7 Implementierung

> Talk is cheap. Show me the code.
>
> *(Linus Torvalds)*

Artefakt der Phase Implementierung: Implementierungsplan zu Beginn, Implementierungsbericht als PDF Dokument und vollständiger Sourcecode

7.1 Plan

- Klarer zeitlicher Ablauf der Implementierungsphase, um frühzeitig Verzögerungen zu bemerken.
- Abhängigkeiten aus dem Entwurf müssen beachtet werden. Wo ist der Critical Path?
- Wie können Teile der Anwendung möglichst früh getestet werden? Braucht es dafür Stub-/Mock-Klassen?
- Klare Aufgabenverteilung im Team. Dabei muss eine faire Verteilung und die Abhängigkeiten beachtet werden.
- Als Form bietet sich ein GANTT Chart an, wie das Beispiel in Abbildung 2. Verpflichtend ist es aber nicht. Zum Beispiel lässt sich auch eine Tabelle oder `dot` benutzen.
- Einzelne Jobs kann man Klassen oder Packages zuordnen, aber häufig ist das nicht möglich. Zwischen vielen Klassen gibt es zirkuläre Abhängigkeiten. Manche Aufgaben erfordern nur Teile von Klassen. Da bietet es sich an, gröbere Aufgaben festzulegen. Auch könnte man die Testszenarien als Aufgabenbeschreibung nutzen.
- Zu *Beginn* der Implementierungsphase (d.h. nach 1-2 Tagen) beim Betreuer abzugeben.

7.2 Bericht

- Erfahrungsgemäßer Umfang: ca. 20 Seiten
- Einleitung mit Anschluss auf Pflichtenheft und Entwurf
- Dokumentation über Änderungen am Entwurf, beispielsweise entfernte oder neu hinzugefügte Klassen und Methoden. Gruppiert (und zusammengefasst) werden sollte nach dem Grund für die Änderung und nicht nach der geänderten Klasse.
- Welche Muss- und Wunschkriterien sind implementiert?
- Welche Verzögerungen gab es im Implementierungsplan? Kann beispielsweise als zweites GANTT Diagramm am Ende dargestellt werden.
- Übersicht zu Unittests

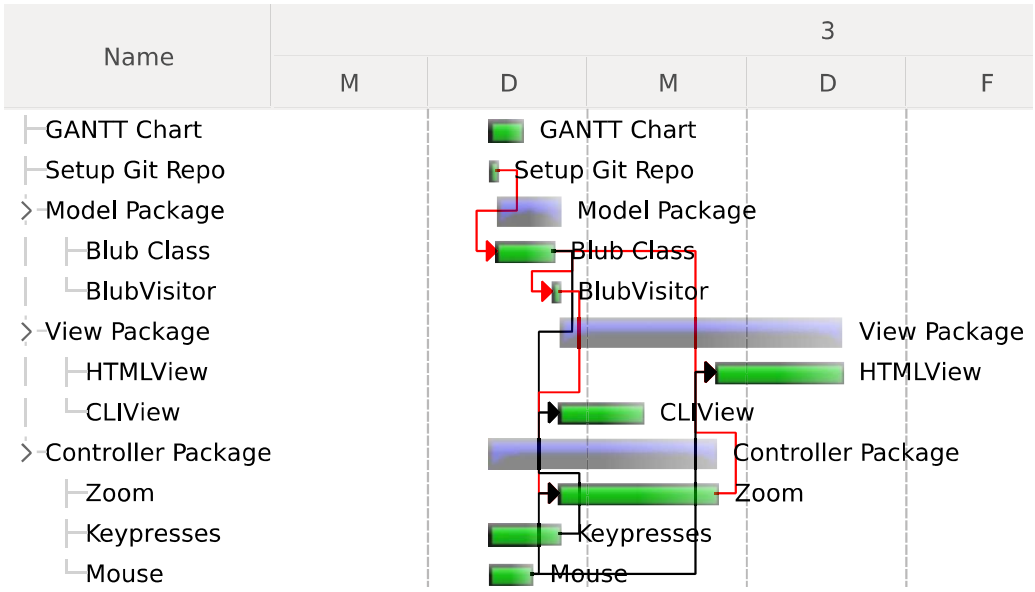

Abbildung 2: Beispiel GANTT Chart: Jede einzelne Klasse ist als Aufgabe eingetragen. Abhängigkeiten sind als Pfeile erkennbar. Der kritische Pfad ist in rot gekennzeichnet.

