Leitlinie für Fortgeschrittenenpraktikum / Projektarbeit

Christoph Bockisch, Gabriele Taentzer (05.04.2018)

1. Ablauf

Für ein Fortgeschrittenenpraktikum (Fopra) bzw. eine Projektarbeit (im Folgenden beides „Projekt“ genannt) sollten Sie zu Beginn mindestens vier Kontakttermine mit Ihrem Betreuer absprechen. Außerdem müssen Sie einen Abschlussbericht erstellen, der aus den unten aufgeführten Bausteinen besteht. Dabei sollten Sie zu Beginn mit Ihrem Betreuer absprechen, welche der Bausteine für Ihr Projekt relevant sind (evtl. kommen auch zusätzliche Anforderungen an die Dokumentation hinzu).

Zu jedem Kontakttermin (außer den ersten beiden) sollten Sie ein Teil-Dokument fertigstellen. Stellen Sie dieses ca. eine Woche vor dem Termin zur Verfügung, um dann Feedback zu bekommen. Zusätzlich zu den unten vorgeschlagenen Terminen für die Dokumente sollten in dem Projekt individuell Meilensteine für die Implementierung abgesprochen werden, die jeweils auch dem Auftraggeber/Betreuer vorgestellt werden. Insbesondere in der Projektarbeit sollte darauf geachtet werden, dass *mindestens ein* solcher Meilenstein noch im ersten Semester liegt, also spätestens circa in Woche 13 des Projekts.

> **Hinweis**: Bei den genannten Projekten mit der Zielsetzung eine Software zu entwickeln ist typischerweise eine agile Vorgehensweise empfehlenswert. In diesem Fall sind häufigere Termine mit dem Auftraggeber/Betreuer notwendig. Außerdem werden die Teil-Dokumente bei einem agilen Vorgehen vorab nicht vollständig erstellt, sondern im Verlauf des Projekts kontinuierlich erweitert und überarbeitet. Reichen Sie daher ggf. zu bereits abgegebenen Teil-Dokumenten aktualisierte Fassungen ein (machen Sie die Änderungen kenntlich). Die finale Version muss zur Abnahme vorliegen.

Die Erstellung der Dokumente und anderer Artefakte (wie Code, Modelle, etc.) dürfen und sollen Sie im Team aufteilen. Es sollten sich jedoch alle Teammitglieder darin auskennen und Fragen zu den Inhalten beantworten können. Jedes Teammitglied sollte zumindest einen Teil eines Dokuments und einen Teil der Implementierung hauptverantwortlich erstellt haben. Beachten Sie auch die unten definierten *Anforderungen an den Code*.

Kontakttermine	Termin FoPra	Temin Projektarbeit
Themenvorstellung	Woche 1 (W1): 1. Vorlesungswoche	Woche 1 (W1): 2. Vorlesungswoche
Vorbesprechung	W2	W2
1. Meilenstein	W4	W6
2. Meilenstein	W8	W13
3. Meilenstein	W14	W26
4. Meilenstein	W19	W37
Abnahme inkl. kommentiertem Code	W20	W38

Zu jedem Kontakttermin sollten Sie eine aktualisierte Zeitplanung erstellen, die jeweils zunehmend präzise wird.

> **Hinweis**: Sie sollten außerdem Ihren Aufwand erfassen, also *wann* Sie *wie viel Zeit* für *welche Aufgabe* aufgewendet haben. Nur indem Sie dn tatsächlichen Aufwand mit dem geschätzten vergleichen, können Sie Ihre Effizienz einschätzen lernen und Ihre Planung verbessern. Geben Sie in Ihrem Zeitplan an, welche Planungsziele Sie bereits erreicht haben und bei welchen Sie Ihre zeitliche Planung eingehalten haben.

