Handreichung zum SWT-Praktikum 2019/20

H-G. Gräbe

Version 2.3., 19. September 2019

Inhaltsverzeichnis

1 Vorbemerkungen und Vorgehensrahmen

1.1 Allgemeines

Im Rahmen des Softwaretechnik-Praktikums hat Ihr Team den Auftrag, ein umfangreicheres Software-Projekt von der Anforderungserhebung bis zu einem lauffähigen Prototypen in einem arbeitsteiligen, werkzeuggestützten Prozess selbstständig zu planen und umzusetzen.

Für viele Teilnehmer wird es das erste Mal sein, dass sie vor einer solchen den Berufsalltag eines Informatikers prägenden Herausforderung stehen. Im Zuge der Reorganisation der Ausbildung „Softwaretechnik“ nehmen die aktuell parallel im dritten Semester liegende Vorlesung/Übung sowie das Praktikum keinen Bezug aufeinander. Das Praktikum orientiert sich inhaltlich und methodisch an den Begrifflichkeiten des Buchs

Helmut Balzert. Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering. 3. Auflage, Spektrum Verlag, Heidelberg 2009.

Teile dieses Buches sind im digitalen Semesterapparat verfügbar. Im wöchentlichen Rhythmus werden weiterhin bis Ende November Tutorien angeboten, in denen einzelne für das Praktikum relevante Themen (Einsatz von git, Anforderungsanalyse, Design, Testen, Continuous Integration) vorgestellt und besprochen werden.

Das Praktikumskonzept orientiert sich an einer agilen Vorgehensmethodik. Agile Vorgehensmodelle zeichnen sich durch eine enge Verzahnung und *inkrementelle Weiterentwicklung* von Anforderungsanalyse einerseits sowie Design und Implementierung andererseits aus, wobei die *Anforderungen* sowohl die inhaltliche Seite des Projekts (Produktdimension) als auch die Organisation und Qualitätssicherung (Prozessdimension) berücksichtigen müssen.

Weitere Informationen zum Praktikum sowie Erfahrungsberichte und Übersichten zu Praktika vergangener Jahre finden Sie im OPAL. Dort wird insbesondere die Bedeutung eines arbeitsteiligen Vorgehens, der Kommunikation untereinander sowie des Einsatzes adäquater technischer Hilfsmittel betont, um den Aufwand für Ihr Projekt in einem überschaubaren Rahmen zu halten. Bitte beachten Sie, dass konsolidierte Ergebnisse Ihrer eigenen Arbeit auf dieselbe Weise veröffentlicht werden[1].

Zulassungsvoraussetzung zum Praktikum ist das erfolgreich absolvierte OO-Praktikum (oder eine vom Studienbüro als vergleichbar anerkannte Leistung). Die Voraussetzung wird an Hand einer uns vorliegenden Liste von Matrikelnummern geprüft. Im Einzelfall muss die Zulassungsvoraussetzung durch Vorlage einer entsprechenden Bescheinigung nachgewiesen werden.

Weiter müssen Sie im Almaweb eingeschrieben sein. Bei Problemen im Einzelfall wenden Sie sich bitte an das Studienbüro.

[1] Im Einzelfall können Sie einer solchen Veröffentlichung widersprechen. In diesem Fall wird nur die verbale Beurteilung der Arbeit der Praktikumsgruppe veröffentlicht.

1.2 Phasen des Praktikums

Agile Vorgehensmodelle bestehen aus

- einer *initialen Phase*, in der sich das Team konstituiert, sich einen generellen Überblick über das Zielsystem verschafft und die Grundlagen für die weitere imkrementelle Entwicklung legt,
- mehreren *Zyklen*, in denen die einzelnen Inkremente des Systems entwickelt werden,
- sowie der *Projektabschlussphase*, in der auch benötigte Dokumentationen und Erfahrungsberichte in der geforderten Qualität fertiggestellt bzw. final überarbeitet werden.

Der Ablauf des Praktikums orientiert sich an diesen Standards, wobei wir Ihnen empfehlen, möglichst frühzeitig auch erste Codeentwürfe als „proof of concept“ einzuplanen. Wir unterscheiden vier Phasen des Praktikums

1. Team formieren und Projektressourcen einrichten,
2. konzeptionelle Vorbereitung,
3. organisatorische Vorbereitung und
4. Umsetzung

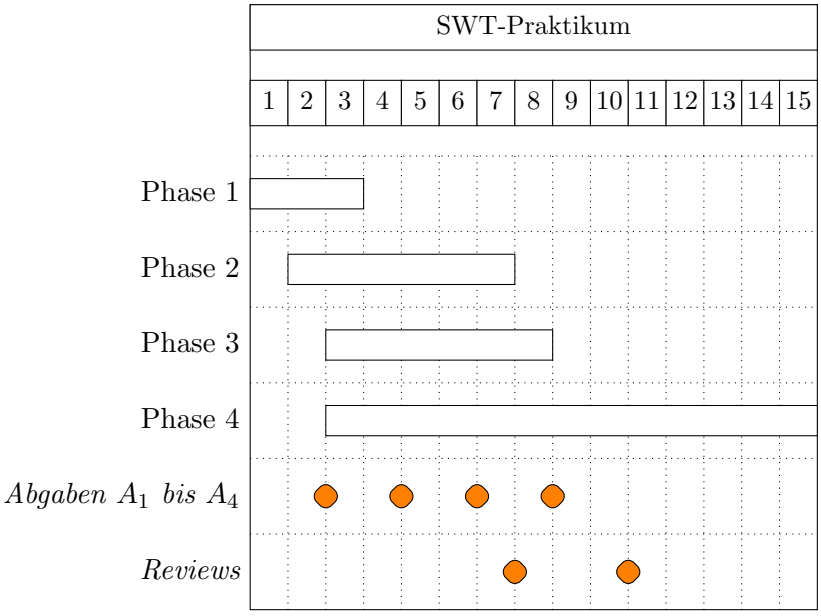
mit verschiedenen Arbeitsschwerpunkten.

Zielstellungen für die einzelnen Phasen

Als Zielstellungen für die einzelnen Phasen ergeben sich folgende Punkte:

- *Phase 1:* Stärken und Schwächen sind analysiert und Rollen verteilt, Projektrisiken in einer *Risikoalayse* zusammengetragen, die Webseiten sind online, die Projektressourcen sind eingerichtet und alle Mitglieder des Teams haben den Zugang zum Team-Repo nachgewiesen.
- *Phase 2:* Das Team hat sich inhaltlich mit dem Projektthema vertraut gemacht, dazu in einem *Recherchebericht* wichtige Konzepte und Aspekte zusammengetragen, mit dem Betreuer diskutiert und auf dieser Basis das *Lastenheft* erstellt.
- *Phase 3:* Das Team hat sich mit den für die Qualitätssicherung und Tests relevanten Konzepten vertraut gemacht und einen *Projektplan*, einen *Releaseplan* sowie einen *Qualitätssicherungsplan* aufgestellt.
- *Phase 4:* Das Projekt wird zügig in fünf Iterationen von einem ersten „Proof of Concept“ (Vorprojekt) zum finalen Release entwickelt, wobei auch die begleitenden konzeptionellen und organisatorischen Dokumente (Projektplan, Releaseplan, QS-Plan, Entwurfsbeschreibung, ggf. Handbuch) angemessen fortgeschrieben werden.

Sie sollten arbeitsteilig vorgehen und beachten, dass es nicht unbedingt sinnvoll ist, die Phasen nach dem Wasserfallmodell zu durchlaufen. Wir orientieren auf einen Verlaufsplan, wie er im folgenden Gantt-Diagramm visualisiert ist (1 = KW 43.2019, 10=KW 2.2020).



Mit dem *Lastenheft* werden funktionale und nichtfunktionale Anforderungen an die von Ihrem Team zu erstellende Softwarelösung detailliert formuliert und in Muss- und Kann-Ziele aufgegliedert. Das Lastenheft ist Gegenstand des *ersten Reviews*, in dem das Team die bisher entwickelten Vorstellungen zur Projektumsetzung präsentiert und zur Diskussion stellt. Bitte beachten Sie die Ausführungen zur Durchführung von Reviews im Abschnitt 2.5.

Im *Projektplan* sind Arbeitspakete inhaltlich zu spezifizieren und deren Aufwand abzuschätzen als Grundgerüst der Umsetzungsplanung. Im weiteren Projektverlauf werden die Arbeitspakete im *Releaseplan* detailliert, zeitlich eingeordnet und in *Issues und Meilensteine* (Issue Tracker) für das jeweilige Release heruntergebrochen.

In der *Umsetzungsphase* sind auf der Basis von Projektplan und Releaseplan für jede Iteration umzusetzende *Issues* festzulegen und einem *Meilenstein* zuzuordnen. Jede Iteration wird mit einem *Releasebündel* abgeschlossen, das die bisherigen Entwicklungen integriert und so den aktuelle Stand der Umsetzung Ihres Projekts dokumentiert. Der Arbeitsstand nach dem ersten Releasebündel ist Gegenstand des *zweiten Reviews*, in dem schwerpunktmäßig die Modellierung auf der Basis der aktuellen *Entwurfsbeschreibung* (siehe Abschnitt 3.6) thematisiert wird. Zum *Projektabschluss* ist ein finales Releasebündel zu erstellen, abzugeben und die Ergebnisse in der *Endabnahme* zu präsentieren.

