UNIVERSITY OF GOTHENBURG

Negotiating Complexity in Test Automation Tool Support

A Design Research Approach

Bachelor of Science Thesis in Software Engineering and Management

JOAKIM GROSS
KRISTOFER HANSSON ASPMAN

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, May 2012

Negotiating Design Complexity in Test Automation Tool Support
A Design Research Approach

JOAKIM GROSS
KRISTOFER HANSSON ASPMAN

Examiner: HELENA HOLMSTRÖM OLSSON

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden May 2012

Negotiating Design Complexity in Test Automation Tool Support

A Design Research Approach

Joakim Gross
Department of Computer Science and Engineering
University of Gothenburg
Göteborg, Sweden
gusgrossjo@student.gu.se

Kristofer Hansson Aspman
Department of Computer Science and Engineering
University of Gothenburg
Göteborg, Sweden
k.hansson.aspman@gmail.com

***Abstract*—Fully realizing the vision of agile processes might require practical tool support to enable activities like test driven design, refactoring, and regression testing. In this paper we will examine in detail, the design process of developing a test automation framework for a company in need of reducing time consuming manual testing. By employing an iterative research process, we will investigate and present what considerations, in both practice and academia, must be taken in order to reach a suitable tool design. The depth of the problem is acknowledged and, as we will see, calls for a complex design process. This process, along with the framework design, is comprehensibly described within this paper. In addition, we will also assess the framework's potential impact on the company's work process.**

Keywords-agile testing; test automation; design science research

1. INTRODUCTION

Within this paper we will report on design science research (Hevner et al., 2004) conducted in close collaboration with a company developing in-vehicle infotainment systems. These systems are designed for user-interaction, and as such, related testing activities often involve much manual interaction. At the company, in order to support the testing activities in agile processes (Fowler and Highsmith, 2001), a need for a test automation framework that facilitates automation of user-interaction dependent tests has been identified. In this study we will focus on the design of such a framework as well as assess how such a framework could potentially impact the current work and development processes.

An increased need for shorter development cycles, shorter time-to-market, and higher quality assurance, has resulted in a growing interest in agile development processes (e.g. Lindvall et al., 2002). Agile allows companies to be efficient, stay flexible, and stay up-to-date with the latest technology, while still producing high quality products (Highsmith and Cockburn, 2001). There are many reports on the adaption to agile software development processes in software organizations (e.g. Puleio, 2006; Shaye, 2008; Moe et al., 2009; Conboy et al., 2011). It is, however, noteworthy that these studies rarely describe the activity as smooth and free of challenges. In fact, many case studies report on challenges and lessons learned when adapting to agile (e.g. Puleio, 2006; Shaye, 2008; Conboy et al., 2011). Testing has been identified as one such challenge and, according to Puleio (2006) and Shaye (2008), it is quite difficult to address.

As agile development processes gains ground, and with the number of systems that require user interaction and the ability to interact with external devices increasing, we argue that the problems addressed in this paper are highly relevant. Other organizations that find themselves in similar situations will benefit from the lessons learned in this paper, both in respect to the implementation itself, and in what it enables. An important aspect for the company in this study is how their product development has been affected by the previous situation where some tests has been time consuming. By considering the studied company's situation and needs, along with related literature, we will investigate how a test automation tool is best designed, and how the complex issues involved can be addressed. Furthermore, we will evaluate the potential impact on the work process, by introducing a prototype test automation framework.

The structure of this paper is an adaptation of the structure suggested by Peffers et al. (2006), made to better suit the nature of our research setting. Section 2 underlines the characteristics of the problem and describes the related literature. In section 3, the essential objectives of a solution are described along with related work. Section 4 begins with a presentation of the process used and then proceeds with a comprehensive report on how our artifact evolved through iterative design and evaluation phases, within the context of the studied company. Design considerations are discussed continuously throughout the section which, eventually, culminates in a more general discussion on the perceived usefulness and impact of the artifact. Finally, in section 5, we present our conclusions and suggest entry points for future work.

2. PROBLEM IDENTIFICATION

With the intention of outlining the characteristics of the problems at the studied company we will begin this section by examining the literature on challenges related to testing

procedures within agile methodologies. By the same reasoning, we will then proceed to investigate the literature on test automation and along with this present a brief description of the risks related to the lack of such automation.

Literature on agile methodologies quite often describes challenges faced by companies trying to adapt to agile ways of working (e.g. Moe et al., 2009; Conboy et al., 2011). Both Puleio (2006) and Shaye (2008) identify testing as the most daunting task faced by teams involved in such adaptions. Puleio (2006) recognized that the importance of automating tests was underestimated by the team described in his report. The organization reported on by Shaye (2008), on the other hand, expected testing to be difficult and they invested much time in approaching it in a proper way. Still, those involved in the transition faced many challenges, e.g. when management demanded prioritization of upcoming releases of the product in development over the construction of automated regression tests, and having to spend much time on trying to write test for modules that were not originally written with product testability in mind.

Within the literature, it has been acknowledged that testing in general, and automated tests in particular, becomes a necessity when changes to the code base are frequent (Coram and Bohner, 2005), for instance when engaging in the activity of refactoring code (Fowler et al., 1999), which is a cornerstone in agile methodology (e.g. Lindvall et al., 2002; Coram and Bohner, 2005). Fowler et al. (1999) state that with frequent refactoring, equally frequent testing is necessary as each refactoring made may introduce defects to the system in question. Thus, an infrastructure for comprehensive regression testing should be considered essential for any company claiming to be agile. The importance of such automated regression testing is discussed by Sommerville (2007), Coram and Bohner (2005), and George and Williams (2004), and benefits of automation are brought up by Shaye (2008) and Sumrell (2007). Also relevant to that discussion, Fowler et al. (1999) describes the risks related to the lack of automated tests e.g. that manually executed tests risk becoming entirely neglected.

The particular area of regression test automation has within the research domain of software testing proven highly relevant to studies of the agile software development processes (e.g. Coram and Bohner, 2005; Shaye, 2008). This is mainly because regression testing helps ensure functionality after a change to the code base has been made (Coram and Bohner, 2005), thus aids the developers engage in e.g. refactoring tasks often necessary in order to respond to changing requirements (Moser et al., 2008). Additionally, the activity of continuous integration becomes less challenging as proper automation of tests and frequently executed regression tests reduce the time spent on both debugging (Fowler et al., 1999; Coram and Bohner, 2005) and testing (Berner et al., 2005; Puleio, 2006; Cervantes, 2009). Such automation could, according to Cervantes (2009), be facilitated by the use of a test automation framework that is easily extended to support future functionality.

