Chapter 8
Reusing Requirements in Global Software Engineering

Juan Manuel Carrillo de Gea, Joaquín Nicolás, José Luis Fernández Alemán, Ambrosio Toval, A. Vizcaíno, and Christof Ebert

Abstract Knowledge sharing and reuse in global software engineering (GSE) are challenging issues. Knowledge management (KM) is specifically impacted because on top of distance, culture and language mismatches, there is also the perceived risk of sharing something which could mean that others could take over some work. Mistrust and protectionism are often the consequence, leading to insufficient reuse. This is visible specifically in requirements engineering (RE), where all reuse should start. In this chapter, we will look to reuse in RE with a detailed look on how to improve knowledge sharing and collaboration in distributed environments. We first look into the state of the practice. Then we present a lightweight, reuse-based, global RE method called PANGEA (*Process for globAl requiremeNts enGinEering and quAlity*), based on natural language requirements and software engineering standards. Based on this method, we also build a prototypical tool, called PANTALASA (*PANgea Tool And Lightweight Automated Support Architecture*) which provides automated support for PANGEA. Its features are drawn from PANGEA and the state of the practice commercially available RE tools. A prototype of PANTALASA was developed by using Semantic MediaWiki and Facebook and applied to a case study in the domain of hotel management. We could show with this method and prototype that collaboration and thus KM and reuse in RE are improved.

J.M. Carrillo de Gea (✉) • J. Nicolás • J.L. Fernández Alemán • A. Toval
Universidad de Murcia, Murcia, Spain
e-mail: jmcdg1@um.es; jnr@um.es; aleman@um.es; atoval@um.es

A. Vizcaíno
Universidad de Castilla - La Mancha, Ciudad Real, Spain
e-mail: aurora.vizcaino@uclm.es

C. Ebert
Vector Consulting Services, Stuttgart, Germany
e-mail: christof.ebert@vector.com

W. Maalej and A.K. Thurimella (eds.), *Managing Requirements Knowledge*,
DOI 10.1007/978-3-642-34419-0_8,

8.1 Introduction

Brooks stated more than 20 years ago [1] that there are approaches in software engineering (SE) that target the *accidental* complexity of software, while others are targeted at its *essential* complexity. As regards this difficulty, and discussing the role of requirements engineering (RE) in software development, Brooks considered the *refinement of requirements and rapid prototyping*. Among the SE strategies which attack the conceptual essence of the problem, he also mentions the idea of *buy* versus *build*: in other words, he stated the need for software reuse. Software reuse is for Meyer [2] "the ability of software elements to serve for the construction of many different applications". Meyer summarises the benefits of reusability: (1) improved timeliness, in the sense of decreased time to market; (2) reduced software maintenance efforts; (3) improved reliability, efficiency and consistency of the developed software and (4) enhanced investment, through the preservation of the know-how. Mili et al. [3] affirm that software reuse is "the (only) realistic approach to bring about the gains of productivity and quality that the software industry needs".

There is a line of thinking within software reuse that considers that every software artefact (asset) produced during the development process is reusable, including system or product specifications, design, source code, test cases, project plans, quality plans, etc. Since the mid-1990s, specifications and requirements reuse are postulated by a number of authors as a promising path towards quality and productivity in software development, a way that has been less explored than reuse of source code or designs. There is some consensus, in that the higher the abstraction level and the more not only source code but also design and specifications are reused, the greater the reusability benefits are [4, 5]. In this respect, Favaro [6] affirms that "a well-formulated, measurable, reusable requirement [. . .] is every bit as valuable as a reusable software module". In addition, Robertson and Robertson [7] claim that if the development starts with a set of requirements that were specified for other projects or domains, the accuracy of the requirements specification is improved, and the time to develop this specification is reduced. Cheng and Atlee [8] also consider the identification of sets of reusable requirements for particular domains or types of applications to be of interest. To the best of our knowledge, Rine and Nada [9] are the first authors who demonstrated empirically that the reusability level determines the effectiveness of improvements in productivity, quality and development time, concluding that greater benefits are obtained when reusability is applied during the initial processes of the software development life cycle.

Cheng and Atlee [8] highlighted *globalisation* and *requirements reuse* as two of the more urgent needs and grand challenges in RE research and expected the solutions to these issues to produce a great impact on both research and practice in SE. Globalisation arises from growing and relatively new software needs, while requirements reuse focuses on extending and maturing existing technologies. Global software engineering (GSE) implies a paradigm shift towards globally

distributed development teams [10], and it has become a business need for various reasons: to decrease costs, capitalise on global resource pools owing to the scarcity of resources, locate development closer to the customers, exploit around-the-clock development to achieve cycle-time acceleration and cater to local markets [11]. In contrast, GSE leads to an increased risk of communication gaps, given the temporal, geographic, cultural and linguistic nature of the distance imposed by GSE [12], which might hinder collaborative activities that require stakeholders to share a mental model of the problem and requirements [8]. Indeed, due to its collaboration-intensive nature, RE presents several specific challenges and difficulties when the stakeholders are distributed [13, 14]. Gallardo-Valencia and Sim [15] are convinced that the success of a software product depends on the proper understanding of requirements among stakeholders. Herbsleb [10], for his part, emphasised that "getting the requirements right, and dealing with unstable requirements" are notoriously difficult problems to address, even in a traditional, collocated environment. A shared understanding of the requirements is even more difficult to achieve in a GSE context, "both because of loss of context and loss of communication". As well as this, Herbsleb pointed out that research on *eliciting and communicating requirements* has made substantial progress in addressing the issues posed by globalisation, although impediments still exist.

Knowledge is considered to be one of the major resources of an organisation, and this is further emphasised in organisations dedicated to software development. SE and RE are knowledge-intensive activities [15, 16] and hence, the growing interest of software development organisations in providing methods to help with its appropriate management [17]. By means of knowledge management (KM), software development organisations might obtain certain potential benefits: decrease the development time and cost of software projects, avoid mistakes and reduce rework, increase productivity through repetition of successful processes, increase quality and make better decisions [18]. Thus, achieving good KM is very important if competitive levels are to be maintained in an increasingly globalised and demanding world. Nevertheless, challenges to KM increase when the development activities are geographically distributed [16]. According to Ebling et al. [19], proposals related to KM challenges in the field of RE in distributed software development are needed; they encourage further research on these issues.

Both GSE and requirements reuse are therefore relevant approaches for the software industry. As far as we know, however, there are no proposals which tackle both GSE and requirements reuse together. Since one of the current problems in GSE is the existence of knowledge which is not properly shared and reused, KM and awareness in distributed settings is a challenging task [20] which might be addressed by reusing, sharing and collaboration mechanisms in GSE. The proposal which is presented here treats knowledge in the form of natural language requirements and aims at laying out the basis for a reuse-based RE method for GSE environments, named PANGEA (*Process for globAl requiremeNts enGinEering and quAlity*). It also includes PANTALASA (*PANgea Tool And Lightweight Automated Support Architecture*), the automated support for the PANGEA method.

The rest of the chapter is organised as follows: Sect. 8.2 provides an overview of the field. Current and relevant challenges for both RE and KM in GSE are studied in Sect. 8.3. Section 8.4 includes our proposal for a method to address these issues. Section 8.5 focuses on the tool architecture supporting this method. Section 8.6 reports on a preliminary evaluation of both method and tool. Finally, our conclusions and future work are presented in Sect. 8.7.

8.2 Foundations

At this point, and in the context of this chapter, a brief description of KM, architectural knowledge management (AKM) and KM as the basis for RE is provided.

8.2.1 Knowledge Management

Al-Ani [21] cites a generally accepted definition for the term *knowledge*. It is located at the top of a hierarchical structure, and this representational structure sees *information* as standing above *data* and knowledge standing above of both. In the model described, data is processed and transformed into information; information is then interpreted and contextualised by individuals and transformed into knowledge. In addition, the term KM suggests that knowledge is something tangible which is possible to manipulate. According to Ebert and De Man [22], KM is "the process that deals with systematically eliciting, structuring and facilitating the efficient retrieval and effective use of knowledge".

In those organisations involved in GSE projects, an adequate KM is necessary to mitigate those factors derived from the geographical, temporal and sociocultural distance [23–25] that might hamper communication and relationships between stakeholders. Indeed, knowledge changes quickly during software development, and so all the knowledge generated in the project should be made as accurate, complete and updated as possible. Furthermore, if software development is distributed globally, many more people are involved in the development activities, and thus organisations tend to have problems in terms of content, location and use of knowledge that can make it difficult to take advantage of this knowledge. Moreover, Al-Ani [21] states that ineffective KM can lead to disastrous consequences, showing some examples and lessons learnt which illustrate ineffective KM practices and their aftermath.

