Automatic Status Updates in Exacts Global Development Process

Version of November 30, 2009

Menno Valkema

Automatic Status Updates in Exacts Global Development Process

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Menno Valkema
born in Leiden, The Netherlands

Delft University of Technology
Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

And it all comes together.
Exact Asia Development Center
Petronas Twin Towers, Tower 2
Level 34/47 KLCC
Kuala Lumpur, Malaysia
www.exact.com

Automatic Status Updates in Exacts Global Development Process

Author: Menno Valkema
Student id: 1178326
Email: m.valkema@student.tudelft.nl

Abstract

Due to a competitive business environment and demanding customers, many companies turn to globally distributed software development and agile methodologies [27]. Globally distributed software engineering can bring great advantages in reducing costs, reducing time to market, and give access to a larger pool of skilled resources [19, 20]. Agile methodologies acknowledge many of the development challenges companies face, such as changing customers requirements and the necessity to have frequent releases [10]. Introducing agile by itself can be a challenge for a big organization [14], especially when a blend is made between agile and distributed development, since these have some contradictory features [46]. The global company Exact faces some challenges when introducing agility into its business processes. This research has as a goal to introduce agile into the globally distributed development process of Exact. It is hard to introduce agile as a whole, therefore we address the biggest challenges faced within Exacts development process first [9]. One of the main challenges faced within Exact is communication between the globally dispersed product management and product development teams. An agile practice addressing this communication challenge is to use automatic status updates and product updates generated by an automated build process [27]. This research will explore what features of continuous integration are useful for Exact and the created system will be evaluated. The ideas presented in this thesis have not yet been tested and evaluated on a large scale, due to time restrictions. However a prototype has been tested on a small scale for one project, and the initial responses from product development and product management were positive. More important, a shift in perception occurred at product development to support a more open development process.

Thesis Committee:

Chair:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Dr. Ir. D.M. van Solingen, Faculty EEMCS, TU Delft
Company supervisor:	Mr. T. Hurkmans, Exact
Committee Member:	Ir. B.R. Sodoyer, Faculty EEMCS, TU Delft

Preface

Many people have helped me to make this project what it is now, here an incomplete list: Christian Visser, thanks for our weekly meetings on our assignment while I was in Kuala Lumpur. Ken Leong Lee, thanks for supervising my project while I was working in Kuala Lumpur: thinking along with my ideas, adding your own ideas, and giving feedback greatly helped to get the project where it is now. Toine Hurkmans, thanks for your support by pointing me in the right directions, getting me in touch with the right people when necessary. Michiel van Rooijen, thanks for sharing your views as a product manager on the early prototypes of our system. All people I interviewed within Exact, thanks for providing your view on the development process, sharing your ideas on how to make things even better, and being open and honest about the challenges you face in your job. Time was too short to address everything you mentioned, though I hope the result of this project helps you in your daily tasks, and helps Exact to make even better software. All people within ADC, thanks for making my stay in Malaysia a time to remember. Cheng Meng Hoh, thanks for sharing the picture of the Petronas towers to put on my thesis cover: it is more beautiful than I could ever make it. Last but not least: Rini van Solingen, thanks for all your help and support during my research.

Menno Valkema
Delft, the Netherlands
November 30, 2009

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

This chapter introduces the graduation project. The context, the goal, and the research questions for the project; finally an overview of the report is given.

1.1 Context

This section describes the context of the project. The project is part of the TU Delft Computer Science Masters Program and its focus is the area of globally distributed software engineering. The project is part of a duo assignment executed together with another TU Delft student working in Exact, the assignment is partially executed in Malaysia, and both TU Delft and Exact have put some constrains upon the assignment.

1.1.1 Exact

Exact is a software company developing software for small- and mid- sized businesses. Requirements for the software are defined by product management in the headquarters in Delft, and the software is developed by product development in Kuala Lumpur, Malaysia. Exact has a collaboration program with the TU Delft: Exact finances research of TU Delft MSc. and PhD. students to do research on challenges Exact faces. This project is part of this collaboration program, investigating challenges Exact faces in its globally distributed software development process. For more information on Exact and the development process see Chapter 2 and Chapter 3.

1.1.2 Duo Assignment

The thesis is part of a duo assignment for two MSc. students, doing research on challenges in Exact's globally distributed development process. One student approaches problems in the development process from the product management perspective, the other student focuses on challenges from a product development perspective. For results on challenges faced from a product management perspective, see [60]. In this thesis is investigated what challenges in the global development process are faced by product development in Kuala Lumpur.

1.1.3 Abroad

To see both perspectives of product development and product management, the project is partially executed in Delft and for the biggest part in Kuala Lumpur. Since this project addresses challenges in the globally distributed development process from the perspective of product development in Kuala Lumpur, it is natural to go there to see what problems are there. However, to get good insight in the development process as a whole, it is also important to see the challenges faced by the major counterpart of product development: product management in Delft. The project starts out with 2 months of preparation and initial research on the development process in Delft. These 2 months are followed by 6 months in Kuala Lumpur. In Kuala Lumpur the research on the development process is continued, followed by a problem definition, creation of a set of solution proposals for the challenges faced, and a pilot is executed with one of the proposed solutions. The project is concluded with another month in Delft to process results and to finish the report. Thus, the thesis project consists of 9 months in total: 3 months in Delft and 6 months in Kuala Lumpur.

1.1.4 Constraints

Both TU Delft and Exact put constraints on the project.

Exact The project is executed at Exact to resolve challenges faced by Exact. Therefore, the project should result in an improvement or a recommendation on a possible improvement on the development process iterating between Delft and Kuala Lumpur. Since Exact faces challenges (see Chapter 2) which are recognized and addressed by the agile community, Exact attempts to include agile practices in its development process. For that reason, the recommendations made in this report should be aimed at making the development process in Exact more agile. Furthermore, Exact is a big organization in which big changes can be hard to introduce. So, a goal is that all proposals have minimum impact with maximum result, making it easy to introduce the change; still having results.

TU Delft While the project is executed at Exact, the project is a MSc. graduation project for the TU Delft where certain constraints apply. First of all, all solutions and changes implemented should be related to the Computer Science MSc. curriculum. Furthermore, because the project is a duo assignment, the two projects should be as close related as possible, and preferably be connected.

1.2 Research Goal

The research goal for this thesis is: "To design and validate a process which Exact can use successfully to create a development product backlog in a distributed requirements engineering and design setting between Delft and Kuala Lumpur."

1.3 Research Questions

The goal defined in the previous section is rather broad and needs to be scoped down, therefore the initial question in this research is “What are the major challenges and bottlenecks in the global Exact development process from an agile development perspective?”. This research question is addressed in Part II. Based on the answers to our innitial question, an agile solution should be developed. Part II concludes with a proposal for this agile solution, which is further investigated in Part III. In Part III in-depth research questions are defined regarding the proposed agile solution.

1.4 Content

The thesis is divided in three parts: Part I, an introduction to the report, and the next parts cover different phases in our research: Part II, challenge identification and Part III, challenge resolution.

In Part I the thesis topic and the environment at Exact are introduced. Since the project is carried out at Exact, Chapter 2 introduces Exact, its products, culture, and the challenges Exact faces. The assignment is on software development and should integrate with the current development process in Exact. Therefore, in Chapter 3 the currently used development process is described: the “Refined Development Process”.

In Part II the challenge identification phase is described, an attempt is made to identify challenges within the global development process in Exact, and proposes resolutions for these problems. In Chapter 4 is discussed the need for problem identification in our research. Our approach to challenge identification interviews is described. In Chapter 5 is described what problems were mentioned during the interviews. A selection of challenges is made, which are most suitable for further research in the context of a thesis in the area of globally distributed software development. Based on the challenges identified in Chapter 5, some proposals on how to resolve these issues is made in Chapter 6. For each proposal is described what issues from Chapter 5 are resolved. Finally one proposal is selected to address in the next thesis part: Challenge Resolution.

In Part III, the last part of the thesis, the challenge resolution is covered. Research is done on automated status updates, to support the feedback loop between product management and product development. In Chapter 7 is defined what the goal of implementing automated status updates is. In Chapter 8 an introduction is given to the concept of continuous integration and automatic building. Later in this thesis the contents of this chapter are used to define what features of continuous integration are useful for Exact and to evaluate the success of the project. The approach to implement status updates is iterative. Each iteration an improvement is made. In Chapter 9 is described for each iteration what improvements are made, for which reason, and what the effects are. In Chapter 10 will be described how the tooling can be effectively used in combination with the RDP. The thesis concludes with some evaluation and recommendations in Chapter 11 based on evaluation of the proposals, and experiences we got during our stay at Exact.

Chapter 2

Exact Software

This graduation project will be executed at Exact Software as a research project for the TU Delft. In this chapter a summary is given on the history, culture, and products of Exact, and conclude with the types of challenges Exact faces.

2.1 History

Exact was founded in 1984, creating business software for small and medium sized companies. Through the years Exact expanded its product line from financial support software to enterprise resource planning software to online accounting software and grew to be a global company with divisions and customers all over the world.

2.2 Culture

Exact has a culture of continuous change and improvement. Change can be initiated company wide or by an individual employee. Whenever there is a possibility to improve the current methodologies, it is expected from employees to improve on this. Through this culture of change, the development and business processes are constantly evolving. At the time of this project, the development process iterating between Delft and Kuala Lumpur is altered from a waterfall approach, to a more agile development process (See also Chapter 3).

2.3 Products

Currently the product portfolio of Exact consists of many products including an process management package called Exact Synergy, and an enterprise resource planning package called Exact Globe.

Both Exact Globe and Exact Synergy are a result of the development process iterating between product management in the Delft, Exact Head Quarters (Exact HQ), and product development in Kuala Lumpur, Exact Asian Development Center (Exact ADC, short ADC). Requirements are elicited from customers all over the world, prioritized and elaborated in Delft; designed, implemented and tested in Kuala Lumpur. Reasons for this globally distributed software development process include cost reduction, availability of staff, and acquisitions.

Exact Synergy While both products are developed in Kuala Lumpur, their development processes are similar. However, the development of each product has different challenges. Exact Synergy is a relatively new product of which only a few versions have been released. The main challenge for Exact Synergy is the development of new experimental features. Most communication with Kuala Lumpur covers the details and the context of these new features.

Exact Globe Exact Globe is a more mature product which already includes many features and has many versions used by customers. The main challenges for Exact Globe are retaining compatibility with previous versions of the product, and integrating new features without breaking old ones. Most communication with Kuala Lumpur covers the integration of the new features in the product.

2.4 Challenges

Implementing an agile methodology in a large organization can be difficult [14]. Some efforts already have been made in Exact by introducing the Refined Development Process (See also Chapter 3), as a first step to fully implement Scrum [50]. However Exact recognizes there is more space for improving agility and this thesis is a part of this step-by-step improvement process on top of the RDP.

Companies have to deal with higher customer expectations on quality and time to marked [51], and a more competitive business environment [16, 15]. Agile methodologies address challenges such as changing customer requirements and short iterations reducing time to market of a feature [10, 34]. Much research has been done on the effectiveness of agile methods [47, 13, 57, 45], and numerous agile methods have been proposed [50, 37, 9, 5, 21, 6, 56]. Introducing agile in companies is a challenge [14, 52] especially for big companies [13, 22, 25, 42, 41] this challenge becomes bigger when a company operates globally [46]. Globally distributed software development addresses some business challenges such as reducing costs, access to large workforce, close customer proximity, and global presence, to name a few [20, 19, 8], however globally distributed software development is challenging because of time differences, geographical dispersion [18] and cultural differences [33, 8, 35]. Therefore its effectiveness is not guaranteed [23]. Because of the benefits of combining agile and distributed development research is being done on blending these methods [55, 44, 40]. Some argue that these challenges in blending agile and globally distributed software development should be addressed with technical support [27].

Exact recognizes the benefits of agile mentioned above, though also faces the challenges of blending agile and global software development and introducing agile in a big company. Some efforts already have been made in Exact by introducing the Refined Development

Process there is more space for improving agility and this thesis is a part of this step-by-step improvement process on top of the RDP.

Chapter 3

Refined Development Process

Exact embraces change: the business processes and way of working at Exact constantly evolve. The development process of Exact is also subject to continuous improvements. The latest change in the development process is the introduction of the “Refined Development Process” (RDP). Before software was developed in a waterfall [49] approach combined with ideas from the unified software development process [38], while its strong points were that it is structured, proved and detailed, there were several drawbacks:

- Difficulties in estimating required time and resources, resulted in project delay. Making planning estimates in the area of software engineering is difficult in general.
- Long release cycle. From request of a feature to release could take 2 years. Changing market conditions, require a shorter response time.
- Because of the waterfall methodology it could be hard to return to a previously made decision or ask clarification on a decision. This could result in challenging PRD’s and misunderstood requirements, making projects as a whole more challenging. Change is a constant factor in development and should be supported by the methodology.

The RDP is introduced to solve these issues, with three goals in mind: iteratively, agility and efficiency and introduce some SCRUM [50] based ideas into Exact. This chapter will describe the RDP. First an overview will be given the new process and each of the individual deliverables will be described.

3.1 Overview

The RDP is a modification on the previously used waterfall development process, to preserve its strengths and address its weaknesses. The RDP touches upon communication between parties and scheduling of development phases.

One of the main changes is the introduction of the “boat”-concept to improve communication. The boat consists of all stakeholders of the project. During the entire project there is continuous communication between all parties in the boat: This ensures that all parties are constantly aware of the decisions made.

