Software Engineering: Mini-project - Group sw605f12

Our project

Our project is part of the “Autism software on the android platform”-multi-project. An android-based platform is to be developed which helps autists in their daily life.
In short, our part of the project aims to split the usage of android tablets up in two basic modes: “Guardian mode” and “Autist mode”. Guardian mode aims to help the guardians, by allowing them to customize and personalize the look and feel of each app which the guardian uses in their daily work with autists. As autists sometimes have trouble handling all the options which the android platform by default provides, the “autist mode” aims to lock down the device such that all not-required buttons, notifications etc. are removed such that the autist cannot “mess up” the configuration of the phone, by entering the settings and switching random features on/off.
We will be creating an application, which will be responsible of launching all other applications. We call it *the launcher*.

Multi-project work interface

As the multi-project is composed of several independent groups working together, we have all agreed on an interface that we must adhere to. This interface sets requirements to the work done in the groups and the way the groups communicate, in order to create a uniform standard that allows for effective communication and work across the groups.
There is one weekly meeting for the entire multi-project, where every group shows up and gives a short presentation on their progress. This helps keep each group aware of how far they are compared to the other groups, and makes it easy for groups to communicate with each other about the progress of their work.

Choice and modifications

It will be clear to the reader that we have chosen to use the XP development method. We acknowledge right away, that we are set out to fail, as we know that we will be committing the worst of all errors, listed at the top of the “How to Fail with Extreme Programming”-list, namely “no on-site customer”. However, we will try to compensate for this by exchanging email addresses, phone numbers, Skype contact information, and make a plan for how we can stay in touch as much as possible.

Rationale

Pair programming

The study regulation notes that this semester project should focus on building ‘High quality software’. The use of pair programming increases the quality of the software, with the possible tradeoff of lower quantity, but as the focus this semester is high quality software, we find it suitable to use pair programming. The pair programming-concept is new to all the members of the group, which means the use of it would increase the overall learning of this project and increase our experience with pair programming. As pair programming enforces you to work together in pair, it helps creating a common understanding of the code within the group as a whole, as more people are involved in each line of code.

The contrary, individual programming, would result in code which would be harder for the group to read and understand, as bigger parts of the code would be the result of one man’s labour, and therefore also written more in his own taste and preference than the group. This issue could be handled, by writing and maintaining a traditional bundle of documentation, explaining the code in detail.

Refactoring & Simple design

All code we write should be accessible, both for ourselves but also for other sub groups of the multi-project, and easy to document in the report. Refactoring helps make the code simpler and more readable, making it easier to share and easier to discuss, an advantage when working across project groups but also when writing the report. Therefore we choose to apply refactoring.

In order to reach simple code and high readability, other development practices such as top-down programming could be applied. Top-down programming requires a certain degree of knowledge of what the final result should be, which we did not have constantly, due to the changing nature of our project. Refactoring gives us an advantage as it adapts more to an agile development method.

Test driven development

Having good understanding of what the functionality of the product should accomplish is important, and having clear use cases and tests outlined before the code is written helps clarify it. As we do not have an on-site customer, the clarification provided by test driven development can be essential in staying true to the goals set by our customer.

Test driven development stands in contrast to traditional waterfall methods, where testing is performed late in the project. As other groups in the project are dependant on the services we provide, being able to deliver correct and reliable code to them is important. Doing test driven development allows us better to maintain our code as usable throughout the project, and

matches well with delivering releases in every sprint.

Whole team together

The concept of everyone working in the same direction decreases the amount of time wasted on unwanted code and tasks. Communication is essential for us to be working in the same direction. The feeling of working as a group also lowers individual ownership of member created parts of the project.

In contradiction, working isolated can reduce time spent on management. The reduced time spent on coordinating can end up in the team working in slightly different directions, and therefore time would be needed to synchronize every now and then.

Working together as a team will support us in gaining experience and knowledge on a common level, and help preventing mismatch between the group members' experience and knowledge.

Implementation plan

We will be implementing the XP approach by setting our iteration length to the same of the multi projects scrum-iteration length, such that we can deliver working code on time. We will both use the test driven development which is part of XP, and perform additional integration tests as part of the multi project.

Having the implementation planned, instead of just using shotgun programming, provides each group member with a understanding of how everything should be done. This makes the group stronger because everyone is sure of how long the iteration length is and how the test should be performed.

SWOT Analysis 2012-02-02

Strengths

By working in smaller groups, each member of the group is more involved in the project than they would have been in a bigger group, which raises engagement and work morale. A smaller number of group members also reduces overhead in organization.
Pair programming helps this further, as it increases code ownership between the members of the group and reduces the need for documentation as well, again reducing overhead.

Weaknesses

Our project has many 'critical' features, which need a lot of attention, but giving too much attention to specific features will reduce the amount of time available to the other critical features. Deciding the right scope can be hard.

Opportunities

If we succeed in creating high quality software which the customers like, then we might get national/international fame. The high quality also enables the possibility of others continuing our

work, which can lead to more features.

Threats

Many small groups have to agree what to do. If we fail to collaborate then the customers will not get satisfied at all. Having no leaders across the multi-project groups can lead to a lot of overhead of when deciding on specific solutions, that is rather equal, and the actual choice have a minor impact.

Risk management

Risk	C(r)	RMMM
Global backlog confusion	SSE	● Mitigation ○ communicating the groups backlogs ○ create global backlog ○ groups which depend on each other meet ● Monitoring ○ confusion at meetings ○ lack of features ● Management plan ○ feature lockdown

Global backlog confusion

One of the biggest risks in this multi-project is the basic understanding of what each of the small groups have in their backlog.

