Vorkurs in Informatik: Tag 2

Eine Einführung in die Systematische Software Entwicklung

Emanuel Giger, Giacomo Ghezzi, Michael Würsch, and Harald Gall

University of Zurich, Switzerland

University of Zurich
Department of Informatics

s.e.a.l.
software evolution & architecture lab

Ablauf

1. Tag: Grundlagen
2. Tag: Software Engineering by Example
3. Tag: Einführung in die Programmierung

Ablauf: Tag 1

09:30 bis 12:00

- Was ist ein Computer?
- Wie ist ein Computer aufgebaut?
- Das Rechnen mit Wahrheitswerten
- Zahlensysteme
- Wie bringe ich den Computer dazu, für mich Probleme zu lösen?

13:00 bis 16:00

- Eine Einführung in die Programmierung mit Scratch

Ablauf: Tag 2

09:30 bis 12:00

- Eine Einführung in das systematische Entwickeln von Software (aka. Software Engineering)
- Beginn Gruppenarbeiten: Ein kleines eigenes Projekt mit Scratch

13:00 bis 16:00

- Fortsetzung vom Morgen

Ablauf: Tag 3

09:30 bis 12:00

- Kurzpräsentationen der Gruppenarbeiten vom Vortag
- Eine Einführung in die Programmierung mit Groovy

13:00 bis 16:00

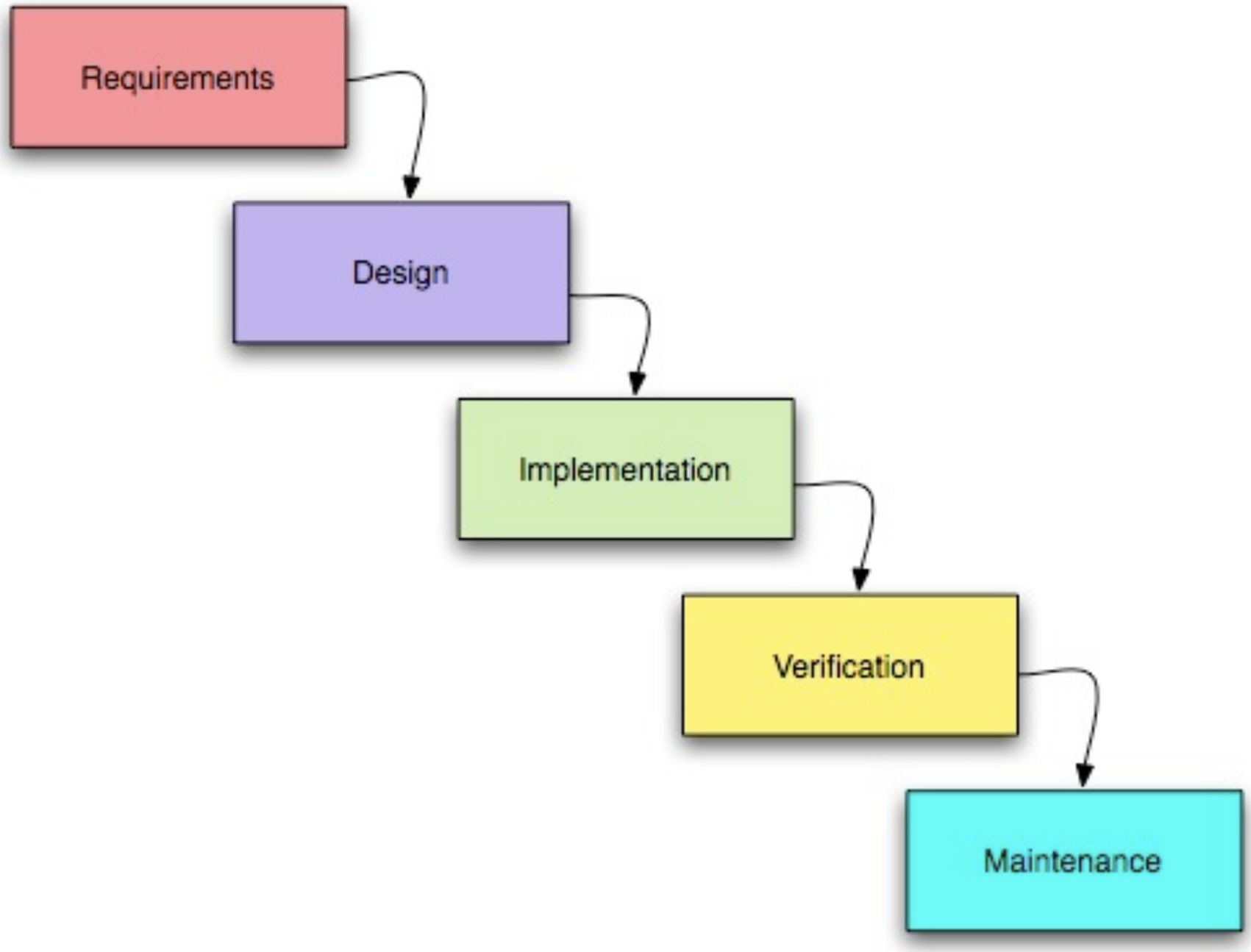
- Fortsetzung vom Morgen

Zielsetzung

Einen Vorstellung davon erhalten, welche Schritte von der Anforderung bis hin zum fertigen Softwareprodukt nötig sind.