Some phases in the development process are more closely related than others, especially the processes that follow each other. Therefore some processes require more explicit communication and partially overlap in time, to promote simultaneous development and a close verification cycle, along the way.

To spread risk and allow for easy feedback and re-planning, development phases itself consist of multiple iterations. Each iteration is used to complete a given task and based on the experience gained, a new iteration is started.

The schedule of the different development phases is displayed in Figure 3.1. In the following sections each of the development phases will be described.

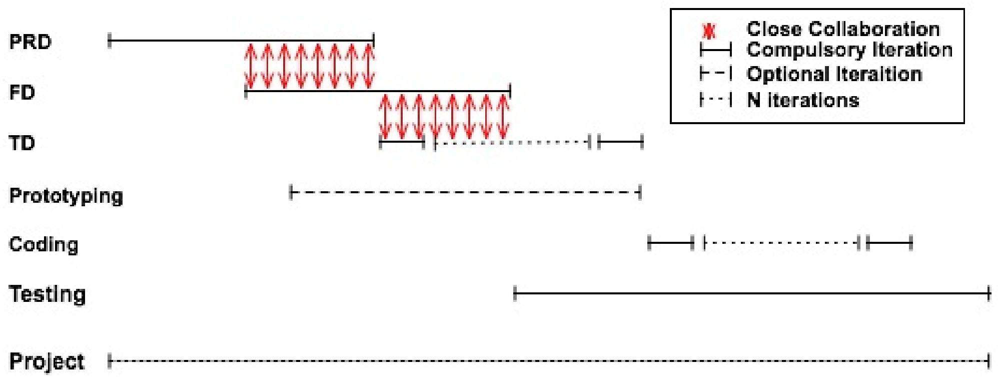


Figure 3.1: Overview Refined Development Process at Exact

3.1.1 Product Requirements Document

The product requirements document (PRD) is developed in the first phase. The PRD consists of the customers requirements for a new feature, its purpose is to represent many customers. The PRD is developed by product management in Delft, based on customer input, technical needs and own observations.

3.1.2 Functional design

In the second phase the functional design (FD) is created. The FD is developed based on the PRD. When the PRD is halfway completed, the development of the FD is started. The FD describes what exactly should be implemented based on the customers requirements.

The FD is developed in close collaboration with the PRD since it is required to verify whether the customers requirements are realistic.

3.1.3 Technical design

In the technical design phase is specified how the functional design will be implemented. This is specified in the technical design (TD). The TD is created in multiple iterations based on the functional design. The TD-phase is started when the PRD is completed, partially in parallel with the creation of the functional design. During the TD-phase is verified whether the FD is feasible. This is done in close collaboration with the design team.

3.1.4 Prototyping

An optional phase, closely related to the technical design- and development- phase, is the prototyping phase. The main goal of the prototyping phase is to build a try-out version (the prototype) of ideas upfront- or during- the technical design phase. By developing a prototype the development team can assure itself an idea works out both in theory and practice. Preferably the prototype is used as a basis for the coding phase, however it might be thrown away.

3.1.5 Coding

During the coding phase the technical design is converted to a working information system and unit tests are created. The technical design gets the development started, and the implementation details are worked out. The coding phase consists of multiple 2 week iterations, where in each iteration a subset of the project is implemented. The project subsets can be selected based on priority, shared functionality or development skills of the team members.

3.1.6 Testing

During the testing phase is verified whether the implementation of the system is conform with the specifications in the FD. This phase starts right after the FD is completed, since the test scripts can already be developed based on the contents of the FD. The testing phase is completed after development is done, and no more significant errors can be found in the implementation.

The RDP provides a set of guidelines on how projects are managed: which parties communicate when, when close collaboration is required, and which deliverables are expected when. This set of guidelines is not yet complete, and there are many possible improvements. To improve this process step by step, a set of improvement areas should be defined. The next part of this thesis will investigate in which areas most improvements can be made.

Part II

Challenge Identification: Interviews

Chapter 4

Problem Identification

This part of the thesis describes the first part of my research project at Exact ADC: problem identification. We conducted a series of interviews with key people working in the globally distributed development process in Exact, focusing on the collaboration between Delft and Kuala Lumpur. This part will describe the setup of the interviews, initial results and propose some possible solutions for the identified problems.

In this chapter will be described what the goals of the interviews were and how the interviews were setup.

4.1 Goal of Interviews

The goal of the interviews is twofold A) get insight in Exacts global development process iterating between Delft and Kuala Lumpur and B) to identify bottlenecks to be addressed in this thesis. The goal of this initial research is to explore the development process and the challenges in the development process, therefore semi structured interviews seem to be most suitable at this stage of research [62, 61, 54, 58].

The development process as described in Chapter 3 seems simple, however in real life it is a complex global process with many stakeholders involved. The process is documented fairly well, however documentation can be outdated or inaccurate. Furthermore many stakeholders are involved in the development process, all having their own interpretation and application of the development process. Furthermore for our purposes we needed to focus on special needs and experiences with the current process. Since no party has the whole picture of the process and using documentation to get insight in the process is hard, interviews with stakeholders were used to get insight in the development process.

The initial goal of the research was to address the following problem from a development perspective: *To design and validate a process which Exact can use successfully to create a development product backlog in a distributed requirements engineering and design setting between Delft and Kuala Lumpur.* This, however is not a concrete problem that can be addressed at Exact ADC, therefore the interviews were conducted to identify bottlenecks in this area.

4.2 Setup of Interviews

Setup of the interviews involved to parts: selection of interviewees and the structure of the interviews. Interviewees were selected based on their involvement in the development process and their level of contact with Exact ADC. The group of interviewees consisted of people fulfilling different roles in exact:

- Regional Managers
- Product Line Managers
- Product Managers
- Software Architects
- Software Engineering Managers
- Functional Designers
- Quality Engineers
- Software Engineers

Interviews were conducted as semi-structured interviews. For each of the roles in the list above set of relevant interview topics was constructed. Relevant topics were selected from SWEBOK [7], to make sure all items relevant to their roles are discussed.

In the following chapters, the interview results are presented and based on these results, actions are proposed to resolve the issues.

Chapter 5

Findings

In this chapter the findings of the interviews will be presented. The full interview notes are not included, however the main ideas are summarized in this chapter. The development process will be described and the bottlenecks will be displayed.

5.1 Bottlenecks

During the interviews many bottlenecks or possible ways to improve the development process at Exact were mentioned. Unfortunately we are not able to address everything mentioned, during our short time at Exact ADC. Therefore a selection should be made from all comments made.

Together with a senior employee at Exact ADC we made a selection of most relevant bottlenecks to address in my assignment based on the following selection criteria:

- Should be in the problem area of software engineering (in the scope of my research).
- Should be a significant, real problem which the actors in the development process have to deal with.
- Should not be covered by the design goals of the RDP.

The following table summarizes the bottlenecks mentioned during the interviews. The challenges matching all of the 3 criteria are marked with an “X”.

#	Challenge, mentioned by interviewees	
C_1	Maturity of software, sometimes bugs are discovered in previous versions which need to be tested in the new release?	
C_2	Lack of resources	
C_3	Some people lack seniority. Sometimes people need to be introduced to their task on the job.	X
C_4	People get involved late in the project. These people need to be taught on the project; this is a hard job since people on the project are already busy and do not have time to teach others.	X

C_5	People used to work at many projects at once, because there were many projects and a lack of resources	X
C_6	Time differences make it hard to resolve issues with Delft. It can take days before issues are resolved.	X
C_7	Collaboration with SE and QE can be too close, since test-scripts represent the ease of SE instead of the customers requirements.	
C_8	Some of the code in Exact is hard to maintain and test. More effort should be put in code quality like refactoring, rewriting and maintenance of code quality.	
C_9	Testing takes long for Exact, especially when many test cases are involved. Automated test runs should solve this problem	X
C_{10}	So much time is spent on working on projects, that there is no time to improve on coding practices and development practices	X
C_{11}	Scrum should be used to cut a big project in to independent small projects when possible	X
C_{12}	Automatic and manual code reviews on quality could improve the code.	
C_{13}	Code reviews are done, however no attention is paid to the use of design patterns and object design	X
C_{14}	It could be easier to design when coding, especially when the design document is not perfect.	
C_{15}	Automated QE test scripts could save QE lots of time if there was tooling available for this.	
C_{16}	Direct customer input can be an eyeopener since they comment on a product from a purely functional point of view, while developers focus on code/design flexibility	X
C_{17}	Direct customer contact can be impossible, since a product is made for thousands of customers.	
C_{18}	Spend more time on early phases of the release to do the preparation work; PRD, TD and IA.	
C_{19}	Retain people having critical information on the product.	
C_{20}	Big challenge is changing requirements along the project. Small changes are fine though big changes during the process are hard to implement. Changing requirements result in rework on documents.	X
C_{21}	Major development challenge is selecting the right design and thinking about the future. The design should be reusable and created with the future in mind.	
C_{22}	Some SE know about flexible design however some should be educated on changing design.	
C_{23}	Communicating functional knowledge to SE who used to work on other areas.For example it's hard for someone to work on finance who mainly got experience in human resource management.	
C_{24}	Sometimes not enough communication with all parties in the boat. This results in lack of information, rework, not sure everyone is in on the same page in the projects.	X

C_{25}	Size of team can vary from 6 to 20 people. 20 people is too big for efficient communication	X
C_{26}	At the end of the project many issues arise, which results in extra work to fix these issues.	
C_{27}	Hard to estimate the project length.	X
C_{28}	Some times projects are too flexible with people moving out of the project.When issues arise it can be hard to get these people back to resolve these.	
C_{29}	Everything is flexible for a project except from the delivery date.	
C_{30}	Confusing input from Delft, results in rework	
C_{31}	New development process result in too broad focus: impossible to focus at one task at a time like the waterfall process.	
C_{32}	People should be more open and honest in the project about the status of their work.	
C_{33}	More face to face communication with the project manager to create a clear view of the project when he's here.	
C_{34}	Not all developers are involved in the deep-dive meetings. This could be dangerous since you're never sure the lead passes the messages correctly.	X

Chapter 6

Proposals and Selection

This chapter will describe four proposals addressing different challenges described in Chapter 5. Each proposal is designed with one or more of the "X" marked challenges from Chapter 5 in mind. However the proposal may also addresses some of the unmarked challenges. For each proposal will be described what the general idea is, a goal will be set, and which challenges might be addressed.

6.1 Proposal I: Create space in projects to improve on agile development practices

Addresses: C_8, C_9, C_{10}, C_{12}, C_{14}, C_{15}, C_{21}, C_{22} from Chapter 5.

In this section an enhancement, in the local development process at Exact ADC will be proposed, to create space for introducing more agile development practices.

While many management-level Scrum practices are already adapted at Exact ADC, some of the more development aimed agile practices (such as described in Pragmatic Programming [37], eXtreme Programming [9], and The Joel checklist [53]) are not yet in use. To reduce development effort, investments should be made in automatic integration of projects and automated testing[1]. To support automated testing the code, especially legacy code, and newly developed code has to be improved by refactoring and design patterns.

During the interviews problems on code quality, automatic builds, automatic testing, and lack of source version control were mentioned. These problems are all development related, and the software engineers (SE) and quality engineers (QE) know best where these problems exists and how these problems should be resolved best. Therefore these problems can be solved best in a bottom-up approach letting the engineers decide on which solution to choose. However time is required for engineers to let them work on new solutions: Reserve half a day or a full day per team member per week, in which team members can work on improvements research and implementation of tooling, design patterns, automatic testing: see Table 6.1 (Mainly addresses C_{10}).

[1]Note, that this might be challenging for some projects; reasons for this are legacy elements, integration with 3rd party tools, and product specific build requirements. Globe for example, would require a custom build tool aimed at Exact's situation only.

h!

Table 6.1: Roles for improvement

Role	Description
Build	Investigates and implements tools to make nightly builds possible, and integrate it with the current version control system and codebase. This role is probably most suitable for a SE. (Improves on cutting down on test time C_9, by automation of testing process C_{15} , and improves code quality C_8, by optionally implementing automated code reviews C_{12}).
Unit Test	Investigates and implements tooling for automatic (daily, hourly, after check-ins) unit-tests/test-coverage and integrates it with the existing version control system and codebase. This role is probably most suitable for a SE. (Shortens testing process C_9, by automation of testing process C_{15})
Functional Testing	Investigates and tests tooling for (partially) automatic testing of user functionality. This role would be most suitable for a QE. (Focuses on C_{15})
Code quality	Investigates and implements tooling to guard for usage of design patters, code duplication and need for refactoring. This role is most suitable for an SE. (Focuses on code that is difficult to maintain C_8, and improve code quality C_{12}, and getting the right design C_{21} by automating code reviews)
OO-Evangelist	Educates and stimulates people in the use of design patterns, testable-, reusable- code and refactoring. This role is probably most suitable for a senior SE in the team, having experience with OO software design. This is a challenging role since its goal is to stimulate people to improve code and working practices. (Tries to improve design C_{21}, knowledge on design C_{22}, to improve code quality C_8)
Version Control	Sets up product wide version control to use version control to allow for automatic -integration and -testing for all projects.
Automatic Deployment	Investigate tools and techniques to allow for automatic deployment of development and production code.

The remaining time can be used by members to fix minor bugs, and improve the existing code by refactoring and creating unit tests. Knowledge, results and proof of concepts from the projects started in the different roles can be shared in “Friday Afternoon Tech Talks”, pair-programming and technical solutions might be deployed Exact-wide.

