Chapter 1
An Introduction to Requirements Knowledge

W. Maalej and A.K. Thurimella

Abstract Requirements represent a verbalisation of decision alternatives on the functionality and quality of a system. Engineering, planning, and implementing requirements are collaborative, problem-solving activities, where stakeholders consume *and* produce considerable amounts of knowledge. Managing requirements knowledge is about efficiently identifying, accessing, externalising, and sharing this knowledge by and to all stakeholders, including analysts, developers, and users. This chapter introduces five foundations of managing requirements knowledge, which are discussed in the book parts. First, *identifying* requirements knowledge aims at externalising tacit knowledge such as rationale or presuppositions. Second, *representing* requirements knowledge targets an efficient information access and artefact reuse within and between projects. Third, *sharing* requirements knowledge improves stakeholders' collaboration and ensures that their experiences do not get lost. Fourth, *reasoning* about requirements and their interdependencies aims at detect inconsistencies and deriving new knowledge. Finally, *intelligent tool support* reduces the overhead to manage requirements knowledge.

1.1 What Is Requirements Engineering?

We use the term requirements in the context of systems engineering**,** which is the discipline concerned by designing, developing, deploying, and maintaining systems. A system is an organised set of communicating parts designed for a specific purpose

W. Maalej (✉)
University of Hamburg, Department of Informatics/MOBIS, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
e-mail: maalej@informatik.uni-hamburg.de

A.K. Thurimella
Harman Becker Automotive Systems GmbH, Moosacher Str. 48, 80809, Munich, Germany
e-mail: anil.thurimella@gmail.com

W. Maalej and A.K. Thurimella (eds.), *Managing Requirements Knowledge*,
DOI 10.1007/978-3-642-34419-0_1,

[1]. For example, a mobile phone composed of a display, a battery, an antenna, a microphone, a speaker, and a processor is designed to enable users to make calls, while they are on the go. In this book we target software engineering, which is a subdiscipline that focuses on engineering software-intensive systems. Bruegge and Dutoit characterise software engineering as a modelling and problem-solving activity, which is knowledge intensive and rationale driven [1].

Participants in a software engineering project create formal and informal models to (a) reason about software systems, (b) communicate about them, and (c) document their properties. These might be functional models, object models, dynamic models, or feature models. A functional model, for example, a use case diagram, describes the *functionality* of a system, for instance, dialling a number or terminating a call. An object model, for example, a class diagram, describes the *structure* of a system in terms of components, objects, attributes, and operations. A dynamic model, for example, a sequence diagram, represents the interactive behaviour of the system or of its parts, for instance, how the user interacts with the keyboard to dial a number or how the antenna interacts with the processor to send signals. The high-level description of a system is often communicated to clients and end users in terms of features, which are prominent or distinctive visible characteristics or qualities of a system [2]. For example, an mp3 player and a multitouch interface are two features of a modern mobile phone.

The term *requirement* is similar to the term feature but has a larger scope and a more technical focus. The IEEE standard glossary of software engineering terminology defines a requirement [3] as "a statement of what the system must do, how it must behave, the properties it must exhibit, the qualities it must possess, and the constraints that the system and its development must satisfy [4]".

Requirements engineering (RE) is the branch of systems engineering concerned with the *desired properties and constraints* of software-intensive systems, the goals to be achieved in the software's environment, and assumptions about the environment [5]. In another frequently cited definition, Sommerville and Sawyer state that requirements engineering is the activity that emphasises the utilisation of *systematic* and *repeatable* techniques that ensure the completeness, consistency, and relevance of requirements [6]. We call this the engineering view of requirements. Nuseibeh and Easterbrook [7] define requirements engineering as the process of *discovering* the purpose of the system being developed, by identifying stakeholders and their needs and *documenting* these in a form that is *amenable* to analysis, communication, and subsequent implementation. We call this the life cycle view of requirements. Finally, Aurum and Wohlin consider requirements as *verbalisation of decision alternatives* regarding the functionality and quality of a system [8]. Requirements engineering can then be considered as the complex task of dealing with, making, and documenting these decisions. We call this the decision or the knowledge view of requirements.

Requirements engineering is considered as one of the most critical phases in software projects [9]. Poorly implemented requirements engineering is a major risk for projects failure [10]. Today's software projects still have a high probability to be cancelled or to significantly exceed available resources [11]. For example, Leffingwell [12] found that 40 % of the total project costs are associated with rework triggered by low-quality requirements.

Requirements engineering rarely receives more than 2–4 % of the overall project effort [13], even if more effort in getting the requirements right results in significantly higher project success rates. A recent Gartner report [14] states that requirements defects are the third most significant source of product defects after coding and design but are the first source of delivered defects (i.e. defects delivered to the user). Fixing a defect in production is approximately 200 times more expensive for a software project than fixing it during requirements engineering, Gartner says. The damage and costs caused to the customers and their users when delivering a defect are excluded from this calculation and cannot be truly quantified since it depends on the domain and the business. Improving the quality of requirements and the efficiency requirements engineering can reduce the overall cost of software, improve its quality, and dramatically shorten the time to market.

1.1.1 Requirements Engineering Activities

Requirements engineering covers several activities, including requirements elicitation, analysis, specification, verification, and management [15]:

- Requirements elicitation is the process of discovering, reviewing, documenting, and understanding the user's needs and constraints for a system.
- The process of refining the user's needs and constraints is called requirements analysis.
- Requirements specification is the process of documenting the user's needs and constraints clearly and precisely.
- Ensuring that the system requirements are complete, correct, consistent, and clear is done as a part of requirements verification.
- Scheduling, negotiating, coordinating, and documenting the requirements engineering activities are called requirements management.

Requirements engineering overlaps with planning. A project plan, including work packages, releases, iterations, or milestones, is created by analysing requirements. Later, the plan is detailed further. Tasks and action items are created and assigned to the project participants. The delivery of requirements is committed to the customer based on the project plan.

