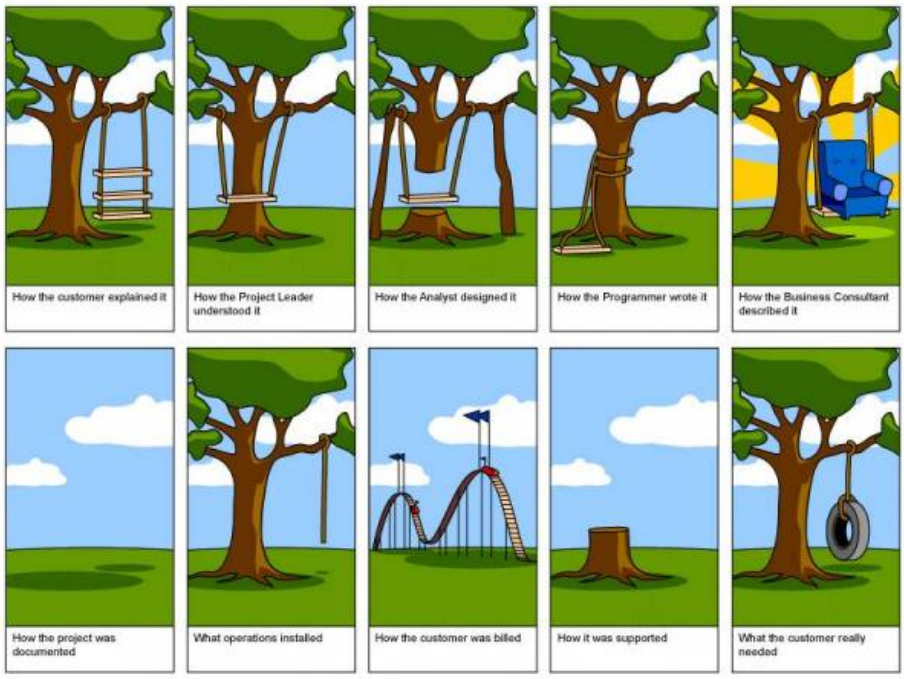
Handbuch Praktikum Software Engineering

Frühjahrssemester 2016



Inhaltsverzeichnis

1 Einführung

Im Praktikum Software Engineering führt jede Gruppe ein kleines Software-Projekt eines externen Auftraggebers vom Anfang bis zum Ende durch. Jede Gruppe ist in der Wahl der Vorgehensweise relativ frei. Es ist jedoch vorgeschrieben, dass die Entwicklung in Anlehnung an das sogenannte "Agile Programming"-Modell **vier Iterationen** durchläuft. Die Entwicklungsziele jeder Iteration müssen vorgängig mit dem Kunden vereinbart werden (siehe auch Abschnitt 2). Zudem muss jede Gruppe eine sogenannte **Versionsverwaltung** verwenden (z.B. Git, Trello oder CVS) und für die wichtigsten Funktionen ihrer Software **automatisierte Tests** schreiben (z.B. sogenannte Unit Tests).

2 Planung der Iterationen (Planning Game)

Zu Beginn jeder Iteration müssen das Entwicklungsteam und der Kunde gemeinsam die verbindlichen Ziele dieser Iteration festlegen. Es wird empfohlen, dies anhand eines sogenannten *Planning Games* zu tun. Der Kunde formuliert dabei seine Wünsche und das Projektteam schätzt den Zeitaufwand um diese Wünsche zu implementieren. Das Planning Game sollte nicht länger als eine Stunde dauern und folgendermassen ablaufen:

1. Der Kunde formuliert alle Funktionen – *Stories* genannt –, die er am Ende der nächsten Iteration implementiert haben möchte. Der Kunde (nur der Kunde!) schreibt jede Story auf eine separate Karte.
2. Die Entwickler schätzen den Arbeitsaufwand für die Implementierung jeder Story in *idealen Personentagen* und notieren den Aufwand auf die jeweilige Karte.
3. Der Kunde sortiert die Karten nach deren Wichtigkeit.
4. Die Entwickler sortieren die Stories absteigend nach Risiko. Der Kunde kann sich anschliessend die Priorisierung noch einmal überlegen. Es kann z.B. sinnvoll sein, einer Story mit hohem Risiko eine höhere Priorität zu geben.
5. Das Entwicklungsteam schätzt die Anzahl idealer Personentage, die bis zum Ende der Iteration für die Implementierung der Stories zur Verfügung stehen.
6. Der Kunde entscheidet – basierend auf diesem "Budget" – welche Stories in der nächsten Iteration implementiert werden.

Wegen der kurzen Dauer des Planning Games ist die **Vorbereitung** wichtig. Dem Kunden sollte der gewünschte Umfang der nächsten Iteration schon vor der Sitzung klar sein und das Entwicklungsteam sollte sich vorgängig mit den Produkteanforderungen vertraut machen und die für die Entwicklung zur Verfügung stehende Zeit bestimmen.

Wichtig: Die Arbeitsplanung – z.B. das Herunterbrechen der Stories in Tasks – gehört *nicht* zum Planning Game. Der Kunde soll mit der internen Organisation des Entwicklungsteams nicht belästigt werden. Siehe auch Abschnitt 2.1.

Siehe auch "Learning the Planning Game" unter:
http://csis.pace.edu/~bergin/xp/planninggame.html

2.1 Detailplanung der Stories

Anschliessend an das Planning Game zerlegt das Entwicklungsteam (in Abwesenheit des Kunden) jede Story in einzelnen **Engineering Tasks**. Diese Tasks basieren auf der Sichtweise der Entwickler. Der Kunde muss damit nichts anfangen können.

Beim Festlegen der Tasks ist das Folgende zu beachten:

- Der Umfang eines Tasks ist möglichst klein zu halten.
- Es kann sinnvoll sein, Tasks Story-übergreifend zu definieren.
- Technische Abhängigkeiten zwischen Tasks sollten besprochen und notiert werden.
- Unit-Tests sind keine eigenständigen Tasks sondern gehören zu deren Umsetzung.

Sobald die Tasks definiert sind, melden sich die Teammitglieder – möglichst freiwillig – als Hauptverantwortliche für einen oder mehrere Tasks. Die Umsetzung eines Tasks muss nicht von dem/der Hauptverantwortlichen allein übernommen werden. Alle Hauptverantwortlichen schätzen anschliessend den Arbeitsaufwand für die Erledigung ihrer Tasks.

Die Detailplanung ist die Grundlage für die Erstellung des Arbeitsplans (siehe Abschnitt 4.3)

3 Teamarbeit

Die Arbeit in einem Team ist eines der Lernziele des PSE. Alle Gruppenmitglieder müssen Verantwortung übernehmen und sich für das Gelingen des Projekts einsetzen.

Jede Gruppe wählt ihre Organisationsform selbst. Z.B. kann das Bilden von Untergruppen sinnvoll sein. Dabei ist entscheidend, Schnittstellen (z.B. zwischen Server und GUI) präzise zu definieren damit die Untergruppen unabhängig arbeiten können. Nicht alle Teammitglieder müssen zwingend programmieren. Auch systematisches Testen und Dokumentieren sind wichtige und anspruchsvolle Aufgaben.

Für kritische und anspruchsvolle Programmteile bietet sich *pair programming* an: zwei Personen arbeiten gleichzeitig an einem Bildschirm; eine schreibt den Programmcode, die andere überprüft ihn fortlaufend.

Wichtig: An mangelhafter Kommunikation scheitern viele Projekte. Die Kommunikation innerhalb des Teams ist ebenso wichtig wie die Kommunikation mit dem Kunden. Allfällige Probleme und Unstimmigkeiten müssen rechtzeitig in der Gruppe oder mit dem Betreuer diskutiert werden.

