
Handbook for Clients (AGs) in the Bachelor Internship (BP)

Version vom August 28, 2023



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Bachelor-Praktikum
Informatik
bp@cs.tu-darmstadt.de
Fachbereich Informatik

Contents

1 Overview	4
1.1 Dates during winter term 2023/24	4
1.2 Checklists	5
2 Suitable Projects & Project Submission	7
2.1 Suitable Projects	7
2.2 Project Presentation Slides	7
2.3 Project Submission	7
3 Framework and Procedure of the Bachelor Internship	9
3.1 Components of the Bachelor Internship	9
3.2 Time Required for the Group	10
3.3 Role of the Team Supporters	10
3.4 The Agile Software Development Process	10
3.5 Procedure during the Internship	11
3.5.1 User Stories	11
4 Tasks of the client	14
4.1 Project supervision	14
4.2 Resource provision	14
4.3 Utilization and License of the Software	15
5 Quality Assurance (QA)	16
5.1 What is QA?	16
6 Evaluation of the project group	21

Preface

Dear Client (AG),

thank you very much for your decision to offer a project in the context of the bachelor internship (BP) of the Computer Science Department. This handbook provides an overview of the framework and the course of the event, as well as the tasks that fall to you as the client. To ensure the successful completion of your project, we ask you to read this handbook carefully. The overview in section 1 will help you to do so. If you have any further questions, please contact us at bp@cs.tu-darmstadt.de, we are happy to help.

This year, we will be hosting a kick-off meeting for clients where we will provide all the important information about the undergraduate internship and answer questions. We kindly ask you to attend one of the two kick-off dates.

With kind regards, Your BP organization team

1 Overview

This manual is divided as follows: Section 2 provides guidance on suitable projects and details on project submission. Section 3 gives an overview of the components and objectives of the BP, the people involved and their roles, and the software development process. Section 4 explains the tasks that fall to you as the client (AG). Section 5 explains the quality assurance goals and measures you need to establish with your groups. Section 6 supports you in the evaluation of your group.

While all of these notes are important, we especially recommend that you actively use the checklists and to pay attention to the sections on project submission and evaluation at the respective times.

1.1 Dates during winter term 2023/24

Below you will find the schedule for the winter term 2023/24. The most important dates for you:

- **06.10.2023:** Deadline project submission
- **23.10.2023, 09:00** or **02.11.2023 16:00:** Kickoff event for clients (we offer 2 dates)
- **06.11.2023 - 12.11.2023:** First meeting with the group
- **11.03.2024 - 17.03.2024:** Project handover to you by the group

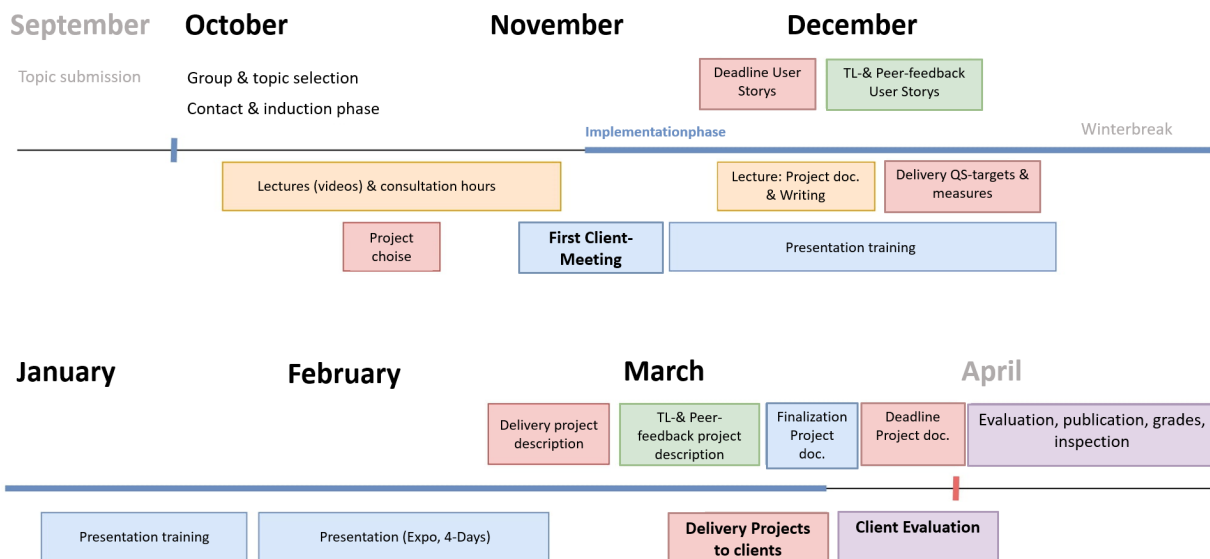


Figure 1: General Timeline of the BP.