Requirements evolve over time. Change requests are often used to refer to changes on requirements. Change requests might originate from customers after the initial requirements elicitation phase, as well as from other sources such as regulators, development, testing, or marketing. Change requests are decided by analysing corresponding change impacts on the system. A well-implemented requirements elicitation often reduces the number of change requests in a project. Similarly, a large number of change requests are an indicator for poor requirements engineering.

Requirements engineering can also be performed for a product family (or a family of related systems) for systematically reusing artefacts and assets that are shared across multiple systems such as requirements, decisions, activities, and

processes. Such a product family (e.g. a particular generation of mobile phones such as data phones or smart phones) is called a software product line.

Clements and Northrop define a *software product line* (SPL) as a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [4]. Product line engineering uses variability as an abstraction to deal with customisation and reuse [2, 16].

SPLs offer several benefits such as improved reuse, quicker time to market, and decreased defect rates, in particular, for manufacturing and mass customisation companies. These benefits have been reported in the form of experiences and best practices, for example, in the product line hall of fame [17]. On the other hand, SPLs need large upfront investments, in particular, when systems have been in place over decades or in IT service industry where the customers' needs, infrastructures, and constraints drive the projects.

1.1.2 Requirements Artefacts

Requirements engineering involves various artefacts and document types. Business and marketing stakeholders, for instance, typically conduct negotiations with customers in the early project phases based on customer requirements and feature catalogues. Detailed requirements can be captured in natural language text, mathematical models, visual models such as UML, or using multimedia. A mathematical model represents the requirements formally, for example, using the Z specification language [18] and is used if, for example, the correctness of the system behaviour is critical. For instance, errors while making an emergency call can cost human lives. The behaviour of the system should be formally specified and validated in such cases – in particular if regulators mandate this. UML use case diagrams, sequence diagrams, and activity diagrams can be used to model requirements semiformally. For instance, these diagrams might show, respectively, use cases provided by a phone; interactions between the phone, its user, and its environment (e.g. an operator network or a Bluetooth device); and the overall process flow from dialling to getting the invoice.

More recently, the requirements engineering community started exploring and using multimedia requirements [19], including images (e.g. pictures of a typical hand positions during a call), drawings (e.g. mock-ups of the screen), or videos (e.g. showing a typical scenario of a mobile user in the metro). In industry, requirements are often documented in unstructured or semi-structured natural language documents [20], called requirements specification documents.

Requirements are classified into functional and non-functional requirements. A *functional requirement* expresses functionality (or a functional property) of a system and is specified based on inputs, outputs, and a process or a functional behaviour. An example for a mobile phone functional requirement is "the user shall be able to send and receive SMS to other users of mobile phones". A functional requirement can be decomposed into several sub-requirements.

A *non-functional requirement* (NFR) also called a quality requirement should express measurable properties of the system [79]. For example, "when switched on, the user should be able to dial a number within 3 s", which constrains the performance of the system. Other categories of NFRs include usability, availability, safety, security, privacy, and maintainability.

Requirements are documented and handled based on their types. Functional requirements are often described in use case documents or mock-ups, while NFRs are captured using text documents, in change requests, or as comments in source code. Different NFRs are handled differently. For instance, safety is often handled by specifying additional requirements to address hazards and misuse cases identified by safety engineers.

1.1.3 Stakeholders, Collaboration, and Decisions

Requirements engineering involves people with different backgrounds including business, marketing, law, project management, design, development, and testing. These people are called *stakeholders*. They perform relevant tasks in a requirements engineering process depending on their backgrounds and collaborate to capture requirements and make decisions about them and their priorities.

Making decisions is about choosing between alternative solutions for an issue [21]. An issue can be, for example, which authentication feature should the phone provide? The alternatives are a username and password, a pin code, a lock screen pattern, or face recognition. Effective decision-making occurs when stakeholders select the best choice based on the knowledge available at the time [22].

Rationale is the *reasoning* behind decisions, that is, the answer to the why question. Rationale can be described in natural language text or can be structured based on alternatives, reasons, and justifications. Kunz and Rittel introduced a rhetorical model to mange rationale called IBIS (issue-based information systems) [23]. IBIS uses abstractions like issue, option, argument, and resolution. An option is a potential solution for an issue. An assessment is a stakeholder argument that supports or hinders an option. A resolution contains a set of options that solve an issue. IBIS allows the expression of interdependencies between issues, which can lead to complex issue networks. QOC [24] extends IBIS with criteria, for example, the cost of implementing the authentication option, or its usability. Dutoit [25] introduced QOC to requirements engineering by modelling QOC criteria as goals or non-functional requirements, supporting decision-making in product management meetings.

Research has shown that rationale knowledge is useful in many ways. For example, it may be helpful to assess changes [26]. Alternative solutions and arguments documented in rationale may be used to forecast potential changes. Furthermore, rationale may be reused when similar issues are raised or when changes in previous decisions occur. However, rationale is barely managed systematically and often remains in the mind of people. When people leave the organisation, this knowledge gets lost [27]. In practice, rationale is found sporadically across documents, emails, or discussion threads [27].

As software projects are getting more distributed and the development cycles are getting shorter, stakeholders are often located in different places [28], sometimes even without having the resources to get to know each other in a face to face meeting. For example, the clients and the users might be in an Asian country with special regulations and infrastructures for communication systems and where people present certain habits and preferences in how they use their phones. The requirements engineers and the developers might be at the development site in a different country, speaking a different language, knowing different laws, and having different habits and preferences. Distributed development settings introduce additional collaboration challenges for stakeholders to understand each other, reach a common understanding of requirements, and make effective decisions.

1.2 What Is Managing Requirements Knowledge?

We introduce the terms "knowledge", "knowledge management", and "requirements knowledge" and motivate the need for managing requirements knowledge.

1.2.1 What Is Knowledge?

Knowledge is a popular term used in our everyday language as well as in several disciplines such as philosophy, management science, and computer science. According to the Oxford Dictionary, the term knowledge refers to facts, information, and skills acquired by a person through *experience or education*. It also refers to *the awareness or familiarity* gained by experience of a fact or situation.

