Table of contents

1 Introduction
   1.1 General information
   1.2 Goal and rationale of the course
   1.3 Required Knowledge

2 Motivation on Project Work and Group Dynamics
   2.1 About the Course
   2.2 Project work in a didactic perspective
   2.3 Training in group dynamics

3 Administrative information
   3.1 Work load
   3.2 Timeline
   3.3 Group assignments
   3.4 Rating of project work
   3.5 Supervision and meetings
   3.6 Project deliverables to examiners and jury on November 16
   3.7 Final presentation and demonstration on November 23
   3.8 Anti-plagiarism
   3.9 Copyright or Intellectual Property Rights (IPR)
   3.10 Course reflection, evaluation and feedback
Appendix A – The project plan
A1. Overall project plan
A2. Concrete project work plan
A3. Project organization
A4. Templates and standards
A5. Version control procedures
A6. Documentation of project work
A7. Quality Assurance (QA)
A8. Test plan

Appendix B – Suggestion for appendices in your project plan
B1. Partners
B2. Concrete project plan
B3. Detailed phase plans
B4. Table for handling of risks
B5. Table for effort registration

Appendix C – Content of the phase/sprint documents/chapters
C1. Introduction
C2. Planning
C3. Pre-study of the problem space vs. solution space
C4. Lifecycle model: waterfall vs. agile?
C5. Requirements Specifications
C6. Estimation of realization effort of a use-case model
C8. Programming
C9. Testing
C10. Internal and external documentation
C11. Evaluation
C12. Project presentation and demonstration
C13. Appendices

4 APPENDIX D – Administrative and Technical Resources
D4.1 Office Resources
D4.2 Technical Resources
D4.2.1. Workstations
D4.2.2. Source Control Management
D4.2.4. Use of collaboration technology in the project

5 Appendix E – SCRUM – a popular agile method
E5.1. Product backlog
E5.2. Sprint planning meeting
E5.2.1. Daily Scrum status meeting
E5.2.2. Irregularities
E5.2.3. Sprint review meeting
E5.2.4. How do I prepare a project for Scrum (short tutorial)

6 Appendix F – Use-case based effort estimation
F6.1. Introduction to use-case estimation
F6.2. More on the estimation method
F6.3. A mini-discussion
F6.4. References
F6.5. Appendix: A small use-case diagram, with extra comments
F6.6. Add-on to use-case example

7 Appendix G – Guest Lectures

Introduction

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 50-75 and 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), plus Erasmus and other guest students (usually a handful of Europeans).

This compendium contains all the necessary information for this course, the assignments (one for each project-group), and a suggested outline for the final project report. In addition, some examples of what a project plan should contain.

Practical information regarding project-group composition, dates etc. can also be found on the web page of the https://www.ntnu.no/wiki/display/tdt4290/. All students should check this web page for updates. In case of mismatch between information in this compendium, information given during lectures, by email, and on the above web page, the last updated information should be regarded as correct. The following IDI people are involved with this course:

Course responsibles:
Prof. Jon Atle Gulla Course responsible jag@idi.ntnu.no tel 73591847
Letizia Jaccheri Course responsible letizia.jaccheri@ntnu.no tel 91897028
Kristin S. Karlsen Practical coordinator kristika@idi.ntnu.no tel 73595204

Group advisors:

<table>
<thead>
<tr>
<th>Name</th>
<th>Picture</th>
<th>Email contact</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Email</td>
<td>Phone</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>Prof. Jon Atle Gulla</td>
<td><a href="mailto:jag@idi.ntnu.no">jag@idi.ntnu.no</a></td>
<td>73581847</td>
<td></td>
</tr>
<tr>
<td>Prof. John Krogslet</td>
<td><a href="mailto:letizia.jaccheri@ntnu.no">letizia.jaccheri@ntnu.no</a></td>
<td>91897028</td>
<td></td>
</tr>
<tr>
<td>PhD-stud. Francesco Gianni</td>
<td><a href="mailto:francesco.gianni@idi.ntnu.no">francesco.gianni@idi.ntnu.no</a></td>
<td>73551189</td>
<td></td>
</tr>
<tr>
<td>PhD-stud. Sofia Papavlasopoulou</td>
<td><a href="mailto:spapav@ntnu.no">spapav@ntnu.no</a></td>
<td>73593444</td>
<td></td>
</tr>
<tr>
<td>PhD-stud. Peng Liu</td>
<td><a href="mailto:peng.liu@idi.ntnu.no">peng.liu@idi.ntnu.no</a></td>
<td>73550241</td>
<td></td>
</tr>
<tr>
<td>PhD-stud Simon Elias Bibri</td>
<td><a href="mailto:simoe@ntnu.no">simoe@ntnu.no</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Students will be divided in project groups of 6-8 students (detail in Section 3.3). Each group will be allocated an advisor from IDI - either a faculty member, a postdoc or a PhD student - plus a main customer representative.

**The mandate of the advisor:**

The group's advisor serves as a one-person "steering committee" for your project. His/her responsibility is to keep an eye on the main process of the work, and to oversee that sufficient contact with the customer is maintained. The advisor must therefore regularly receive updated status reports, copies of relevant work plans and technical documents from the group, so that all this can be discussed in a **weekly advisor meeting**. In addition, the group should have several, weekly *internal meetings*, and regular *customer meetings* (detail in Section 3.5).

**Goal and rationale of the course**

The **goal** of course TDT4290 is to teach you and your fellow students – by working in groups – software engineering (SE) skills in the context of a **development project** to make a realistic, **prototype** of an Information System (IS) *"on contract"* for a real-world customer.

Each project group is initially given a one-page, **project assignment** from an external customer. All the phases of a typical IS/IT project are covered, e.g., project management and planning, pre-study, requirements, design, programming, testing, evaluation and documentation, but no "maintenance".

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

Due to resource constraints, the focus should nevertheless be on delivering a system or a prototype of system of called the **minimum viable product** (MVP). Sometimes the focus may be on the *"early" lifecycle phases*, i.e., project planning, pre-study, requirements specifications and system design – but the exact focus and emphasis will be decided by the group, in dialog with their customer. 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 prototype, e.g. by applying agile iterations in Scrum-style (see C.4 and Appendix F). The important issue is that the group clearly justify 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.

Each group should write a project report in English, and hold a presentation and demonstration of the final **prototype/product** for the customer, while an external examiner (censor) is present. 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 exam (report) worth 15 Sp. But 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.

**Required Knowledge**

Required theories and methods for making large and long-lived SE/IS systems are mostly covered in previous, bachelor-level courses. This knowledge base is supplemented by a double seminar on group dynamics, and four guest lectures on project management, Scrum, IT architecture, and use-case based effort estimation. In addition, we arrange a course in presentation techniques and a seminar on technical writing. These seminars, lectures and courses are mandatory. We will also recommend you to attend the course in drawing. That is voluntary.

Since 2008 in this course, the students have been encouraged to use the Agile software development method so-called "Scrum". Given the time constraints of this student project, there is hardly time for more than 2-3 increments, called "sprints" in Scrum. 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 is a composition of many methods. This should be carefully documented in the project documentation.

Many groups have since 2008 chosen an agile development model and had good results. As mentioned above, the structure of the final project report must reflect the choice of overall lifecycle model, i.e. waterfall vs. iterative/agile development methods.

