete failures (Standish Group International 1995). Successful projects were those that completed on time and budget and produced a software product with all features and functions as initially specified. The low success rates described in the Standish report generated substantial attention by industry and politics.

The Standish survey pointed to software project practices that needed improvement. According to the respondents, the most frequently stated factors that influenced project success were user involvement, executive management support, and a clear statement of requirements. These factors show that requirements engineering is crucial to achieve project success. User involvement is critical for building a software that will be understood by the users, that will be used appropriately, and that creates joy (Hassenzahl, Beu et al. 2001). Management support is critical to align the software with the strategic goals of the organization (Gorschek and Wohlin 2006). Clearly stated requirements contribute to a shared understanding between the project team and the software product's users, management, and other stakeholders. The shared understanding reduces the

risk of unsatisfactory outcome and rework of project results (Glinz and Fricker 2013). Influenced by these insights, software engineering practice matured over time. 32% of the software projects were successful according to the Standish survey published in 2009 (Eveleens and Verhoef 2010).

Even-though many requirements engineering techniques exist for involving users, for obtaining management support, and for achieving shared understanding, we lack an understanding of whether these techniques make requirements engineering successful. Some researchers believe that no technique would do and claim that good requirements practices are neither sufficient nor necessary (Davis and Zowghi 2006). The best we can say today is that the techniques are used inconsistently: some techniques get used by some projects but not by others (Neill and Laplante 2003, Paech, Koenig et al. 2005). Companies that care about requirements engineering seem have a preference for Quality Function Deployment, prototyping, Data Flow Diagrams, role playing, and decision trees (Rouibah and Al-Rafee 2009). However, we do not know whether any of these techniques correlates with requirements engineering success, thus should be used systematically.

This chapter intends to develop an understanding of what practice makes requirements engineering successful by reporting the results from an own large-scale industry survey. The survey investigated whether the use of requirements engineering techniques differed between projects with successful and unsuccessful requirements engineering. The results show that a few techniques indeed correlated with success. In addition, also the ability to apply a broad variety of requirements engineering techniques is important. These results imply that best practice would be to utilize the few effective techniques and pragmatically select complementing techniques that suit well the type of software being developed and the situation that requirements engineering is confronted with.

The remainder of this chapter is structured as follows. Section 2 gives an overview of requirements engineering state-of-art that was studied in the industry survey. Section 3 describes the survey methodology. Section 4 characterizes the projects that have responded to the survey, gives an overview of the requirements engineering practice of these projects, and shows the correlation of requirements engineering practice with success. Section 5 discusses the obtained results, gives recommendations for practice, and suggests implications for research. Section 6 summarizes and concludes.

2 Requirements Engineering State-of-Art

2.1 Requirements Engineering Techniques

There is a long tradition of research and practice in requirements engineering. One of the early influential works describes requirements engineering as inquiry (Potts, Takahashi et al. 1994). During an inquiry the requirements engineer asks questions about a future software product to stakeholders and turns the obtained answers into a specification. While doing so, new questions emerge that are posed again to the stakeholders to initiate the next inquiry.

Since these early days, a large number of techniques have been investigated to advance requirements engineering state-of-the-art (Cheng and Atlee 2007). Still, Potts's inquiry remains a good model of how to think of requirements engineering in a software project. Today, a requirements engineer is expected to elicit needs and expectations from stakeholders, to model and analyze the impact of these inputs on the system together with the development team, and to check proposed implementations for acceptance by the stakeholders (Pohl and Rupp 2011). Once both the stakeholders and the development team agree, the requirements are used to steer development and, upon release of the solution, check whether the developed product fulfils the agreement. If the inquiry is done well, one can observe that a shared understanding emerges, requirements stabilize, and the stakeholders become satisfied (Fricker and Glinz 2010, Glinz and Fricker 2014).

During elicitation the requirements engineer aims at understanding the project vision and constraints, the context that the product will be deployed into, and the stakeholders that will need to accept the product (Hickey and Davis 2004, Zowghi and Coulin 2005). Such requirements elicitation results in an overview of users, external systems, and other stakeholder viewpoints and a description of their respective background, interests, and expectations. A large number of techniques are known to elicit such information about the system requirements (Hickey and Davis 2003, Davis, Dieste et al. 2006, Dieste, Juristo et al. 2008). Table 1 gives an overview of selected elicitation techniques.

Table 1: Selected requirements elicitation techniques.

Technique	**Description**
Archaeology	Analysis of existing systems to understand their functionality, quality, and usage (Pérez-Castillo, Garciía-Rodríguez de Guzmán et al. 2011).
Creativity	The generation and selection of ideas to innovate or solve a difficult problem (Maiden, Gizikis et al. 2004, Gorschek, Fricker et al. 2010).
Data Mining	Search and filtering of requirements databases to identify relevant knowledge about stakeholder needs (Cleland-Huang and Mobasher 2008).
Interview	Meeting between a requirements engineer and a stakeholder to discuss topics of relevance for the system (Alvarez and Urla 2002).
Introspection	Use of domain knowledge in combination with reflection and empathy to base requirements on experience (Vermersch 2009).
Observation	Study of system use, possibly in the target environment and by real users, to understand usage processes and strengths and weaknesses of a current system (Beyer and Holtzblatt 1995).
Questionnaire-based Survey	Paper or electronic form with questions and space for answers distributed to stakeholders to obtain an overview of stakeholder opinion (Ng, Barfield et al. 1995).
Reuse	Use of existing specifications to avoid re-invention of requirements that already are adequate (Lam, McDermid et al. 1997).
Workshop	Meeting between a requirements engineer and stakeholders to reach agreement between the workshop participants (Gottesdiener 2002).

During analysis the requirements engineer aims at understanding how the requirements will be implemented by the software system (Mylopoulos, Chung et al. 1999), how they will be considered in the development plan (van de Weerd, Brinkkemper et al. 2006), and how they will be used for the testing of the system (Martin and Meinik 2008). Requirements analysis typically results in one or more prototypes, a definition of project scope or release plan, and a requirements specification for the system. Table 2 gives an overview of selected analysis techniques, Table 3 of planning techniques, Table 4 of relevant requirements types, and Table 5 of specification techniques.

Table 2: Selected system analysis techniques.

Technique	**Description**
Domain-driven Development	Specifying the concepts of relevance in the context the system will be deployed into that are to be implemented or respected by the system (Danevas and Garva 2012).
Formal Specification	Use of mathematical or formal-logic expressions to enable automated checking of completeness consistency, and correctness (Holtmann, Meyer et al. 2011).
Informal Modelling	Sketching a model of something of relevance to reflect and discuss how the parts of that thing interrelate (Glinz 2010).
OOA	Specifying the structure, functionality, and behaviour of the system usually with the object-oriented analysis language UML (Arlow and Neustadt 2005).
Prototyping	Paper- or tool-based approximation of the end-systems to increase the tangibility and authenticity of the planned system (Rettig 1994).
Quality Checks	Checking whether the system fulfils its goals and whether functionality and quality are adequate and needed (Chung, Nixon et al. 2000).
SA	Specifying the structure, functionality, and behaviour of the system with a structured analysis language (Ross 1977, DeMarco 1979).