In philosophy the study of knowledge is called epistemology. Plato gave one of the oldest and most famous definitions for knowledge as *justified true belief* [29]. However, there exists a large and still active debate between philosophers about Plato's definition and about the term knowledge and associated concepts [30].

In computer and management science, the term knowledge is often mixed with data and information. According to Theirauf [31]: "*data* represents the unstructured facts and figures, which has the least impact for the typical manager. [...] At the next level *information* is structured data that is useful to the manager in analysing and resolving critical problems. [...] At the next level there is *knowledge*, which is obtained from experts based upon actual experience. While information is data about data, knowledge is basically information about information".

In the late 1990s and beginning of the 2000s, a new field called *knowledge management* emerged and has become popular amongst people from academia and industry. Wikipedia defines knowledge management as *a range of practices used by organisations to identify, create, represent, and distribute knowledge for reuse, awareness and learning across the organisation* [32]. In this book we adopt Hansen's definition [33]:

Def. 1. Knowledge management is the dual process of accessing (searching for and identifying) and sharing (capturing and transferring) knowledge across organisational subunits.

Knowledge transfer as an aspect of knowledge management has always existed in one form or another, for example, through on-the-job peer discussions, formal apprenticeship, corporate libraries, professional training, and mentoring programmes. However, since the late twentieth century, additional theories and technologies have been applied to this task, such as knowledge bases, expert systems, and knowledge repositories. Knowledge management initiatives attempt to manage the process of creation or identification, accumulation, and application of knowledge or intellectual capital (i.e. the intangible assets of a company which contribute to its valuation) across an organisation.

1.2.2 What Is Requirements Knowledge?

Requirements knowledge can be any kind of knowledge, which emerges during requirements engineering or more generally while working with requirements:

Def. 2. Requirements knowledge consists of the implicit or explicit information that is created or needed while engineering, managing, implementing, or using requirements, and that is useful for answering requirements-related questions in any phase of a software project.

Requirements knowledge is diverse, because requirements affect different engineering activities (including design and implementation) and because requirements engineering involves different stakeholders. We distinguish between five types of requirements knowledge:

- *Domain knowledge* refers to common knowledge in a particular area or a specialised discipline. This is usually the domain, for which a system should be developed. Domain knowledge includes a vocabulary, standards used in the domain (e.g. telecommunication or banking standards), and business rules (i.e. domain constraints, standards, and regulations to be satisfied when designing or using the system).
- *Engineering knowledge* includes requirements "content", such as the requirements specifications, dependencies between requirements, as well as other artefacts needed to understand and implement the requirements such as models, test cases, or system architecture. Also informal notes and personal comments typically annotating artefacts such as models, requirements, or plans might include useful engineering information.
- *Management knowledge* includes quality measures, templates, and properties of requirements such as status, priority, and stakeholder preferences. Moreover, emerging requirements-related issues, decisions, and action items are part of this

knowledge. For example, during a requirement review, open issues, decisions, and action items on requirements might be identified, discussed, and planned.
- *Collaboration knowledge* includes information about people, their interactions, discussions, argumentation chains, and presuppositions. Discussions include information exchanged or shared between different stakeholders on various problems related to requirements. Discussion might also include requirements rationale or the reasoning behind the requirements, a crucial piece of knowledge to understand and implement requirements especially when people leave the projects. Finally, presuppositions are assumptions for realising a requirement. The lack of common understanding of presuppositions often leads to misunderstanding of requirements [34].
- *How-to Knowledge* includes information on tools, methods, and processes to be used for a particular situation while engineering and managing requirements. Organisation or vendor guidelines include information on how to perform requirements engineering activities or how to use a tool.

Requirements knowledge does necessarily exist in the form of information or data. A considerable amount of requirements knowledge such as rationale behind decisions or domain assumptions is tacit and remains in the heads of people. For example, aspects of the system that seems trivial such as the performance, usability, or localisation of a mobile phone might not be captured and discussed explicitly during the requirements engineering work.

Def. 3. Managing requirements knowledge is about efficiently identifying, accessing, externalising, and sharing all types of requirements knowledge by and to all stakeholders, including analysts, developers, and users.

Managing requirements knowledge aims at externalising tacit knowledge and solving requirements-related issues by using the dual process of accessing and sharing knowledge (see Def. 1). For example, during requirement elicitation a requirements analyst might spend weeks to access (i.e. searching and identifying) privacy regulations and laws about mobile telephony. The analyst might then share this knowledge to other stakeholders such as architects, managers, or users by *capturing* and *communicating* summaries and links to the regulations that are relevant for the envisioned system.

1.2.3 Why Managing Requirements Knowledge?

Requirements engineering, management, and implementation are complex, knowledge-intensive activities. Working with requirements involves many stakeholders from different backgrounds working in different phases and activities. To make, document, refine, or understand the requirements decisions, stakeholders need diverse information from diverse sources. For example, requirement analysts need information on the domain for defining correct and complete requirements. Change requesters need information on the processes followed for tracking the

status of their requests. Architects need information on the technologies used in order to assess the requirements feasibility [35].

Moreover, software engineering is a field where constant changes take place, making the work of stakeholders extremely dynamic. New problems are discovered (e.g. users do not want to use an extra pen to enter information to the mobile phones), new solutions are designed (e.g. new multitouch technology which enable tracking multiple fingers at same time), and experiences are made (e.g. users prefer simple user interfaces or that the operator infrastructure API only allows for a certain error tolerance) on a daily basis. The requirements knowledge in software projects is diverse and its proportions immense and continuously expanding. Thus, a systematic way of managing and treating the knowledge and its owners as valuable assets could help organisations leverage the knowledge they possess.