Halten Sie regelmäßig Team-Sitzungen ab und fertigen Sie für jedes Treffen ein Protokoll an. Das Protokoll sollte enthalten, welche Mitglieder teilgenommen haben, entschuldigt und unentschuldigt gefehlt haben. Halten Sie weiterhin kurz die Ziele, besprochenen Themen, Ergebnisse und Vereinbarungen fest. Um Team-Sitzungen effizient abzuhalten, ist es hilfreich, die Tagesordnungspunkte vorab zu definieren. Außerdem sollten Sie im Vorhinein bestimmen, wer Protokollant ist und ob diese Rolle über das Projekt hinweg fest besetzt ist oder rotiert (legen Sie dann auch ein Rotations-Schema fest). Nach jeder Sitzung schicken Sie bitte das Protokoll an Ihren Betreuer und stellen es allen Team-Mitgliedern bereit.

2. Anforderungen an Dokumentation

2.1. Aufbau

Im Folgenden werden Inhalte für mögliche Bausteine der Dokumentation im Projekt definiert, gegliedert nach verschiedenen Themengebieten. Dabei sollten Sie alle Themengebiete in Ihrer Projektdokumentation abdecken. Es sind aber nicht alle Bausteine in jedem Projekt passend, z.B. abhängig davon, ob das Projekt eher praktisch oder theoretisch ausgerichtet ist, eher eine Forschungs- oder Entwicklungs-Ausrichtung hat, etc.

> **Hinweis**: Bei einem Projekt, in dem Softwareentwicklung im Vordergrund steht und das in einem Wasserfall-artigen Vorgehen entwickelt wird, sollten folgende Dokumente jeweils spätestens zu dem angegebenen Meilenstein erstellt werden (wobei die Ziffern wie „A1“ oder „A2“ die unten näher beschriebenen Inhalte verweisen):
>
> 1. Planung (zum Meilenstein 1)
>   - (A1) Kurzbeschreibung
>   - (B1) Entwicklungsprozess
>   - (B2) Team
>   - (B3) Risikomanagement
>   - (B4) Zeitplan
> 2. Anforderungsanalyse (zum Meilensteil 1)
>   - (C1) Definition des Zielsystems
>   - (C2) Funktionale Anforderungen
>   - (C3) Nicht-funktionale Anforderungen
> 3. Entwurf (zum Meilenstein 2)
>   - (D1) Grober Entwurf (Architektur)
>   - (D2) Technologien
>   - (D3) Detaillierter Entwurf
> 4. Qualitätssicherung (zum Meilenstein 3)
>   - (E1) Testplan
>   - (E2) Testprotokoll
> 5. Abschlussbericht (zum Meilenstein 4)

- (A2) Zusammenfassung
- (E3) Beispielanwendungen
- (F1) Benutzerdokumentation
- (F2) Entwicklerdokumentation
- (A3) Erfahrungsbericht

Hinweis: In einem Softwareentwicklungsprojekt mit einem agilen Vorgehen lässt sich meist die Architektur nicht vorab entwerfen. Daher sollte hier *(D1) Grober Entwurf (Architektur)* in *5. Abschlussbericht* verschoben werden. Weiterhin empfiehlt es sich insbesondere die Dokumente 2. – 4. parallel zu erstellen und jeweils zu jedem Meilenstein eine Fassung einzureichen, die die bisher abgeschlossenen sowie die unmittelbar folgende Iteration abdeckt.

2.2. Bausteine

Projekt

(A1) Kurzbeschreibung

- Inhaltliche Beschreibung des Projekts
- Rahmenbedingungen
  - evtl. eingebettet in ein größeres Projekt
  - Auftraggeber Uni/Arbeitsgruppe/Unternehmen
  - besondere Vorgaben
  - zeitlicher Rahmen
  - Qualitätsanforderungen, z.B. Skalierung
  - etc.