3.1 Vorgegebene Rollen

Unabhängig von der gewählten Organisationsform müssen die folgenden Rollen in jedem Team besetzt werden:

Key Account Manager: Ist verantwortlich für den Kundenkontakt. Der Kunde soll nur eine Ansprechperson haben.

Chief Deliverable Officer: Ist verantwortlich für die abzugebenden Dokumente (siehe Abschnitt 4). Sorgt dafür, dass Termine und Vereinbarungen eingehalten werden.

Quality Evangelist Ist verantwortlich für die Durchführung der Softwaretests und die Abnahme des Produktes. Erstellt das Testkonzept.

Master Tracker Hat den Überblick über den aktuellen Projektstatus und erstellt die Statusberichte.

Die Besetzung dieser Rollen muss auf ILIAS jederzeit ersichtlich sein.

Für **jede Kundensitzung oder interne Sitzung** werden zudem ein(e) **Sitzungsleiter(in)** und ein(e) **Protokollführer(in)** bestimmt. Die Sitzungsprotokolle haben die Form von Beschlussprotokollen und müssen auf ILIAS zugänglich sein.

4 Abzugebende Dokumente (Deliverables)

Als **Deliverables** bezeichnen wir Erzeugnisse aller Art, die während des Projektverlaufs an vorgegebenen Terminen pünktlich abzugeben bzw. auf ILIAS zu deponieren sind. Es müssen stets alle Versionen der Deliverables zugänglich sein, z.B. auch die erste Version eines verbesserten Dokuments.

Ab der 3. Woche sind folgende Deliverables **wöchentlich auf ILIAS** abzugeben:

- Siztungsprotokolle mit Beschlüssen (auch vom Planning Game)
- Statusberichte (siehe Abschnitt 4.1)
- Risikoanalyse (siehe Abschnitt 4.2)

Die folgenden Deliverables müssen einmalig oder in unregelmässigen Abständen abgegeben werden:

- Arbeitsplan (siehe Abschnitt 4.3)
- Analyse der ersten Iteration (siehe Abschnitt 4.4)
- Testkonzept und Testresultate (siehe Abschnitt 4.5)
- Produkt (siehe Abschnitt 4.6)
- Dokumentation (siehe Abschnitt 4.7)

Es folgen Erklärungen zu den obigen Deliverables.

4.1 Statusberichte (wöchentlich)

Es ist das oberste Ziel jeder Iteration, die mit dem Kunden vereinbarten Stories zu implementieren. Der *Master Tracker* (siehe Abschnitt 3.1) überwacht dabei den Fortschritt der Arbeiten und erstellt jede Woche einen kurzen Statusbericht. Der Statusbericht besteht aus:

1. einen kurzen schriftlichen Bericht (ein paar Sätze)
2. eine Grafik:

Das Projekt ist auf gutem Wege.

Einige Tasks sind im Verzug, das Projekt jedoch nicht gefährdet.

Der Projekterfolg ist gefährdet.
Sofortige Massnahmen erforderlich.

Vorgängig erkundigt er sich bei allen Teammitgliedern nach dem Status ihrer Tasks:

- die bisher für den Task aufgewendete Zeit,
- dem geschätzten Zeitaufwand um den Task abzuschliessen (in der gleichen Zeiteinheit wie bei der Planung).

Der Master Tracker "alarmiert" bei Problemen das ganze Team. Für Stories, deren Fertigstellung in Gefahr ist, ist das ganze Team verantwortlich. Falls eine Story nicht fertig gestellt werden kann, muss der Kunde umgehend informiert werden. **Das Weglassen oder Verändern einer Story liegt nicht in der Kompetenz des Entwicklungsteams.** Der Master Tracker ist zusätzlich "Historiker" des Teams. In den nachfolgenden Planning Games macht er das Team auf vergangene Fehleinschätzungen aufmerksam.

Wichtig: Der Arbeitsplan wird bei Verzögerungen stets aktualisiert.