Desired end result: A software process in which developers are able to optimize their way of working based on their own experiences.

6.2 Proposal II: Create a methodology to create a product backlog

Addresses: C_6, C_{11}, C_{20}, C_{23}, C_{24}, C_{25}, C_{26}, C_{27}, C_{29}, C_{30}, C_{34} from Chapter 5.

For product development at Exact ADC a project is started when product management hands over the first version of the product requirements document (PRD). Often the product management is required to come over from Delft to Kuala Lumpur to discuss the PRD in the “deep dive meeting”. During the deep dive meeting the key members of the development team can ask for clarification and make assessments on the technical feasibility and planning of the project. Based on this information the PRD is modified and the development of the functional design (FD) and technical design (TD) is started.

This approach has several drawbacks:

- Project managers need to come over for several days or weeks to discuss, technical issues, planning issues and clarify the PRD.
- The PRD needs to be in a certain state of completeness for the project to start. This is hard due to requirements that can change later in the process (C_{20}), scope may explode (C_{27}) and projects can get large (C_{11}).
- The development team has to wait for the PRD to start a new project.
- It is non trivial to convert the PRD to tasks for the development backlog.
- The deep dive meeting is not enough, and later on in the project there is still need for clarification from both sides on the contents of the project. This can be difficult since not everyone can attend these meetings (C_{34}), due to time differences (C_6), narrow communication channels to transfer knowledge on complex business information (C_{23}, C_{24}, C_{25},C_{30})

Resolving these drawbacks the ideal approach would be one having the following features:

- Project managers are not required to come over to Kuala Lumpur to elaborate on the PRD, however issues on planning, technical challenges and uncertainties are resolved in continuously.
- Only a subset of the requirements from the PRD is completed and these are enough to get a developer working on a feature.

- product development can pull a requirement from the basket of prioritized features anytime there are enough free resources.
- Each task in pulled from the requirement basket represents a feature to be implemented and tested on its own in a short period of time (¡ 10 days)
- Allow product management and product development continuously to discuss on:
  - Planning
  - Technical feasibility's
  - Misunderstandings

Desired end result: A process and optional proof-of-concept software to support the distributed creation and usage of a product backlog used by product management and product development[2].

6.3 Proposal III: Automate status updates to product management

Addresses: C_6, C_9, C_{12}, C_{15}, C_{16}, C_{17}, C_{20}, C_{23}, C_{24}, C_{25}, C_{26}, C_{27}, C_{30}, C_{34} from Chapter 5.

Currently test releases for product management created upon request and manually. Feedback is given at the end of projects during the short time frame in which office hours between KL and NL overlap. This has the following drawbacks:

- A test/tryout version of the most recent synergy version is unavailable for product management up until the last weeks of development. (C_9, C_{12},C_{16}, C_{17},C_{23}, C_{24}, C_{25},C_{30},C_{34})
- No feedback can be given on the most recent modifications (C_{26}, C_{27})
- Creation of a test version of the software always requires human effort and could involve human errors. (C_9)
- The valuable office hours that overlap between KL and NL are spent on giving feedback on the product and the feedback is received in the afternoon instead of the morning, when the daily planning is made. (C_6)

By releasing early and often a short(er) feedback loop can be created with product management, allowing for early and frequent feedback on completed and uncompleted features. This could reduce rework, allow for easy feature-tryouts, and reduce misunderstandings. Feedback could be on look and feel, however also on the business logic involved. Another problem often mentioned is the small time frame in which feedback can be requested on the current system: for KL this would be between about 3pm and 7pm. A more natural time to

[2]At the time of writing the product management team started working using a product backlog based approach. See also e-Portal document 19.532.411.

receive feedback would be at the beginning of the workday before scheduling the tasks for that day.

Currently the setup of a Synergy try-out environment is a manual process. The code needs to be checked out, compiled, tested, and composed into a release package, and deployed manually. This is done upon request.

By automating this process and performing it on a daily/continuous basis, product management would be able to monitor the development of new features closely and give feedback when product development sleeping; able to process it in the morning.

Benefits of automatic deploying nightly test versions are:

- Allows for frequent feedback moments on work delivered
- Extends the time frame in which feedback can be provided, since the test version is released when product development is asleep and product management are able to review the features
- Product development can receive feedback when they arrive at the office before planning the day.

An additional benefit is that:

- Creating a production release becomes a natural automated step in the process, reducing errors and the need for big bang testing of the release version of the software.

Dependencies: This proposal goes probably together with the implementation of a couple of roles mentioned in Proposal I: version control, automatic building and automatic deployment. Note that I maybe able to fulfill some of these roles.

Desired end product: A system that does nightly code checkouts, nightly builds, optional automatic unit testing[3], and automatic deployment of the most recent e-Synergy version. This system should allow for a process in which product managers can give frequently asynchronous feedback outside the valuable overlapping office hours[4].

6.4 Proposal IV: Close customer involvement

Addresses: C_{33}, C_{16}, C_{17}, C_{23}, C_{30} from Chapter 5.

SEs tend to focus on creating software that is modular, reusable and with future modifications in mind. While this software might be useful for power users, regular users may find it hard to use and inconvenient.

By involving a representative customer in the development process, allowing her to give frequent feedback on the created screens and functionality more user friendly software can be created. The feedback could be from a customer, but it could also be a Exact employee specialized in usability of software.

From [59]:

[3]The automatic unit testing requires unit tests to be available in the source code.

[4]Choose this proposal as graduation project

> "The goal is to involve the customer in the development process, and to be able to use his feedback to improve the quality of the PRD.
>
> The main question will be how to involve the customer in a distributed agile software development process, where customer should be read as being a market segment, or at least multiple customers.
>
> Sub-questions are:
>
> - why involve the customer
> - what to share
> - when to share/how often
>
> Ideas for a supporting tool are:
>
> - user groups (with a group moderator as single proxy-customer for physical contact)
> - round-the-table meetings
> - etc
>
> An ideal image of the future would be a situation that the Scrum team would consult the customer themselves, while product management is just product owner."[5]

6.5 Proposal Selection

The proposals described above have been discussed with several parties within Exact and the university: Product management, development managers, and university professor, to choose a feasible graduation topic.

Different parties had set different constraints on the assignment. From the Exact perspective the assignment should have potential for more efficient agile working methods with a minimal impact. From a university perspective the assignment should consist of a clear goal, have scientific relevance and be relevant in the area of globally distributed software development.

Based on these constraints some of the proposals are not useful for further exploration. Proposal I lacks a clear end goal, and is therefore not useful. Proposal II is scientifically interesting, though from an Exact perspective it would be really challenging to develop this in one assignment, and should be done in smaller steps. Proposal III is interesting, since it has a clearly defined technical end product, involves GDSE, and has a low impact since some parts are already available. Proposal IV is already addressed by Christian Visser [59] since it makes more sense to address this issue from a product management perspective.

The initial goal was to "To design and validate a process which Exact can use successfully to create a development product backlog in a distributed requirements engineering and

[5]Discussed this proposal with Ken and we agreed that is less relevant for product development is more applicable to product management. Christian Visser is currently working on this topic as his MSc project from a product management perspective.

design setting between Delft and Kuala Lumpur." and it would be addressed as a duo assignment. However such a process already exists and has been heavily invested in, in this research paper will be investigated what tools are required to support this process. This tool will be focused on automating status updates from product development to product management and enable a faster feedback loop between product management and product development; in this way laying the foundations for a more agile global development process. Our team member will further investigate customer involvement and at this point in time there is no reason do further research as in a duo assignment.

Thus, the most suitable proposal to be addressed in Exact is "Proposal III: Automate status updates to product management".' This proposal will be further addressed in Part III.

Part III

Challenge Resolution: Automated Status Updates to Product Management

Chapter 7

Automated Status Updates

This thesis part is on automated status updates. This section will set the goals for this part, set clear objectives, define the research questions, define the deliverables and describe the approach.

7.1 Goal

The goal of this part is to ease communication between PM in Delft and PD in Kuala Lumpur. Currently, status updates on the latest software version are communicated manually at irregular intervals. This part will explore possibilities to automate the update process: making the development process more transparent and allowing product management to give continuous feedback on the current version of the product.

Requirements development and product development take place at different locations. Requirements are developed in Delft and implemented in Kuala Lumpur. After the requirements are completed PM and PD collaborate to create the product. Updates on the development process are composed manually at irregular intervals which has multiple drawbacks. It is time consuming and allows for human errors. It is hard to keep PM up-to-date with current information on the development progress, making it hard for PM to act upon the most recent events.

7.2 Objective

The objective of the project is to make the development progress in Kuala Lumpur more transparent to product management in Delft. To accomplish this tools and techniques for build automation and communication of results will be implemented and tested in the development environment at Exact. The benefits will be both for product management and product development:

1. Product Management:
    - Create a more transparent software development process.
    - Provide instant updates on the status of projects.

- Have instant information on the health of code base and product status.

2. Product Development has instant information on the health of the code base

    - Save time on creating intermediate- and release- versions of their software.
    - Save time on manually executing test cases.
    - Save time on manually creating status reports.

7.2.1 Scope

For this project the focus will be only on the Synergy project. In the past, research has been done on the automatic builds for the Globe project, however this gave too many technical challenges.

7.3 Questions

In Chapter 1 we defined our research question as "What are the major challenges and bottlenecks in the global Exact Software development process from a development perspective?" to give find challenges with in Exact to supporting our research goal. In Part II addressed this question and proposed to touch upon the topic of automatic status updates.

This part will dive further in the topic of automated status updates within Exact, therefore the a research question needs to be defined addressing this topic. The research question answered in this part will be:

How to automate the process of giving product management insight in the current development status of e-Synergy projects in a distributed development process on a daily basis?

Subquestions are:

- What information on the development status would be useful for product management to receive on a daily basis?
- What information on the development process can be automatically provided on a daily basis?
- How does the information provided on a daily basis, affect the communication between product development and product management?

7.4 Deliverables

This part of the thesis has as deliverable: an automated build system that is able to build the Synergy product, run automated tests and report results. This system can be scheduled to run at any time by itself and can be activated and monitored remotely from any geographical location within Exact. Users consist of, but are not limited to, product management and product development in Delft and Kuala Lumpur. Product management tends to be more interested in project status and functional tests, while product development will be more interested in technical tests such as unit tests and build results. Besides the system itself, also working guidelines and work processes will be developed as part of the system.

7.5 Approach

Our research will use the current TFS pilots used within ADC as a basis. Currently some projects use TFS, to try out its features. The set of features used varies from project to project, however the source control feature is always used. Therefore the usage of source control will be assumed as a basis for further improvements. During our research sprints will be used to improve on this system.

The approach in our research will be agile, using 2 week sprints. At the beginning of each sprint will be defined what will be implemented. The rest of the sprint will be used for implementation and evaluation. Each sprint will be concluded with an evaluation of the existing system. Results of this evaluation will be used as an input for goal definition for the next sprint. The next chapter (Chapter 8) will give an introduction to continuous integrating and automatic building, to see which features are useful for Exact later in the sprints.

Chapter 8

Continuous Integration and Potential Benefits

A tool to generate automatic updates is a family of techniques often called continuous integration [28]. Automated builds and continuous integration form an important part of the agile methodologies that offer concrete development guidelines [6, 37, 9, 56] and therefore seem suitable to apply within this project.

This chapter will give an introduction to continuous integration and its potential benefits. These benefits will be linked to product management and product development in Exact. Later in this thesis the possible features and benefits of continuous integration will be used to decide what should be implemented at Exact and to evaluate the success of the project.

8.1 Continuous Integration

Continuous integration is a method to continuously integrate the work of different people in the team, with the goal to verify the correctness of the different pieces of work integrated, to improve product quality. To improve product quality continuous integration offers a set of features, however to use these features with a maximum effect a set of best practices is often mentioned. In the following subsection these features and best practices are described.

8.1.1 Features

In [43] a basic set of features for continuous integration is given. The most basic system consists of a single source repository, and an automated build:

Single source repository In [37, 43, 56, 28] is argued that there is no good reason not to have a source code management system in place. As a basis for continuous integration lays a source code management system in which all code can be found. This to ensure each team member is able to check in his code in the same repository, allowing the build system to continuously verify whether all projects still integrate well.

Automatic build The building process should be fully automated [37], and one single action should allow the system to be build from scratch. An automated build should be in the most basic continuous integration system, since it is the most basic check available: to release a software system, the source code should at least compile.

Continuous builds After automating the build, the build should be triggered. This can be nightly, hourly, or ideally whenever new change sets are available. The purpose is to check on a regular basis whether the system is in a healthy[1] state.

Publication of results Once the integration step is performed the results should be published to the stakeholders, so that when errors occur necessary steps can be taken to resolve these. In a globally distributed development process it is even more important that all stakeholders get notified on the build results to enable for a transparent development process [19].

The basic system can be extended with for example automated testing and deployment which will be described below. For a more complete list on possible extensions of continuous integration see [28].

Self testing The code should consist of a test suite of unit test to check the system for different types of bugs. Types of tests that can be included are: unit tests [12], functional tests [3], programming errors [24], security checks [1], and code quality [4]. Testing comes back each release, multiple times. Ofter tests have to be executed multiple times for multiple releases, to verify new functionality did not break existing functionality. By automating the testing process lots of time can be saved on each release, making it possible to shorten the release cycle.