1.2 Checklists

Before the start of the lectures

- Obtain current information from the website¹
- If necessary, clarify any questions by e-mail: bp@cs.tu-darmstadt.de.
- Prepare slides for the project presentation.
- Submit topic, for winter term 2023/24 at <https://goto.dm.informatik.tu-darmstadt.de/bp/23-AG/>.
- Receive confirmation of the project from the organisation team.
- Read AG handbook.
- Keep proposed dates (>1, min. 1,5 h) free in the week of the first meeting with the project group.
- Create and upload project presentation.
- Participate in one of the two client kick-off meetings.

The 1st meeting with the project group will take place in the week of 06.11.2023 - 12.11.2023.

- Round of introductions.
- Arrange regular meetings (every 2-3 weeks, second meeting max. 2 weeks later).
- Discuss project idea and clarify questions.
- Clarify scope: programming language, framework, infrastructure, licence, etc..
- Define tasks for the 1st sprint (familiarisation, preparatory programming tasks, etc.).
- Gather initial content for user stories.

Quality assurance meeting takes place during the 2nd meeting. (s. also next section) At least 45 minutes should be planned for QA. The meeting must take place at the latest in the week from 20.11.2023 - 26.11.2023.

- Which aspects of the project need special attention in terms of quality?
- Is there a prioritisation within these aspects that is important to you?
- What are the resulting quality assurance objectives?
- What measures could be taken to achieve these goals (suggestions should come from the group)?

Detailed explanations on QA can be found in chapter 5.

From the 2nd meeting

- Accept completed User Stories.
- If necessary, give new User Stories to the team.
- Prioritise user stories for the next iteration.

¹https://www.informatik.tu-darmstadt.de/studium_fb20/im_studium/studiengaenge_liste/bachelor_praktikum.de.jsp

Project completion

- Agree on the handover date before 17.03.2024 (inclusive).
- Clarify what must be handed in by 17.03.2024 and in which form.
- Determine a score between 0 and 100 for the group by 31.03.2024 based on the evaluation criteria (see Chapter 6) and send it to us.

2 Suitable Projects & Project Submission

2.1 Suitable Projects

The Bachelor-Praktikum is a software development project. A large part of the task should be related to designing, implementing and testing software, and less to researching new issues (in this respect the BP is different from internships and project internships courses that many disciplines offer). Students should be able to complete the assignments without in-depth knowledge of the domain in question. Computer science knowledge is not required on your part - you will be acting from your subject matter perspective. If you are unsure whether your project is suitable for the BP, please contact the organization team.

2.2 Project Presentation Slides

Students select projects according to their interests and prior knowledge. In order to facilitate the students' selection, you as the client provide one set of slides per project. In which the project is presented. The more detailed the slides describe the project, the more likely that you will get a project group that is interested in your topics.

When creating the slides, keep in mind that they will be the basis of the students' choice of project. The slides should put the project in context and describe in as much detail as possible, what is to be implemented by the students. The slides will be provided to students digitally as PDFs, so feel free to include links to existing projects, videos, etc. that are relevant to the project. You are also welcome to create a recording in which you present the slides and present your project. You should then link the recording in the slides as well. In addition, you should include any required previous knowledge and the technologies to be used (frameworks, programming languages, etc.), if they are already known. If you are not yet set on specific frameworks, etc., feel free to add this as well. The set of slides for the project presentation should be uploaded to our online system when submitting your project.

2.3 Project Submission

Registration is done through an online system. If you want to offer a project, please visit <https://goto.dm.informatik.tu-darmstadt.de/bp/23-AG/> and fill in the form. If you want to offer more than one project, you have to fill in the form multiple times.

The form first asks for the contact information (email address and name) of the main supervising person. This person should be available throughout the whole semester for regular (online) meetings with the group.

