Agile and Traditional Requirements Engineering: A Survey

Heba Elshandidy, Sherif Mazen

Abstract— In the past few years, there has been an increasing awareness of the important role of requirements engineering (RE) in a project's success, in both research and industry. Developing consistent requirements specifications that meet the customers' needs, in traditional development, is likely to be infeasible. For one reason, customers do not usually have a clear picture of what they really want. Secondly, the business domain could be changing quickly, especially if it heavily depends on technologies. Agile software development (ASD), on the other hand, supports iterative and incremental development and emphasises customers' involvement in the development process. We argue that adopting ASD in RE overcomes the limitations of the traditional development. ASD, however, is no silver bullet and its adoption comes at a price. This paper helps the reader to: (1) get a quick yet a comprehensive grasp of RE in traditional and ASD; (2) understand the synergies/commonalities between the two approaches in handling RE; (3) recognise the associated challenges of adopting ASD; and (4) identify the current prominent agile RE research areas.

Index Terms— agile RE challenges, agile RE practices, agile RE research areas, agile requirements engineering, agile software development, requirements engineering, traditional RE

1 Introduction

Requirements engineering (RE) is the process of determining what the system does and not how it does it. The increasingly changing business environments have challenged the principles of traditional RE, in a sense that, in the past few years, the awareness of organisations to adopt changes instead of resenting them has significantly increased, as suggested by many experienced practitioners. Since requirements often evolve, then the only way to guarantee the success of a given project is to embrace these changes.

Embracing changes throughout the different stages of development is likely to be infeasible in the traditional RE aproach, as the RE process takes place only once prior to the start of development; thus, any changes that are presented afterwards will be costly, if they are doable at all. This is one of the main reasons why agile software development (ASD) has received such a great attention from many practitioners in the RE field. Unlike the predictive and process-oriented traditional development, ASD methods are adaptive and people-oriented [1]. According to the studies in [2,3], more than two thirds of the factors that contribute to a project's success fall under the umbrella of RE and ASD. Hence, surveying the potential of bringing them together will undoubtedly benefit both research and industrial communities.

This paper attempts to answer the following research questions: What is the current research state-of-the-art in traditional RE (Section 2)? What are the RE practices adopted in ASD (Section4)? What are the commonalities and differences between agile and traditional RE approaches (Section 4)? What are the challenges associated with the agile RE practices adoption (Section 5)? And lastly, what are the agile RE areas that are still in need for research (Section 6)? The paper also gives an overview about ASD in Section 3 and it finally concludes the presented work in Section 7.

2 Requirements Engineering

Successful RE requires a very good understanding of the business domain, the environment in which the system will be running, and the needs of the project's stakeholders (customers, users, developers, etc.) [4]. In traditional development, the following is assumed: (1) the customer precisely knows, from the beginning, what they need from the system; (2) the development team understands the customer's needs correctly and clearly; (3) only one or more stakeholders are in charge of elaborating the requirements; and (4) there is a strict separation of different functions with little focus on cross-functional teams [5]. Somerville and Sawyer claim that the RE process is composed of five main tasks: elicitation, analysis and negotiation, modelling, validation and verification, and management [4]. The rest of this section gives further details about these tasks including the most popular work offered in each activity.

2.1 Requirements Elicitation

Requirements elicitation is the process of discovering and elaborating the requirements of a system. These requirements could be very well understood, fuzzily understood, or innovative problems. Generally, this task requires a very good knowledge not only of the application domain, but also of the specific problem the system to-be is trying to solve. Previous work in requirements elicitation focused mainly on solutions that improve the precision, the accuracy, and the different

- *Heba Elshandidy is currently a researcher in Information Systems Dept., Faculty of Computers and Information, Cairo University, Egypt. E-mail: helshandidy@gmail.com*
- *Sherif Mazen is currently an associate professor in Information Systems Dept., Faculty of Computers and Information, Cairo University, Egypt. E-mail: s.mazen@fci-cu.edu.eg*

levels of details for requirements [6]. To this end, the related work will be summarised subsequently.

- Notations are developed to help better understand the requirements and expedite the process of exploring and learning the stakeholders' needs, while facilitating discussions through using models that are unlikely to be misinterpreted by the different stakeholders. The following are examples for such notations: scenarios [7], agent-based [8],[9],[10], sequence charts [11], use cases [12], anti-models [13],[14],[15], non-functional requirements [16], goal models [17,18], and policy [19]. Such models tend to be inexpensive and they significantly ease the process of collecting feedback from the stakeholders as early as possible.
- Many methodologies have been proposed for the elicitation of requirements, in which they are used for: (1) dentifying stakeholders [20]; (2) ensuring that the requirements are not diverted away from the scope of the project, such as personas [21,22] and metaphors [23]; (3) considering the context and the environment, in which the system will be deployed, as well as the personal requirements when extracting the requirements of the system [24,25]; and (4) identifying the luxurious requirements to help making the final product more attractive and likeable, such as brainstorming and creativity workshops [26]
- Many tools have been offered to elicit feedback on the early representation of a system, such as model-driven development [27], simulations [28], storyboards [29], and model animations [30,31]

2.2 Requirements Analysis and Negotiation

The main purpose of this task is to identify and resolve any conflicts, overlaps, omissions, and inconsistencies in the elaborated requirements. The output list of this task is then negotiated and prioritised, in a sense that an agreement, among the system's stakeholders, is reached about what requirements to be implemented and in what order [4]. Most of the research done in this area focuses on the development of new, or the improvement of existing, methodologies and techniques for efficiently and precisely evaluating the quality of the produced requirements [6].

- Methodologies[1] in that area proposed solutions and guidelines for the problems of: negotiation [32], requirements alignment with commercial-off-the-shelf (COTS) products and open systems [33,34], and the management of conflicts of the unknown interactions between requirements [35].
- There are plenty of analyses conducted to help in locating linguistic issues, which could be a barrier for having understandable and conflict-free requirements. The work in [36],[37],[38],[39],[40] investigated cases where errors could be ambiguous, while the work in [41],[42],[43],[44], [45],[46] investigated cases where errors could be inconsistent or incomplete. Further work includes analyses for locating: (1) possible obstacles to the satisfaction of requirements, goals, and elaborated assumptions [47,48]; (2) missing assumptions [49]; or (3) unknown interactions among requirements [35],[50],[51].
- Techniques[2] in [52],[53],[54] address risk and impact analysis, so that the team can accurately understand the requirements, their inter-associations, and the consequences of that interdependencies. Techniques for prioritisation, visualisation, and analyses have also been presented; thus enabling the stakeholders to adequately select the most favourable set of requirements to be implemented [55],[56],[57],[58],[59], or identify well-fitted solutions for COTS based projects [33,60].

