Hinweise zur finalen Abgabe im Software-Projekt

5. Oktober 2023

Dieses Dokument beschreibt die finalen Abgaben für das Software-Projekt.

2 Finale Abgabe

Alle Dokumente müssen elektronisch bis spätestens 04. Februar 2024, 23:59:59 Uhr (MEZ) abgegeben werden. Die Abgabe erfolgt auf zwei getrennten Wegen:

1. Die **Benutzungsdokumentation** wird digital via MEMS-NG[1] abgegeben. Das Format der digitalen Abgabe muss PDF sein. Es müssen zu jedem Abschnitt die Autor:innen genannt werden.

   Falls MEMS-NG nicht verfügbar ist oder Probleme auftreten, erfolgt die Abgabe der Benutzungsdokumentation als E-Mail an die Tutor:in.

2. Für den **Quelltext** (siehe Abschnitt 2.3) gilt das folgende Prozedere:

   a) **Archiv erstellen:** Alle in Abschnitt 2.3 geforderten Abgaben sind in einem zip-Archiv zusammenzufassen.

   b) **Archiv abgeben:** Das Archiv wird über MEMS-NG[1] abgegeben.

   c) **Problemfall 1:** Sollte die Abgabe scheitern (zum Beispiel weil das Archiv zu groß ist), dann **und nur dann** ist wie folgt zu verfahren:

      i. **Branch erstellen:** Ihr legt in eurem Gitlab-Projekt einen neuen Branch namens *Endabgabe* an. Dort hinein committed ihr alle Bestandteile des eigentlichen Abgabearchivs (wählt dabei eine sinnvolle Commit-Message) und pusht **vor** Ablauf der Abgabefrist.

      ii. **Commit-ID an Tutor:in senden:** Vor Ablauf der Abgabefrist ist der Tutor:in eine E-Mail zu senden, die den Namen der Gruppe und die Commit-ID des in Schritt 2(c)i durchgeführten Commits für die Endabgabe enthält.

      Diese E-Mail ist Teil der Abgabe! Erfolgt sie nicht oder nicht rechtzeitig, so wird dies als *nicht abgegeben* bewertet.

[1] https://exmatrikulator.informatik.uni-bremen.de:8000/mems_ng-0.1/jgradebook/user/submission.xhtml?course=294680&submit=proceed&mode=abgabe

Die Email mit der Commit-ID dient als Nachweis dafür, dass der Quelltext vor dem Abgabetermin committed und somit rechtzeitig fertiggestellt wurde. Es werden nur Abgaben akzeptiert, zu denen rechtzeitig eine Commit-ID gesendet wurde.

d) **Problemfall 2:** Sollte auch dieses scheitern (z. B. weil Gitlab nicht erreichbar ist), dann **und nur dann** ist wie folgt zu verfahren:

   i. **Prüfsumme erstellen:** Von dem in Schritt 2a erstellten Archiv ist eine SHA256-Prüfsumme zu erstellen. Dies kann auf Unix-artigen Systemen mit dem Kommando `sha256sum <dateiname>` und unter Windows 10 in der PowerShell mit `Get-Filehash <dateiname> -Algorithm SHA256` erfolgen. Nach der Erstellung der Prüfsumme darf das Archiv nicht neu erstellt oder verändert werden!

   ii. **Prüfsumme an Tutor:in senden:** Vor Ablauf des Abgabetermins ist der Tutor:in eine E-Mail zu senden, die den Namen der Gruppe und die in Schritt 2(d)i errechnete SHA256-Prüfsumme enthält.

   Diese E-Mail ist Teil der Abgabe! Erfolgt sie nicht oder nicht rechtzeitig, so wird dies als *nicht abgegeben* bewertet.

   iii. **Abgabe des Quelltextes:** Das in Schritt 2a erstellte Archiv muss entweder auf einen Datenträger (USB-Stick) kopiert oder online zum Download bereitgestellt werden.

   Im Falle des Datenträgers ist dieser mit „SWP 2023/24“, dem Namen der Gruppe und dem Namen der Tutor:in zu beschriften und bei der Abschlusspräsentation abzugeben bzw. der Tutor:in anderweitig zugänglich zu machen (z. B. Postfach im MZH). Der Datenträger muss das Archiv enthalten, das zu der zuvor per E-Mail abgegebenen Prüfsumme passt. Ist dies nicht der Fall, wird die Abgabe nicht akzeptiert! Nochmal: Der Datenträger muss das in Schritt 2a erstellte Archiv enthalten. Die Daten dürfen nicht entpackt werden!

   Falls die Datei online zum Download zur Verfügung steht, ist der Tutor:in die entsprechende Download-URL per E-Mail mitzuteilen.

2.1 Benutzungsdokumentation

Die Benutzungsdokumentation besteht aus

- einer Benutzungsdokumentation für die entwickelte Software (alle Dialoge in Form eines Referenzhandbuchs, typische Abläufe in Form eines User-Guides; mit Screenshots) und
- einer Installationsanleitung. Es müssen alle Installationsschritte dokumentiert werden, die zur Installation des Systems nötig sind. Dies schließt auch Dinge wie das ggf. notwendige Einrichten von Datenbanken ein!

Das Format der Dokumente muss PDF sein, die Sprache ist Deutsch. Achtet bei der Benutzungsdokumentation besonders auf gute Lesbarkeit und Fehlerfreiheit. Außerdem sollte Euer Benutzungshandbuch vollständig und benutzer:innenfreundlich sein.

