2 Project guidelines¶

Contents

Important Dates:

- **Kick-off session:** Tuesday, October 19, 2021 at 14:15
- **Project topic/group questionnaire (deadline):** Friday, October 22, 2021 at 23:59
- **Project group distribution and confirmation:** Tuesday, October 26, 2021 at 23:59
- **Project plan submission (commit) to git (deadline):** Friday, November 5, 2021 at 23:59
- **Project final commit to git (deadline):** Friday, December 12, 2021 at 23:59
- **Project evaluation (deadline):** Friday, December 17, 2021 at 23:59

Error

If you have not been able to collect enough points from the Exercises, or you feel like you will not be able to contribute to the project implementation, **please consider your possible group members, and do not sign up for the software project.**

2.1 Choosing topic and group¶

After you select any two of the project topics, you are required to go ahead and fill the Project topic/group assignment questionnaire. In Project topic/group assignment questionnaire, you are also required to indicate your classmates that you have agreed to work with.

Important notes

1. Please indicate your (group's) two preferred topics (in the order of preference) in Project topic/group assignment questionnaire **before Friday, October 22, 2021 at 17:59.**
2. If you don't have enough group members, just indicate the names you know (even if it is just yourself). We will then do some matchmaking to fill the remaining slots for your group. In all cases, we will aim to have four people in all groups whenever possible.
3. If you wish to propose your own topic, **you must have a group of four people**. Guidelines for please see proposing a new topic subsection of the project topics section.

2.2 Topic and group assignment¶

- After submitting your project group/topic preferences, the course staff will assign you to a group.
- Each group will have an *advisor*, who will act as a project owner with the following responsibilities:
  1. Creating a git repository in `version.aalto.fi` for your group. The repository has a template layout, and **you are required to use this template for your implementation**.
  2. Following your progress in git commit history and `Meeting-notes.md`.
  3. Organizing online meetings to guide you through difficulties.
  4. Organizing online sessions similar to Exercise sessions for all the groups working on the same topic.

2.3 Project plan¶

Your first task as a group is to prepare a project plan. The plan should be a PDF document describing the scope of the project, major architectural decisions, preliminary schedule and distribution of roles in the group, design rationale and so on.

The project plan should describe at least the following aspects:

- Scope of the work: what features and functionalities will be implemented, how is the program used, and how does it work
- The high-level structure of the software: main modules, main classes (according to current understanding)
- The planned use of external libraries
- Division of work and responsibilities between the group
- Planned schedule and milestones before the final deadline of the project

Caution

The document should be roughly five pages long, and **should contain mandatory diagrams illustrating the program design (for example, the planned class relationships)**.

As the project progresses, there may be changes relative to the project plan, and that is fine. The actual project implementation details will be described in the project documentation.

Important

The project plan must be committed to the Git repository in the `plan/` directory.

In the week following your plan submission, the advisor will provide feedback on the project plan. If needed, the group or advisor can request a meeting to discuss the plan.

2.4 Project implementation¶

1. You must use the Git repository that will be created for you. An introductory material on Git is available in the Git section of the next module.
   - In order to clone the created repository, you are required install `git client` to you computer. Follow the git installation instructions if you do not have an already installed `git client`.
   - After you assigned to a group and topic, you will receive the invitation for joining a repository. You must accept that invitation. After joining to the project, and possibly arranging `ssh keys`, you should clone the repository into your local development environment.
   - The repository will have the following directory structure:

     > ../_images/project-repo-template-directory.png
     > Project repository directory structure¶

     You are expected to obey this layout and use the folders for the intended purpose.
   - Each sub-folder has a `readme.md` file which contains information about the content of the folder.

     Note

     `md` files follow Markdown language syntax and they can be easily rendered to HTML using several different tools. The most notably, Visual Studio Code supports Markdown out of box.

     It is strongly recommended to us Markdown syntax when modifying these files.

     It is to be noted that Markdown has a very simple syntax, and you are not required to use all of its features.
2. The generated Git repository will have a single branch `master`. You are required to use this branch as the main production branch of your project, and you are free to create any number of topic branches. See Git branching - branching workflows for terminology and a brief introduction to common branching workflows.
3. The repository root directory contains general files that should be modified during the course of the project.

   Important files

   1. `README.md`: General information about the project.
      - **You are required to modify this file just after you are assigned to a project group and a topic.**
      - After selecting the build automation tool, you are required to add build instructions in this file.
      - You may also note down **TODO** list (with a date) in this file.
   2. `Meeting-notes.md`: The meeting notes of weekly group meetings.
      - **You are required modify this file after each weekly meeting.**
      - The meeting topics are provided below, and problems and hypothesized solutions are summarized in this file.