Additionally, a title for the project and a brief description must be provided. Further details should be included in a project presentation (PDF) (see sec. 2.2). These, as well as the subsequent questions about the nature and implementation of the application, serve to help students choose a project that interests them in terms of content and technology.

Hardware, software or other material that a project team may need to carry out the project must be provided by the client. All necessary equipment and systems must be ready for the start of the implementation phase early November. If you are not entirely clear about what is needed for the project or who should best be approached for this, we will be happy to help. Please mark the appropriate field in the form, and we will contact you.

After filling out and submitting the form, a link will be sent to you automatically. You can use this link to edit the details of the registration afterwards until 06.10.2023. We will check the proposed project for its suitability for the Bachelor-Praktikum and approve it, for which you will receive another confirmation email. If we see any difficulties, we will get back to you.

If you have problems with the application, please send us an email.² An announcement of the project beyond filling out the form is not necessary.

²bp@cs.tu-darmstadt.de

3 Framework and Procedure of the Bachelor Internship

A team of four to five students will work on the project you provide. The employers do the goal setting of the project, discussion of the requirements and the progress of the project with the team, and supervision regarding domain knowledge. The department of computer science does the organization and technical supervision of software aspects. The organization within the groups is supported by student team supporters who are not part of the development team themselves. The bachelor internship is obligatory for students in the study program B.Sc. Computer Science.

3.1 Components of the Bachelor Internship

The bachelor internship consists of four essential components, namely software project, lecture accompanying the project, project presentation and quality assurance:

Software Project The focus of the bachelor internship is your software project. The goal of the group is to implement your requirements to mutual satisfaction, as far as possible in the time and with the students' previous knowledge.

Lecture Accompanying the Project The bachelor internship is an integrated course. The accompanying lecture complements the *Software Engineering* course and teaches and repeats knowledge necessary for systematic software development. In particular, the use of helpful tools for the implementation of software engineering projects is discussed. A further emphasis forms the production of the project documentation.

Project Presentation Over the course of the BP, each group presents their project's content and the approaches to quality assurance (QA) in two short pitches. In preparation for this, students attend a presentation training session. The presentation training and the pitches teach essential presentation skills and are organized by the project support.

Quality Assurance (Document and Evidence) The final major component of the bachelor's internship is the QA of the software. For proper QA, you and the students should agree on three QA goals that the students concentrate on for this project. The QA G goals are achieved by regular execution of QA measures. To document the proper implementation of a QA process, the students must log the execution of their QA measures. At the end of the project, each group must hand in a project documentation. Among others, this project documentation must contain a description of QS goals and measures as well as prove that the measures have been carried out. In order to adapt the quality of the software to the individual project requirements, we ask you to define the key objectives essential for your application together with the group during the project meetings.

3.2 Time Required for the Group

The bachelor internship should be feasible in 1000 person-hours with four students or 1250 person-hours with five students (i.e., 250 person-hours per person). This includes the software creation, training phase, lecture work, preparation of the project presentation and the creation of the QA documents.

3.3 Role of the Team Supporters

During the bachelor internship, the student group is assisted by a student team supporter who (generally) has already completed the bachelor internship and is studying in a higher semester. The student team supporter does not develop the project himself but supports the students in the organization, the first communication with the clients, and implementing the development process. The student's team supporter provides feedback on submissions as part of the lecture but is not involved in grading. If you have any problems with the group, that you cannot solve with the group directly, the student team supporter is your first point of contact.

3.4 The Agile Software Development Process

The goal of the agile software development process is the rapid and purposeful development of software under constantly changing requirements. In order to achieve this goal, it is necessary to apply design principles that make the software flexible, adaptable, and/or maintainable. Therefore, you must be aware of appropriate design patterns. In addition, it is necessary to apply procedures that ensure the discipline required and determine progress. The essential goal is to enable you to guide the development team in the direction that promises the highest value. This requires flexibility in responding to change. Essential points are:

- Regular and frequent collaboration with you as an employer.
- The incremental delivery of new functionalities. This is how project progress is measured.
- Self-organizing teams with motivated individuals.
- Team reflection on development processes.
- The pursuit of technical excellence and sound design.