**Motivation on Project Work and Group Dynamics**

**About the Course**

The goal of this course is to teach 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 is your studies. During this course, you will experience situations that will require:

- **Decision**, solving and design and development of 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. You will have to show creativity, be pragmatic and be capable of solving fuzzy tasks under heavy time and resource constraints; i.e. fast decision making under great uncertainty.
- **Coordination of efforts and distribution of work and responsibilities**.
- **Project management**, cooperation, decisions, follow-ups, and a dispute resolution.
- **Ability to adapt to no-ideal working situations**.
- **Planning and execution of plans**. This involves creation of project plans and registration and monitoring of effort and resource usage.
- **You must handle difficult customers**. They can be 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. This requires routines for quality control. Each group should deliver a project plan with resources and milestones, coupled to quantitative measures, and use both "dry" verification (mainly reviews) and "wet" testing.

- Structuring of requirements specifications.
- Documentation. The project documents must be complete, well structured and target the technical knowledge level of the customer.
- Decide that decisions are taken on behalf of the customer. You should document all delays, overruns, and weaknesses, so that they can be explained and argued. 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).
- To present (and sell) 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.

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. See Markus Sorge: "Evaluering av prosjektundervisningen ved IDI, NTNU", Program for lærerutdanning, NTNU, spring 2000, 63p, http://www.idi.ntnu.no/undervisning/sving/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 recent 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 solutions must be modified. The project documents 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. You have to use a considerable amount of effort in cooperation, communication, planning, and improvisation and 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 possess 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. The project report (written in English) should not be more than 150 pages, exclusive appendices and graphics.

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.

So, welcome to an interesting and hectic semester in this course!

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 following time slots should be used wisely to create a good team atmosphere among your group, particularly at the beginning of the project.

Administrative information

Work load

NTNU has officially an autumn semester with 19 weeks, and a spring semester with 21 weeks. Of the former 19 weeks, two are spent on continuation exams (and immatriculation etc.), and three are spent on exams in December which you don't have. This implies 17 "study weeks" in the autumn semester, each of 40 person-hours (work-hours) per student. This again corresponds to 340 (17 * 25% * 40) person-hours per student for a *15-Sp* course (50% of 30-Sp semester total). Note that an hour has 60, not 45 minutes. Since this project runs in 14 weeks (really 13.6) instead of 17 weeks, the weekly effort per student then be adjusted to 40 * 50% * 17/14 = 24.3 person-hours.

Furthermore, the official web page of the course http://www.ntnu.edu/studies/courses/TDT4290 specifies 24 weekly person-hours per student, but this is for 14 weeks. So let us round off to 25 person-hours per week and student, i.e., 350 person-hours per student for totally 14 weeks. For a project group of 5-6 students, the available effort per group will lie between 1750 and 2100 person-hours, including own reading, meetings, lectures, and seminars. Earlier projects have shown that it is possible to deliver really good results within that timeframe.

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 work load. 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!

Timeline

This is the preliminary time plan for the activities of this course. For further details and updates check http://www.idi.ntnu.no/emner/tdt4290/.
Group assignments

We expect about 80 students in total. This gives in total 12 groups with 6-8 students per group. Each group is preallocated to one customer and one group advisor. The groups should therefore 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 a real life, especially when working as a consultant.

Since many of the students in the course are foreign, with limited or no knowledge to Norwegian language, the lectures and seminars will be held in English.

Overview of groups, customers and advisors is available at https://www.ntnu.no/wiki/display/tdt4290/

Rating of project work

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 as well as and the students’ reflections on the project work. The project is intended to be conducted as a team work effort. This also means that the team dynamics will have an impact on the final grade. The final grade is a composition of the abovementioned components.

The project report must be written in English, and the presentation must be done in English. The project report, the video and the presentation count towards the grade in an integrated way (they are not formally weighted against each other). How the group actually has worked, technical problems, customer behaviour and availability, etc. are part of the reflections report. The group is asked to deliver a 3-4 page report, reflecting upon their experiences during the project and reflecting upon what they had learned and how they could have done things differently. The focus of this part is on the process rather than the product. This report is due at the end of the course and will be evaluated by the group advisor and the course coordinator.

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.

The criterias are not formally weighted against each other. 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 demonstration of the program.

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. It is up to you! Furthermore, your group should have a main advisor meeting with your advisor once a week, normally lasting one hour. Such meetings will have a group-specific content, but share a template agenda from Appendix A.7. All written documents for such meetings (agenda, weekly status report, phase-specific documents etc.) must be delivered on paper or by email to the advisors before 14:00 the day before. In Appendix A – Project plan, you will find more information about such meetings.

Thus, during the first pre-planned advisor meeting between your group and your advisor on Wednesday (see room/time in Appendix I), you will have to agree upon when and where the weekly advisor 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). During this hour, the advisor will also focus on the teamwork and group dynamics aspects and support you to establish a good group atmosphere.

Group rooms and room reservation You will probably need several weekly, internal group meetings. The groups have been allocated a group room once a week. For updated details see at https://www.ntnu.no/wiki/display/tdt4290/

If needed, you can book a room by contacting the reception at IDI on 735 93440.

Customer meetings are held when needed, starting on the kick-off day. The next customer meetings arranged in dialog with the customer, but the group is responsible for booking a room and other logistics. We recommend taking more contact with the customer, before the second advisor meeting in the following week.

Note: Before the first advisor, the group is collectively responsible for making a written resume of the first customer meeting held on the kick-off day. This resume should be sent by email to the persons involved (group members, advisor, customer) later on the same day. So take good notes of this first customer meeting!

All material included in the final delivery (project documentation, prototype of the system, reflection report, final presentation) should be delivered to the group’s advisor by midnight of the final delivery date. The group agrees with the advisor the delivery protocol (email, dropbox, google docs, other).

Project deliverables to examiners and jury on November 16

By November 16 (23:59) every group needs to email the following to the course responsibilities (Letizia Jaccheri and Jon Atle Gulla) and the course coordinator (Kristin Kartsen):

- URL of final project report (pdf)
- URL of 1 minute video presenting their project and product

Note that there is no printed deliverables in the course. Since it is only the report that is handed in on November 16, you may still modify your code until the final presentation on November 23 (as long as it does not affect the report itself).
Final presentation and demonstration on November 23

Follow up on https://www.ntnu.no/wiki/display/tdt4290/ for details about the arrangement for the day.

Room: Rooms are reserved for the final presentation. If the project demonstration requires special facilities (such as virtual reality or cave equipment), the groups can also book and have the presentation in other rooms. If your group needs to have the presentation in a specific room, please notify the course coordinator (Kristin Karlsen). Remember that the room must have space for 10-15 persons. The time and place for each presentation, will be published on the course webpage.

Laptop: Most groups use one of their personal laptops for the demonstration. If your group do not have a suitable laptop for the presentation, please notice drift@idi.ntnu.no two weeks before the presentation. A beamer will be made available in all presentation rooms.