Wöchentliche Treffen

Während der Vorlesungszeit finden wöchentlich Treffen des Teams mit Betreuer und Tutor statt, auf denen das Team über den Fortgang der Arbeiten berichtet und offene Fragen mit dem Betreuer abstimmt. In der vorlesungsfreien Zeit werden diese Treffen nach Bedarf angesetzt.

Abgabetermine

In den Phasen 1–3 sind zu vier fest vorgegebenen Terminen Dokumente abzugeben und im ersten Review das Lastenheft zur Diskussion zu stellen. Diese Arbeiten sollen bis Ende dezember abgeschlossen sein. Für die in Phase 4 abzugebenden Releasebündel räumen wir Ihnen mit Blick auf die Prüfungszeit sowie die vorlesungsfreie Zeit eine größere zeitliche Variabilität ein.

Als *Abgabetermin* ist jeweils ein Montag 24 Uhr vorgesehen, um etwaige Arbeiten übers Wochenende noch im Team besprechen und konsolidieren zu können.

Da die Umsetzungsphase zu einem großen Teil in die Prüfungszeit fällt, können Sie selbst planen, zu welchen der vorgegebenen Termine (jeweils montags 24 Uhr) die Releasebündel abgegeben werden und wann das zweite Review angesetzt wird. Die entsprechenden Termine sind im Releaseplan zu fixieren.

Sie haben damit die Möglichkeit, Code Sprints oder Hackathons zu organisieren, um das Projekt rasch voranzutreiben, können aber auch längere Pausen vereinbaren. Wir orientieren darauf, das Projekt bis Ende März 2020 zu Ende zu bringen.

1.3 Allgemeine Vorgaben

Textdokumente

Textdokumente müssen in einer Kopfzeile auf jeder Seite das Erstellungsdatum, die Gruppe und den für die Erstellung des jeweiligen Dokuments Verantwortlichen ausweisen. Textdokumente sind im pdf-Format (Schriftgröße 11pt für den fortlaufenden Text, einzeilig) abzugeben.

Abgaben und Abgabetermine

Abgabe bedeutet, dass alle Aufgaben zum jeweiligen Arbeitsauftrag erfüllt wurden und die zu erstellenden Materialien im pdf-Format (Textdokumente) bzw. als zip-Datei (Releasebündel) oder xml-Datei (Aufwandsbericht) *im Verzeichnis Ihrer Gruppe im Upload-Ordner* des OPAL-Portals bereit liegen. Die Abgaben werden nach dem Termin zentral eingesammelt und bewertet. Beachten Sie, dass der Inhalt des Verzeichnisses bei der Abgabe verschoben wird.

Als *Abgabetermin* ist jeweils ein Montag 24 Uhr vorgesehen, um etwaige Arbeiten übers Wochenende noch im Team besprechen und konsolidieren zu können. Die Termine für die Abgaben A1 bis A4 sind vorgegeben. Die terminliche Planung der Abgabe der Releasebündel vereinbaren Sie mit Ihrem Betreuer. Die entsprechenden Termine (jeweils montags 24 Uhr) sind im Releaseplan zu fixieren.

1.4 Fragen und Antworten

Die folgenden Fragen und Bemerkungen ergaben sich aus den Evaluierungen der vergangenen Jahre.

Aufwand. Für das Praktikum muss ein zu hoher Aufwand getrieben werden. Der Aufwand ist ungleich verteilt. Härterer Durchgriff bei „faulen“ Studenten.

Das Praktikum steht mit einem Workload von 150 h im Modulkatalog. Die ausgewerteten Aufwandsanalysen der letzten Jahre zeigen, dass dieser Aufwand im Normalfall auch eingehalten wird. Problematisch wird es natürlich in Gruppen, in denen die Arbeit sehr ungleich verteilt ist, wenn also einzelne versuchen, sich auf Kosten anderer durch das Modul zu „schmuggeln“. Bitte suchen Sie in solchen Fällen rechtzeitig das Gespräch mit dem Betreuer oder auch der Praktikumsleitung, wenn die Steuerungsmöglichkeiten im Team nicht ausreichen, um das Problem in den Griff zu bekommen. Dass der Aufwand in anderen 5-LP-Modulen gefühlt deutlich geringer ausfällt, spricht nicht unbedingt gegen das Praktikum.

„The proof of the pudding is the eating.“ Das Ziel des Praktikums, zu zeigen, dass sich kluge Planung und die Umsetzung software-technischen Know-Hows auszahlt, ist nur zu erreichen, wenn ein Projekt hinreichender Komplexität bis zur prototypischen Implementierung geführt wird, die arbeitsteilig organisiert werden muss.

Aufwandsberichte. Die Aufwandsberichte sind zu strikt, das Format zu formal.

Aufwandsberichte werden in größeren Unternehmen formalisiert erhoben und automatisiert ausgewertet, um verschiedene Projektparameter zu monitoren. Die Aufwandserfassung ist meist Teil einer umfassenderen Projektmanagementumgebung. Auf die Vorgabe einer solchen Umgebung wird im Praktikum verzichtet, um die Aufmerksamkeit auf die fachliche Aufgabenstellung zu konzentrieren. Für die formal korrekte Aufwandserfassung haben die Gruppen 1) Validierungswerkzeuge wie `xmllint` oder 2) eigene Transformatoren eingesetzt, welche gruppenintern verwendete Formate zur Aufwandserfassung in das geforderte Abgabeformat verwandelt haben.

Die Erfahrungen der letzten Jahre zeigen, dass die formalen Vorgaben spätestens mit der dritten Abgabe von der Mehrzahl der Gruppen ohne Probleme umgesetzt werden.

Betreuer und Tutoren. Die Betreuer und Tutoren kennen teilweise die Festlegungen in der „Handreichung“ gar nicht oder halten sich nicht daran.

Wir sind auf die Betreuer und Tutoren angewiesen, die wir gewinnen können. Es gibt anfangs eine Einweisung der Betreuer und Tutoren sowie regelmäßige Treffen des Betreuerteams. Bei Problemen informieren Sie bitte rechtzeitig die Praktikumsleitung, gern auch über das Forum des Kurses im OPAL.

Dokumentation. Viel zu großer Fokus auf bürokratische Schreibarbeit. Weniger Textarbeit. Gefühlt wurde die gleiche Grundaussage in fünf Dokumenten leicht umformuliert wieder verlangt. Dokumentation ist „Wall of Text“.

Dokumentation gehört zu einem Projekt dazu, sowohl der Produkt- als auch der Prozessdimension. „… die gleiche Grundaussage in fünf Dokumenten“ ist auf der Produktebene (Recherchebericht, Lastenheft, Arbeitsplan, Releaseplan) intentional, denn Sie arbeiten ja die ganze Zeit am gleichen Produkt. Die Vorgaben sind

so strukturiert, dass Sie Teile früherer Dokumente in modifizierter Form wiederverwenden können (und sollen).

Protokolle und ein konsistentes Issuemanagement sind wichtig für die Gruppenarbeit, damit jede(r) weiß, wie gerade die Beschlusslage ist.

Handreichung. Die Handreichung ist zu kompliziert und nicht lesbar.

Mit der „Handreichung“ reagierte das Betreuerteam 2016 auf frühere Kritik, dass die Anforderungen auf zu viele Stellen verteilt seien und man damit leicht die Übersicht verliere. Die Handreichung ist klar in vier Teile gegliedert (1) Allgemeines, (2) die Prozessdimension des Praktikums, (3) die Artefaktdimension des Praktikums, (4) Optionale Ansätze und Werkzeuge und wurde 2018 noch um diesen Abschnitt „Fragen und Antworten“ erweitert. Aus unserer Sicht hat sich die Zusammenfassen der Arbeitsblätter zu einer Handreichung bewährt.

Praktikumskonzept. Stärker auf die Möglichkeit von Hackathons und Code Sprints hinweisen. Mehr Zeit für Einarbeitung und Anforderungsanalyse. Früher mit Programmieren beginnen. Extensives Rollenkonzept zurückfahren und Detaillierung in die Verantwortung des Teams geben. Rollenverteilung erst im Laufe des Projekts detaillieren.

Die Wünsche sind vielfältig und zeigen, dass die Freiräume, welche das aktuelle Praktikumskonzept bereits bietet, noch nicht hinreichend genutzt werden. Vorgegeben sind vier Phasen, die auch nicht seriell durchlaufen werden müssen bzw. sollen.

Das Praktikum in seiner gegenwärtigen Stellung im Curriculum kann nur propädeutischen Charakter haben – Einlesen in konkrete Aspekte der Softwaretechnik wie mögliche Vorgehensmodelle und Vorgehensweisen in einzelnen Abschnitten, etwa der Anforderungserhebung, des Entwurfs oder des Testens und Umsetzen dieser Vorschläge im praktischen Projekt. Sie haben dabei viele Freiräume. Die Tutoren und Betreuer steuern gern eigene Erfahrungen als Consultant bei, die Entscheidung über das konkrete Vorgehen liegt aber beim Team. Das Praktikum ist genügend agil angelegt, um eventuelle Fehlentscheidungen später noch zu korrigieren und einen Prototyp mit geringerem Leistungsumfang zu erstellen als ursprünglich geplant.

Konkrete Freiräume auszufüllen verlangt allerdings zwei Dinge: 1) den Freiraum erkennen und 2) sich im Team einig werden.

Themen. Die Themen sind zu schwierig – umfassendes Wissen über Frameworks fehlt bei den meisten, so was kam bis dahin im Studium noch nicht vor. Anforderungen des Praktikums: Programmierkenntnisse, Domänenwissen, Frameworks. Das ist in der Zusammenschau zu viel.