Table 3: Selected requirements planning techniques.

Technique	**Description**
Business Case	Evaluating whether a set of requirements has to good return-of-investment and should be included into project scope (Schmidt 2002).
Prioritizing	Ranking the requirements to obtain an order of how they shall be addressed by the project work (Achimugu, Selamat et al. 2014).
Release Planning	Defining the contents of one or more releases to define the scope of the software system (Svahnberg, Gorschek et al. 2010).
Roadmapping	Corse-grained, long-term planning to agree with stakeholders and suppliers for how the software system shall evolve (Phaal, Farrukh et al. 2003, Fricker and Schumacher 2012).
Triage	Filtering the requirements to determine what requirements are relevant and what requirements are not (Davis 2005).
Vision	Defining the problem that is addressed, the key idea of the solution, and how the solution improves state-of-art to align the work of developers and stakeholders (McGrath 2001).

Table 4: Selected requirement types.

Type	Description
Behaviour	Behaviour is a sequence of states that determine how a system, artefact, or class reacts to events (Whittle and Schumann 2000).
Formal Property	A formal property can be tested for correctness, completeness, and consistency with automated tools (Berard, Bidoit et al. 2010).
Function	Function is a reaction to inputs or an action of a system (IEEE 1990).
Glossary	A glossary defines terms, abbreviations, acronyms, synonyms, and homonyms (Pohl and Rupp 2011).
Interface	An interface connects a system with its environment. Typical interfaces are user interfaces (Myers 1989) and interfaces to other software systems (Chung, Lin et al. 2003).
Process	A process is a series of actions or operations implemented by people, organizations, or software to achieve a goal (Melão and Pidd 2000).
Quality	Quality is a characteristic of a software system such as performance, reliability, security, compatibility, portability, usability, and maintainability (ISO/IEC 2010).
Scenario	A scenario is a story of how users and systems interact to achieve a goal (Alexander and Maiden 2005).
Stakeholder	A person, group, or organization who gains or loses something with the software (Alexander and Robertson 2004). May be denoted agent (van Lamsweerde 2001) or actor (Arlow and Neustadt 2005).
Structure	Structure refers to entities or systems with their attributes and relationships (Arlow and Neustadt 2005).

Table 5: Selected specification techniques.

Technique	Description
i* or KAOS	Specifying agents, goals, and formal properties with formal languages to enable reasoning about goals and goal-achievement (van Lamsweerde 2001).
Natural Language	Specifying requirement with words and sentences to achieve specification flexibility and understandability. Language templates may be used to improve precision (Denger, Berry et al. 2003).
SA Diagrams	Specifying functions, processes, structure, and behaviour with one of the graphical notations proposed by structured analysis to achieve precision and make structure visible.
Tables	Specifying concepts to achieve an understanding of the terminology (Dwarakanath, Ramnani et al. 2013) and or rules for how conditions affect system behaviour (Bajec and Krisper 2005).
UML Diagrams	Specifying functions, scenarios, processes, rules, relations, behaviour, and deployment with graphical notations from the Unified Modelling Language to increase precision and show structure.
User Screens	Specifying the user interface with paper or tool-based mock-ups to increase the tangibility and authenticity of the planned system.

During requirements checking, the requirements engineer checks that the right approach has been selected for fulfilling the vision and achieving the system goals and that the system will be accepted by the stakeholders. Requirements checking initiates a new inquiry cycle if the checked requirements turn out to be not good-enough. Requirements checking marks the agreement of the stakeholders on contents and scope of the development project if the checking has been successful. Table 6 gives an overview of selected checking techniques.

Table 6: Selected checking techniques.

Technique	**Description**
Automated Checking	Testing a formal specification of the system to detect conflicting and missing requirements (Easterbrook, Lutz et al. 1998).
Inspection	Review of the requirements specification by all relevant stakeholders with a formal process that is effective at discovering problems and leads to in-depth understanding of the specification (Porter, Votta et al. 1995).
Peer review	Feedback by one or more requirements engineers to support and assure the quality of the specification work.
Prototype review	Discussion and use of the prototype, for example in a role-play, to explore uses and check acceptance of the system.
Simulation	Approximation and review of the behaviour of the system with an appropriate tool to check correctness of the behaviour (Glinz, Seybold et al. 2007).
Walk-through	Efficient review of the requirements specification by discussing the requirements specification in their sequence with stakeholders.

The inquiry cycle leads to a dialogue between stakeholders and development team that can be seen as a negotiation (Fricker 2009). The negotiation results in an agreement between the stakeholders and the development team about the product to be developed. This agreement, represented by the approved requirements specification, is then baselined and used to manage the development project and the release of the developed product. Table 7 gives an overview of selected requirements negotiation techniques and Table 8 of requirements management techniques.

Table 7: Selected requirements negotiation techniques.

Technique	**Description**
Conflict Management	Discovering and resolving conflicts among stakeholders and between stakeholders and development team (Pohl and Rupp 2011).
Handshaking	The review and discussion of implementation proposals to align the planned implementation of the software system with stated and unstated stakeholder needs (Fricker, Gorschek et al. 2010).
Negotiation Analysis	Analysing possible negotiation outcomes and selecting a value-creating, fair agreement (Raiffa 2007).
Power Analysis	Analysing power and influence of stakeholders and planning how to interact with them (Milne and Maiden 2012).

Technique	Description
Prioritizing	Ranking the requirements to obtain an order of how they shall be addressed by the project work (Achimugu, Selamat et al. 2014).
Strategy Alignment	Aligning requirements with company strategy, for example through explicit traceability (Gorschek and Wohlin 2006).
Variant Analysis	Analysing and selecting alternative features or ways of solving a problem (Schobbens, Heymans et al. 2007).
Win-Win Negotiation	Structured, possibly tool-supported approach to identification of options for agreement and selection of the appropriate option (Boehm, Grünbacher et al. 2001).

Table 8: Selected requirements management techniques.

Technique	Description
Baselining	Versioning requirements and specifications and communicating these as a baseline to stakeholders (Conradi and Westfechtel 1998).
Change Management	Controlled process of collecting change requests, analysing impact, and deciding about the change (Kobayashi and Maekawa 2001).
Process Measurement	Measuring requirements engineering and implementation efficiency, for example in the form of value stream analysis (Petersen and Wohlin 2010).
Progress Tracking	Monitoring the lifecycle of requirements from discovery to selection, implementation, and release (Kniberg and Skarin 2010).
Report Generation	Generation of reports, such as requirements specifications, from a database of requirements.
Traceability Management	Maintaining relationships between requirements and possibly other artefacts to express dependencies, conflicts, and synergies (Cleland-Huang, Settimi et al. 2007).

2.2 Requirements Engineering Success

For evaluating how requirements engineering practices are one needs to understand how to measure requirements engineering success. The most through study that answered this question was a survey that tested 32 indicators with 30 requirements engineering experts (El Emam and Madhavji 1985). It showed that requirements engineering success can be measured with *quality of requirement engineering service* and *quality of requirements engineering products*. Table 9 gives an overview of the indicators.

Table 9. Success measurements for requirements engineering (El Emam and Madhavji 1985).