2.3 Requirements Modelling

Modelling represents the step that bridges the gap between analysis and design through providing formal visual representations for the system. A model is considered a good one if it is unambiguous, complete, precise, and can be easily communicated with all of the involved stakeholders [61]. Having a good modelling process means: (1) there is a consistent and precise defined vocabulary across the system; (2) diagrams are used to adequately visualise the system specification (i.e. each model records specific details about the requirements); (3) the different points of view of the system are properly considered; and (4) the elaborated requirements are validated through animation [61]. For further details, the main modelling approaches/methods are summarised next.

- The main traditional modelling approaches are data flow diagrams (DFDs), entity relationship diagrams (ERDs), and statecharts. While DFDs show how data is processed at different levels of abstractions, ERDs show the conceptual representation of the requirements [62]. Statecharts, on the other hand, are used to describe the behaviour of a system [63].
- The main object oriented approaches are class diagrams and use cases. While class diagrams represent a system as a set of objects encapsulating the details of their behaviour and characteristics, the use cases define the interactions between the users of a system and the system itself.
- The literature also spoke of three main types of methods that are used for modeling requirements, namely: viewpoint, object oriented, and formal methods. The viewpoint methods (e.g. CORE [64], SADT [65], and VORD [66]) believe that requirements should be elaborated from the perspectives of all the stakeholders. The object-oriented methods (e.g. OOA [67], OMT [68], and the Booch Method) were initially adopted by companies believing that time-to-market and resistance to change were paramount. Formal Methods (e.g. Z, VDM, LOTOS, and the B-Method), on the other hand, believe in

[1] A methodology is usually a guideline system for solving a problem, with specific components such as phases, tasks, methods, techniques and tools

[2] A technique describes how to perform a particular technical activity, and, if appropriate, how to use a particular notation as part of that activity.

a more rigorous representation based on mathematics [69].

- Notations such as the Unified Modelling Language (UML) are of a critical importance in modelling, because of the number of models they offer, and in which each model represents a different phase of development and collectively they describe the system under development [70].

2.4 Requirements Validation and Verification

Requirements validation is the process of ensuring that the development team is building the right product. This is achieved through reviewing the requirements artefacts with the stakeholders, to guarantee that the models and the documentation reflect their needs accurately [71]. The main research done in this area was based on simulations [72], animations [30],[31],[73], and derived invariants [74].

Requirements verification, on the other hand, is the process of ensuring that the development team is building the product right. Formal requirements can be verified through model checking, model satisfiability, and notations. While model checking, an algorithmic analysis of programmes, automatically tests whether a system model meets a given formal specification [75], model satisfiability checks that there exist valid instantiations of constrained object models, and that operation on object models preserve invariants [76]. Notations, on the other hand, facilitate automated verification, where these verification models are the simplifications and the abstractions of the specification to be verified [77,78].

To sum up, the typical requirements validation and verification approaches used in traditional development include: tracing approaches, prototyping, testing, user manual writing, formal validation, and reviews and inspection (e.g. walkthroughs, formal inspections, and checklists) [4].

2.5 Requirements Management

Requirements management is primarily concerned with all the activities associated with changing the requirements of the system from change/version control to requirements tracing and status tracking. There are plenty of tools and techniques to simplify, and partially automate, the task of identifying and documenting traceability links among requirements [79],[80],[81],[82],[83]. Also, there exist analyses that determine the stability of requirements to adequately manage future changes, while making and evaluating the architectural choices accordingly [84].

3 Agile Software Development

ASD is a group of software development methods based on incremental and iterative development. According to the agile manifesto [1], ASD is defined in terms of values, principles, and practices. Unlike traditional development methods, agile development does not encourage upfront detailed planning for the entire project, and promotes both quick clean delivery and stakeholders' involvement in the entire process of development [85]. Table 1. summarises the main differences between traditional and agile development [86].

TABLE 1
Traditional Versus Agile Development

	Traditional	Agile
Planning	Predictive	Adaptive
Focus	Process-oriented	People-oriented
Documentation	Comprehensive and constant	Minimal and evolving
Development team members' roles	Strict separation of roles through specialisation	Self-organising and cross-functioning teams
Customer's Role	Important	Critical
Development Model	Life-cycle model (waterfall, spiral, or any other variation)	Iterative and incremental
Communication	Formal	Informal
Management Style	Command and control	Leadership and collaboration
Quality Control	Heavy planning and strict control with late heavy testing	Continuous control of requirements and development with continuous testing

Agile development encompasses six main methods, where they all share the core values and principles plus some common practices such as on-site customer, short iterations, frequent releases, prioritisation based on features delivering the highest business value for the customer, simple design, and peer reviews [87]. Intuitively, each method defines its own practices based on the vision of that method. Next, these methods will be briefly addressed.

3.1 Scrum

It focuses on management in situations where it is difficult to plan ahead. The features to be implemented in the system are listed in the backlog, where the product owner (the voice of the customer) decides which of these items are to be developed in each sprint (iteration); the development team, then, self-organises and coordinates their work through various practices such as daily scrums and sprint planning. The Scrum method is greatly supported by a Scrum Master, who makes sure that the process is followed and there are no obstacles hindering the team from working effectively [88].

3.2 Extreme Programming (XP; XP2)

Extreme Programming believes in implementing the best practices of software development. XP adopts 12 practices, which are: simple design, test-driven development, refactoring, pair programming, small releases, metaphor, collective code ownership, planning game, continuous integration, on-site customers, 40-h week, and coding standards. The revised version of XP is called XP2 and it

adopts similar practices to these of XP with slight variations such as: quarterly cycle, weekly cycle, and 10-minute build [89,90].

3.3 Feature-Driven Development (FDD)

FDD is a short-iteration process that focuses on the development of critical systems, where an iteration of a feature consists of both design and building. FDD is a blend of model-driven and agile development with assertion on object models, division of work in features, and iterative design for each feature [91].

3.4 Dynamic Systems Development Method (DSDM)

DSDM is a rapid application development framework that consists of 3 phases: (1) pre-project (for the feasibility study); (2) project life-cycle (for the business study); and (3) post-project (for the development). DSDM adopts 9 main principles: empowering the project team, iterative and incremental development, testing throughout the development lifecycle, frequent delivery, efficient and effective communication, user involvement, addressing current business needs, allowing reversing changes, and high-level scope being fixed before project starts [92].