Der Charakter von Softwareprojekten hat sich in den letzten 10 Jahren grundlegend geändert. Während früher Standalone-Projekte die Regel waren, in denen man sich „austoben“ konnte, spielt heute die Wiederverwendung auf verschiedenen Abstraktionsebenen – Einbindung von Bibliotheken, Wiederverwendung von

Codebeispielen, Wiederverwendung von Designprinzipien, Architekturen und Frameworks – eine zentrale Rolle.

Softwareprojekte stehen also heute grundsätzlich vor der Frage, den komplexen Möglichkeitsraum von Realisierungen sowohl hinsichtlich der Angemessenheit des Ansatzes für das konkrete Problem als auch hinsichtlich der Fähigkeiten des Teams einzuschränken und bis zur praktischen Lösung zu konkretisieren.

Die meisten Themen ergeben sich aus Drittmittelprojekten, womit zwei wesentliche Punkte abgesichert werden: 1) die Themen orientieren sich am aktuellen state of the art und 2) Betreuer und Tutor, die meist aus dem Drittmittelkontext kommen, sind mit den wesentlichen Architekturen und Frameworks hinreichend vertraut.

Programmieren. Kann man das Praktikum bestehen ohne eine einzige Zeile Code geschrieben zu haben?

Im Prinzip ja, denn das Ergebnis ist eine Teamleistung. Sie können die Teamarbeit so aufteilen, dass einzelne Studierende nichts programmieren. Primäres Ziel des Praktikums ist es nicht, ein Projekt umzusetzen, sondern am Umsetzen eines Projekts praktische Fertigkeiten im strukturierten Einsatz software-technischer Konzepte und Instrumente zu erwerben.

Wir empfehlen eine solche Aufteilung allerdings **nicht**. Im – nicht unüblichen – Fall eines größeren Gefälles im Bereich der Programmiererfahrungen sollten sie eher rechtzeitig über Maßnahmen im Team nachdenken, um dieses Gefälle abzubauen, siehe dazu Abschnitt 4.4.

2 Vorgaben zu den einzelnen Phasen des Praktikums

2.1 Team bilden und Projektressourcen einrichten

Ziel dieses Praktikumsabschnitts ist es, die Projektrisiken sowie die Stärken und Schwächen des Teams zu analysieren, die Arbeitsfähigkeit des Teams herzustellen, Rollen zu definieren und zu verteilen, die Projektressourcen zu aktivieren und die Webseiten des Projekts einzurichten.

Die Stärken und Schwächen jedes Teams ergeben sich aus den Stärken und Schwächen der einzelnen Teilnehmer. In der gemeinsamen Arbeit ist es wichtig, die individuellen Stärken im Team zur Geltung zu bringen und die individuellen Schwächen zu kompensieren.

> Legen Sie im git-Repo ein Verzeichnis `Protokolle` an, in dem später die Protokolle aller Gruppentreffen eingestellt werden.
>
> Tragen Sie in diesem Verzeichnis im git-Repo die Stärken und Schwächen der einzelnen Mitglieder in einer Datei `StaerkenUndSchwaechen.md` im Markdown-Format zusammen. Jedes Teammitglied hat dabei wenigstens einen Commit selbst erstellt und gepusht.
>
> Erstellen Sie eine *Risikoanalyse*, in welcher Sie die zehn wichtigsten Projektrisiken zusammentragen und für jedes dieser Risiken erläutern, mit welchen präventiven Maßnahmen Sie gegensteuern wollen.

Besetzen Sie auf dieser Basis die Rollen **Projektleiter** und **Stellvertreter**. Besprechen Sie, welche weiteren Rollen im Team besetzt werden sollen.

Die **Besetzung der Rolle des Projektleiters** mit einer Person, welche die Fäden straff in der Hand hält, ist für den Erfolg der Projektarbeit besonders entscheidend. Es geht dabei eher um organisatorisches Geschick und Durchsetzungsfähigkeit als um detaillierte fachliche Kenntnisse. Sie sollten deshalb diese Rolle mit besonderer Sorgfalt und (möglichst) für die gesamte Dauer des Praktikums vergeben.

Der Projektleiter ist erster Ansprechpartner für Betreuer und Tutor.

Vorgehen und Abgabe A1

Legen Sie für die Arbeiten zu diesem Punkt (Projektressourcen einrichten, Webseiten aufsetzen, Rollen besetzen usw.) Issues an und fassen Sie diese zu einem Meilenstein „Projektumgebung aufgesetzt“ zusammen.

Zum Abgabetermin A1 ist die **Risikoanalyse** (2 bis 3 Seiten im pdf-Format) sowie der **Aufwandsbericht** Ihres Teams abzugeben.

Zum Abgabetermin A1 prüft der Tutor weiterhin, ob die Issues angemessen und vollständig sind, alle Arbeitsaufträge abgearbeitet wurden und die Materialien wie gefordert im Team-Repo sowie auf den Webseiten vorhanden sind.

2.2 Konzeptionelle Vorbereitungen

Recherche zum Projektthema

Die genauere Erschließung des Themas erfolgt durch eine Recherche zum Projektthema, mit der sich das Team mit dem Umfeld der Themenstellung genauer vertraut macht, um auf dieser Basis die Erstellung des *Lastenhefts* vorzubereiten. Eine solche Klärung ist auch für die gruppeninterne Kommunikation wichtig, denn so wird deutlich, wie weit im Team einheitliche Vorstellungen über die mit der Aufgabenstellung verbundenen Begrifflichkeiten sowie ein detailliertes Bild der Anforderungen entwickelt werden konnten.

Wichtige Grundlage für die Verhandlungen mit dem Auftraggeber (Betreuer) ist die angemessene Durchdringung der Thematik Ihres Projekts und die Identifizierung relevanter Begriffe und Konzepte der Anwendungsdomäne sowie der verschiedenen Aspekte Ihres Themas. Das Ergebnis der thematischen Recherche ist in einem **Recherchebericht** zu dokumentieren als Grundlage für ein ausführliches **Gespräch mit dem Auftraggeber** zur Klärung offener Fragen und Aspekte des zu bearbeitenden Themas und während des wöchentlichen Treffens mit dem Betreuer abzustimmen.

Der Recherchebericht soll unter deutlicher Bezugnahme auf Ihr Thema einerseits die Ergebnisse Ihrer konzeptionellen Analyse wiedergeben und andererseits wichtige themenrelevante Begriffe und Zusammenhänge erläutern.

Teilen Sie den Recherchebericht in drei Kapitel *Begriffe* (als Grundlage für das Glossar), *Konzepte* (domänenspezifisches Was und Wie – Rahmenbedingungen) und *Aspekte* (projektspezifische Momente – Gestaltungsspielräume).

Vorgehen und Abgabe A2

Legen Sie für die Arbeiten zu diesem Punkt Issues an und fassen Sie diese zu einem Meilenstein „Recherchebericht“ zusammen. Zum Abgabetermin A2 prüft der Tutor, ob die Issues vollständig sind, alle Arbeitsaufträge abgearbeitet wurden und die Materialien wie gefordert im Team-Repo sowie auf den Webseiten vorhanden sind.

Weiterhin sind zum Abgabetermin A2 der **Recherchebericht** (5 bis 8 Seiten im pdf-Format) sowie der aktuelle **Aufwandsbericht** Ihres Teams abzugeben.

Bitte beachten Sie die Formatvorgaben für die Abgabe von Textdokumenten sowie die Hinweise zum Thema „Abgaben“ im Abschnitt 1.3 „Allgemeine Vorgaben“.

Lastenheft

In praktischen Softwareprojekten werden die Anforderungen an die zu erstellende Software-Lösung sowie die Qualitätsparameter und Abnahmekriterien oft in *Lastenheft* und *Pflichtenheft* fixiert, wobei der Auftraggeber im Lastenheft seine grundlegenden Anforderungen an die zu erstellende Softwarelösung zusammenfasst (was soll gebaut werden?), auf dessen Basis im Pflichtenheft genauere Parameter der Lösung vereinbart werden (wie soll die Lösung gebaut werden?). Oft arbeitet der Auftraggeber bei der Erstellung des Lastenhefts mit IT-Beratern zusammen, um die eigenen Anforderungen mit der nötigen Genauigkeit und Klarheit zu fixieren.

Im Praktikum spielt das *Lastenheft* als Anforderungsdokument die zentrale Rolle, wobei Ihr Team den Auftraggeber bei der Erstellung des Lastenhefts unterstützt, indem Ihr Team eine Version des Lastenhefts mit allen umzusetzenden funktionalen und nichtfunktionalen Anforderungen sowie der Fixierung von Qualitätsparametern erstellt, dieses im ersten Review präsentiert und auf dieser Basis eine finale Version mit dem Betreuer abstimmt und vereinbart. Die Vorgaben zum Lastenheft sind im Abschnitt 3.3 genauer ausgeführt. Das Lastenheft soll Muss- und Kann-Ziele ausweisen und ist die Basis für die Erstellung von *Projektplan* (thematische Gliederung) und *Releaseplan* (zeitliche Gliederung) als Mittel der organisatorischen Steuerung der Projektarbeit.

Im Lastenheft ist insbesondere ein Teilziel zu fixieren, das als *Vorprojekt* bereits mit dem ersten Releasebündel fertiggestellt werden soll.

Vorgehen und Abgabe A3

Legen Sie für die Arbeiten zu diesem Punkt Issues an und fassen diese zu einem Meilenstein zusammen.