Quality of RE Service	Quality of RE Products
Business-technical alignment: fit with strategy, ability and willingness to make business changes, and management support. *Stakeholder acceptance*: awareness of business changes, extent of consensus, willingness to defend solution, and relationship to users.	*Quality of cost/benefits analysis*: completeness and coverage of cost/benefit analysis, new benefits created by the new solution, and sufficient accuracy of cost estimates. *Argumentation of impact*: diagnosis of existing solution, traceability of supported processes to problem to be solved and to system goals, and traceability of strengths and weaknesses of new solution to replaced solution.

The quality of requirements engineering service refers to the effects a requirements engineer wants to achieve. These concern the alignment of the software product with business objectives and the alignment of it with stakeholder needs and expectations. Such alignment can be checked by asking the concerned stakeholders of whether they agree that system will deliver the desired impacts.

The quality of requirements engineering products refers to the work results delivered by the requirements engineer. These should include a comprehensive cost/benefit analysis and a description of impact with detailed traceability to supported processes, system goals, and the replaced solution. Such work results are tangible and can be easily inspected if they are presented in the form of a requirements specification.

In this chapter we use El Emam and Madhavji's success measurements to inquire what requirements engineering goals were important and whether these goals were achieved. This way of assessing the quality of requirements engineering service and products allows taking into consideration the many possible variations of what is important in given projects. It allows the respondents to judge whether requirements engineering was successful according to their own specific contexts.

El Emam and Madhavji's success measurement have the advantage of being measurable immediately when requirements engineering is concluded. However, they fall short in capturing the ultimate objective of requirements engineering. No measurement has been proposed to assess whether the specified system will be successful. For that reason, we extend the measurement framework outlined in Table 9 with the additional dimension of requirements engineering outcome. In the survey we thus ask the respondents whether the specified product met the goals the product was conceived for.

3 Industry Survey

We investigated the use of requirements engineering techniques and how much they contributed to requirements engineering success with an online survey (Rea and Parker 2005). We distributed an online questionnaire to people involved in software projects in an attempt to answer the following main research questions.

- RQ1: What requirements engineering techniques are used in software projects?
- RQ2: What are the goals pursued in requirements engineering?
- RQ3: Which requirements engineering techniques correlate with requirements engineering success?

The answers to RQ1 show how frequently each of the requirements engineering techniques is used, thus allows us to say what common practice is. Besides benchmarking practice, these results allow judgment whether techniques that were investigated in research were successfully transferred or not. The answers to RQ2 tell us what requirements engineers try to achieve with their work and with the systems they specify. The results show the priorities that are set for requirements engineering work. The answers to RQ3, finally, tell us what requirements engineering techniques matter most because they are associated with success more than other techniques do.

We built the questionnaire by first basing it on requirements engineering state-of-art (Cheng and Atlee 2007, Pohl and Rupp 2011) and then adjusting it based on suggestions from practitioners with broad overview on the software industry. A focus group with experienced practitioners evaluated adequacy, coverage, and understandability of the questionnaire. We then tested and further improved the form by letting respondents fill it in and give feedback in interviews.

To know who was answering, the questionnaire asked respondents to characterize their most recent software project. To answer RQ1, the questionnaire asked multiple-choice questions about requirements-related inquiry, specification, and management techniques in the characterized project. To answer RQ2, it asked multiple-choice questions about the goals of requirements engineering in the project and of the specified product. To answer RQ3, it asked questions about the requirements engineering success. Free-text areas allowed expanding or qualifying the answers.

The theoretical population of the survey were all software projects that were recent when the survey was administered in 2012. The sampling frame were the industrial contacts of our partners in academia and industry. To increase the reach of the survey we encouraged subjects to recommend the questionnaire to their own contacts.

625 respondents, about 10% of the invited persons, answered the online questionnaire. Filtering for completeness and plausibility reduced the data to 419 valid answers. This number of answers makes this requirements engineering survey by far the largest ever published. The obtained number of samples allowed describing requirements engineering practice with a margin of error smaller than ±5% for 95% confidence.

We answered the research questions with statistical analysis. Descriptive statistics of proportions were used for answering RQ1 and RQ2. The difference of proportions test, a variation of the independent samples *t* test, was used for answering RQ3. To control the accumulation of type I error, Holm's step-down method (Holm 1979) was used to prune the *t* test results for statistical significance. Wilcoxon's rank-sum test, finally, allowed us to explore an additional angle to answer RQ3, whether the number of requirements engineering practices that are used in a software projects would correlate with success.

4 Requirements Engineering Practice and Success

4.1 Responding Projects

A diverse mix of software projects answered the survey. Fig. 1 gives an overview of the answering projects, the kinds of software products they developed, and the companies they belonged to.

Project		
Category		
Research	32	8%
Product, Platform	87	21%
Bespoke	237	57%
Tender	42	10%
Other	21	5%
Process (Lifecycle Model)		
Prototyping	23	5%
Evolutionary	39	9%
Incremental	113	27%
Hybrid	104	25%
Sequential	135	32%
Other	5	1%
Size (number of staff)* **		
< 4	54	13%
4-9	107	26%
10-19	89	21%
20-49	84	20%
>= 50	82	20%
n/a	3	1%
Duration (calendar months)* **		
< 4.5	27	6%
4.5-9	97	23%
9-18	119	28%
>= 18	165	39%
n/a	11	3%

Product		
Industry (application domain)		
Banking, Finance	98	23%
Automotive, Transport	54	13%
Software, IT	51	12%
Government, Military	40	10%
Healthcare, Medical	35	8%
Insurance	31	7%
Telecommunications	31	7%
Manufacturing, Supply	22	5%
Other	57	14%
Type		
Information System	260	62%
Software-Intensive	62	15%
Engineering	28	7%
Other	69	16%
Innovation		
Completely New	149	36%
New Features or Use	107	26%
Changed Technology	79	19%
Maintenance	84	20%

Company		
Size (number of employees)* **		
< 10	23	5%
10-49	39	9%
50-249	42	10%
250-4499	144	34%
>= 4500	168	40%
n/a	3	1%
Location		
Europe	368	88%
Switzerland	248	59%
Germany	69	16%
Other Europe	51	12%
Americas	26	6%
Asia-Pacific	25	6%

***: n/a was a possible answer,
light color: sub-categories,
"Other": answers with less than 5% frequency.

Fig. 1. Demography of projects that responded to the survey.

Many projects were performed at large companies in Switzerland and developed information systems. This distribution is consistent with the Switzerland-oriented contact networks we used for soliciting responses. The key employer in this country is the service sector with IT departments that produce information systems.