3.5 The Crystal Methodologies

They are a family of methodologies: Clear, Yellow, Orange, Red, Magenta, Blue, and Violet. As with real crystals, each colour of these methodologies corresponds to the project size and criticality. The Clear methodology is considered the most agile or the lightest weight crystal methodology and hence it gained a lot of popularity [93]. The main focus of the Clear methodology is communications in small teams developing non-life critical systems and it adopts the following 6 principles: frequent delivery, regular reflection workshops, osmotic communication, easy access to expert real users, and code versioning tools [93].

3.6 Adaptive Software Development

Provides a framework, with enough guidance, to save large and complex projects from falling in chaos; it strongly encourages incremental iterative development with constant prototyping. Adaptive software development consists of three phases: (1) speculation (recognising the uncertain nature of complex problems and encouraging exploration and experimentation); (2) collaboration (working jointly to produce results, share knowledge, or make decisions); and (3) learning (acknowledging failure and positively reacts to mistakes). The principles of this method are: mission focused, feature based, iterative, time-boxed, risk driven, and change tolerant [94].

4 Agile Requirements Engineering

Both vagueness and frequent changes in requirements are unavoidable. Thus, following the traditional RE practices to develop clear, consistent, and complete requirements, before the design and the implementation begin, seems to be infeasible.

The ability of ASD to embrace continuous changes in requirements, accommodate the growing technology evolution, and deliver working software early to market, has made it a very appealing software development approach. In this section, the light is shed on the agile RE practices.

4.1 Requirements Elicitation

The techniques used in this phase of development are much like those of the traditional development. However in agile, the elicitation process is performed iteratively and continuously, before each development iteration, accentuating the communication of the elaborated requirements with the customer. The most common techniques that are used in agile requirements elicitation are:

- **Face-to-Face Communication**: the main purpose of RE in an agile environment is to help or guide the customer to articulate and communicate his/her needs. This is done through: (1) Interviews, which is the most traditional and commonly used technique. They can be structured with a predetermined set of questions or unstructured where the emphasis is placed more on open discussions [95]; and (2) Joint Application Development (JAD), in which a well structured group session with defined steps, actions, and roles for participants is followed [96,97].
- **User stories**: is the practice used by agile methods to record the system requirements. A user story describes a feature that delivers a value for the customer and usually it is written on a paper note card. On each note card, the description of the story is written at the front while its acceptance test goes at the back. They are also used for planning and documenting the requirements [98,99]. The customer's team takes the responsibility of writing the user stories to guarantee that the stories are written in the business language they understand [98].

  It is intuitive then to conclude that, before writing a user story, the team has to think about what they expect the system to do, and then think about a specific function-ality to be delivered. That process can be regarded as the brainstorming practice in traditional RE.
- **Prototyping**: is one of the fastest and most effective ways to help the customer visualises requirements. Prototyping narrows the gap between what the customer desires to see in the system and what the requirements, when implemented, will actually look like. There are three main types of prototypes: low-fidelity, high-fidelity, and wizard-of-oz prototypes. (1) The low-fidelity prototype is a mockup of user interfaces sketched on a paper or with a computer-based sketching tool (e.g. Balsamiq) [100,101]. (2) The high-fidelity prototype is a realistic mockup of user interfaces that are rapidly implemented in software or as a web page (e.g. Axure), providing limited aspects of functionalities to be simulated or demonstrated (e.g. navigation or a walk-through of a task scenario) [102]. (3) The wizard-of-oz prototype is an acting prototype, in

which a person simulates the responses of the system in response to some user inputs [4].

Unlike traditional development, prototypes in agile are used to help understand the difficult requirements and bridge the gap between the various perspectives of the different stakeholders, and not to be part of the developed system; thus, they are mostly treated as throw-away prototypes.

4.2 Requirements Analysis and Negotiation

Like traditional development, the analysis and negotiation phase in agile checks the requirements for completeness, consistency, essentiality, and feasibility. This is achieved through conducting JADS for all the involved stakeholders to prioritise the already made user stories and to sort out any conflicts/omissions, if any, in the requirements. Since JADs and user stories were covered earlier, next prioritisation is summarised.

- **Prioritisation**: is the practice where the order of the features, to be implemented, is defined. Usually, the core requirements are to be implemented firstly, followed by those of lesser importance. Though in many cases, different stakeholders will assign different priorities to the same requirement – indicating either different perceptions or real needs – a consensus has to be reached to determine one priority level for each requirement [4].

  Despite the fact that the prioritisation practice exists in both approaches (agile and traditional), there are still two main differences: the criteria on which prioritisation is based and the frequency of performing prioritisation [103]. In agile, prioritisation takes place before each development iteration, and not just one time, as in the traditional approach, before the development starts. Unlike traditional development that prioritises requirements based on many criteria (i.e. risks, cost, implementation dependencies, business value, and time), the agile approach prioritises requirements based mostly on the highest business value they deliver.

4.3 Requirements Modelling

Agile modelling is very different in purpose when compared to that of the traditional development. The basic idea of agile modelling is to guide the developing team to build models that resolve the problems of design without the need to overbuild these models [104]. In other words, agile models are used to help understand the small part of the system currently under development without the need to invest a lot of time and effort in overdeveloping them, because most probably they will not become a part of the system documentation. They are mostly temporary throw-away models that are drawn on a whiteboard or a paper that discarded after fulfilling their purpose [104].

Unlike ASD, traditional development keeps all the models, at the different levels of abstractions, to become a part of the system documentation, and that needs to be kept up-to-date with any future alterations.

4.4 Requirements Validation

Agile methods use frequent review meetings and various tests to validate requirements. Though ASD shares the same techniques as those of the traditional approach, there are still some differences; further details come next.

- **Review Meetings:** unlike traditional development, when agile validates requirements, it actually is validating a working piece of software and not a huge requirements specifications document. At the end of each development cycle, a review meeting is held to communicate and resolve the concerns and issues that have been raised during that cycle. Review meetings keep the development team along with the rest of the stakeholders on the same track, while highlighting any problems as early as possible. In addition, conducting review sessions helps increasing the trust and confidence of the customer in the developing team when the output of the review dictates that the project is on the right track.
- **Testing:** is another method for validating the output of a development cycle. The most common testing methods are acceptance test (AT) [98], test-driven development (TDD) [105,106], and acceptance test-driven development (ATDD) [107]. AT is considered the satisfaction condition(s) that determines whether a feature is successfully implemented. It treats the requirement as a black box, once that feature passes the test, then no more work will be introduced to that feature and the team starts to develop the next feature in queue. TDD, on the other hand, is an evolutionary testing approach where tests are developed before the code is written. Prior coding, these tests are expected to fail, but eventually, when the implementation is finished, they should be passed. The ATDD test is a set of process patterns that helps the team build the right software product. It has four main advantages: (1) keeps the documentation living; (2) guarantees a higher product quality; (3) decreases the amount of rework; and (4) enhances the alignment of activities of the different roles on a project. Hence, it can be concluded that ATDD mostly sorts out the challenges of change/version control, requirements tracing, and requirements status tracking faced in the traditional approach.

