

Advanced Techniques and Tools for Software Development - Exam Guidelines

These are a few primary guidelines for the exam. It also contains information about the 9 credit exam for the previous years.

These are not meant to be exhaustive: we rely on common sense.

Remember that these guidelines will make sense only after you study the contents of this course.

The exam date on the official website is fictitious: the actual date is scheduled by the teacher and the student(s). However, it is mandatory to register for an exam session using the official website. After you have registered, the exam date can be scheduled in that session (or outside the session, before the next session, though it is preferable to do the exam in the session "window"). Thus, even if you are not 100% sure to give the exam in a given session, please register for that session anyway. There is no penalty for registering for a session and not giving the exam. For the above reasons, I suggest you register for the first session. This way, you will not forget to register when you decide you want to give the exam.

The exam consists of

- a project to be developed at home
- an oral exam

Ph.D. students only need to do the project and do not have to register for an exam session.

The project must be developed by a single student.

The student proposes a project to the teacher by email. The teacher must approve the proposal (possibly with some modifications). Then, the student can start working on the project.

However, I suggest you work on "a proof of concept implementation" of the final project before you propose it (this way, you have confidence in the feasibility of your proposal). A "Proof of Concept" (PoC) is a realization of a small and simple project to demonstrate its feasibility. A proof of concept is usually small and may or may not be complete. The PoC must not be proposed nor sent to the teacher. You do it to have confidence in the feasibility of your project before you propose it to the teacher.

IMPORTANT: Before you start thinking about a project proposal and working on the project, ensure that you have already studied and well comprehended all the course topics.

The complexity of the main code is not essential (indeed, we suggest you implement a simple application). At the same time, the techniques taught during the course must be applied (the following list is not exhaustive, and it only contains the main mandatory parts):

- Test-Driven Development, unit tests, integration tests, and end-to-end tests
 - Mocking should be used appropriately
 - The "shape" of the testing pyramid should be taken into account and respected
 - The strategies for testing will be crucial for the evaluation of the project
 - Tests must be readable and easily understandable (use additional testing libraries for this aim)

- Code Coverage (100% is required)
 - a part of the code can be excluded from code coverage, according to the suggestions explained in the book
 - Code Coverage report must be available on the web, e.g., by relying on Coveralls or some similar services
- Mutation Testing (“no survived mutant” is required)
 - Mutation Testing must be configured appropriately to avoid a tremendous amount of time for the build (focus on the essential parts of the code with business logic and apply mutation testing only there)
 - Some mutators can be excluded, but only by providing excellent motivations.
- Docker must be used in the project, especially for testing purposes when a server is needed (e.g., a database). Docker container(s) must be started automatically during tests, both when running tests from Eclipse and Maven.
- Code Quality with SonarCloud (0 technical debt is required, 100% code coverage also in SonarCloud is required)
 - Exclusions of rules and parts of the code are permissible if done appropriately and with excellent motivations
- The code must be clean and neat, including excellent and consistent formatting and indentation of the code
 - do not mix spaces and tabs (use the command "Show Whitespace Characters" in Eclipse)
- Code modularity, which includes the structure of the project; e.g., in a Maven project, you might want to split the whole project into several submodules appropriately
- Build automation (with Maven)
- Continuous Integration (with GitHub Actions)
- VCS (Git & GitHub)
 - Although a single student works on the project, GitHub pull requests should be used for implementing single features
 - The teacher will inspect the Git history for the evaluation of the project
 - Make sure that no useless file (e.g., generated files) gets into the Git repository
- No *warning* must be present in any file of the project (see also the note about the Java version in the following)
 - warnings can be *suppressed* or ignored only with an excellent motivation

Things illustrated in the book and during the lessons should be starting points. However, we expect the students to be able to perform slight variations (e.g., in configurations) and to use more recent versions of dependencies and plugins.

The implementation of a user interface is a strict requirement. The user interface must be thoroughly tested with the appropriate framework (unit, integration, and end-to-end tests).

You must use Java as the programming language.

Concerning Java, your project must be configured appropriately. In particular, it must be configured with the required version of Java. Use ONLY an LTS version of Java: Java 8 or Java 11, or Java 17. No other Java versions are allowed.

In the case of the 9 credits course, you have to implement a web application. See the end of this document.

The project must be accompanied by a brief report (about 10 pages) describing

- what you implemented
- applied techniques and frameworks
- design and implementation choices
- possible problems encountered and how you solve them

- descriptions of the development (and testing) of the most interesting parts

In the report, DO NOT define tools and mechanisms described in the course (e.g., do not explain what TDD is, etc.)

If you choose to implement a tutorial/project (see also the end of the document), you must provide a document describing step-by-step how to reproduce your project (including explanations of mechanisms used).

If you implement a tutorial, the tutorial document replaces the report document.

In the case of a tutorial, we suggest you use *learning tests* (see the book's final chapter) throughout the tutorial. Even in the case of a tutorial, the code must be provided and fully integrated into a CI process, as detailed above.

Italian students should write the report in Italian.

The complete build and test of the project must also be reproducible easily on the local machine (of the teacher), and the instructions to achieve that must be part of the report. The instructions to run must be straightforward and very short (e.g., a single Maven command, assuming that Java 8, 11, and 17 are already installed on the teacher's machine; you can assume that Docker and Docker compose are already installed on the teacher's machine).

The project and the report must be made available to the teacher only in their final shape, at least 5 working days before the oral exam (remember the date of the oral exam has to be scheduled with the teacher). After the final delivery, no further modification/correction will be allowed. Thus, ensure that everything is correct and working, and all requirements are fulfilled before the final delivery.