During the activities of the software life cycle, knowledge which is important for the subsequent activities is generated [26], and these are commonly carried out by different people to those involved previously. It is important to ensure that this knowledge is accessible for them. The software development process generates documents and other software engineering artefacts. In this sense, KM plays a very

important role, since this knowledge has to be captured, and it emerges from the solution to problems encountered during the course of past and current projects. However, some common issues in software development organisations are actually problems in the flow of knowledge, e.g. lack of documentation [27]. To be successful, organisations that apply KM techniques and processes should create an organisational culture that fosters and promotes the dissemination and sharing of knowledge among its employees, through encouraging them to document and store their knowledge in a KM repository [18]. This is especially true in GSE settings, where according to Ebert and De Neve [28], a relevant step towards creating such a common culture is to choose a specific, common language to be used within the organisation. However, these authors highlight that since "a common syntactical language does not necessarily mean the same semantics and pragmatics", team members should rotate across locations to live within different cultures and gradually build mutual understanding. On the other hand, this issue might be also addressed by means of *ontologies*, which formalise concepts and relationships among them as well as enable automatic reasoning with knowledge [29].

8.2.2 Architectural Knowledge Management

According to Beecham et al. [30], AKM involves capturing, sharing and managing the information resulting from the software architecture process, in the form of knowledge of the problem domain, the solution domain and knowledge artefacts used throughout the whole process. It supports the creation, storage and dissemination of all the knowledge used in defining and using the architecture, including the requirements documentation [31] and the functional and non-functional requirements [30]. In fact, there are similarities between requirements and architecture [32]. We could go even further; Clerc [33] notices "a close resemblance between a set of requirements for a software system and a set of architectural decisions taken for that software system: what one person regards as requirements for a software system, another person may regard as architectural decisions".

Architectural knowledge can serve to support the collaboration needs of distributed software development organisations [31]. In addition, as was stated above, KM is even more challenging in a GSE context [16], and exactly the same situation occurs in the case of AKM [30, 31], which faces the same risks as RE in GSE settings [33] (for a detailed study of threats and safeguards in RE for GSE environments, see, e.g. [34]). There is a need to capture, share and manage architectural knowledge, in particular in the form or requirements, among different and distributed sites, but the related tasks become more demanding than in a collocated setting. Thus, the challenges of GSE have to be addressed by the members of globally distributed teams by relying on appropriate AKM practices; Beecham et al. [30] identified that one important area to achieve AKM is represented by KM practices for creating and disseminating architectural knowledge.

In this context, Clerc [33] identified seven essential KM practices to achieve effective AKM in GSE environments that build on the RE discipline. In summary, these practices are (1) *frequent interaction across sites*, encouraging team members to interact frequently with each other; (2) *cross-site delegation*, improving integration of distributed teams by means of mutual delegation of team members from a local site to a remote site; (3) *face-to-face project kickoff meetings*, assisting the establishment of initial relationships between and among teams; (4) *urgent request*, identifying expert project members to collect information on a given topic quickly; (5) *collocated high-level architecture phase*, creating a sound high-level architecture efficiently; (6) *clear organisation structure with communicating responsibilities*, maintaining clear lines of communication among stakeholders' roles; and (7) *establishing of a repository for architecture artefacts*, built to store architectural knowledge. Moreover, Ebert and De Neve [28] emphasise that customer requirements might be mapped to architectural units in order to cluster activities and split a globally distributed project into different, collocated work teams. In this way, each team assumes the responsibility for a set of functionally related customer requirements, and teamwork is therefore reinforced.

According to Desouza et al. [35], there are three distinct strategies for AKM: (1) *codification*, which refers to the use of a central repository for storing the architectural knowledge; (2) *personalisation*, which alludes to stakeholders and the interaction between them to get the knowledge when they needed it; and (3) a *hybrid approach*, in which there is a central knowledge repository shared among stakeholders, addressing questions such as "when", "how" or "who" in relation to knowledge, in order to enable personalised knowledge sharing.

8.2.3 Knowledge Management for Requirements Engineering

Regarding reusability, there are various studies which have centred on applying KM in software development organisations, most of which focus on the reuse of experiences, such as best practices or lessons learned, in order to improve the quality of processes and products or to facilitate the reuse of software artefacts [36–38]. In this context, Rus and Lindvall [18] consider software reuse to be a KM activity that supports software development. These authors affirm that repeated implementation by programmers of the same or very similar solutions, along with the rework resulting from this, might be avoided or reduced through establishing a reuse repository, which would contain software previously submitted by others, and "this same concept can apply to all software engineering artefacts". This approach requires a change in the software development process, since the first step would be to search the repository for reusable parts developing the solution from scratch only if nothing useful is found. Furthermore, information is often reused with high redundancies or manual overhead, eventually leading to rework or even errors in the product [22]. Thus, only reuse of information is not enough, and knowledge

should be embedded into integrated workflows for specific tasks. This strategy "generates immediate returns by making engineers more flexible".

For Gallardo-Valencia and Sim [15], requirements knowledge is ideally captured in a requirements specification document using a written format, although such written knowledge is complemented by requirements knowledge that is shared in an informal manner through conversations among and between stakeholders. Moreover, Maalej et al. [39] affirm that it is necessary to capture and share tacit knowledge about requirements and make it explicit, in order to be able to manipulate it, because (1) reuse is enhanced, (2) traceability is enabled, (3) requirements evolution is supported and (4) collaboration between participants in distributed projects is improved. Besides, Ma et al. [40] have noted that the presence of tacit knowledge might have a negative impact on communication through requirements documents; such knowledge should therefore be properly managed with the intention of avoiding miscommunication, misinterpretation and inappropriate contextualisation among stakeholders, especially in the case of a GSE context [21]. Even though recent technologies and advancements have boosted KM support within distributed software development teams [21], in particular KM support to RE [39], usual technologies and infrastructures typically focus only on addressing issues related to the management of explicit knowledge, whereas they should capture, formalise and manipulate tacit or implicit knowledge [21, 39]. Maalej et al. [39] demand improvements in RE processes and tools in order to achieve better management of requirements knowledge, owing in particular to the growing tendency towards agile methods and distribution in software development, while "the major constraint is to have a lightweight, usable, intelligent and personalised capturing and sharing approach".

Another key issue for achieving successful results in global RE is requirements awareness. Informal communication is also important in global, distributed development because it contributes to project awareness [41]. In a software development setting, and in particular in RE, *awareness* happens if "a software developer working in a project has knowledge of events, such as changes to a requirement suggested by a customer, that occur in the project" [42]; this becomes even more essential in GSE [13]. Kwan et al. [42] conducted an industrial study of two GSE projects; one of them is an example of offshore outsourcing, and the other project has outsourced collocated development. These authors identified three main factors that produce certain effects which might influence awareness: (1) the distributed development reduces awareness, (2) the experience of the team members bridges awareness gaps and (3) the centralised communication structure might prevent awareness problems. All this being so, project members should keep abreast of any issues that take place in the scope of the project. The lessons learned are (1) experienced team members should be accessible; (2) a centralised communication structure can help new teams to remain aware, whereas a decentralised structure decreases communication bandwidth and improves response times; (3) frequent meetings improve awareness among distributed sites and (4) each distributed team member should be assigned to a set of stable requirements and an unstable requirement, in order to allow him or her to experience minimal downtime when there are delays. In conclusion, requirements awareness is neither a banal nor an easy issue, and a lack of awareness can lead to problems with design, quality and cost within the distributed project [42].

8.3 Practical Challenges

Distance hinders KM and requirements management processes in GSE. Issues concerning GSE have been given a lot of attention in literature in the last years. To begin with, Cheng and Atlee [8] affirm that new or extended techniques are needed to overcome the challenges posed to RE by globalisation; they are the following: (1) obtain proper support to outsourcing of downstream software development tasks (e.g. design, coding, testing, etc.), bearing in mind that distance complicates the collaboration between the requirements and the development teams, and (2) enable effective distributed RE activities, since analysts and geographically distributed stakeholders will work together and distributed software development teams might work with in-house customers; practitioners therefore need techniques to facilitate distributed requirements elicitation, modelling, negotiation and management of distributed teams.