The main difference between traditional and agile development in requirements validation is the strong emphasis placed on testing in the agile methods.

4.5 Requirements Management

In traditional development, requirements are primarily managed through keeping a documentation that captures, stores, and traces all the states of each requirement since it was initially elaborated in the elicitation stage until its implementation is finished. Though keeping all these information provides relationships between the requirements, the design, and the implementation of the system, writing and managing such a huge documentation is a complicated and time-consuming task. Not to mention, the high risk that one

might get lost when searching for a piece of information in that huge documentation.

Agile development, on the other hand, believes that changes in the system requirements is inevitable; thus, welcoming and accommodating changes at any stage in the project is the core essence of the agile methods. The agile approach makes it easier and less expensive to implement changes compared to that of the traditional development. Managing requirements changes in agile methods is achieved through three main practices: iterative requirements engineering, frequent short releases, and immediate customer's feedback.

- **Iterative RE**: requirements are not defined prior to the start of the development but rather they emerge throughout the process. In the beginning, the development team obtains the big picture of the most critical (with the highest business value) features of the application, through performing a high-level requirements analysis (no detailed specifications at that stage). Then, before the start of each development iteration, the RE process is performed in order to collect more details about the features to be implemented in that iteration.
- **Frequent Short Releases**: is the practice of delivering working pieces of software frequently, in order to make the customer realises the expected results faster, and to enable them to form a mental map of how the system would look like. If the output does not conform to the customer's needs, then the required changes will be pointed out at the earliest possible time of the project. Additionally, each release adds up to the experience of the development team; thus leading to a continuous improvement in the performance of the developing team in the subsequent releases.
- **Immediate Customer's Feedback**: at the end of each iteration, the customer shares their feedback with the development team, requesting changes if their expectations are not met. Moreover, the customer conducts a planning meeting, before each iteration, to align their purpose with the development team and to ensure that the team has all the necessary details that will enable them to successfully deliver the next iteration.

To conclude, managing requirements change through constant planning minimises the probability of having post-development changes; hence, no or very little extra cost is incurred by the project.

To this end, Table 2. summarises the commonalities and differences between traditional and agile approaches for the main tasks of RE, underlining the associated agile RE practices for each task.

TABLE 2
TRADITIONAL VERSUS AGILE RE TASKS

RE Tasks	Traditional	Agile	Agile RE Practices
Requirements Elicitation	Discovers all the requirements upfront prior to developing the system	Discovers the requirements iteratively and incrementaly throughout the development process	▪ Interviews and persona discovery ▪ JADs ▪ User stories ▪ Prototyping
Requirements Analysis and Negotiation	Checks the requirements' feasibilty, essentiality, consistency, and completeness; in addition to prioritising them	Refines, changes, and prioritises the requirements iteratively	▪ User stories ▪ JADs ▪ Prioritisation
Requirements Modelling	Provides a formal visual representation to the entire system	Communicates understaning of the small part of the system currently under development	Low-fidelty and traditional models
Requirements Validation	Ensures the consistency and the completeness of the requirements document	Ensures that the current software release reflects the current needs of the customer	▪ Review meetings ▪ AT ▪ TDD ▪ ATDD
Requirements Management	Tracks changes in requirements, design, or documentaiton to understand why any changes were made, through keeping intensive documentation of the system	Tracks changes with minimal documentation; user stores are written on index cards or electronically, and are maintained in the product backlog/feature list	▪ Iterative RE ▪ Frequent short releases ▪ Immediate customer's feedback

5 CHALLENGES OF AGILE RE PRACTICES

Despite the many advantages offered by ASD and the various merits that come along with its practices, there are still some associated challenges. The main challenges of the agile approach are cost and schedule estimation, non-functional requirements, and customer's access and participation.

1. **Cost and Schedule Estimation**: it is very difficult to provide an accurate cost and schedule estimation in ASD, because the initial estimation of the project size is based on the known user stories at that time. Actually, estimation for completion dates in agile projects is not advised because embracing changes will undoubtedly make these dates obsolete [108]. However, the short iterations and the frequent feedback, eventually, enable the development team to provide better estimates for each iteration. Hence, it can be concluded that agile development delivers more precise cost and schedule estimates, but cannot deliver such an accuracy level at the beginning of the project.
2. **Non-Functional Requirements (NFR):** in spite of the critical role of NFR in RE for the success of a project [109], neither agile nor traditional development has addressed it properly. Though the research in the past decade is urging the development of not only adequate functional requirements but also adequate NFR [110], the majority of

the work focusing on NFR is still partial and incomplete, as integrating NFR into the different phases of software development is still difficult and very much challenging [111].

3. **Customer's Access and Participation**: In custom-developed projects, agile development strongly believes in the effective and intensive communication between the customer and the developing team [112]. In order to realise this, three factors are assumed: (1) the developing team has an on-site customer; (2) the customer trusts the developing team; and (3) the development team manages to reach a consensus among all the customer's representatives.

The first factor is difficult to accomplish except only in XP teams, where a full-time on-site customer is dictated. Since the customer's representatives are expected to be usually busy doing their original jobs, the best-case scenario, then, is to have a part-time on-site customer's representative. The second factor is most likely to come true probably after seeing the output of the first few iterations and how the time-to-market has been minimised, while delivering stable working software with the most important business value. The last factor is quite challenging, as it is very hard to unify the perspectives of the different customer's representatives involved in the project. To resolve this issue, the team will be spending extra effort to negotiate the requirements with each representative and then integrate the resulting individual outputs.

6 Promising Agile RE Research Areas

In spite of the large amount of literature and work explored in the earlier sections, the RE field is still full of plenty of rich research areas. In this section, the most promising agile RE-related areas are explored.

