The Norwegian University of Science and Technology (NTNU)
Faculty of Information Technology and Electrical Engineering (IE)
Department of Computer Science (IDI)
Version 03.05.2018

Compendium TDT4240 Customer Driven Project

Abstract: This compendium is an introduction to the course TDT4240 Customer Driven Project, where groups of students work on a software engineering project with a real life customer. The compendium provides useful information about the course in general, the goal and rationale of the course, as well as some software engineering guidelines meant for helping the students with their project.

This document has evolved during the years and the last revision 18.05.2018
by Letizia Jaccheri; Anniken Holst; Jon Atle Gulla

Table of Contents

1 Introduction

1.1 General Information

This **master-level** course **TDT4290 Customer Driven Project** deals with a **project assignment** that is **mandatory** for all computer science ("Datateknologi") students in their 4th study year at NTNU/IDI, typically with mostly Norwegian students. In addition, there are participants from the two-year, international master program in Information Systems at IDI (with 5-10 students from all over the world), and Erasmus and other guest students (usually Europeans). In 2017 there were more than 100 students in total. Since many of the students in the course are foreign, with limited or no knowledge to Norwegian language, the lectures and seminars are held in English.

In the beginning of the course, the students will be divided in project groups of 6-8 students (detail in Section 3.2). Each group will be allocated an advisor from IDI - either a faculty member, a postdoc or a PhD student - plus a main customer representative. Towards the end of the semester, the groups must deliver a project report in English, and hold a presentation and demonstration of the final *prototyped* product for the customer, while an external examiner (censor) is present.

This compendium contains all the necessary information for this course and a suggested outline for the final project report, in addition to some examples of what a project plan should contain. Practical information regarding project-group composition, dates etc. can be found on the course page on Blackboard.

1.2 Goal and Rationale of the Course

The **goal** of course TDT4290 is to learn – by working in groups – software engineering skills in the context of a *development project* to make a realistic *prototype* of an information system *"on contract"* for a real-world customer.

Through the project, all the phases of a typical software project are to be covered, e.g. project management and planning, pre-study, requirements, design, programming, testing, evaluation and documentation, but no "maintenance". Due to resource constraints, the focus should nevertheless be on delivering a system prototype called *the minimum viable product (MVP).*

1.3 Required Knowledge

Required theories and methods for making large and long-lived systems are mostly covered in previous, bachelor-level courses. This knowledge base is supplemented by a set of relevant lectures held by representatives from the IT industry. Attendance to the lectures is mandatory.

Since 2008 in this course, the students have been encouraged to use the Agile software development method Scrum. Given the time constraints of this student project, there is hardly time for more than 2-3 increments, called "sprints" in Scrum[1]. Other agile methods, namely Extreme Programming, Kanban, Lean Software Development, Mobile-D and others may be considered as well. In reality, the method followed in the course will be a composition of many methods. This should be carefully documented in the project documentation. Information about Scrum and some of the other software methodologies can be found in section 4.3.

[1] Atlassian's description of Scrum: https://www.atlassian.com/agile/scrum

2 Motivation on Project Work and Group Dynamics

2.1 About the Course

The goal of this course is to learn fundamental software engineering skills through realistic training in software development and project management. You will have the opportunity to apply the knowledge you have gained previously in your studies. During the course, you will experience situations that will require:

- Making decisions for, designing and developing a relatively large and complex system.
- Creative and collaborative problem solving. Earlier in your studies, the tasks have been smaller and more well-defined. In this project, there are (conflicting) decisions to be made with short time limits.
- Coordination of efforts and distribution of work and responsibilities.
- Project management, cooperation, decisions, follow-ups, and dispute resolution.
- Ability to adapt to non-ideal working situations.
- Planning and execution of plans. This involves creation of project plans and registration and monitoring of effort and resource usage.
- Handling of difficult customers, who might be unreliable and/or unavailable. An important part of this course is to manage the group project, so that the results match the customer's needs, even though the situation may turn difficult.
- Structuring of requirements specifications.
- Documentation. The project documents must be complete, well structured and target the technical knowledge level of the customer.
- Defend decisions that are taken on behalf of the customer. You should document all delays, overruns, and weaknesses, so that they can be explained and argumented for. Ideally, all decisions should match conditions coming from the customer (the customer has the right to complain on any aberration that is not his/her fault).
- Presenting (and selling) the final product for the customer/external examiner. Under the final presentation and demonstration, it is important to give the customer a complete and good impression of the system delivered.

2.2 Project Work in a Didactic Perspective

Several evaluations have been carried out of previous versions of TDT4290 (i.e. "Systemering prosjektarbeid" and "Programmering prosjektarbeid"). These evaluations are generally very positive.[2]

[2] See Markus Sorge: "Evaluering av prosjektundervisningen ved IDI, NTNU", Program for lærerutdanning, NTNU, spring 2000, 63p, http://www.idi.ntnu.no/undervisning/siving/docs/prosjektevaluering.pdf.

Technology is experience-based knowledge – composed and refined over many years − to be able to satisfy human-societal needs in a cost-effective way. **Engineering** is the process of combining and applying suitable technologies to construct specific means, such as houses, food, clothing, roads, bridges, vehicles, books, sewing machines − and recently − computer- and information systems. That is, an engineer creates new reality (e.g., kitchen tables) − not only studies the existing one (e.g., a humming bee in a forest).

Engineering requires a domain-specific methodology (a technology itself) for how to describe the actual context − being farming, bridge-building, or banking. An engineer will apply scientific insight (both technical and social), combined with knowledge and experience from many sources – all representing different technologies. It is often a strong relationship between what is being constructed and the available time, budget, and tools/methods. Because of the substantial complexity and diversity of the engineering work and the characteristics of the processes, it is often necessary for several people to work together. That means that engineering has a social dimension, since it is executed as group work. Cooperative and communicative skills are therefore essential.

Project work in teams is an important part of the engineering discipline. Your study program at NTNU is among the ones with most emphasis on project work. Project work means on the one hand that you need to make an agreement with a customer (customer/organization) about what should be constructed. On the other hand, you have to design and implement technical solutions that satisfy the elicited constraints and requirements. You also have to consider changes over time, as most customers are not sure about what they really want. As a consequence, the proposed unfinished solution must be modified. The project groups must also be well organized and effective, and try to avoid destructive internal conflicts.