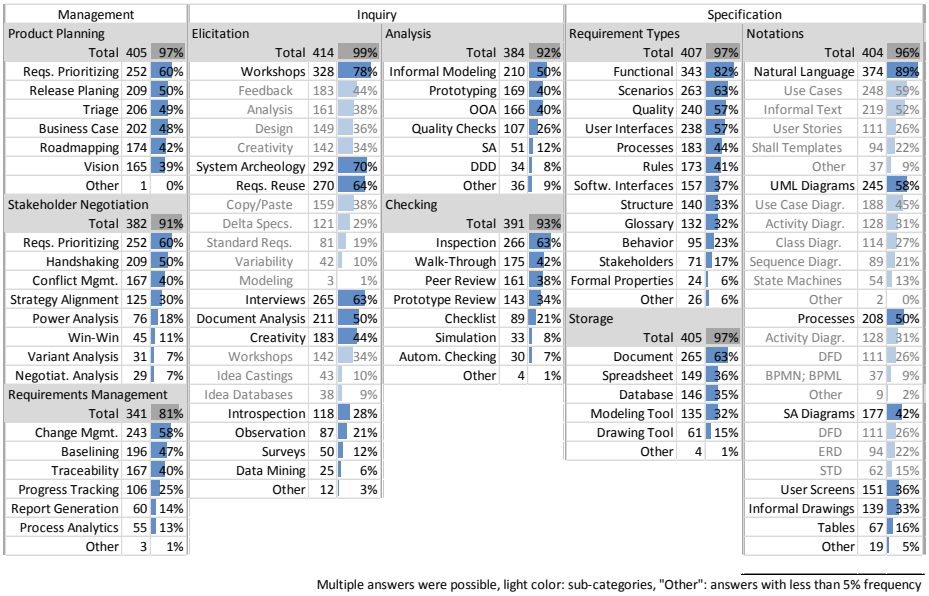
A wide spread of industries were addressed with the developed products. 35 responses, 8% of all responses, were given by projects that developed products for healthcare. Thus the results reflect practice across industries and are not specific to one of them, for example healthcare. Product novelty was relatively evenly spread.

A majority of the projects were bespoke and developed tailor-made solutions. The projects used a sequential, incremental, or hybrid development process. Only few did research or used a process like the Spiral model that is designed for experimentation. The long duration of the projects may be explained by the prolonged relationship that IT departments have with the business units they support. The same relationship can

also explain the bespoke nature of the projects. Information systems developed for business units are used by a pre-determined set of users that can be actively involved into the requirements engineering process.

4.2 Common Practice

Our data shows that requirements engineering was widely established. However, there was not one way of doing requirements engineering. While only few of the techniques are employed by almost all projects, e.g. workshops, many of the techniques are used by some of the projects only. This result indicates a wide variety of how requirements engineering is done. Fig. 2 gives an overview of how frequently each requirements engineering practice was used.



Management		
Product Planning		
Total	405	97%
Reqs. Prioritizing	252	60%
Release Planing	209	50%
Triage	206	49%
Business Case	202	48%
Roadmapping	174	42%
Vision	165	39%
Other	1	0%
Stakeholder Negotiation		
Total	382	91%
Reqs. Prioritizing	252	60%
Handshaking	209	50%
Conflict Mgmt.	167	40%
Strategy Alignment	125	30%
Power Analysis	76	18%
Win-Win	45	11%
Variant Analysis	31	7%
Negotiat. Analysis	29	7%
Requirements Management		
Total	341	81%
Change Mgmt.	243	58%
Baselining	196	47%
Traceability	167	40%
Progress Tracking	106	25%
Report Generation	60	14%
Process Analytics	55	13%
Other	3	1%

Inquiry		
Elicitation		
Total	414	99%
Workshops	328	78%
Feedback	183	44%
Analysis	161	38%
Design	149	36%
Creativity	142	34%
System Archeology	292	70%
Reqs. Reuse	270	64%
Copy/Paste	159	38%
Delta Specs.	121	29%
Standard Reqs.	81	19%
Variability	42	10%
Modeling	3	1%
Interviews	265	63%
Document Analysis	211	50%
Creativity	183	44%
Workshops	142	34%
Idea Castings	43	10%
Idea Databases	38	9%
Introspection	118	28%
Observation	87	21%
Surveys	50	12%
Data Mining	25	6%
Other	12	3%

Analysis		
Total	384	92%
Informal Modeling	210	50%
Prototyping	169	40%
OOA	166	40%
Quality Checks	107	26%
SA	51	12%
DDD	34	8%
Other	36	9%
Checking		
Total	391	93%
Inspection	266	63%
Walk-Through	175	42%
Peer Review	161	38%
Prototype Review	143	34%
Checklist	89	21%
Simulation	33	8%
Autom. Checking	30	7%
Other	4	1%

Specification		
Requirement Types		
Total	407	97%
Functional	343	82%
Scenarios	263	63%
Quality	240	57%
User Interfaces	238	57%
Processes	183	44%
Rules	173	41%
Softw. Interfaces	157	37%
Structure	140	33%
Glossary	132	32%
Behavior	95	23%
Stakeholders	71	17%
Formal Properties	24	6%
Other	26	6%
Storage		
Total	405	97%
Document	265	63%
Spreadsheet	149	36%
Database	146	35%
Modeling Tool	135	32%
Drawing Tool	61	15%
Other	4	1%

Notations		
Total	404	96%
Natural Language	374	89%
Use Cases	248	59%
Informal Text	219	52%
User Stories	111	26%
Shall Templates	94	22%
Other	37	9%
UML Diagrams	245	58%
Use Case Diagr.	188	45%
Activity Diagr.	128	31%
Class Diagr.	114	27%
Sequence Diagr.	89	21%
State Machines	54	13%
Other	2	0%
Processes	208	50%
Activity Diagr.	128	31%
DFD	111	26%
BPMN; BPML	37	9%
Other	9	2%
SA Diagrams	177	42%
DFD	111	26%
ERD	94	22%
STD	62	15%
User Screens	151	36%
Informal Drawings	139	33%
Tables	67	16%
Other	19	5%

Multiple answers were possible, light color: sub-categories, "Other": answers with less than 5% frequency

Fig. 2. Common requirements engineering practice.

Almost every project elicited requirements. The projects tended to do with stakeholder workshops, by studying existing systems, or by re-using specifications. Workshops dominated requirements elicitation practice. Only few projects used techniques like observation, ethnography, surveys, or data mining. These techniques are thus used in special situations only.

Almost every project planned the product to be developed, often by prioritizing requirements. Often a mix of planning techniques were used. No technique was dominant.

Almost every project analyzed requirements. Often a mix of informal modeling, prototyping, and object-oriented analysis was used. No analysis technique was dominant. Historically important techniques like structured analysis, quality function deployment, and decision trees or specialized techniques such as domain-driven development were uncommon.

Almost every project specified requirements. A majority specified functionality, quality, use scenarios, and user interfaces of intended solutions. Functional requirements dominated. Concepts commonly used for formal reasoning, such as agents, goals, and formal properties, were rare. For specifying the requirements, natural language dominated as the notation. Natural language was often complemented with UML diagrams. The use of other diagram types, user screens, and informal drawings varied. Formal-logic and goal-oriented languages like i* or KAOS were almost never used.

The frequency of notations that match requirements analysis techniques was inconsistent with requirements analysis practice. Object-oriented and structured diagrams were much more common than the use of corresponding analysis techniques. User screens were much less frequently documented than prototypes created. Formal specification languages were used as rarely as the corresponding formal methodology.

Almost every project stored requirements. Requirements documents were the most common type of storage. The use of spreadsheets, requirements databases, and modeling tools varied. Drawing tools, wikis, and cards for capturing requirements backlogs were uncommon.

Almost every project checked requirements. Projects tended to prefer manual requirements checking, preferably with rigorous inspections. Simulation and automated formal checking were uncommon.