Wenn Ihr völlig hilflos seid, könnt Ihr Euch an der folgenden Struktur orientieren:

1. Einführung
    a) Adressierte Leser
    b) Zweck
    c) Verwandte Dokumente
    d) Konventionen
    e) Information über die Verwendung des Dokuments
    f) Instruktionen für Problemberichte
2. Übersicht
    a) Zugrunde liegende Konzepte, Einbettung
    b) Funktionale Beschreibung
    c) Gefahren und Warnungen
3. Instruktionen (pro Benutzertyp/Persona)
    a) Produktfunktionen (mit Beispielen)
    b) Problembehandlung: Ursachen und Gründe
4. Referenz
    a) Fehlermeldungen und Ursachen
    b) Index der Operationen
5. Anhang: Glossar, Index

Kriterien:

- Sind alle Interaktionen vollständig beschrieben?
- Lassen sich alle Interaktionen mit dem System leicht finden?
- Sind die Interaktionen verständlich beschrieben?
- Ist die Information korrekt (d. h. stimmt mit der Implementierung überein)?
- Ist die äußere Form tadellos?

2.2 White-Box-Tests

Es sind White-Box-Tests als Teil eures Maven-Projekts (unterhalb von `src/test/java`) abzugeben. Die White-Box-Tests müssen dabei eine Anweisungsabdeckung von 100% für drei nicht triviale Methoden (inklusive möglicher Hilfsmethoden) erreichen. Die Anweisungsabdeckung der abgegebenen Tests wird mit JaCoCo gemessen. Wird die Anweisungsabdeckung von 100% nicht erreicht, so werden die White-Box-Tests entsprechend schlechter bewertet.

Die White-Box-Tests dürfen als echte Komponententests unter Einsatz des Frameworks

`Mockito` erstellt werden. Der korrekte Einsatz von Mockito kann hierbei Bonuspunkte für die Bewertung der Tests einbringen. Es dürfen aber auch Integrationstests abgegeben werden.

Es ist nicht nötig, die White-Box-Tests mit Javadoc zu dokumentieren, da sie selbstdokumentierend sein sollten.

Kriterien:

- Wie hoch ist die Anweisungsabdeckung der Tests?
- Wird tatsächlich etwas überprüft (`JUnit Assertions` bzw. `verifiy`)?
- Wird eine 100%-ige Zweigabdeckung durch zusätzliche Tests erreicht?

2.3 Quelltext

Der Quelltext muss in Form eines Maven-Projektes abgegeben werden. In den Quelltexten müssen geeignete Kommentare in Form von Javadoc enthalten sein. Die Kommentare dürfen dabei einheitlich entweder in deutscher oder englischer Sprache verfasst sein. Insbesondere müssen die jeweiligen Autor:innen klar kenntlich gemacht werden (durch JavaDoc-Kommentare mit `@author` in der Dokumentation jeder Klasse). Autor:innen werden zu einer Klasse nur hinzugefügt, wenn sie signifikant inhaltlich beigetragen haben. Formatierungen oder Klammerungen etc. gelten nicht als signifanter inhaltlicher Beitrag. Es ist nicht erforderlich und auch unsinnig, die generierte JavaDoc-HTML-Dokumentation separat abzugeben.

Wir erwarten ein Build-File (Maven), mit dem die Quellen durch ein einzelnes Kommando in ein deploybares Format (WAR-Datei) übersetzt werden können. Sollte dies nicht auf dem Standardweg mittels `mvn clean package` möglich sein, so muss die entsprechende Anweisung in der Installationsanleitung angegeben werden.

Insbesondere müssen den Tutor:innen mit Hilfe der abgegebenen Dateien und der Installationsanleitung die Installationen problemlos möglich sein. Falls eine Initialisierung der Datenbank nötig ist, dann denkt daran, auch Skripte zur Erstellung der benötigten Datenbanken/-tabellen (evtl. auch mit Beispieldaten) abzugeben.

In dem Archiv der Endabgabe sollten sich nur die Dateien befinden, die zum Bauen der Applikation zwingend nötig sind. Insbesondere sollten keine erzeugbaren Binärdateien (wie .class oder die WAR-Datei), das .git-Verzeichnis oder die IntelliJ-Projektdateien enthalten sein.

Kriterien:

- Sind genug Anforderungen (mindestens ausreichend) implementiert?
- Ist der Quelltext frei von offensichtlichen Fehlern? Insbesondere: Ist der Quelltext fehlerfrei Java-konform?
- Ist die Benutzbarkeit über die UI akzeptabel? Enthält die UI Rechtschreibfehler oder ist sie nicht in einheitlicher Sprache (erlaubte Sprachen sind deutsch und englisch – bei Mehrsprachigkeit auch weitere Sprachen)?

- Sind die Build-Files vollständig? Insbesondere: Lässt sich das Projekt auf einem frischen System mit Maven ohne Weiteres fehlerfrei bauen?
- Ist der Quelltext ausreichend, verständlich und einheitlich dokumentiert (Javadoc)?
- Ist der Quelltext wohl strukturiert, frei von Redundanzen und verständlich geschrieben?
- Enthält das Archiv unnötige/unsinnige Dateien (wie .git, .class, .iml, ...)?