Furthermore, because manual testing is an expensive activity, the need for automation increases as testing becomes more frequent (Berner et al., 2005). Even if there is a solid regression test suite in place, if the tests involve time consuming manual interaction (Shaye, 2008; Sumrell, 2007), the tests are at great risk of becoming neglected. User-interaction dependent systems, e.g. systems that interface to a multitude of hardware and wireless services, are thus prone to inadequate quality control.

The practical implications of the literature described in this section have been identified at the studied company, within both testing activities and the work process:

Time consuming testing and *insufficient quality assurance:* The relatively large amount of tests depending on manual interaction from a tester has resulted in infrequent test execution and low test coverage for some components of the product. In a response to this, the company has identified a need for simulation of hardware and user interaction in a way that would enable automation of such manual tests so that time spent executing manual tests is reduced.

Insufficient support for agile work processes: To support the latest in popular services and hardware requires an ability to respond to rapidly changing requirements. At the company this, in turn, implies frequent changes to, and integrations with, the product under development. These frequent updates to the code base of course have to be accompanied by as frequent testing to ensure that no new defects have been introduced to the product. The lack of sufficient support for automated testing has made such testing both time consuming and, in some cases, even non-existent, ultimately impacting responses to change negatively. Additionally, other agile activities such as that of refactoring have been obstructed as a result of difficult and insufficient regression testing, and the company's test-driven approach is hindered by the overall difficulties in designing tests. So, in short, the lack of a support for automated testing makes it difficult for the employees to fully engage in agile activities.

3. OBJECTIVES OF A SOLUTION

Based on the problem identification in section 2, it became apparent that the introduction of a test automation framework would help addressing some core issues at the company. The framework would have to provide the means necessary to increase the amount of test cases possible to execute automatically, increase the frequency by which those test cases could be executed, make it easier to write automated tests, and facilitate refactoring practices. In other words, we were to provide the developers with the tool support necessary for increased developer flexibility.

For the design and development of this framework, we chose an agile process. This allowed iterative work and

suited the study well as iterations were also part of our research process, and because the development work at the company already followed an agile process. During our iterations, a set of quality attributes and requirements was defined and formulated as objectives of a solution. Based on both literature and data collected at the company, these objectives became: To design a test automation framework that fulfills the requirements that it should [1] integrate with the current systems architecture, [2] facilitate automation of previously manual tests, and to have the design fulfill the quality attributes of [3] usability and [4] extendibility.

How these objectives were approached is discussed thoroughly throughout the Design Iteration Focus Process (see section 4.2). There we describe in detail how they were defined from the literature and the data collected during the design iterations. And, through the detailed description of our process, a way to meet these objectives, by considering both the company's practical needs, and relevant literature, is presented. It is worth noting that, during implementation, general literature on development best practices (e.g. Fowler et al., 1999; Sommerville, 2007; Martin, 2009; McConnell, 2009) acted as our primary source of knowledge as the specific nature of our implementation made it difficult to turn to literature in pursuit of specific details of test framework implementation. The resulting design of section 4 can subsequently be viewed as responding to the problem identification of section 2, and the design decisions taken to approach it will be thoroughly described.

4. DESIGN ITERATION FOCUS

Hevner et al. (2004) draws on prior work of e.g. Nunamaker et al. (1991), March and Smith (1995), and Markus et al. (2002) and use this to propose a research approach tailored to suit the iterative nature of design research. In their article they present the *design science research* (DSR) approach and provide the reader with recommendations on how to conduct research where focus lies on the design and development of an artifact.

The approach has received much attention (e.g. Gregor, 2006; Winter, 2008), however, according to Peffers et al. (2006), without resulting in much actual DSR being published. Peffers et al. (2006) argue that the lack of a proper conceptual process and a mental model prevented DSR from gaining the necessary foothold. With the intention of making design science more tangible they propose the *design science research process* (DSRP) model.

The structure, as well as the underlying research process, of this paper is influenced by the DSRP model. Peffers et al. (2006) consider their approach to be well-suited for research carried out in an iterative manner and where focus lies on the design of an artifact. However, due to the nature of our research setting, an adaptation of the DSRP was made. The original process by Peffers et al., our adapted process, and the motivation for the adaption are described in the sections below. From this point on, we refer to the adaption made to the process as the Design Iteration Focus, and the adapted process itself as the Design Iteration Focus Process. Towards the end of section 4 there is a discussion and reflection on the Design Iteration Focus Process.

4.1. The Peffers et al. Design Science Research Process (DSRP)

The DSRP model proposed by Peffers et al. (2006), illustrated in figure 1, not only facilitates the reporting of inherently iterative DSR but also provides a structured way of executing preparation, design, and evaluation phases. The model allows for different entry points depending on the nature of the studied problem and the possibility to iterate over certain phases makes it suitable for research focused on the design and implementation of an artifact. A brief description of the individual activities follows.

The first two phases of the DSRP, i.e. the *Problem identification & motivation* and the *Objectives of a solution* components of figure 1, follows traditional research processes as they are focused on defining the problem, the relevance, and a potential solution. Within the succeeding phases, *Design & development, Demonstration,* and *Evaluation,* the artifact is developed, demonstrated so that feedback can be collected, and then evaluated. Depending on the outcome of the evaluation phase, the researcher may go back to redefine the initial objectives of a solution, back to improve the design of the artifact, or proceed to communicating the results, for instance by summarizing the findings in an article to be submitted for publishing in a scientific journal.

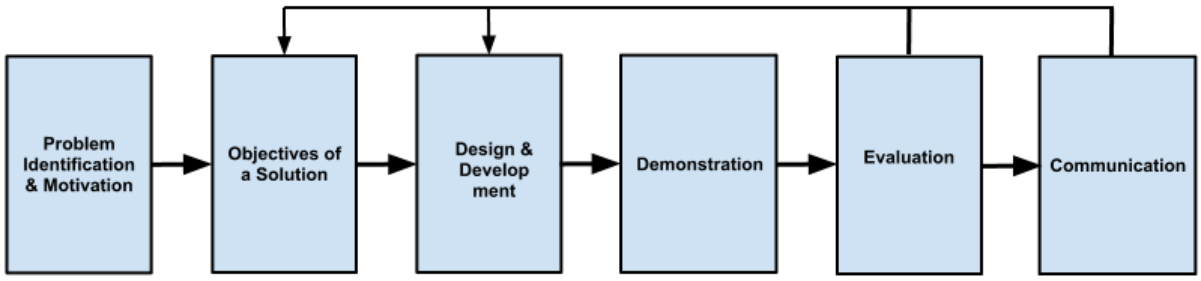

Figure 1: The design science research process (DSRP) model as proposed by Peffers et al., (2006).