Almost every project negotiated requirements. To reach an agreement on requirements, most common was requirements prioritization. Uncommon were analytical techniques such as power, variant, and negotiation analysis, and advanced techniques such as win-win negotiations.

Four out of five projects managed the requirements. This means at the same time that one out of five projects did not use the requirements once they were inquired. Requirements management tended to focus on the handling of requirements. Most common was change management. Requirements were rarely used for analyzing project progress, for reporting, or for measuring the development process.

Overall, the large variety of techniques indicates that there was no one-size-fits-all in requirements engineering practice. Still, there are a few practices that could be seen in a large majority of projects. These include the use workshops to discuss requirements with stakeholders, the specification of functional requirements, and the use of natural language for specification. Among the established methodologies, object-oriented analysis and specification appears to be widely adopted, although not dominating. The older counter-parts, for example structured analysis, were much less important in comparison. Extremely rare were formal techniques. Even-though they are widely researched, they have hardly found their way into current practice.

4.3 Requirements Engineering Success

The data from the responding projects showed that there was no dominant way of judging requirements engineering success. Fig. 3 gives an overview.

Requirements Engineering Goals		
Total	419	100%
Shared Understanding	214	51%
Specification Quality	197	47%
Clear Scope	160	38%
Efficiency	155	37%
User Satisfaction	145	35%
Timeliness	139	33%
Fit of Solution	94	22%
Estimation Reliability	65	16%
Architecture Quality	58	14%
Cost/Benefit Analysis	26	6%
Other	4	1%

Software Product Goals		
Total	419	100%
Productivity	228	54%
Effectiveness	156	37%
Compliance	143	34%
Satisfaction	137	33%
Flexibility	86	21%
Safety	73	17%
Environment	7	2%
Other	8	2%

Fig. 3. Success factors for requirements engineering process and outcome.

The most important requirements engineering goals were shared understanding between the project team and its stakeholders and good quality of the requirements specification. These two objectives of requirements engineering were often complemented with the need for a clear scope for the development, for using little time and resources for requirements engineering, for satisfying the users, and for delivering the requirements engineering work results in time.

The most common goal pursued by the software products that were specified with the requirements was productivity improvement. This goal was complemented by a variety of goals that included effectiveness, e.g. to enable users to do things they could not do before, compliance with laws and regulations, and satisfaction of users and stakeholders with the product. Important societal topics, like environmental or societal challenges, were rarely considered to be a goal of the software product.

Fig. 4 shows how many projects were successful and how many have not been successful when judged according to the criteria summarized in Fig. 3. A bit more than half of the projects judged that they fulfilled the requirements engineering goals. A bit less than half judged they did too little. Almost none stated they would have done too much. While positive and negative satisfaction with requirements engineering were rather balanced, product goal achievement was a sharp success. About 9 out of 10 of the specified products were judged to be a success. Only few were considered failures.

Achievement of RE Goals		
Total	419	100%
Too little	181	43%
Just enough	229	55%
Too much	9	2%

Achievement of Product Goals		
Total	419	100%
Rather Yes	385	92%
Rather No	34	8%

Fig. 4. Requirements engineering success.

These success rates appear to contradict the success rates identified in other studies. In comparison, Standish presented 32% project success rate in 2009 by taking into consideration scope, time, and budget adherence (Eveleens and Verhoef 2010). The staggering success rate of 92% for product goal achievement we observed thus says that

while the project may have been problematic, the outcome of the project was not. Also, 55% satisfaction with the requirements engineering experience is significantly larger than then 32% project success rate. This may indicate that requirements engineering practice had matured and was less problematic than other disciplines.

4.4 Success-Correlating Practice

To identify effective requirements engineering practice we correlated technique use with requirements engineering success. The result of this analysis indicates the techniques that are used in projects with successful requirements engineering significantly more often than in projects that did not meet requirements engineering goals or produced products that did not achieve their goals. Whether practice use leads to success or whether good projects select these practices cannot be concluded from these results and needs to be investigated in future research.

Our survey data showed 221 projects with successful requirements engineering and 189 failures according to our success criteria. Only three techniques correlated with requirements engineering success with $p < 0.05$ significance after pruning the results with Holm's step-down method to remove false positives. None of the other techniques correlated significantly with success, and no technique correlated negatively. Fig. 5 gives an overview.

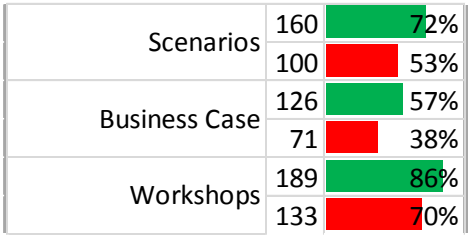


Fig. 5. Techniques that correlated with requirements engineering success (upper rows: successes, lower rows: failures)

Scenarios are exemplary sequences of system usage (Alexander and Maiden 2005). In requirements engineering, they are used to describe concrete stories of how users and external systems interact with the system under consideration to achieve goals that are of value to the user. Scenarios make the functionality of the system concrete and thus enable users to judge whether they feel to be able to use the system meaningfully and whether they like it. Scenarios also allow capturing interaction design knowledge from user experience experts. A common format used to document scenarios is the use case template (Cockburn 2001).

Business cases are used to document predicted financial results and other business consequences for one or multiple alternative ways of how a product is built, deployed, and maintained (Schmidt 2002). The business case planning work and results that are obtained with it are determinant for selecting what is in the product scope and what not and for evaluating whether a chosen scope is attractive for the customer of the project.

The understanding of a business case allows to stop work on a product that does not make sense or to re-scope the product to make it more attractive.

Workshops create an efficient, controlled, and dynamic setting for quickly eliciting, prioritizing, and agreeing on requirements (Gottesdiener 2002). The discussion of requirements by the critical stakeholders makes a requirements workshop to be one of the most efficient techniques to perform inquiry and to achieve shared understanding. No other technique allow exposure and resolution of conflicts between stakeholders so efficiently. The same applies for discovery and resolution of misunderstandings.

We also studied whether the number of techniques used and the number of requirement types documented in a project correlates with requirements engineering success. Fig. 6 shows the distributions with box plots.

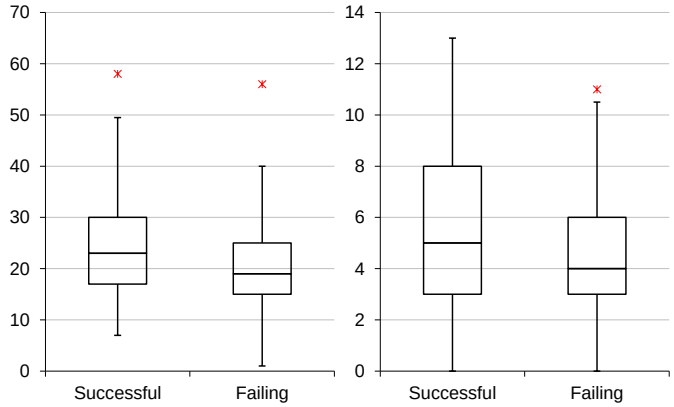


Fig. 6. Number of requirements engineering techniques (left) and number of requirements types (right) used in projects with successful, respectively failing requirements engineering.