1. **Scalability**: the ongoing growth of size in software systems has drawn the attention of many practitioners and researchers even more than it did before. Scale is not monopolised by its usual referral to significant size; there are many other scale factors such as complexity, variability management in software product lines (SPL) [113], and degree of heterogeneity in distributed systems.
2. **Software Reuse**: facilitating the reuse of existing software components is a critical step in making the RE tasks more prescriptive and systematic. This research area is of a paramount importance SPL. The RE challenges in SPL are: developing strategic and effective techniques for analysing domains, and how to efficiently and effectively manage and document variability.
3. **Self-Adaptive Systems**: are systems that can automatically maintain themselves throughout the different scenarios, such as the self-healing systems that dynamically recover from a system failure, faults, errors, or security breaches [114]. The RE research problems in self-adaptive systems include but not limited to: (1) identifying and specifying thresholds for when the system should adapt; (2) specifying variable sets of requirements; (3) monitoring the system and environment in comparison with the elicitated requirements, in order to ensure that the behaviour of the system meets the specified requirements when operating in a dynamic environment; and (4) verifying models of adaptive systems and their sets of possible behaviours [115].
4. **Cyber-Physical Systems (CPS):** the increasing dependence on systems that are strongly coupled with the monitoring and the controlling of entities in the physical world has widely encouraged the research in the CPS field. A CPS is an engineered system where there is a tight combination of, and coordination between, its computational and physical elements; for example: sensor-based and autonomous automotive systems [116]. One of the RE dilemmas in CPS of assigning responsibilities to: the software system under consideration, the peer software systems, the hardware interface devices, the human operators, and the users [117]. With such new challenges, a call for new concepts to model, simulate, verify, validate, and visualise the behaviour of the physical and the human entities and their interfaces with the computing system has become crucial.
5. **Non-Functional Requirements**: the obvious neglect for NFR in research and practice, in comparison with the received attention in functional requirements, made it a very rich area for research.
6. **Methodologies, Patterns, and Tools** [6]: information on applying RE technologies, methodically, is essential to take these technologies from research to practice. Due to the partial solutions offered or suggested by the existing patterns and strategies, some level of uniformity and predictability is entailed in the resulting requirements. Research into how to incorporate requirements technologies into an integral requirements process is needed. This is a core challenge that requires approaches to cross-connect the different elements (goals, scenarios, data, functions, state-based behaviour, and constraints) of requirements modelling [6].

7 Conclusion

In the past decade, agile development has gained a significant global recognition affined to software projects' success due to its core concept of valuing individuals and interactions, working software, customer collaboration, and responding to change. We strongly believe that agile perfectly fits the changing nature of RE, thus bringing agile into the world of RE will guarantee a greater and a faster success.

RE researchers have the opportunity to enhance and further extend their work, through the adoption of ASD methods. There are many promising RE research areas in which agile can be perfectly fit such as solving problems related to variability and components reuse in SPL, verifying models of adaptive systems, or developing tools/approaches/ techniques/patterns to help taking technologies from research to practice. Other RE areas that can benefit from the agile

adoption include but are not limited to: NFRs, CPS, and scale factors. As for our work, we are much interested in two directions: SPL and CPS. We, however, have not yet decided towards which area our research is heading; a further in-depth investigation will be carried out.

In this paper, we reviewed the research state-of-the-art in traditional RE, the various methods of the ASD, the most common RE practices used in agile and the challenges imposed by their adoption, the commonalities and differences between agile and traditional RE approaches, and finally the prime research opportunities in the agile RE field.

REFERENCES

[1] K. Beck, M. Beedle, A.V. Bennekum, A. Cockbur, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R.C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development," http://www.agilemanifesto.org. 2001.

[2] L. Westfall, "Software Requirements Engineering: What, Why, Who, When and How," The Westfall Team, http://www.westfallteam.com/Papers/The_Why_What_Who_When_and_How_Of_Software_Requirements.pdf. 2006.

[3] The Standish Group, "The CHAOS Manifesto 2011 Report," http://blog.standishgroup.com/cm2011. 2011.

[4] I. Sommerville and P. Sawyer, *Requirements Engineering*, John Wiley & Sons, 1997.

[5] A. Sillitti, M. Ceschi, B. Russo, and G. Succi, "Managing uncertainty in requirements: a survey in documentation-driven and agile companies," *Proc. 11th IEEE Int. Symposium on Software Metrics*, pp. 10–17, 2005.

[6] B. H. C. Cheng and J. M. Atlee, "Research Directions in Requirements Engineering," *Proc. Future of Software Engineering*, pp. 285–303, 2007.

[7] A. Alfonso, V. Braberman, N. Kicillof, and A. Olivero, "Visual Timed Event Scenarios," *Proc. 26th International Conf. on Software Engineering*, pp. 168–177, 2004.

[8] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "TROPOS: An Agent-Oriented Software Development Methodology," *J. Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[9] E. Letier and A. V. Lamsweerde, "Agent-based Tactics for Goal-Oriented Requirements Elaboration," *Proc. Int. Conf. on Software Engineering*, pp. 83–93, 2002.

[10] E. Yu., "Agent Orientation as a Modelling Paradigm," *J. Wirtschaftsinformatik*, vol. 43, no. 3, pp. 123–132, 2001.

[11] W. Damm and D. Harel, "LSCs: Breathing Life into Message Sequence Charts," J. *Formal Methods in Systems Description*, vol. 19, no. 1, pp. 45–80, 2001.

[12] A. Cockburn, *Writing Effective Use Cases*, Addison-Welsey, 2001.

[13] G. Sindre and A.L. Opdahl, "Templates for Misuse Case Description," *Proc. 7th Int. Workshop on Requirements Engineering, Foundation for Software Quality*, pp. 125–136, 2001.

[14] S. Uchitel, J. Kramer, and J. Magee, "Negative Scenarios for Implied Scenario Elicitation," Newsletter *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 6, pp. 109–118, 2002.

[15] A.V. Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-models," *Proc. 26th Int. Conf. on Software Engineering*, pp. 148–157, 2004.

[16] L. Chung and J.C.P. Leite, "On Non-Functional Requirements in Software Engineering," *Conceptual Modeling: Foundations and Applications*, Springer-Verlag Berlin, Heifelberg, pp. 363–379, 2009.

[17] A.V. Lamsweerde, "Goal-Oriented Requirements Engineering: A guided tour," *Proc. 5th IEEE Int. Symposium on Software Engineering*, pp. 249–263, 2001.

[18] S. Anwer and N. Ikram, "Goal Oriented Requirement Engineering: A Critical Study of Techniques," *Proc. 13th Asia Pacific Conf. Software Engineering*, pp. 121–130, 2006.