4.2. Design Iteration Focus Process

The demonstration context (Peffers et al., 2006) of this study is given by the nature of the study itself. The study has been carried out on site at the company, with us having continuous access to the employees and the ability to take part in everyday activities. The design and development, and evaluation activities therefore became an integral part of everyday work, and demonstrations of partly implemented design, ideas, strategies, and concepts, could be done on a daily basis. In addition, a meeting structure with the specific purpose of demonstrating and evaluating the design was used. Given these conditions, and our interpretation of the DSRP model, we considered it more practical for this study to incorporate the *Design & development*, and *Evaluation* phases with the *Demonstration* phase. For this reason, we chose to adapt the process from Peffers et al. (2006), resulting in a process better suited to the nature of this study (see figure 2).

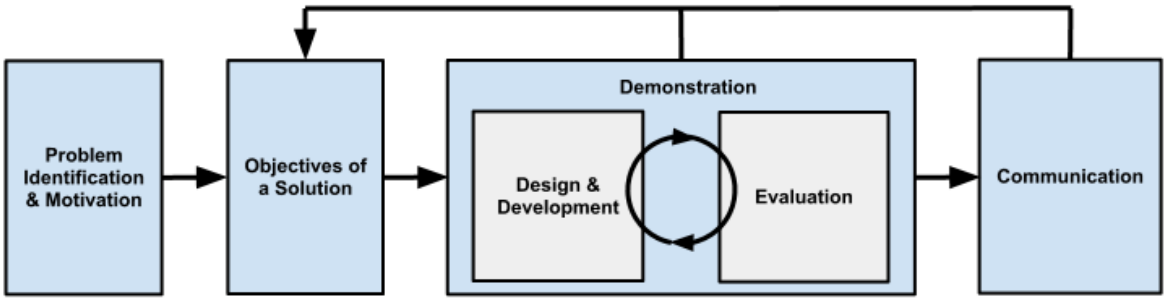

Figure 2: The Design Iteration Focus Process, where the demonstration context is part of the setting in which Design & Development, and Evaluation is carried out in focused iterations, the Design Iteration Focus. The process is a modification of the process described by Peffers et al. (2006).

Our Design Iteration Focus Process places the *Design & development* and the *Evaluation* phase, along with an emphasis on the frequent iterations between these, within the demonstration context. This differs from the original model of the DSRP, where *Demonstration* is a separate phase positioned between the other two, and where the phases are carried out sequentially, cf. figure 1. Section 4.2.1 describes the demonstration context in this study, and section 4.2.2 describes the design and evaluation process with related discussions.

4.2.1. Demonstration Context

The demonstration context of this study consisted of, and was defined by: [1] The physical setting along with the daily workflow of our research and workflow of the company employees, and [2] the meeting structure and other demonstration activities used to continuously evaluate the design.

Setting and workflow: During the four month period in which this study took place, we were situated on-site at the company main office together with the employees. This gave us the opportunity for continuous evaluation of the design, as feedback was always accessible. Partly, feedback was gathered during many informal meetings with the employees, typically during normal breaks from work, or by asking for quick feedback as the need arose.

Demonstration activities: A meeting structure where formal demonstration and evaluation took place was agreed upon with the company. The meetings were our primary source of structured evaluation and a part of the research workflow. The intention was to have meetings once a week with one senior developer and, when needed, a project manager as well. However, the dynamic workflow of our own research, as well as the company's own dynamic daily work, required this structure to be flexible. Moreover, the state of the design also influenced the meetings, both in schedule and content. During the initial design and evaluation iterations, frequent meetings were held to define business needs and core requirements and quality attributes. As these issues became better understood, the objectives of a solution could be formulated. And as the definitions of these objectives settled, the design iterations and meetings became more focused on design and evaluation of specific issues in order to meet the objectives. Towards the later stages of the study, we held demonstration sessions with all five employed developers and a project manager. The sessions were semi structured and conducted with one participant at a time. Each session started with us presenting the framework and then continued with us interviewing the participant. During the sessions, the participants were free to try out the framework for themselves. Interview notes, meeting notes, and field notes from all these occasions were kept and analyzed as part of continuous evaluation.

In the sections below, we refer to the software system currently in place at the company, i.e. the system to be tested, as the *current system*. The prototype test automation framework that we have developed as part of this study is referred to as the *framework*.

4.2.2. Design and Evaluation Iteration

Due to the exploratory nature of our research process we anticipated frequent changes to the requirements to be implemented in the design (Sommerville, 2007). This contributed to our decision to use an agile process for development activities because, as McConnell (2009) also acknowledges, choosing an agile process maximizes the ability to respond to changes. The development process followed a structure of sprints, which was useful for planning and prioritization purposes. However, the actual design and evaluation iterations were not necessarily limited or scheduled according to the sprints. Rather, the various design issues were iterated when appropriate, some issues more often than others depending on evaluation results during the iterations. In practice, there was no reason for us to separate the design and evaluation iterations as part of the research process, and the design and evaluation iterations inherent in our agile development process. During this study, these two iteration concepts served the same purpose, and are both a part of the Design Iteration Focus.

Recognizing the importance of understanding the business needs of the studied company, in order to reduce the risk of not capturing all relevant quality attributes and requirements (Bass et al., 2003; Clements and Bass, 2010), we used the initial design iterations to define the problem at hand. This also reduced the risk of implementing a design based on faulty prerequisites (McConnell, 2009) and helped defining the quality attributes and requirements that served as a basis for our objectives of a solution. These objectives were refined during subsequent iterations and discussed whenever needed. However, as the study progressed, the objectives became increasingly stable and iterations were instead more focused on specific design features.

The specific design features of the implementation, along with related design and development tasks, were derived from use cases based on the objectives of a solution. These use cases were described from a users perspective, where the intended primary user was a developer writing tests for the current system. The bulk of the time spent during this study, was spent on these key design features, their definition, design, and evolution through the Design Iteration Focus Process.

During the design iterations, we turned to literature on various subjects to find support for our interpretation of the business problem, how the identified quality attributes and requirements could be implemented, and for general best design practices. At times, decisions were made by us in the role of developers, rather than researchers, in order to keep a momentum in daily work and drive the design. This means that all detailed design choices made, are not necessarily found specifically in the literature, but rather sometimes based on our own knowledge and experience of software design.