Deliverables on November 23:

- USB key with final report, video, code and presentation slides to customer
- Presentation slides URL and code (e.g. Github) to course coordinator

The presentation schedule of November 23 is as follows:

<table>
<thead>
<tr>
<th>Time</th>
<th>Room G54</th>
<th>Room 242</th>
<th>Room 454</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:30 – 10:00</td>
<td>Information in Drivhuset to all students</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:15 – 11:00</td>
<td>Group 1</td>
<td>Group 6</td>
<td>Group 11</td>
</tr>
<tr>
<td>11:15 – 12:00</td>
<td>Group 2</td>
<td>Group 8</td>
<td>Group 12</td>
</tr>
<tr>
<td>12:15 – 13:00</td>
<td>Group 4</td>
<td>Group 9</td>
<td>Group 13</td>
</tr>
<tr>
<td>13:15 – 14:00</td>
<td>Group 3</td>
<td>Group 10</td>
<td>Group 14</td>
</tr>
<tr>
<td>14:15 – 15:00</td>
<td>Group 5</td>
<td>Group 7</td>
<td></td>
</tr>
<tr>
<td>15:15 – 17:00</td>
<td>Celebration and Bekk Prize in Drivhuset</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please note the following:

- The present day starts with an obligatory introduction at 9.30 in Drivhuset. All students should attend.
- All members of the groups are to be present during their own presentations.
- The presentations are in English and open to the public. The groups must invite their customers, but may also invite other people that have been involved in the project or may find the project results interesting. The supervisors will be present.
- Normally a presentation includes both a description of the project as well as a video and demo of the implemented system.
- Each presentation is up to 30 minutes long, including the video and demo. The examiner and the general audience have up to 15 minutes for questions.
- The group hands over the USB key to the customer at the end of their presentation.
- The presentation rooms are equipped with projectors, but other necessary equipment must be brought and installed by the student groups themselves.

Anti-plagiarism

The rules for this 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/h/20050401-015.html#4-7. See also http://www.idi.ntnu.no/grupper/su/publ/ese/plagiarism.html.

Copyright or Intellectual Property Rights (IPR)

For the entire lifetime of this course, it has been “unclear”, although rather frictionless, which “legal person” actually owned the IPR for the produced work, typically a project report and associated software. The Norwegian copyright law stands in LOV-1961-05-12-2 (http://www.lovdata.no/all/nl-19610512-2.html), and follows the Berne Convention for the Protection of Literary and Artistic Work from 1886.

Note: In Norway there is no need for a © symbol as in USA. Patents on software does not apply in Europe, but can be awarded in USA. Your employer owns the copyrights for software written as part of your employment contract (EØS rule).

IPR can be dealt with in four ways:

1. Just letting all parties (students, advisors, companies, anybody) do whatever they want with the work and with no explicit rules, just as now and compliant with the principle that scientific results (including implementation) shall be open and free of charge to everybody. However, legally and by default (Berne convention), the copyright (or IPR) belongs to those (the students) that have their names on the front page of the actual work.
2. Modifying NTNU’s new IPR policy, aimed at covering master’s theses - alone or in a group - but not project (i.e. pre-master) reports, see http://www.ntnu.no/studieavd/skiema/standardavtale07e.doc. 2p, dated 2007-08-28.
3. Using a revised, liberal and BSD-inspired IPR policy (about free and open software), formulated and adapted by Reidar Conradi in August 2006: http://www.idi.ntnu.no/grupper/su/publ/ese/new-standardavtale08e.doc. It is a combination of point 1) and 2).
4. Applying NTNU’s new IPR policy (also) for the TDT4290 project course, possibly revising point 2): The original IPR policy proposal was drafted by NTNU’s legal advisor Morten Øien, but withdrawn at its board meeting on June 12, 2008, awaiting vital clarifications, see http://www.
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.
- **Mid-term assessment by advisors:** the advisors will be asked to provide a mid-term assessment on how their project group(s) are progressing and notify if they foresee any challenges with their groups.
- **Customer feedback:** Customers will be asked to fill in a short questionnaire at the beginning and at the end of the course. These will be used to gather data on if and how the customers’ 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.

Appendix A – The project plan

This section gives an example of how to structure a project plan. The project plan is a dynamic document that will evolve and change throughout the whole project. The project plan regulates the administrative part of the project and guides the project. Depending on the type of lifecycle model you use you will have to structure the project plan differently.

A1. Overall project plan

Recommended content of the project plan ("project directive"):

- **Project name**
- **Project sponsor** (customer)
- **Partners including responsible third party providers**
- **Background for the project:** software system development
- **Measurement of project effects**, i.e. goals like:
  - Reduce the time it takes to create a daily production report with 3 hours...
  - 30% cost reduction of...
  - 40% increase in sales...
  - etc.

The effect measures are typically stated by the project sponsor, but it is likely that the group has to take the initiative to specify these in detail.

- **General terms.** What are your limitations, tool selections, organizational demands from the customer, resources etc.?
- **Based on the planned effort:** How many person-hours are to be used?
- **Schedule of results.** When should deliverables be available as milestones or sprints/iterations?

A2. Concrete project work plan

Recommended content:

- phases/sprints
- activities
- milestones
- person-hours per activity and phase + lectures + project management

The project plan (in form of a Gantt diagram) can be attached as appendices. It is also recommended to attach the detailed plan of the phases as appendices.

The workload (measured in %) of this course is normally scattered out on different project phases, as shown in table 2. A suggestion of a relative workload is found in column "Norm", while the experiences from groups 1 to 7 in 1997 are found in the subsequent columns. As you can see from the table, the relative workload varies a lot from group to group. This is a normal variation and is caused by the different assignments, the groups working differently, and that it is not strictly defined what each phase should contain. The suggestions given in "Norm" are a good starting point for the project plan.

For projects based on iterative development like Scrum, you will have to incorporate the phases which are needed into each sprint into the documents for each sprint.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Share in %</th>
<th>Norm</th>
<th>Gr.1</th>
<th>Gr.2</th>
<th>Gr.3</th>
<th>Gr.4</th>
<th>Gr.5</th>
<th>Gr.6</th>
<th>Gr.7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ideas to the content of the different phases are found in Appendix B. It is also recommended to look at previous project reports, which can be found at the web site of the course.

A3. Project organization

Recommended content:

- An organizational diagram of how the group is organized
- Roles, i.e., project leader, system analysis, system architect, system designer, test leader, customer contact, QA responsible, etc. Try to be inventive in role allocation!
- Responsibilities of the different roles
- Weekly schedule

A4. 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:

- phase documents
- 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.

A5. Version control procedures

The group must create a systematic procedure for version control for all textual documents, source code, etc., see Appendix D.4.2.3 on actual tools like CVS, SVN, Make etc.

A6. Documentation of project work

Internal project meetings
Try to have internal meetings at least once per week. In these meetings you should present the status, coordinate activities, divide tasks, and check the “mood” of the project. Set up an agenda and write precise minutes for each meeting.

Internal reports
A typical internal report:

- Person-hours - used and remaining (according to plan)
- Activities - done and remaining
- Achieved and not achieved milestones. These are important progress indicators.

Reporting of person-hours used should be done in written form to a specified time, e.g. as a part of the weekly reports.

A7. 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). More information can be found on Wikipedia, see http://en.wikipedia.org/wiki/ISO_9126

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)
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, etc. The number of people involved should be weighed against available resources.

Routines for approval of phase documents
Specify how you are going to approve the phase results (deliverable), which mainly consists of the phase documents. It is natural to involve the customer in the approval of pre-studies and requirements specification (or whatever you might call these documents). You must, as stated earlier, agree upon a time of response with the customer.