[19] T.D. Breaux and A.I. Anton, "Analysing Goal Semantics for Rights, Permissions, and Obligations," *Proc. 13th IEEE Int. Conf. on Requirements Engineering*, pp. 177–188, 2005.

[20] H. Sharp, A. Finkelstein, and G. Galal, "Stakeholder identification in the Requirements Engineering Process," *Proc. 10th Int. Workshop on Database and Expert Systems Applications*, pp. 387–391, 1999.

[21] A. Cooper, *The Inmates are Running the Asylum*, Macmillan Publishing Co., Inc., 1999.

[22] M. Aoyama, "Persona-and-Scenario Based Requirements Engineering for Software Embedded in Digital Consumer Products," *Proc. 13th IEEE Int. Conf. on Requirements Engineering*, pp. 85–94, 2005.

[23] C. Potts, "Metaphors of Intent," *9th IEEE Int. Conf. on Requirements Engineering*, pp. 31–39, 2001.

[24] T. Cohene and S. Easterbrook, "Contextual Risk Analysis for Interview Design," *13th IEEE Int. Conf. on Requirements Engineering*, pp. 95–104, 2005.

[25] A. Sutcliffe, S. Fickas, and M.M. Sohlberg, "PC-RE: A Method for Personal and Context Requirements Engineering with Some Experience," *J. Requirements Engineering*, vol. 11, no. 3, pp. 157–173, 2006.

[26] N. Maiden and S. Robertson, "Integrating creativity into requirements processes: Experiences with an air traffic management system," *Proc. 13th IEEE Conf. on Requirements Engineering*, pp. 105–116, 2005.

[27] R. France and B. Rumpe, "Model-Driven Development of Complex Systems: A research Roadmap," *Proc. Future of Software Engineering*, pp. 37–54, 2007.

[28] G. Gabrysiak, H. Giese, and A. Seibel, "Interactive Visualization for Elicitation and Validation of Requirements with Scenario-Based Prototyping," *Proc. 4th Int. Workshop on Requirements Engineering Visualisation*, pp. 41–45, 2009.

[29] H. L. McQuaid, A. Goel, and M. McManus, "When You Can't Talk to Customers: Using Storyboards and Narratives to Elicit Empathy for Users," *Proc. 2003 Int. Conf. on Desigining Pleasurable Products and Interfaces*, pp. 120–125, 2003.

[30] H.T. Van, A.V. Lamsweerde, P. Massonet, and C. Ponsard, "Goal-Oriented Requirements Animation," *Proc. 12th IEEE Int. Conf. on Requirements Engineering*, pp. 218–228, 2004.

[31] J. Magee, N. Pryce, D. Giannakopoulou, and J. Kramer, "Graphical Animation of Behavior Models," *Proc. 22nd Int. Conf. on Software Engineering*, pp. 499–508, 2000.

[32] H. In, B. Boehm, T. Rodgers, and M. Deustsch, "Applying WinWin to Quality Requirements: A Case Study," *Proc. 23rd Int. Conf. on Software Engineering*, pp. 555–564, 2001.

[33] C. Rolland and N. Prakash, "Matching ERP System Functionality to Customer Requirements," *Proc. 5th IEEE Int. Symposium on Requirements Engineering*, pp. 66–75, 2001.

[34] C. Alves and A. Finkelstein, "Challenges in COTS Decision Making: A Goal-Driven Requirements Engineering Perspective," *Proc. 14th Int. Conf. on Software Engineering and Kowledge Engineering*, pp. 789–794, 2002.

[35] W.N. Robinson, S.D. Pawlowski, and V. Volkov, "Requirements Interaction Management," *J. ACM Computing Surveys*, vol. 35, no. 2, pp. 132–190, 2003.

[36] Y. Lee and W. Zhao, "An Ontology-Based Approach for Domain Requirements Elicitation and Analysis," *Proc. 1st Int. Multi-Symposiums on Computer and Computational Sciences*, vol 2, pp.364–371, 2006.

[37] F. Chantree, B. Nuseibeh, A.D. Roeck, and A. Willis, "Identifying Nocuous Ambiguities in Natural Language Requirements," *Proc. 14th IEEE Int. Conf. on Requirements Engineering*, pp. 59–68, 2006.

[38] K.S. Wasson, "A case study in systematic improvement of language for requirements," *Proc. 14th IEEE Int. Conf. on Requirements Engineering*, pp. 6–15, 2006.

[39] P. Sawyer, P. Rayson, and K. Cosh, "Shallow knowledge as an aid to deep understanding in early phase requirements engineering," *IEEE*

Trans. on Software Engingeering, vol. 31, no. 11, pp. 969–981, 2005, doi: 10.1109/TSE.2005.129.

[40] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, "Application of Linguistic Techniques for Use Case Analysis," Proc. *IEEE Joint Int. Conf. on Requirements Engineering*, pp. 157–164, 2002.

[41] C. Nentwich, W. Emmerich, A. Finkelsteiin, and E. Ellmer, "Flexible Consistency Checking," *J. ACM Trans. on Software Engineering and Methodolofy*, vol. 12, no. 1, pp. 28–63, 2003.

[42] C.L. Heitmeyer, R.D. Jeffords, and B.G. Labaw, "Automated Consistency Checking of Requirements Specifications," *J. ACM Trans. on Software Engineering and Methodology*, vol. 5, no. 3, pp. 231–261, 1996.

[43] G. Engels, J.M. Kuster, R. Heckel, and L. Groenewegen, "A Methodology for Specifying and Analyzing Consistency of Object-Oriented Behavioral Models," *Proc. 8th European Software Engineering Conf. jointly with the 9th ACM SIGSOFT Int. Symposium on Foundations of Software Engineering*, pp. 186–195, 2001.

[44] L.A. Campbell, B.H.C. Cheng, W.E. McUmber, and R.E.K. Stirewalt, "Automatically Detecting and Visualizing Errors in UML Diagrams," *J. Requirements Enginering*, vol. 7, no. 4, pp. 264–287, 2002.

[45] T.C. de Sousa, J.R. Almeida Jr., S. Viana, and J. Pavón, "Automatic Analysis of Requirements Consistency with the B Method," *Newsletter SIGSOFT Software Engineering Notes,* vol. 35, no. 2, pp. 1–4, 2010.

[46] M. Kamalrudin, "Automated Software Tool Support for Checking the Inconsistency of Requirements," *Proc. 2009 IEEE/ACM Int. Conf. on Automated Software Engineering*, pp. 693–697, 2009.