As described above, we held demonstration sessions during the later stages of the study. At this point, we considered the key design features of the framework to be mature enough for the framework to be used and evaluated as one tool (rather than as individual components without proper interfaces and abstraction levels). The purpose of these sessions was to gather feedback on the framework as a tool, where a potential user (i.e. a developer at the company) could use it as intended by the design. This provided us the opportunity to evaluate quality attributes such as usability and extendibility, along with general impressions of the design through the eyes of a user. During the sessions we briefly presented the architecture and design, together with concepts of implemented features, and intended usage. After the presentation, the participant had the opportunity to use the framework and become a bit familiarized with it, and ask questions about the design and intended usage. Following this, we asked the participant a set of open-ended questions on perceived usefulness, implications, applicability in other areas, and how the participant believed the framework could affect everyday activities.

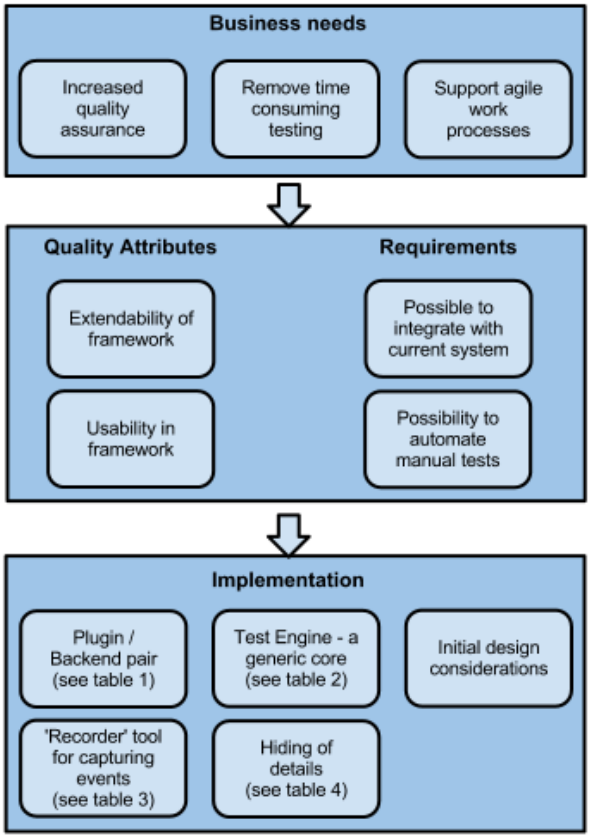

Figure 3: Relation of business needs, quality attributes and requirements, and the implemented solution.

Figure 3 shows the business needs together with the quality attributes and requirements forming the objectives of our solution, and the implemented solution. The design process of the implemented solution is, along with important design decisions and the evolution of the design

over iterations, described throughout the rest of this section. First, the initial design decisions related to the fundamental choices are described. These choices were not iterated in the same manner as the other design features, but are still important, and therefore they are presented together as a separate 'feature'. Following this, the key design features that were iterated through the Design Iteration Focus Process are described. Towards the end of the section, the final evaluation of the framework as a tool is presented.

Initial Design Considerations

The objectives of our solution to the problem identified in section 2 of course played an important role during all design iterations. Initially, though, design considerations were mostly influenced by two of the objectives; *integration with current system* and *usability* respectively.

Early evaluation showed that if it is possible to integrate the framework with the current system's architecture, it is more likely to be useful. Thus, even though the company put no constraints on particular technology to be used, we chose to align the framework architecture with the overall architecture of the current system. This alignment allowed the framework to fit as a part of a combined architecture without changes to the current system. The framework attaches to the current system's integration layer as an integral part of the same software, rather than having its own integration layer (see figure 4). The framework is, however, a separate system running in its own process. The inter-process communication (IPC) between the integration layer and the framework is realized by D-Bus[1] calls, in the same way as the integration layer communicates with the current system over D-Bus. Subsequently, as these design decisions together enabled integration they also contributed to fulfilling the objectives described in section 3.

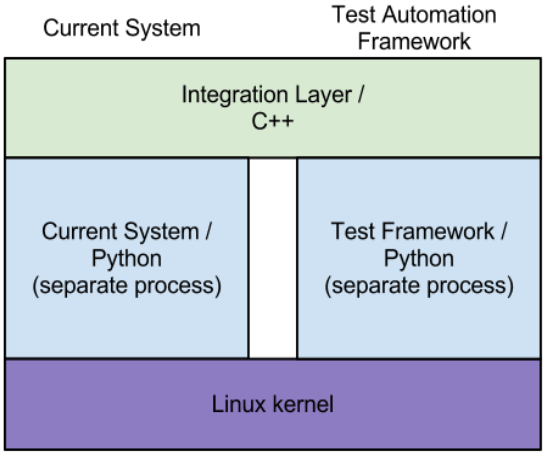

Figure 4: The alignment of technology of the current system and the test automation framework.

[1] For a description of the D-Bus inter-process communication system, see Love (2005).

Moreover, for the framework to be useful, it should be deployable on the same platforms as the current system and should also introduce as few new dependencies and constraints as possible. In addition, to avoid creating a negative impact on productivity, any writing of code as part of the normal usage of the framework should be in a familiar language (Finsterwalder, 2001; Boehm et al., cited in McConnell, 2009, p. 62), i.e. the test code for the framework should be written in the same language as the current tests. Likewise, any future extensions to the framework should be easily added by the company's developers. For these reasons, we chose to implement the framework in the programming languages used in the current system. This meant we implemented the framework in Python and Qt/C++, running on a Linux kernel, ultimately increasing usability and, thus, kept the usability objective of section 3 in consideration.

Plugins and Back-ends

With the current system relying heavily on a multitude of hardware devices and wireless services, our framework was required to be highly modular so that simulation of existing and new technology could be easily added and managed. As a response to this requirement, expressed as extendibility within the objectives of a solution in section 3, the proposed architecture for our system featured, already at an early stage, a plugin-based design (see figure 5). In the figure, a plugin refers to an item with a set of properties that defines a state. Examples of such items include Bluetooth adapters or remote Bluetooth devices, audio streaming services, social networking services, and storage media such as USB flash drives. The purpose of the back-end, then, is to expose the state of the plugin.