Calling for a meeting with the customer
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.

Minutes of a customer meeting
You must write a summary of the meetings with the customer. It is vital that you write down decisions, actions (what, who, and deadline), clarifications etc. that are important for further work in the project. The customer must approve the minutes of the meeting, to make sure there where no misunderstanding of decisions made etc. The minutes of meetings are part of the "contract" with the customer. In normal working life it is not uncommon that the minutes of meetings are part of the contract document with the customer.

In the project plan you should specify when the summary of the meeting should be done, when it is to be given to the customer for approval, how to distribute (e-mail, fax etc.) and expected time of response from the customer. It ought to be written by 12:00 o'clock the following day and should be distributed as soon as possible when you are done with internal approval within the group. It is vital that you get an approval as soon as possible to avoid misunderstandings.

Calling for the weekly advisor meeting between the group and its advisors
12:00 o'clock the day before the meeting, with print out of all needed paper.

Agenda for the weekly meeting with the advisor - a template that is to be followed

1. Approval of agenda
2. Approval of minutes of meeting from last advisor meeting
3. Comments to the minutes from last customer meeting or other meetings
4. Approval of the status report, which may be structured as follows:
   4.1 Summary
   4.2 Work done in this period
   4.3 Problems – what is interfering with the progress or taking resources? Problems are often risks that have taken effect.
   4.4 Planning of work for the next period
   4.5 Other

The status report (see Section 4.1-4.5 above) should be handed in as a separate document.

Minutes of the weekly meeting with the advisors
Is attached to the next calling for meeting and is a fixed subject on the agenda.

A8. Test plan
The project has to have an overall test plan, which either can be part of the project plan or as a test document (the latter is recommended, see Appendix C6).

Appendix B – Suggestion for appendices in your project plan

B1. Partners
Owners, target audience, customer representative(s), project group, advisors. For each person, record:

- name
- address
- phone number
- e-mail
- etc.

B2. Concrete project plan
The current project plan and old project plans. By also keeping the old plans the group can see how they have evolved and also possible learn from previous experience.

B3. Detailed phase plans
A detailed description of what each phase consists of.
When ending a phase, the next phase is fine planned in detail. The detailed plans for each phase are put here and not in the end of last phase document.

B4. Table for handling of risks

<table>
<thead>
<tr>
<th>Nr</th>
<th>Activity</th>
<th>Risk factor</th>
<th>Consequences</th>
<th>Probability</th>
<th>Strategy and actions</th>
<th>Deadline</th>
<th>Responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All</td>
<td>Hans is involved in UKA</td>
<td>H: The quality of the project results will decrease</td>
<td>M or L</td>
<td>Select strategy: Avoid, Transfer, Reduce, or Accept. Then on the next lines describe the measures</td>
<td>Set a clear deadline for ...</td>
<td>Give one person the responsibility</td>
</tr>
</tbody>
</table>

L = Low, M = Medium, H = High

B5. Table for 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 time sheet – 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 (a textual email-message format will do), and establish reporting procedures from the very project start. The reported effort data should be delivered ca. two days before the weekly advisor meeting. 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 spread-sheet.

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 (ca. 1700 person-hours) into a dozen main activities or phases, which again are allocated to periods (week no 1-13), cf. Appendix A2. 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:

<table>
<thead>
<tr>
<th>Activity/Period</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>Activity sums</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1. Pre-study</td>
<td>40</td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>A2. Requirement</td>
<td>40</td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>A3. Implementation</td>
<td>20</td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Period sums</td>
<td>40</td>
<td>40</td>
<td>20</td>
<td>100</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Activity/Period</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>Activity sums</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1. Pre-study</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>A2. Requirement</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>
### A3. Implementation

#### Period sums

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>15</td>
<td>15</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>Estimated</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>60</td>
</tr>
</tbody>
</table>

#### 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):

<table>
<thead>
<tr>
<th>Group no:</th>
<th>Date:</th>
<th>Activity/Period</th>
<th>Start</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>Activity sums</th>
<th>Activity comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1. Pre-study</td>
<td></td>
<td>E:20 A:0</td>
<td>E: 20/20 A:13/13</td>
<td>E: 10/30 A:12/25</td>
<td>E: 10/40 A: ..</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2. Requirement</td>
<td></td>
<td>E=10 A=0</td>
<td>E: 10/10 A:11/11</td>
<td>E: 20/30 A:19/30</td>
<td>E: 10/40 A: ..</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3. Implementation</td>
<td></td>
<td>E=0 A=0</td>
<td>E: 0/ 0 A: 2/ 2</td>
<td>E: 5/ 5 A: 7/ 9</td>
<td>E: 15/20 A: ..</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Period comments

- A1 delayed
- A3 before

#### Period comments

- A1 delayed
- A3 before

#### Observation

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?

### Appendix C – Content of the phase/sprint documents/chapters

This appendix contains information about what the different phase documents (or report chapters) should include. See also former project reports for more details.

That is, it is common to divide a waterfall-based software project into the following 13 (or so) phases, whose documentation then becomes a chapter in your final project report:

- C1. Introduction
- C2. Planning
- C3. Pre-study of the problem space vs. solution space
- C4. Choice of lifecycle-model: waterfall vs. agile?
- C5. Requirements specifications
- C6. Estimation of realization effort for use-case model
- C7. Construction/ design
- C8. Programming
- C9. Testing
- C10. Documentation
- C11. Evaluation
- C12. Presentation and demonstration
- C13. Appendices

If an agile lifecycle model is chosen, like in Scrum, we recommend one "phase" (or report chapter) per Scrum-sprint, typically 2-3 such. These "Scrum phases" will then replace the waterfall phases C5-C10. It may still be useful to include an initial requirements chapter and an initial Design and Architecture chapter, but this may vary from one project to another. It may also be a good idea to include a final Test chapter with test plans and test results.

Below you will find some more information on what each phase document is expected to contain. Note that the group can also select another phasing.

#### C1. Introduction

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

#### C2. Planning

See appendix A – Project plan.

#### C3. 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, see 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. Try also to make use-case diagrams to express the major functional requirements, cf. the simple effort-estimation method in Appendix F. 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.

To conclude, the pre-study should have two main deliveries:
- A (partly) prioritized set of requirements – cf. the Scrum "backlog".
- A proposed system architecture to enable a fast break-down (modularization) of the technical work

This is also the time to revise your project plan, possible due to use-case based estimation or clear choices of 3rd party software.

**C4. Lifecycle model: waterfall vs. agile?**
For the realization of the recommended requirements above, the groups are fairly free to choose between a waterfall or an agile lifecycle model to make a prototype for this project. However, some of the assignments do require use of agile development methods, and the customer will anyway like to have his/her hand on the steering wheel.

The concept of waterfall development and its properties are considered known from previous courses, and are not covered here. See also phase descriptions in sections C5-C10 below.

Use of the agile methodologies called Scrum, Extreme Programming and Kanban board are covered briefly in Appendix E.

**C5. Requirements Specifications**
In the requirements specification phase, 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.

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 a lot of figures! Good figures say more than a thousand words. We strongly recommend making use-case diagrams here, also because we then can make quick and reliable estimates of the ensuing design, programming and test effort (Appendix F).