        > It is important to assign group members to identified problems.
      - The commit messages should be aligned with the meeting notes.
   3. `Makefile`: Project makefile for `make` build automation tool.
      - It is not mandatory to use `make`, instead `CMake` can be used. Both options are introduced in the Build automation tools section of the next module.
      - If you choose to use `CMake`, delete `Makefile` from the repository, and create and commit `CMakeList.txt` file.
      - If you choose to use `make`, you are required to modify this `Makefile`. Although it is easier to put the `Makefile` into the `src` folder, it is a better practice to provide a project level `Makefile` in the root directory, and call other `Makefile` in the subfolders from this one.

2.4.1 Weekly meetings¶

The groups should meet weekly. The weekly meeting does not need to be long if there are no special issues to discuss and can be taken remotely as voice/video chat on the group Teams channel (or Zoom or other similar tools), preferably at a regular weekly time.

Everyone may not be able to participate in all meetings, but at least a couple of members should be present in each meeting. Regular absence from the meetings will affect individual evaluation.

In the meeting, the group updates:

- What each member has done during the week
- Are there challenges or problems? Discuss the possible solutions
- Plan for the next week for everyone
- Deviations and changes to the project plan, if an

After the meetings, the meeting notes will be committed to the project repository in the `Meeting-notes.md` file.

- The commits within the week should have some commit messages referring to the meeting notes so that the project advisor can follow the progress.
- **The meeting notes should be in English.**
- See the `Meeting-notes.md` in the repository for required content of a meeting log.

2.4.2 Development practices¶

2.4.2.1 Coding style¶

For any software development project, it is important to decide on the coding style and source code organization beforehand. This does not only make your development consistent, but also drastically increases code readability.

- In this course, we recommend to follow Google C++ Style.

  This style is used by many open-source projects and it is worthwhile to familiarize yourself with it.
  - cpplint can be used to check your code for style errors. This tool can be called from many development environments to perform the style checks.
  - C/C++ extension for Visual Studio Code supports source code formatting out of box based on clang-format.

    In order to use formatting feature with `Google style`, modify `C_Cpp: Clang_format_style` preference to `Google`, and state when the formatter should run (`formatOnSave` and/or `formatOnType`).

    Hint

    In Visual Studio Code, open `Preferences` and write `Clang_format_style` in the search box. Then, follow the instructions on the preferences page to configure `Google` style.
  - You can install cpplint extension to Visual Studio Code, and check your files for style errors.
    1. Install `cpplint.py` to your default python distribution by following the instruction provided in extension front page.
    2. Install and configure the `cpplint` extension.
    3. Run the `cpplint` by opening command palette
       - Click `View - Command Palette` in the menu bar, or
       - Use CTRL + ⇧ + p (⇧ + ⌘ + p in MacOS) key combination.
    4. Write `cpplinter: Analyze current file` in the command entry box `>`.

2.4.2.2 Source code documentation¶

Since you will be implementing the project as a group, it is a very good idea to document your source code to allow your group members to quickly check what a certain function is intended to do, and how it should be used. Although this can be achieved by appropriately commenting the source files, the developers should later create a source code documentation. In order to combine both of the efforts, documentation is usually generated from the annotated source files.

In this course, we recommend using Doxygen for this purpose. It is the de facto tool for generating software documentation, and has very simple source code comment annotation requirements. You can check the Doxygen comment blocks to see how the source code can be easily annotated.

Several development environments support Doxygen comment blocks out-of-box including Visual Studio Code. Although Visual Studio Code can render the Doxygen comment blocks when you hover your mouse over a commented function, it does not help you for creating required comment blocks. However, there are several extensions that can accommodate you with such a functionality, for example Doxygen Documentation Generator.

1. Install the Doxygen Documentation Generator extension.
2. When commenting your source files, start typing `/**`, and the extension will automatically create Doxygen comment blocks for the software construct in the next line.

In order to create the final source code documentation, you should follow the instructions in Doxygen getting started pages.

2.4.2.3 Using external software libraries¶

When designing and implementing your project, you will need several libraries. Several important libraries are listed for each topic at the end of their description. The most important three of these are presented in Recommended libraries section. These software libraries, as described in Software libraries section of the next module, can be linked along with your source code to generate binary files.

After selecting suitable library, you should use them appropriately in your project. Instructions on how to link a library in your executable is described in Software libraries section. However, it is still required to distribute some of the libraries and associated header files along with your source code. Please use `libs/` folder of the project repository for this purpose.

2.4.2.4 Testing and validating¶

In software development, a development stage is followed by a testing phase, where the functionality is tested against the requirements. This can be achieved by creating simple executables that:

1. Checks the operation of the development under required operation conditions.
2. Checks and reports the functionality under abnormal operation conditions, in which the expected argument ranges are outside of the assumed ranges.