The need for systematic knowledge management in requirements engineering has its root in the following:

- Acquiring knowledge about the application domain. This is one of the major challenges in software engineering, since this discipline supports diverse industries such as telecom, health care, insurance, or gaming. Many software vendors are discovering more and more the importance of "mastering" domain knowledge as a way of distinction from competitors. Knowing programming languages, application programming interfaces, engineering tools, and techniques is one half of the assets of a software vendor. The other half is to know the domain, its customers, and its users.
- Capturing and using process and product knowledge. While process knowledge describes how particular tasks should be achieved and how to handle certain issues, product knowledge rather focuses on the work product itself, what it does and how it does it. Capturing and using process and product knowledge allows for shortening the time and cost for developing software systems and for increasing the quality of the delivered software.
- Acquiring knowledge about new technologies, which often affect not only the design of the system but also its goal and behaviour. New technologies can set trends and change the behaviour of users and customers and then the goal of the system itself. For example, modern hardware changed the purpose of a mobile phone from communication to a personal computing device used for work, entertainment, and communication.
- Knowing who knows what. Since software organisations get more and more distributed, this aspect becomes more and more relevant. Software projects are rarely conducted in an isolated manner and from the scratch. Even small projects carried on by a few developers often reuse open-source frameworks with complex functionality, where numerous stakeholders were involved. People get to know each other due to informal talks in the coffee hall [36]. A colleague might then report to the others about issues encountered, how they were solved, and which experiences were gained. Informal knowledge sharing and knowing "who knows what" becomes difficult in distributed settings [37, 38].

Unfortunately, managing requirements knowledge is often pushed to the extreme, that is, either formalised for the purpose of validation and completeness or considered as a "second class" citizen, creating requirements documents just because someone ordered that this task must be done. Our vision of requirements engineering and managing its knowledge is different. *We think that requirements engineering is a knowledge access and sharing activity. In addition to the validation, completeness, and formality of requirements, there must be a second dimension of efficiently capturing this knowledge and sharing it with the right people in the right context*. Managing requirements knowledge is therefore crucial to both requirements engineers and analysts as well as other stakeholders. Examples of the questions that should be answered are who needs this information? What exactly should be implemented in this feature? Why is requirement important? Which restrictions must be considered? What does this concept mean?

Systematically managing requirements knowledge brings several advantages:

- Improved understandability of requirements [39, 40] and reduced mismatch between requirements and their implementations [41]
- Identification of new requirements from the knowledge that is captured in the previous projects [42]
- Solving repetitive problems that occur in requirements engineering by systematising experiences and guiding stakeholders [43]
- Speeding up decision-making by sharing relevant information [40]
- Increased requirements reuse [44] and hence components reuse in general
- Improved evolvability of requirements by providing rationales helpful to decide on future changes [26, 45]
- Improved traceability by capturing implicit links [46] and identifying hidden interdependencies

These advantages are discussed in detail in the following chapters of this book.

1.2.4 Knowledge Management in Software Engineering

Over the last two decades, software engineering researchers have given special attention to studying developers' knowledge needs and to suggesting approaches and tools, which improve the access and sharing of software engineering knowledge [47, 48]. Some of the popular and early approaches include rationale management [27], design patterns [49], the experience factory [50], the knowledge dust-to-pearls approach [51], the personal software process [52], the team software process [53], and process-based knowledge management support for software engineering [54]. Except rationale management these approaches have been paid little attention to requirements engineering, focussing on design, implementation, and maintenance. For example, unlike architecture patterns there has been little research on systematically collecting, managing, and using requirements patterns, despite their wide usage in practice.

Also recent empirical studies on the knowledge needs in software engineering barely focussed of requirements stakeholders. For instance, Ko et al. [36] studied the information needs of source code developers at Microsoft and identified 21 questions such as "What is the programme supposed to do" or "What code could have caused this behaviour". Similarly, Sillito et al. [55] observed developers and identified 44 questions specific to software maintenance tasks. Robillard [56] studied obstacles faced by developers when reusing components. We are unaware of studies on the knowledge needs of stakeholders and questions encountered while capturing or implementing requirements. Such studies are essential for understanding the nature of stakeholders' work and providing effective tool support.

Finally, knowledge management tools such as document management systems [57], information retrieval and search tools [58], ontology-based repositories [59], wikis [60], and recommendation systems [61, 62] are getting more popular in the software engineering and more recently, also in the requirements engineering community, as the following chapters of this book show.

1.3 Foundations of Managing Requirements Knowledge

Managing requirements knowledge involves tasks, methods, and tools, which are scattered across all phases of a software engineering project. We see it as an integrated, continuous process, which includes two main activities as introduced in Def. 1: accessing and sharing knowledge [63]. There are however at least five main foundations for this process, which we introduce below and which correspond to the parts of this book. These are identifying requirements knowledge, representing requirements knowledge for reuse, sharing requirements knowledge, reasoning about requirements, and intelligent tool support. These foundations correspond roughly to the main research topics of this emergent field.

1.3.1 Identifying Requirements Knowledge (Part I)

The first goal of managing requirements knowledge is the identification of relevant knowledge, in particular, in its tacit form, answering the following questions:

- What is requirements knowledge and what are its forms and types?
- How can requirements knowledge be identified, extracted, and externalised systematically?

Identifying tacit requirements knowledge is a complex task. In projects which evolve over a long period of time, or which reuse existing frameworks and libraries, information related to requirements remains "unknown" or undocumented for "historical reasons". A mobile phone is, for example, developed over decades, and new generations are based on features of older generations. Often several

questions remain in the minds of people or "somewhere" in a non-updated requirements document: What exactly does this system or this component do and what it does not? Why is this functionality or quality provided? What are stakeholders' preferences? Or who knows more about this requirement? In contrast, design and engineering questions can at least be partly answered by studying the source code.

Moreover, requirements engineering tasks are often based on assumptions and presuppositions [34]. Customers assume that the developers know the domain, while the developers assume that the customers will tell them about everything that should be implemented – a vicious circle.

Also identifying requirements knowledge in documents and artefacts is not a trivial task. The various stakeholders might have different understanding on what is relevant knowledge, where it should be documented, with which level of detail, and how. As a result, requirements knowledge can easily get scattered across different sources including requirements documents, emails, websites, marketing brochures, contractual documents, and technical documentation. Thus, identifying requirements knowledge is also about identifying which types of information can be captured and found where.