Before requirements are stabilized, you should give an overview of the software architecture. This means including figures that show how separate modules are related. Prototype-diagrams of the user interfaces are often very helpful for communicating with the customer.

**Suggested outline of a more detailed Requirements specifications**
The IEEE (Institute of Electrical and Electronics Engineers) have made a good recommendation paper on recommended practices for writing requirements specifications which can be found on: (http://ieeexplore.ieee.org/iel4/5841/15571/00720574.pdf). If you are using this recommendation, please note that it is designed as a general-purpose software requirement specification plan and that some parts may not be relevant for your project.

The outline of the IEEE software requirement specification is:

<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
</tr>
<tr>
<td>1.1 Purpose</td>
</tr>
<tr>
<td>1.2 Scope</td>
</tr>
<tr>
<td>1.3 Definitions, acronyms, and abbreviations</td>
</tr>
<tr>
<td>1.4 References</td>
</tr>
<tr>
<td>1.5 Overview</td>
</tr>
<tr>
<td>2. Overall description</td>
</tr>
<tr>
<td>2.1 Product perspective</td>
</tr>
<tr>
<td>2.2 Product functions</td>
</tr>
<tr>
<td>2.3 User characteristic</td>
</tr>
<tr>
<td>2.4 Constraints</td>
</tr>
<tr>
<td>2.5 Assumptions and dependencies</td>
</tr>
<tr>
<td>3. Specific non-functional requirements (e.g. performance requirements, database requirements, security, reliability etc.)</td>
</tr>
</tbody>
</table>

**Appendices**

**Index**

**Glossary**

**C6. Estimation of realization effort of a use-case model**
For TDT4290 Customer-driven Project, IDI, NTNU - by Reidar Conradi, Aug. 2011. See also Appendix F for an extended presentation; here comes the short one.

Use-case based effort estimation is a very simple and cheap method to estimate the remaining implementation effort of a requirements specification, when the latter is expressed as a set of upgraded use-case diagrams. This means that a textual specification (resembling "pseudo-code") must be added to the graphical actor and use-case models. The actors and use-cases must then be categorized as Simple, Average, and Complex - by you.

An effort estimate in person-hours can now be calculated, by you or by a spread-sheet tool (www.idi.ntnu.no/grupper/su/publ/reidar/uc-ProjectEstimateMethod-2011-v2.xls). It typically takes 30 minutes for you to execute the entire method, given that "upgraded" use-case diagrams are available. And only six "numbers" need to be given by you!

An effort estimate in person-hours can now be calculated, by you or by a spread-sheet tool (www.idi.ntnu.no/grupper/su/publ/reidar/uc-ProjectEstimateMethod-2011-v2.xls). It typically takes 30 minutes for you to execute the entire method, given that "upgraded" use-case diagrams are available. And only six "numbers" need to be given by you!

**Example:** Appendix F contains a small use-case diagram with 3 actors and 10 use-cases. All 3 actors are termed Complex. Further, 2 of the use-cases are termed Simple, 3 are Average, and 5 are Complex. We then multiply the actor numbers with the "cost points" 1,2,3 for respectively Simple, Average, Complex actors, i.e. 1*0 + 2*0 + 3*3 - or totally 9 actor points. We similarly multiply the Use-case numbers with the cost points 5,10 and15 for respectively Simple, Average, Complex use-cases, i.e. 5*2 + 10*0 + 15*5 or totally 115 cost points. The sum is 9+115 = 124 cost points, which should be multiplied by a Productivity factor to get real person-hours. For small systems like yours, this factor can be set to 1.0. Thus, the estimated implementation effort becomes124*10 = ca. 1250 person-hours (~ 20%)!

**C7. Construction / Design**
Construction is all about getting a vague system description, to a specific and detailed description that can be implemented and realized. A client-server software system typically includes three tiers:

1. User interface
2. Business or application logic
3. Database or file system

Pseudo-code and UML are useful for describing solutions in the construction phase.
Regardless of which type of development strategy is chosen, (waterfall, incremental) most software implementation projects start with system architecture and a sketch of the desired design, in order to ease later division into parts. The project most likely will be further developed later, so a "modular" design is to prefer. Designing a modular system also makes testing much more easily, since defects can be tracked down to individual modules.

C8. Programming

In the very beginning of the project a lot of important technical decisions have to be made. This is vital for the remainder of the project, but there are certain practices that make it all easier.

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 have to plan how to obtain this knowledge, maybe in cooperation with the customer and the advisor.

Best practices

The code you write might be used as a base for further development, and may also be used by the other team members. Inside the group, it will also be practical to have common design and code conventions that all group members understand and practice. If the customer has a coding convention, they probably would like you to use it also.

Knowledge to and use of patterns is also 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 patterns is "Design Patterns: Elements of Reusable Object-Oriented Software", see also: http://en.wikipedia.org/wiki/Gang_of_Four_(software)

Legal issues

Please observe that some freeware or trial ware licenses of code editors etc. states that it is 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.

Technical support tools

Versioning control and backup CVS / SVN tools should, as mentioned in Appendix A5, be used for all technical artefacts (UML-diagrams, code, test data, etc.) and documents in the project. When you set up a development environment, make sure to set up a functional versioning system with backup as well. You never know when you might need it! It is possible to use CVS/SVN on IDI, for more info, contact the Gurutjenesten in the P-15 building.

Coding style

How to write source code should also be specified? Such documentation should typically contain:

- 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 so well, 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:

- 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 CVS/SVN tools to do the review. Also, you need to review your code before commit to the remote repository.

C9. 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 need to be carried out, – This should be done in the end of the construction phase.
3. Creation of detailed test specifications or checklists for each test. – This should be done in the end of the construction phase.
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 persons? (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)

<table>
<thead>
<tr>
<th>Tests carried out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit test (programming phase)</strong></td>
<td>Testing of the smallest units in the projects, i.e., user interface, methods, stored procedures, objects, classes, etc.</td>
</tr>
<tr>
<td><strong>Model test (design phase)</strong></td>
<td>Entities integrated into bigger software components. Modules are tested to assure that the coordination and communication between the entities are as expected.</td>
</tr>
<tr>
<td>Tests carried out by the end users</td>
<td>These are tests that assure that the interaction between users and the system is as expected. The goal is to get user friendly applications.</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acceptance tests (non functional)</th>
<th>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.</th>
</tr>
</thead>
</table>

When testing, you should perform the test after completing a phase/design/component etc. This is very close to the V-model used in software development. For more information about this, the Wikipedia article gives a good introduction ([http://en.wikipedia.org/wiki/V-Model_(software development)](http://en.wikipedia.org/wiki/V-Model_(software development))).

**C10. Internal and external documentation**

A user- and installation-guide for the final product must also be created. It should include an installation guide, which describe the installation process step by step. Note, that your system and its installation will be tested by your advisor.

**Hint:** start on the documentation as soon as possible, as it describes the current state of the project for the project team.

**C11. 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 to 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 hidden problems that can 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.

**C12. Project presentation and demonstration**

To be used in Section 3.7. The start-up is divided into three parts:

1. **Presentation og video**
   - Explain the project assignment and goals
   - Problems and priorities
   - Which solution did you choose? What were the alternatives? Why did you choose the final solution?
   - A description of the final solution
   - Some final reflections

2. **Demo**

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

3. **Questions:** 5-10 minutes

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

**C13. Milestones and Feedbacks**

At the beginning and the end of this class, each team will be asked to fill some questionnaires about this classes and yourself. For example, they may ask “What do you expect from this course?”, “Have you participated any project before?”, “Which parts do you think need to improve? And how to improve?” etc.