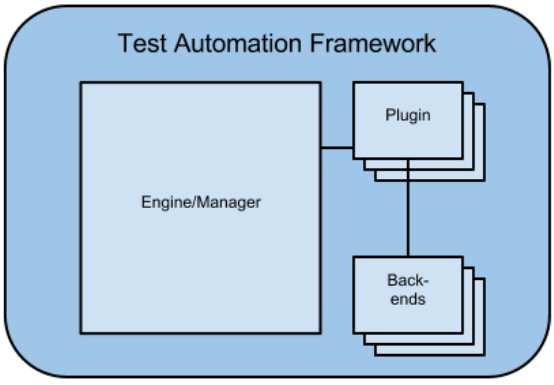

Figure 5: Internal architecture of the test automation framework

Because the current system interfaces to a wide variety of hardware and wireless services through D-Bus, we chose to implement our back-ends as D-Bus services as well. The user of the framework is however not restricted to using this particular service. If the back-ends are cleverly built they may very well be re-used, and if effort is put into creating a large pool of back-ends, quite complex simulations could be

made possible. The evolution of the plugin and back-end components is presented in table 1.

During design, the extendibility and usability quality attributes were constantly kept in consideration. Ultimately, this resulted in easily implemented, highly extendable components with very few constraints. In fact, the only constraints on the two components were that they had to be Python modules, and that the plugin has to implement a specific set of methods for it to be recognized by the framework as a plugin. In effect, the level of complexity within the plugin and back-end modules scales along with the developer's needs. Potentially desirable features of the plugin and back-end, e.g. plugin inheritance or parallel execution of back-ends, could thus be easily added. As mentioned by McConnell (2009), leaving such doors open preserves modularity, and in our case this in turn promotes a higher degree of extendibility.

By using the recorder tool (see table 3), the state of real devices or services could be captured and then loaded into the plugin, thus increasing the level of authenticity of the plugin's properties while at the same time reducing the demand for specific knowledge from the user. Again, the implementation of the plugin states is not restricted to the use of the properties captured by the recorder tool; the recorder tool merely facilitates the capturing of these. Moreover, the validity of our simulated plugins and back-ends was confirmed through both code coverage reports and by analysis of the current system's response. The tasks of verification and validation were then continuously carried out during all consecutive design iterations to ensure that functionality had not been skewed.

Plugins and back-ends	Design	Evaluation
Iteration 1 Rough design	A plugin-based architecture is proposed. Plugins represent simulated devices, e.g. USB flash drives, and back-ends represent e.g. simulated drivers or D-Bus interfaces. A test engine (see table 2), with the responsibility to connect the two components is suggested.	The proposed architecture is considered to be in alignment with what the company had envisioned. Developers at the company are curious to see a first prototype.
Iteration 2 First prototype	A simulation of the D-Bus UDisks[2] interface is implemented as a back-end service and exposed on the D-Bus *session bus* instead of the *system bus* as the real interface is. The plugin holds a limited set of properties possible to expose through the simulated back-end service. All communication between the two components is done via the test engine.	The response to the simulated back-end from the current system is similar to the response to the real back-end. The approach is considered to be working as expected but back-end services needs refinement.
Iteration 3 Refining the design of the backend service	The implementation of the recorder tool (see table 3) facilitates further development of the back-end service. By using the tool, expected back-end behavior is identified and can then be implemented. The set of properties of a block device are recorded from UDisks and used as properties in the plugin device.	The current system is tested with simple scripts that, via the engine initializes the back-end service and the plugin, simulates the insertion and removal of a simulated USB flash drive. The current system responds in the same way as with real hardware and code coverage reports from the related modules are generated, showing that the same code is executed in both situations. Design needs to be tested in other contexts than UDisks.
Iteration 4 Simulating BlueZ	Bluetooth adapter and device plugins as well as a simulated back-end of the D-Bus interface to the BlueZ[3] Bluetooth stack are implemented.	The kinship relationships between Bluetooth adapter, devices and services demand that plugins and back-ends have more knowledge of each other.
Iteration 5 Plugin and back-end knowledge	The test engine's responsibility to route traffic between plugin and back-end is removed. Plugin and back-end now communicates directly with each other, and the functionality to add children to a back-end or plugin is implemented.	Responsibilities of the components are ambiguous. There is a need to clearly define these.
Iteration 6 Separating responsibilities	A major redesign of the components is done. The plugin is now the only component with knowledge of the back-end.	Evaluated as part of a complete tool, see table 5.

Table 1: The evolution of the plugin and back-end components.

[2] The UDisks interface is described in Freedesktop.org (2012)

[3] For a description of the BlueZ D-Bus interface, see Holtmann (2006).

Test Engine	Design	Evaluation
Iteration 1 Rough design	A test engine to drive and manage the tests is conceived. The test engine only manages the initialization and removal of plugin/back-end pairs.	There is a need to be able to specify timing in the test code, in order to write more useful tests, especially for tests other than 'happy path' scenarios. Specified timing gives opportunity to drive tests in a way more similar to real events.
Iteration 2 Events specified in test code	The manager responsibilities in the test engine are put in a separate module. The test engine now understands timing and the simulation of events are decoupled from the manager part of the framework.	The new test engine design allows more control of the test to be put in the test code and XML. There is still partial plugin and back-end logic implemented in the test engine. For example, when test engine removes a plugin, it also tells the corresponding back-end to take appropriate actions.
Iteration 3 Test engine becomes generic	The test engine only tells plugins to change state, and adds and removes plugin/back-end pairs by interaction with the manager. No logic of what this means to the plugin/back-end pairs is in the test engine or manager.	Evaluated as part of a complete tool, see table 5.

Table 2: Evolution of the test engine.

Test Engine

As already mentioned, for the framework to be useful as a tool supporting the agile work process, it needed to be designed with future extension in mind (see section *Hiding of detail* for more discussion on this). This extendibility objective has been a major driver in the evolution of the engine and its related modules. Considering the functional requirement of test automation, the test engine is the core part of the framework that allows tests to be run as defined by an external source, allowing for automation. The generic nature of the engine design means that this external source could be implemented as an XML file, but the module parsing the XML could just as well be made to parse any other data format. Table 2 describes the evolution of the test engine through the Design Iteration Focus Process.

To design with extendibility in mind meant that logic related to the behavior of the plugins and back-ends, and all things with a design specific for a certain type of test, needed to be removed from the core parts of the framework, i.e. the engine and manager. Otherwise, changes in the design of above mentioned parts, i.e. extension of the framework, would risk having ripple effects through the whole framework. The separation of concerns by decoupling the engine from the plugins and back-ends, allows the user writing tests to ignore internal details of the engine, thus playing a part in providing usability in the framework.