Outcome: Wasted time due to working in the wrong direction.
Reason: Not enough understanding of the different groups backlog.

Post Mortem

This is our review of how we have been able to use the tools gained from the Software Engineering Course (spring 2012), and how they worked out.

Tying the system together

The launcher is an essential component in the GIRAF system, providing the main interface for users to manage and launch their apps. This gives the launcher a big responsibility in making the user experience satisfying. So despite the fact that no leader has been chosen for the multiproject, we in the launcher group have had tasks that help tie the system together, and have led us to have more of a leading role in the project.

These tasks include e.g. overall design tasks, consistency and usability, where we have made a GUI library available to other groups, such that the same components can be used consistently in the system, but also tasks like managing users for the other apps and ordering materials for the groups, like logos and log-in cards.

We felt this made a good fit for our own ambitions, so it was not a bad thing in our case. It could however be alleviated by having a central leader who could distribute these tasks, and decide how to best have them fit into the project. This leader could also make major decisions regarding design, consistency and usability.

To backlog, or not to backlog

A flaw in our inexperience in using agile methods showed midway through the project. The fact that we didn't have a scrum master for the multi-project ended up with the multi-project having no actual global backlog. All of the information about what features would actually be implemented was pushed to the mini-group's backlogs, and the responsibility of informing the others about what features were in the individual mini-group backlog was then directed to the mini-groups themselves.

Having no global backlog also caused confusion over whether a feature was an idea, or an actual feature that would be implemented during our time with the project. The confusion led to some wrong expectations within the mini-groups, and created frustration within the mini-groups and the communication between the mini-groups involved was weakened for a period of time.

This problem could be solved by building a clear backlog for the entire multi-project, and keeping it maintained for the duration of the project, so it is also accessible and usable by those who will pick up the project later. A possible leader to make decisions regarding to this backlog and make sure it is kept up to date, this could also help solve confusion between the groups.

Program first, write later

One of the choices we made early on was to program our solution first, and then afterwards

write our report. One might argue that a major disadvantage of this approach is that we most likely will have forgotten why we took the choices we took when we start writing, as we did not document all our choices.

One of the decisions which counteract this disadvantage is the decision to have meetings regularly, where all major choices were made. These meetings were documented with a summary written during the meetings, which we can consult, should we forget why we did as we did.

One might also argue that, if we really do forget an important choice, then it probably was not that important, and does not deserve substantial attention in the report anyway.
We have in previous projects experienced the need to modify, delete and refactor big parts of the report, where the programming and the writing have happened concurrently. One of the reasons for this have been that we learned more during the project and therefore needed to update the report with this new knowledge. We expect this time consuming disadvantage to not be encountered during this project.

Communication, communication, communication

In order to make a multi-project work, all groups involved need to communicate their goals and tasks clearly, to make sure everyone understands what will be implemented and what will not. We had early on decided to hold meetings to accommodate the need for clear communication, which was largely a success. We were not used to having such meetings in the beginning however, so they were also a learning experience, with many early meetings being plagued by inefficient discussion and time usage.

We also tried to accommodate communication by keeping an open door policy, where in groups are encouraged to meet and discuss subjects that relate to them.
Even with these arrangements however, misunderstandings occurred. These were mainly down to the division of labor, where it was unclear whose responsibility it was to implement certain features, and communication of ideas versus actual tasks, where it was not always clear if the features being presented were out of scope or were actual goals for the project.

A leader to direct the project could choose which groups should handle which functionality, to stop conflicts from occurring. Clearer communication would be good and would help avoid holes in the road.

Pair programming and refactoring

We chose to use pair programming, which can seem a bit slow because it cuts down the quantity of code written, but the upside is that it should bring more quality code to the table.
A second choice we made was to use refactoring. Again a process that slows down the overall written lines of code, but should hopefully make more readable and accessible code, a step for quality code.

We think that with these two choices, we have ended up with better quality code that is also

more readable than it else would have been. As said before, one of the major things in this project is quality software and we took a step in the right direction with a combination of pair programming and refactoring. The tradeoff we got using this methods is as mentioned that we write less code. Refactoring takes a bit of time, but we think that in this project it was worth it especially because there will come another group after us and pick up this project so it is very important that the code is of high quality and has high readability.

The backside of pair programming is that when someone is sick or in another way can not be in the group room, the pair programming falls a bit apart because the partner in the pair programming team is set back to the traditional programming style.

We choose refactoring because it got us two major benefits: As our code got more simple the quality went up. Second, the readability became better because of the more simplified code. We choose to look after common code smells such as long classes, method and too many parameters etc.

Testing the code

While we had planned to do test driven development to maintain our code and features as well functioning and well defined, much of our code revolves around GUI creation. This is not as suitable for test driven development as non-GUI code, so our test driven approach was dropped in favor of testing after the last release was made. Any bugs found during testing are evaluated and put on the backlog to be fixed.

We have also done continuous, informal testing to make sure we catch any obvious bugs in the GUI and functionality, and our refactoring process has additionally worked as a form of code review. This approach allows us to get a better overview of what needs to be tested, as we can inspect the finished code to look for suitable test subjects.

If we on a multi-project level had agreed that we would all strive to incorporate test-driven development on all of our individual projects, some wasted time could have been spared. For example, we have a group responsible for an API which three other groups use, and a lot of time have been wasted for the three groups, because of errors in features which were not properly tested before they were released to the three groups.