A good practice is to validate these assumptions just after completing a significant part of the software component. For this purpose, a simple source file with `main` function is created and compiled. After running the test, the identified errors are fixed.

Hint

For each unit test, it might be a good idea to run Valgrind memcheck to see whether the dynamic memory allocation/release is done appropriately.

Important

Test files should be placed under `tests/` folder. The testing methodology and the summary of the test results should be included in the Project documentation.

2.5 Project documentation¶

After completing the development, you must create a documentation for your project. The documentation should be a PDF document, and should be composed of following parts:

1. **Overview**: what the software does, what it doesn't do? (this can be taken/updated from the project plan)
2. **Software structure**: overall architecture, class relationships diagrams, interfaces to external libraries
3. **Instructions for building and using the software**
   - How to compile the program ('make' should be sufficient), as taken from the git repository. If external libraries are needed, describe the requirements here
   - How to use the software: a *basic user guide*
4. **Testing**: how the different modules in software were tested, description of the methods and outcomes.
5. **Work log**: a detailed description of the division of work and everyone's responsibilities. For each week, a description of what was done and roughly how many hours were used for the project by each project member.

If you have used Doxygen for documenting your source code, **a separate PDF file must be generated.** A well organized software documentation might contribute to your grade by 15%.

Important

The project documentation must be committed to the Git repository in the `doc/` directory.

Caution

Project documentation and final developments should be committed to the Git repository before the specified deadline.

The committed changes after the deadline will not be considered. Therefore, no changes should be committed to the repository after the deadline.

2.6 Project evaluation¶

As indicated in the course syllabus, the course grade is calculated using a weighted average of the following grades:

1. Exercises **(45%)**
2. Software project **(45%)**
3. Peer evaluations **(10%)**

2.6.1 Software project evaluation¶

For project evaluation, the following evaluation matrix will be used as a guideline. For each category, a subgrade between 0 and 5 is given, and the project grade is calculated as a weighted average of all the categories. The project grade constitutes 45% of the total course grade.

Caution

The guideline below is for guidance only, and the actual evaluation is based on overall judgment on each category. For each category, Grades 2 and 4 are also in use.

Project category	Grade 1	Grade 3	Grade 5
Project plan [10%]	Minimal description written	The project plan contains the requested parts but it is not complete. For example, libraries unspecified, functional requirements unclear, division of work not specified, etc…	The project plan contains the required descriptions and gives a convincing description of features, technical ideas, and work plan for the intended outcome.
Design [15%]	The software does not apply coherent structure, external libraries are not used appropriately, design principles are not documented.	Basic class structure exists, C++ standard and third-party libraries are used where appropriate (e.g. for containers)	Advanced C++ constructs are used appropriately (abstract classes, inheritance, templates, algorithms), use of libraries is good. Design is well documented and justified. Design is extensible for future additions.
Working practices [15%]	Use of git is inappropriate (confusing commits, unclear log messages, etc). The distribution of work is unclear. Communication with the advisor is lacking. No steps are taken towards testing the quality of implementation.	Git is used appropriately. Communication between group and advisor is regular, and a well-balanced work plan is maintained throughout the project. There is documentation about the testing of the software.	The use of Git features and version management tools is good (e.g. feature branches, issue tracking, etc.) Work distribution is well tracked and documented. Testing and quality assurance is well documented, and perhaps automated.
Implementation [20%]	Building software is difficult. The software does not work robustly and crashes e.g. with unexceptional user behavior. Memory management is inadequate	There is a make/Cmake or equivalent for easy building of software. The basic operation works without crashing. There are no problems with memory management.	Make/Cmake works robustly and can adapt to different environments. The software works robustly also under exceptional user behaviors. Memory management is pedantic and well documented, e.g. smart pointers are used and RO3/5 is followed.
Topical features [40%]	Minimal features from the topic description are implemented.	Basic features are implemented, and they work.	Many additional features are implemented, and they work robustly.

2.6.2 Peer evaluation¶

The 10% of this course is collected by reviewing and commenting the other projects and own group's members.

Your group will be provided three evaluation forms for reviewing and grading other groups' projects. Group members will get one point for each sufficient evaluation.

You will get maximum 3 points for other groups' projects

Your will also need to evaluate your group member in a confidential evaluation form.

2.7 Frequently asked questions¶

- *Can we implement our software on platform X?*

  A: Our wish is that the software works on Aalto Linux machines, using the readily installed libraries there. You must negotiate with your project instructor on exceptions (e.g. using Windows instead).
- *I work in company X on their software, can I make my project on our software component Y?*

  A: Likely not. All four project members should be able to fairly participate in the project, and in many cases, the rights of the other members would be unclear if the code becomes a part of a commercial product.