Make the latest software version available to all stakeholders Software should interact with other parties such as its users and other software systems. This feature is to get feedback on how the software interacts with other parties. Once the software compiles and all other tests pass, the challenge is to get the software out there and receive feedback from other parties involved with the product you made. These other parties can be within your own company like managers, product managers, or marketing employees, however this can also be partners developing software based upon your own products. The feedback from these parties early on in the development process allows to resolve issues early on.

8.1.2 Practices

With a set of tests and features comes a set of practices to get a maximum benefit from the tooling. The most important prerequisite for continuous integration are daily commits; for a more complete list see [43, 28].

[1]The definition of healthy depends on the checks that are executed during the automatic build, though in the most basic system, this would mean that the sources compile.

Daily commits To continuously integrate changes, each developer should publish his change sets on a regular basis. It is hard to give a definite interval on this but experienced developers recommend to commit changes on a daily basis [37].

8.2 Benefits

Continuous deployment has some potential benefits; the following subsections will discuss the benefits for both product development and product management.

8.2.1 Product Development

Since automatic builds generate mostly information on the work produced by product management, and the benefits for product development are mostly on increasing agility as described in [6, 37, 9, 56]. A more detailed overview of the benefits of continuous integration can be found in [43].

- Reduces the risk of integrating everything at the end of the project
- Early detection of bugs
- Early detection of regression bugs
- Easier to track the cause of a bug
- Reduction of the “Broken window syndrome” [37, 39][2]
- Removes barriers to frequent deployment
- Allows for frequent feedback

However the authors of [43] argue that the value of continuous integration depends largely on the quality of the set of tests run on the code.

8.2.2 Product Management

Since product management and product development are far apart in geographical-, -temporal, and sociocultural- distance, the potential benefits can be found in the paper “How Technological Support Can Enable Advantages of Agile Software Development in a GDSE Setting” [27].

The authors of [27] argue that tools to support “Short iterations, frequent builds and continuous integration” offers benefits in the distributed projects on sociocultural-, geographical- and temporal- distance in the following ways:

[2] In [39] is argued that to retain quality of life in a neighborhood and reduce crime, it is important to maintain order an a preventive manner. In [37] this concept is applied to software, arguing that it is important to keep the bug rate low, encouraging developers to keep high quality standards for the work they produce, resulting in high quality software being produced.

- Feedback motivates the developer (affects sociocultural distance)
- Getting to see high quality work from team members increases trust (affects geographical- and temporal- distance)
- Makes process more transparent (affects geographical- and temporal- distance)

And helps to elevate the following challenges:

- Reduce communication overhead (affects all distances)
- Spontaneous meetings are hard to initiate (affects geographical and temporal distance)
- Consistent global view of system is hard to maintain (affects geographical and temporal distance)

On a local level short iterations, frequent builds and continuous integration can support the following goals of the agile manifesto [27, 10]:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design enhances agility.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

In Chapter 9 will be described on a per sprint basis which of the features described in this chapter are implemented in Exact, what challenges are resolved, and which challenges arise.

Chapter 9

Iterations

In this chapter each iteration will be described. Per iteration will be described what the goals are, what has been implemented and conclude with an evaluation which serves as an input for the goal definition of the next iteration.

Initially TFS [2] was used as a pilot in several Synergy projects. The TFS features used vary from project to project, however most teams already use the source control feature: sources can be shared through the TFS-webinterface or a Visual Studio plugin. Therefore we assume this as a basis for our iterations: the use of TFS source control. Each iteration we build further upon the use of TFS source control.

Consecutively we have gone through several iterations: building Synergy from sources, mirroring the source repository, publishing the nightly build, automate deployment of the nightly build, and finally designed and implemented a process to make effective use of these new features in the communication between Delft and Kuala Lumpur.

9.1 Iteration I: Building Synergy

The basis for automatic software updates is the compilation of the software system. The generation of reliable status updates depends on the quality of the build. If the build consists of errors, or the build is not a representation of the sources, the generated status updates are not guaranteed to represent the sources in version control.

9.1.1 Goal

Synergy is a large system and has been under development for over 10 years. It consists of many source files, database updates, and data files. These have been enhanced over and over again by different people to solve bugs and add new features. This history of modifications introduced circular dependencies between the sources the Visual Basic compiler is unable to deal with, which makes building Synergy a challenging task.

To deal with the circular references, developers install the latest Synergy version locally and use installed libraries as dependencies for their local builds. Visual Studio has to be configured to use the proper dependencies, and set the right output directory. This approach has several drawbacks. 1) it does not guarantee every developer has the same build: developer A might choose a slightly different Synergy version than developer B, which can result in different results for different builds. 2) It can not be guaranteed that the binaries resulting from the build are a representation of the sources in version control: binaries may still represent the previous version instead of the most current one from version control.

Therefore the first goal is to automate and create a universal build process, creating a Synergy build from scratch, executed by TFS.

9.1.2 Approach

Building Synergy involved multiple steps: Build Synergy from scratch, build Synergy by TFS, and publish results of this build.

Universally building synergy from scratch

Synergy should be build from scratch in a universal way.

Building Synergy from scratch would require either to resolve the circular references or make the Visual Basic compiler deal with circular references. Since Synergy is a large project it would not be practical for one person to refactor all Synergy code, removing the circular references. Making the Visual Basic compiler support circular references does not seem to be a good idea either, since this option can not be enabled, neither it is possible to modify the compiler itself, since it is not open source. Therefore it seems Exact is currently stuck with a source base, that does not compile from scratch. (We will get back on this problem in Chapter 11).

Not all is lost though, since there still might be a way to come up with a universal build method. A universal build would require a standard set of binaries as dependencies to avoid, issues with the circular references. And this build should be used by both automated team builds[1] as well as individual developer builds. TFS and Visual Studio offer a feature called “Build Definitions”. A build definition defines the way the compilation is performed. Possible options to specify are the files to be included in a build. The build definitions can be used in individual builds or shared team builds.

Building Synergy using TFS

Now a universal method of building Synergy exists, Synergy should be automatically build from the source repository. TFS offers a feature to trigger for nightly, daily or continuous builds. When TFS triggers for a build, a build agent is notified. Using the build definition and the sources, the build agent tries to compile the Synergy sources, and returns the build status results to the TFS server, and the binaries to an optional shared directory (See Figure 9.1).

[1]Team builds are builds that are automatically performed by the TFS system. The work of the entire team is compiled together, on a nightly basis.

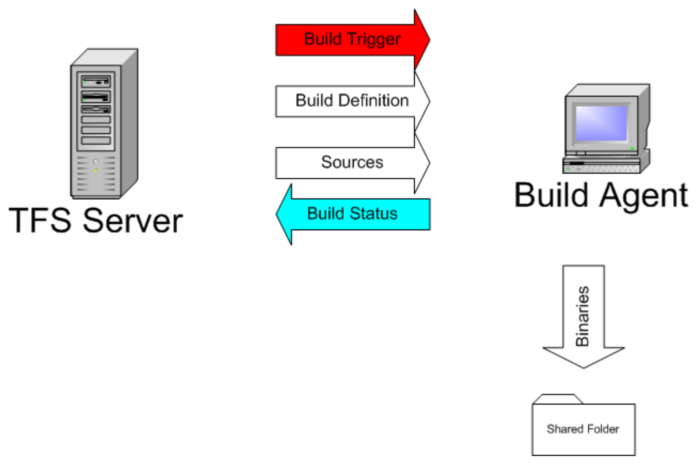
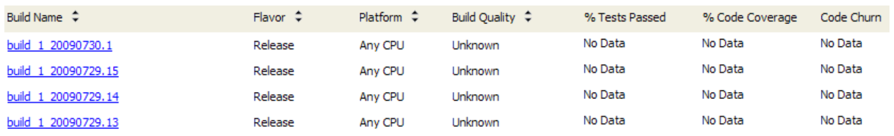


Figure 9.1: Build setup, with a build agent, publishing build status and resulting binaries.

Publish build results

For publications of these results the standard available TFS features are used. These are enabled by default, and publish build results in the TFS web-interface. A list of performed builds is displayed (See Figure 9.2), when a build is selected, the web-interface shows whether the build is successful (See Figure 9.3).



Build Name	Flavor	Platform	Build Quality	% Tests Passed	% Code Coverage	Code Churn
build_1_20090730.1	Release	Any CPU	Unknown	No Data	No Data	No Data
build_1_20090729.15	Release	Any CPU	Unknown	No Data	No Data	No Data
build_1_20090729.14	Release	Any CPU	Unknown	No Data	No Data	No Data
build_1_20090729.13	Release	Any CPU	Unknown	No Data	No Data	No Data

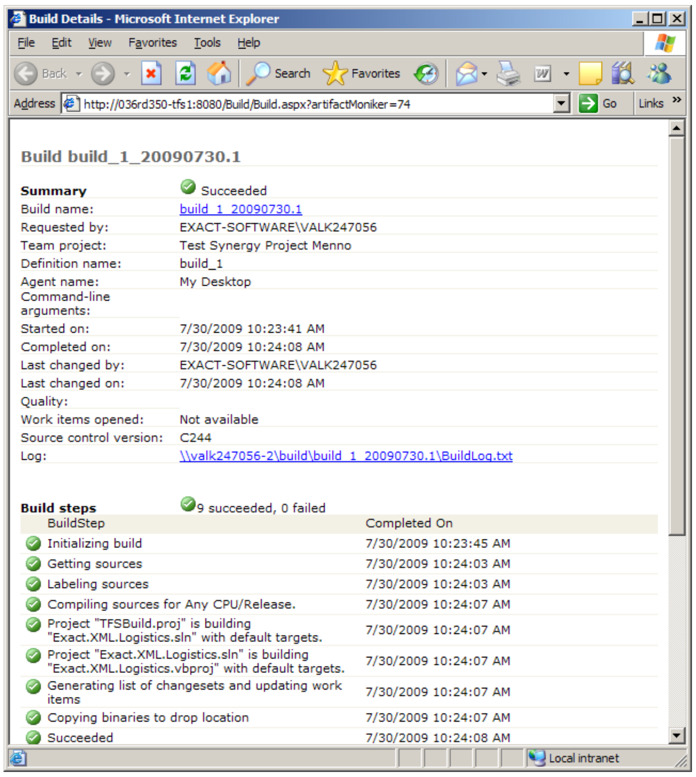
Figure 9.2: Overview of builds completed in the past

9.1.3 Intermediate evaluation

Research in this iteration has resulted in a system that allows Exact employees to build Synergy in a universal way, that can be used by both individual developers, and by an automated build agent as well. On a nightly basis the automated build agent creates a new version of Synergy that is published to a shared folder.

Currently the TFS system holds the most recent sources and generates a nightly version of Synergy, details on nightly builds, the sources available. This is a basic small set of features offered by TFS, though this is what minimally can be expected from a system performing automatic integration.

This minimal system has been evaluated with product management on how it can be extended. First the value of the existing features have been discussed, and a set of possible additional features have been discussed which have a potential of adding value. This set of


Build Details - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://036rd350-tfs1:8080/Build/Build.aspx?artifactMoniker=74
Go Links
Local intranet

Build build_1_20090730.1

Summary	✓ Succeeded
Build name:	build_1_20090730.1
Requested by:	EXACT-SOFTWARE\VALK247056
Team project:	Test Synergy Project Menno
Definition name:	build_1
Agent name:	My Desktop
Command-line arguments:	
Started on:	7/30/2009 10:23:41 AM
Completed on:	7/30/2009 10:24:08 AM
Last changed by:	EXACT-SOFTWARE\VALK247056
Last changed on:	7/30/2009 10:24:08 AM
Quality:	
Work items opened:	Not available
Source control version:	C244
Log:	\\valk247056-2\build\build_1_20090730.1\BuildLog.txt

Build steps ✓9 succeeded, 0 failed

	BuildStep	Completed On
✓	Initializing build	7/30/2009 10:23:45 AM
✓	Getting sources	7/30/2009 10:24:03 AM
✓	Labeling sources	7/30/2009 10:24:03 AM
✓	Compiling sources for Any CPU/Release.	7/30/2009 10:24:07 AM
✓	Project "TFSBuild.proj" is building "Exact.XML.Logistics.sln" with default targets.	7/30/2009 10:24:07 AM
✓	Project "Exact.XML.Logistics.sln" is building "Exact.XML.Logistics.vbproj" with default targets.	7/30/2009 10:24:07 AM
✓	Generating list of changesets and updating work items	7/30/2009 10:24:07 AM
✓	Copying binaries to drop location	7/30/2009 10:24:07 AM
✓	Succeeded	7/30/2009 10:24:08 AM

Figure 9.3: Detailed overview of the build, showing whether a build is successful

features was extracted from [43]. The question is whether the basic system functionality fulfills the needs of product management and where it can be improved.

From interviewing product management and a review of literature [31, 29, 30], it seems product management focuses on two types of information in its communication with development: 1) Whether the project is on schedule, and how to take action when the project is not on schedule. And 2) functional feedback on the features being developed.

This does not match with the information currently provided by TFS. A status update on compiling code only says something on the health of the code. Source code could provide some information on the functionality of the system, though according to product management this provides information on a too low level to be useful. The feature that lasts is the version of Synergy compiled on a nightly basis. This version is however difficult to access and install. Accessing the version requires a copy operation that takes long and often fails. Installing the version sometimes fails because of conflicting dll-versions. All in all there is space for improvement.