Ebling et al. [19] have conducted a systematic literature review of RE in distributed software environments, concluding that the challenges identified in the field of KM issues are related to inappropriate sharing of requirements information with distributed stakeholders [10, 41], which eventually damages the interaction between them [13]. Moreover, there are no available methods, models, techniques or approaches to RE in GSE environments in relation to the KM challenges identified [19].

Process mismatches, differing technical and domain vocabularies, incompatible environments and conflicting assumptions can be particularly problematic in a GSE context [43]. Cultural differences can also pose formidable challenges for achieving a shared understanding of the requirements [44], and all these factors can hamper discovery and integration of knowledge [10]. Knowledge transfer is "the transmission of general or project specific knowledge which is needed to understand and execute the project requirements" [45]. It is also a challenging activity of KM in GSE environments, because of the significant reduction in communication frequency and speed among remote teams [46], which leads knowledge to become fixed to locations in which it is produced, hindering the transfer of such knowledge from one site to another [47].

Manteli et al. [16] classify the challenges of KM in GSE under three main categories: *communication*, *knowledge creation and storage* and *knowledge transfer*. The coordination of communication between remote teams is included in KM [20, 23], but it is a fact that distance introduces barriers to both informal and face-to-face communication in GSE. Project members have to rely on synchronous communication tools (e.g. chats, phone calls, videoconferences), or asynchronous ones (e.g. discussion forums, emails), in order to collaborate [41]. Since communication speed and frequency is relevant in a GSE context, any communication delay can slow down or even detain the project course, producing delivery delays [16], so synchronous communication is generally preferable in distributed environments [48], as it boosts real-time, interactive communication and improves collaboration among stakeholders. By means of synchronous communication tools, analysts can check

each other's work, see if certain features have been implemented the right way or solve problems together and assist each other. Nevertheless, the most appropriate communication media depend on the team member's role [16]. It is also worthwhile to reduce tasks by coupling as much as possible, since the interdependencies among distributed tasks introduce important communication overload, thus affecting communication speed and frequency between distributed sites [16].

The effectiveness of knowledge capture (the process of making it explicit), that is, how knowledge is captured into software development artefacts and acquired by other team members, is a critical success factor for projects [49]. Among the distinct KM strategies for knowledge capture and management [35], *codification* is pursued when knowledge is documented and stored in a central repository, *personalisation* relies on the tacit knowledge of stakeholders and knowledge sharing through person-to-person communications and the *hybrid approach* combines the previous two, being the recommended strategy towards AKM in GSE [50]. An important challenge for KM is found in personalisation strategies, which are typical of agile development methods, in which employees are encouraged to cooperate with each other, taking initiative and responsibility without being constrained by strictly defined processes [16]. Furthermore, with this approach, much of the knowledge remains tacit, and the explicit knowledge is not always updated in the corresponding documents. However, documentation has an important role [27], and it should consequently be updated; it ought to reflect what distributed teams are working on, in order to keep requirements awareness [42]. According to Manteli et al. [16], the use of different repositories and tools for storing and sharing documents is not recommended. The knowledge search through all these resources to locate the right document and the up-to-date information is complicated, leading in turn to the adoption of local codification strategies that hamper knowledge sharing between sites. In addition, if no appropriate mechanisms for storing and sharing documentation are provided, more communication between distributed project members is needed, decreasing stakeholders' productivity.

Knowledge transfer was identified as a critical success factor for software development projects with an onsite/offshore structure [45], since some circumstances obstruct knowledge transferability across sites, including the use of different working methods [51] or the differences in skills and expertise of remote project members [30]. Manteli et al. [16] point out that when the greater part of a software development project is developed at one site, most of the *system-generic* knowledge ("the comprehensive knowledge of the entire system that teams are working on") resides only there, which causes knowledge to be fixed to that location [47]. It thus takes additional effort for that knowledge to be transferred to the remote sites, which only retain the *unit-specific* knowledge ("the particular knowledge that the individual has, for the specific unit he or she is working on"). Another challenge in knowledge transfer is "how" to locate the knowledge [16]; an effective knowledge-sharing strategy should enable project members to know "who", in addition to providing the ability to know "what" and "where" knowledge resides [52]. This approach is also known as *transactive memory* [51], and

according to Kotlarsky and Oshri [53], it constitutes a means of knowledge sharing that contributes decisively to successful collaboration between and among remote teams. A personalisation strategy in which people transfer more knowledge in a person-to-person way leads to know "who" knows "what" more efficiently [16].

Tools help in managing requirements and are key success factors in GSE [28, 54]. Moreover, Ebert [54] affirms that change management is unmanageable without automated tools in a GSE environment. Traceability facilitates change management and must include horizontal and vertical dependencies – between artefacts in the same and in different abstraction levels, respectively. Heindl et al. [55] detected a lack of traceability and computer-aided requirements engineering tools in RE for GSE (from now on, these will be referred to as CARE tools or simply RE tools). Mistrík et al. [56] point out that a considerable number of recent advances in collaborative SE are related to the development of supporting tools for certain collaborative practices. Portillo-Rodríguez et al. [24] have conducted a systematic literature review of GSE tools supporting the ISO/IEC 12207 processes, concluding that the majority of the tools were developed within research groups or labs. In addition, all the tools analysed included web-based or client–server access, as well as communication and coordination features for globally distributed teams. Herbsleb [10] highlighted that *environments and tools* are important areas of research in GSE. This author affirms that awareness and communication are relevant and interrelated issues, since the reduced communication bandwidth in GSE makes it much more difficult to face the problem of understanding what other project members are doing and thus coordinate effectively with them. This means that synchronous and asynchronous communication features should be integrated into RE tools, avoiding practices that might lead to low efficiency and productivity, such as using e-mail instead of web-based tools or shared repositories to manage requirements [57]. Besides the awareness and communication features, Herbsleb [10] detected a need for collaborative capabilities to be integrated into the development environment, as well as more "interoperable tools with standard data formats and interaction protocols". In this regard, software requirements should be suitable for being imported from, or interfaced to, users, hardware and other software systems. This being so, standard file formats are interesting features which should be offered by RE tools. Since companies rarely work on the same requirements repository and they usually do not work with the same RE tools, a generic format for requirements information is needed. In this context, the Object Management Group (OMG) standard ReqIF [58] is an emerging open, non-proprietary exchange format that is a successful step towards satisfying the urgent industry need for exchanging requirement information between different companies, without losing the advantage of requirements management at the organisations' borders and allowing them to interact and collaborate efficiently.

8.4 The Method

Although GSE has recently attracted great interest, to the best of our knowledge, an RE method that specifically addresses GSE and knowledge reuse is lacking. In response to this problem, a global, reuse-based RE method called PANGEA is presented in the following paragraphs. PANGEA allows the sharing and reuse of knowledge between distributed teams through a shared repository of requirements. The PANGEA repository contains both reusable knowledge from earlier projects and the requirements under development in the current one. A proper requirements management is critical to maintaining awareness in any kind of development. For the sake of applicability, PANGEA encodes knowledge in the form of natural language requirements, which are the most widely used requirements in industry.

Industry experience demonstrates that a process model based on accepted best practices that allows tailoring processes for the specific needs of a project contributes to support GSE [28]. Therefore, to propose an RE method addressing reuse and GSE, we have done the following: firstly, we have studied the state of the art on the threats and their solutions identified in literature regarding RE and GSE, through a systematic literature review [34]. Secondly, a repository of risks and safeguards for the global RE has been compiled. Finally, based on this repository, a global RE method called PANGEA, which encompasses all the proposed safeguards, is put forward in the following pages.

PANGEA extends the SIREN (*SImple Reuse of softwarE requiremeNts*) requirements reuse method [59], a practical way of dealing with requirements reuse. SIREN is a method which is simple enough to be adopted easily by organisations that are currently immature in relation to RE. It can improve productivity and quality of software processes and products, as well as affecting the business positively – indeed, Sommerville and Ransom [60] have conducted an empirical study which revealed that improvements in the RE process led to business improvements. SIREN can be used on its own, as the first RE method adopted by a software development organisation, but in addition, SIREN can be considered as a kind of *add-in* whose goal is to extend, with reusability concerns, any RE method based on natural language requirements. Moreover, Toval et al. [61] state eight key issues that should be taken into account to achieve a practical reuse-based RE method. These are based on the lessons learned from the application of SIREN in the security and data protection fields [59, 62], as well as on other related experiences closer to the domain analysis or audit [63, 64], together with the analysis of related research and current RE tools. However, SIREN was conceived as an RE method for collocated settings right from the beginning; in consequence, it does not address the issues that should be considered when working in a global environment. That makes it necessary to extend and adapt the SIREN method.