Zum Abgabetermin A3 sind das **Lastenheft** (6 bis 8 Seiten im pdf-Format) sowie der aktuelle **Aufwandsbericht** abzugeben.

Zu diesem Abgabetermin werden der Aufwandsbericht, der Stand der Webseiten, die Führung der Protokolle sowie die Führung der Issues durch das Team bewertet. Die Bewertung des Lastenhefts geht in die Gesamtbewertung des ersten Reviews ein.

Erstes Review

Im ersten Review wird die vorgelegte Version des Lastenhefts begutachtet, mit den Vorstellungen des Auftraggebers abgeglichen und Vereinbarungen für die Erstellung der finalen Version des Lastenhefts getroffen, die für das weitere Vorgehen verbindlich ist. Dazu ist vom Team eine Präsentation vorzubereiten.

Die Review-Sitzungen finden nach dem im Web veröffentlichten Plan statt. Damit sich alle Seiten auf dieses Treffen gut vorbereiten können, ist die **rechtzeitige Vorlage** des Lastenhefts Zulassungsvoraussetzung. Beachten Sie die *Hinweise zum Ablauf eines Reviews* im Abschnitt 2.5.

2.3 Organisatorische Vorbereitungen

Projektplan, Releaseplan und Qualitätssicherungsplan

Mit der Erstellung von Projektplan, Releaseplan und Qualitätssicherungsplan werden die organisatorischen Vorbereitungen der Umsetzungsphase abgeschlossen. Die Vorgaben zum Projektplan, Releaseplan und Qualitätssicherung sind in den Abschnitten 3.4 und 3.5 genauer ausgeführt und im Abschnitt 2.4 in die Methodik der Umsetzungsphase eingeordnet.

Vorgehen und Abgabe A4

Legen Sie für die Arbeiten zu diesem Punkt Issues an und fassen diese zu einem Meilenstein zusammen.

Zum Abgabetermin A4 sind der **Projektplan** (1 bis 3 Seiten im pdf-Format), der **Releaseplan** (1 bis 3 Seiten im pdf-Format), das **Qualitätssicherungskonzept** (4 bis 6 Seiten im pdf-Format) sowie ein aktueller **Aufwandsbericht** abzugeben.

2.4 Umsetzungsphase

Parallel zur Erstellung des Lastenhefts sowie der Pläne zum organisatorischen Ablauf, spätestens aber nach deren Verabschiedung, beginnt die Umsetzung des Projekts. Hierzu sind in regelmäßigen Abständen *Releasebündel* bereitzustellen, in denen der Projektfortschritt sichtbar wird.

Nach Abgabe des ersten Releasebündels ist eine *weitere Präsentation* des Arbeitsstands durch das Team vorgesehen, die mit Betreuer und Tutor besprochen wird. Im Mittelpunkt dieses **zweiten Reviews** steht der Stand der Modellierung an Hand der *Entwurfsbeschreibung* sowie die Umsetzung des Vorprojekts.

Methodik

Das Projekt ist in fünf Iterationsphasen umzusetzen, in denen die Funktionalität Schritt für Schritt erweitert und zu deren Ende der aktuelle Arbeitsstand jeweils in einem **Releasebündel** integriert und dokumentiert wird. Damit soll gewährleistet werden, dass zum Abschluss jeder Iteration die Umsetzung der funktionalen und nichtfunktionalen Anforderungen, Dokumentation und Tests zueinander passen, um die Vollständigkeit und praktische Realisierbarkeit der entwickelten Konzepte zu sichern.

Ein solches Vorgehen erleichtert es, Entwurfsentscheidungen schon in einer frühen Phase auf praktische Realisierbarkeit hin zu überprüfen und gegebenenfalls in einer weiteren Iteration mit nicht zu hohem Aufwand zu korrigieren.

In den Iterationsphasen müssen die personellen Projektressourcen klug eingesetzt werden. Es ist unbedingt zu vermeiden, dass nur einzelne Personen zum Projekterfolg beitragen. Das erfordert vorausschauende Planung, Herunterbrechen von größeren, grob granularen Epics und Stories auf handhabbare Issues und deren parallele Umsetzung.

Ein *Issue* ist so zu dimensionieren, dass es vom Assignee in der vorgegebenen Zeit realisiert werden kann. Der Assignee ist neben dem zu leistenden Beitrag zum Quellcode des Systems auch für die Zuarbeiten verantwortlich, die zur Ergänzung des Entwurfs, der Dokumentation sowie der Suite der Komponententests erforderlich sind. Zum Ende jeder Iteration wird Bilanz gezogen, Code, Entwurf, Dokumentation und Testsuite zu einem neuen Release integriert und der Releaseplan fortgeschrieben.

Releaseplan

Um die strategische Dimension der zeitlichen Abläufe der Projektrealisierung im Auge zu behalten, ist auf der Basis der Arbeitspakete aus dem Projektplan und der dort enthalte-

nen Aufwandsschätzungen (inhaltliche Dimension) ein *Releaseplan* (zeitliche Dimension) zu erstellen und fortzuschreiben, in dem ausgewiesen ist, welche Funktionalität in welchen Iterationen umgesetzt werden soll. Der Releaseplan detailliert damit die Arbeitspakete und ordnet die Aufgaben einzelnen Iterationen zu. Der Releaseplan ist die *Roadmap* des Projekts und zeigt, wie Sie die gestellte Aufgabe mit den verfügbaren Ressourcen in der verfügbaren Zeit erfüllen wollen.

Die *Termine für die Abgabe der fünf Releasebündel* sind entsprechend den Vorgaben im Abschnitt 3.4 mit dem Betreuer zu vereinbaren und im Releaseplan zu fixieren.

Erstellen Sie eine erste Version des Releaseplans als grobgranulare Epics auf der Basis der Arbeitspakete aus dem Projektplan und schreiben Sie diese Planung durch Verfeinerung, Ergänzung und Rescheduling regelmäßig fort.

Die *wöchentlichen Treffen* des Teams mit dem Betreuer und/oder Tutor dienen auch der inhaltlichen Strukturierung von Epics, Stories, dem Herunterbrechen auf Issues, der Zuordnung von Issues zum aktuellen Meilenstein, der Zuweisung von Issues zu Assignees sowie der *inhaltlichen* Fortschrittskontrolle.

Vorprojekt

Zur Darstellung der Leistungsfähigkeit Ihres Teams ist mit dem ersten Releasebündel eine erste Softwareskizze als *Proof of Concept* (Vorprojekt) fertigzustellen, die zeigt, dass Sie mit den wichtigsten Architekturprinzipien Ihres Themas vertraut und bei der Umsetzung des Projekts auf dem richtigen Weg sind. Das Thema des Vorprojekts soll direkten Bezug zum Hauptprojekt haben und ist mit dem Betreuer während des ersten Reviews abzustimmen.

Zweites Review

Die Bewertung der Modellierungsstands auf der Basis der aktuellen *Entwurfsbeschreibung* sowie die Darstellung des Arbeitsstands erfolgt im Rahmen des **zweiten Reviews**. Die relevanten Dokumente werden wieder mit Betreuer, Tutor und dem Team durchgesprochen. Das Team bereitet dazu eine Präsentation vor. Damit sich alle Seiten auf diese Treffen gut vorbereiten können, ist die **rechtzeitige Vorlage** der Unterlagen Zulassungsvoraussetzung. Die Review-Sitzungen finden nach dem im Web veröffentlichten Plan statt.

Vorgehen, Abgabe und Bewertung

Jedes Release schließt eine weitere Iteration der Umsetzungsphase ab. Eine solche *Iteration* beginnt mit der Identifizierung und ggf. Verfeinerung von *Issues* entsprechend der *Releaseplanung*, deren Zuordnung zu einem (neuen) *Meilenstein* im git-Repo des Teams und der Zuordnung der einzelnen Issues zur Bearbeitung.

Releasebündel sind zu den im Releaseplan festgelegten Terminen entsprechend den Vorgaben im Abschnitt 3.7 zusammenzupacken und abzugeben. Jedes Releasebündel schließt eine Iterationsetappe im Projekt ab – die Issues des zugehörigen Meilensteins sollten abgearbeitet oder für das Rescheduling vorgemerkt sein. Zugleich ist die Demoversion auf der Projekt-Webseite zu aktualisieren, so dass deren Funktionalität vom Tutor bzw. Betreuer ohne weitere Probleme getestet werden kann.

Abzugeben bzw. bereitzustellen: Eine zip-Datei des aktuellen Release sowie eine lauffähige neue Demoversion. Vorab ist der *Releaseplan* zu aktualisieren wie im Abschnitt 3.4 erläutert.

Bewertung: Bewertet wird für jedes Release die organisatorische Umsetzung (Pünktlichkeit der Einreichung, Konsistenz der Unterlagen, Aufwandsbericht, Funktionalität der Demoversion – zusammen max. 7 Punkte), die Fortschreibung des Releaseplans sowie das Issue Management (max. 2 Punkte) und die Aktualität der Webseiten des Projekts (max. 1 Punkt).

2.5 Zur Durchführung von Reviews und Endabnahme

Ein Review ist eine manuelle Qualitätssicherungsmethode, in der Arbeitsprodukte begutachtet werden, die in einem fortgeschrittenen, aber noch nicht finalen Stadium vorliegen. Zentrales Ziel eines Reviews ist die Verbesserung der Qualität der untersuchten Arbeitsprodukte.