4.2 Risikoanalyse (wöchentlich)

Selten läuft während eines Projekts alles nach Plan. Um auf unvorhergesehene Ereignisse besser reagieren zu können, werden Projektrisiken laufend eingeschätzt und – mit entsprechenden Gegenmassnahmen – festgehalten. Zusätzlich zu den Statusberichten erstellt jedes Team wöchentlich eine Liste von **mindenstens 1 und maximal 5** Projektrisiken, bestehend aus:

- Eintrittswahrscheinlichkeit (klein, mittel, gross),
- Gewichtung (wie gravierend wirkt sich das Ereignis auf das Projekt aus?),
- Gegenmassnahmen beim Eintreten des Ereignisses.

Zu den möglichen Risiken gehören z.B. das Unterschätzen der Projektkomplexität, das Auftauchen neuer Anforderungen, Wissenslücken (z.B. bei Programmiersprachen), unerwartete Abwesenheiten, Belastungen durch andere Vorlesungen und Tätigkeiten.

4.3 Arbeitsplan

Basierend auf der *Detailplanung der Stories* (siehe Abschnitt 2.1) führt jede Gruppe einen **laufend aktualisierten** Arbeitsplan und macht diesen auf ILIAS verfügbar.

Im Arbeitsplan werden **alle Tasks** mit Verantwortlichkeiten aufgelistet und Abhängigkeiten zwischen Tasks dargestellt. Zu den Tasks gehören nicht nur Programmieraufgaben sondern auch: Planung, Design, Erwerben von Know-How, Testen, Dokumentieren, Präsentationen vorbereiten und halten.

Der Arbeitsplan besteht aus **einer Liste** aller Tasks mit Deadline, möglichen Abhängigkeiten und verantwortlicher Person.

4.4 Analyse der ersten Iteration

Nach Abschluss der **ersten Iteration** analysiert jede Gruppe die eigene Arbeit. Diese Analyse soll bei der Vorlesung präsentiert werden (siehe Zeitplan) und beantwortet die folgenden Fragen:

- War der Inhalt der Stories nach dem Planning Game klar?
- War der Umfang der Stories zu gross/zu klein?
- War die Aufwandschätzung der Stories realistisch?
- Wurde der Aufwand, sich in neue Programmiersprachen/Technologien einzuarbeiten, realistisch eingeschätzt?
- Wurde das Entwicklungstempo realistisch eingeschätzt? Gab es Engpässe?
- Kann die gruppeninterne Kommunikation verbessert werden?
- War die Arbeitsbelastung aller Teammitglieder ähnlich? Sind alle zufrieden?
- Gab es "Leerläufe" oder Wartezeiten aufgrund der Abhängigkeiten zwischen den Tasks?
- Wieviel Zeit hat jedes Teammitglied investiert für
  - Implementation von Stories,
  - Implementation von Testfällen,
  - Testen,
  - Einarbeiten in neue Technologien,
  - Systemadministration?

  Wo ist für die nächste Iteration diesbezüglich der grösste Aufwand zu erwarten?

Es wird empfohlen, eine solche Analyse auch nach Abschluss der zweiten und dritten Iteration zu machen.

4.5 Testkonzept und Testresultate

Systematisches Testen ist unabdingbar um die Qualität von Software zu garantieren. Jede Gruppe erstellt ein Testkonzept (Abgabetermin siehe Zeitplan), das die folgenden Fragen beantwortet:

Unit Tests Werden Unit Tests gemacht? Falls nein: warum nicht? Falls ja: in welchem Umfang, für welche Units?

Datenbank Tests Wie wird die Datenbank getestet? Woher kommen die Testdaten?

Integrationstest Wie werden die Storys getestet? Welches sind die Use Cases? Wie werden spezielle Fälle (z.B. sog. "Race conditions") getestet?

Installationstest Auf welchen Systemen wird die Installation (anhand der Installationsanweisungen) getestet?

GUI Test Wird das GUI automatisiert oder "von Hand" getestet?