The development period is divided into iterations (2-3 weeks) in which software functionalities (as user stories, see 3.5.1) are designed, implemented, and tested. At the end of the iteration, a meeting must take place with you as the employer, in which the results are presented. This approach should enable faster testing, a constant consensus on the understanding of the requirements for the application between the you as the employer and the team, as well as a fast reaction to changing

requirements. If a user story planned for the current iteration is not completed before the end of the iteration, this is **not unusual**. It will then be moved to the next iteration.

3.5 Procedure during the Internship


Project start In the following, it is assumed that the technologies to be used (programming languages etc.) are already set – if this is not the case, an exploration phase can be added first, if necessary. The same applies if the team does not have the necessary technical knowledge, or your ideas about the software are still vague at the beginning. In this case, it is advisable to use further techniques for requirements elicitation (e.g., use cases, open interviews or scenario techniques) in order to gain a broad understanding of the application to be developed. During the regular meetings with you, the requirements for the software to be developed are to be determined or refined using User Stories. You discuss the requirements with the students, who then formalize them as User Stories. During the first meeting, an attempt should be made to capture all essential User Stories at a high level of abstraction. At a later point in time, the requirements can be adapted, elaborated and finalized in accordance with the evolution of the project. The User Stories that are to be implemented in the first iteration must, of course, already be suitably broken down and formulated within the first iteration. Based on the identified User Stories, release planning is performed by the students for the entire project (two to three releases). A release is characterized by covering a functional scope that is reasonable from a technical point of view. This plan will then be presented during the second meeting with you.

Process of an iteration At the end of an iteration, you will have a meeting with the team to discuss the results. During this meeting, the team presents the implemented User Stories, which you need to approve. At your request, the team should be able to provide an executable version at the end of each iteration so that you might test it to determine the subsequent development direction, if necessary. If new requirements arise, they must be captured immediately as new User Stories. Furthermore, it needs to be discussed which User Stories will be implemented in the next iteration (planning meeting). It is important to ensure that the team merely focuses on as many User Stories as can actually be implemented in one iteration. If the team's speed is not yet known or varies greatly, it makes sense to assign priorities to the User Stories so that the team gets a clear idea of in what order the User Stories are to be implemented. In particular, large User Stories should be broken down into several smaller ones to control the level of complexity.

Project handover In order to grant the group sufficient time to prepare the documentation about quality assurance, the software project is already due by **March 18**. No implementation work is to be done after that date.

3.5.1 User Stories

User Stories are work packages the team works on within an iteration. The students formalize User Stories, an example of such a User Story can be found in Figure 2.



ID	2
Name	Login
Description	As an administrator, I must be able to authenticate myself to the system using a username and password in order to make changes.
Acceptance criterion	The login dialog is displayed correctly, and it is possible to authenticate as an administrator. Invalid entries are ignored, and normal users do not get the "Admin" role.
Expected effort (Story Points)	3
Developer	Michaela Müller
Implemented in iteration	2
Actual effort (h)	12
Velocity (Story Points / h)	0,25
Comments	-

Figure 2: Example of a formalized User Story.

The essential components of a User Story include the following:

1. A description of the User Story that summarizes a requirement of yours in such a way that the team and yourself remember what exactly needs to be done. User Stories are further broken down into Tasks, if necessary. Breaking down the User Story into Tasks is the responsibility of the group. A proven format for User Stories is "As <user role>, I want <the goal> [, so that <reason for the goal>]". It is important that everyone understands the content and that each person has an idea of what needs to be done.
2. Priority of the User Story or whether the User Story currently is in progress.
3. An acceptance criterion that allows judging whether the User Story has been implemented completely as well as correctly.
4. Estimated effort - this is based on the expected complexity compared to other User Stories, especially compared to already implemented User Stories. It is important that the estimated effort is up-to-date; if there are several iterations between the first estimation of effort and the time the User Story is to be implemented, it may be useful to re-estimate the User Story.
5. Actual effort in clock hours and the velocity (story points / hr.), the latter is necessary for calculating the speed. Discrepancies between estimated and actual effort are to be expected – more in the initial than in the final phase. These variances should decrease over the course of the project.
6. Who (developer) took responsibility for the User Story and for which period of time, and when was the User Story completed/accepted.

Furthermore, it is possible to include the role of the users for whom the corresponding User Story is of particular importance. In general, the collection of User Stories does not require any special tools and the use of – for example – a spreadsheet is sufficient. However, there are also (free) tools on the Internet that can be used (for example, issue tracking in GitLab). At the end of the project, the User Stories have to be handed over to the organizers.