In recent years, researchers paid more attention to the importance of (tacit) requirements knowledge and suggested novel approaches to understand, identify, externalise, and extract this knowledge. This book will discuss some of these approaches such as using machine-learning techniques to extract requirements knowledge [64] from bug repositories or formally capturing requirements using predicate logic to deduce tacit knowledge. Chapter 2 provides a theoretical foundation for tacit knowledge by considering multidisciplinary views of tacit knowledge. Chapter 3 reports on an empirical study on how recording knowledge about defects aids identification of requirements for SPLs. Chapter 4 introduces guidelines to identify and manage requirements knowledge in practice, for example, by drawing a knowledge landscape, interacting with external communities, and establishing a knowledge culture.

1.3.2 Representing Requirements Knowledge for Reuse (Part II)

Knowledge representation is a key issue in successfully applying any knowledge management programme in an organisation. Representing requirements knowledge includes two main challenges: (a) the *efficient* access by all stakeholders and (b) the support of *reuse* in case similar issues arise.

Several requirements-related tasks are repetitive, time consuming, and require a lot of human involvement [39]. For example, requirements analysis for safety critical systems may include hazard and operability analysis or failure mode and effect analysis tasks in order to identify potential system hazards and risks and to mitigate them to acceptable levels before a system is certified. Another example of repetitive time-consuming tasks can be found in large contract-based projects. There, requirements elicitation typically includes the creation of a requirements

specification document, which is used as a contractual document. Large IT service providers, which conduct similar projects in the same domain, typically spend valuable time on creating separate requirements specification documents for each project – with copy and paste as the only reuse instrument if at all. Requirements knowledge should be represented in a way that allows for reuse to cope with the repetitive tasks.

There are different approaches to represent requirements knowledge, each with advantages and disadvantages:

- Natural language: This is perhaps the most widely used approach to capture requirements knowledge because it is the most convenient for stakeholders. No tools need to be installed or new techniques learned. The disadvantage is that both querying and reuse are difficult. A promising approach from the social computing paradigm is to annotate text with tags (e.g. #screen, #version4, #issue), which can be either predefined or freely defined and refined by the stakeholders. Over time a folksonomy of tags emerge [65], which can be used to easily browse and find particular information related to a tag. One other important strength of tagging and folksonomies is that they "directly reflect the vocabulary of users" [65]. On the other hand, tags might be ambiguous and allow redundant synonyms.
- Models: Even if the software engineering community has not explicitly considered modelling as a knowledge representation approach, we think that it provides a toolkit for externalising, formalising, and communicating knowledge about complex and manifold software systems. One particular type of requirements knowledge is rationale to decisions. Researchers have suggested different decision models such as "Issue-Based Information System", "Question, Option, and Criteria", "Decision Representation Language", and "NFR Framework" and identified similarities between them. These models enable the representation of rationale knowledge but are rarely used in practice, due to the overhead needed to create them.
- Patterns: Generally speaking, patterns are solution templates to recurring problems. Patterns can be used in requirements to, for example, guide the capturing of specific types of requirements (e.g. patterns to capture use cases or NFRs). Their potential, however, are not yet exploited such as in design or management. Recurrent business rules, domain-specific issues, user tasks, or preference conflicts are just a few examples where patterns can be used to provide solution templates.
- Cases: Capturing requirements as issues with their context enables reuse by analogy, so-called case-based reasoning. Also machine-learning, heuristics, or pattern-matching techniques can be used to identify similar requirements knowledge for the purpose of reuse [64, 66].
- Ontologies: These are formal, explicit specification of shared conceptualisations, which can be understood by both machines and humans. Since the emergence of the Semantic Web standards, ontologies have become a popular alternative for representing reusable knowledge, also in requirements engineering. Several researchers have suggested ontology-based tools and methods to

capture and reason about requirements knowledge [67, 68]. This book includes an experience report on such approaches in Chap. 7.
- Formal approaches: Formal approaches have been studied for decades to capture and validate requirements. This knowledge representation approach focuses on the computer rather than on humans as its correctness is typically high but its usability and understandability is low. These approaches are especially used in to develop safety critical systems.

This book includes three chapters on representing requirements knowledge for reuse. Chapter 5 focuses on eliciting, documenting, and reusing requirements based on patterns. Chapter 6 presents an approach that combines case-based reasoning, natural language processing, and ontologies to systematise the representation of NFR knowledge, in particular security and safety. Chapter 7 presents a similar approach based on ontologies and Web 2.0, focussing on reusing domain knowledge between projects within the same domain.

1.3.3 Sharing Requirements Knowledge (Part III)

Sharing requirements knowledge forms the bridge between capture and reuse. This activity is of particular importance in large distributed projects, where the means for informal exchange "during the coffee break" or "quickly asking questions to the neighbour colleague" are limited.

Methods such as agile include instruments, which systematically encourages knowledge sharing. For instance, the daily stand-up meetings in Scrum enforce people to share the problems they have encountered and the solutions they used. Other methods such as code reviews also enforce knowledge sharing but focussing on design knowledge.

Unfortunately, software engineering processes and tools do not give enough room for sharing requirements knowledge. Most knowledge sharing occurs in meetings and during discussions or at best by delivering requirements documents between stakeholders, which might include hundreds of pages. Distributed settings, lack of domain knowledge, different vocabularies and background, as well as the complexity of requirements knowledge frequently lead to misunderstanding of these documents. It is then more about sharing data and at best information, then sharing knowledge.

Collaborative approaches such as wikis or social media bring new potentials for tightening requirements knowledge sharing. Several authors have suggested the use of wikis to capture and share requirements and related knowledge. For example, Uenalan et al. [69] argue that traditional features of requirements engineering such as projects, folders, specification modules, traceability, and baselines may be provided by simple extensions of wikis. Lohmann et al. [68, 70] introduce a promising approach based on semantic wikis, which enables all stakeholders to

collect and semantically annotate requirements. Underlying ontologies enable reasoning about various properties of requirements.

This book includes three chapters on sharing requirements knowledge amongst stakeholders. Chapter 8 focuses on global distributed project and introduces a new knowledge-sharing method and tool called PANEGA. Chapter 9 reports on an empirical study about requirements knowledge sharing in agile projects, distinguishing between performative knowledge, which occurs through actions such as question asking, gestures, or informal speeches, and lexical knowledge sharing, which occurs through inscribed texts. Chapter 10 introduces a Web 2.0 approach for identifying and prioritising stakeholders (i.e. who should know what) and reports on a large empirical evaluation of the approach.