All this means that you will get a hectic work situation – sometimes at the edge of chaos. You have to combine your theoretical knowledge from previous courses to solve specific and practical problems. A considerable amount of effort has to be spent on cooperation, communication, planning, and improvisation and you must show capabilities of working under pressure.

Your project will give you essential training to become a professional software engineer. Feedback from industry says, that it is almost impossible to get more done in 3 months than what such a group of students is capable of. Further, software engineers from NTNU are useful from day one: they posses the theoretical knowledge and know how to work efficiently in teams.

So, the expectations are great from all participants: the IDI department, lecturers, advisors, external customers and of course the students themselves. Welcome to an interesting and hectic semester in this course!

2.3 Training in Group Dynamics

Good teamwork and group dynamics are essential for the success of any collaborative project. Therefore, "social" skills are of utmost importance to become a successful project co-worker. A seminar on *group dynamics* is planned as a part of this course, to support the project groups to learn more about team work and group dynamics. In addition to the seminar, the team should spend some time in order to create a good atmosphere among your group, particularly at the beginning of the project.

3 Project Management

3.1 Workload

The official web page of the course[3] specifies 24 weekly person-hours per student for 14 weeks. This means that the total expected amount of workload per student is 336 hours (24*14). For a project group of 6-8 students, the available effort per group will lie between **2016** and **2688** person-hours, which is somewhere between 1 and 1.5 man-years. These hours include own reading, meetings, lectures, and seminars. Earlier projects have shown that it is possible to deliver really good results within that time frame.

It is important that everyone is honest and registers all effort (as person-hours) spent on the project. This means that the project documents must show the real workload. Effort overruns will result in less sparetime for you personally and less time for other courses. Inflated work effort does not affect the grades given in this course!

3.2 Allocation to Groups

In 2017, the course had 100 students in total. This gives in total 14 groups with 6-8 students per group. Each group is preallocated to one customer and one group advisor. Each group should have a tight cooperation with their advisor.

Group assignments are essentially made *randomly*. This is done intentionally to create groups where the members generally do not know each other beforehand. This is a typical situation in real-life, especially when working as a consultant. It is therefore encouraged to spend some time to create a good team atmosphere among your group, particularly in the beginning of the project.

If your group experiences that some of the team members are not participating satisfactorily, you should immediately contact your advisor. If you experience other minor problems, the advisor is the one to contact. However, most (minor) problems are to live with; in fact, it is a part of the course to learn to deal with such issues in a project.

Overview of groups, customers and advisors is available on Blackboard.

3.3 Customer

Each project group is initially given a one-page *project assignment* from an external customer. We do *not* accept customers that just want the group to write a "summary and evaluation" of some hot topic, with no ensuing implementation. And inversely, we do *not*

[3] http://www.ntnu.edu/studies/courses/TDT4290

want customers that come with pre-made requirements, and just want the group to complete a pre-designed system architecture.

The customer of a software project can sometimes be difficult to work with, unreliable and/or unavailable. They might change directions, come up with new ideas, and have an unclear picture of what they really want. An important part of this course is to manage the group project, so that the results match the customer's needs, even though the situation may turn difficult.

If a fundamental disagreement with the customer arises, the group has, if needed, "the final word" since the group members gets the credit through a final report worth 15 Sp. Fortunately such a dead-lock situation has hardly happened in the 40 years that this course has been arranged! However, the group and their advisor should do what is possible to resolve any major disagreements. Conflicts are to be explained, negotiated and resolved (managed), as this is part of the real world work. **It is therefore crucial that the group is focused and has a good dialog with the customer.**

The group should together with the customer agree on how the meetings and other communication should take place. Preferably the group should meet with the customer at the end of each sprint at a set time and place. However, as some customers are not based in Trondheim, such frequent meetings might not be possible. An option could be to replace some of the meetings with video conferences.

For all the meetings with the customer you should send a call for the meeting, specifying time, place, intention (result), agenda, and background documents. It is vital to specify what preparations you expect of the customer and the group before the meeting. You have to agree with the customer how long in advance the calling for meeting should be sent, e.g. at 12:00 two working days before the meeting is going to take place. During the meetings you must write meeting minutes, which should be sent to the customer after the meeting is over.

3.4 Supervision and Meetings

Your very first *group-internal meeting* is scheduled for the same day as the kick-off day. Each of you should introduce yourself to the others in the group, and try get the group organized for the first *customer meeting* the following hours.

On *the first customer meeting* later on the kick off day you will meet your advisor and your customer for the first time. The group is collectively responsible for making a *written resume* of this meeting. The resume should be sent by *email* to the persons involved (group members, advisor, customer) later the *same day*. So take good notes of this meeting!

Throughout the semester you will probably need several weekly, internal group meetings. The groups have been allocated a group room once a week, and you can find the room of your group on Blackboard. If needed, you can book a room at IDI by contacting the reception at IDI[4], or on another location on campus through https://tp.uio.no/ntnu/rombestilling/.

[4] The IDI reception can be reached on 735 93 440

Furthermore, your group should have a main ***advisor meeting*** **with your advisor once a week**. During the *first, pre-planned* meeting, you will have to agree upon *when and where* these meetings shall take place for the rest of the semester. The group is responsible for booking a meeting room for these meetings (possibly helped by the advisor). The meetings will have a group-specific content, and the advisor will also focus on the teamwork and group dynamics aspects and support you to establish a good group atmosphere. The meeting agenda must be sent to the advisor before 14:00 the day before, and should contain the following:

- Work done since last meeting
- Problems encountered
- Planning of work for the next period
- Other issues (if there are any)

3.5 Evaluation of Project Work

Towards the end of the semester, the group must hand in some deliverables. These are a project report, source code of the final product, and a video which explains and promotes the product. More information about what the report can be found in section 4.1 - Project Report and details about the video will be posted on Blackboard.

The project work will be evaluated on the basis of the quality of the project report, the functioning prototype of the system, the video and the presentation delivered at the end of the course. These all count towards the grade in an integrated way (they are not formally weighted against each other). The project is intended to be conducted as a team work effort. This means that also the team dynamics will have an impact on the final grade.

The following criteria are evaluated in an integrated way:

- Whether the group has solved the given assignment, according to the customer's objectives of the project.
- Team work efficiency and team dynamics.
- Team work process improvement efforts.
- Reasonable grounds for decisions taken.
- Logical flow in the report.
- Visibility of limitations imposed.
- Layout and structure readability.
- The students' ability to reflect on the process during the project.