At each stage or phase of the development, each team needs to write their results into documents. Then the supervisors will collect them as feedbacks and basis of the next plan. We recommend the stage/phase report contain the following parts:

1. A summary of progress to date against the plan
2. A summary of the project’s performance (the completed parts)
3. Issues and risks and their impact
4. Benefits from the work (what do you learn from the work)
5. Plan for the next step

**C14. Appendices**

These may - for instance – include the following, but see also section C10 above:

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

This chapter presents an overview over the resources available during the project work.

D4.1 Office Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Printouts</td>
<td>The group has a quota of 500 pages each.</td>
</tr>
<tr>
<td>Photo Copying</td>
<td>All the documents you deliver during the project period can be copied for free at the main copier in IT-154. The secretaries at the information desk will give you the information to operate the copier.</td>
</tr>
<tr>
<td>Telephone</td>
<td>If you need to use a phone as a part of your project work, please contact the course coordinator or your advisor.</td>
</tr>
<tr>
<td>Home page</td>
<td>The course webpage contains a lot of important information. It should therefore be checked frequently, see <a href="https://www.ntnu.no/wiki/display/tdt4290/">https://www.ntnu.no/wiki/display/tdt4290/</a></td>
</tr>
</tbody>
</table>

D4.2 Technical Resources

D4.2.1. Workstations

Fourth year computer science students have access to the 5th floor in the P15 building. Additionally, IDI technical group offers virtual servers for all the groups to host/run any special software necessary that is not available on the normal servers available to students. If technical problems occur or you have particular needs regarding the computers, contact the IDI technical group, drift<##>idi.ntnu.no

D4.2.2. Source Control Management

It is usually essential for groups to maintain some sort of source control management. We will not advise on a specific tool, but not that groups have had success with CVS, SVN, Git, Mercurial, Arch and Bazaar in the past. It is worth to keep in mind that CVS and SVN both adopt a centralized repository design, a lot of developers are shifting towards distributed repositories due to weaknesses in a centralized design. Distributed alternatives to look at would be such as Git (and github), Mercurial, Arch and Bazaar (all open source).

D4.2.4. Use of collaboration technology in the project
The groups are encouraged to use some kind of collaboration technology (file sharing, defect registration etc.) to coordinate the different tasks among group the members. There exist several tools for this. The following is a list with some of the more well-known software packages for collaboration and information gathering:

**Gathering of static information on the web:**
Mediawiki: Open-source program for gathering static information on the web.
Example of use: wikipedia.org

**Management software for defects etc:**
Trac: Great for registration of defects, things to be done etc. Open-source.
Example of use: http://trac.edgewall.org/report

**Sharing of files:**
Microsoft Sharepoint: For sharing of files, documents etc. Commercial, but is available to NTNU students for free, under special licensing rules.
Dropbox is another popular example, http://www.dropbox.com
Additionally Stud.ntnu.no offers group areas: http://www.stud.ntnu.no/kundesenter/

Appendix E – SCRUMP, Extreme Programming (XP) and Kanban board
Compared with the traditional projects development which usually is delivered in a series of phases, agile projects break down the development durations into releases and iterations. At the end of every releases and iterations, some small parts of the projects with functionality could be released. Figure 1 is the differences between traditional and agile methodologies.

The combination of SCRUM and XP is one of the most popular agile methodologies in use today. They complement each other well with tandem schedule.

**Fig. 1. The basic workflow of Traditional development and agile development**

**E1. SCRUM**
SCRUM is an incremental and iterative framework for agile software development and project management. 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 mainly procedures include product backlog, sprint planning meeting, daily stand up, the sprint review and the sprint retrospective. Meanwhile, SCRUM teams comprise of three distinct roles - the product owner, the SCRUM master and development team members. The detailed information and steps will be described below.

**E2. XP**
XP focuses on customer satisfaction. It also maintains deviding development process into small iterations that a team could handle during a small period of time. Not like 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:

- **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 mitigate 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.

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 resonable changes for unstarted features.
3. XP team should always obey the priority order while SCRUM team 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. (see Fig. 2)
E3. Kanban

Lean development practices are based on the lean methodologies that have been used successfully in manufacturing processes. Kanban 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. Since the progress of works can be seen explicitly, project schedule could be handled more easily.

E4. SCRUM + XP + Kanban

The team could start with SCRUM, then use XP and Kanban during every iterations of project realizing. (see Fig. 4)

E.4.1. Product backlog

A key concept of SCRUM is the product backlog, or just backlog items. It is the list containing all requirements, or stories, or features with priority in the product. This 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 could contain an ID, Name, Importance, Initial Estimation Time, Demo and Notes.
E.4.2. 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 over which features and functions that have the highest priority. Based on this, the Scrum team should be able to determine which task 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. 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 is focusing on collectively complete the tasks within the sprint.

E.4.3. 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 others than the team members are only allowed to 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 to 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 either) it is not always possible to have a daily Scrum meeting every day or in the morning. The team have to work out a solution that every team member feels comfortably and at the same time makes it possible to monitor the progress of each team member.

E.4.4. 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.

E.4.5. Sprint Retrospectives

Sprint retrospective should 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 places, you could do it in the cafe or rest room.
- Within this process, the SCRUM master shows the sprint backlog and with help from the team, summerizes the sprint, such as important events and descision, etc.
- Everyone get 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.
- Always keep 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.
**E.4.6. Pair Programming**
All code to be sent into production is created by two people working together at a single computer. Pair programming increases software quality without impacting time to deliver. The best way to pair program is to just sit side by side in front of the monitor and coding together.

**E.4.7. Incremental design**
This means keeping the design simple from start and continuously improving it, rather than trying to get it all right from the start and then freezing it. But still keep in mind the deadline before further optimization.

**E.4.8. Continuous Integration**
Without controlling integration developers test their code and integrate believing all is well. But because of parallel integration there is a combination of source code which has not been tested together before. Continuous integration often avoids diverging or fragmented development efforts, where developers are not communicating with each other about what can be re-used, or what could be shared. Everyone needs to work with the latest version. Changes should not be made to obsolete code causing integration head aches.

**E.4.9. Kanban Board**
All teams have access to whiteboards and empty wall space and plaster it with all kinds of information about the product and project. The process visualisation techniques of Kanban make it ideally suited to co-located teams who are working on a XXXunderstand items that is subject to frequent change.

**E.4.10. Testing**
The testers access the system in exactly the same way as the end users, they will find bugs and then developer members need to fix them. Testing job should be done during every sprint and by other members within the team, not just the developer his/her own. Some detailed description about testing could be seen in Appendix C9.

**E5. 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 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 takes responsibility so that someone else is 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. Here give some examples of some risks and impediments.

### 1. Miscommunication
Misinterpretations of what other team members say and write might stand in the way of a common XXXunderstandingo f what to do and how to do it. This might lead to delays, unwanted results and double work.
**Prevention:** Throughout the project, and especially during weekly meetings, every team members should understands the task given to him or her, by having frequent talks with each other about their tasks within 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 designing the product backlogs.
**Prevention:** Every items 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 product and carry out in the order of priority.
**Correction:** By closely monitoring progress the decision to drop certain requirements can be made in time.