The project must be available as a public Git repository (e.g., GitHub).

The report can be part of the Git repository or sent to the teacher via email with the link to the Git repository when the project is finally delivered.

The oral exam consists of questions on the project and on the topics of the course.

During the oral exam, the student must be able to describe any part of the project (i.e., every single line of code or configuration file) and reproduce the build on their computer.

The students might also be asked to implement or modify something on the fly during the oral exam.

During the oral exam, the students must be able to provide formal and exhaustive answers to questions on any part of the course, that is, the book's contents.

In summary, showing a good knowledge of all the techniques presented in the course is crucial.

Once again, keep the application and its code simple and concentrate on all the “surrounding” mechanisms and techniques.

If you fail the oral exam (or do not accept the proposed final grade), you must develop a new project entirely from scratch.

If you fail the exam, you must wait at least one month before doing the exam again.

If this is the last exam before your degree, please ensure you schedule it a few months before the deadlines for your degree session.

Hints for the projects with the corresponding evaluation/rating metrics

The final grade of the exam highly depends on the complexity of the proposed project, besides, of course, the correctness and cleanliness of the project's development (according to the topics of this course).

Note that a low-rating project could still be enough to pass the exam, but, in any case, the final grade for a low-rating project might not be that high and will never be the highest grade. On the other hand, proposing a high-rating project does not necessarily imply a high grade, but it is a precondition if you aim for the highest grade.

Below are a few suggestions for determining whether a proposed project is considered a low or a high rating. These are only a few suggestions.

Low rating:

- A variation of an example seen in the course, without any new technology/tool/framework

High rating:

- A variation of some of the technologies seen in the course or the use of another technology (this requires learning a slightly different and new technology), possibly writing a report on the main features and a step-by-step tutorial to reproduce it:
 - use another database instead of MongoDB, or use several databases (e.g., MongoDB and MySql) in the same application, making it easy to switch among database implementations
 - use advanced features of MongoDB (e.g., relations and transactions), (possibly with a tutorial with learning tests),
 - use a SQL database and the corresponding Java technologies (e.g., JPA, Hibernate, etc., with relations and transactions), possibly with a tutorial with learning tests,
 - use Guice (see the last chapter of the book) to make an application easily configurable at runtime
 - write an application with both a GUI and a command-line UI
 - use BDD with Cucumber (see the chapter "End-to-End tests" in the book)
 - use another framework for BDD (e.g., JBehave, Jgiven, etc.)
 - use a framework for Property-based testing

The domain model of the application must be kept simple.

An example of an excellent domain model is

- "Todo" with a description and a boolean field (e.g., done)
 - "Todo" has a many-to-many relation with "Tag."
- Instead of "Todo" and "Tag," "Student" and "Teacher," or something similar
- 2 entities are enough for the project; in general, try to use at most 3 entities
- Some kind of relations between the entities should be present, relying on the mechanisms provided by the database

9 credits exam

Once again, remember that these guidelines will make sense to you only after you study the contents of this course.

The above rules still hold for the 9-credit exam (including the low and high-level rating concepts). However, for the 9 credits exam, you must implement a web application instead of a desktop one.

You must implement a web user interface (also tested): you cannot simply implement a REST server. End-to-end tests are also a strict requirement.

For the domain model of the web application, refer to what has already been said in the previous parts of this document.

Here are some suggestions for the high-level rating.

High rating:

- Use Spring Boot with one of the following variations:
 - use several databases in the same application, making it easy to switch among database implementations (e.g., for Spring Boot, you might use Spring Boot Profiles or other configuration mechanisms)
 - use BDD
 - use another framework for BDD (e.g., JBehave, Jgiven, etc.)
- Additional testing frameworks that could be used are
 - Wiremock (for mocking the server)
 - Pact (for contract testing, e.g., when implementing both a REST server and a REST client)

IMPORTANT

The following points are necessary for the acceptance of the final project.

The project must be made available as an Eclipse project (or several Eclipse projects if you use multi modules). If you plan to use another IDE, you must still provide Eclipse projects. For this reason, we suggest you use Eclipse from the beginning instead of using another IDE.

Ensure all the Eclipse projects' metadata files are in the Git repository (e.g., .project, .classpath, .settings, etc.).

Ensure all the tests can be run from Eclipse with "Run As -> JUnit Test".

Do not put generated files (e.g., target folder, .class files) in the Git repository.

Do not use frameworks like Lombok or model mappers.

Never put a token or a password as clear text in the Git repository! Use the techniques provided by the used tools and services to encrypt such sensitive data safely.

The teacher will clone the Git repository and import the contents as Eclipse project(s). Once imported, the projects must not contain any errors or warnings. Please, make sure that this is the case (e.g., before submitting the final project, try to clone your Git repository from scratch and import it into a fresh Eclipse workspace where you configured Java 8 or Java 11, or Java 17, depending on the Java version you used).

Remember to put in the Git repository (e.g., in the README.md file) the badges:

- result of the build (the link must lead to the log of the GitHub Actions workflows)
- badge with 100% code coverage (e.g., from Coveralls)
- badge (or badges) linking to the SonarCloud results. Remember: no issues, 100% code coverage, no technical debt, and no code duplication.

Pay attention to the material you find online, which will not necessarily be considered authoritative. When in doubt, ask the teacher. In general, only refer to the contents of the course (book and examples).

If you use some documentation on-line for new mechanisms or using new libraries/frameworks/tools not seen in the course, ensure you cite the on-line sources you used in the report.

Finally, NEVER EVER talk with your colleagues about your project: parts that look similar in different projects will nullify the project! When in doubt, ask the teacher, do not ask your colleagues!