3.6 Project Presentation and Demonstration

The project presentation is divided into three parts:

Presentation and video: Here you explain the project assignment and goals and the problems and priorities you have faced through the project. You should also describe the solution, the alternatives and why you chose this solution. The final product/prototype should be described, and you should include some final reflections about the solution, development process and the project in general.

Demo: Show your implemented prototype and its main functionalities. The presentation, video and demo should not exceed 30 minutes.

Questions: 5-10 minutes

Remember to give a hand-out of the presentation to your advisor, customer and external examiner (censor) at the presentation.

Note that since the presentation counts towards the grade, it is important that you maintain a functioning version of your program in case you (the group) appeal the result (grade). If an appeal is made, you will have to make your presentation for the new examiner, including a demonstration of the system.

3.7 Anti-plagiarism

The rules for anti-plagiarism are very strict, see §36 in "Forskrift om studier ved NTNU" (page 23 in "Studiehåndbok for Sivilingeniørstudiet 2011-12") regarding cheating and http://www.lovdata.no/all/hl-20050401-015.html#4-7.

See also http://www.idi.ntnu.no/grupper/su/publ/ese/plagiarism.html.

3.8 Intellectual Property Rights (IPR)

Intellectual property rights issues have to be handled according to NTNU and Norwegian regulations.

3.9 Course Reflection, Evaluation and Feedback

We intend to do a systematic evaluation of this project course. For this purpose, a "student reference group" must be established among the course participants. The course will be evaluated in the following ways:

- **Student surveys**: individual students will be asked to fill in a questionnaire at the beginning and at the end of the course. The questionnaires at the beginning of the course will be used to gather data on the students' expectations and the questionnaires at the end will be used to gather data on if the students' expectations have been met and other relevant feedback from the students.

- **Customer feedback**: We ask customers for feedback both before and after their projects have been carried out in order to see to that expectations have been met and to gauge customer satisfaction with their project group and the course in general.

The feedback received from the different parties will be used to improve the course for the future students.

4 Software Engineering Guidelines

This chapter contains some software engineering guidelines that you may find helpful for the project work.

4.1 Project Report

The project report should be written in English and contain no more than 100 pages (excluding appendices and graphics) and should be delivered as a pdf document.

The following subsections explain what the different sections of the report should contain. The groups are not obliged to follow this report structure to the point, it is meant as a guideline. Project reports from previous semesters can be found on Blackboard for more inspiration.

Even though the project report is to be delivered by the end of the semester, we strongly encourage you to start working on the report from day one.

4.1.1 Introduction

Write a good, one-page abstract early, and explain the overall context, motivation, demands and results.

4.1.2 Planning

The details of what to include in a project plan, as well as how to organize your project work, is explained in section 4.3 - Project Planning and Management.

4.1.3 Pre-study of the Problem Space vs. Solution Space

The preliminary studies are vital for the group to obtain a good understanding of the total problem. Here, you will have to describe the problem at hand. You should describe the current system and the planned solutions (text, workflow, use-case scenarios, information flow, and other graphical presentations you can use). It is all about getting a good understanding of the challenges ahead! The group should investigate if existing and potentially competing solutions exists on the market. If such solutions exist, they should be described. You should also describe alternative solutions that fully or partially require custom implementations. The group must also set up evaluation criteria that form the basis for choice of a solution. Software by third party software providers (as OSS or COTS) should be actively pursued as candidates for implementation of large parts of your software system[5].

[5] http://sourceforge.net

In cases where existing components can be applied as modules in the project solution, a simple cost-benefit analysis should be carried out.

Summary:

- Describe the main business requirements, both functional and non-functional, that will constitute the requirements for the final solution and its functionality. These requirements will later form the base for later formalization of requirements.
- Describe the situation and solutions of today ("as-is")
- Describe the wanted situation and its possible solutions ("to-be")
- Evaluation criteria
- Market investigations
- Description of alternative solutions
- Evaluation of alternative solutions, including adjusted requirements and potential costs and benefits
- Choice of solution, in dialog with customer

4.1.4 Development Methodology

For the realization of the recommended requirements above, the groups are fairly free to choose the methodology that they wish in cooperation with the customer and under the supervision of the supervisor. In the last ten years, it has been common to use variants of the agile method. Use of the agile methodologies Scrum, Extreme Programming and Kanban are covered briefly in section 4.2 - SCRUM, XP and Kanban.

In the report you should explain which development methodologies you have chosen to use, and reflect on why you have chosen these instead of other methods.

4.1.5 Requirements Specifications

It is important to explicitly state the system requirements and link them to the business requirements from the pre-study phase. Typically, requirements are divided into functional and non-functional requirements. Structure the requirements such that the presentation is well organized. The IEEE provide recommended practices for writing requirement specifications[6].

Some persons like to enumerate requirements (R1, R2 ...), which may create "boring" reading where it is easy to lose track of the content. The advantage with numbering is that it is then easy to separate the requirements from the rest of the text, each becomes explicit, and you achieve traceability and structure.

Use figures! Good figures say more than a thousand words. We strongly recommend making use-case diagrams[7] here, also because we then can make quick and reliable

[6] IEEE recommendation practices for writing requirement specifications: http://ieeexplore.ieee.org/iel4/5841/15571/00720574.pdf
[7] An explanation of use-case diagrams: https://www.lucidchart.com/pages/uml-use-case-diagram

estimates of the ensuing design, programming and test effort.

4.1.6 Architecture

Regardless of which type of development strategy is chosen, most software implementation projects start with system architecture and a sketch of the desired design, in order to ease later division into parts. The architecture should be designed in order to meet the established requirements of the product. The project most likely will be further developed later, so a "modular" design is to prefer. Designing a modular system also makes testing a lot easier, since defects can be tracked down to individual modules. When documenting the architecture, figures should be included, for instance one that shows how separate modules are related.

Knowledge about and use of patterns is useful when it comes to programming. This is briefly covered in previous courses, but examples of patterns from architecture and coding are found practically anywhere on the internet. A good book resource on patters is "Design Patterns: Elements of Reusable Object-Oriented Software", see also: http://en.wikipedia.org/wiki/Gang_of_Four_(software).