7.3 Unittests

- Von Anfang an Unittests benutzen. Diese Tests gehören *nicht* in die Phase Qualitätssicherung.
- Vor allem nicht-graphische Klassen können so frühzeitig getestet und Regressionen vermieden werden.
- Nur öffentliche Schnittstellen testen. Private Methoden werden nicht getestet.
- Unittests testen nur kleine „units“ (üblicherweise einzelne Klassen) für sich. Es sollen nicht ganze Testszenarien getestet werden.

7.4 Sonstiges

- Warnungen reparieren. Es ist empfehlenswert, dass ein Projekt beim Bauen keine Warnungen ausgibt. Eine Warnung bedeutet, dass der Code üblicherweise aber nicht garantiert fehlerhaft ist. In Ausnahmen kann der Programmierer in Java dann Annotationen einfügen um Warnungen zu unterdrücken.
- Warnungen anschalten. In Eclipse kann man zusätzliche Warnungen anschalten: Project properties → Java → Compiler → Errors/Warnings. Standardmäßig wird das meiste ignoriert. Beispielsweise kann man aktivieren, eine Warnung anzuzeigen wenn eine Klasse `equals` überläd aber nicht auch `hashCode`.
- Kommentare im Code (also nicht API Doku) sollten das „Warum“ klären, nicht das „Wie“. Es ist naturgemäß schwierig vorherzusehen, welche Warum-Fragen sich jemand stellt, der ein Stück Code liest. Die Fragen sind üblicherweise „Warum ist X hier notwendig?“, „Warum ist kein X hier?“ oder „Warum X, wobei Y doch besser wäre?“.

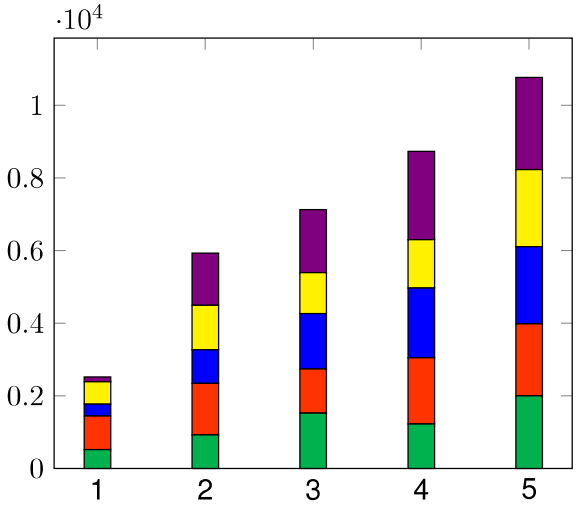

Abbildung 3: Anzahl Codezeilen je Person über 5 Wochen hinweg. Diese Art von Graph zeigt dass die Verteilung im Team fair aussieht. Man sieht auch das kontinuierliche Wachstum.

- Exceptions (nicht RuntimeException) für Fehler beim Aufrufer. Assertions für interne Konsistenzfehler bzw. um Annahmen explizit zu machen.
- Keine Magic Numbers. Explizite Zahlen im Quellcode sind entweder selbsterklärend oder durch benannte Konstanten zu ersetzen. Am besten immer letzteres.

7.5 Inhalt der Präsentation

- Was wurde implementiert? Welche Wunschkriterien gestrichen?
- Zeitplan eingehalten? Wo nicht?
- Unerwartete Probleme bei der Implementierung?
- Größere Änderungen am Entwurf?
- Grobe Statistiken. Lines of Code, Wieviele Commits, Arbeitsaufteilung im Team.
- Entwicklungsmodell. Wie wurde im Team kommuniziert? Wie wurde Git benutzt (Gab es Feature-Branches? War das Mergen in den master-Branch beschränkt? . . .)