Mit dem Review sollen deshalb Stärken und Schwächen der vorgelegten Arbeitsprodukte identifiziert werden. Dabei steht der Qualitätsverbesserungsaspekt durch Hinzuziehen externen Sachverstands im Mittelpunkt. Das erfordert natürlich, dass die vorgelegten Arbeitsprodukte bereits einen ausgereiften Stand erreicht haben und interne Diskussionsprozesse weitgehend abgeschlossen sind.

Ein Review gliedert sich in die Phasen Vorbereitung, Review-Sitzung und Nachbereitung. Die wichtigsten Rollen im Review sind die des Moderators (Betreuer), der Autoren (hier: ein Mitglied des Teams als „Wortführer“), der Gutachter (Betreuer und Tutor) und des Protokollführers (Tutor). Die Gutachter arbeiten die vorgelegten Unterlagen vorab durch und notieren entsprechende Fragen. Es geht dabei und auch in der Review-Sitzung um die Identifizierung von Defekten, nicht um die Diskussion, wie diese Defekte beseitigt werden können.

Die im Review zu begutachtenden Arbeitsprodukte müssen allen Beteiligten rechtzeitig vorliegen. Die Beteiligten bereiten sich auf die Review-Sitzung individuell vor und arbeiten dazu die vorgelegten Unterlagen durch. Zur Review-Sitzung stellt das Team in einer Präsentation die Schwerpunkte der vorgelegten Arbeitsprodukte noch einmal vor. Es ist sinnvoll, die Präsentation in einem gruppeninternen Meeting vorzubereiten und dabei genaue Absprachen zu treffen. Zu diesem Meeting können Sie Ihren Tutor hinzuziehen oder sich mit ihm im Vorfeld verständigen.

Für die Review-Sitzung ist eine Zeitspanne von 45 Minuten vorgesehen. Davon stehen 20 Minuten für die Präsentation zur Verfügung, 20 Minuten für die Diskussion und 5 Minuten für die Aus- und Bewertung. Die Review-Sitzung wird vom Betreuer geleitet, der streng auf die Einhaltung des Zeitplans achtet. An der Review-Sitzung sollen alle Teammitglieder teilnehmen. Der Tutor führt ein (inhaltliches) Protokoll. Im Ergebnis werden Hinweise gegeben, welche in der Regel eine Nachbearbeitung der vorgelegten Arbeitsprodukte erforderlich machen. Diese Nacharbeit ist bis zu einem festzusetzenden Termin auszuführen und beim Tutor abzurechnen.

Als Teil des SWT-Praktikums wird ein Review ebenfalls bewertet, wobei sich die Bewertung aus der Qualität der Dokumente (80 %), der Präsentation selbst (20 %) und der Qualität der evtl. erforderlichen Nacharbeiten zusammensetzt. Zum Ende der Review-Sitzung gibt der Betreuer eine Einschätzung sowie eine Bewertung in Form einer Punktspanne (z.B. 5. . . 7). Der genaue Punktwert ergibt sich aus der Qualität der Nacharbeiten, die vom Tutor durchgesehen werden.

In Einzelfällen kann bei schwerwiegenden Bedenken eine Wiederholung des Reviews angesetzt werden.

Endabnahme

Bis zur Deadline für die Arbeit an Ihrem Softwareprojekt soll das Release Ihres Teams so weit entwickelt sein, dass alle vorgesehenen Funktionalitäten implementiert sind, sich Entwurfsbeschreibung und Testmaterial auf dem aktuellen Stand befinden und auch die Anwender- und Installationsdokumentation, so weit erforderlich, fertiggestellt ist. Dazu empfiehlt es sich, die letzte Iteration als *Projektabschlussphase* komplett für Bug fixing, Tests und Konsolidierung der Dokumentation vorzusehen.

Die relevanten Materialien sind als finales Release in Form eines weiteren *Releasebündels* abzugeben und bilden die Grundlage für die Endabnahme. Die Endabnahme läuft ähnlich wie ein Review ab, nimmt aber etwa 90 Minuten pro Gruppe in Anspruch, davon 40 Minuten Präsentation der inhaltlichen Ergebnisse sowie der Erfahrungen in der Organisation Ihrer Projektarbeit.

3 Beschreibungen der einzelnen Artefakte

3.1 Projektressourcen

ifi-gitlab

Für jedes Team ist im ifi-gitlab `https://git.informatik.uni-leipzig.de` ein git-Repo als Klon unseres Basis-Repos eingerichtet, das der Tutor an das Team durch Eintragen weiterer Mitglieder an das Team übergibt. Dazu müssen Sie Ihren gitlab-Account aktiviert haben, d.h. sich einmal am ifi-gitlab mit Ihren studserv-Daten angemeldet haben.

Das gitlab-System bietet neben der eigentlichen Verwaltung der Quellen über `git` die Möglichkeit, *Issues* anzulegen und zu *Meilensteinen* zusammenzufassen sowie *Prozesse des Continuous Integration* einzurichten. Von diesen Möglichkeiten soll im Praktikum Gebrauch gemacht werden.

Praktikumsserver

Auf dem Praktikumsserver `pcai042.informatik.uni-leipzig.de` steht für Ihr Team ein eigener **Projekt-Account** zur Verfügung (mit Datenbank und Port-Range), über den Sie die Fortschritte Ihrer Arbeiten auf den Webseiten des Teams dokumentieren und später die Funktionsfähigkeit Ihres Prototyps demonstrieren.

Die Einrichtung und Verwaltung dieses Accounts liegt in den Händen des technischen Assistenten Ihres Teams. Die Zugangsinformationen erfragt der technische Assistent beim Tutor.

Webseiten des Projekts

Auf den Webseiten des Projekts

`http://pcai042.informatik.uni-leipzig.de/~<gruppenname>`

ist der Fortgang der Arbeit Ihres Teams darzustellen. Auf der Webseite des Teams sind (wenigstens) folgende Materialien zu veröffentlichen:

- die Teamzusammensetzung und Rollenverteilung,
- allgemeine Informationen über das von Ihnen bearbeitete Projekt,
- die erstellten Dokument-Artefakte als pdf sowie die Releasebündel als zip-Dateien sowie
- zu gegebener Zeit der Link zu einer Demo der Applikation.

Protokolle der Arbeitstreffen

Über die wöchentlichen Treffen mit dem Betreuer sowie weitere Arbeitstreffen ist Protokoll zu führen. Die Protokolle sind in einem eigenen Verzeichnis `Protokolle` im git-Repo zusammenzutragen. Damit stehen sie allen Teammitgliedern jederzeit zur Verfügung.

3.2 Aufwandserfassung

Für die qualifizierte Führung eines Projekts spielt auch die Aufwandsschätzung, -analyse und -erfassung eine wichtige Rolle, um die personellen Projektressourcen effizient und den jeweiligen Fähigkeiten angemessen einzusetzen. Dazu orientieren wir auf folgendes Vorgehen:

- Die auszuführenden Arbeiten werden in Teilaufgaben (Issues) zerlegt und zugeordnet (V: Projektleiter).
- Jedes Teammitglied führt selbst Buch über die aufgewendete Zeit und bewertet den Aufwand (A) sowie die Schwierigkeit (S) der Issues auf einer Skala $1 \ldots 5$ (1 = viel zu niedrig, 2 = zu niedrig, 3 = angemessen, 4 = zu hoch, 5 = viel zu hoch). Dazu gehört natürlich auch die Zeit für Projekttreffen und ähnliche Abstimmungs- und Diskussionsrunden.
- Der Projektleiter sammelt diese Informationen regelmäßig ein, prüft sie im Team auf Plausibilität und erstellt den Aufwandsbericht, aus dem ersichtlich wird, welche Teammitglieder für welche Issues wie viel Zeit verwendet haben und wie die Arbeit eingeschätzt wurde.
- Die Analysen der abzugebenden Aufwandsberichte **sind so zu erstellen, dass sich die Berichtszeiträume *nicht* überschneiden**. Für Issues, die mehreren Personen zugeordnet sind (etwa Gruppentreffen), sind Zeitstunden aufzuschreiben (und keine Personenstunden).
- Der Bericht muss in einem standardisierten XML-Format erstellt sein, welches durch das XSchema `Aufwand.xsd` beschrieben ist. Die eingereichten Dokumente müssen valide bzgl. der dort definierten Syntax sein.
- Die Berichte sind im Upload-Verzeichnis Ihrer Gruppe als Datei `Aufwand.xml` abzulegen oder diese Datei in das Releasebündel zu integrieren.

Beispiel für einen Aufwandsbericht:

```
<Analyse von="2016-12-10" bis="2016-12-16" gruppe="uhu17" createdBy="CK"
      createdAt="2016-12-16">
  <done who="alle" A="3" S="2" Zeit="3"> Aufgabenverteilung</done>
  <done who="DB,JM,SS,CK" A="3" S="3" Zeit="1">
     Präsentation Vorprojekt
  </done>
  <done who="SS" A="4" S="4" Zeit="6">
     Ergaenzungen Entwurfsbeschreibung Vorprojekt
  </done>
  ...
  <done who="GK" A="3" S="3" Zeit="2">
     Weitere Goldstandard-Testsätze gesucht
  </done>
</Analyse>
```

Im Ordner des OPAL-Kurses finden Sie die Datei `Aufwand.xsd` sowie als Beispiel die Datei `Aufwand-Beispiel.xml`.

3.3 Lastenheft

Das Lastenheft ist das zentrale Anforderungsdokument, an dem sich die weitere Umsetzung des Projekts orientiert.