4.1.7 Implementation/Sprints

General knowledge about programming is expected to be covered in previous courses. Depending on the actual project commitment, this project might require that the team members learn new programming language(s), new concepts of programming, various technical skills etc. The group has to plan how to obtain this knowledge, maybe in cooperation with the customer and the advisor.

The group can also decide, in dialog with their customer, the focus and scope of the project. For instance, groups with focus on the early phases should not omit making a working prototype of some system parts. On the other hand, "programming-eager" groups may try to make a rather complete product. The important issue is that the group clearly justifies their decisions, and that there is a logical flow in the project report from start to end of all the phases, and that all the phases and iterations build naturally on each other.

If an agile development methodology, such as scrum, is used, the iterations should each have one chapter in the project report. For each iteration you should document the planning, implementation, feedback from customer and effort estimation, as well as other aspects you may find relevant. In section 4.2.4 Project Work Organization you can find an example of how to structure your sprints.

4.1.8 Security

Security is explained in detail in section 4.4 - Software Security.

4.1.9 Testing

Testing is usually planned and carried out in five parts:

1. Overall test plan – This should be created as the last part of the requirement specification phase.
2. A plan for each test that needs to be carried out. – This should be done in the end of each iteration.
3. Creation of detailed test specifications or checklists for each test. – This should be done in the end of each iteration.
4. Execution of tests, including correction of defects, re-testing and documentation of test results.
5. Approval of test results.

When you create a test plan it is important to specify:

- Which tests should be carried out?
- Which tests should contain checklists? (Checklists are most common for entity and module testing)
- Which tests should contain detailed specifications? (Detailed test specifications are common for testing systems, integrations, usability and acceptance)
- Who are the test subjects? (Project, customer, others...)
- When should the tests be carried out?

The level of detail should fit the nature of your project.

The detailed test specifications should contain:

- Test descriptions (the operations that should be carried out)
- Data that will be tested (input and expected output)

Tests carried out	Description
Unit test (programming phase)	Testing of the smallest units in the projects, i.e., user interface, methods, stored procedures, objects, classes, etc.
Model test (design phase)	Entities integrated into bigger software components. Modules are tested to assure that the coordination and communication between the entities are as expected.
System test (requirement phase)	All modules that together form a complete version of the system should be tested. The system are tested to assure that the coordination and communication between models are as expected.

Integration test (design phase)	This is a complete test of the system and its interfaces to the world around. The last defects should be found and it should be verified that the system behaves well according to the requirement specifications. In some projects integration and system tests are merged.
Usability tests (non functional)	These are tests that assure that the interaction between users and the system is as expected. The goal is to get user friendly applications.
Acceptance tests (non functional)	Here, the end users should test if the system and its user interface to its environment are as expected. Based on this acceptance test, the management or customer make decision on whether the product should be used or not.

4.1.10 Internal and External Documentation

A user- and installation-guide for the final product must be created. The installation guide should describe the installation process step by step. Note that your system and its installation will be tested by your advisor.

It is wise to start on this documentation as soon as possible, as it describes the current state of the project for the project team.

4.1.11 Evaluation

The groups decide themselves what to include in the project evaluation, but we recommend including the following elements:

1. The internal process and results: How have you worked together as a team? What have you done well? What have you not done so well? What would you have done differently? Conflicts that arose and how these were handled? Did you reach the project goals? What did you learn?
2. The customer and the project task: How was the communication with the customer? How did you experience the project assignment?
3. The advisors: How was communication with the advisors? Was the supervision good enough? How could the course be improved for the next year?
4. Further work: Give an estimate for how much effort that is necessary to complete the product/project.
5. Suggestions for improvement. What is missing to make this course better for both students, customers and advisors.

It is important to describe problems that may have affected the work but is not shown in the project report. Make sure that you also describe any additional work that is not shown in the project report.

4.1.12 Report Appendices

The appendix of your project report may for instance include the following:

1. User and installation guides (external documentation)
2. Technical/internal documents (internal documentation)
3. Other, e.g. special material provided by the customer.
4. Possible contracts and non-disclosure agreements.

4.2 Scrum, Extreme Programming (XP) and Kanban

Compared to the traditional project development processes, which usually deliver in a series of phases, agile projects break down the development durations into releases and iterations. At the end of every release and iteration, some small parts of the projects with functionality could be released.

A common feature of most agile methodologies is incremental design. Incremental design involves keeping the design simple from start and continuously improving it, rather than trying to get it all right from the beginning and then freezing it. Despite this, it is important to still keep the deadline in mind before further optimization.

4.2.1 Scrum

Scrum is an incremental and iterative framework for agile software development and project management[8]. It contains sets of predefined roles and practices by breaking down development procedures into small pieces, separating features into manageable items of work which they tackle in time-boxed iterations called sprints. The main procedures include product backlog, sprint planning meetings, daily stand ups, the sprint review and the sprint retrospective. These procedures will be explained in more detail in section 4.3.4 Project Work Organization where an example of how to perform the project management is provided. Scrum teams comprise of three distinct roles - the product owner, the Scrum master and development team members.

4.2.2 Extreme Programming (XP)

XP focuses on customer satisfaction[9]. It also maintains dividing development process into small iterations that a team is able to handle during a small period of time. Unlike Scrum, XP empowers the developers to confidently respond to changing customer requirements, even late in the life cycle. Teamwork is the key factor for XP process. Like Scrum, XP teams also comprise of three roles – manager, customers and developers. They are all equal partners in a collaborative team, which could guarantee the effectiveness of development progress. In XP progress, there are five essential ways help to improve the project:

[8] Atlassian's description of Scrum: https://www.atlassian.com/agile/scrum
[9] For further details about XP: https://en.wikipedia.org/wiki/Extreme_programming

Communication: Everyone is part of the team, and should communicate face to face daily, or in our case every few days.

Simplicity: Promote the development progress from the most basic needs from the beginning. And maximize the value the team could get while mitigating failures as they happen. Keep the cost into reasonable value.

Feedback: Get feedbacks from tests and customers. Talk about the project and progresses to each other and then make any changes and adaptations when needed.

Respect: Give respect to each other with their efforts and ideas.

Courage: Often tell truth about progress and estimates. And have the courage to adapt changes whenever they happen.

A key concept of XP is pair programming. Pair programming that all code to be sent into production is created by two people working together on a single computer. Pair programming increases software quality without impacting time to deliver. The best way to pair program is to sit side by side in front of the monitor and code together.