1.3.4 Reasoning About Requirements (Part IV)

Reasoning about requirements means considering the requirements as a set rather than single entities, analysing their interdependencies to derive a new knowledge and discover inconsistencies.

Reasoning about requirements and their *interdependencies* is essential in particular for consistency and compatibility management as well as for requirements prioritisation and release planning [71]. Requirements planned for a certain release should be compatible. Incompatibilities can be triggered by not having enough time for consistency checking or by stakeholders' different perceptions and goals. Karlsson et al. [72] indicate that requirements prioritisation and planning approaches have to support handling the interdependencies. Requirements should not be treated independently: Choosing a low-cost–high-priority requirement may also entail the need to include a low-priority–high-cost requirement.

A pairwise comparison of requirements becomes infeasible for larger projects. Ramesh and Jarke [73] point out that traceability maintenance then becomes an issue and that stakeholders should focus on the traceability maintenance for the critical requirements. A common problem of traceability tools is that they do a good job in storing the relationships, but they do not provide clear semantics for the concepts used, which would enable to reason about the basic properties of a given set of requirements. Therefore, it is important to provide a means to identify the most critical dependencies [74].

Especially for informally defined requirements, the complete automation of consistency management is unrealistic [75], but semiautomated tools can help to keep the efforts acceptable. For example, Göknil et al. [76] introduce a requirement meta-model and formalise its language elements. Based on this formalisation, the authors show how to detect inconsistencies in a given instantiation of the meta-model (concrete set of requirements and their interdependencies).

A recent promising approach to reason about requirements uses semantic wiki technologies, enabling all stakeholders (especially in large, distributed settings) to collect and semantically enrich requirements [68]. In order to establish a conceptual

foundation, Lohmann et al. [68] have developed the SoftWiki ontology for RE (called SWORE [70]). This ontology defines major RE modelling concepts, such as goal, scenario, or textual descriptions. Furthermore, different types of dependencies between requirements such as "requirement A1 details requirement A" or "requirement A is in conflict with requirement C" are taken into account. Requirements are associated with stakeholders who define and maintain them. Stakeholders discuss the requirements and positively or negatively evaluate them [70]. The dependency types enable the definition of the relationships between requirements and also to reason about different properties. For example, can requirements A, B, and C be part of the same release? Existing semantic wiki-based environments applied in RE require a huge set-up overhead and are limited in the way stakeholders are supported [77].

This book includes three chapters on reasoning about requirements. Chapter 11 suggests a courteous logic-based approach to resolve inconsistency and incompleteness issues. Chapter 12 presents a rule-based approach for detecting dead features and defects in variability. Chapter 13 discusses how reasoning about requirements and their interdependencies should also be propagated to the other activities such as design and implementation.

1.3.5 Intelligent Tool Support (Part V)

Requirements knowledge can become huge and scattered across different sources. Much effort is needed to identify and retrieve relevant information in requirements repositories. This would entail an overhead of capturing, maintaining, and accessing requirements knowledge.

To address these problems, researchers started investigating techniques like data mining, social network analysis, and recommendation technologies and developing information retrieval tools to enable efficient capture, access, and sharing of requirements knowledge.

Recently, traditional requirements databases have been enhanced such that data is modelled and stored in a way that allows learning and querying. Furthermore, researchers have started investigating how recommendation technologies [64, 78] can be applied to existing requirements infrastructures and tools.

Intelligent tool support is crucial for the implementation of any requirements knowledge management programme. Thereby intelligent does not only mean the ability of the tool to reason about knowledge, derive new knowledge, or deal with incomplete and scattered knowledge. Intelligent also means integrated and pragmatic solutions, which neither require additional learning effort, nor impose heavyweight processes and workflows, nor introduce new interruptions to the stakeholders workflows, for example, by having to switch back and forth between tools.

This book includes three chapters on intelligent tool support for managing requirements knowledge. Chapter 14 introduces various recommendation technologies

and relevant discusses visionary scenarios of applying them to support stakeholders' tasks. Chapter 15 proposes the use of experience-based tools to improve the quality of software requirements specification by learning from previous experiences. Finally, Chap. 16 introduces the requirements modelling framework, which is integrated into the Eclipse Development Environment allowing a traceability of different types of knowledge such as natural language requirements and formal models.

1.4 Summary

In this chapter, we reviewed the concepts of requirements engineering from the knowledge management perspective. We discussed the needs for establishing a new field for managing requirements knowledge and defined its key concepts. Finally, we introduced five foundations for this field: identifying requirements knowledge, capturing requirements knowledge for reuse, sharing requirements knowledge, reasoning about requirements, and intelligent tool support. These foundations are discussed in detail in the corresponding parts of the book.

Acknowledgements We are grateful to Pete Sawyer, Smita S Ghaisas, Yang Li, and Zardosht Hodaie for their constructive reviews.

References

1. Bruegge B, Dutoit A (2010) Object-oriented software engineering using UML, Patterns, and Java, vol 3. Prentice Hall, Upper Saddle River
2. Kang K, Cohen S, Hess J, Nowak W, Peterson S (1990) Feature-oriented domain analysis (FODA) feasibility study. Technical report, CMU/SEI-90-TR-21. Software Engineering Institute, Carnegie Mellon University, Pittsburgh
3. Institute of Electrical and Electronic Engineers (1990) IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990). Institute of Electrical and Electronics Engineers, New York
4. Clements P, Northrop L (2006) A framework for software product line practice-version 4.2 (2006). Carnegie Mellon, Software Engineering Institute, Pittsburgh. http://www.sei.cmu.edu/prodvolnuctlines/framework.html. Last visited Nov 2012
5. Davis A (2003) The art of requirements triage. IEEE Comput 36(3):42–49
6. Sommerville I, Sawyer P (1997) Requirements engineering: a good practice guide. Wiley, New York
7. Nuseibeh B, Easterbrook S (2000) Requirements engineering: a roadmap. In: Proceedings of the conference on the future of software engineering (ICSE'00). ACM, New York, pp 35–46
8. Aurum A, Wohlin C (2003) The fundamental nature of requirements engineering activities as a decision-making process. Inf Softw Technol 45(14):945–954
9. Pohl K (1996) Process-centered requirements engineering. Wiley, New York
10. Hofmann H, Lehner F (2001) Requirements engineering as a success factor in software projects. IEEE Softw 18(4):58–66