The PANGEA method is based on SIREN to a great extent, owing to the success of the latter in practice. Nevertheless, the process model (new activities and new tasks, along with a new set of roles responsible for carrying them out) and part of the techniques (in particular, the requirements reference model, with new

requirements attributes and new attribute values, as well as new traceability relations) have undergone modifications. As well as all this, another part of the techniques, namely, the hierarchy for the requirements documents and the reusability bases (the repository of requirements arranged into catalogues and the requirements reuse guidelines), have been inherited unchanged from SIREN. Thus, the remaining part of this section is devoted to the method resulting from the process explained above, i.e. the method known as PANGEA. Furthermore, having presented PANGEA, Sect. 8.5 goes on to explain the architecture of PANTALASA, the automated support for PANGEA, given that it has evolved along with the process model and the techniques for addressing the GSE issues.

8.4.1 Process Model

The process model proposed for PANGEA combines a set of initial sequential tasks with other cyclical, iterative tasks. This approach therefore includes a spiral model of software development (Fig. 8.1).

In Fig. 8.1, IRD 6, IRD 7, IRD 9 and IRD 10 tasks move from the original model of SIREN, while IRD 1–5 and IRD 8 are brand new. The first five tasks serve as a preparation for making the globally distributed method successful. For this reason, they only have to be performed in the first iteration. The last five tasks make up the effective execution of the RE method, and they are conducted on a cyclical basis for as many iterations as necessary.

Before examining each task, it is necessary to introduce the roles involved in the process model: *coordinator* (coordinates the work of all the project's participants), *moderator* (moderates the requirements negotiation meetings), *team leaders* (represent their work team and speak in their name with the *coordinator* and other *team leaders*), *analysts* (or requirements engineers), *key users* (know the whole system or a part of it and give the necessary knowledge for the production of the requirements documents) and *users* (know part of the system and give the knowledge that is needed for the creation of the requirements documents). Each role has specific responsibilities and a given participation in the tasks and subtasks.

8.4.1.1 IRD 1: Cultural Analysis

This task analyses the different cultures of the participants in the project, using cultural indicators. The role responsible for conducting this study is the *coordinator*, and the reports obtained become part of a catalogue of cultural descriptions, so that they can be reused. The subtasks within this task are:

- IRD 1.1: Nationality identification. The nationalities of all the work teams involved in the project are registered, considering the term *nationality* as the

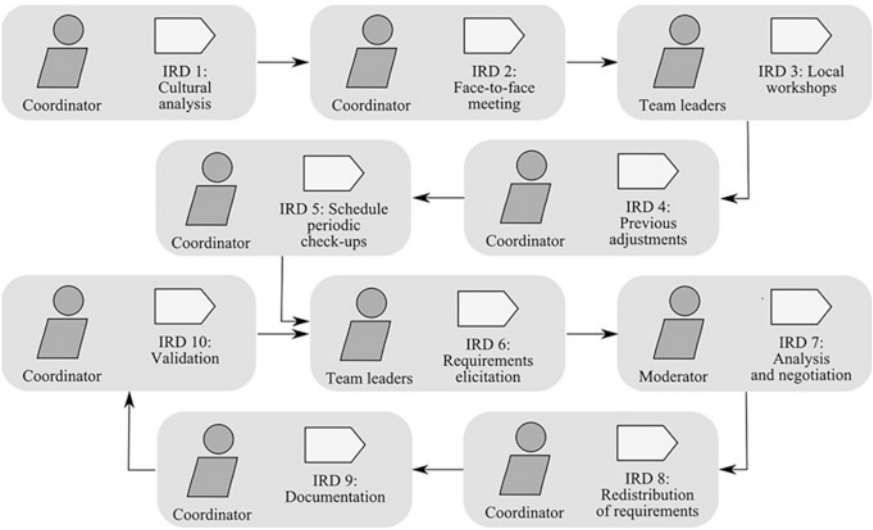

Fig. 8.1 PANGEA process model with SPEM notation

national culture in which people have grown and acquired their values scale. If the work team is heterogeneous, individuals should be analysed.

- IRD 1.2: Report retrieval. The catalogue of cultural descriptions is consulted, and existing reports on the cultures involved are obtained.
- IRD 1.3: Cultural reporting. Preparation of reports on the cultures that are not in the catalogue of cultural descriptions. If they are already in it, review them with the purpose of refining its descriptions and finding errors.
- IRD 1.4: Improvement of the catalogue of cultural descriptions. The reports produced for the first time, along with those already existing and which were improved, are included in the catalogue.

8.4.1.2 IRD 2: Face-to-Face Meeting

It is essential to hold at least one face-to-face meeting at the beginning of the project, not only because it is a richer and more efficient form of communication than any other but because it is needed if a trust relationship is to be established between the participants in a distributed project.

- IRD 2.1: Face-to-face meeting. A meeting for all participants in the project is organised by the *coordinator*. Attendance of at least one representative on behalf of the customer is recommended. All members of all work teams should take part in this meeting; if this is not possible, it will involve at least the *team leaders*. The event should make it possible both to discuss the initial questions about the project and to allow the participants to get to know each other a little, at least.

8.4.1.3 IRD 3: Local Workshops

Cultural understanding is critical to establishing trust relationships.

- IRD 3.1: Local workshops. The *team leaders* organise workshops locally to study the differences between the cultures of the other teams and their own, by means of the indicators established in IRD 1. They are responsible for disseminating the information within their team, the other *team leaders* and the *coordinator*. When the *analysts* and the *team leader* of a team are aware of the major cultural differences, the *team leader* has to report to the *coordinator*. The task ends when the *coordinator* has received confirmation from all the *team leaders*.

8.4.1.4 IRD 4: Previous Adjustments

This task aims to make final adjustments prior to elicitation of requirements.

- IRD 4.1: Key user identification. This problem is complicated in GSE, since the distance and the inability to observe the users performing their regular work may hinder proper identification of people playing the *user* and *key user* roles.
- IRD 4.2: Assignment of key users to work teams. A set of *key users* is assigned to each work team to elicit requirements. They should be collocated whenever possible. Otherwise, the elicitation techniques selected must be compatible with electronic communication media. Only *key users* should collaborate in the elicitation activity, but if this is not possible, then *users* can also be assigned.
- IRD 4.3: Selection of an official language. Although all roles can use the language with which they feel most comfortable in informal social interactions, the *coordinator* must establish an official language for the project. This is to be used in formal requirements negotiation meetings or any situation involving participants from different native languages.
- IRD 4.4: Compilation of a common vocabulary. The terms used in the project-specific domain should be documented, but in GSE the language differences may be substantial. Ontologies are thus useful to alleviate these problems. At this point, a catalogue of ontologies is consulted, and a relevant ontology is retrieved or built from scratch if there is not yet an ontology for the domain.

8.4.1.5 IRD 5: Schedule Periodic Check-Ups

The distance in GSE projects entails many problems, but also enables a follow-the-sun or around-the-clock working model, achieving 24-h global workdays.

- IRD 5.1: Schedule periodic check-ups. The *coordinator* examines the location of the work teams and their time differences, with the intention of scheduling daily meetings. These meetings might be uncomfortable sometimes, but are necessary for tracking the work of the other teams.

8.4.1.6 IRD 6: Requirements Elicitation

Each work team can reuse and extract requirements locally or in a distributed manner. In both cases, different groups of *analysts* extract requirements from a repository of requirements and from different *users*. As a result, a mostly coherent requirements documents hierarchy is obtained.

- IRD 6.1: Requirements reuse. Firstly, each work team selects the catalogues of requirements related to the project from the repository of requirements. Secondly, the team instantiates the parametrised requirements. Finally, the team adds the requirements to the current requirements document. The automated tool support should avoid most of the problems related to the multiple sources of requirements (see Sect. 8.5). Inconsistencies that cannot be confronted automatically should be added to the agenda of the next analysis and negotiation meeting (IRD 7).
- IRD 6.2: Project requirements elicitation. The *analysts* within each work team develop the requirements obtained from the *users* who are working with them and add these to the current requirements document. The *moderator* reviews the requirements included by the different work teams and adds all the inconsistencies found to the agenda of the next analysis and negotiation meeting. In addition, the *analysts* can add any other issue to the agenda that they need to address, by creating a discussion thread.