4.2.3 Differences between Scrum and XP

1. Scrum often has two-week to one-month long sprints, versus usually one or two weeks long iterations of XP.
2. Scrum doesn't allow any changes to the chosen sprint backlog after planning, while XP allows reasonable changes for unstarted features.
3. XP teams should always obey the priority order while Scrum teams could choose lower priority items during development.
4. XP focuses on engineering practices, while Scrum mandates planning ceremonies and artifacts. More specifically, XP is a test-driven development which focus on automated testing, pair programming, simple design, refactoring and so on.

It could be said that Scrum is a methodology, which is more concerned with productivity, while XP is more concerned with engineering.

The combination of Scrum and XP is one of the most popular agile methodologies in use today. Some steps of XP can be directly addressed by Scrum and can be seen as overlapping, like Sprint Planning Meeting of Scrum and Planning Game of XP, Sprint Review of Scrum and Small Releases of XP.

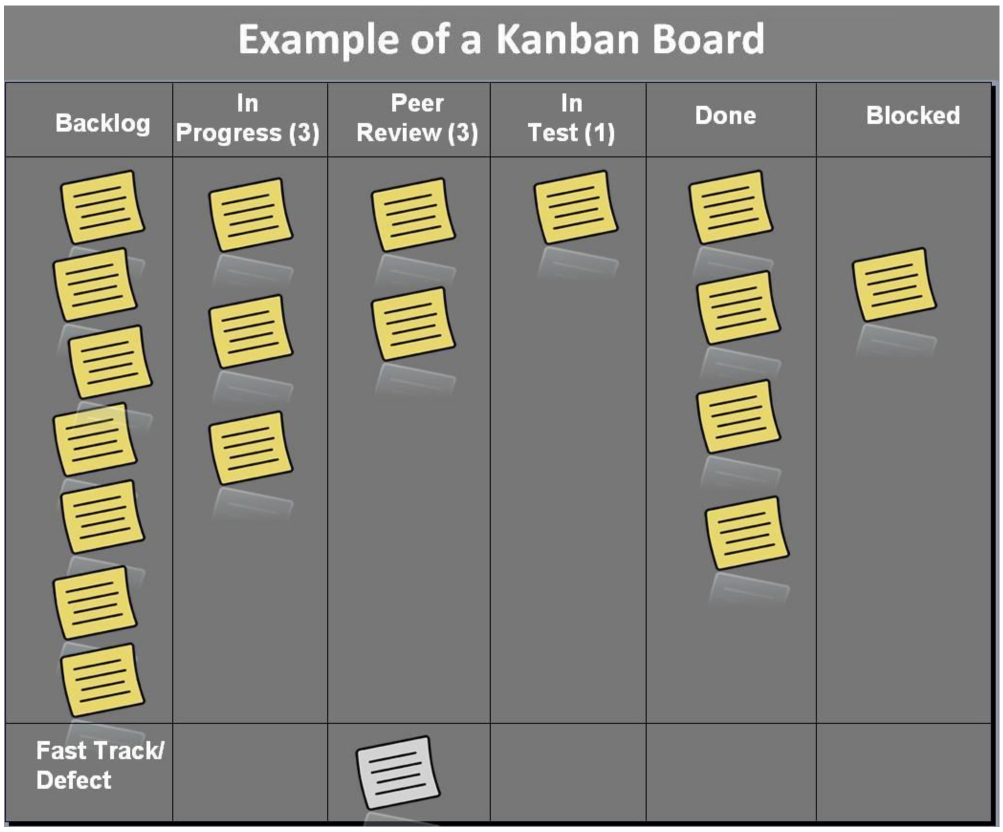
4.2.4 Kanban

Lean development practices[10] are based on the lean methodologies that have been used successfully in manufacturing processes. Kanban[11] is a lean software development methodology that focuses on just-in-time delivery of functionality and managing the amount of work in progress (WIP). It is a way to make the development process visualizing and vivid.

[10]Lean software development: https://en.wikipedia.org/wiki/Lean_software_development
[11] Kanban: https://en.wikipedia.org/wiki/Kanban_(development)

Since the progress of work can be seen explicitly, the project schedule could be handled more easily.

A feature of Kanban which is commonly used in combination with other agile methodologies is the Kanban board. These are typically presented on a big whiteboard or empty wall space where the team can place post-its with all kinds of information about the product and project. The process visualisation techniques of Kanban makes it ideally suited for co-located teams who are working on items that is subject to frequent change.



4.2.5 Other Agile Methodologies

There exist many other agile methodologies and you could always find the resources you need online. Lean, Feature Driven Development (FDD), Crystal, and Dynamic Systems Development Method (DSDM) are some examples[12]. You can choose one or combine several according to your needs and requirements.

4.3 Project Planning and Management

This section gives an example of how to structure a project plan. The project plan is dynamic and will evolve throughout the whole project. It regulates the administrative part of the

[12] A quick glance at other agile methodologies: https://www.versionone.com/agile-101/agile-methodologies/

project and guides the project. Depending on the type of lifecycle model you use you will have to structure the project plan differently.

4.3.1 Project Schedule

The project schedule should contain information about sprints/iterations, activities in each of these, milestones and person-hours spent on each activity and iteration, lectures, project management, and so on. It is suggested to include a Gantt diagram[13] to represent the schedule, and this can be attached as an appendix to the report.

4.3.2 Team Organization

Before starting a project, roles should be determined within a team. Each of the members could have one or two roles without prejudicing to the development processes. A description of typical roles used in Scrum is presented in the table below.

Roles	Responsibility
Project Manager (PM)	1. Ensure the completion of the project in the available time, within budget; 2. Organize weekly meeting and customer meeting; 3. Assign tasks and check progress; 4. Manage the time budget;
Quality Manager (QM)	1. Ensure the quality of the end product and the overall process; 2. Check that all project documents are consistent; 3. Arrange internal and external reviews; 4. Monitor and review all testing activities.
Product Owner/Customer	1. Make sure that all requirements asked by the customer are represented 2. Check that the items in the product backlog are user centered rather than technical
SCRUM Master	1. Ensure that the SCRUM process is followed; 2. Create the sprint backlog and check that the backlog is updated; 3. Lead the daily/weekly scrum and make sure that afterwards everybody knows what to do;

[13] Gantt Diagram: https://en.wikipedia.org/wiki/Gantt_chart

Development Team Member	1. Assisting the Team Leader or Project Manager by signaling problems in an early stage; 2. Executing plans made by the Team Leader and by the Project Manager; 3. Keeping track of time spent on various tasks; 4. Following procedures and plans.
Team Leader	1. Planning and coordinating team activities; 2. Providing feedback about team progress to the PM; 3. Motivating team members; 4. Chairing internal reviews of the items made by his/her team.