Stress-Test Muss die Software hohen Belastungen (z.B. viele und rechenintensive Zugriffe) standhalten? Wie werden solche Szenarien getestet?

Usability-Test Wer sind die künftigen Anwender der Software? Über welche Vorkenntnisse verfügen sie? Wer testet die Software auf ihre Bedienbarkeit? Wie läuft dieser Usability-Test ab?

Nachdem alle im Testkonzept vorgesehenen Tests durchgeführt wurden, sind die **Testresultate** zu dokumentieren und bei Projektende als Teil der Dokumentation abzugeben. Siehe auch Abschnitt 4.7.1.

Hinweis: Die Ergebnisse der Usability-Tests müssen Sie präsentieren (siehe Zeitplan).

4.6 Produkt

Das am Semesterende abzugebende Produkt besteht aus der entwickelten Software und den für Installation, Ausführung, Wartung und Weiterentwicklung notwendigen Informationen. Dies beinhaltet insbesondere den Quellcode, Konfigurationsdateien und die Dokumentation (siehe Abschnitt 4.7).

Zum **Quellcode** gehören folgende (Text-)Dateien und Verzeichnisse:

- `README`: Kurz-Beschreibung der Software, Kontaktadressen und eine Beschreibung der wichtigsten Dateien und Verzeichnisse
- `LICENSE`: Lizenzerklärungen (mit dem Kunden abzusprechen)
- `AUTHORS`: Am Projekt beteiligte Personen und deren Funktionen
- `INSTALL`: Anleitung für Installation und Kompilieren
- `src/`: Quellcode der Software
- `src/tests`: Unterverzeichnis mit allen Unit-Tests
- `Makefile`: Automatisierung des Kompilierprozesses (ant, make, Shellscript,...)
- `bin/`: Kompilierte Software

Softwareparameter sollen unter keinen Umständen im Quellcode "hartcodiert", sondern in **Konfigurationsdateien** ausgelagert werden. Dabei ist speziell auf Abhängigkeiten von absoluten Pfaden, Maschinennamen und Netzwerkadressen zu achten. Alle Parameter müssen in der `INSTALL`-Datei erklärt werden.

4.7 Dokumentation

Für die Fehlersuche oder die spätere Weiterentwicklung und Wiederverwendung von Softwarekomponenten ist eine sorgfältige Dokumentation unerlässlich. Weil in den meisten Softwareprojekten nicht nur der Quellcode sondern auch das Design laufenden Änderungen unterliegen, besteht die Gefahr, dass die Dokumentation vernachlässigt wird und nicht mit dem Stand der Software übereinstimmt. Es ist deshalb sinnvoll, nur soviel Dokumentation wie nötig zu erstellen und den Programmcode möglichst direkt in den Quelldateien – während des Programmierens – zu dokumentieren.

Die Dokumentation besteht aus:

- **Design-Dokumentation:** UML Klassen- und Sequenzdiagramme
- **Quellcode-Dokumentation:** Kurzbeschreibung aller Klassen und Methoden im Quellcode (z.B. Javadoc)
- **Benutzerhandbuch**

4.7.1 Design-Dokumentation

Zur Design-Dokumentation gehört ein oder mehrere UML-Klassendiagramme mit allen relevanten Klassen. Nur die wichtigsten und "öffentlichen" Klassenmethoden sind aufzuführen. Getter- und Setter-Methoden gehören z.B. nicht in ein Klassendiagramm. **Achtung**: Ein Bild allein genügt nicht. Die Beziehungen zwischen den Klassen / Objekten müssen grob erklärt werden. Hinweise auf verwendete Entwurfsmuster ("Design Patterns") sind sinnvoll.

Weiter ist von allen relevanten und nicht-trivialen Abläufen, Anwendungsfällen ("Use Cases") und Algorithmen ein UML-Sequenzdiagramm zu erstellen.