8.4.1.7 IRD 7: Analysis and Negotiation

Starting from a requirements document with inconsistencies, in which there are issues to discuss, the goal is to obtain a refined version of it in which all the inconsistencies are resolved.

- IRD 7.1: Preparation of the meeting. The agenda, which should be available in the automated tool support, should be read by all the participants involved in the meeting.
- IRD 7.2: Development of the meeting. The discussion is initially carried out by means of a structured, synchronous and textual communication system. The *moderator* takes part in the discussion to impose order, to maintain the progress of the meeting on a virtual blackboard and to use a vote utility if needed. Only the *team leaders* are allowed to participate in representation of their teams. A videoconference system can only be used later on; the reasons for doing it this way are the following: (1) the participants have had enough time to study all the information concerning the agenda, without the pressure of a synchronous communication; (2) the discussion has been conducted so far by means of a textual synchronous tool controlled by the *moderator*, which means that the written interventions have had more chance of being well thought out; and (3) the face-to-face communication allows the reactions of the participants to be evaluated better and helps to solve any issue that still remains open.

- IRD 7.3: Extraction of conclusions. The *moderator* publishes the agreements of the meeting in the automated tool support, including the conclusions reached and the results of any voting carried out. The agreements will be given a unique code to identify them in the project. A discussion thread will be linked to the agreements, for the purpose of storing all the written discussions, the results of the eventual votes and the changes in the requirements documents.

8.4.1.8 IRD 8: Redistribution of Requirements

The requirements documents contain information about the team which has elicited each requirement. In this task, these assignments of requirements to work teams are revised so that requirements can be allocated to different teams.

- IRD 8.1: Assignment proposal. The *coordinator* prepares a proposal for assigning requirements to each work team, based on their most prominent skills, current or expected workload, etc. Such a proposal is made available to all teams through the supporting tool, and a meeting is convened with them to discuss it.
- IRD 8.2: Validation of the assignment. A virtual meeting involving all the *team leaders* and the *coordinator* takes place to discuss the issues related to the assignments. An analysis and negotiation meeting (IRD 7) is performed if needed. Eventually, a validated requirements redistribution is obtained and published in the supporting tool.

8.4.1.9 IRD 9: Documentation

We start from a requirements document that is coherent and without inconsistencies, ideally, and which can be more or less detailed depending on the current iteration of the method; the result is the establishment of such a document as a baseline.

- IRD 9.1: Formalisation of documentation. The *coordinator* creates a baseline from a stable version of the documentation, stores it in a catalogue of releases and exports it to a standard document format by means of the supporting tool.

8.4.1.10 IRD 10: Validation

This task is identical to that carried out in collocated development of software.

- IRD 10.1: Validation of requirements. The *coordinator* provides the document output of IRD 9 to the customer, negotiates all the change requests and forwards the information to the *team leaders*, through a list of changes published in the supporting tool.

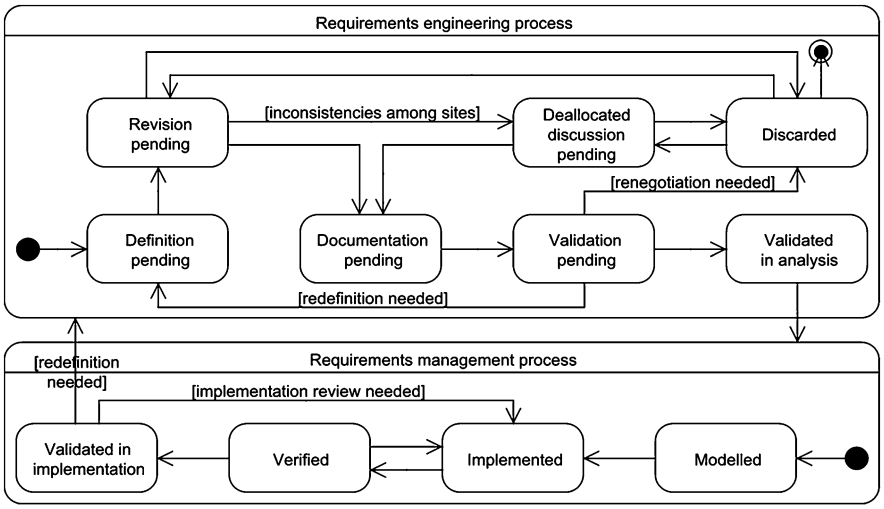

Fig. 8.2 State diagram of a requirement

8.4.2 Requirements Reference Model

There are different types of requirements in PANGEA, but all have the minimum set of associated attributes shown below (the requirements that must be initialised when created are marked as "compulsory"):

- *Text* (compulsory): natural language sentence that specifies the requirement.
- *UniqueIdentificator* (compulsory): identifier of the requirement in the project.
- *Risk*: relative risk of the requirement compared with the rest.
- *Criticality*: relative importance of the requirement for the customer.
- *Priority*: helps to establish an order for the development (set by the analyst).
- *Rationale*: reason why the requirement is included in the project.
- *State*: situation of the requirement (see Fig. 8.2).
- *Source*: origin of the requirement (if it was reused from the repository, then the attribute reflects the catalogue and reusable requirement it comes from).
- *ValidationCriteria*: validation criteria needed to test the requirement (included in the STS document).
- *Responsible*: person responsible for the implementation of the requirement.
- *Section*: document section in which the requirement is specified.
- *VersionLog*: historical record of all versions of the requirement (including author, date, version number and text).
- *RequestedBy* (compulsory): user asking for the inclusion of the requirement (particularly useful when the person who elicited the requirement is not in the team in charge of implementing it).
- *SourceTeam* (compulsory): identifier of the team that elicited or reused the requirement.

- *SourceAnalyst* (compulsory): analyst who drew up or reused the requirement.
- *DiscussionThread*: link to the identifier of the agreements of the meetings in which the requirement was discussed (empty until IRD 7).
- *DestinationTeam*: identifier of the team which has been assigned the requirement for refinement (empty until IRD 8).

PANGEA includes the concept of parametrised requirement. The parameter allows us to specify a variation point in the requirements specification, i.e. when it is necessary to choose between various alternatives in order to configure a specific product. For instance, "the system shall make it possible to export to external files for report generation in [Format]", where the value of the parameter "Format" would be comma-separated values (CSV) or Excel.

The traceability model defined in PANGEA includes a set of traceability relations that enable different links to be established between requirements. Such traces are described below:

- *Parent–child*: relationship describing a more general requirement by a sequence of more specific requirements.
- *Requires*: directional dependency relationship between two requirements. A *Requires* B means that the reuse of A necessarily involves the reuse of B.
- *RelatedTo*: bidirectional dependency relationship between two requirements. A is *RelatedTo* B if (1) B refines or supplements A in some way, so that when A is reused, then B should also be considered for reuse; or (2) A and B belong to the same cluster of requirements.
- *MutuallyExclusive*: mutually exclusive relationship between two requirements. A *MutuallyExclusive* B means that if A is present in the specification, then B cannot be, and vice versa.
- *Reifies*: relationship between a requirement and an artefact of the development process in which it materialises (e.g. class, module, component, etc.).
- *DiscussionThread*: relationship between a requirement and the negotiation meeting in which it was discussed, recorded in the minutes of the meeting and the object *DiscussionThread* that contains the complete record of the discussion, together with the results of any vote taken.

A repository of reusable requirements arranged into catalogues for managing requirements knowledge is included in PANGEA. These catalogues can be (1) *domains*, "vertical" application domains (e.g. insurance or banking), or (2) *profiles*, "horizontal" application domains (e.g. security or personal data protection). Moreover, they are organised in a hierarchy of requirements documents, which, in turn, are structured according to standards (IEEE Std. 1233, IEEE Std. 12207.1, and IEEE Std. 830).

With regard to the reuse of requirements, once the scope of the project has been established, the requirements repository should be searched to find a domain catalogue within that area. If so, the project probably corresponds with the development of a particular product in the domain specified in the catalogue. In this case, the searches in the repository might begin with the requirements with the highest

value in the attribute *Criticality* in the domain catalogue. Since these requirements are mandatory, they are part of any product for that domain, and as such all of them should be reused. Their traces should be analysed to determine other non-mandatory requirements which ought also to be part of the specification of the current project. After the common specification of the product is established, new searches should be defined, guided by the business requirements and the features or system objectives identified. These guidelines may involve searching the same domain catalogue or different profile catalogues. If, as a result of any of the searches, a requirement is reused that has trace relations of type *RelatedTo* other requirements, then a cluster of requirements is found. In this case, we should consider the selection of the requirements within that cluster. In summary, reusing the requirements selected involves the resolution of the variation points found in them: (1) instantiation of the parameters of the parametrised requirements with values appropriate to the current project; (2) resolution of *MutuallyExclusive* traces; (3) resolution of *RelatedTo* traces, which are optional; and (4) resolution of *Requires* and *Parent–child* traces, which should normally be included.

