

Project PM: OO programming project, TDA367/DIT212, LP4 2015

General

- The project group's first task is to decide on one of the project proposals on course page (or better if you have an idea of your own). This isn't critically related to the grade, almost any application can be "complexified" by adding more requirements.
- Second task is to name the project. The group number (confirmed by us via email) and the name must be used in all communication with assistants, course responsible, etc.!
- The third task is to setup the project site. The project must use the Git versioning system (use Google Code, GitHub or other).
- Final task is to send a mail to course responsible (hajo@chalmers.se) with the following content; Group name and number, selected project (very short description), URL to Git repository and lastly info about all members. The info format should be (phone optional);

```
cid, Lastname, Firstname, email, pnumber, phone // Contact person
cid, Lastname, Firstname, email, pnumber
cid, Lastname, Firstname, email, pnumber
cid, Lastname, Firstname, email, pnumber
```

Requirements

This is what you are supposed to handle in. All required documents and code should be downloadable from the Git repository. It's assumed that the branch "master" is the final delivery.

The application

Your group is supposed to develop a standalone desktop Java application with a graphical user interface. The application must use some kind of MVC design, either an "in-house" solution (you code it yourself, full control, but takes some time) or possibly use some framework (lack of control but possibly faster). Using suitable libraries or frameworks will put an edge on the project.

The project must be possible to run on Win/Mac/Linux. There should be script to launch the application if needed. If additional information is required to run the project

there should be a README-file explaining what to do.

You may extend the application (client/server, database, etc.) but note;

- It's much harder to get a clean design when combining different technologies.
- It's *not* necessary to extend to get the highest grade.

The documentation

All documentations should be in pdf format. Standard templates should be used (see course site).

- The RAD and SDD
 - A section in RAD or SDD should be short and concise probably require at most 1-3 paragraphs. If something isn't applicable just add a NA (not applicable) under the actual section (the sections are there to guide you, they are not mandatory).
 - UML-diagrams should be on class/package/module-level. We are normally not interested in variables and methods except for methods in interfaces.
INITIAL UML SKETCHING USING PAPER AND PEN IS OPTIMALLY EFFICIENT (use tool later to get a nice look)
 - For the dynamic model (RAD) it suffices with two sequence-diagram.
 - NO auto generated UML (other UML welcome).
 - NO Javadoc.
- The Use cases
 - There should be at least 4 documented use cases (texts).
- The group meeting agendas.

Project grading

It's very hard to formulate exact criterion's for the projects but if a project consists of less than 2804 SLOC¹ (source lines of code) including comments the project should have other qualities that compensate for the "smallness" of the project. That is, at least 701 SLOC/person. The official SLOC calculator is gitinspector (link on course page). Here are some points we will consider:

BASIC: Size of application (classes), Total number of code lines (incl. test, Java),
 Other code (XML. ...), Code style (naming,...) Development environment
 organization (packages etc.) Other

¹This is a very crude measurement, but it gives you some idea of what we expect

DESIGN: Quality of analysis model (compare use cases, class diagram and code), Partitioning, Possible to find cohesive parts of appl. (high cohesion), Layering: Possible to identify abstraction layers, Clean subsystems, Advanced design (plugins, etc...), Comments from tools: Findbugs, STAN, Exception handling, Other

IMPLEMENTATION: General technical level for implementation, Abstraction: Use of interfaces abstract classes, generics, ... Motivated use of DP, Clean MVC model (or other clean solution), Other

GUI: GUI (simple/advanced, modular), GUI features (Animations/2d/3d libraries), Other

TEST: Unit tests (how many, quality), Test code separated from appl. code, Code coverage (Jacoco), Other

FUNCTIONALITY: Number of working use cases (compare RAD, use cases and handled in list from presentation), Average complexity of use cases, Other

ADVANCED: Special features (distributed, i18n, other...), Use of external libraries, Use of frameworks, Other

DOCUMENTATION: RAD (Would customer understand) SDD (Possible to understand, compare implementation), Use cases Agendas (possible to trace the process?), Traceability (from use case to code), Class comments for all classes (purpose of class, class used by?), Other

Examples

Below are some typical examples (non exhaustive).

U (U)

Not possible to run. Project too small, too few use cases, design too strange/bad (hard to understand). Documentation missing or poor. Not possible to trace any process from agendas.

Grade 3 (G)

A project fulfilling the base requirements for all grades. Which means: Functionality; 5-10 use cases implemented and working. Simple but functional GUI. In-house MVC. Clean implementation of at least one subsystems, application uses interfaces to reduce dependencies. There are some usable tests. Documentation is short but correct (and in sync. with the application). It's possible to trace requirements and follow the process. The presentation is ok.

Grade 4 (G)

(NOTE: This is not the union of ..., it's a list of possibilities) A somewhat larger application with a more sophisticated design, possibly use of (needed) design patterns. Solid code and packaging, everything is easy to locate. Clean subsystems. More functionality (use cases) implemented, possibly attention to non-functional requirements. Good control over dependencies, clean interfaces. Possibly use of external libraries. A more advanced GUI. External configuration and data. Test suites cover a lot of application code. Documentation is short and correct and obviously useful for others. The presentation gives a good view of the strength and weakness of the application.

Grade 5 (VG)

Like grade 4 but even more and with higher technical level. Possibly some kind of modular design with plug-ins, advanced use of frameworks.