9.2 Iteration II: Publishing the Nightly Build

To define the goal for this iteration list of possible features to extend the current TFS system is composed based on book "Continuous Integration: Improving Software Quality and Reducing Risk", by Andrew Glover [28] and Martin Fowler's introductory text [43] (See also Chapter 8):

Tests Automated tests (unit-, functional- and integration- tests), run tests on classes and functions, they can serve multiple purposes. First of all automated tests can be used to give insight in the health of the code base: when tests fail there are bugs in the code. Secondly, automated tests can display progress of a project or feature: When unit tests are developed prior to functionality, the percentage of succeeding unit tests gives insight in the progress of the project.

Code Inspection Code inspection automates the process of source code analysis. Many different types of checks are available, to name a few: checks for programming style, security checks, checks for common coding errors, checks for code complexity, checks to detect need for refactoring, etc (See also [32]). This feature can help development to check the health of their codebase.

Deployment Software deployment is the releasing the software to a runtime environment. This environment can be production, but it can also be staging or testing: the point is that the software is being publish, and that other parties can use or play around with it. This feature is meant to automatically publish functional changes in software to other parties.

Since product management is most interested in functional information and information whether the product is on track, it seems that automated tests and deployment are most interesting; Code inspection is mainly of interest for development. Automated tests however would require a lot of effort from development to implement it: the ideal test/production code ratio is about 1:1, which means that the amount of code to be developed doubles [26] which would require a big change in the way software is developed in Exact. Therefore we focus on functional changes and try to ease the release process of code under development, and attempt to implement automatic deployment of the software.

9.2.1 Goal

Deploying the Synergy development version to a staging environment has several prerequisites:

1. A compiled Synergy version should be generated from sources
2. The version should be made available, which requires
    a) Storage
    b) A sharing protocol
    c) A retrieval method
3. A staging server

4. Software to automate the deployment

Even though the first item is already available as a result from previous iterations, these are quite some steps to perform in a 2 week iteration, and therefore we cut this goal into two sub goals: 1) Make the Synergy version publicly available, and find a method to install this version. 2) Set up a staging environment where the Synergy version is updated on a nightly basis.

For this iteration the goal is sub goal 1: To make a Synergy version publicly available, and find a method to install this version.

9.2.2 Approach

In this iteration storage, a sharing protocol and a retrieval method have to be found fitting within the global infrastructure of Exact to share the latest development version of Synergy. These three dependencies are interrelated: a certain sharing protocol probably requires a certain retrieval method; and a device having certain storage properties may not be able to support any sharing protocol.

This functionality must be easy to use for its end users, since previous attempts have been made with sharing the development version of Synergy which failed due to usability issues. These previous attempts failed since the method was too much hassle to setup and prone to network errors. Therefore the update method should be easy to use.

Within Exact two major data sharing methods are used to distribute software and documents: the company's E-portal, and windows network shares: these two methods will be analyzed for their usefulness for distributing the development version of Synergy.

E-Portal

E-Portal is Exact's internally used Synergy deployment. E-Portal has historically been used to share documents and software updates between employees and customers. It provides features for managing different software versions and tracking the history of files. E-portal works together with the SynergyProductUpdater to allow customers, to update their local Synergy installations with the most recent features and bug fixes.

Each new release and each bug fix are currently applied manually in source and binary form to the right releases of Synergy in E-Portal. There are few releases, and these updates are not done frequently.

Using E-Portal to distribute nightly Synergy versions has as an advantage that it integrates flawlessly with the updating method currently used by customers and product managers. Storing all nightly versions of Synergy raises a storage challenge. While E-Portal is able to store lots of data, storing a Synergy version on a nightly basis, and tracking its history would require an unnecessary amount of storage: Each version has to be stored separately, while only few of these versions will be used by a small amount of users.

Network share

Besides E-Portal, network shares are a common method to share data and documents within Exact. Data on network shares is not exposed to customers, though it is easy for employees to access network shares. Network shares act as a storage device over a computer network, without offering features of file history tracking. Network shares can be accesses trough well known protocols and the SynergyProductUpdater supports updates via network shares.

Network shares seem suitable for distributing local software updates, within Exact, without the waste of disk space that E-Portal usage brings along.

The goal of this iteration is to make the nightly development version of Synergy available through network shares, and allow users to update their local Synergy installation using the SynergyProductUpdater.

9.2.3 Intermediate evaluation

The goal of this iteration is to publish the nightly Synergy build to a network share and allow Exact employees to update their local Synergy installation to this version using the Synergy product updater.

The first step is to copy the nightly Synergy build to this network share. As shown in Figure 9.1 TFS team build can be configured to copy the resulting nightly build to a shared folder. Therefore the nightly build is copied each night to a centrally available network share.

The second step is to configure the Synergy product updater to update using this network share. An example can be seen in Figure 9.4.

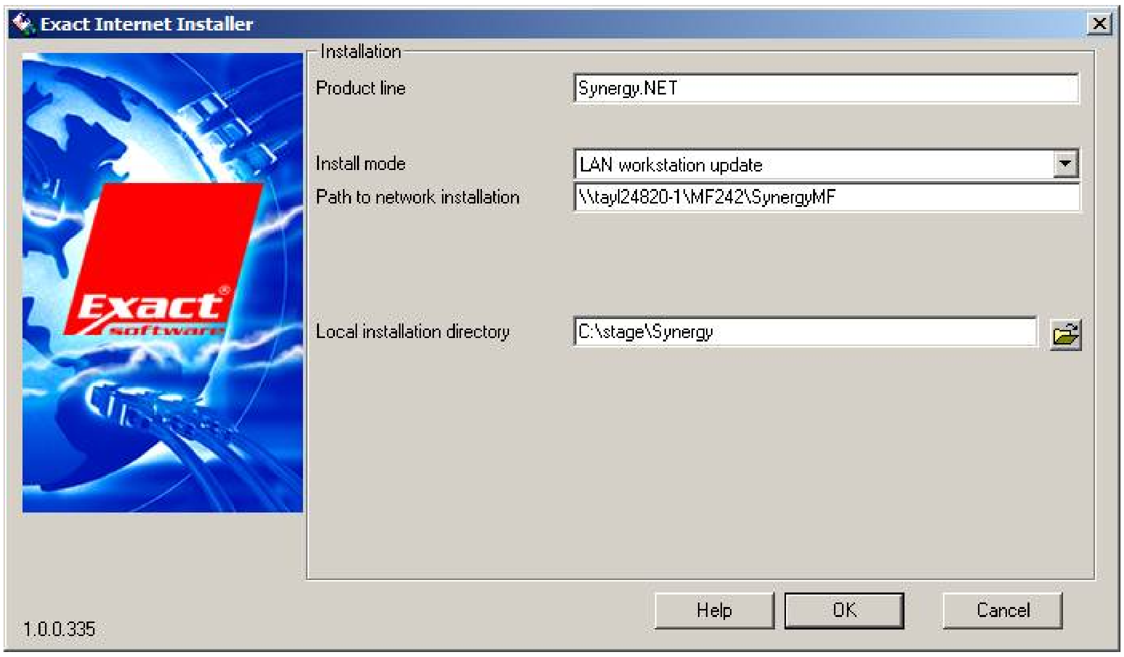
According to a discussion with some of the product managers: "This is exactly what we need". This could help in future projects to help to fasten the feedback loop. The update method helps to update the local Synergy installation relatively fast, without the errors in the previous methods. Unfortunately not all exact employees have the knowledge and time to setup a local Synergy installation and update it on a daily basis, therefore a central solution needs to be developed.

9.3 Iteration III: Deploy the Nightly Build

The solution developed so far requires users to create a local Synergy installation. Not each user has the knowledge to do this, therefore this iteration will be focused on nightly installing the Synergy version on a central place.

9.3.1 Goal

As defined in Section 9.2.1, in sub-goal 2: "Set up a staging environment where the Synergy version is updated on a nightly basis." is not yet fulfilled, and needs to be implemented. That's the goal of this iteration.



Product line: Synergy.NET

Install mode: LAN workstation update

Path to network installation: \\tayl24820-1\MF242\SynergyMF

Local installation directory: C:\stage\Synergy

Figure 9.4: SynergyProductUpdater configured to update using a network share.

9.3.2 Approach

To setup a publicly available version of the most recent Synergy version a server should be selected and software must be developed to periodically perform the software update.

Since the server should be publicly available for Exact employees, the server should have a good connectivity and be powerful enough to serve some users. Previous approaches failed because of slow connections or server environments. For our approach, a server at Exact Delft headquarters is selected with close proximity to product management.

The problem of installing the latest development Synergy version was addressed in the previous Section 9.2, and merely needs to be enhanced for scheduled software updates. The first step was to setup a normal Synergy installation on the server. This installation is updated by scheduling the SynergyProductUpdater to retrieve the latest version from the network share on a nightly basis.

9.3.3 Intermediate evaluation

This approach indeed installed the latest version of the Synergy product. However this approach comes with several drawbacks.

First of all Synergy is a highly configurable product: each Synergy installation can be setup in different ways. A Synergy installation can be configured to behave differently for each installation since each company is different. Therefore a one-size-fits-all installation, testing all functionality, does not exist. A solution for this problem is that special features are still tested out in an individual environment as described in Section 9.2.

A second problem is that the Synergy installation does not enable all users in the Microsoft Domain automatically; for each individual Synergy installation needs to be specified which users are allowed to login to the system. While any user can reach the system, the user's account should still be manually enabled in the staging environment before the user can try out the latest Synergy version.

By allowing users to install their own customized Synergy version and granting users individual access to the staging environment, users are able to try out the newest Synergy version to provide feedback. These technical enhancements make the Synergy development version available for anyone, these features should be used in an effective way.

9.4 Iteration IV: Large Scale Implementation and Process

All previous sections were on improving the tooling to improve the feedback loop between product management and product development. To effectively use the tooling product management should be notified functionality is completed and feedback is required: product development commits changes on a daily basis, and when a functional mile stone is reached product management is notified to give feedback in a meeting. In Chapter 10 this process is described in more detail.

9.4.1 Goal

The goal of this iteration is to verify the usefulness of the technical tools and the process in a real project at Exact. The previous iterations involved small scale trails with the developed functionality. Once the functionality was created a PM was asked to provide his view on it and how to improve.

Because in this iteration the process is tested out and not a single feature of the continuous integration, it should be tested on a larger scale with a real team.

9.4.2 Approach

The technical tools described above combined with the process described in Chapter 10 will be used for development of features for the upcoming Synergy release.

To test out the continuous integration tools and process in this iteration, the continuous integration tools need to be setup again for two reasons. For starters, all projects used for testing in the previous iterations have been completed, and new projects have started. All systems setup for previous iteration were used on a per-project basis, and therefore ran useless when new projects were started. Besides this, project management was enthusiastic about the continuous integration tools and the nightly development release; therefore product management wants to make this idea a Synergy wide standard and decided to roll it out on a large scale. For these two reasons the continuous integration tool chain needed to be rebuild, this time usable on a larger scale.

To implement continuous integration on a larger scale, multiple steps should be taken. All Synergy sources should be moved into TFS source control; Synergy should be build on a nightly basis from source control and released to product management as described above;

a branching methodology should be set up as described in [11] to allow multiple teams to work on multiple Synergy versions in the source control system; finally all projects should be using the version control system.

Move all Synergy sources into TFS

All sources should be moved into the same TFS system. In the previous iterations, each team had its own TFS repository; this TFS repository only consisted of the sources the project team actually worked on. When a software release was near, or functionality nearly completed, these sources were merged back into the e-Portal system. From a continuous integration perspective this is undesirable for multiple reasons. First of all, while the work of team members is continuously integrated, the work of different teams is only merged when the release date draws near. Second, in this way it is impossible to create a nightly development release consisting of all modifications: at best, each team gets their own nightly release. To resolve these issues all sources have to be copied into the same TFS repository.

Setup the continuous integration tool chain

After all sources are moved into the same TFS repository, the complete Synergy system should be build using these sources. It should be possible to build the entire set of sources at any given moment into a installable Synergy system to release a nightly development version to the product management team. For this step the experience gained in the previous iterations could be used. However, this step is more challenging than the one in the previous iterations for two reasons: More sources are build, therefore more powerful hardware is required. Second, building the entire Synergy system, has more dependencies between the different sources and therefore it is harder to configure the build environment to deal with this. Thus, to release a nightly version of Synergy the build tool chain should be setup on a larger scale.

Setup a source branching methodology

Having multiple teams developing and maintaining, different software versions in the same source control system, posses some challenges on proper version control. On one hand all modifications and features should be added to the same branch: this results in one development release in which all bug fixes and features can be found and are applied only once. On the other hand, some changes and software versions should be isolated to reduce the risk of major features, and separate development from bug fixes in software releases [11]. These contradicting goals should be addressed and managed well, in a suitable company wide branching methodology.

Development teams and product managers start using the tool chain and process

Once the infrastructure is setup and ready for use, the development teams should start using the continuous integration tool chain, and use the process to sent nightly updates to product management. To roll out the system on a large scale licenses are required: each installation of the TFS client requires a license.

9.4.3 Evaluation

Due to the decision to implement the TFS tool chain Synergy wide, this last iteration is more challenging than previous iterations. Due to this scope growth the assignment required more expertise and help from other parties within Exact. This section will discuss for each of the steps described above what the challenges were.

Moving all Synergy sources into TFS was no problem since the TFS server itself already existed, consisting of some small projects. The only step was to import all Synergy sources in a newly created project.