Das Testkonzept und die Testresultate (siehe Abschnitt 4.5) sowie allfällige weitere Dokumente – z.B. Sicherheitskonzepte – gehören ebenfalls zur Design-Dokumentation.

4.7.2 Quellcode-Dokumentation

Die Quellcode-Dokumentation gehört in die Quellcode-Dateien. Alle Klassen, Funktionen und wichtigen Codepassagen sollen verständlich und präzise dokumentiert werden. Nach Möglichkeit soll aus den Kommentaren im Quellcode eine Dokumentation in HTML generiert werden (mit Javadoc oder ähnlichen Systemen).

4.7.3 Benutzerhandbuch

Den genauen Inhalt des Benutzerhandbuchs bestimmt grundsätzlich der Kunde. Das Benutzerhandbuch muss es den künftigen Benutzern ermöglichen die Software zu verwenden. Alle nicht selbsterklärenden Funktionen müssen erläutert werden. Screenshots zur Veranschaulichung sind erwünscht. Für Server-Anwendungen muss das Handbuch einer SystemadministratorIn ermöglichen, die Software zu installieren und in Betrieb zu nehmen.

5 Präsentationen

Während des PSE wird jede Gruppe regelmässig kurze Präsentationen halten (siehe Zeitplan). Jedes Gruppenmitglied übernimmt mindestens eine Präsentation. Befolgen sie dabei einige Ratschläge:

Hilftsmittel. Zu einer guten Präsentation gehört der zweckmässige Einsatz von Hilfsmitteln:

- Folien nicht überfüllen und grosse Schriften wählen. Auf Kontrast achten: gelb auf weiss ist z.B. schlecht lesbar. Probieren Sie Laptop und Beamer vorher aus und bringen Sie die Präsentation auf einem USB-Stick mit am

besten als **PDF-Datei**. Eventuell können Sie Ihr eigenes Laptop für die Präsentation nutzen.
- Achten sie bei Software-Demos auf die Schriftgrössen. GUIs sind unvergrössert auf dem Beamer kaum erkennbar.
- Schreiben Sie an der Tafel gross und leserlich, überlegen sie sich die Darstellung vorgängig.
- Für LaTeX-Präsentationen können Sie die Beamer-Klasse verwenden: https://bitbucket.org/rivanvx/beamer/wiki/Home

Struktur der Präsentation. Präsentationen sollen auf keinen Fall langweilig sein und müssen keinem vorgegebenen Schema folgen. Beachten sie jedoch Folgendes:

- Stellen Sie in den ersten Wochen das Projekt und die Gruppe kurz vor.
- Kündigen Sie den Ablauf der Präsentation an (mündlich, wenn die Präsentation kurz und übersichtlich ist).

Auftreten.

- Gute Vorbereitung mindert die Nervosität.
- Sprechen Sie zum Publikum und schauen Sie nicht immer die gleiche Person an.
- Sprechen Sie laut.
- Überlegen Sie sich bei Software-Demos vorher, welche Funktionen Sie in welcher Reihenfolge zeigen wollen. Klicken und Tippen Sie langsam, so dass das Publikum folgen kann.
- Verzichten Sie auf zuviel Förmlichkeit – Sie sprechen nicht zu einem Parlament sondern zu einer Gruppe von Kollegen. Vermeiden Sie abgedroschene Floskeln (”Ich begrüsse Sie auch von meiner Seite“, ”Ich bedanke mich für Ihre Aufmerksamkeit“).

6 Benotung

Alle Studierenden erhalten am Ende des Semesters eine individuelle Note basierend auf der Qualität des Produkts, der Mitarbeit im Team und den gehaltenen Präsentationen.