Along with the aforementioned plugin component, the test engine also provides the means necessary to model hardware devices and wireless services as state machines. We want to emphasize though, that this depends on how the user decides to implement the plugin and the external source driving the test engine. The use of state machines in testing frameworks has been studied in other contexts where the product to be tested also relies heavily on user interaction. For instance, Memon et al. (2003) describe a test framework for GUI testing that utilizes concepts similar to state machines in order to identify the allowed state transitions of a GUI-based application.

Recorder tool

The plugin properties and the complex relationships between these could potentially demand much knowledge from the user writing tests. We experienced already at an early stage the difficulties of knowing when a property should be changed and what behavior this should trigger. With the desired usability quality attribute in mind, we complemented our framework with a recorder tool able to record the states of an actual device or service, so that the user could easily record the sometimes quite complex set of properties and relations between these.

The role of the recorder tool was further incorporated into the testing framework, though still as a standalone tool, and ultimately it supported the possibilities to record an entire script, including temporal aspects of e.g. mounting/unmounting a block device partition or pairing/unpairing with a Bluetooth device, for both the UDisks daemon and the BlueZ daemon for the Linux Bluetooth stack. The outputted XML script could be used as input to the test engine and as such, drive the sequence of state transitions forward.

The concept of capturing user interaction is not new. Indeed, Finsterwalder (2001) describes that, within the field of GUI testing, it has been common practice to record the often complex interaction between user and software in scripts. These scripts are then used to simulate the interaction during testing. During the design of the recorder tool, by drawing on the approach from GUI testing, we were able to hide the complexity of user interaction, triggered responses, and plugin properties. This, we claim, contributed to the ease of use of the recorder tool, and thus, in turn, contributes to the objectives of a solution. Additionally, the recorder tool not only proved useful to capture the data described above, but also for exploratory tasks where the user would only be interested in monitoring and learning more about e.g. the emitted signals and triggered responses of a specific D-Bus service.

Recorder tool	Design	Evaluation
Iteration 1 Hardcoded scripts	Scripts that manipulate individual, hardcoded, properties of a plugin drive the framework.	The design is inflexible and requires the user to have much knowledge of individual device properties. During design and implementation of the block device plugin and back-ends, a need to easily gain knowledge of emitted signals and triggered responses is identified.
Iteration 2 Signal listener	A simple D-Bus listener is implemented. It monitors the signals emitted and responses to method calls from the UDisks D-Bus daemon.	The listener provides the user with readable and easily understood information on the event sequences. Possibility to record the actual states of the block devices is desired.
Iteration 3 Recording of states	The listener is improved and now enables the recording of device properties to file. The files are used as input for the plugin modules. The listener is now referred to as the recorder.	The recorder's output is lacking temporal aspects of state transitions as well as kinship and causal relationships.
Iteration 4 Capturing temporal aspects and relationships	The design is improved and an XML script defining the temporal, kinship and causal relationships of the device's states is now possible to generate.	The XML script appears to be well suited as input to the test engine. From test cases it is possible to control when a transition between two states should occur. However, a need for defining breakpoints within the XML script is identified.
Iteration 5 Breakpoints	The support for hardcoded breakpoints in XML script is implemented.	The breakpoints serve their purpose but a discussion on whether it should be possible to define them already during the run-time of the recorder tool takes place. It is decided that, by hardcoding breakpoints, the user will retain control over when they should occur, without introducing more complexity in the recorder usage. The recorder tool design needs to be tested for other D-Bus interfaces, e.g. the BlueZ D-Bus daemon.
Iteration 6 A recorder for the BlueZ D-Bus daemon	A recorder tool that listens to signals and triggered responses on the BlueZ D-Bus interface is implemented.	There is much redundant source code and the tools for UDisks and BlueZ should share whatever code they have in common.
Iteration 7 Reducing amount of redundant source code	A component containing source code common to both tools is implemented.	Evaluated as part of a complete tool, see table 5.

Table 3: Evolution of the recorder tool.

Hiding of Details

The evolution of abstractions, by hiding of details and separating concerns throughout the design of key features, has helped reach the quality attributes of usability and extendibility. Table 4 exemplifies this by highlighting evolution of abstraction in the recorder tool and test engine.

At times, ideas about potential future application and usage of the framework, or parts of the framework, were discussed during the design and evaluation iterations. While the ideas were largely vague and not possible to fully explore during our research, we considered extendibility to be of importance in order to allow for future extension, a consideration that is supported by Cervantes (2009). In order to facilitate future possibilities, while not committing to "design ahead" (McConnell, 2009), we constantly refactored existing code, keeping abstractions clear, to allow for easier adaptation later (Fowler et al., 1999; Beck, 2000).

In striving for usability, evaluation showed there was a need to hide details from the user. While searching for a good balance between detailed control and usability, the external API of the framework, the format of the XML, among other things, went through frequent changes. This in turn highlighted ripple effects, caused by an initially convoluted design of the core framework modules, while we modified the design. In essence, while working with the functionality, we got the chance to evaluate the extendibility and modifiability of the framework. This parallel design and evaluation is also reflected in table 4.

Evaluation of framework as a tool

During the demonstration sessions, general feedback on the framework as a complete package was gathered, along with opinions about possibilities, problems, and what else that came into the mind of the participant. With the intention of probing further into certain areas of interest, we had a set of questions asked during all the sessions. The feedback related to the key design features is summarized in table 5.

The high degree of integrability, the possibility to capture actual states of devices and services, and the possibility for the developer to scale plugin and back-end complexity according to need, enables implementation of highly authentic simulations. It is important, though, to emphasize that the level of authenticity, as well as the level

Hiding of details	Design	Evaluation
Iteration 1 *Usability*: Individual manipulation of properties *Extendibility*: Monolithic manager module	*Usability*: Manipulating individual properties trigger plugin state changes. *Extendibility*: The module managing the tests internally to the framework also contains engine aspects of functionality, e.g. deciding when state changes should be made in plugins.	*Usability*: Requires much domain specific knowledge from the user. *Extendibility*: Highly coupled design gives ripple effects when even minor changes has to be made.
Iteration 2 *Usability*: Introducing the recorder *Extendibility*: Separation of concerns	*Usability*: Recorder reduces the need for knowledge of specific properties by capturing entire states. *Extendibility*: The engine and manager becomes different modules.	*Usability*: User still needs to know how timing and parent and child plugins relates. *Extendibility*: A clear separation of concerns makes changes to functionality and refactoring easier and less time consuming. Still logic related to plugins and back-ends in engine.
Iteration 3 *Usability*: Introducing XML defined behavior *Extendibility*: Reducing coupling	*Usability*: Temporal, causal and kinship relationships are incorporated into the XML. The XML now contains all necessary details, thus removing the need for detailed domain specific knowledge when running tests. *Extendibility*: Plugin and back-end communication is done directly without the test engine as a mediator. No logic related to plugin or back-ends left in engine or manager.	*Usability*: Evaluated as part of a complete tool, see table 5. *Extendibility*: Evaluated as part of a complete tool, see table 5.