According to the Wilcoxon's rank-sum test, successful projects use a significantly larger number of requirements engineering techniques and specify a significantly larger number of requirement types than unsuccessful ones.

Again, the causes for these correlations should be investigated with future research. A hypothesis that should be tested is whether there is a large variety of project context that require more techniques to be used. The observed large variety of requirements engineering techniques used across projects would support this claim. Another hypothesis may be that the experience of a requirements engineer may lead to both greater use of techniques and greater likelihood of project success. Experienced requirements engineers will use more techniques and specify more types of requirements than less experienced requirements engineers. At the same moment they are the better guarantors of requirements engineering success.

5 Discussion

5.1 Contribution

This chapter has provided an overview of common and best practice in requirements engineering. Based on answers from 419 projects, it has shown how frequently the many requirements engineering were used and which of these techniques correlated with requirements engineering success. The presented frequencies of requirements engineering technique use extend the results from earlier surveys (Neill and Laplante 2003, Paech, Koenig et al. 2005) with an updated set of practices. For example, prior surveys did not evaluate whether common requirements engineering research such as i* or KAOS had been transferred to practice. Our results showed that these two techniques were not. The presented results also extend prior research with a number of samples that is by far larger than any previous requirements engineering survey was based on. We could indicate state-of-practice with a level of accuracy that previous surveys could not.

Earlier surveys did not have enough data to correlate requirements engineering practice with success, thus gave others fertile grounds to claim that no technique would lead to requirements engineering success (Davis and Zowghi 2006). The results shown in this chapter provide counter-evidence. While we could not demonstrate sufficiency or necessity of any requirements engineering practice, we could show that there are indeed a few requirements engineering techniques that are associated with significantly higher success rates. The relevant techniques were scenarios, business cases, and workshops. Interestingly, the three techniques we identified were not the same that were used by companies that valued requirements engineering (Rouibah and Al-Rafee 2009). The latter included prototyping, data flow diagrams and techniques like quality function deployment and decision trees that are hardly used according to our data. To understand why the identified techniques correlate with success, while others do not, future research should investigate what causes the correlations.

The presented results are relevant for education and practice. They enable comparison of own practice and competences with common practice. Requirements engineers should be well-versed in the use of the techniques that are used frequently. According to our data, almost any requirements engineer will be requested to perform workshops and specify functional requirements in natural language. The larger number of techniques used and requirements types specified in successful requirements engineering also shows that a requirements engineer should know many practices, rather than few. Comparison of the frequencies of the various requirements engineering techniques may be used to guide the development of needed competencies. For example, object-oriented analysis techniques should be learned before structured analysis techniques because the former are much more common in real-world projects than the latter.

The presented results can be used for advice about effective practice. If the three success-correlating techniques cause requirements engineering success, they should be used whenever possible. Accepting this assumption, we started utilizing requirements workshops systematically. We integrated scenarios into the workshops by role-playing and discussing the intended system use with the participating real-world stakeholders.

We integrated business case by discussing whether and why each of the product's features are needed and by estimating feasibility and cost. Fig. 7 show a photograph from one of these workshop, which involved real stakeholders, users, project members, and experts.

Fig. 7. Requirements workshop for exploring scenarios and business case (participants shown in photograph from left to right: development manager, project leader, requirements engineer, domain expert, domain expert, quality of experience expert, lead engineer, user, developer)

5.2 Threats to Validity

Any research results should be considered with caution because none is free from threats to validity. We discuss here the threats to conclusion, internal, construct, and external validity that were suggested for empirical software engineering research by (Wohlin, Runeson et al. 2000).

Conclusion validity is concerned with the relationship between treatment and outcome. The here presented results are based on a survey with a large-enough number of samples that were analyzed by keeping the accumulation of the probability of false positives in mind. As a result we can claim that there is a statistically significant relationship between the identified three practices scenarios, business case, and workshops and the success of requirements engineering.

Internal validity is concerned with the causal relationship between treatment and outcome. The survey does not tell us anything about such causal relationships. We have no evidence that it is the scenarios, the business case, or the workshops that caused success, but just know that their frequency of use is different significantly between successful and failing projects. Further research is needed to understand what the right causal relationship is.

Construct validity is concerned with the relation between theory and observation. Potential problems may be misunderstanding of questions and answers and inadequate operationalization of the concepts we evaluated. We used survey pre-tests with experts and with selected respondents, thus have reduced the likelihood of misunderstanding the questionnaire. Also, we filtered responses that were incomplete or unreasonable based on the free-text answers that were given. The potential problem of inadequate

operationalization was addressed by basing our definition on prior research of requirements engineering success and by adding to the original success measurement the impact of requirements engineering: whether the specified system achieved its goals. The study thus reasonably reflects the constructs were intended to evaluate.

An important threat to validity of this survey concerns external validity. Alike other surveys in requirements and software engineering, we used nonprobability sampling to reach respondents. Hence, the frequencies we presented are valid for populations of projects that have a profile similar to the one presented in Fig. 1. As requirements engineering practice varies a lot across projects, other populations may have different frequencies and other conclusions about practice use may need to be drawn. Also availability and knowledge of techniques evolves over time and affect the frequency distribution of practice use. For example, 30 years ago Structured Analysis would have been much more frequent than Object-Oriented Analysis. We conjecture, however, that change in the frequency of technique use does not affect the effectiveness of the techniques. The causes that make business cases, scenarios, and workshops effective are hardly affected by such changes. Thus we expect the results about practice effectiveness to be replicable also in populations other than the studied one.

5.3 Need for Research

The large variations of the requirements engineering techniques used by the software projects indicates that technique use may depend on context. Such context dependency was shown in other related domains as well (Khurum, Fricker et al. 2014). For software built for digital health, the context dependency may imply shifts in the frequency of practice use. Also, the regulated nature of the healthcare industry may imply that additional techniques, such as document analysis and reuse of requirements may correlate with requirements engineering success. The former may help in the identification of compliance requirements (Thuemmler, Fricker et al. 2013), the latter in how to specify a system that is compliant.

Another venue of research is the construction of models that explain how requirements engineering success can be achieved. For example, control theory of goal-oriented systems was used to explain how stakeholder needs and development intentions can be aligned in a one-to-one situation between a product manager and an architect (Fricker, Gorschek et al. 2008) and successfully validated (Fricker and Glinz 2010). However, other approaches may be needed to achieve socio-technical alignment of a software solutions and technologies in large-scale. Once validated in large-scale, these models help us to develop and justify causes and effects behind the correlations that are observed in surveys such as the one presented here. They are also the basis to design techniques that help practitioners to be successful.

6 Summary and Conclusions

This chapter has presented the results of a large-scale survey of requirements engineering practice. The results were obtained by analyzing the answers of 419 projects. Many of the answering projects developed of information systems in a bespoke manner

with known stakeholders. The projects implemented agile, waterfall, and hybrid development processes. A wide variety of industries were addressed by the developed software products. 221 projects did requirements engineering successfully, 189 not.

Almost all projects elicited, planned, analyzed, specified, checked, and managed requirements. The most common techniques were stakeholder workshops and the specification of functional requirements with natural language. For most of the remaining techniques can be concluded that requirements engineering practice varies across the software projects. Formal techniques were extremely rare, even if they were researched intensively.