[47] A.V. Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Trans. on Software Engineering*, vol. 26, no. 10, pp. 978–1005, 2000, doi: 10.1109/32.879820.

[48] R. Lutz, A. Patterson-Hine, S. Nelson, C.R. Frost, D. Tal, and R. Harris, "Using Obstacle Analysis to Identify Contingency Requirements on an Unpiloted Aerial Vehicle," *J. Requirements Engineering*, vol. 12, no. 1, pp. 41–54, 2007.

[49] P. Baker, P. Bristow, C. Jervis, D. Kind, R. Thomson, B. Mitchell, and S. Burton, "Detecting and Resolving Semantic Pathologies in UML Sequence Diagrams," *Proc. 10th European Software Engineering Conf. jointly with the 13th ACM SIGSOFT Int. Symposium on Foundation of Software Engineering*, pp. 50–59, 2005.

[50] J.H. Hausmann, R. Heckel, and G. Taentzer, "Detection of Conflicting Functional Requirements in a Use Case-Driven Approach: A Static Analysis Technique Based on Graph Transformation," *Proc. 24th Int. Conf. on Software Engineering*, pp. 105–115, 2002.

[51] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J.N.O. Dag"An Industrial Survey of Requirements Interdependencies in Software Product Release Planning," *Proc. 5th IEEE Int. Symposium on Requirements Engineering*, vol. 84–93, 2001.

[52] M.S. Feather, "Towards a Unified Approach to the Representation of, and Reasoning with, Probabilistic Risk Information about Software and its System Interface," *Proc. 15th Int. Symposium on Software Reliable Engineering*, pp. 391–402, 2004.

[53] A.V. Knethen and M. Grund, "QuaTrace: A Tool Environment for (Semi-) Automatic Impact Analysis Based on Traces," *Proc. Int. Conf. on Software Maintenance*, pp. 246–255, 2003.

[54] S. Imtiaz and N. Ikram, and S. Imtiaz, "Impact Analysis from Multiple Perspectives: Evaluation of Traceability Techniques," *Proc. 3rd Int. Conf. on Software Engineering Advances*, pp. 457–464, 2008.

[55] A. Sutcliffe, W.C. Chang, and R. Neville, "Evolutionary Requirements Analysis," *Proc. 11th IEEE Int. Conf. on Requirements Engineering*, pp. 264–273, 2003.

[56] A. Moreira, A. Rashid, and J. Araujo, "Multi-Dimensional Separation of Concerns in Requirements Engineering," *Proc. 13th IEEE Int. Conf. on Requirements Engineering*, pp. 285–296, 2005.

[57] B. Regnell, L. Karlsson, and M. Host, "An Analytical Model for Requirements Selection Quality Evaluation in Product Software Development," *Proc. 11th IEEE Int. Conf. on Requirements Enginering*, pp. 254–263, 2003.

[58] S. Liaskos, A. Lapouchnian, Y. Yijun, E. Yu, and J. Mylopoulous, "On Goal-Based Variability Acquisition and Analysis," *Proc. 14th IEEE Int. Conf. Requirements Engineering*, pp. 79 –88, 2006.

[59] B. Gonzalez-Baixauli, J.C.S.D.P. Leite, and J. Mylopoulos, "Visual Variability Analysis for Goal Models," *Proc. 12th IEEE Int. Conf. on Requirements Engineering*, pp. 198–207, 2004.

[60] S. Lauesen, "COTS Tenders and Integration Requirements," *J. Requirements Enginering*, vol. 11, no. 2, pp. 111–122, 2006.

[61] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*, Springer-Verlag, 2011.

[62] P.P.S. Chen, "The Entity-Relationship Model – Toward a Unified View of Data," *J. ACM Transactions on Database Systems – Special Issue: Papers from the Int. Conf. on Very Large databases*, vol. 1, no. 1, pp. 9–36, 1976.

[63] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *J. Science Computer Programming*, vol. 8, no. 3, pp. 9 231–274, 1987.

[64] P. Darke and G. Shanks, "User Viewpoint Modelling: Understanding and Representing User Viewpoints During Requirements Definition," *J. Information Systems*, vol. 7, no. 3, pp. 213–219, 1997.

[65] D.T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Trans. on Software Engineering*, vol. 3, no. 1, pp. 16–34, 1977, doi: 10.1109/TSE.1977.229900

[66] G. Kotonya and I. Sommerville, "Requirements Engineering with Viewpoints," J. Software Engineering, vol. 11, no. 1, pp. 5–11, 1996.

[67] P. Coad and E. Yourdon, *Object-Oriented Analysis*, Prentice-Hall, 1991.

[68] J.R. Rumbaugh, M.R. Blaha, W. Lorensen, F. Eddy, and W. Premerlani, *Object Modeling and Design*, Prentice-Hall, 1991.

[69] X. Liu, H. Yang, and H. Zedan, "Formal Methods for the Re-Engineering of Computing Systems: A comparison," *Proc. 21st Ann. Int. Conf. on Computer Software and Applications*, pp. 409–414, 1997.

[70] OMG, "*The Unified Modelling Language Version 2"*, www.omg.org. 2003.

[71] K. Ryan, "The Role of Natural Language in Requirements Engineering," *Proc. IEEE Int. Symposium on Requirements Engineering*, pp. 240–242, 1993.

[72] J.M. Thompson, M.P.E. Heimdahl, and S.P. Miller, "Specification-Based Prototyping for Embedded Systems," *Proc. 7th European Software Engineering Conf. jointly with ACM SIGSOFT Int. Syposium on Foundations of Software Engineering*, pp. 163–179, 1999.

[73] C. Heitmeyer, J. Kirby, B. Labaw, and R. Bharadwaj, "SCR*: A Toolset for Specifying and Analyzing Software Requirements," *Proc. 10th Ann. Int. Conf. on Computer-Aided Verification*, pp. 526–531, 1998.

[74] R. Jeffords and C. Heitmeyer, "Automatic Generation of State Invariants from Requirements Specifications," *Newsletter ACM SIGSOFT Software Engineering Notes*, vol. 23, no. 6, pp. 56–69, 1998.

[75] R. Jhala and R. Majumdar, "Software Model Checking," J. ACM Computing Surveys, vol. 41, no. 4, pp. 21:1--12:54, 2009.

[76] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*, The MIT Press, 2006.

[77] L.K. Dillon and R.E.K. Stirewalt, "Inference Graphs: A Computational Structure Supporting Generation of Customizable and Correct Analysis Components," *IEEE Trans. on Software Engineering*, vol. 29, no. 2, pp. 133–150, 2003, doi: 10.1109/TSE.2003.1178052.