Table 4: Evolution of abstractions throughout the framework design, exemplified as usability and extendibility.

of automation, depends on how the framework is used, and of course it is not necessarily entailed by the use of it alone. Conversely, and as was frequently noted during the demonstration sessions, the framework could be used to expose the current system to rare or unnatural behavior, i.e. "corner cases", for instance for stress-testing purposes.

Hiding details from the user with the intention of increasing usability gave rise to some tension between detailed control and a desire to reduce the amount of knowledge of implementation details required. For example, the hiding of D-Bus properties in state files which do not allow the user to manipulate all details separately, was also perceived as removing control from the user. The responses to this were mixed during the demonstration sessions; some requested more control and some saw the advantages of not having to learn all the specifics of a device or service to simulate. Similarly, hiding details which meant that previously direct method calls in the test code, were substituted with generic calls to the framework, seemed off-putting to some developers. The limited scope of this study leaves it difficult to determine if this perceived loss of control is due to lack of familiarity with the tool or a perception that will remain. In any case, keeping all previous control and level of detail at the hands of the test writer, would have rendered many of the features that enables automation impossible.

Key design feature	Evaluation
Plugins and back-ends	The responsibilities of the modules are considered well-defined. The design is modular, user-friendly and extendable. The design enables simulation of devices and services, not only for testing purposes, but also for development, e.g. when access to actual hardware is limited.
Test engine	Not having to modify the core components when extending functionality of the framework increases extendibility. In addition, the test engine's ability to drive pre-recorded scripts provides opportunities in debugging, testing, and presentation contexts.
Recorder tool	Future suggestions include the incorporation of the two recorders into one tool. Possibilities not considered during the design iterations are found, e.g. using the recorder when reporting bugs, so that system states and events can be reproduced. Making the recorder more general could lead to increased usefulness.
Hiding of details	*Usability*: Interfaces and responsibilities are largely evaluated as clear and unambiguous. Some developers do not like the perceived loss of control in cases where an otherwise direct method call is now substituted by a more generic call to the test engine. Hiding some test details in XML creates concerns about maintainability and readability. *Extendibility*: The generic nature of the engine gives less ripple effect when extending simulation functionality in plugins/back-ends, or when implementing new plugin/back-ends. The idea of extending the framework in this area is perceived as easy.

Table 5: Evaluation of the key design features in their final prototype state.

A few themes in the concerns of the company have been emerging from the continuous evaluation during this study. One concern is that a tool like this kind of framework has to introduce as little work overhead as possible. Otherwise it will not be perceived as helpful. An example of this is the implicit requirement of the framework to be easily integrated with the current system with as few additional dependencies as possible. If the framework is perceived by management and developers as consuming more time than it helps gain, there is a risk it will not even be used. This also creates a tension between usefulness and overhead. For example, if the framework opens up new possibilities, it will as a consequence require work to exploit, since the opportunity is new. One way to try and mitigate this tension, could be to let things be only as complicated as needed. The minimal requirements the framework places on plugins and other modules typically modified to extend the frameworks utility, contributes to a complexity scalability in the framework design. This scalability implies that a quick extension of the framework, e.g. for prototyping tasks, can be produced with minor complexity involved, while a full simulation of a device could become just as complex as needed.

During the demonstration sessions, a majority of the participants agreed that the tool could facilitate regression testing and automated nightly tests. The participants also saw how this in turn could impact their current work process positively, e.g. by provided support for test driven approaches to design and development. In addition, other usages of the framework were discussed and several of the developers, as well as the project manager, envisioned how the tool could be complemented with a UI from which the developer could trigger state changes, e.g. the mounting of a USB flash drive partition, and in turn examine the triggered system response. Similarly, a UI like that could be used during presentations of the product, e.g. to show how the system handles certain resources. Moreover, the role of the recorder tool was discussed and several of the employees at the company visualized how it could be further developed into a debugging tool that also monitors the current system's response. This was regarded as something potentially useful in situations where customers need to send bug reports to the company.

4.2.3. Design and Evaluation Reflection

We have found that a complex design process is essential to address the problems identified in section 2, the business needs found during our Design Iteration Focus Process, and the quality attributes and requirements derived from these. This complexity is necessary because the literature, on e.g. agile, merely helps define the problems and goals, while a practical implementation of a tool is sometimes required to actually enable people to work as intended. Designing such tool while considering all relevant issues, demands a design process that allows design decisions to be iteratively refined, and that incorporates relevant feedback and analysis. To address these concerns, we followed our adapted version of the DSRP (Peffers et al., 2006). Still, even when the business needs and objectives of a solution have been defined, the actual process of design is left to deal with, a process which is full of details that need to be discovered, and decisions to be made. While developing and designing, we have had to work extensively as developers within our own research process, in order to drive the design forward based on the evaluations made throughout the Design Iteration Focus Process.

The continuous evaluation throughout our research process has led to the key design features that, combined, successfully meet the objectives of a solution described in section 3. The initial design considerations together with the evolution of the plugins and back-ends, recorder tool, test engine, and the hiding of detail feature, all create a way to meet the quality attributes of *extendibility* and *usability*, as well as the requirements of *integration* and *automation*. Emerging from this continuous evaluation is the tension between desired functionality and the opportunities it would create, and the time needed to exploit the new opportunities. As previously mentioned, the design of the framework allows things to get as complex as they need to be, but does not add complexity automatically. In this sense, the framework in itself does not consume more time, but rather, the company can choose when there is a need to add complexity, and consequently spend more time.

By keeping the design modular and extendable, the usage of the framework for other purposes, e.g. the simulation tool brought up during the demonstration sessions, could easily be supported. It is interesting to note how our process may have contributed to these quite colorful insights from the employees of the company, i.e. how the continuous demonstration, design and evaluation has given them time to reflect and refine their own visions of the framework, and ultimately present these when opportunity was given during the demonstration sessions.