8.5 The Tool Architecture

As is shown in Sect. 8.3, literature reflects the need for RE tools that support GSE. In this respect, the PANGEA method is supported by PANTALASA (*PANgea Tool And Lightweight Automated Support Architecture*), which is the underlying tool architecture for the global RE processes and models. Current RE tools' capabilities [65] and the ISO TR 24766 [66] have been taken into account in its conception.

PANTALASA is responsible for managing the requirements knowledge in a coherent way, giving a boost to reuse, automatising some repetitive tasks and, in short, facilitating the distributed stakeholders' activities carried out within the framework of PANGEA. Among its features, PANTALASA allows multiple users to edit the same requirements document simultaneously, so when a user reuses a requirement from a catalogue of the requirements repository, the tool automatically has to keep track of all the existing relationships, i.e. the tool will check if the parent requirement has to be included, if other requirements are involved or if the reused requirement or its dependencies violate any exclusive relationship with any requirement previously added.

By means of these automated verification mechanisms, it is ensured that the requirements are consistent at all times and that there is a single shared working document for all stakeholders, promoting proper control and coordination of the team members' work. Moreover, if different analysts introduce the same parametrised requirement and assign different values to the same parameter, when the second analyst attempts to insert the troublesome requirement in the requirements document, the tool will detect the inconsistency and will notify the analysts involved about such a situation. These analysts can then discuss what

the correct value of the parameter is, and, if necessary, they can take the matter to the agenda of the next requirements analysis and negotiation meeting (IRD 7).

With regard to specific technologies that might be particularly useful for PANTALASA, we considered the use of semantic wikis and social networks, because we believe that they turn out to be complementary. Semantic wikis are organised around an ontology of requirements, so that the requirements repository is structured more consistently than in a *plain* wiki, whereas social networks leverage the communication strategies between project members. In this regard, wikis were originally conceived for distributed collaborative content creation [67], but it is possible to use them to capture requirements and domain knowledge [67, 68] or even to support AKM in GSE [50], improving domain knowledge reuse and tacit knowledge acquisition [68]. Furthermore, as a result of their underlying information models, semantic wikis can provide support to reasoning with the requirements knowledge [50, 69]. This is especially relevant for traceability approaches and also enables automated information retrieval [50]. On the other hand, Whitehead et al. [70] raised the possible application of new trends in networking and social networks to improve formal and informal communication. Following this trend, Lim et al. [71] at first proposed a social network system for stakeholder analysis and later, an extension of this tool was developed to identify and prioritise software requirements [72].

8.6 Prototype Implementation and Validation

We have recently developed an automated tool support proposal for PANGEA by means of the integration of (1) a well-known social network like Facebook,[1] which integrates both synchronous and asynchronous communication, and that serves to establish and strengthen trust relationships between distributed software development teams, and (2) a semantic wiki like Semantic MediaWiki (SMW),[2] which is suitable for supporting the PANGEA requirements reference model and the reusable-requirements repository, taking into account issues such as security, concurrency and discussion threads. Some of the KM challenges previously identified in Sect. 8.3, such as communication, knowledge capture and knowledge transfer issues, are therefore addressed by relying on such collaborative and open source technologies.

A block diagram of the prototype is shown in Fig. 8.3. Users connect with the application in an automatic and transparent manner through Facebook. The application serves as a connection bridge to SMW, where the requirements repository is located. Figure 8.4 shows a SMW page in the Facebook interface for enabling distributed stakeholders' work.

[1] http://www.facebook.com.

[2] http://semantic-mediawiki.org.

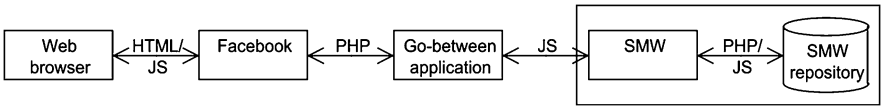

Fig. 8.3 Free-form architecture diagram of the prototype

Fig. 8.4 Graphical user interface (GUI) of the prototype

Facebook offers an application programming interface (API) to support the execution of external applications. In addition, these applications have authentication mechanisms at their disposal, so it is possible to access personal data such as name, email, friends list or even wall comments. If the user gives permission for the external application, Facebook is able to send to our application the user information needed. The aim is to provide a personalised experience, but a user authorisation process must be carried out to make that possible.

This implementation of PANTALASA includes an external, go-between application, whose main task is to communicate Facebook and SMW (see Fig. 8.3). Hence, this piece of software allows us to perform the following duties in a transparent manner, automatically: (1) gather the data of a Facebook user, (2) register Facebook users in the SMW database, (3) authenticate users in SMW, (4) configure SMW in order to adapt it to the use requested by PANGEA and (5) execute SMW under the Facebook applications interface.

It is important to note that the validation of the prototype was conducted in an academic environment by college students who worked with the application in a distributed and collaborative way, encoding requirements catalogues in the prototype, reusing the requirements in a new project and taking advantage of the Facebook capabilities for supporting communication between them. More detailed information about the validation is provided below.

Two students participated in the study with the intention of validating the prototype; this lasted for 2 weeks. They introduced three requirements catalogues in the requirements repository. The catalogues were made up of both functional and non-functional requirements. Two of these catalogues were *profiles*, as they codified security requirements and personal data protection requirements based on the respective original catalogues developed for SIREN [59, 62]. The other catalogue, on the other hand, was a *domain*, since it included domain-specific requirements extracted from a case study on the field of hotel management, which was presented in a course on RE belonging to a degree programme on computer science and engineering. The full case study was composed of approximately 150 requirements, and about 50 of these were inserted in the system.

Note that the mentioned catalogues are located in the SMW requirements repository. All the requirements that make up these three catalogues are thus available for reuse in new projects that can be created with the tool, following the proposed method. In fact, the students created a new project in which specific requirements were extracted from the catalogues listed above in order to check the reuse functionality of the tool. Such a project was aimed at the development of a software application for a particular hotel. In this project, the different sections of the SRS were generated, including the functional requirements of the project, which were mostly reused from the functional requirements of the hotel industry catalogue.

The feedback after using the system was mainly satisfactory. However, some difficulties in reusing the requirements from the catalogues in the new project were reported; these problems come about from technical concerns related to SMW, since the tool is still in an incipient stage. In addition, the students worked in a distributed manner, but not in a global environment, since they were located in different cities about 100 km apart. Moreover, they have reported neither communication nor concurrency problems.

From our point of view, the main limitation of the validation of the prototype is the fact that it was conducted by students. Nevertheless, students play a very important role in software engineering experimentation, because before performing studies in industrial environments, which requires a lot of resources and time, it is generally useful for researchers to carry out pilot studies with students in academic environments [73, 74]. In addition, students are the next generation of professionals [75], and under some conditions, there is not a great difference between students and professionals. In situations in which the tasks which are to be performed do not require industrial experience, experimentation with students is viable [76, 77]. Another relevant threat to the validity of the study is that the students were not globally distributed. Nonetheless, this was to some extent mitigated by the fact that they were not collocated, and even relatively small geographic distances between offices profoundly affect the ability to collaborate [10]. Indeed, Allen [78] found that there is a strong negative correlation between physical distance and frequency of communication between sites and any distance greater than the critical threshold of about 50 m led to a dramatic drop in spontaneous communication and collaboration between individuals.

8.7 Summary

Since one of the current problems in GSE is the existence of knowledge which is not properly shared and reused, thus hampering awareness, KM in global, distributed settings is a challenging task. It can be faced by improving reuse, sharing and collaboration in global RE. To the best of our knowledge, there is no proposal which tackles both GSE and requirements reuse. In this chapter, we have presented PANGEA, a reuse-based RE method for GSE that specifies knowledge in the form of natural language requirements. PANGEA encompasses a process model, a requirements reference model and PANTALASA, the supporting tool architecture. PANTALASA has been developed by means of (1) a semantic wiki, in an effort to implement a reusable-requirements repository, and (2) a social network, to improve communication issues. Before describing PANGEA and PANTALASA, this chapter provides brief insights into the relationship between KM and (global) RE, as well as the practical challenges concerning RE and KM in GSE.