11. Yang D, Wu D, Koolmanojwong S, Brown A, Boehm B (2008) Wikiwinwin: a wiki based system for collaborative requirements negotiation. In: Proceedings of the HICCS, p 24, Waikoloa
12. Leffingwell D (1997) Calculating the return on investment from more effective requirements management. Am Program 10(4):13–16
13. Firesmith D (2004) Prioritizing requirements. J Object Technol 3(8):35–47
14. Gartner Group (2011) Hype cycle for application development: requirements elicitation and simulation. Gartner Group
15. Dorfman M, Thayer RH (1997) Software requirements engineering. IEEE Computer Society Press, Los Alamitos
16. Pohl K, Böckle G, van der Linder F (2005) Software product line engineering foundations, principles, and techniques. Springer, New York
17. Software Engineering Institute (2012) Product line hall of fame. http://www.sei.cmu.edu/productlines/plp_hof.html
18. Smith G (2000) The object-Z specification language, Advances in formal methods series. Kluwer, Boston
19. Creighton O, Software Cinema (2006) Employing digital video in requirements engineering. Dissertation, Technische Universtät München
20. Neill CJ, Laplante PA (2003) Requirements engineering: the state of the practice. IEEE Softw 20(6):40–45, IEEE CS
21. Peterson M (2009) An introduction to decision theory. Cambridge University Press, Cambridge/New York
22. Cooke S, Slack N (1984) Making management decisions. Prentice Hall, Englewood cliffs
23. Kunz W, Rittel H (1970) Issues as elements of information systems. Working paper no. 131. University of California at Berkeley, Institute of Urban and Regional Development, Berkeley
24. MacLean A, Young RM, Bellotti VME, Moran TP (1991) Questions, options, and criteria: elements of design space analysis. Hum Comput Interact 6(3):201–250
25. Dutoit AH (1996) Rationale management in requirements engineering. Ph.D. dissertation, Carnegie Mellon University
26. Dutoit A, Paech B (2003) Eliciting and maintaining knowledge for requirements evolution. In: Aurum A, Jeffery R, Wohlin C, Handzic M (eds) Managing software engineering knowledge. Springer, Berlin
27. Dutoit A, McCall R, Mistrik I, Paech B (2006) Rationale management in software engineering. Springer, Berlin
28. Damian D, Zowghi D (2003) Requirements engineering challenges in multi-site software development organizations. Requir Eng J 8:149–160
29. Chisholm RM (1982) The foundations of knowing. The University of Minnesota Press, Minneapolis
30. Resher N (2003) Epistemology: an introduction to the theory of knowledge. State University of New York Press, Albany
31. Thierauf RJ (1999) Knowledge management systems for business. Praeger
32. Wikipedia, the free encyclopaedia (2012) http://en.wikipedia.org/wiki/Knowledge_management. Last visited in Nov 2012
33. Hansen MT (1999) The search-transfer problem: the role of weak ties in sharing knowledge across organization subunits. Adm Sci Q 44(1):82–111
34. Ma L, Nuseibeh B, Piwek P, De Roeck A, Willis A (2009) On presuppositions in requirements. In: 2nd international workshop on managing requirements knowledge, MaRK'09 IEEE, Atlanta, USA, pp. 27–31
35. Finkelstein A, Kramer J, Nuseibeh B, Finkelstein L, Goedicke M (1992) Viewpoints: a framework for integrating multiple perspectives in system development. Int J Softw Eng Knowl Eng 2–1:31–57
36. Ko AK, DeLine R, Venolia G (2007) Information needs in collocated software development teams. In: Proceedings of the 29th international conference on software engineering, Minneapolis, USA, pp 344–353