Mit dem Lastenheft soll deutlich werden,

- in welchem Kontext Ihre Softwarelösung entwickelt wird,
- welche funktionalen und nichtfunktionalen Anforderungen Ihre Softwarelösung erfüllen soll,
- welche der acht Qualitätsparameter der Norm ISO/IEC 25010[2] in welcher Ausprägung für das Projekt relevant sind,
- welche Aufgabe als *Vorprojekt* bereits mit dem ersten Release umgesetzt sein soll und
- was die *finalen Abnahmekriterien* für Ihre Softwarelösung sind.

Im *Projektplan* wird auf dieser Basis dargestellt, in welche größeren Arbeitspakete sich die Umsetzung *inhaltlich* untergliedern lässt, wie der Projektfortschritt überwacht werden soll und mit welchem Aufwand Sie für jedes der Arbeitspakete rechnen.

Im *Releaseplan* werden diese Arbeitspakete *zeitlich* weiter aufgeschlüsselt und auf die einzelnen Iterationen verteilt. Der Releaseplan ist nach jedem Release zu überprüfen und ggf. anzupassen.

Vorgaben

Gliedern Sie das Lastenheft in Anlehnung an (Balzert 2009) in die Abschnitte

1. Visionen und Ziele
2. Rahmenbedingungen und Produkteinsatz
3. Kontext und Überblick
4. Funktionale Anforderungen
5. Produktdaten und nichtfunktionale Anforderungen
6. Qualitätsmatrix nach ISO 25010
7. Lieferumfang und Abnahmekriterien
8. Vorprojekt
9. Glossar

Das Lastenheft soll mehr Funktionalität beschreiben als im Rahmen des Projekts implementiert werden kann. Gliedern Sie dazu die Anforderungen in Muss- und Kann-Ziele und zeigen Sie, welche Arbeiten evtl. von Nachfolgeprojekten in Angriff genommen werden können.

[2] Qualitätskriterien und Bewertung von System und Softwareprodukten (SQuaRE) – Qualitätsmodell und Leitlinien. `http://iso25000.com/index.php/en/iso-25000-standards/iso-25010`

Erstellen Sie für den Abschnitt *Qualitätsmatrix* eine tabellarische Ansicht, welche der acht Qualitätsparameter nach ISO 25010 in welcher Ausprägung (hoch, mittel, niedrig, nicht anwendbar) für Ihr Projekt bedeutsam sind.

Erläutern Sie im Abschnitt *Vorprojekt*, was Sie im Rahmen des Vorprojekts umsetzen wollen und welche Arbeitsprodukte zum Ende des Vorprojekts vorliegen sollen.

Übernehmen Sie im Abschnitt *Glossar* eine konsolidierte Version des Glossars aus dem Recherchebericht und verwenden Sie die Glossarbegriffe in den Ausführungen des Lastenhefts auf die vereinbarte Weise. Damit soll erreicht werden, dass sich die Begriffe auf konsistente Weise in der konzeptionellen (Lastenheft – was?) und organisatorischen (Arbeitsplan und Releaseplan – wie und wann?) Planung wiederfinden und damit begriffliche Konsistenz in den Dokumenten erreicht wird.

3.4 Projektplan und Releaseplan

Projektplan

Mit dem *Projektplan* wird eine grobe inhaltliche Strukturierung für die Umsetzungsphase Ihres Projekts entwickelt. Teilen Sie Ihr Projekt dazu in wenigstens vier Arbeitspakete, beschreiben Sie das jeweilige Arbeitspaket genauer und geben Sie jeweils an, mit welchem Aufwand in Prozent des Gesamtprojekts Sie rechnen. Die geschätzten zeitlichen Aufwendungen sollen sich als inhaltliche Proportionen der Ausführungen zu den einzelnen Arbeitspaketen wiederfinden.

Formulieren Sie auch dabei Muss- und Kann-Ziele. Die Aufwandsrechnung kann also in der Summe 100 % überschreiten.

Releaseplan

Ihr Projekt soll in fünf Releasezyklen umgesetzt werden. Im *Releaseplan* wird festgehalten, welche Features in welchem Release umgesetzt werden und welche Arbeitsprodukte dabei fertigzustellen sind.

Für den aktuellen Releasezyklus werden die Aufgaben aus dem Releaseplan in Tasks heruntergebrochen, als *Issues* aufbereitet und konkreten Personen im Team zur Bearbeitung zugewiesen.

Zum Abschluss jedes Releasezyklus ist zu prüfen, welche Aufgaben erfüllt wurden, offene Tasks sind neu einzuordnen und der Releaseplan entsprechend zu überarbeiten.

Weiterhin sind im Releaseplan die Termine für die Abgaben der fünf Releasebündel entsprechend den Vorgaben (jeweils montags 24 Uhr) zu fixieren.

3.5 Qualitätssicherung

Um ein größeres Projekt erfolgreich und termingerecht zu bewältigen ist es sinnvoll, vorab auch einige Energie auf die organisatorische Aufstellung des Teams (die Prozessdimension) zu verwenden und dabei neben der eigentlichen Software-Entwicklung auch das Qualitätsmanagement im Blick zu behalten. Dazu sollen Sie sich in Ihrem Team über die Grundanforderungen an das Projektmanagement verständigen und diese in einem **Qualitätssicherungskonzept**

verbindlich fixieren.

Die **Qualität einer Software** wird neben ihren funktionalen Leistungsparametern auch wesentlich durch die Qualität von Entwurf und Code bestimmt. Ein Maß für diese Qualität ergibt sich aus den verschiedenen Anforderungen an die Software. Dabei sind auch Fragen des Bugfixing, der Wartbarkeit und Änderbarkeit zu beachten, für welche es wichtig ist, dass sich projektfremde Entwickler schnell in Entwurfsaspekte der Software einarbeiten können.

Die Einhaltung einer Reihe von **allgemeinen Prinzipien** unterstützt diese Anforderungen. Es ist sowohl ein Aspekt der Selbstkontrolle als auch Aufgabe des Qualitätsmanagements, die Einhaltung dieser Regeln zu überwachen. Dazu sind Qualitätsstandards auszuarbeiten, im Team verbindlich zu vereinbaren sowie deren Einhaltung zu organisieren und zu kontrollieren.

Dokumentationskonzept

Die (technische) Dokumentation Ihres Projekts soll sich aus drei wesentlichen Komponenten zusammensetzen,

- der *internen Dokumentation* des Quelltexts durch Kommentare,
- einer *quelltextnahen strukturierten Dokumentation*, die mit einem geeigneten Werkzeug (z. B. `javadoc`) extrahiert werden kann,
- sowie der *Entwurfsbeschreibung* (siehe Abschnitt 3.6), in der alle wichtigen Architekturentscheidungen und Entwurfsaspekte verständlich dargelegt und begründet sind.

Beachten Sie hierzu auch die Ausführungen im Dokument *Softwaredokumentation im Praktikumseinsatz der Abteilung BIS* im Materialordner des Kurses, in dem verschiedene Aspekte genauer erläutert sind, die Einfluss auf das zu entwickelnde Dokumentationskonzept haben.

Coding Standard

Zur besseren Lesbarkeit und schnelleren Verständlichkeit des Quellcodes ist es sinnvoll, sich an allgemein übliche *Coding Standards* zu halten.

Coding Standards können mit Code Sniffern[3] überprüft werden, die entweder über die Kommandozeile aufgerufen oder im Rahmen von *Continuous Integration* (CI) als pre-commit-hooks in den git Workflow integriert werden können, siehe Abschnitt 4.2. Mit dieser *konstruktiven Qualitätssicherungsmaßnahme* wird verhindert, dass Code eingecheckt wird, der nicht den vereinbarten Coding Standards entspricht.

Ist Ihr Projekt Teil eines größeren Projekts, so sollte der dort verwendete Coding Standard und auch die dafür verwendeten Werkzeuge Anwendung finden. Anderenfalls sollten Sie sich auf einen verbreiteten Coding Standard für Ihre Programmiersprache einigen, für den ebenfalls Werkzeuge existieren.

Dies ist im Dokumentationskonzept ebenso zu fixieren wie die Konzepte und Werkzeuge, mit denen Sie Ihre quelltextnahe Dokumentation verwalten, damit alle Coder im Team diese Rahmenbedingungen bei ihrer Arbeit beachten und die vereinbarten Werkzeuge nutzen.

[3] etwa `phpcs` und `phpcbf` für PHP-Code.

Testkonzept

Das Testkonzept berücksichtigt die Qualität der einzelnen Teile Ihrer Entwicklung (Komponententests) sowie die Qualität der Zusammenführung der Teile (Integrations- und Systemtest).

Im Mittelpunkt von **Komponententests** steht die Funktionalität der einzelnen Komponenten, die am besten durch eine gut überlegte Auswahl von Testfällen überprüft werden kann. Ziel von Komponententestfällen ist es, einen Indikator dafür zu haben, ob eine Komponente oder Teile davon spezifikationsgemäß funktionieren. Dazu dient eine Suite von Testfällen, mit denen alle mehr oder weniger wichtigen funktionalen Aspekte und Programmzweige abgedeckt werden und die vor dem eigentlichen Programmieren der Komponente oder der Änderungen erstellt sein sollten. Mit entsprechenden Werkzeugen kann auch der Grad der Testabdeckung (test coverage) bestimmt werden, siehe Abschnitt 4.2.