4.3.3 Project Work Organization

Here follows an example of how to organize your project work using the agile methodologies Scrum, Extreme Programming (XP) and Kanban as described in section 4.2 - SCRUM, XP and Kanban. The example starts out with Scrum, and then in addition uses XP and Kanban during every iteration.

Product backlog

A key concept of Scrum is the product backlog, or just backlog items. It is a list containing all requirements, stories, or features of the product, with priorities. The backlog should be made in the initial phase of the project, but it can be changed and altered throughout the project as the requirements evolve. The backlog must not be mistaken with the requirements specifications. Consider the backlog as a more informal document. A backlog often contains an ID, Name, Importance, Initial Estimation Time, Demo and Notes.

Sprint Planning

Before beginning a new sprint you should have a sprint planning meeting. This meeting is attended by the product owner (customer), the Scrum master and the rest of the Scrum team. The meeting could also be open for any interested.

During the meeting the Scrum team and the product owner should come to an agreement about which features and functions that have the highest priority. Based on this, the Scrum team should be able to determine which tasks they will move from the product backlog to the sprint backlog. The Scrum team and the product owner should collectively define a sprint goal - a short description of what the sprint will achieve. The fulfillment of this goal will later be discussed during the sprint review meeting. After the sprint planning meeting, the Scrum team will have to discuss how much they are able to commit during the sprint. This might lead to renegotiation with the product owner, but it will always be up to the team how much they can achieve during the sprint.

The Scrum team organization is well suited for a group of students, because none of the traditional software engineering roles like programmer, architect, designer or tester exist. Instead the SCRUM team focuses on collectively completing the tasks within the sprint.

Daily SCRUM

The daily Scrum(-meeting) should be held every day during a sprint. Usually these meetings are held in the morning, so that the team members can plan the rest of the day. Anyone can attend these meetings, but only the team members do the talking, the other should only listen. The daily Scrum should not be used as a meeting for problem solving; this should rather be discussed after the meeting only by the involved team members. During the daily Scrum every team member should answer the three following questions:

1. What did you do yesterday?

2. What will you do today?

3. Are there any impediments in your way?

The daily Scrum is not a status update, it is more like a commitment to the other team members of what you will do till the next day, and what you have done since last meeting. The good thing about having a daily Scrum meeting is that it helps the team see how important these commitments are to themselves and the team. Since this is a student project, and you all have other courses to attend (and students are not known to work from 08-16) it is not always possible to have a Scrum meeting every day or in the morning. The team has to work out a solution that every team member feels comfortably about and which at the same time makes it possible to monitor the progress of each team member.

Sprint Review Meeting

After each sprint a sprint review meeting is held. This is an informal meeting, which typically can consist of a demo of the new features made during the sprint. As for the other meetings this one is also open for everyone interested, but the Scrum team, Scrum master and the customer should normally participate this meeting.

Sprint Retrospectives

A sprint retrospective should be held before a new sprint. During the retrospective, everyone is allowed to contribute and discuss the ideas. Some suggestions might help for you:

- Sprint retrospective could happen at any place, you could do it in a cafe or rest room.
- Within this process, the SCRUM master shows the sprint backlog and with help from the team, summarizes the sprint, such as important events and decisions, etc.
- Everyone gets the chance to say something, like what they thought was good, what they think could have been better, and what they would like to do differently next sprint.

- Look at the estimated vs actual velocity. If there is a big difference, try to analyze why.
- Alway assign a secretary when doing retrospective, and in the end, he/she should try to summarize concrete suggestions about what they can do better next time.

4.3.4 Version Control Procedures and Tools

The group must create a *systematic procedure* for version control for *all* textual documents, source code, etc. Some recommended tools to use are git, Google Drive, OneDrive etc. You are also encouraged to use other types of tools to help you in the development process or project management. The tools used must be documented and described.

4.3.5 Quality Assurance (QA)

QA assumes that the relevant product qualities have been identified, so that the development process can be tailored to achieve these, e.g. reliability, performance, usefulness etc. There exists an ISO-standard for this (ISO 9126)[14].

Time of response

Make agreements with the customer. There should be time of response on:

- Approval of minutes of customer meeting (e.g. 24 hours)
- Feedback on phase documents the customer would like for review (max 48 timer)
- Approval of phase documents (max 48 hours)
- Answer to a question (e.g. 24 hours)
- To get agreed documents etc (e.g. 24 hours)
- Other

Routines for producing high quality internally

This has something to do with how you organize the specification and programming work, e.g. user involvement, "pair programming", design examination, peer reviews etc. The number of people involved should be weighed against available resources. Some useful practices are described below.

Best practices: The code you write might be used as a base for further development, and will probably be used by the other team members. It will be practical to follow common design and code conventions that all group members understand and practice. If the customer has a coding convention, they may want you to use this.

Legal issues: Please observe that some freeware or trial ware licenses of code editors etc. states that is it prohibited to use them to write code for commercial use. Check the license of the software that you decide to use in the project, and discuss it with the customer if there are such clauses that you might be in conflict with.

Coding style: How to write source code should also be specified. Such documentation should typically contain:

[14] http://en.wikipedia.org/wiki/ISO_9126

- Programming conventions, e.g. in use by the customer
- Standards for commenting source code.
- Show examples of source for how the programming conventions look like in practice.

The source code should be commented and documented proficiently, so that the customer easily can make modifications and build on your work after the project is finished. At the end of your project, the source code and necessary resources should be referenced and supplemented with a Readme.txt file.

Code Review: Code Review, or Peer Code Review, is the act of consciously and systematically convening with one's fellow programmer to check each other's code for mistakes, and has been repeatedly shown to accelerate and streamline the process of software development like few other practices can. There are some crucial insights about the code review from the book Best-Kept Secrets of Peer Code Review, which is about Cisco's peer-code review process conducted by SmartBear team[15]:

- Lines of code (LOC) under review should be less than 200, not exceeding 400, as anything larger overwhelm reviewers and they stop uncovering defects;
- Authors who prepare the review with annotations and explanations have far fewer defects than those that do not;
- Inspection rates should not be too fast or you might miss a significant percentage of defects.