[78] T. Bultan, "Action Language: A Specification Language for Model Checking Reactive Systems," *Proc. 2000 Int. Conf. on Software Engineering*, pp. 335–344, 2000.

[79] J. Clelland-Huang, R. Settimi, O. BenKhadra, and E. Berezhanskaya, "Goal-Centric Traceability for Managing Non-Functional Requirements," *Proc. 27th Int. Conf. on Software Engineering*, pp. 362–371, 2005.

[80] M. Sabetzadeh and S. Easterbrook, "Traceability in Viewpoint Merging: A Model Management Perspective," *Proc. 3rd Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 44–49, 2005.

[81] A. Marcus and J. Maletic, "Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing," *Proc. 25th Int. Conf. on Software Engineering*, pp. 125–135, 2003.

[82] J.H. Hayes, A. Dekhtyar, and S.K. Sundaram, "Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods," *IEEE Trans. on Software Engineering*, vol. 32, no. 1, pp. 4–19, 2006, doi:

10.1109/TSE.2006.3.
[83] J. Cleland-Huang, G. Zemont, and W. Lukasik, "A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability," *Proc. 12th IEEE Int. Conf. Requirements Engineering*, pp. 230–239, 2004.
[84] D. Bush and A. Finkelstein, "Requirements Stability Assessment Using Scenarios," *Proc. 11th IEEE Int. Conf. Requirements Engineering*, pp. 23–32, 2003.
[85] L. Williams and A. Cockburn, "Agile Software Development: It's About Feedback and Change," *Computer Magasine*, vol. 36, no. 6, pp. 39–43, 2003.
[86] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of migrating to agile methodologies," Communication of the ACM Magasine – Adaptive Complex Enterprises, vol. 8, no. 5, pp. 72–78, 2005.
[87] B. Bohem and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, 2003.
[88] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Prentice Hall, 2001.
[89] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.
[90] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2004.
[91] S. R. Palmer and J. M. Felsing, *A Practical Guide to Feature-driven Development*, Prentice Hall, 2002.
[92] DSDM Consortium and J. Stapleton, *DSDM: Business Focused Development*, Pearson Education, 2003.
[93] A. Cockburn, *Crystal Clear: A Human-Powered Methodology for Small Teams*, Addison-Wesley, 2004.
[94] J.A. Highsmith, *Adaptive Software Development*, Dorset House, 1996.
[95] K. Holtzblatt and H.R. Beyer, "Requirements Gathering: The Human Factor," *Communication of the ACM Magasine*, vol. 38, no. 5, pp. 30–32, 1995.
[96] L.A. Macaulay, *Requirements Engineering*, Springer-Verlag, 1996.
[97] J. Wood and D. Silver, *Joint Application Development*, John Wiley & Sons, 1995.
[98] M. Cohn, *User Stories Applied for Agile Software Development*, Addison-Wesley, 2004.
[99] J. Savolainen, J. Kuusela, and A. Vilavaara, "Transition to Agile Development – Rediscovery of Important Requirements Engineering Practices," *Proc. 18th IEEE Int. Conf. on Requirements Engineering*, pp. 289–294, 2010.
[100] M. Kamalrudin and J. Grundy, "Generating Essential User Interface Prototypes to Validate Requirements," *Proc. 26th IEEE/ACM Int. Conf. on Auted Software Engineering*, pp. 564 – 567, 2011.
[101] A. Hosseini-Khayat, T.D. Hellmann, and F. Maurer, "Distributed and Automated Usability Testing of Low-Fidelity Prototypes," *Proc. Ann. Agile Conf.*, pp. 59–66 , 2010.
[102] Z. Hussain, W. Slany, and A. Holzinger, "Investigating User-Centered Design in Practice: A Grounded Theory Perspective," *5th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society*, Springer Berlin, pp. 279–289, 2009.
[103] D. Firesmith, "Prioritizing requirements," *J. Object Technology*, vol .3, no. 8, pp. 35–47, 2004.
[104] S.W. Ambler, *Agile Modeling*, John Wiley & Sons, 2001.
[105] K. Beck, *Test Driven Development: By Example*, Addison-Wesley Professional, 2003.
[106] D. Astels, *Test Driven Development: A Practical Guide*, Prentice Hall, 2003.
[107] G. Adzic, *Specification by Example: How Successful Team Deliver the Right Software*, Manning Publications Co., 2011.
[108] K. Beck, "Embracing change with extreme programming," *Computer Magasine*, vol. 32, no. 10, pp. 70–77, 1999.
[109] A. Matoussi and R. Laleau, "A Survey of Non-Functional Requirements in Software Development Process," Laboratory *of Algorithmic, Complexity, and Logic*. Technical Report. TR–LACL–2008–7, 2008.
[110] L.M. Cysneiros and J.C.S. do Prado Leite, "Non-Functional Requirements: From Elicitation to Modeling Languages," Computer Magasine, vol. 35, no. 3, pp. 8–9, 2002.
[111] S. Ullah, M. Iqbal, and A.M. Khan, "A survey on Issues in Non-Functional Requirements Elicitation," *Proc. 2011 Int. Conf. on Computer Networks and Information Technology*, pp. 333–340, 2011.
[112] M. Keil and E. Carmel, "Customer-Developer Links in Software Development," *Communication of the ACM Magasine*, vol. 38, no. 5, pp. 33–44, 1995.
[113] L. Chen and M.A. Babar, "A Systematic Review of Evaluation of Variability Management Approaches in Software Product Lines," *J. Information and Software Technology*, vol. 53, no. 4, pp. 344–362, 2011.
[114] S. Ramamoorthy, S.P. Rajagopalan, and S. Sathyalakshmi, "Process for Security in Self-Healing Systems' Architecture," *Proc. Int. Conf. on Sustainable Energy and Intelligent Systems*, pp. 839–843, 2011.
[115] N.A. Qureshi and A. Perini, "Engineering Adaptive Requirements," *Proc. ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pp. 126–131, 2009.
[116] R. Rajkumar, L. Insup, S. Lui, and J. Stankovic, "Cyber-Physical Systems: The Next Computing Revolution," *Proc. 47th ACM/IEEE Design Automation Conf.*, pp. 731–736, 2010.
[117] E. Letier and A.V. Lamsweerde, "Agent-based tactics for goal-oriented requirements elaboration," *Proc. 24th Int. Conf. on Software Engineering*, pp. 83–93, 2002.