Für das (dokumentierte) Ablaufen solcher Testfälle ist eine Umgebung erforderlich, in der die Abhängigkeiten der Komponenten schon zu einer Zeit aufgelöst werden können, zu der das gesamte System noch nicht fertig ist. Für derartige Szenarien gibt es verschiedene Frameworks wie *JUnit*, *Selenium* oder *phpunit*, die auch automatisierte Tests erlauben, die mit entsprechenden Testwerkzeugen wie *Jenkins*[4] in einen CI-Prozess integriert werden können. Continuous Integration als konstruktive Qualitätssicherungsmaßnahme garantiert hierbei, dass nur Code ins Team-Repo eingecheckt werden kann, der alle Tests erfolgreich passiert hat. Dies ist im Abschnitt 4.2 genauer erläutert.

Ist Ihr Projekt Teil eines größeren Projekts, so sollte das dort verwendete Test-Framework und dessen Werkzeuge für Ihre Tests nachgenutzt und angepasst werden.

Neben automatisierten Tests spielen in fortgeschrittenen Projektphasen auch manuelle Tests insbesondere verschiedener GUI-Funktionalitäten eine wichtige Rolle.

Fehler, die während der Tests gefunden werden, sind zu erfassen und als Issues aufzubereiten, damit die weitere Fehlerbeseitigung verfolgt werden kann.

Das Qualitätssicherungskonzept

Die wesentlichen Eckpunkte der Qualitätssicherung Ihres Projekts sind im **Qualitätssicherungskonzept** zu fixieren. Neben inhaltlichen Vereinbarungen gehören dazu auch organisatorische Festlegungen über Termine und Verantwortlichkeiten für Zuarbeiten sowie über das Zusammenführen dieser Teile zu einem Gesamtbild.

Gliedern Sie Ihr QS-Konzept in die Abschnitte *Dokumentationskonzept*, *Testkonzept* und *Organisatorische Festlegungen.*

3.6 Entwurfsbeschreibung

Die Entwurfsbeschreibung ist ein zusammenfassendes Dokument, das alle wichtigen Informationen zu den Struktur- und Entwurfsprinzipien der Software enthält, welche ein durchschnittlicher, bisher mit dem Projekt nicht befasster Programmierer kennen sollte, bevor er Änderungen oder Ergänzungen des Quellcodes vornimmt (und dabei die detailliertere, etwa

[4] `https://jenkins.io`

javadoc basierte Dokumentation des Quellcodes nutzt). Die Entwurfsbeschreibung ist damit der Einstiegspunkt für qualifizierte externe Dritte, die sich mit der Umsetzung Ihres Projekts genauer vertraut machen möchten.

In der Entwurfsbeschreibung sind alle wichtigen Modellierungs-, Struktur- und Designentscheidungen zu begründen. Erläutern Sie dabei insbesondere, mit welchen Entscheidungen Sie welche Konzepte und Ansätze umgesetzt haben.

Im Praktikum ist eine Entwurfsbeschreibung in folgende Hauptpunkte zu gliedern:

1. Visionen und Ziele
2. Rahmenbedingungen und Produktübersicht
   Beschreibung der äußerlichen Funktionsmerkmale des Systems.
3. Grundsätzliche Struktur- und Entwurfsprinzipien
   Was sollte eine Informatikerin über das Gesamtsystem wissen, ehe sie sich allgemeinen Details zuwendet?
4. Struktur- und Entwurfsprinzipien einzelner Pakete
   Was sollte ein Informatiker über 3. hinaus wissen, ehe er sich Details eines speziellen Pakets zuwendet?
5. Datenmodell
6. Glossar

3.7 Releasebündel

Mit einem iterativen Entwicklungsmodell soll das System als eine Folge von Release-Versionen entwickelt werden, in denen die Funktionalität Schritt für Schritt erweitert wird. Es ist besonders wichtig, *während* jeder Iteration arbeitsteilig vorzugehen, um die gesetzten zeitlichen Rahmen einzuhalten. Dazu muss alles genau geplant und sinnvoll aufgeteilt, am Ende aber auch geprüft werden, ob alles zusammenpasst. Jede Iteration umfasst deshalb neben einer Arbeits- auch eine Integrationsphase, in der geprüft wird, dass bzw. ob alles zusammenpasst *und auch Dokumentation und Tests auf dem aktuellen Stand* sind.

Zur Umsetzung dieser Integrationsanforderung werden zum Abschluss jeder Iteration die Zuarbeiten zu einem neuen lauffähigen (!) Release mit erweiterter Funktionalität integriert. Das hat den Vorteil, dass bereits während der Entwicklung dem Auftraggeber der Prototyp mit eingeschränkter Funktionalität vorgeführt werden kann. In der Praxis wirkt sich das vertrauensfördernd aus und erlaubt es, Anforderungen auf der Basis gewonnener Erfahrungen auch noch unterwegs zu modifizieren, wenn alle beteiligten Seiten mitspielen. Zugleich können mit der Auswahl und Abgrenzung der Tasks Projektteile verschieden priorisiert werden.

Ein **Releasebündel** besteht aus

- dem Quellcode (mit standardisiertem Inline-Anteil der Dokumentation),
- dem Testmaterial,
- der aktuellen Version der Entwurfsbeschreibung,
- einem aktuellen Testbericht,
- dem aktualisierten Releaseplan,

- einer aktuellen Aufwandsanalyse sowie
- einer Demoversion des aktuellen Release auf der Webseite Ihres Teams.

  Es kann für einzelne Themen mit dem Betreuer eine andere Form der Präsentation vereinbart werden.

Quellcode, Beschreibungen und Testmaterial sind in jeweils eigenen Verzeichnissen abzulegen und danach alles zu einer gemeinsamen zip-Datei zusammenzupacken.

4 Optionale Konzepte und Ansätze

4.1 Deploy der Webseiten mit Jekyll

Autor: Natanael Arndt, AKSW-Gruppe

Es ist möglich, die Webseiten in einem eigenen Ordner im Team-Repo zu verwalten und mit Jekyll zu deployen. Um diesen Builtservice zu nutzen müssen drei Schritte durchgeführt werden:

1. Es müssen dem Nutzer `wwwrun` Schreib- und Leserechte auf ihrem `~/public_html` Ordner eingeräumt werden. Dies erfolgt mit dem Befehl

   ```
   setfacl -m u:wwwrun:rwx ~/public_html
   ```

2. In ihrem Gruppen-Repository muss ein Ordner `/Webseiten` angelegt werden, in den die auszuliefernden Seiten eingecheckt werden.

3. Ein *push event webhook* muss für den Service mit der URL

   ```
   http://pcai042.informatik.uni-leipzig.de/cgi-bin/buildsite.sh
   ```

   eingerichtet werden.

   Dies geschieht im ifi-gitlab über das Menü mit dem Zahnrad am rechten Rand der Seite unter dem Menüpunkt *Webhooks*. Es muss lediglich die URL eingetragen und sichergestellt werden, dass der Haken vor *Push events* gesetzt ist.

Der Buildservice ist vergleichbar zu GitHub pages und erlaubt Ihnen sowohl einfache statisch HTML Seiten zu deployen als auch mit Jekyll[5] Seiten zu erstellen. Wenn Sie ihre Seite mit Jekyll erstellen, muss in der Datei `_config.yml` der Wert `baseurl:` auf `/~<gruppenname>/jekyll` gesetzt werden. Die durch den Service ausgelieferten Seiten sind unter der URL

```
http://pcai042.informatik.uni-leipzig.de/~<gruppenname>/jekyll
```

erreichbar.

Möchten Sie Ihre komplette Projektseite mit dem Deployservice verwalten, so sollten Sie eine einfache Weiterleitungsdatei in `~/public_html` einrichten.[6]

Bemerkung: Es empfiehlt sich, die neue Jekyll Seite im Repository mit den zwei folgenden Befehlen zu erstellen:

```
jekyll new --skip-bundle Webseiten
cd Webseiten && bundle install --path vendor/bundle
```

Bitte checken Sie in diesem Fall den Ordner `vendor/` **nicht** in das Repository ein. Sie können dazu zusätzlich einen Eintrag für `vendor/` in die Datei `.gitignore` eintragen.

[5] `https://jekyllrb.com/`

[6] Entweder mit einer HTML-Weiterleitung in einer `index.html` oder mit einer Datei `.htaccess` in `/~<Gruppenname>/`. Weitere Hinweise finden sie unter `https://de.wikipedia.org/wiki/Weiterleitung`.

4.2 Continuous Integration

Autor: Roy Meissner, AKSW-Gruppe

Um Ihnen eine effektive Möglichkeit des Testens an die Hand zu geben und für die Tutoren und Betreuer den Testverlauf und das Setup Ihres Projektes sichtbarer zu machen, empfehlen wir die Nutzung von Continuous Integration (CI). Zu diesem Zweck ist im Gitlab `Gitlab CI` integriert, mit dem sie CI auf einfache und effiziente Weise aufsetzen und nutzen können.

Gitlab CI führt für jeden Commit auf ihren Master-Zweig des Gitlab Repos einen sogenannten Build aus. Ein Build läuft immer exakt gleich ab: Sie starten mit einem blanken System (bspw. Ubuntu 16.04). Gitlab checkt den aktuell betrachteten Commit Ihres Repositories aus und übergibt Ihnen im Anschluss das System. Sie beschreiben als Skript, welche Abhängigkeiten installiert werden müssen (bspw. Java JRE und JDK) und testen im Folgenden Ihre Anwendung auf verschiedenste Aspekte, bspw. auf Code-Style, ob das Programm kompiliert, auf Unit- und Integrationstests, etc. Gitlab führt für Sie alle diese Schritte für jeden Commit aus und benachrichtigt Sie per E-Mail bzw. auf Gitlab, sobald es bei einem der Schritte Probleme gibt (also bspw. der Code-Style verletzt wurde). Gleichzeitig ergänzt Gitlab die Information zu den jeweiligen Commits.