The team members could use version control tools to do the review. Also, you need to review your code before commit to the remote repository.

Templates and Standards

The group should create templates for all relevant document types. Even though it will take some time to create these in the beginning, the group will benefit from these in two ways: 1) the layout will be correct when creating project documents and 2) reduction of irritation and stress within the group. Templates ought to be made for:

- agenda for meetings
- weekly status reports for the advisor meetings
- etc.

The group should also create pragmatic standards for:

- organization of files
- naming of files
- coding style
- etc.

4.3.6 Risk Management

Most likely there will sometimes be impediments between the scheduled plans and what is actually committed from the team members. Impediments could have many and various

[15] https://smartbear.com/SmartBear/media/pdfs/best-kept-secrets-of-peer-code-review.pdf

causes, but it is the Scrum masters' responsibility to resolve them as soon as possible. In cases where the impediments regard the Scrum master, he or she should delegate tasks so that someone else can help solving them. In more extreme cases (one on the team is not doing his workload, serious illness etc.) the team should contact the advisors of the course. Below are some examples of risks and impediments.

1. Miscommunication
Misinterpretations of what other team members say and write might stand in the way of a common understanding of what to do and how to do it. This might lead to delays, unwanted results and overlapping work.

Prevention: Throughout the project, and especially during weekly meetings, every team member should understand the task given to him or her, by communicating openly about the tasks during a week.

Correction: When a problem occurs, all team members should have a common understanding of the situation and discuss the remedial measures.

2. Too many planned features lead to infeasible design
This is a problem with high frequency, especially when designing the product backlogs.

Prevention: Every item in backlogs should have a time estimation, and all of the functions should be ready before the deadline. Always keep in mind the important dates when developing the product and carry out execution of tasks in order of priority.

Correction: By closely monitoring progress the decision to drop certain requirements can be made in time.

In order to handle risks, the different risk impediments that can happen in a project should be identified and prioritized. A template for a table for handling of risks can be seen below.

Nr	Activity	Risk factor	Consequences	Probability	Strategy and actions	Deadline	Responsible
	Which of the activities of the project are affected	Catching the name of the risk factors	Start with **H**, **M** or **L** before describing the consequences	H, M or L	Select strategy: Avoid, Transfer, Reduce, or Accept. Then on the next lines describe the measures	Set a clear deadline for ...	Give one person the responsibility

1	*All*	*Hans is involved in UKA*	***H:*** *The quality of the project results will decrease*	*M*	*Reduce* *Assign delimited tasks to Hans with clear deadlines*	*Continues*	*Project leader*

L = Low, M = Medium, H = High

4.3.7 Effort Registration

All projects needs to register the effort spent by each project participant on the different activities (e.g. Prestudy, Programming etc.) and in what period (week 1, week 2 etc.). This is needed to ensure that the project is on track according to the project plan. A weekly registration or periodization is common.

So *each* of you must *weekly* report - in a so-called *timesheet – seven data items per relevant activity and period*: *project group no, person name, date of registration, period no, activity name or id, your effort spent and given in person-hours* (possibly zero).

Make a *template* time-sheet for this information as soon as possible, and establish reporting procedures from the very project start. The project manager (or a delegated person) should be responsible to collect and synthesize the individual effort data into an updated *project effort-matrix* on a spreadsheet. The matrix data will be used to regularly monitor the planned (or estimated) effort vs. the actual one for the whole project. This matrix has *time* (period number) as the horizontal dimension, and *activity* as the vertical dimension. Each *matrix cell* contains a number measured in person-hours (ph).

So very early in the project, as part of making a Project Plan, you must *break down* the project's total available or estimated effort into a dozen main activities or phases, which again are allocated to periods. Naturally, activities belonging to the last part of the project cannot be broken down in detail in the start.

Thus the project plan must be adjusted over time.

Example: Assume that we have a software project with *three estimated activities (A1-A3) over three time periods (T1-T3)*:

- A1. Prestudy, whose estimated effort is 40 ph (person-hours).
- A2. Requirements, with 40 ph.
- A3. Implementation, with 20 ph.
- A. Total of 100 ph.

The project has an unspecified number of participants, so our project manager must keep track of the total resource usage (effort, time).

Version 1: Initial effort-matrix with very uneven effort estimates in the three periods:

Group no: ... Date: ...				
Activity\ Period	T1	T2	T3	Activity sums
A1. Pre-study	40			**40**
A2. Requirement		40		**40**
A3. Implementation			20	**20**
Period sums	**40**	**40**	**20**	**100**

Comment: It makes sense to overlap the three activities a bit, to get a more even effort distribution over the three periods.

Version 2: Reconciled matrix version, where the three "**diagonal**" ph-estimates (40, 40, 20) are spread out to get a more even effort distribution over time - please discuss the revised ph-estimates

Group no: ... Date: ...				
Activity\ Period	T1	T2	T3	Activity sums
A1. Pre-study	20	10	10	**40**
A2. Requirement	10	20	10	**40**
A3. Implementation	0 (OK)	5	15	**20**
Period sums	**30**	**35**	**35**	**100**

Version 3: Now introducing **estimated** (E) vs. **actual** (A) effort per period (T1-T3), both per **running** period (as above) and **accumulated** over several periods (see after the "/"-symbol in the below effort-matrix):

Group no: ... Date: ...						
Activity\Period	Start	T1	T2	T3	Activity sums	Activity comments
A1. Pre-study	**E:20** A:0	**E: 20/20** A: 13/13	**E: 10/30** A:12/25	**E: 10/40** A: /	E: */40 A: ..	...
A2. Requirement	**E=10** A=0	**E: 10/10** A: 11/11	**E: 20/30** A::19/30	**E: 10/40** A: /	E: */40 A: ..	...
A3. Implementation	**E=0** A=0	**E: 0/ 0** A: 2/ 2	**E: 5/ 5** A: 7/ 9	**E: 15/20** A: /	E: */20 A: ..	...
Period sums	E=30 **A=0**	E:30/30 **A: 26/26**	E: 35/65 **A: 38/64**	E:35/100 **A: /**	E: */100 A: ..	...
Period comments		**A1 delayed** **A3 before**	**A1 delayed** **A3 before**	**...**	...	