Only three techniques correlated with requirements engineering success: scenarios of system use, business cases, and stakeholder workshops. We recommend that all projects implement these practices. In addition we observed that projects with successful requirements engineering used more requirements engineering techniques and specified more types of requirements than unsuccessful ones. We thus recommend the use of experienced requirements engineers that are able to identify and apply the right technique for the many situations that may be encountered in a real-world project.

The obtained results extend previous surveys that were substantially smaller with updated frequencies of requirements engineering technique use and with the really new insights into what practices are correlate with success. Further research should look at specific project constellations, for example at projects that target digital health, and at building theories for what can be done to achieve requirements engineering success.

References

Achimugu, P., A. Selamat, R. Ibrahim and M. N. r. Mahrin (2014). "A Systematic Literature Review of Software Requirements Prioritization Research." Information and Software Technology **56**(6): 568-585.

Alexander, I. and N. Maiden (2005). Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle, John Wiley & Sons.

Alexander, I. and S. Robertson (2004). "Understanding Project Sociology by Modeling Stakeholders." IEEE Software **21**(1): 23-27.

Alvarez, R. and J. Urla (2002). "Tell me a Good Story: Using Narrative Analysis to Exmine Informaton Requirements Interviews during an ERP Implementation." Data Base for Advances in Information Systems **33**(1): 38-52.

Arlow, J. and I. Neustadt (2005). UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design, Pearson Education.

Bajec, M. and M. Krisper (2005). "A Methodology and Tool Support for Managing Business Rules in Organisations." Information Systems **30**(6): 423-443.

Berard, B., M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci and P. Schnoebelen (2010). Systems and Software Verification: Model-Checking Techniques and Tools, Springer Berlin Heidelberg.

Beyer, H. and K. Holtzblatt (1995). "Apprenticing with the Customer." Communications of the ACM **38**(5): 45-52.

Boehm, B., P. Grünbacher and R. Briggs (2001). "Developing Groupware for Requirements Negotiation: Lessons Learned." IEEE Software **18**(3): 46-55.

Cheng, B. and J. Atlee (2007). Research Directions in Requirements Engineering. Future of Software Engineering (FOSE'07). Washington, DC, USA.

Chung, J.-Y., K.-J. Lin and R. Mathieu (2003). "Web Services Computing: Advancing Software Interoperability." IEEE Computer **36**(10): 35-37.

Chung, L., B. Nixon, E. Yu and J. Mylopoulos (2000). Non-Functional Requirements in Software Engineering. Boston, USA, Kluwer Academic Publishers.

Cleland-Huang, J. and B. Mobasher (2008). Using Data Mining and Recommender Systems to Scale up the Requirements Process. 16th IEEE International Requirements Engineering Conference (RE'08). Barcelona, Spain.

Cleland-Huang, J., R. Settimi, E. Romanova, B. Berenbach and S. Clark (2007). "Best Practices for Automated Traceability." Computer **40**(6): 27-35.

Cockburn, A. (2001). Writing Effective Use Cases, Addison-Wesley Professional.

Conradi, R. and B. Westfechtel (1998). "Version Models for Software Configuration Management." ACM Computing Surveys **30**(2): 232-282.

Danevas, P. and G. Garva (2012). Domain Driven Development and Feature Driven Development for Development of Decision Support Systems. 18th International Conference on Information and Software Technologies (ICIST 2012). Kaunas, Lithuania.

Davis, A. (2005). Just Enough Requirements Management, Dorset House Publishing.

Davis, A., O. Dieste, A. Hickey, N. Juristo and A. Moreno (2006). Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review. 14th IEEE International Requirements Engineering Conference (RE'06). Minneapolis, MN, USA.

Davis, A. and D. Zowghi (2006). "Good Requirements Practices are Neither Necessary nor Sufficient." Requirements Engineering **11**(1): 1-3.

DeMarco, T. (1979). Structured Analysis and System Specification, Yourdon Press.

Denger, C., D. Berry and E. Kamsties (2003). Higher Quality Requirements Specifications through Natural Language Patterns. IEEE International Conference on Software: Science, Technology and Engineering (SwSTE'03). Herzelia, Israel.

Dieste, O., N. Juristo and F. Shull (2008). "Understanding the Customer: What do we Know about Requirements Elicitation?" IEEE Software **25**(2): 11-13.

Dwarakanath, A., R. Ramnani and S. Sengupta (2013). Automatic Extraction of Glossary Terms from Natural Language Requirements. 21st IEEE International Requirements Engineering Conference (RE'13). Rio de Janeiro, Brazil.

Easterbrook, S., R. Lutz, R. Covington, J. Kelly, Y. Ampo and D. Hamilton (1998). "Experiences Using Lightweight Formal Methods for Requirements Modeling." IEEE Transactions on Software Engineering **24**(1): 1-11.

El Emam, K. and N. Madhavji (1985). A Field Study of Requirements Engineering Practices in Information Systems Development. 2nd IEEE International Symposium on Requirements Engineering (RE'95). York, England.

Eveleens, L. and C. Verhoef (2010). "The Rise and Fall of the Chaos Report Figures." IEEE Software **27**(1): 30-36.

Fricker, S. (2009). Pragmatic Requirements Communication: The Handshaking Approach, Shaker.

Fricker, S. and M. Glinz (2010). Comparison of Requirements Hand-Off, Analysis, and Negotiation: Case Study. 18th IEEE International Requirements Engineering Conference (RE'10). Sydney, Australia.

Fricker, S., T. Gorschek, C. Byman and A. Schmidle (2010). "Handshaking with Implementation Proposals: Negotiating Requirements Understanding." IEEE Software **27**(2): 72-80.

Fricker, S., T. Gorschek and M. Glinz (2008). Goal-Oriented Requirements Communication in New Product Development. International Workshop on Software Product Management (IWSPM 2008). Barcelona, Spain.

Fricker, S. and S. Schumacher (2012). Release Planning with Feature Trees: Industrial Case. Requirements Engineering: Foundations for Software Quality (RefsQ 2012). Essen, Germany.

Glinz, M. (2010). Very Lightweight Requirements Modeling. 18th IEEE International Requirements Engineering Conference (RE'10). Sydney, Australia.

Glinz, M. and S. Fricker (2013). On Shared Understanding in Software Engineering. Software Engineering 2012. Aachen, Germany.

Glinz, M. and S. Fricker (2014). "On Shared Understanding in Software Engineering: An Essay." Computer Science - Research and Development.

Glinz, M., C. Seybold and S. Meier (2007). Simulation-driven Creation, Validation and Evolution of Behavioral Requirements Models. Dagstuhl-Workshop MBEES: Modellbasierte Entwiclung eingebetteter Systeme III (MBEES 2007). Braunschweig, Germany.

Gorschek, T., S. Fricker and K. Palm (2010). "A Lightweight Innovation Process for Software-Intensive Product Development." IEEE Software **27**(1).

Gorschek, T. and C. Wohlin (2006). "Requirements Abstraction Model." Requirements Engineering **11**(1): 79-101.

