Title page

Contents

Introduction

What is this project about

Short introduction of overall idea and how we plan to implement it

What are the contents of this report

Stating all parts of report plus connecting them to our project

THEORY

Plan driven vs Agile development

Plan driven

What is

A way to organize activities and artifacts which will lead to the production of a finished software system. Requirements and plans defined from the start of the project

Includes what

Waterfall

What is

Very basic development model. First specify requirements, then develope, finally test and deliver.

Use when requirements are known up front and easy to understand. And where we expect few changes

Pros

Predictable

Defines the basic terms in system development: Requirement, design, implementation and test

Is structured (detailed plans, fixed timeframe, etc)

Good visibility

Good documentation

Cons

Requires known requirements in the beginning of the process

Changes are hard to implement

Since testing is done in the end, errors are discovered late (no room for errors)

UP

What is

Incremental software development process. First gathering requirements, then making models, finally coding then releasing.

Pros

Fixed tasks

Easy for beginners

Cons

Agile

What is

Iterative development, lots of interactions with customer – good feedback, team based – close communication. Requirements and plans are created during the entire project

Since changes are allowed, fixed price products are a challenge

Includes what

XP 02-XP

What is

4 values, 12 principles

The principles are connected, if one is omitted then the whole suffers

Values: Communication – lack of comm produces many problems

Simplicity – neccessary for making changes

Feedback

Courage

XP Practices: Design: Simple desing, refactoring

Coding: Pair Programming, Test First, Continuous Integration, Collective Ownership, Coding Standards

Development method in which very minimum is coded first, so the program works, only then more complex elements are added. XP uses TDD, meaning that test is created to test code before actual implementation of code is made.

Extreme programming is a software development method based on 4 values and 12 principles. Part of XP are small releases, simple design, Test driven development (TDD) and continious integration. Four values (communication, simplicity, feedback and courage) are used for implementing XP. To make sure that errors are spotted fast and all the best implementation ideas actually get implemented, good communication in team is essential. Many of XP practises also require communication between team members. Software releases in XP are small. It is necessary for further code development. It is way simpler to implement changes to small and easy-to-read piece of code, than to complex hardcoded piece of software.

Pros

Cons

SCRUM 03Intro-scrum

What is

Project Backlog, sprints(1-4 weeks), scrumMaster, doesn’t say anything about how to develope code, product owner, sprint backlog, daily standup meetings, 3+3+3 (3 roles, 3 ceremonies, 3 artifacts).

Roles – product owner, scrum master, team

Ceremonies – sprint planning and review, daily meetings, sprint retrospective

Artifacts – product backlog, sprint backlog, burndown chart

The whole idea of Scrum is to forge selforganizing teams that continue to learn

Pros

Cons

KANBAN 05KANBAN

What is

For experienced developers. Focus: Lean optimisation, i.e. removal and limitation of bottlenecks in the development process. 3 rules – visualise workflow (kanban board), limit on „work in progress“, measure average lead time (lead – time it takes to finish a task).

Pros

More freedom for developers

Cons

Can’t be done with novice coders

Plan driven vs Agile

Showing results of pros and cons

When is plan driven/agile better 03 selecting methods

Choice of method

XP-SCRUM mix

Based on that picture plus whatever criteria suits us more in each development method.

Main principles in planning and QA

Planning principles

gathering requirements, choosing development method

User stories, acceptance test at back of story card, burndown chart

use what works for you not what the book says

Scope (adding or removing items)should only be altered as a LAST resort!

QA principles

Making sure that the result of our project will be high quality product is something that we must start working towards from the beginning. There are several different ways to assure high quality results. To assure external quality towards customer we have decided to try pair programming, test driven development and continuous integration.

Pair programming is a nice way to code, because it gives members of the group a chance to exchange ideas and come up with various ways to solve a single problem. In pair programming finding and fixing bugs also happens a lot faster, two people are more likely to spot a mistake than one individual. Test driven development is basis of quality assurance for our project. TDD allows us to build software that compiles. First, we write the simplest test, which captures main point of testable method. Then we write the method itself and make sure it compiles. Next, we modify test and actual method, so they would correspond more to the result we want to get. If all goes as planned, eventually we should end up with piece of software that does exactly what is expected of it. Conditions integration is also an important part of quality assurance. We push and pull code from GitHub repository very often. Every time a part of system is coded, the code is uploaded by creator and downloaded by other group members. That way we assure that each group member has the latest version of the project and people do not write code that doesn’t match with already existing code. Most of our group’s work will be done at a specific location where everyone must show up. We work as a team not as individuals. That way the communication inside the team is way better, clearer and simpler, which also helps with assuring high-quality results.

Quality control is also an important factor in our project. Testing the product when we finally get it will be crucial. We must make sure that all user stories, different scenarios, all attributes and overall appearance work. We will compare every user story to the product to see if every one of them can be done. We will test all input field, so even if given incorrect information, the system still knows how to continue working. Overall appearance is something that cannot be overlooked either. Product must have appearance that is appealing, simple and yet complex enough to keep it interesting. Testing all these factors on product is how we plan to implement quality control in our project.

In software it is also essential to keep high maintainability, reliability, usability and reusability. For code to be easily maintainable it is useful not to use many layers of inheritance. It’s also a good idea not to put several short lines of code into the same line. Easily maintainable code has clear communication lines inside the system. UI shouldn’t be able to directly call DB in one method if in every other method UI has to call Business layer first. It also helps to ensure reliability of the code, if method needs to be changed. Another way to keep up reliability is to reduce the number of errors in code and to keep different parts of code separated. Putting all methods that deal with users, products and services into the same class reduces reliability and makes it harder to read and use this piece of code. To make sure one doesn’t lose points in reusability it is advised to write simple methods that other bigger methods can call instead of writing smaller method into ever bigger method. The simpler and shorter those simple methods are, the more they can be used in various other methods. For example, a method that only searches for a person based on first name, can be called from far more methods than a method that searches for a person based on first name, last name and email together. To make software as user-friendly from developer’s point as possible, it is not good idea to make too long pieces of code or to merge methods.

Quality criteria and architecture !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

I’m not sure what this is. Have to figure that out

PRACTISE – Process and reflection

How we implemented XP-SCRUM

User stories, backlog

When created, how many, selecting importance of each one, putting stories into backlogs, creating sprint backlogs out of product backlogs

TDD

Is it worth, where we implemented it, how long we did it for

Daily meetings

Was it useful

Sprints, Iterations

Planning for them, actual vs planned amounts of code, how long iterations, actions done in each iteration

Pair programming

Is it worth, where we implemented it, how long we did it for

Spikes

What did we learn, how long did it take, how was it organised, how many different spikes

Solo programming

How much time was spent on solo coding tasks, which tasks were solo, comparison to pair programming, how much tdd used when coding solo

Most interesting pieces of code

Most complex parts of code (max 3), No huge pictures, more talk. What calls what,

Meeting the goals set in QA

Is code maintainable, usable, rausable, reliable. Did pair programming help, what about tdd

Conclusion

What we learned

Based on spikes, sys dev topics, planning, using scrum/xp/tdd

How was teamwork

Did people follow group contract, which tasks were signed to who, conflicts, a day at work

Did we get expected outcome of the application

Comparing promised features to existing features (did we have shallow depth inheritance trees, did quality control check out, did everything go as planned)

Literature

Not wiki pages

Not slides

Appendices

If we have any