Let us assume that two time periods (T1-T2) have passed, with T3 just about to start.

Observation: in activity A1 after time T2 the Estimated running effort is 10 ph and the estimated accumulated effort (i.e. including T1) is 20+10 = 30 ph. However, the Actual effort for A1/T2 is 12 ph, and the accumulated effort is 13+12 = 25 ph. So it seems that A1 is a bit behind the estimated effort ("plan") – but that can have many valid reasons. We are only measuring resource usage (effort, time), not the actual state of the software under development! Ex. what advice will you give to all the activities A1-A3 for the last T3 period?

4.4 Software security[16]

Today, nearly all sectors of society depend on software systems to operate efficiently. As the dependence on software has grown, so have the threats towards these systems and the potential consequences of incidents in the systems. Though network security measures

[16] Written by Inger Anne Tøndel, PhD student IDI, Aug 2016.

(such as firewalls and anti-virus software) can improve the security of the software systems, these only address the symptoms of the real problem: software that is crippled with vulnerabilities. Because of the general dependence on software systems, software security now needs to be taken into account for all software, not only for security-critical software.

The mantra in all information security work is that it should be risk based, so also for software security. It is way too costly (and probably impossible for complex systems) to aim for 100 % secure software, thus it is necessary to identify which part of the software is more critical regarding security, and which activities will be most efficient and effective in securing the software product.

4.4.1 Typical Software Security Activities – the Touchpoints

Taking security into account for the system, means to build security into the system throughout the development process. One commonly referred approach for this is the seven touchpoints[17] that give an overview of seven activities that are in general recommended to do throughout development. These touchpoints are:

1. Static analysis tools – to automatically identify security bugs in the code
2. Risk analysis at the design and architecture level
3. Penetration testing
4. Risk-based security tests
5. Abuse cases – to get into the mind of potential attackers
6. Security requirements
7. Security operations

In this course the work on software security is expected to include some activities related to touchpoints 2, 5 and 6: Students are expected to identify key security requirements to their system and discuss the security implications of the functionality of the system and how attackers may go about attacking their systems, and they are expected to discuss security related to their architecture.

4.4.2 Software Security Games for Analyzing Security Risks and Threats

Two techniques are used in this course, but students are only expected to use the technique they receive training on, that is only one of these techniques. In this section the two techniques are briefly described.

Protection Poker

Protection Poker[18] is a security risk assessment technique for agile development teams proposed by professor Laurie Williams at NCSU.

In a risk analysis potential unwanted incidents in the system is identified (e.g. an attacker can get access to the database through some functionality and make changes) and one evaluates how likely this is and potential consequences. By doing this, one is able to gain an

[17] The seven touchpoints: http://www.drdobbs.com/the-7-touchpoints-of-secure-software/184415391
[18] Details on how to play: http://www.sintef.no/protection-poker

awareness of what are the main security issues of the product and make decisions on where to prioritise the security effort and what security mechanisms are needed

Protection Poker is played during every iteration planning meeting, and it is recommended that the full team (including developers, testers, product managers or business owners, project managers, usability engineers, security engineers, software security experts, and others) participates. One person should have the role as moderator, and this person will be responsible for leading the team through the game, and point the discussions in a good direction. One person should be responsible for making notes from the discussion, e.g. on potential issues, threats etc.

In one round of protection poker, the team plays about the features to be implemented in this iteration. The team starts with one feature and identifies the assets this feature touches upon. For assets that have not previously been assigned a value the team uses the protection poker cards and votes and discusses about the value until some reasonable agreement has been made. Afterwards the same thing is done for the exposure – to what extent the feature to be implemented makes it easier to attack the system.

After playing Protection Poker the team should have gained a common understanding of the security risks related to the features to be implemented in the iteration, and have the necessary foundation to make decisions on how much security and what type of security is needed.

Microsoft Elevation of Privilege (EoP)

Microsoft EoP[19] is a game for threat modelling related to an architecture. Before playing the EoP game, there should be a sketch of the architecture in place, and it is beneficial if this sketch includes an overview of the flow of the information in the system. Then EoP is used to identify any security problems in this architecture.

EoP is played like many other card games, and looks much like an ordinary stack of cards except that there is different suits and that there are security hints on the cards. Players gain points, and the one that ends up with most points win. Points are made for winning rounds, but also for laying cards with hints that are considered to be potential issues in the system. These issues should be noted and looked into after the game.

4.4.3 More on Threats, Vulnerabilities and What You Can Do About It

As input to playing Protection Poker and Microsoft EoP it is useful with an overview of some common threats, vulnerabilities and things to do about them. First a few things about terminology:

- An *asset* is something that is of value and that you want to protect, e.g. personal information or payment data
- A *threat* is what you try to protect against, e.g. loss of this data or someone tampering the data so that you can no longer trust it. When working with security, the

[19] Details on how to play the game: https://www.microsoft.com/en-us/sdl/adopt/eop.aspx

source of the threat is often thought to be an attacker, but harm to assets may also be caused by errors without there being an attempt to create any harm.

- A *vulnerability* is a weakness in your system that make the threat possible, e.g. sending information in clear text, trusting input from users without checking it, having some bugs in the code that can be exploited by attackers.
- A *risk* is the combination of all these – you need to have both something of value (an assets) a threat and a vulnerability for something to be a risk. The size of the risk depends on how likely it is that this will actually happen and the consequences in case it happens.
- A *countermeasure* is something that reduces the risk associated with a threat to your system, e.g. encryption or input validation.

A commonly used threat mnemnoic is STRIDE[20] (developed by Microsoft), and this can be used as a checklist to see if one has covered the main types of threats. The main threat categories are:

- Spoofing
- Tampering
- Repudiation
- Information disclosure
- Denial of service
- Elevation of privilege

Information on common vulnerabilities and how to mitigate them (what countermeasures to use) can be found at the following resources:

- For web-applications see the OWASP top 10[21]
- For a more general overview see the taxonomy of security errors known as the seven kingdoms[22]

[20] STRIDE: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)
[21] OWASP Top 10: https://www.owasp.org/index.php/Top_10-2017_Top_10
[22] Seven kingdoms: https://cwe.mitre.org/documents/sources/SevenPerniciousKingdoms.pdf