**E6. Roles Assignments**
Before starting a project, there are some roles should be determined within a team and other members. Each of the members could have one or two roles without prejudicing to the development processes.

**Table 1. Role description**

<table>
<thead>
<tr>
<th>Roles</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager (PM)</td>
<td>1. Ensure the completion of the project in the available time, within budget;</td>
</tr>
<tr>
<td></td>
<td>2. Organize weekly meeting and customer meeting;</td>
</tr>
<tr>
<td></td>
<td>3. Assign tasks and check progress;</td>
</tr>
<tr>
<td></td>
<td>4. Manage the time budget;</td>
</tr>
<tr>
<td>Quality Manager (QM)</td>
<td>1. Ensure the quality of the end product and the overall process;</td>
</tr>
<tr>
<td></td>
<td>2. Check that all project documents are consistent;</td>
</tr>
<tr>
<td></td>
<td>3. Arrange internal and external reviews;</td>
</tr>
<tr>
<td></td>
<td>4. Monitor and review all testing activities.</td>
</tr>
<tr>
<td>Product Owner/Customer</td>
<td>1. Make sure that all requirements asked by the customer are represented</td>
</tr>
<tr>
<td></td>
<td>2. Check that the items in the product backlog are user centered rather than technical</td>
</tr>
<tr>
<td>SCRUM Master</td>
<td>1. Ensure that the SCRUM process is followed;</td>
</tr>
<tr>
<td></td>
<td>2. Create the sprint backlog and check that the backlog is updated;</td>
</tr>
<tr>
<td></td>
<td>3. Lead the daily/weekly scrum and make sure that afterwards everybody knows what to do;</td>
</tr>
<tr>
<td>Development Team Member</td>
<td>1. Assisting the Team Leader or Project Manager by signaling problems in an early stage;</td>
</tr>
<tr>
<td></td>
<td>2. Executing plans made by the Team Leader and by the Project Manager;</td>
</tr>
<tr>
<td></td>
<td>3. Keeping track of time spent on various tasks;</td>
</tr>
<tr>
<td></td>
<td>4. Following procedures and plans.</td>
</tr>
<tr>
<td>Team Leader</td>
<td>1. Planning and coordinating team activities;</td>
</tr>
<tr>
<td></td>
<td>2. Providing feedback about team progress to the PM;</td>
</tr>
<tr>
<td></td>
<td>3. Motivating team members;</td>
</tr>
<tr>
<td></td>
<td>4. Chairing internal reviews of the items made by his/her team.</td>
</tr>
</tbody>
</table>

**E7. Other Agile Methodologies**
If you don't satisfied with the combination of SCRUM, XP and Kanban, there are many other agile methodologies and you could always find the resources you need online. Lean, Feature Driven Development (FDD), Crystal, Dynamic Systems Development Method (DSDM) etc.
A quick glance at these methods could see from [https://www.versionone.com/agile-101/agile-methodologies/](https://www.versionone.com/agile-101/agile-methodologies/). You could choose one or some of them according to your need and requirements.

**Appendix F – Use-case based effort estimation**
F6.1. Introduction to use-case estimation

It is quite hard to get a useful (i.e. reliable) cost or effort estimate (measured in system size or work effort) to implement a set of high-level requirements, when almost no design decisions have been made. IBM et al. developed in the 1970s a Function Point (FP) method (http://www.functionpoint.com/) to convert ER- and flow-diagrams into Function Points (FPs), which can be converted into Lines-Of-Code (LOC), and then person-hours. The FP method has recently been adapted for use-cases by Univ. of Linköping (Karner 1993), Univ. of Oslo (Anda et al. 2001) and NTNU (Mohagheghi et al., 2005). Use-case based effort estimation aims to predict the implementation effort - in person-hours - of a requirements specifications given as a set of textually upgraded (also called “structured”) use-case diagrams. Such an estimate covers all remaining design and programming work, plus unit and module testing.

Our industrial results have so far demonstrated satisfactory precision (+-20 %) for systems of moderate size, i.e. about 3000 person-hours or 20,000 Java-LOC.

We need your help! to try out and improve this estimation method, which only takes 30 minutes per upgraded use-case-diagram. The added text (“pseudo code”) in the diagrams will anyhow be useful during remaining implementation work! Some background other estimation methods include the Delphi approach, i.e. letting 3-5 “human experts” try to agree upon a common prediction - or simply use the median. There is also the Work Breakdown into Subsystems, which at least assumes a high-level design specification. Furthermore, estimates done by human experts often fares better than estimates built on formal models. Lastly, Prof. Magne Jørgensen at the Simula Research Laboratory outside Oslo claims that the industrial overrun of cost- or schedule estimates in software projects is about 30% - and has remained so for 30 years (Jørgensen 2005).

Findings from the literature: An industrial software developer spends 40% of his time on Requirements and Design, 20% on Programming, and 40% on Testing and documentation. Given a work year of 1500 person-hours, he/she will annually produce 4 lines of C-code (C-LOC) per person-hour for mega-LOC systems. This means 6000 C-LOC or 3500 equivalent Java-LOC, or 60 Function Points per year per person. The number of defects “injected” by programmers in this software is about 4 per Function Point, or 40 defects per 1000 C-LOC, or 240 (6’40) defects totally per year. Of these defects, 204 (86 %) are pre-release ones, each costing 1-3 person-hours to correct. However, 36 (15%) of the defects will survive into the first release, and each will later cost 10-30 person-hours to correct (Jones 2007). This means that almost half the programming effort goes to defect correction!

F6.2. More on the estimation method

The idea behind use-case based estimation is to add textual specifications, written in “pseudo-code”, to document and thus upgrade the graphical actor and use-case models. This extra textual information is used to classify each upgraded actor and use-case model as either Simple, Average, or Complex - respectively with "cost points" 1, 2, 3 for actors and 10, 15, 20 cost points for use cases. From the aggregated and weighted cost points, a total effort estimate in real person-hours can be produced. The estimation method takes only 30 minutes to perform, based upon upgraded use-case diagrams. The estimation method may also use a spreadsheet (www.idi.ntnu.no/grupper/ss/publ/reidar/us-ProjectEstimateMethod2011v2.xls), or you can do the conversion yourself with a pen and pencil in the same time. You should later compare (“cross-check”) the effort estimate against the person-hours actually consumed in the total design, programming and testing of each use-case, or accumulated for all the use-cases.

F6.3. A mini-discussion

Challenges: How to achieve incremental maintenance - i.e., not only for the first release? How to handle reuse from 3rd party software providers, such as Open Source Software (OSS) and Commercial-Off-The-shelves Software (COTS)? And how to scale up to larger systems: - student software has Productivity of 10-12, the smallish “office” systems in the Anda paper have a Productivity of 60-70, while the Ericsson telecom systems in the Mohagheghi paper have a Productivity of 60-70.

F6.4. References


F6.5. Appendix: A small use-case diagram, with extra comments

A more detailed explanation of some points in the estimation method stands below.
1. Make first a well-structured, textual requirement specification, with requirements numbered e.g. as Ra-b.y (a,b:letters, y:digit). This is assumed made on beforehand.