Ebert and De Man [22] state that a software company or department is confronted with many challenges that must be mastered through continuous improvement, along the following axes: (1) *consolidating*, focusing on a few essential products and maximising their business value; (2) *industrialising*, mastering projects, processes and knowledge by intelligent collaboration to improve predictability, repeatability and affordability; and (3) *globalising*, depending on the needs of the target market and the size of the company, but its success relies on the other two axes. An approach has been presented in this chapter that leverages this continuous improvement strategy by means of proper management of requirements knowledge. Firstly, an organisation that usually develops products in a domain eventually has enough expertise to generate high-quality requirements catalogues dealing with common issues in that domain (consolidating). Secondly, once such an organisation manages domain knowledge appropriately by means of requirements catalogues, the entire software development process benefits and is greatly improved in terms of cost, time and effort (industrialising). Finally, the particularities of globalisation are taken into account and its demands materialised in the RE process in order to be successful in a global environment (globalising).

Future work includes research on data mining techniques that can be applied to requirements, the aim being to build prediction models and help developers make better decisions on the subsequent stages of the software development process. We are also interested in supporting project management issues by relying on RE, so that project management and decision making processes within the organisation could take advantage of explicit or derived requirements knowledge. Finally, an academic case study between the University of Murcia (Murcia, Spain), the University of Castilla-La Mancha (Ciudad Real, Spain) and the University Mohammed V – Souissi (Rabat, Morocco) is planned, with the intention of validating our proposal in a nearshore environment. We are also planning to conduct a case study in a real industry environment later on, in a subsequent stage of the validation.

Acknowledgments This work has been funded by the PEGASO/PANGEA project (TIN2009-13718-C02-02), the ORIGIN Integrated Project (IDI-2010043 (1–5)) and the ENGLOBAS Project (PII2I09-0147-8235).

References

1. Brooks FP Jr (1987) No silver bullet: essence and accidents of software engineering. IEEE Comp 20:10–19
2. Meyer B (1997) Object-oriented software construction, 2nd edn. Prentice-Hall, New York
3. Mili H, Mili F, Mili A (1995) Reusing software: issues and research directions. IEEE Trans Softw Eng 21:528–562
4. Cybulski JL, Reed K (2000) Requirements classification and reuse: crossing domain boundaries. In: Proceedings of the 6th international conference on software reuse: advances in software reusability, Vienna, pp 190–210
5. Sommerville I (2004) Software engineering, 7th edn. Pearson Addison Wesley, Boston
6. Favaro J (2002) Managing requirements for business value. IEEE Softw 19:15–17
7. Robertson S, Robertson J (2006) Mastering the requirements process, 2nd edn. Addison-Wesley, Upper Saddle River
8. Cheng BHC, Atlee JM (2007) Research directions in requirements engineering. In: Future of software engineering, IEEE Computer Society, Minneapolis, USA, pp 285–303
9. Rine DC, Nada N (2000) An empirical study of a software reuse reference model. Inf Softw Technol 42(1):47–65
10. Herbsleb JD (2007) Global software engineering: the future of socio-technical coordination. In: Future of software engineering, IEEE Computer Society, Minneapolis, USA, pp 188–198
11. Damian D, Moitra D (2006) Global software development: how far have we come? IEEE Softw 23:17–19
12. Noll J, Beecham S, Richardson I (2010) Global software development and collaboration: barriers and solutions. ACM Inroads 1(3):66–78
13. Damian D (2007) Stakeholders in global requirements engineering: lessons learned from practice. IEEE Softw 24:21–27
14. Sinha V, Sengupta B, Chandra S (2006) Enabling collaboration in distributed requirements management. IEEE Softw 23:52–61
15. Gallardo-Valencia RE, Sim SE (2009) Continuous and collaborative validation: a field study of requirements knowledge in agile. In: Proceedings of the 2nd international workshop on managing requirements knowledge, IEEE Computer Society, Atlanta, USA, pp 65–74
16. Manteli C, van den Hooff B, Tang A, van Vliet H (2011) The impact of multi-site software governance on knowledge management. In: Proceedings of the 6th IEEE international conference on global software engineering, IEEE Computer Society, Helsinki, Finland, pp 40–49
17. Aurum A, Jeffery R, Wohlin C, Handzic M (eds) (2003) Managing software engineering knowledge. Springer, Berlin
18. Rus I, Lindvall M (2002) Knowledge management in software engineering. IEEE Softw 19:26–38
19. Ebling T, Nicolas Audy JL, Prikladnicki R (2009) A systematic literature review of requirements engineering in distributed software development environments. In: Proceedings of the 11th international conference on enterprise information systems, Milan, Italy, pp 363–366
20. Berenbach B (2006) Impact of organizational structure on distributed requirements engineering processes: lessons learned. In: Proceedings of the international workshop on global software development for the practitioner, ACM, Shanghai, China, pp 15–19

21. Al-Ani B (2010) Questions regarding knowledge engineering and management. In: Proceedings of the 5th IEEE International conference on global software engineering, IEEE Computer Society, Princeton, USA, pp 324–329
22. Ebert C, De Man J (2008) Effectively utilizing project, product and process knowledge. Inf Softw Technol 50(6):579–594
23. Ågerfalk PJ, Fitzgerald B, Holmström H, Lings B, Lundell B, Conchúir EO (2005) A framework for considering opportunities and threats in distributed software development. In: Proceedings of the international workshop on distributed software development, Austrian Computer Society, Paris, France, pp 47–61
24. Portillo Rodríguez J, Ebert C, Vizcaíno A (2010) Technologies and tools for distributed teams. IEEE Softw 27:10–14
25. Portillo Rodríguez J, Vizcaíno A, Ebert C, Piattini M (2010) Tools to support global software development processes: a survey. In: Proceedings of the 5th IEEE international conference on global software engineering, IEEE Computer Society, Princeton, USA, pp 13–22
26. Edwards JS (2003) Managing software engineers and their knowledge. In: Aurum A, Jeffery R, Wohlin C, Handzic M (eds) Managing software engineering knowledge. Springer, Berlin, pp 5–27
27. Lethbridge TC, Singer J, Forward A (2003) How software engineers use documentation: the state of the practice. IEEE Softw 20:35–39
28. Ebert C, De Neve P (2001) Surviving global software development. IEEE Softw 18(2):62–69
29. Mika P (2007) Social networks and the semantic web, Semantic web and beyond. Springer, New York
30. Beecham S, Noll J, Richardson I, Ali N (2010) Crafting a global teaming model for architectural knowledge. In: Proceedings of the 5th IEEE international conference on global software engineering, IEEE Computer Society, Princeton, USA, pp 55–63
31. Ali N, Beecham S, Mistrík I (2010) Architectural knowledge management in global software development: a review. In: Proceedings of the 5th IEEE international conference on global software engineering, IEEE Computer Society, Princeton, USA, pp 347–352
32. Hall JG, Jackson M, Laney RC, Nuseibeh B, Rapanotti L (2002) Relating software requirements and architectures using problem frames. In: Proceedings of the 10th anniversary IEEE joint international conference on requirement engineering, IEEE Computer Society, Essen, Germany, pp 137–144
33. Clerc V (2008) Towards architectural knowledge management practices for global software development. In: Proceedings of the 3rd international workshop on sharing and reusing architectural knowledge, ACM, Leipzig, Germany, pp 23–28
34. López A, Nicolás J, Toval A (2009) Risks and safeguards for the requirements engineering process in global software development. In: Proceedings of the 4th IEEE international conference on global software engineering, IEEE Computer Society, Limerick, Ireland, pp 394–399
35. Desouza KC, Awazu Y, Baloh P (2006) Managing knowledge in global software development efforts: issues and practices. IEEE Softw 23:30–37
36. Kucza T, Nättinen M, Parviainen P (2001) Improving knowledge management in software reuse process. In: Proceedings of the 3rd international conference on product focused software process improved, Kalserslautern, pp 141–152
37. Schneider K, von Hunnius JP, Basili V (2002) Experience in implementing a learning software organization. IEEE Softw 19:46–49
38. Seaman CB, Mendonça MG, Basili VR, Kim YM (2003) User interface evaluation and empirically-based evolution of a prototype experience management tool. IEEE Trans Softw Eng 29:838–850
39. Maalej W, Thurimella AK, Happel HJ, Decker B (2008) Managing requirements knowledge (MaRK'08). In: Proceedings of the 1st international workshop on managing requirement knowledge, IEEE Computer Society, Barcelona, Spain, pp i–ii
40. Ma L, Nuseibeh B, Piwek P, Roeck AD, Willis A (2009) On presuppositions in requirements. In: Proceedings of the 2nd international workshop on managing requirement knowledge, IEEE Computer Society, Atlanta, USA, pp 27–31