Im Endeffekt werden Sie so wesentlich schneller benachrichtigt, ob ein Problem vorliegt, und können dieses beheben, bevor es sich zu einem größerem Problem entwickelt, dessen Behebung viel Zeit und Aufwand kosten würde.

Die ausgeführten Tests müssen von Ihnen definiert werden. So können Sie bspw. über die Software Maven definieren, was (bzw. welches Programm) ausgeführt werden soll, um Ihre Unit-Tests auszuführen oder den Code-Style zu überprüfen. Die meisten Package Manager (bspw. Maven oder NPM) bieten hierzu die Möglichkeit und Plugins, über die bereits viele Aufgaben standardisiert sind.

Um CI für Ihr Projekt einzurichten, folgen Sie bitte den beschrieben Schritten in der Anleitung von Gitlab: `https://docs.gitlab.com/ee/ci/`. Es sollte inklusive dem Einlesen weniger als 30 Minuten dauern, bis Sie eine funktionierende CI für Ihr Projekt aufgesetzt haben.

4.3 Continuous Code Quality mit Sonarqube

Autor: Johannes Frey, Gruppe AKSW KILT

Analog zu Continuous Integration/Continuous Deployment (CI/CD) wird auch die Anwendung von Konzepten der Continuous Code Quality empfohlen.

Im Beispiel des Einsatzes von Sonarqube für Java `https://www.sonarqube.org/` können, basierend auf vordefinierten Regeln, automatisiert Verstöße gegen Code-Conventions (z.B. klein geschriebene Konstantennamen), Bugs (potentielle Nullpointer, nicht abgefangene Exceptions, nicht geschlossene Dateien etc.) und Sicherheitslücken gefunden werden. Die Problemreporte können (z.B. automatisiert mit Maven) nach jedem Commit generiert und unter verschiedenen Aspekten im Browser analysiert werden. Dies ermöglicht dem Projektleiter und Qualitätssicherungsverantwortlichen ein kontinuierliches Monitoring der Codequalität im Detail oder aggregiert nach verschiedenen Indikatoren.

Sonarqube unterstützt diverse Programmiersprachen und ist überdies als Erweiterung für verschiedene IDEs verfügbar, um die Entwickler bereits interaktiv beim Programmieren auf Probleme der Code-Qualität hinzuweisen. Der Einsatz empfiehlt sich besonders für unerfahrene Programmierer, denn er unterstützt diese, sauberen, performanten, fehlertoleranten und wartbaren Code zu schreiben. Jedoch auch erfahrene Programmierer können sich durch den Einsatz spielerisch mit Standards wie CERT Secure Coding Standards oder MISRA (Motor Industry Software Reliability Association) vertraut machen. In Kombination mit dem Einsatz von CI/CD ist es möglich, Commits automatisiert abzuweisen, wenn diese den konfigurierten Standards nicht genügen.

4.4 Umgang mit Wissensgefälle in Softwareprojekten

Autoren: Stefan Koch, Sebastian Volke, Softwareforen

Immer wieder kommt man in Softwareprojekten in die Situation, dass der Kenntnisstand der einzelnen Entwickler auseinander geht. Das Team muss damit umgehen können und im Idealfall schwächere Teammitglieder ausbilden. Dazu wird eine geeignete Team-Kultur benötigt. Darauf aufbauend wird exemplarisch das Werkzeug „Pair Programming“ konkret vorgestellt.

Eine Team-Kultur, die Entwickler entwickelt

Ein Wachstum ist nur möglich in einer Kultur, die Verbesserung fördert. Das setzt eine Kultur voraus, die Fehler zulässt und unverkrampft damit umgehen kann. Dafür gibt es einige Grundhaltungen, die vom Entwicklerteam gelebt und von den Führungskräften gezielt implementiert werden müssen:

- Niemand muss alles richtig machen. Wir respektieren jeden unabhängig von seinem Output.
- Jeder kann noch etwas dazulernen. Deshalb strebt jeder nach ständiger Verbesserung.
- Wir ringen um ein gemeinsames Verständnis vom „besten Code“.
- Wir sind offen für neue Herangehensweisen und Experimente.
- Es darf über alles diskutiert werden und es darf auch alles in Frage gestellt werden. Wir nehmen das nicht persönlich, sondern verstehen es als Einladung zum Lernen.

Das Ziel ist es, eine Atmosphäre zu erreichen, in der gerne und viel diskutiert wird. So können unterschiedliche Sichtweisen zu Stilfragen, aber auch zu technologischen Präferenzen und Architekturentscheidungen ausgetauscht werden. Nicht immer ist es unter wirtschaftlichen Gesichtspunkten möglich, die Ergebnisse der Diskussion in laufenden Projekten auch direkt anzuwenden. Aber erzielt wird mindestens eine Verbreiterung des Erfahrungshorizonts und eine bessere Zusammenarbeit der Entwickler in zukünftigen Projekten.

Diese Kultur des gemeinsamen und voneinander Lernens kann ganz besonders intensiv im Pair Programming gelebt werden.

Entwickler entwickeln praktisch: Pair Programming

Beim Pair Programming arbeiten zwei Entwickler an einem Arbeitsplatz. Während ein Entwickler als „Navigator“ sich vorrangig mit der Planung der Arbeit und Seiteneffekten befasst, übernimmt der „Driver“ die Arbeit an der Tastatur. Der Navigator „steuert“ den Driver und gibt ihm Anweisung, was jeweils als Nächstes zu tun ist. Abhängig von der Erfahrung des Drivers variiert die Flughöhe der Anweisungen. Grundsätzlich ist auch ein regelmäßiger Wechsel dieser Rollen vorgesehen.

Im Folgenden werden kurz die Vorteile von Pair Programming erläutert.

Fokussierung, Code-Qualität und Effizienz. Pair Programming erhöht die Fokussierung des einzelnen Entwicklers. Dadurch, dass gemeinsam an einer Sache gearbeitet wird, reduziert sich die Gefahr, abgelenkt zu werden. Durch die unterschiedliche Betrachtungsweise der Entwickler auf den gleichen Code (Driver: Details, Navigator: Gesamtbild) ergibt sich gleichzeitig ein deutlich größeres Potential zur frühzeitigen Fehlererkennung. Gleichzeitig werden Code-Smells, im Sinne von „schlechtem Code“ (was natürlich Verhandlungssache ist...) tendenziell eher vermieden, weil sie vom jeweils anderen Entwickler nicht unbedingt akzeptiert werden. Weiterhin steht das gesammelte Wissen zweier Programmierer zur Verfügung, was die zur Verfügung stehenden Ideen und damit auch die Qualität der gebauten Lösung steigert.

Somit wird erwartet, dass im Pair Programming wesentlich weniger Aufwand für Refactoring und Debugging anfällt. In der Praxis zeigt sich, dass Pair Programming deutlich effektiver ist als Einzelprogrammierung mit anschließender Qualitätssicherung durch einen zweiten Entwickler.

Teambildung und Lerneffekte. Ein weiterer Vorteil von Pair Programming ist die Förderung der Teambildung. Durch die unmittelbare Zusammenarbeit lernen sich die Teammitglieder besser kennen und festigen ihre fachlichen Kommunikationsfähigkeiten. Außerdem entsteht durch die gemeinsamen Erfolge (und möglicherweise Misserfolge) ein Gemeinschaftsgefühl: Pair Programming schweißt zusammen.

Auf der anderen Seite befördert Pair Programming das Lernen. Jeder Entwickler kann von der Erfahrung des anderen profitieren bzw. ihn in gleicher Weise herausfordern, seine etablierten Praktiken zu hinterfragen. Das gilt auf allen Ebenen, von der Bedienung des Rechners oder der IDE über die Verwendung der Infrastruktur bis zur Strukturierung des Quellcodes auf Funktions-, Klassen-, Modul- oder Projektebene.

Paarkonstellationen. Spannend ist, wann der Lerneffekt beim Pair Programming am Größten ist. Verschiedene Konstellationen hinsichtlich des Erfahrungsgrads der Entwickler sind möglich:

Senior–Senior

Hierbei kann man die höchste Produktivität beim Programmieren erreichen, da beide Entwickler sehr selbstständig sind, sich der Navigator ganz auf die Seiteneffekte konzentrieren kann und die Steuerung des Drivers auf einem sehr hohen Level erfolgt. Der Lerneffekt ist aber geringer, da Senior-Entwickler tendenziell seltener ihre etablierten Praktiken hinterfragen. Hier hat Pair Programming vor allem die Funktion der 4-Augen-Kontrolle.

Junior–Junior

Diese Form findet seltener Anwendung, kann jedoch auch zu verbesserten Resultaten führen. Die gemeinsame Arbeit verstärkt oft das Vertrauen in den eigenen Code, was wiederum die Produktivität begünstigt.

Senior–Junior

Diese Aufstellung bietet den idealen Lerneffekt. Der Junior-Entwickler kann von der Erfahrung des Senior-Entwicklers lernen. Dieser wird wiederum zu größerer Selbstreflexion motiviert, da er den Sinn seines Vorgehens erklären muss. Wichtig ist hierbei: Der Senior-Entwickler soll

offen für alle Rückfragen sein und muss aufpassen, den Junior nicht durch seine feste Meinung oder sein Auftreten zu „erdrücken“. Wenn das gelingt, kann die Ausbildung sehr effektiv sein.