2. Make a graphical use-case diagram in the usual way. That is, an actor is depicted as a "match-stick" figure, and a use-case as an "oval". Lines or arrows between these symbols show relations between the underlying entities.

1. Upgrade the graphical diagrams with textual specification for the actors and users. The actual use-case diagram has 10 use-cases and 3 actors (see example later). You must therefore make textual descriptions for totally 13 entities, see below.

1. For each participating actor, categorize him/her as easy, average, or complex:
   2. A simple actor relates to a single system with a defined programming interface (API).
   3. An average actor relates to either another system through a communication protocol, e.g. TCP/IP, or communicates to another actor via a textual interface.
   4. A complex actor communicates via a graphical interface.
   5. Count up the number of actors in each category.

Ex. Here, the 3 actors are all persons who shall communicate with the system through a graphical interface. Thus, we have 3 complex actors (all are complex, so we do not need textual actor descriptions to clarify this).

1. Similarly, for each use-case, categorize it as simple, average, or complex, depending on the number of major or alternative transactions in the event flow. A transaction is defined as an event, that occurs between an actor and a system:
   2. A simple use-case has 3 or fewer transactions.
   3. An average use-case has 4 to 7 transactions.
   4. A complex use-case has 8 or more transactions.
   5. Count up the number of use-cases in each category.

Ex. The use-case "Transfer application to another clerk" is one of 10 such, and is marked in the diagram with **. It contains totally 10 transactions – 7 in normal event flow and 3 in its only variation flow (5a1-5a3). This use-case must therefore be categorized as complex. The other 9 use-cases are up to you to categorize. The spread-sheet has tentatively been filled up with 5 complex use-cases, 3 average and 2 simple.

1. A total of 21 (!) context factors to express the effect of technical and environmental issues. These must be provided by you in the general case. In the spread-sheet, all these are just pre-set to 3, i.e. "neutral", giving an overall multiplicative context factor of 1. So just drop these!

1. Summing up. We first multiply the actor numbers (0,0,3) with "cost points" 1,2,3 for respectively Simple,Average,Complex actors, i.e. 1*0 + 2*0 + 3*3 - or totally 9 such points. We then multiply the use-case numbers (2,3,5) with cost points 5,10,15 for respectively Simple,Average,Complex use-cases, i.e. 5*2 + 10*3 + 15*5 or totally 115 points. The total is 9+115 = 124 so-called Use-case-Cost-Points (UCP). Congratulations, you are now done!! That is, only six "numbers" on actors and use-cases must be provided by you.

2. Productivity per UCP. At last, a Productivity factor must be multiplied with UCP above to get an effort estimate in person-hours. With a Productivity of 10, we will get 10*124 = ca. 1250 person-hours!!

1. Apply the calculated effort estimate to either:
   a. Down-scale the requirements: Your group project will have a maximum budget of about 1800 person-hours, with perhaps 1/3 or 600 person-hours for implementation. Thus, your requirements must be trimmed to almost half. And this is definitely useful to know beforehand!
   b. Start implementing, record the actual effort – and compare with the estimate!

F6.6. Add-on to use-case example
Graphical use-case diagram, with text specification added for 3 Actors and one Use-case (**).

Example of added textual specification for actor A1.Clerk:

<table>
<thead>
<tr>
<th>Actor ID:</th>
<th>A1.Clerk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Decides who will have a bank loan, specified by a loan Application.</td>
</tr>
<tr>
<td>Examples of actions:</td>
<td>Has a full graphical interface, to help with getting all the details of a loan Application in place.</td>
</tr>
</tbody>
</table>

Example of added textual specification for actor A2.Provider:

<table>
<thead>
<tr>
<th>Actor ID:</th>
<th>A2.Provider (of capital)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The capital part of a bank that actually pays out accepted loans.</td>
</tr>
<tr>
<td>Examples of actions:</td>
<td>Has a full graphical interface, to help with the legal parts of an accepted loan Application.</td>
</tr>
</tbody>
</table>

Example of added textual specification for actor A3.Customer:

<table>
<thead>
<tr>
<th>Actor ID:</th>
<th>A3.Customer (in bank)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Wants a bank loan, as requested by an emitted loan Application.</td>
</tr>
<tr>
<td>Examples of actions:</td>
<td>Has a full graphical interface, to help an actual Customer to fill in the application.</td>
</tr>
</tbody>
</table>

Example of added textual specification for a simple use-case:

<table>
<thead>
<tr>
<th>Use-case name:</th>
<th>Transfer loan Application to another clerk. **</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors related:</td>
<td>Clerk (only one actor).</td>
</tr>
<tr>
<td>Trigger:</td>
<td>Some parts of an Application need to be discussed by other clerks.</td>
</tr>
</tbody>
</table>
**Pre-conditions:** <that must be satisfied before the use-case can start execution>.

The actual clerk must be logged on the system.

**Post-conditions:** <that must be satisfied before the use-case can finish execution>.

The Application must be stored in the database with a valid and consistent state, and has been assigned to a specific clerk or a group of similar clerks.

**Normal event flow:** A list of transactions that will be executed in a normal event flow of the use-case.

1. a. The clerk announces that he/she will transfer an Application under treatment to another clerk.
   b. The system presents the name of the applicant and the reference number of his/her Application.
   c. The clerk checks all this information.
   d. The system presents a list over suitable groups of clerks and the customers now assigned to each group.
   e. The clerk proposes another clerk, and possibly a second one just in case.
   f. The (previous) clerk asks the system to transfer the Application to the proposed clerk.
   g. The system transfers the Application to the proposed (new) clerk.

**Variations in the event flow:** A list with descriptions of possible transaction variations (not necessarily errors) in the normal flow of the use-case.

5a. The previous clerk may order the system to generate an email message to notify the new clerk about the transferred Application.
   5a1. The previous clerk asks for such notification.
   5a2. The system generates an email message to the new clerk.
   5a3. The use-case continues from point 7.

**Related information**

**Comments on the description of event flow:**

- The event flow shall be described as a list of transactions on the form:

  <no> <event description>.

- An <event description> shall be on the form:

  <actor> <verb> <direct object> <prepositional sentence>

- Such an <event description> may be one or more of the following:

  - An actor sends a request or data to the system.
  - The system validates the received data.
  - The system changes its internal state.
  - The system responds to the actor with the result.

**Appendix G – Software security**

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

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 (such as firewalls and antivirus 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.

**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 (see e.g. [http://www.drdobbs.com/the-7-touchpoints-of-secure-software/184415391](http://www.drdobbs.com/the-7-touchpoints-of-secure-software/184415391)) 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.
Software security games for analysing 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 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 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. Full details on how to play Protection Poker can be found here: http://www.sintef.no/protection-poker.

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 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. Details on how to play the Microsoft EoP game can be found here: https://www.microsoft.com/en-us/sdl/adopt/eop.aspx (see the related downloads part)

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 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 mnemonic is STRIDE (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

These are explained here: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx. 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 (file:///C:/Users/ingat/Downloads/OWASP%20Top%2010%20-%202013%20(4).pdf)
- For a more general overview see the taxonomy of security errors known as the seven kingdoms (https://buildsecurityin.us-cert.gov/sites/default/files/balttax-11-taxonomy_0.pdf).

Appendix H – Guest Lectures, meetings, presentations and rooms