5. CONCLUSION

Within this paper we have explored how business needs related to test automation within agile work processes can be addressed through careful design of a test automation framework. The problems giving rise to the business needs, however, turned out to be rather complex and demanded a suitable research process that would allow these to be addressed properly. Much of this paper's emphasis thus rests upon the design and evaluation process, and we have given a comprehensive description of how we planned and carried out the design science research focused on the development of a test automation framework. We consider this detailed description to be a part of our contribution.

Our modified version of the DSRP (Peffers et al., 2006), which we have referred to as the Design Iteration Focus Process, enabled us to conduct the research in a setting where our prototype was continuously demonstrated to, and evaluated by, its potential users. This, in turn, enabled

shorter and more frequent design-and-evaluate cycles throughout the study, and ultimately resulted in a test automation framework that fulfilled the quality attributes and requirements brought forth by the employees of the company, while still conforming to best-practices described in the literature. The evaluation of the framework's potential impact on the work process at the company could clearly be helpful to other companies finding themselves in similar situations.

During the study, the continuous evaluation has also led to new ideas and insights at the studied company, as to what other possibilities there might be, compared to the original purpose of the framework. Our research process, in which both the studied company's concerns as well as relevant literature have been considered, has been efficient in creating a design well suited to address the defined problems. However, it becomes apparent that identifying a need for a tool does not mean that the development and introduction of such a tool is obvious. An organization with a need for tool support must also be willing to spend time on exploiting the opportunities that arise. Still, a design well suited for the purpose, i.e. a design that has addressed all necessary concerns, seems more likely to be used.

Suggested entry points for future work include an examination of the framework's long term impact on the agile work process and testing procedures at the company, and an assessment of how usable the employees at the company considered the framework to be after working with it for a period of time. In addition, the extendibility quality attribute could in a separate study be evaluated by attempting to extend functionality, for instance by using the framework for other purposes than those originally intended, e.g. the implementation and design of a demonstration UI.

6. REFERENCES

Bass, L., Clements, P., Kazman, R. (2003). *Software Architecture in Practice* 2:nd Edition. Addison-Wesley, Boston.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change.* Addison-Wesley, Reading.

Berner, S., Weber, R. and Keller, R.K. (2005). Observations and Lessons Learned from Automated Testing. In: *Proceedings of the ICSE'05*, May 15 - 21, 2005, pp. 571-579. St. Louis, Missouri, USA.

Cervantes, A. (2009). Exploring the Use of a Test Automation Framework. In: *Proceedings of the 2009 IEEE Aerospace Conference*, March 7 – 14, 2009, pp. 1-9. Big Sky, MT, USA.

Clements, P. and Bass, L. (2010). The Business Goals Viewpoint. *IEEE Software*, 27(6), pp. 38-45.

Conboy, K., Coyle, S., Wang, X. and Pikkarainen, M. (2011). People Over Processes: Key Challenges in Agile Development. *IEEE Software*, 28(4), pp. 48-57.

Coram, M. and Bohner, S. (2005). The Impact of Agile Methods on Software Project Management. In: *Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)*, 4 - 7 April, 2005, pp. 363-370.

Finsterwalder, M. (2001). Automating Acceptance Tests for GUI Applications in an Extreme Programming Environment. In: *Proceedings of the 2nd International Conference on eXtreme Programming and Flexible Processes in Software Engineering*, pp. 114-117. Addison-Wesley: Boston MA.

Fowler, M., Beck, K. and Brant, J. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

Fowler, M. and Highsmith, J. (2001). The Agile Manifesto. *Software Development,* 9(1), pp. 28-32.

Freedesktop.org (2012). *Software/udisks*. [online] Available at: <http://www.freedesktop.org/wiki/Software/udisks> [Accessed May 15 2012].

Gregor, S. (2006). The Nature of Theory in Information Systems. *MIS Quarterly*, 30(3), pp. 611-642.

Hevner, A.R., March, S.T., Park, J. and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly,* 28(1), pp. 75-105.

Holtmann, M. (2006). Playing BlueZ on the D-Bus. In *Proceedings of the Linux Symposium Volume One*, 19 - 22 July, 2006, pp. 421-426. Ottawa, Ontario, Canada.

Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L. and Zelkowitz, M. (2002). Empirical Findings in Agile Methods. In: *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe,* 4 - 7 August, 2002, pp. 197-207. Chicago, IL, USA.

Love, R. (2005). *Get on the D-BUS*. [online] Available at: <http://www.linuxjournal.com/article/7744> [Accessed May 5 2012].

March, S.T., and Smith, G.F. Design and Natural Science Research on Information Technology. *Decision Support Systems,* 15(4), pp. 251-266.

Markus, M.L., Majchrzak, A. and Gasser, L. (2002). A Design Theory that Support Emergent Knowledge Processes. *MIS Quarterly*, 26(3), pp. 179-212.

Martin, R., C. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice-Hall.

McConnell, S. (2009). *Code Complete: A Practical Handbook of Software Construction* 2nd ed. Microsoft Press.

Memon, A., Banerjee, I., Hashmi, N. and Nagarajan, A. (2003). DART: A Framework for Regression Testing "Nightly/daily Builds" of GUI Applications. In: *Proceedings of the International Conference on Software Maintenance (ICSM'03),* September 22 – 26, 2003, pp. 410-419. Amsterdam, The Netherlands.

Moe, N.B., Dingsøyr, T. and Dybå, T. (2009). Overcoming Barriers to Self-Management in Software Teams. *IEEE Software*, 26(6), pp. 20-26.

Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A. and Succi, G. A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team.

Nunamaker, J.F., Chen, M. and Purdin, T.D.M. (1991). Systems Development in Information Systems Research. *Journal of Management Information Systems,* 7(3), pp. 89-106.

Peffers, K., Tuunanen, T., Gengler, C.E., Ross, M., Hui, W., Virtanen, V.,and Bragge, J. (2006). The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. *DESRIST*, 24(3), pp. 83-106.

Puleio, M. (2006). How Not to do Agile Testing. In: *Proceedings of AGILE 2006 Conference*, July 23 - 28, 2006, pp. 305-314. Minneapolis, MN, USA.

Shaye, S.D. (2008). Transitioning a Team to Agile Test Methods. In: *AGILE 2008 Conference,* pp. 470-477. IBM, Armonk, NY, USA.

Sumrell, M. (2007). From Waterfall to Agile - How does a QA Team Transition? In: *AGILE 2007*, pp. 291-295, Misys Healthcare Syst., Raleigh, NC, USA.

Sommerville, I. (2007). *Software Engineering* 8:th Edition. Addison-Wesley, London

Winter, R. (2008). Design science research in Europe. *European Journal of Information Systems*. 2008(17), pp. 470-475.