(A2) Zusammenfassung

- Zusammenfassung der erreichten Ergebnisse
- Mögliche zukünftige Arbeiten

(A3) Erfahrungsbericht

- Entscheidungen, die sich als gut / schlecht herausgestellt haben
- Tipps für zukünftige Entwickler

Vorgehen

(B1) Entwicklungsprozess

- Beschreibung des Software-Entwicklungsprozesses
- Anzuwendende Praktiken (z.B., Pair-Programming, Test First, inkrementell, Anwendungsfälle oder User Stories, etc.)
- Zu erstellende Dokumente
- Rollenvergabe fest oder rotierend
- Eingesetzte Programmiersprache, Modellierungssprachen und andere Formate (sofern diese nicht erst im Entwurf festgelegt werden kann)
- Verwendete CASE-Werkzeuge
  - Integrierte Entwicklungsumgebung
  - Modellierungswerkzeuge
  - Continuous Integration Server
  - Test-Framework
  - Testabdeckung

    - Qualitätssicherungswerkzeuge (z.B. Sammeln von Code Metriken, Überprüfen von Codekonventionen)
    - Projektmanagementwerkzeuge
    - Konfigurationsmanagement
    - Gegebenenfalls weitere

(B2) Team

- Vorstellung der Teammitglieder
- Aufgaben der Teammitglieder
    - Rollen laut Prozessbeschreibung (z.B. Project Owner, Requirements Architect, Quality Assurance)
    - Verantwortlichkeiten für Dokumente, Technologien, Werkzeuge, Protokolle etc.

(B3) Risikomanagement

- Mögliche Risiken, z.B.
    - unbekannte Technologie
    - noch unfertige Spezifikation oder Komponente des Auftraggebers
    - Verfügbarkeit der Teammitglieder (Klausuren, Urlaub, Krankheit?)
    - Heterogene Zusammensetzung des Teams
    - etc.
- Gewichtung der identifizierten Risiken
    - Wie hoch ist die Wahrscheinlichkeit, dass das Risiko eintritt?
    - Wie schlimm wäre die Auswirkung?
    - Welche Risiken müssen vorrangig behandelt werden?
- Strategien pro identifiziertem Risiko
    - Risiko-Vermeidung
    - Risiko-Minimierung
    - Notfallplan

(B4) Zeitplan

- Verfügbarkeitsplanung
- Meilensteine / Releases
- Benötigte Vorgaben (Komponenten, Bibliotheken, Spezifikationen) mit Frist
- Release-Planung
- Erreichte Planungsziele (Welche zeitlichen Ziele wurden erreicht/nicht erreicht?)
- Detaillierte Planung für die kurzfristigen Ziele

Anforderungen

(C1) Definition des Zielsystems

- Vorhandene Hardware, auf der der zu entwickelnde Prototyp laufen muss
- Vorhandene Software (Betriebssystem(e), Bibliothek(en), etc.), mit der der zu entwickelnde Prototyp interagieren muss

(C2) Funktionale Anforderungen

- Überblick, z.B. mittels Use Case Diagramm
- Dokumentation aller funktionaler Anforderungen gemäß des gewählten Prozesses (Use Cases, User Stories, etc.)
- Prioritäten (zumindest Unterteilung in „notwendig“ / „optional“)

(C3) Nicht-funktionale Anforderungen

- Qualität des Systems/Codes (Performanz, Skalierbarkeit, Zuverlässigkeit, Erweiterbarkeit, etc.)
    - Ziele für Code-Metriken
    - Ziele für Code-Konventionen
- Benutzbarkeit (Zielgruppe, Ergonomie, etc.)
- Technische Anforderungen (zu nutzende Technologien, Werkzeuge, Software, etc.)
- Anforderungen an Dokumentation
- Alle nicht-funktionalen Anforderungen müssen möglichst konkret und quantifizierbar (testbar) spezifiziert werden.