37. Herbsleb JD, Mockus A (2003) An empirical study of speed and communication in globally-distributed software development. IEEE Trans Softw Eng 29(6):481–494
38. Milewski AE, Tremaine A, Egan R, Zhang S, Kobler F, O'Sullivan P (2008) Guidelines for effective bridging in global software engineering. In: Proceedings of the 2008 I.E. international conference on global software engineering, pp 23–32. IEEE Computer Society, Washington, DC
39. Daramola O, Stålhane T, Omoronyia I, Sindre G (2013) Using ontologies and machine learning for hazard identification and safety analysis. In: Managing requirements knowledge. Springer
40. Ghaisas S, Ajmeri N (2013) Knowledge-assisted ontology-based requirements evolution. In: Managing requirements knowledge (Chapter 7 in this volume). Springer, Heidelberg
41. Soffer A, Dori D (2012) Model-based requirements engineering framework for systems lifecycle support. In: Managing requirements knowledge (Chapter 13 in this volume). Springer, Heidelberg
42. Lutz R, Lavin M, Lux J, Peters K, Rouquette NF (2013) Mining requirements from operational experience. In: Managing requirements knowledge (Chapter 3 in this volume). Springer, Heidelberg
43. Franch X, Quer C, Renault S, Guerlain C, Palomares C (2012) Constructing and using software requirements patterns. Springer
44. Carrillo de Gea JM, Nicolás J, Alemán JLF, Toval A, Vizcaíno A, Ebert C (2013) Reusing requirements in global software engineering. In: Managing requirements knowledge (Chapter 8 in this volume). Springer, Heidelberg
45. Thurimella AK, Bruegge B (2012) Issue-based variability management. Inf Softw Technol 54(9):933–950
46. Narayan N, Delater A, Paech B, Bruegge B (2011) Enhanced traceability in model-based CASE tools using ontologies and information retrieval. In: Proceedings of the 4th international workshop on managing requirements knowledge (MaRK'11), Trento
47. Bjørnson FO, Dingsøyr T (2008) Knowledge management in software engineering: a systematic review of studied concepts, findings and research methods used. Inf Softw Technol 50:1055–1068
48. Lago P, van Vliet H, Babar MA, Dingsoyr T (eds) (2009) Software architecture knowledge management: theory and practice, 1st edn. Springer
49. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley, Reading
50. Basili VR, Caldiera G, Rombach DH (1994) The experience factory, Encyclopedia of software engineering – 2 volume set. Wiley, New York, pp 469–476
51. Basili V, Costa P, Lindvall M, Mendonca M, Seaman C, Tesoriero R, Zelkowitz M (2001) An experience management system for a software engineering research organization. In: Proceedings of the 26th annual NASA Goddard Software engineering workshop. Greenbelt, Maryland, USA
52. Humphrey WS (2005) PSP: a self-improvement process for software engineers. Addison-Wesley, Reading. ISBN 03213054931
53. Humphrey WS (1999) Introduction to the team software process. Addison-Wesley, Reading. ISBN 0-201-47719-X
54. Holz H (2003) Process-based knowledge management support for software engineering. Doctoral dissertation. University of Kaiserslautern, dissertation.de Online- Press
55. Sillito J, Murphy GC, De Volder K (2008) Asking and answering questions during a programming change task. Trans Softw Eng 34:434–451
56. Robillard MP (2009) What makes APIs hard to learn? Answers from developers. IEEE Softw 26:27–34
57. Rus I, Lindvall M, Sinha SS (2001) Knowledge management in software engineering: a state-of-the-art-report. Fraunhofer Center for Experimental Software Engineering Maryland and the University of Maryland for Data and Analysis Center for Software, Department of Defence
58. Bajracharya S, Lopes C (2009) Mining search topics from a code search engine usage log. In: Proceedings of the 2009 6th IEEE international working conference on mining software repositories (MSR'09). IEEE Computer Society, Washington, DC, pp 111–120

59. Happel H-J, Maalej W, Seedorf S (2010) Applications of ontologies in collaborative software development. In: Mistrík I, Grundy J, van der Hoek A, Whitehead J (Hrsg.) Collaborative software engineering. Springer, Berlin/Heidelberg. ISBN 978-3642102936
60. Aguiar A, Dekel U, Merson P (2009) Wikis4SE'2009: Wikis for software engineering. ICSE companion 2009, pp 480–481
61. Happel H-J, Maalej W (2008) Potentials and challenges of recommendation systems for software development. In: RSSE'08: proceedings of the 2008 international workshop on recommendation systems for software engineering, ACM
62. Robillard MP, Walker RJ, Zimmermann T (2010) Recommendation systems for software engineering. IEEE Softw 27(4):80–86
63. Maalej W, Thurimella A (2013) DUFICE – guidelines for a lightweight management of requirements knowledge. In: Managing requirements knowledge. Springer
64. Dumitru H, Gibiec M, Hariri N, Cleland-Huang J, Mobasher B, Castro-Herrera C, Mirakhorli M (2011) On-demand feature recommendations derived from mining public product descriptions. ICSE 2011, pp 181–190
65. Mathes A (2004) Folksonomies: cooperative classification and communication through shared metadata. In: Computer mediated communication – LIS590CMC http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html
66. Jürgens E, Deissenboeck F, Feilkas M, Hummel B, Schätz B, Wagner S, Domann C, Streit J (2010) Can clone detection support quality assessments of requirements specifications? ICSE (2): 79–88
67. Ajmeri N, Vidhani K, Bhat M, Ghaisas G (2011) An ontology-based method and tool for cross-domain requirements visualization. In: Fourth workshop on managing requirements knowledge, MaRK11, pp 22–23, Trento
68. Lohmann S, Heim P, Auer S, Dietzold S, Riechert R (2008) Semantifying requirements engineering – the softWiki approach, I-SEMANTICS, Graz, pp 182–185
69. Uenalan O, Riegel N, Weber S, Doerr J (2008) Using enhanced wiki-based solutions for managing requirements. First international workshop on managing requirements knowledge (MARK), Barcelona, Spain, pp 63–67
70. Lohmann S, Riechert T, Auer S (2008) Collaborative development of knowledge bases in distributed requirements elicitation. Software engineering (workshops): agile knowledge sharing for distributed software teams, Munich, Germany, pp 22–28
71. Ruhe G, Saliu M (2005) The art and science of software release planning. IEEE Softw 22(6):47–53
72. Karlsson J, Olsson S, Ryan K (1998) Improved practical support for large-scale requirements prioritization. Require Eng J 2(1):51–60
73. Ramesh B, Jarke M (2001) Toward reference models for requirements traceability. IEEE Trans Softw Eng 27(1):58–93
74. Dahlstedt A, Persson A (2003) Requirements interdependencies – moulding the state of research into a research agenda, REFSQ'03, Klagenfurt, pp 71–80
75. Iyer J, Richards D (2004) Evaluation framework for tools that manage requirements inconsistency. In: 9th Australian workshop on requirements engineering (AWRE'04). Adelaide, Australia
76. Göknil A, Kurtev I, and van den Berg K (2008) A metamodeling approach for reasoning about requirements. In: 4th European conference on model driven architecture – foundations and applications, Berlin. LNCS, vol 5095, pp 310–325, Berlin
77. Hoenderboom B, Liang P (2009) A survey of semantic wikis for requirements engineering. Technical report RUG-SEARCH-09-L03, University of Groningen
78. Mobasher B, Cleland-Huang J (2011) Recommender systems in requirements engineering. AI Mag 32(3):81–89
79. Glinz M (2007) On non-functional requirements. In: 15th IEEE international requirements engineering conference, New Delhi, 15–19 Oct 2007, pp 21–26

http://www.springer.com/978-3-642-34418-3

Managing Requirements Knowledge
Maalej, W.; Thurimella, A.K. (Eds.)
2013, XV, 398 p., Hardcover
ISBN: 978-3-642-34418-3