4 Tasks of the client

Below you will find an explanation of the tasks you can expect as a client.

4.1 Project supervision

You should take sufficient time right at the beginning of the project (several hours if necessary) to discuss the essential requirements for the project with the group and also to roughly prioritize them. It is neither necessary nor useful at this time to attempt to discuss or identify all requirements in detail. However, the requirements should be discussed to the extent that the team gets a very good idea of the project as a whole. Requirements can be changed, added to, or deleted "at any time" during the project if needed.

In order for this process to be applied within a project, it is necessary that you commit to regular meetings (ideally every two to three weeks) over the duration of the project. In order for this process to be applied within a project, it is necessary that you commit to regular meetings (ideally every two to three weeks) over the duration of the project. The selection should be made by you from a professional point of view. Please engage with the software regularly to ensure that the application meets your needs and to provide qualitative feedback on the software. This means that before the start of each iteration, you can adjust existing user stories, delete them, or add new ones, if necessary. You should not select more user stories than there is time to implement within the next iteration. Information about how much effort is likely to be required to implement a story and how much time is available within the next iteration is provided by the team.

If you are unable to attend the planning meeting at the end of an iteration, it is your responsibility to appoint a qualified substitute. If necessary, the duration of an iteration can be changed to one, two or three weeks. The iteration that extends over the Christmas vacations will, of course, be extended by the free time.

4.2 Resource provision

In order to get the group started quickly, we ask you to think about the necessary hardware and software requirements before the first meeting with the group. This concerns in particular the provision of necessary resources, as well as the license under which the software is to be developed. Please note that any hardware or software needed by a group to complete the project must be provided by you. The same applies to any rooms that may be required. An infrastructure for communication (Zoom, BigBlueButton) and version management (GitLab) is provided by us/TU Darmstadt, but does not necessarily have to be used. Below you will find a list of frequently needed hardware or software with a reference to possible contact points at the TU Darmstadt:

- Servers allow applications to be accessed centrally, e.g. for web applications the browsers/clients communicate with the server. For development and later productive use it makes sense to

provide a server that is at least accessible via the TU network/the network of your work group. If you have system administrators in your group, they may be able to provide you with servers, e.g. a virtual machine (VM). Furthermore you can ask the ISP for a VM with Linux (Debian).³

- Matlab and other commercial software – please note that if you are using commercial software such as Matlab, you must provide sufficient licenses for the team to work with.

4.3 Utilization and License of the Software

The BP is a mandatory internship. Therefore, the same rules apply with regard to scientific/commercial exploitation as apply to internships, thesis etc. If the work is to be used in a certain way scientifically/commercially after completion, you must mention this when presenting the topic and agree on it again with the groups during the first meeting. There are no requirements on the part of the organization team – however, the organization team may need to be given access to the code if problems arise in the course of determining grades.

To determine the further use of the software, you must agree on a software license with the group. The ChooseALicense⁴ website can help in selecting the license. Below are some common software licenses, and their implications on reuse (sorted from none to very strong restriction):

MIT⁵ – Software license of MIT. The use of the software is possible without any restriction. Allows commercial and non-commercial reuse of the software.

APL⁶ – Apache License (v2.0). Similar to the MIT license, including clauses regarding patenting of the software. Allows commercial and non-commercial reuse of the software.

GPL⁷ – GNU Public License (v3). Open source license. Any reuse of the code is allowed, but it must be re-released.

AGPL⁸ – GNU Affero Public License (v3). Variant of the GPL that is specifically targeted at web applications.

In case none of the licenses mentioned above and none of the well-known open source licenses work for your case, we expect that you and the group agree on a custom license text that the group must add to their project report. This custom license must clarify typical license terms, such as usage and distribution rights, payment, and warranty.

³<https://support.rbg.informatik.tu-darmstadt.de/wiki/en/doku/researchgroups/high-performance-vm>

⁴<https://choosealicense.com/>

⁵<https://opensource.org/licenses/mit-license.php>

⁶<https://www.apache.org/licenses/LICENSE-2.0>

⁷<https://www.gnu.org/licenses/quick-guide-gplv3.html>

⁸<https://www.gnu.org/licenses/agpl-3.0.en.html>

5 Quality Assurance (QA)

In a software project, quality assurance provides a framework for defining and checking properties essential to the project. These properties can be both the describe functionality (e.g., with what accuracy correct results by the software are issued) as well as overall requirements for the system (e.g., how intuitively usable the application is). QA goals describe these properties specifically for your project. We therefore ask you to briefly deal with the following terms to define the important QA goals for your project together with your project group.