41. Damian D, Zowghi D (2002) The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization. In: Proceedings of the 10th anniversary IEEE joint international conference on requirements engineering, IEEE Computer Society, Essen, Germany, pp 319–330
42. Kwan I, Damian D, Marczak S (2007) The effects of distance, experience, and communication structure on requirements awareness in two distributed industrial software projects. In: Proceedings of the 1st international global requirements engineering workshop, Munich, Germany, pp 29–35
43. Bhat JM, Gupta M, Murthy SN (2006) Overcoming requirements engineering challenges: lessons from offshore outsourcing. IEEE Softw 23:38–44
44. Hsieh Y (2006) Culture and shared understanding in distributed requirements engineering. In: Proceedings of the IEEE international conference on global software engineering, IEEE Computer Society, Florianopolis, Brazil, pp 101–108
45. Betz S, Oberweis A, Stephan R (2010) Knowledge transfer in IT offshore outsourcing projects: an analysis of the current state and best practices. In: Proceedings of the 5th IEEE international conference on global software engineering, IEEE Computer Society, Princeton, USA, pp 330–335
46. Herbsleb JD, Mockus A (2003) An empirical study of speed and communication in globally distributed software development. IEEE Trans Softw Eng 29:481–494
47. Szulanski G, Winter S, Grant R, Spender JC, Kogut B, Miner A, Ghoshal S (2000) The process of knowledge transfer: a diachronic analysis of stickiness. Organ Behav Hum Dec Proc 82:9–27
48. Carmel E, Agarwal R (2001) Tactical approaches for alleviating distance in global software development. IEEE Softw 18:22–29
49. Correia FF, Aguiar A (2009) Software knowledge capture and acquisition: tool support for agile settings. In: Proceedings of the 4th international conference on software engineering advanced, IEEE Computer Society, Limerick, Ireland, pp 542–547
50. Clerc V, de Vries E, Lago P (2010) Using wikis to support architectural knowledge management in global software development. In: Proceedings of the ICSE workshop on sharing and reusing architectural knowledge, ACM, Cape Town, South Africa, pp 37–43
51. Oshri I, van Fenema PC, Kotlarsky J (2008) Knowledge transfer in globally distributed teams: the role of transactive memory. Inf Syst J 18(6):593–616
52. Lee SB, Shiva SG (2010) An approach to overcoming knowledge sharing challenges in a corporate IT environment. In: Proceedings of the 5th IEEE international conference on global software engineering, IEEE Computer Society, Princeton, USA, pp 342–346
53. Kotlarsky J, Oshri I (2005) Social ties, knowledge sharing and successful collaboration in globally distributed system development projects. Eur J Inf Syst 14:37–48
54. Ebert C (2012) Global software and IT: a guide to distributed development, projects, and outsourcing. Wiley, Hoboken
55. Heindl M, Reinisch F, Biffl S (2007) Requirements management infrastructures in global software development – Towards application lifecycle management with role-based in-time notification. In: Proceedings of the international conference on global software engineering (ICGSE), workshop on tool-supported requirements management in distributed projects (REMIDI), Munich, Germany
56. Mistrík I, Grundy J, Van der Hoek A, Whitehead J (2010) Collaborative software engineering: challenges and prospects. In: Mistrík I, Grundy J, Van der Hoek A, Whitehead J (eds) Collaborative software engineering. Springer, Berlin/Heidelberg, pp 389–403
57. Laurent P (2010) Globally distributed requirements engineering. In: Proceedings of the 5th IEEE international conference on global software engineering, IEEE Computer Society, Princeton, USA, pp 361–362
58. Monteiro MR, Ebert C, Recknagel M (2009) Improving the exchange of requirements and specifications between business partners. In: Proceedings of the 17th IEEE international requirements engineering conference, IEEE Computer Society, Atlanta, USA, pp 253–260

59. Toval A, Nicolás J, Moros B, García F (2002) Requirements reuse for improving information systems security: a practitioner's approach. Requir Eng 6:205–219
60. Sommerville I, Ransom J (2005) An empirical study of industrial requirements engineering process assessment and improvement. ACM Trans Softw Eng Methodol 14:85–117
61. Toval A, Moros B, Nicolás J, Lasheras J (2008) Eight key issues for an effective reuse-based requirements process. Comp Syst Sci Eng 23:1–13
62. Toval A, Olmos A, Piattini M (2002) Legal requirements reuse: a critical success factor for requirements quality and personal data protection. In: Proceedings of the 10th anniversary IEEE joint international conference on requirement engineering, IEEE Computer Society, Essen, Germany, pp 95–103
63. Martínez MA, Lasheras J, Fernández-Medina E, Toval A, Piattini M (2010) A personal data audit method through requirements engineering. Comp Stand Interf 32:166–178
64. Nicolás J, Lasheras J, Toval A, Ortiz FJ, Álvarez B (2009) An integrated domain analysis approach for teleoperated systems. Requir Eng 14:27–46
65. Carrillo de Gea JM, Nicolás J, Fernández Alemán JL, Toval A, Ebert C, Vizcaíno A (2011) Requirements engineering tools. IEEE Softw 28(4):86–91
66. ISO/IEC JTC 1 SC 7: ISO/IEC TR 24766 (2009) Information technology – systems and software engineering – guide for requirements engineering tool capabilities, 1st edn. ISO, Geneva
67. Uenalan O, Riegel N, Weber S, Doerr J (2009) Using enhanced wiki-based solutions for managing requirements. In: Proceedings of the 2nd international workshop on managing requirement knowledge, IEEE Computer Society, Atlanta, USA, pp 63–67
68. Ugai T, Aoyama K (2009) Domain knowledge wiki for eliciting requirements. In: Proceedings of the 2nd international workshop on managing requirement knowledge, IEEE Computer Society, Atlanta, USA, pp 4–6
69. Liang P, Avgeriou P, Clerc V (2009) Requirements reasoning for distributed requirements analysis using semantic wiki. In: Proceedings of the 4th IEEE international conference on global software engineering, IEEE Computer Society, Limerick, Ireland, pp 388–393
70. Whitehead J, Mistrík I, Grundy J, Van der Hoek A (2010) Collaborative software engineering: concepts and techniques. In: Mistrík I, Grundy J, Van der Hoek A, Whitehead J (eds) Collaborative software engineering. Springer, Berlin/Heidelberg, pp 1–30
71. Lim SL, Quercia D, Finkelstein A (2010) StakeSource: harnessing the power of crowdsourcing and social networks in stakeholder analysis. In: Proceedings of the 32nd ACM/IEEE international conference on software engineering, ACM, Cape Town, South Africa, pp 239–242
72. Lim SL, Damian D, Finkelstein A (2011) StakeSource2.0: using social networks of stakeholders to identify and prioritise requirements. In: Proceedings of the 33rd international conference on software engineering, Waikiki, pp 1022–1024
73. Carver J, Jaccheri L, Morasca S, Shull F (2003) Issues in using students in empirical studies in software engineering education. In: Proceedings of the 9th IEEE international symposium on software metrics, IEEE Computer Society, Sydney, Australia, pp 239–249
74. Carver J, Jaccheri L, Morasca S, Shull F (2003) Using empirical studies during software courses. In: Conradi R, Wang A (eds) Empirical methods and studies in software engineering, Lecturer notes in computer science, vol 2765, Springer, Berlin/Heidelberg, pp 81–103
75. Kitchenham BA, Pfleeger SL, Pickard LM, Jones PW, Hoaglin DC, Emam KE, Rosenberg J (2002) Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng 28:721–734
76. Basili VR, Shull F, Lanubile F (1999) Building knowledge through families of experiments. IEEE Trans Softw Eng 25:456–473
77. Svahnberg M, Aurum A, Wohlin C (2008) Using students as subjects – An empirical evaluation. In: Proceedings of the 2nd ACM-IEEE international symposium on empirical software engineering & measurement, ACM, Kaiserslautern, Germany, pp 288–290
78. Allen T (1977) Managing the flow of technology. MIT Press, Cambridge, MA