Setting up the continuous integration tool chain working for the entire Synergy project posed some challenges. First of all, while powerful hardware was already available, this hardware was not yet operational. Therefore our personal workstation was used as temporary build agent; later on this could easily be changed to more advanced hardware. Secondly, building Synergy from scratch required expert knowledge on the Synergy architecture and its dependencies. Our stay in Kuala Lumpur was to short to acquire this expert knowledge, and therefore this part of the project is handed over to an expert in Kuala Lumpur. At the time of writing the expert in Kuala Lumpur is still working on setting up the powerful hardware and resolving dependency issues.

Product development and product management, having somewhat contradictory goals, made it a challenge to come to a definite branching methodology to use within the Synergy project. Product development's goal is to reduce project risks by isolating different features and tasks in different branches; giving them the flexibility to make last minute decisions what features to put into which release. Product management on the other hand, wants to have a release which includes all features as soon as possible to see how all features will work together in the final release. A middle way in these goals would be that product development has different development branches for different feature sets, which they merge early and often into the main branch which is available for product management. However, these contradictory goals make that product development and product management are still negotiating on the definite branching methodology at the time of writing.

While all technical challenges (should) have been resolved in the previous stages, the implementation stage posses some licence issues. The TFS system has been used for some pilot projects within Exact ADC, therefore a logical step was to extend these pilot projects with new features. Now the time comes to roll the system out Exact wide, licensing issues come into play. TFS is a highly configurable system providing many features, making an Exact wide license a big investment, especially when considering Exact uses only a small set of these features. Therefore, at the time of writing, the discussion is still going on, whether to acquire this license or to look into alternatives.

Thus, each iteration has, on a small scale within one project, iteratively added functionality to the existing TFS implementation through an evaluation-improvement-loop. In the final iteration an attempt has been made to deploy this system Synergy wide, which is at the time of writing still a work in progress due to unresolved technical and licensing challenges. In Chapter 10 will be described how this tooling fits in the RDP.

Chapter 10

Continuous Status Updates in the RDP

In this Chapter will be described how the new tooling can be used within the context of the RDP and the process enhancements required to run projects and make effective use of continuous integration and status updates to improve the feedback loop between product management and product development at Exact. The enhanced process is fully compatible with the recently introduced RDP (See Chapter 3) and is merely a layer on top of it to enable people in the 'boat' to communicate and share ideas more easily. First the current (to be changed) process will be described, followed by enhancements in the process to allow for continuous integration and automated status updates. This section is followed by changes for specific parties involved.The chapter will conclude with a detailed step by step overview of the resulting process.

10.1 Current (to be changed) Process

The methodology currently used is the RDP as described in Chapter 3. RDP is an enhanced waterfall [49] method[1] and is designed to overcome the drawbacks of the waterfall method [5] and gradually introduce elements of SCRUM/Agile development [50] into the Exact Development Process.

On top of the RDP a semi structured feedback mechanism is used to handle the information flow from product development to product management, or more general share product versions with the "boat". After development is started, status updates to product management are conducted by weekly reports and occasional daily/weekly meetings. In any case the status updates are generated and updated manually by development. When the end of the project is nearing a release candidate is created and published for the first time to product management. Based on this version stabilization is started and feedback conducted.

Drawbacks of this approach lay in workload, continuity and timing. Each time a report or status update is conducted this requires manual actions from team members which costs development time and interrupts the individual developers work flow. Because each update

[1]Waterfall method was used in Exact before the introduction of RDP.

requires manual intervention it is hard to continuously update product management on the project status [37, 28, 36, 43]. Finally the timing: the first time product management gets hands-on-experience with the final product is at the end of the project life cycle which is rather late for feedback, since late product changes are the most expensive ones [17].

10.2 Enhancements Current Process

RDP has, like SCRUM, a high level process focus on the development process and does not do recommendations on development practices [6]. Continuous integration and updates is such a specific development practice (as in described in [37, 9, 56, 28]) and is therefore a possible addition to the Exact development process without changing the management level RDP methodology. The goal of continuous integration and feedback is to automatically generate status updates from the software repository[2]. Since status updates are generated from source code, the generation of automatic status updates starts once development is started and ends once the stabilization phase ends. Examples of status updates could be on build results, unit tests, code coverage, code quality or automatic system deployment. Automatic system deployment is the first pilot in a series improvements of continuous status updates.

The addition to enable continuous feedback is summarized in Figure 10.1. Product development continuously integrates their work by committing and retrieving sources to and from the software repository (“Continuous integration”). The status updates are created from the sources and automatically made published (“Continuous-status-updates). Product management is able to review the latest software version and discuss any remarks and acknowledgments with software development (“Continuous-feedback”).

This process differs from the current development cycle in that the status update step is automated, and naturally allows for a high frequency of feedback. The higher frequency of status updates has implications for the parties involved: product development and product management.

10.3 Main Stakeholders

This section will cover enhancements for specific stakeholders in the boat: administrator, product management and product development. While the boat consists of more stakeholders than just product development and product management, the focus is on improving communication between Delft and Kuala Lumpur. The stake holder in Delft is product management and the rest of the stakeholders are in Kuala Lumpur grouped together as “Product Development”.

[2]Note that the source repository is TFS, however the process description has to be as technique independent as possible, and it could be that TFS is replaced by another software package with similar functionality.

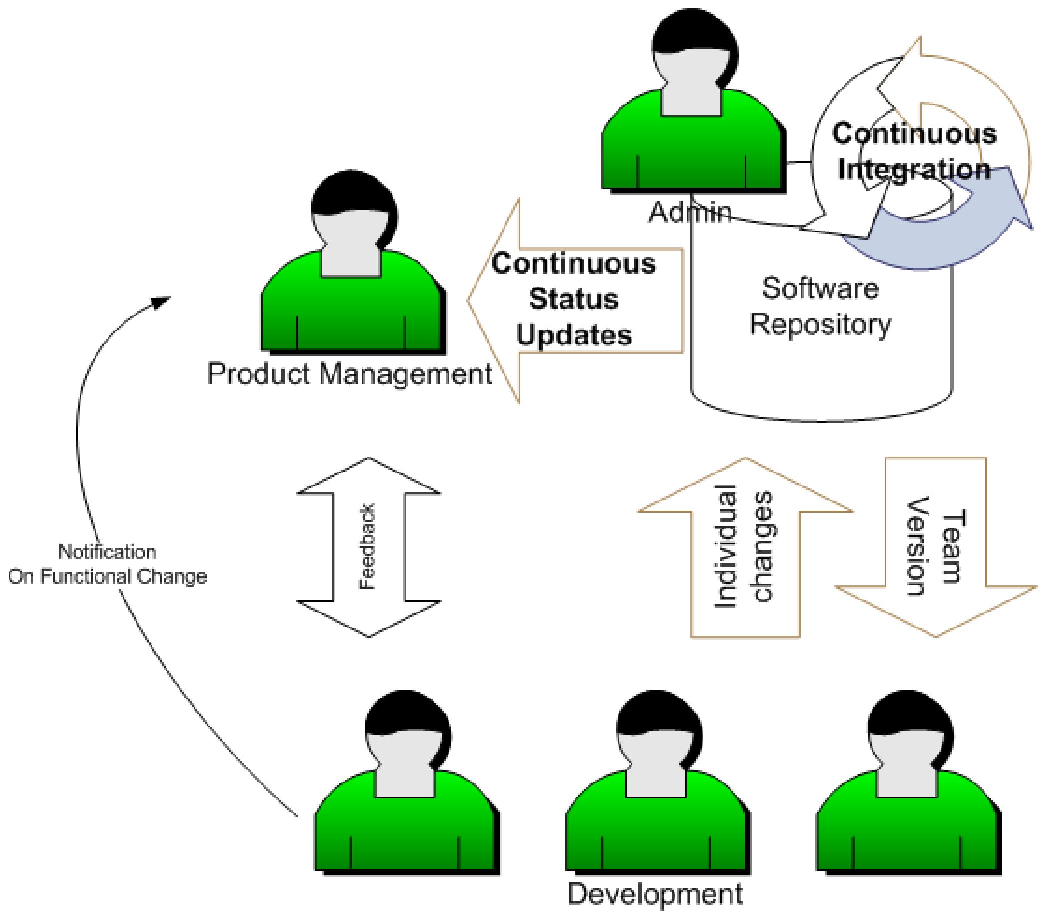


Figure 10.1: Process overview enhanced feedback loop

10.3.1 Administrator

The administrator has no role in the software development cycle itself, however its role is on the maintenance of the software repository, continuous integration process, and the continuous updates. The responsibility is to keep the software repository running: resolve issues and do maintenance. Currently the role of administrator is with the author of this document and a person in the development team. Reason for this is that the system TFS is part of a pilot which is not done on a corporate level.

10.3.2 Product Development

The prerequisite for continuous integration and continuous deployment is that the programmers publish work early and often and update with each others work early and often. If work comes into the version control system early, and development updates their software with version control often it means that their work in progress is close to that of the other people in the team and integration errors can be detected early. Therefore development should commit and update their work at least once a day [37, 28]. Currently some of the projects already use TFS and some form of continuous integration as pilot projects. Thus some software engineers are already used to early publishing their work. Software engineers should keep releasing their work early and often and aim to commit their work at least once a day. To allow other team members to integrate their code, detect errors early and allow for early status updates.

Since product management is mainly interested in functional information like some functional logic, screens, and completed use cases, product development should start working on these features early in the project life cycle to allow product management to give feedback even more early. A meeting on this new functionality with product management should be conducted regularly: at least once per sprint: the end of a sprint would be a logical choice since at the end of each sprint there should be a deliverable product [9, 56, 50]. Sprints should be at most 4 weeks however 2 weeks is preferable [9, 56, 50] this means these meetings are conducted at least once in 4 weeks, however preferably more often, maybe even multiple times per sprint. Thus moments to initiate feedback can be: at the end of a sprint, when functionality is complete: e.g. a use case or screen is completed or a project mile stone[3] is reached: feature complete or stabilization phase complete.

Early software versions are published to product management, this means more exposure. Sharing work early and ofter will result in exposure of features in the earliest stages of development without following the quality assurance procedures that are in place for major software releases. Product management will be able to see all bugs and unfinished functionality. Product development might feel unconformable on this[4]. Product development should however resist the tendency to release their work when it is 'perfect', since this would stand in the way of the early integration with the work of other software engineers and realistic status information for product management. Product development and product management should develop a level of trust through early releasing [27] to overcome the foreignness of the increased in exposure.

The early publication of work allows other parties in Exact, mainly product management, to provide product development with feedback early and often. Product development can be overwhelmed by this feedback. The feedback can be processed when received or buffered in a issue tracking system like the ones already included in e-portal and TFS. The feedback should be managed, prioritized and processed.

10.3.3 Product Management

For product management the continuous availability of new software versions allows a different style of working. Project status is discussed by product management and product development in conference calls. Topics covered in these conference calls are technical problems, plannings issues, and state of functionality. A subset of these functionality should now be communicated to product management in the form of continuous status updates.

[3]Note that this is not frequent and thus is not preferred.

[4]During the interview sessions one of the development managers mentioned that early releasing software might give product management a bad impression of product development.

With continuous status updates available the question arises when product management should review these. Daily review would result in a work overload for product management and an information overload for product development. Product management does not require to examine each update: differences might be too small and insignificant on a daily basis. Review from product management should be requested by product development when functional milestones are reached: completed screens and working use cases.

In the case of continuous deployment, daily availability of software that is under construction requires a different mindset for product management. In the past software was passed to product management when the end of the project cycle was near and the stabilization phase about to start. Features were already tested and polished by development. In contrast the daily development version is under constant development, bugs and runtime errors might exist and debugging information present. Early detection of errors works two ways a) software development is able to detect and resolve issues more early, b) these bugs might also be exposed to product management. Features that are under construction might have some missing functionality, screens can look rough, in some cases crash. Debugging information might makes software unnecessary verbose. To set the right expectations for product management when reviewing the system, feedback should be given based on status report: in this way product management knows whether features are tested/debugged yet or still in heavy development. Besides this product management should get used to inconveniences in the software versions, and develop a radar for inconveniences that do require steering to development and trust development to resolve the other issues without explicit request.

10.4 Resulting Process

In Figure 10.2 the process is displayed in detail. The Administration stakeholder will not be discussed here, since its role not that complex.

10.4.1 Product Development

An individual developer starts its iteration with downloading the sources from the source repository, this can be at the beginning of the day or at the beginning of implementing a functional change; this has to be done at least once a day. This allows him to integrate his current work with the sources in the repository.

Once the sources are downloaded, changes can be applied to these sources based on specifications, existing bug reports or previous feedback sessions.

At any moment an individual developer can decide to share its changes with the other developers by checking them into (committing) the source repository. When the sources are committed development decides whether a functional milestone is reached, if this is the case product management is requested to review this milestone.

During the feedback session the current status of the project is discussed: Bugs, possible changes in screens, use cases and business logic are suggested and possibilities explored with product management.