Gottesdiener, E. (2002). Requirements by Collaboration: Workshops for Defining Needs, Addison-Wesley Professional.

Hassenzahl, M., A. Beu and M. Burmester (2001). "Engineering Joy." IEEE Software **18**(1): 70-76.

Hickey, A. and A. Davis (2003). Elicitation Technique Selection: How Do Experts Do It? 11th IEEE International Requirements Engineering Conference. Monterey Beach, CA, USA.

Hickey, A. and A. Davis (2004). "A Unified Model of Requirements Elicitation." Journal of Management Information Systems **20**(4): 65-84.

Holm, S. (1979). "A Simple Sequential Rejective Multiple Test Procedure." Scandinavian Journal of Statistics **6**(2): 65-70.

Holtmann, J., J. Meyer and M. von Detten (2011). Automatic Validation and Correction of Formalized, Textual Requirements. IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2011). Berlin, Germany.

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology 610.12-1990.

ISO/IEC (2010). Systems and Software Quality Requirements and Evaluation, ISO/IEC. **ISO/IEC FDIS 25010**.

Khurum, M., S. Fricker and T. Gorschek (2014). "The Contextual Nature of Innovation - An Empirical Investigation of Three Software Intensive Products." Information and Software Technology.

Kniberg, H. and M. Skarin (2010). Kanban and Scrum - Making the Most of Both, lulu.com.

Kobayashi, A. and M. Maekawa (2001). Need-based Requirements Change Management. 8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2001). Washington, DC, USA.

Lam, W., J. McDermid and A. Vickers (1997). "Ten Steps Towards Systematic Requirements Reuse." Requirements Engineering **2**(2): 102-113.

Maiden, N., A. Gizikis and S. Robertson (2004). "Provoking Creativity: Imagine What Your Requirements Could Be Like." IEEE Software **21**(5): 68-75.

Martin, R. C. and G. Meinik (2008). "Tests and Requirements, Requirements and Tests: A Möbius Strip." IEEE Software **25**(1): 54-59.

McGrath, M. E. (2001). Product Strategy for High Technology Companies: Accelerating Your Business to Web Speed, McGraw-Hill.

Melão, N. and M. Pidd (2000). "A Conceptual Framework for Understanding Business Processes and Business Process Modelling." Information Systems Journal **10**(2): 105-129.

Milne, A. and N. Maiden (2012). "Power and Politics in Requirements Engineering: Embracing the Dark Side?" Requirements Engineering **17**(2): 83-98.

Myers, B. (1989). "User-Interface Tools: Introduction and Survey." IEEE Software **6**(1): 15-23.

Mylopoulos, J., L. Chung and E. Yu (1999). "From Object-Oriented to Goal-Oriented Requirements Analysis." Communications of the ACM **42**(1): 31-37.

Neill, C. and P. Laplante (2003). "Requirements Engineering: The State of the Practice." IEEE Software **20**(6): 40-45.

Ng, L., W. Barfield and F. Mannering (1995). "A Survey-based Methodology to Determine Information Requirements for Advanced Traveler Information Systems." Transportation Research Part C: Emerging Technologies **3**(2): 113-127.

Paech, B., T. Koenig, L. Borner and A. Aurum (2005). An Analysis of Empirical Requirements Engineering Survey Data. Engineering and Managing Software Requirements. A. Aurum and C. Wohlin, Springer**:** 427-452.

Pérez-Castillo, R., I. Garciía-Rodríguez de Guzmán and M. Piattini (2011). "Business Process Archeology using MARBLE." Information and Software Technology **53**(10): 1023-1044.

Petersen, K. and C. Wohlin (2010). "Measuring the flow in lean software development." Software Practice and Experience **41**(9): 975-996.

Phaal, R., C. Farrukh and D. Probert (2003). "Technology Roadmapping - A Planning Framework for Evolution and Revolution." Technological Forecasting and Social Change **71**: 5-26.

Pohl, K. and C. Rupp (2011). Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level - IREB Compliant, Rocky Nook Computing.

Porter, A., L. Votta and V. Basili (1995). "Comparing Detection Methods for Software Requirements Inspections - Replicated Experiment." IEEE Transactions on Software Engineering **21**(6): 563-575.

Potts, C., K. Takahashi and A. I. Antón (1994). "Inquiry-based requirements analysis." IEEE software **11**(2): 21-32.

Raiffa, H. (2007). Negotiation Analysis: The Science and Art of Collaborative Decision Making, Harvard University Press.

Rea, L. and R. Parker (2005). Designing and Conducting Survey Research: A Comprehensive Guide. San Francisco, CA, USA, Jossey-Bass.

Rettig, M. (1994). "Prototyping for Tiny Fingers." Communications of the ACM **37**(4): 21-27.

Ross, D. (1977). "Structured Analysis (SA): A Language for Communicating Ideas." IEEE Transactions on Software Engineering **3**(1): 16-34.

Rouibah, K. and S. Al-Rafee (2009). "Requirements Engineering Elicitation Methods: A Kuwaiti Empirical Study about Familiarity, Usage and Perceived Value." Information Management & Computer Security **17**(3): 192-217.

Schmidt, M. (2002). The Business Case Guide, Solution Matrix.

Schobbens, P.-Y., P. Heymans, J.-C. Trigaux and Y. Bontemps (2007). "Generic Semantics of Feature Diagrams." Computer Networks **51**(2): 456-479.

Standish Group International (1995). The CHAOS Report, Standish Group International, Inc.

Svahnberg, M., T. Gorschek, R. Feldt, R. Torkar, S. Bin Saleem and M. U. Shafique (2010). "A Systematic Review on Strategic Release Planning Models." Information and Software Technology **52**(3): 237-248.

Thuemmler, C., S. Fricker, O. Mival, D. Benyon, W. Buchanan, A. Paulin, M. Fiedler, B.-J. Koops, E. Kosta, A. Grottland, A. Schneider, T. Jell, A. Gavras, M. Barros, T. Magedanz, P. Cousin, I. Ispas and E. Petrakis (2013). Norms and Standards in Modular Medical Architectures. IEEE 15th International Conference on e-Health Networking, Applications and Services (Healthcom 2013). Lisbon, Portugal.

van de Weerd, I., S. Brinkkemper, R. Nieuwenhuis, J. Versendaal and L. Bijlsma (2006). Towards a Reference Framework for Software Product Management. 14th IEEE International Requirements Engineering Conference (RE'06). Minneapolis; MN, USA.

van Lamsweerde, A. (2001). Goal-Oriented Requirements Engineering: A Guided Tour. 5th IEEE International Symposium on Requriements Engineering (RE'01). Toronto, Canada.

Vermersch, P. (2009). "Describing the Practice of Introspection." Journal of Consciousness Studies **16**(10-12): 20-57.

Whittle, J. and J. Schumann (2000). Generating Statechart Designs from Scenarios. IEEE International Conference on Software Engineering (ICSE 2000). Limerick, Ireland.

Wohlin, C., P. Runeson, M. Host, C. Ohlsson, B. Regnell and A. Wesslén (2000). "Experimentation in software engineering: an introduction."

Zowghi, D. and C. Coulin (2005). Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. Engineering and Managing Software Requirements. A. Aurum and C. Wohlin. Berlin, Germany, Springer.