5.1 What is QA?

Quality assurance deals with defining quality objectives, applicable standards (e.g., documentation standards and process standards like test protocol standards), introducing quality processes, and ensuring their implementation.⁹

The deliverables in the BP include a statement of the planned QA objectives and measures and a description of implementation in the project documentation. In addition, a quality assurance pitch must be prepared and delivered.

Software Quality Goals of the Agile Software Development Process

The following goals are suitable for use with the agile development process that is employed in most cases during the Bachelor-Praktikum.

Important remark: During the Bachelor-Praktikum, we use the term 'Software Quality Goal', 'Software Quality Property' and 'Software Quality Factor' synonymously for simplicity.

The following descriptions are based on ISO25010 [1], the lecture *Software Engineering – Software Quality 2017* by Richard Bubel und Dominic Steinhöfel and material from the Bachelor-Praktikum Summer Term 2019 by Richard Bubel and Dr. Michael Eichberg.

We also give examples of application scenarios of the respective goals. However, the use of a specific goal should be re-considered and justified for each individual project.

Usability (German: Benutzbarkeit)

Usability deals with the software's ease of use. This includes several factors: learnability, recognizability (e.g. via the use of widely accepted GUI elements), user error protection, user interface aesthetics and others. Learnability describes the degree to which the software's target group learns to use it efficiently and effectively. This means ideally, the users' training period with the software will be short.

⁹Ian Sommerville. *Software Engineering*. Pearson 2012, 9. current edition.

Application Persons with differing background knowledge or an often changing user group will use the software. Therefore, it should be intuitive to operate and easy to learn.

Measures User studies are often used to ensure usability. Please be aware that the client's (German: Auftraggeber*in, short AG) feedback is not enough on its own to ensure this goal. Usually, the user study contains five or more participants and should be conducted early enough during the project that its findings can be used to improve the software. In contrast to other, continuously conducted software quality measures, user studies can generally only be performed once or twice during the Bachelor-Praktikum project.

Another measure to ensure usability is to regularly calculate metrics like the amount of interactions or time needed for task completion.

It is also possible to define and regularly check the compliance with usability guidelines.

Efficiency (Effizienz)

Efficiency describes the cost-effective use of resources like memory usage, CPU usage or runtime.

Application Efficiency as a goal might be derived from hardware constraints of a project. There might also be runtime requirements given directly by clients or derived indirectly from user acceptance criteria.

Measures To ensure efficiency it is first necessary to define criteria e.g. thresholds for runtime or memory usage. For some projects, these can be derived directly from hardware constraints.

After this, measurements (of e.g. runtime or memory usage) will have to be taken regularly (e.g. at the end of each iteration) to determine if the implementation will have to be optimized to fulfill the previously defined criteria.

Functionality (Funktionalität)

Functionality describes the degree to which the application's functions fulfill its requirements. This means in particular that the software produces correct results with a certain accuracy. Functions appropriately fulfill user goals i.e. the user is able to accomplish their tasks but is not being hindered by unnecessary functions.

Do not confuse this goal with the goal *correctness* (see section ??) which applies to correctness with regard to a given specification.

Application Functionality is one of the most important software quality goals for validating a software's functions and is appropriate for most projects.

Measures Functionality can be ensured with automated tests that check that methods or classes work as intended. In this case test coverage can be calculated to assess the necessary amount of tests.

Furthermore, manual tests can be conducted by developers to check functional correctness of graphical user interfaces (GUI).

Both kinds of tests need to be executed regularly (e.g. after implementing a feature or user story). It is important to clearly describe consequences of test failures (e.g. assign a specific person to correct the error within the next iteration).

Compatibility (Kompatibilität)

Compatibility describes the effortlessness of combining software systems or components with each other. This means they can easily exchange information (*interoperability*) and share resources without hindering each other (*co-existence*).

Application The software or its components need to exchange information with different systems or components or there is a need for backwards compatibility to previous software versions.