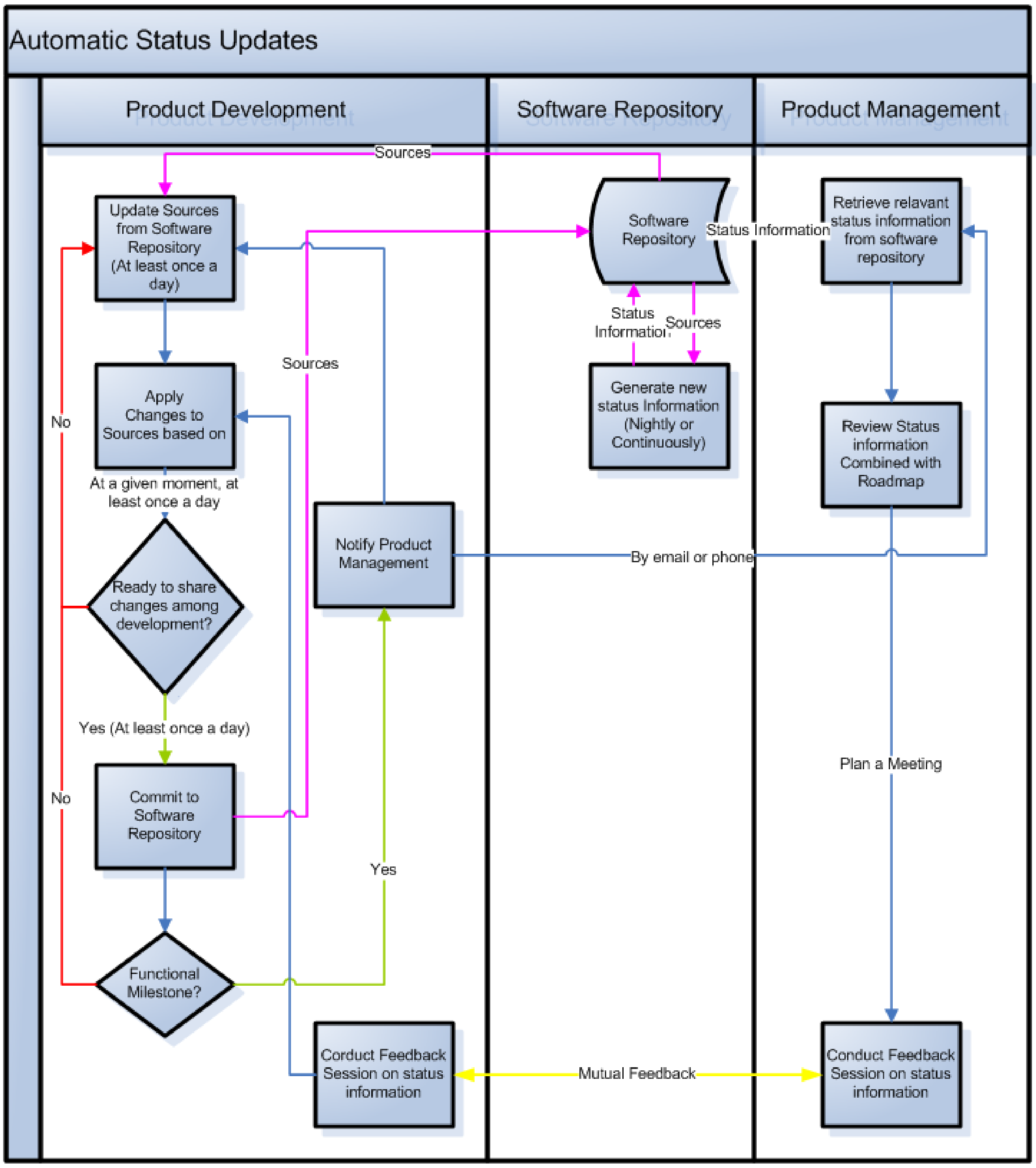

Automatic Status Updates
Product Development
Software Repository
Product Management
Sources
Update Sources from Software Repository (At least once a day)
Software Repository
Status Information
Retrieve relavant status information from software repository
Status Information
Sources
Sources
Apply Changes to Sources based on
Generate new status Information (Nightly or Continuously)
Review Status information Combined with Roadmap
No
At a given moment, at least once a day
Ready to share changes among development?
Notify Product Management
By email or phone
Yes (At least once a day)
Plan a Meeting
No
Commit to Software Repository
Yes
Functional Milestone?
Corduct Feedback Session on status information
Mutual Feedback
Conduct Feedback Session on status information

Figure 10.2: Detailed feedback process

After the feedback session product development continues applying changes to the sources based on the feedback of the meeting.

10.4.2 Software Repository

The software repository's task is to store software versions and generate- and publish- status information. The generation of status information is done nightly or when changes are received. Since product management does not require instant updates on software progress, nightly status updates will do, however it does not hurt to generate status updates instantly when software change sets are received from development.

10.4.3 Product Management

Once product management receives a request for feedback on a functional milestone or prepares himself for a meeting, He has to receive the status information manually from the repository.

The received status information is compared to the reported status of the project. This is to set the right expectations and to prevent product management to give feedback on features that are known to be incomplete or consist of bugs. Based on this review a feedback session, in form of a conference call, with product development is planned.

During the feedback session the current status of the project is discussed: Bugs, possible changes in screens, use cases and business logic are suggested and possibilities explored with product development.

As described in Chapter 9 this process has not yet been tried out on a large scale. While it would have been interesting to study the effects of this process on a large scale, some conclusions can be drawn and recommendations be given.

Chapter 11

Conclusions and Recommendations

This chapter gives an overview of the project's contributions. We will reflect on the results and draw some conclusions; recommendations on future research and future work will be given; and finally an evaluation of our research approach will be done.

11.1 Conclusions

Conclusions can be drawn in three areas: First of all, on the challenges faced in Exact's development process. Secondly, conclusions can be drawn on how to address these challenges. And finally, what have we accomplished by addressing challenges with automated status updates?

11.1.1 Challenge Identification

In the first part of our research we identified challenges that Exact faces to get a more agile development process. We narrowed down our initial goal to investigate tooling to make the development process more agile. Major challenges lay in: A) the communication between two major parties in Exact's development process: product development in Kuala Lumpur and product management in Delft. And B) extending agile practices in a large scale software development organization. Addressing both of these challenges we investigated how the current use of TFS could be extended to improve communication between product management and product development and to get continuous integration on a global, company wide level [43, 28]. Based on literature research [48, 29, 30] and interviews within Exact, we found out that most communication between product management and product development is on project progress and software functionality. Problems discussed are whether the project is on track and how to resolve planning issues, and whether the functionality delivered is correct.

11.1.2 Automated Status Updates

Status updates that can be provided by an continuous integration system are results on: software tests, code inspection, and intermediate releases [43, 28]. Code quality checks are mainly of interest for development itself. For product management, the percentage of succeeding software tests can give insight in the progress of the project and an intermediate release can give insight in the completed functionality of the system. Software tests have as a drawback that these should be actively developed and maintained by product development and therefore require quite an investment. In contrast, intermediate development releases are readily available in the source control system. Therefore the most valuable status update with minimal investment from Exact is to frequently publish an intermediate development release to product management. During our research we have set up a prototype system: create a nightly build, publish build results, implemented a software update procedure, and finally deployed the system on a nightly basis to a staging environment.

11.1.3 Accomplishments

At the time of writing no large scale tests with this system within Exact have been performed. For this reason it is hard to say what the impact of our work has on communication, collaboration, software quality, transparency of the development process, and development speed in the long run. However, during the intermediate evaluation of the functionality delivered during sprints, both product management and product development were enthusiastic about the try outs with our prototype. Reactions from product management varied from “This is exactly what we need” to “I can’t wait to have this working”. Thus the small scale prototype has shown that it is possible to improve the feedback loop between product management and product development, and the initial responses were positive.

Product development went through a shift in perspective: when we first proposed our ideas on a more transparent development process and early releasing, an example of the responses was “product management may gets a bad impressions from development when we release untested software”, shifting to support and proposals of own ideas on the best approach to improve the continuous integration system. On top of this, Exact wants to make the ideas presented during our research, a Synergy wide standard and at the time of writing people are working on setting up this system on a large scale.

11.2 Recommendations

Based on our research we would like to give some recommendations for Exact, and future scientific research.

11.2.1 Exact

Based on research results and our experiences within Exact we would like to do some recommendations.

Evaluate the effects of continuous integration system

During our research it was not possible to evaluate the system’s use on a larger scale for a longer period of time. While the initial responses of the parties involved were enthusiastic, the question remains what the system's effects are on the development process and the software quality. This should be evaluated at a later point in time. Evaluation can be done using interviews with some key parties involved, doing a survey with a large group of people using the system, or by analyzing the bug rate and the number of features that are implemented in the next release. Based on this information can be decided whether it is worth to invest more in this continuous integration system.

Store sources in a single repository

Source code in Exact is controlled in multiple repositories: the Exact Synergy Database (ESD) and the Team Foundation Server (TFS) repositories. The main reason for having two repositories are the needs of the two major stakeholders: product management favors ESD from a business perspective, and product development favors TFS for its support for development practices.

Historically the ESD is used to store revisions of source code and revisions of binaries. The source code is retrieved and updated by product development to resolve bugs and develop new functionality. The binaries are compiled from the sources and placed in the ESD to be distributed in sets as product updates and product releases among stakeholders. Stakeholders can be customers, 3rd-parties developing products based on Synergy, or Exact employees themselves. Thus mainly from a business perspective ESD is a convenient tool to distribute software versions among 3rd parties.

Recently projects started using TFS repositories. Reasons for this are the development oriented features offered to the team. Examples of these features are: integration with Visual Studio, ease of branching and merging releases, support for multiple developers to work on the same sources, and support for team builds based on the shared sources in the repository. When a project is started, sources are often copied from ESD into a TFS repository. Thus from a development perspective the TFS system is a convenient tool to reduce risks and support development practices.

Project development is done in a per project TFS branch. These branches are created from the main TFS development line. Maintenance such as bug fixes is done using the sources in ESD. When a product release is done, these sources need to be merged together and finally stored in the ESD. First all relevant projects need to be merged back to the main line and secondly the bug fixes in the ESD need to be merged into the TFS mainline. Finally the TFS main-line, is merged back into the ESD.

To avoid unnecessary merging steps in the release process the number of repositories should be reduced to one. Since TFS offers many of the features to support agile development, all sources should be managed in TFS only, without the clutter of merging them back into ESD. The only use for ESD is the, periodic (nightly, monthly...), distribution of the binaries for product updates and releases.

Thus: Use TFS or another suitable software package for management of shared sources and team builds, and use the ESD to retain backwards compatibility with the current software release/update system to stakeholders.

Synergy should be build from scratch

One of the goals of the software building process is to have a reproducible way to verify whether the source code of the software is in a deliverable state [28]. Ways to verify this are to run unit tests, regression tests or static code analysis, however the initial step is to check whether the software compiles.

Due to circular references between sources and the Visual Basic compiler's inability to deal with this, Synergy requires the binaries of the former release as dependencies to compile the current version. This approach has several drawbacks: A) This process, being repeated for many releases, makes it hard to verify whether the binaries resulting from a compilation run are a representation of the sources in source control. B) It makes the compilation process more complex since it has the unnecessary dependency on the previous Synergy release.

By resolving the circular references the software could be compiled without the inclusion of unnecessary libraries in a single compilation step allowing the software to be build from scratch.

Thus: Resolve circular references and make the process of compiling Synergy from scratch an easy to perform task, enabling software development to easily verify the correct state of the sources.

Extend continuous integration features

For this thesis we extended continuous integration mainly from a product management perspective (releasing product versions), continuous integration can be used for more than that.

The use of continuous integration can be extended to execute and publish status updates on: unit tests, functional tests, regression tests, integration tests, automated code reviews, and security checks. While automation of these tasks are not as useful for product managers as they are for product development, we think development time can be saved by automating these tasks, since many of them are currently performed manually.

Thus: Extend the use of continuous integration to repetitive and time consuming tasks, saving time and reducing errors.

Use continuous integration to support close customer involvement

Distribution of the Synergy development version is not limited to Exact internally. When both product development and product management are accustomed to the continuous deployment of software versions, the use of this feature might be extended. Currently efforts are made within Exact to get more customer involvement in the distributed requirements engineering process [60]. Thus, this is a point where both assignments may come together again, however this is out of the scope of this research. Here the continuous deployment might be extended to involving customers in the development process by releasing selected development versions of Synergy to a small group of customers.

Look into TFS alternatives for a continuous integration tool chain

Despite the obvious benefits from a development perspective it is not yet sure whether TFS will be implemented company wide, due to high licensing costs. Since TFS offers many advanced features not used by Exact or are already available in Synergy (e.g. work items), it might be an idea to look into alternatives to TFS. Many freely available tools offer similar features as TFS, like source control, continuous integration and Visual Studio integration[1] without the high licensing costs.

Extend continuous integration to existing products and make it a must for future products

In our research we narrowed our focus to the globally distributed development process of Synergy; specifically the communication between product management and product development. Continuous integration can be applied to development of any software product, and as seen in our initial results, can add value to software development in a global company. Companies wanting to introduce continuous integration to their development practices face challenges regarding legacy systems.

Legacy systems are often limited by their design or technology choice to support automated builds, tests, and continuous deployment. Furthermore, it can be a challenging task to develop unit tests for all existing code. Therefore, support for unit tests and continuous integration should be introduced starting with new features being developed. New products can be developed with continuous integration in mind: by technology choice, the design, and disciplined development practices. New technologies, such as cloud computing, make it easier than ever to continuously release and deploy new software features to customers or a selected group of beta testers[2]. A new software product can be designed to make automated testing possible, some argue that this practice improves product design as a whole [37]. By practicing unit test based software development from the birth of a software product, software engineers can develop and maintain automated unit tests for each feature that is newly developed, resulting in a test set to verify the validity of a whole product, to support a short release cycle.