Prototyp-Entwurf

(D1) Grober (Software-)Entwurf (Architektur)

- Aufteilung der Realisierung in Komponenten
- Benötigte und bereitgestellte Schnittstellen der Komponenten
- Verwendete architekturelle Entwurfsmuster / Strategien
- Auswahl von Software/Technologien, die benutzt werden sollen
- Konventionen / Coding Style

(D2) Technologien

- Eingesetzte Technologien und Techniken
    - Programmiersprache
    - Bereits vorhandene Softwarekomponenten
    - Bibliotheken (Version, URL, Abhängigkeiten)
    - Frameworks (Version, URL, Abhängigkeiten)
    - Application Container (Version, URL, Abhängigkeiten)
    - Etc.

(D3) Detaillierter (Software-)Entwurf

- Detail-Entwurf der Komponenten in UML-Diagrammen
    - Mindestens ein Klassendiagramm pro Komponente (mit Schwerpunkt auf den wichtigen Klassen wir persistenten Datenstrukturen und Services)
    - Für komplexere Abläufe bzw. komplexeres Verhalten jeweils ein Aktivitäts- oder Zustandsdiagramm
- Entwurfsmuster
- Abhängigkeiten unter den Komponenten und zu externen Komponenten
- Datenmodell
- Verantwortlichkeiten der Entwickler

(D4) Beweisidee

- Voraussetzungen
- Beweisart (direkt oder durch Widerspruch)

(D5) Algorithmententwurf

- Algorithmenmuster (z.B. Greedy, Backtracking)
- Grobverfahren als Pseudocode

(D6) Datenbankentwurf

- Semantisches Datenmodell (Entity-Relationship Model)

- Logisches Datenmodell
    - Tabellen
    - Datenbankdefinition mittels SQL

Validierung

(E1) Testplan

- Testmethoden (GUI-Tests, Unit-Tests, System-Tests, etc.) und wofür sie eingesetzt werden
- Ziele für die Testabdeckung
    - Code-Abdeckung
    - Spezifikationsabdeckung
    - Begründung dafür, dass die angegebenen Ziele ausreichend sind
- Beschreibung der bisher entwickelten Testfälle
    - Bei programmatischen Tests reicht eine sehr kurze Beschreibung bzw. ein „Titel“
    - Bei manuellen Tests (GUI-Tests, Abnahmetests) eine Beschreibung des Ablaufs und erwarteten Ergebnisses
- Charakterisierung der noch zu entwickelnden Testfälle

(E2) Testprotokoll

- Ergebnis der Durchführung der (manuellen und automatisierten) Tests
    - Liste der ausgeführten Tests mit Ergebnis
    - Liste der nicht ausgeführten Tests
- Bericht über die erzielte Testabdeckung

(E3) Beispielanwendungen

- Ein etwas größeres Beispiel, das zeigt, wie das neue Ergebnis (Software, Algorithmus, etc.) eingesetzt werden kann.

Anleitungen

(F1) Benutzerdokumentation

- Benötigte Abhängigkeiten (Bibliotheken, Betriebssystem, etc.)
- Installationsanleitung
- Handbuch

(F2) Entwicklerdokumentation

- Pro architektureller Komponente eine übersichtsartige Beschreibung der Implementierung
- Erweiterungsszenarien
- Zu beachtende Codekonventionen / Architektur
- Bekannte Fehler, fehlende Eigenschaften

3. Anforderungen an Code

- Verwenden Sie den Kommentierungsstandard und die Best Practices der verwendeten Programmiersprache
    - Z.B. JavaDoc, http://www.oracle.com/technetwork/articles/java/index-137868.html)

    - Z.B. Java Code-Konventionen, https://google.github.io/styleguide/javaguide.html
- Code-Dokumentation:
    - Schreiben Sie einen Kommentar für jedes erstellte Modul
    - Schreiben Sie wenigstens einen Kommentar für jede öffentliche Funktion/Methode (triviale, selbsterklärende Funktionen und Methoden, wie getter/setter, dürfen ausgenommen werden)
    - Dokumentieren Sie komplexe Passagen im Code durch inline Kommentare
- Verwenden Sie immer eine sinnvolle und sprechende Benennung für Komponenten, Klassen, Felder, Methoden, Variablen, etc.