Measures Code reviews can be used to check the code for compliance with communication standards or other properties that are relevant to make components of the specific project compatible with each other. These properties might e.g. be the appropriate design of interfaces (e.g. using REST) or use of data format standards (e.g. JSON).

Automated or manual tests are also suitable to check for compatibility with other systems e.g. if a web application should be compatible with different browsers.

Portability (Portabilität)

Portability describes the ease of transferring software to different hardware or software platforms.

Application The software will be used on multiple platforms e.g. on different operating systems.

Measures The target platforms should be defined and it should be checked that the software can be executed on them. Furthermore, it might be useful to regularly check for platform specific coding or execute automated or manual tests on all defined target platforms.

Security (Datensicherheit)

Security describes a system's ability to prevent data loss and unauthorized data access.

Application The application executes operations or processes data (e.g. storage of personal data or processing sales) whose manipulation or unauthorized access might cause critical damage.

Measures In general, modern web frameworks already provide support against common attacks. However, the use of one of these frameworks is not enough to ensure security and an additional measure is necessary.

For this, simulated attacks can be executed regularly to test the application. Moreover, there are static code analysis tools that check for common security flaws e.g. related to XSS, CSRF or SQL injection. The results of these tools should be checked for false positives and the code should then be improved if necessary.

As another measure, code reviews with regard to *Secure Software Development* elements (e.g. sanitizing user input) can be conducted.

Maintainability (Wartbarkeit)

Maintainability describes the degree to which software can be easily re-used, modified or extended. Software with a high degree of *Extendibility* can be easily adapted to changing requirements. *Reusability* means that components can be easily integrated into other systems.

Maintainability also includes *Modularity* and *Testability*. Modularity means that the software consists of components that are highly independent i.e. they can be modified without affecting other components. Testability describes how easy it is to define test criteria for the software and perform these tests.

Application The software will be developed further by other software developers or changes to the software are highly likely.

Measures Static code analysis tools can be used to improve code quality or readability with regard to previously defined criteria (e.g. regarding coding style) and thereby ensure maintainability.

Moreover, code reviews can be conducted regularly (e.g. when a feature is completed). For this, criteria (e.g. regarding code documentation, data base requests, the use of design patterns etc.) are defined, specified via checklists and then checked repeatedly.

Modularity can be checked by calculating dependency metrics or analysing dependencies, interfaces and the adherence to relevant design patterns or principles during code reviews.

Reliability (Zuverlässigkeit)

Reliability means the degree to which a system is available, operational and functioning as expected over a certain period of time. This includes *availability*, *fault tolerance* and *recoverability*. Fault tolerance in this instance means that the software operates as expected even under unusual circumstances.

Application It is of high importance for the application to be available over long periods of time or the software is employed in critical systems.

Measures Reliability can e.g. be tested via load tests. Moreover, automated or manual tests can be used to simulate unusual operating circumstances via faulty input or corner cases etc.

6 Evaluation of the project group

Since the assignment of the usual grades (1.0, 1.3, ...) has led to difficulties in the past with regard to the calculation of the overall grade, the evaluation scheme has been adapted, to allow a more transparent grade calculation. We therefore ask you to provide us with a score between 0 – 100 after the last meeting with your group.

Grade	Points
1.0	> 95 – 100
1.3	> 90 – 95
1.7	> 85 – 90
2.0	> 80 – 85
2.3	> 75 – 80
2.7	> 70 – 75
3.0	> 65 – 70
3.3	> 60 – 65
3.7	> 55 – 60
4.0	50 – 55
5.0	< 50

Table 1: Rough conversion table of points to grades.

Below you will find some suggestions for evaluation criteria to make your scoring more fine-grained. Please note that the scoring criteria listed here are to assist you, but you are not bound by them. Overall, you should be able to adequately explain the composition of the grade to the group in case of inspection.

- Software:
 - Does the product meet your expectations?
 - Does it include all the desired functionalities?
 - Are there useful enhancements that were not previously included?
- Project Flow:
 - Overall, are you satisfied with the process and the organization of the team regarding the project?
 - Were there any unexpected delays? If yes, why?
- Communication: Were you satisfied with the communication with the group?
- Documentation: If relevant to your project, is the documentation you have sufficient?

-
- Further use: Do you plan to continue using the software?

References

- [1] ISO. “IEC25010:2011 Systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models”. In: *International Organization for Standardization* 34 (2011), p. 2910.