7 Infrastruktur

7.1 ILIAS

Jeder Gruppe steht eine **ILIAS-Gruppe** zur Verfügung. Die ILIAS-Gruppe befindet sich in der ILIAS-Webseite des Kurses und hat den gleichen Name mit der Gruppe (PSE1, PSE2 usw.). Die Studierenden können einen Antrag für Beitreten in der Gruppe stellen, indem sie “Beitreten” bei der entsprechenden Gruppe auswählen (s. Abbildung 1). Danach werden die Assistenten den Antrag annehmen. Jede StudentIn soll die entsprechende Gruppe innerhalb der ersten Woche beigetreten haben.

Abbildung 1: ILIAS-Gruppe

7.2 Virtueller Server

Auf Wunsch stellen wir für jede Gruppe einen virtuellen Server zur Verfügung.

8 Code Review

Der Betreuer jeder Gruppe führt am Ende jeder Iteration ein Code Review durch mit dem Ziel, die Qualität des Programmcodes zu verbessern. Das Review basiert unter anderem auf folgender Checkliste:

1. Erfüllen Variablen- / Funktionsnamen die allgemeinen Namenskonventionen und beschreiben sie die Bedeutung der Variablen / Funktionen angemessen?
2. Gibt es Variablen mit verwirrend ähnlichen Namen?
3. Wird jede Variable initialisiert?
4. Können ”globale“ Variablen in lokale umgewandelt werden?
5. Wird jeder Funktionsparameter geprüft bevor er verwendet wird?
6. Hat jede Klasse einen geeigneten Konstruktur?
7. Kann der Zugriff auf Klassenvariablen und -funktionen weiter eingeschränkt werden?
8. Haben Unterklassen gemeinsame Eigenschaften, die man in die Oberklasse auslagern kann?
9. Kann die Vererbungshierarchie vereinfacht werden?
10. Kann während einer Berechnung ein ”overflow“ / ”underflow“ auftreten?
11. Gibt es Vergleichsanweisungen mit ungewollten Nebeneffekten?
12. Werden alle Dateien vor der Verwendung geöffnet?
13. Sind die Parameter (read, write etc.) beim Öffnen von Dateien adäquat?
14. Werden alle Dateien nach deren Verwendung geschlossen?
15. Ist die Fehlerbehandlung konsistent?
16. Sind alle Codeteile ausreichend dokumentiert?
17. Können sehr umfangreiche Methoden in Hilfsmethoden aufgeteilt werden?
18. Kann doppelt vorhandener Code ausgelagert werden?
19. Können Berechnungen verkürzt werden durch Zwischenspeichern von Werten?
20. Wird jeder berechnete Wert auch tatsächlich verwendet?

A Anleitungen und Tools

A.1 LaTeX

LaTeX ist das Standard Textsatz-System für technische und wissenschaftliche Texte und Präsentationen. Sie können LaTeX beispielsweise für das Verfassen der Dokumentation oder für das Erstellen von Präsentationen verwenden. Die folgenden Pakete könnten dafür nützlich sein:

- Das Listings-Paket ermöglicht das Einbinden von Sourcecode mit Syntax Highlighting:
  http://www.ctan.org/tex-archive/macros/latex/contrib/listings
- PGF erlaubt das Erstellen einfacher Grafiken:
  http://www.ctan.org/tex-archive/graphics/pgf/
- Hyperref stellt Textverweise als Hyperlinks dar:
  http://www.ctan.org/tex-archive/macros/latex/contrib/hyperref/
- Mit Beamer können ansprechende Präsentationen erstellt werden:
  https://bitbucket.org/rivanvx/beamer/wiki/Home

Anleitungen und Einführungen zu LaTeX finden Sie im Internet zuhauf, z.B. "The Not So Short Introduction to LaTeX2ε":
http://tobi.oetiker.ch/lshort/lshort.pdf

A.2 UML Tools

UML-Diagramme können z.B. mit folgenden freien Programmen erstellt werden:

- *ArgoUML* unterstützt u.a. Klassendiagramme, Sequenzdiagramme, Activity-Diagramme und kann Java Code generieren und importieren:
  http://argouml.tigris.org/
- *Dia* ist ein Vektor-Grafikprogramm:
  http://live.gnome.org/Dia