Therefore legacy products should be maintained with continuous integration in mind and where possible introduce support for continuous integration and testing; new systems should be designed and maintained with continuous integration in mind from scratch.

11.2.2 Future Research

This section will describe areas where scientific research is required.

Most agile documentation is on small scale co-located projects, for example [9]. Some case studies are available on distributed use of agile methods, for example [55]. However to our best knowledge there is no structured overview available on how to implement agile in a large scale distributed context. First of all the questions arises whether agile is suitable for

[1]An example of such an alternative could be, SVN (version control) + CruiseControl.NET (continuous integration tool chain) + ANKH (Visual Studio integration of SVN)

[2]Flickr.com for example, on a good day, releases a software update every 30 minutes [28]

large scale distributed software development? Other information that should be covered in this literature would be agile practices and what their benefits are in a large scale distributed environment? What agile elements, if any, are suitable for usage in large scale distributed software development? What challenges arise and how these can be resolved? What is the effect of a distributed working environment on trust, and how to retain a high level of trust between two distributed sites? Based on this information it should be possible to make a selection on which agile elements to implement in what order in a company.

Most literature on agile development is aimed at software projects for one or a couple specific customers [9]. For our project it would be interesting to see what agile methods apply to packaged software, where one software package has to suit thousands of customers. To the authors best knowledge little of such research is available.

11.3 Evaluation

In this section I will reflect upon my time abroad, describe what challenges arose during my research and I will evaluate how these challenges could have been avoided.

11.3.1 Working abroad

Doing research on globally distributed software engineering within a commercial environment has been a great experience, especially because I had the chance to work in Exact ADC, Kuala Lumpur: exploring a new culture, and making a foreign city my home were both a rewarding challenge. The atmosphere within ADC is great: people are really willing to make the best product, and they manage to combine this with many social activities out of office hours.

What I found the most surprising was the effect of working distributed on trust [19]. Both in Delft and Kuala Lumpur the word “they” was often used when talking about one another, while working for the same company on the same product. During my early phone calls with KL it was hard to assess what is going on, how things are working, and why. Especially when you have never been there, it can be hard to understand why things are done in certain ways (seemingly suboptimal), however when getting to know the people and processes, I realized the approach taken is often the most suitable one. The challenge is to create this understanding in a distributed environment.

What I recognized from literature was the effort people have to put in communicating their ideas. New features were specified in great detail, making sure no implementation mistakes were made, later on supported by long conference calls on functionality and planning.

Where I expected cultural differences to be a major challenge, this turned out to be easy to deal with. Some things were indeed different: For example, a Friday night appointment, in KL seems more an indication of what might happen on that specific Friday, where this is more a definite thing in The Netherlands. However, when aware of such things, these do not raise major challenges.

For my assignment the challenges were mostly on organizational matters to implement my prototype.

11.3.2 Challenges

I experienced challenges in the following areas: A project proposal was delivered late in the project; since my assignment was to improve the development process from a development perspective, my time in Kuala Lumpur was far more valuable than my time in Delft; and finally a large(r) scale test of my prototype could not be tried out within the time of my research.

Late project proposal delivery

My initial assignment did not consist of a clear objective; it was broad. To scope down this goal to a proposal, the first three to four months of my research were spend on a broad literature study and interviewing Exact employees; this is a lot of time, not being spent on assignment specific research. While, for me, the ideal situation would be to have a concrete problem definition upfront to my research. Though, this is not always possible. Therefore the second best situation would be to get to a specific proposal earlier in the process.

Two things were standing in the way of getting to a proposal soon. Firstly, it takes time to take on the expert role within a company, because it is initially unknown which practices are within a company by design, and which practices are there because no research was ever done in that area. Secondly, people need time to get accustomed to an idea before becoming enthusiastic about it and support your views.

While continuous integration was an obvious agile practice lacking within Exact, it took time to understand why it was not yet there. When I saw why it was not there, I had yet to get started with exploring the possible benefits for Exact. In future research it might save time to take up the expert role in a more early stage, and start exploring the possible benefits right away.

During my early interview sessions I tried to find validation of my ideas on continuous integration upfront to the proposal. Responses included “This has been tried out”, “We’re using TFS already”, “This is supported by Synergy”, and “We want to test software before releasing it to product management”. As my research progressed and early prototypes were created, people got a chance to get a feeling for what continuous integration is and became more enthusiastic. Thus, by showing people what could be done, support came later in the process.

Combining these two ideas: In future projects I may take up the expert role early on by focusing on added value of new ideas, and getting support by showing how ideas work in practice.

Time spent in Kuala Lumpur

Our goal was to address globally distributed development challenges within Exact from a development perspective. To identify and address these challenges, the best place to go is development itself in Kuala Lumpur. However, our initial planning was to explore Exact in the first two months in Delft and then move to Exact ADC. While the first month in Delft was useful to get to know Exact, the second month we already started interviewing project managers and exploring Exact ADC. This second month would have been spent way more effective in KL, since in KL itself we got the most important input for our project proposal. Thus in future research, we have to get to our problem area as soon as possible.

Large scale test

To get results on the effects of our prototype, a large scale implementation would have been useful, since now we only have initial responses from a small group of Exact employees. Setting up a large scale test posed multiple challenges: Firstly, people's focus was naturally more on completing existing projects, then on implementation of our prototype. Secondly, during the development of our prototype we mostly worked in a staging environment instead of the production environment, which only verified our implementation on a technical level, not on its effects in the development process. Furthermore, the planning of our assignment was out of sync with the project planning, therefore it was hard to integrate our prototype in the later stages of development.

In future projects it is useful to get support for our prototypes early on, and integrate the system with the production system as soon as possible. In this way the evaluation can start immediately on a larger scale, and no time and effort is wasted on maintaining a production and staging environment. By integrating the prototype early on in the production environment, tryouts can start right away, avoiding the challenges of having a big bang integration being out of sync with the existing projects. Later on this system can be extended to company wide usage.

Bibliography

[1] Splint. http://www.splint.org/.

[2] Team foundation server documentation. http://msdn.microsoft.com/en-us/library/ms181232(VS.80).aspx.

[3] Watin. http://watin.sourceforge.net/.

[4] *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[5] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. Agile software development methods - review and analysis. Technical Report 478, VTT PUBLICATIONS, 2002.

[6] Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. New directions on agile methods: A comparative analysis. *Software Engineering, International Conference on*, 0:244, 2003.

[7] Alain Abran, Pierre Bourque, Robert Dupuis, James W. Moore, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2004 version edition, 2004.

[8] Robert D. Battin, Ron Crocker, Joe Kreidler, and K. Subramanian. Leveraging resources in global software development. *IEEE Softw.*, 18(2):70–77, 2001.

[9] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, October 1999.

[10] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001. http://agilemanifesto.org/.

[11] Stephen P. Berczuk and Brad Appleton. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[12] Robert V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[13] Barry Boehm. Get ready for agile methods, with care. *Computer*, 35(1):64–69, 2002.

[14] Barry Boehm and Richard Turner. Management challenges to implementing agile processes in traditional development organizations. *IEEE Softw.*, 22(5):30–39, 2005.

[15] Barry W. Boehm. Software engineering economics. pages 641–686, 2002.

[16] Barry W. Boehm and Kevin J. Sullivan. Software economics: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 319–343, New York, NY, USA, 2000. ACM.

[17] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, August 1995.

[18] E. Carmel and R. Agarwal. Tactical approaches for alleviating distance in global software development. *Software, IEEE*, 18(2):22–29, 2001.

[19] Erran Carmel. *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall, December 1998.

[20] Erran Carmel and Paul Tjia. *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, June 2005.

[21] Alistair Cockburn. *Crystal Clear: A Human-Powered Methodology for Small Teams (The Agile Software Development Series)*. Addison-Wesley Professional, October 2004.

[22] Alistair Cockburn and Jim Highsmith. Agile software development: The people factor. *Computer*, 34(11):131–133, 2001.

[23] Eoin O Conchuir, Helena Holmstrom, Par J. Agerfalk, and Brian Fitzgerald. Exploring the assumed benefits of global software development. *Global Software Engineering, International Conference on*, 0:159–168, 2006.

[24] Ian F. Darwin. *Checking C programs with lint*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1986.

[25] Tom DeMarco and Barry Boehm. The agile methods fray. *Computer*, 35(6):90–92, 2002.

[26] Arie Deursen, Leon M.F. Moonen, A. Bergh, and Gerard Kok. Refactoring test code. Technical report, Amsterdam, The Netherlands, The Netherlands, 2001.

[27] Kevin Dullemond, Ben van Gameren, and Rini van Solingen. How technological support can enable advantages of agile software development in a gse setting. *Global Software Engineering, International Conference on*, 0:143–152, 2009.

[28] Paul Duvall, Steve Matyas, and Andrew Glover. *Continuous integration: improving software quality and reducing risk.* Addison-Wesley Professional, 2007.

[29] Christof Ebert. The impacts of software product management. *J. Syst. Softw.*, 80(6):850–861, 2007.

[30] Christof Ebert. Software product management. *Crosstalk*, 22(1):15–19, 2009.

[31] Samuel Fricker, Tony Gorschek, and Martin Glinz. Goal-oriented requirements communication in new product development. In *IWSPM '08: Proceedings of the 2008 Second International Workshop on Software Product Management*, pages 27–34, Washington, DC, USA, 2008. IEEE Computer Society.

[32] Sayed Ibrahim Hashimi and William Bartholomew. *Inside the Microsoft Build Engine: Using MSBuild and Team Foundation Build.* Microsoft Press, 2009.

[33] James D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *FOSE '07: 2007 Future of Software Engineering*, pages 188–198, Washington, DC, USA, 2007. IEEE Computer Society.

[34] J. Highsmith and A. Cockburn. Agile software development: the business of innovation. *Computer*, 34(9):120–127, 2001.

[35] Geert Hofstede and Gert Jan Hofstede. *Cultures and Organizations: Software of the Mind: Intercultural Cooperation and Its Importance for Survival*, volume 2nd. McGraw-Hill, New York, 2005.

[36] Jesper Holck and Niels Jrgensen. Continuous integration and quality assurance: a case study of two open source projects. *Australasian Journal of Information Systems*, 11(1), 2007.

[37] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, October 1999.

[38] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[39] G. L. Kelling and C. M. Coles. Broken windows: The police and neighborhood safety. *The Atlantic Monthly*, March, 1982.

[40] Jyrki Kontio, Magnus Hoglund, Jan Ryden, and Pekka Abrahamsson. Managing commitments and risks: Challenges in distributed agile development. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 732–733, Washington, DC, USA, 2004. IEEE Computer Society.

[41] Mikael Lindvall, Dirk Muthig, Aldo Dagnino, Christina Wallin, Michael Stupperich, David Kiefer, John May, and Tuomo Kahkonen. Agile software development in large organizations. *Computer*, 37(12):26–34, 2004.

[42] Mark Lycett, Robert D. Macredie, Chaitali Patel, and Ray J. Paul. Migrating agile methods to standardized development practice. *Computer*, 36(6):79–85, 2003.

[43] Matthew Foemmel Martin Fowler. Continuous intergration. 2005.

[44] Maria Paasivaara and Casper Lassenius. Could global software development benefit from agile methods? *Global Software Engineering, International Conference on*, 0:109–113, 2006.

[45] David Parsons, Hokyoung Ryu, and Ramesh Lal. The impact of methods and techniques on outcomes from agile software development projects. pages 235–249. 2007.

[46] Balasubramaniam Ramesh, Lan Cao, Kannan Mohan, and Peng Xu. Can distributed software development be agile? *Commun. ACM*, 49(10):41–46, 2006.

[47] Donald J. Reifer. How good are agile methods? *IEEE Software*, 19(4):16–18, 2002.

[48] Suzanne Robertson and James Robertson. *Mastering the Requirements Process*. Addison-Wesley Professional, August 1999.

[49] Walker W. Royce. Managing the development of large software systems: concepts and techniques. *Proc. IEEE WESTCON, Los Angeles*, pages 1–9, August 1970.

[50] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[51] R.W. Selby, editor. *Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management and Research.* Wiley-IEEE Computer Society., 2007.

[52] Ahmed Sidky, James Arthur, and Shawn Bohner. A disciplined approach to adopting agile practices: The agile adoption framework, 2007.

[53] Joel Spolsky. The joel test: 12 steps to better code, 2000. `http://www.joelonsoftware.com/articles/fog0000000043.html`.

[54] Linda Stroh. *The Basic Principles of Effective Consulting*. L. Erlbaum Associates, Publishers, Hillsdale, 2006.

[55] Jeff Sutherland, Anton Viktorov, Jack Blount, and Nikolai Puntikov. Distributed scrum: Agile project management with outsourced development teams. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 274a, Washington, DC, USA, 2007. IEEE Computer Society.

[56] Kevin Tate. *Sustainable Software Development: An Agile Perspective (Agile Software Development Series).* Addison-Wesley Professional, 2005. (ISBN 0321286081).

[57] Dan Turk, Robert France, and Bernhard Rumpe. Limitations of agile software processes. In *In Proceedings of the Third International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002*, pages 43–46. Springer-Verlag, 2002.

[58] van Solingen and E. Berghout. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill, 1999.

[59] Christian Visser. Assignment definition meeting report. 2009.

[60] Christian Visser. Customer involvement in distributed requirements engineering. 2009.

[61] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[62] Robert K. Yin. *Case Study Research: Design and Methods (Applied Social Research Methods)*. Sage Publications, Inc, December 2002.