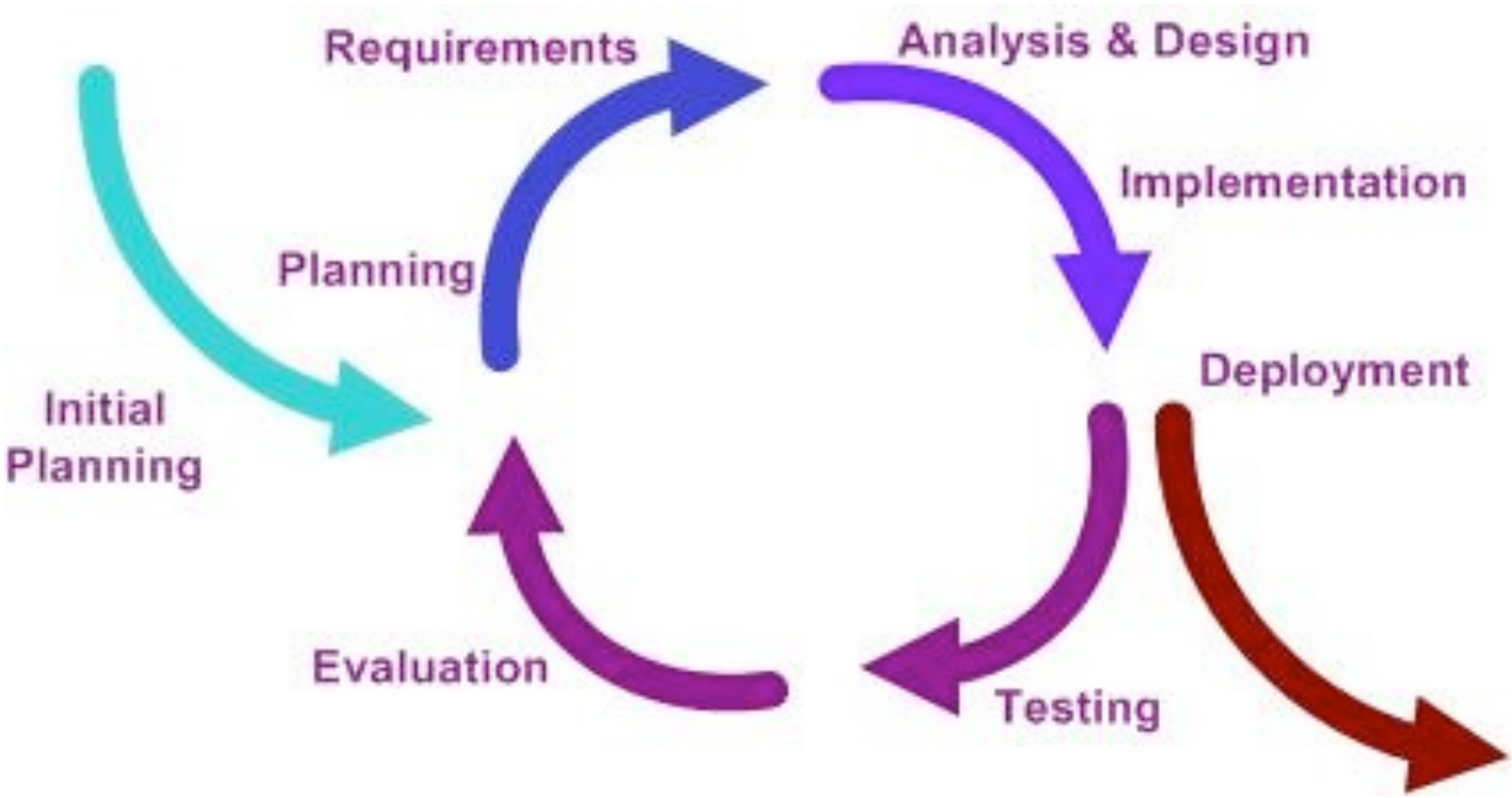
Ein erstes eigenes Produkt entwickeln.

Der Software Lebenszyklus auf dem Papier



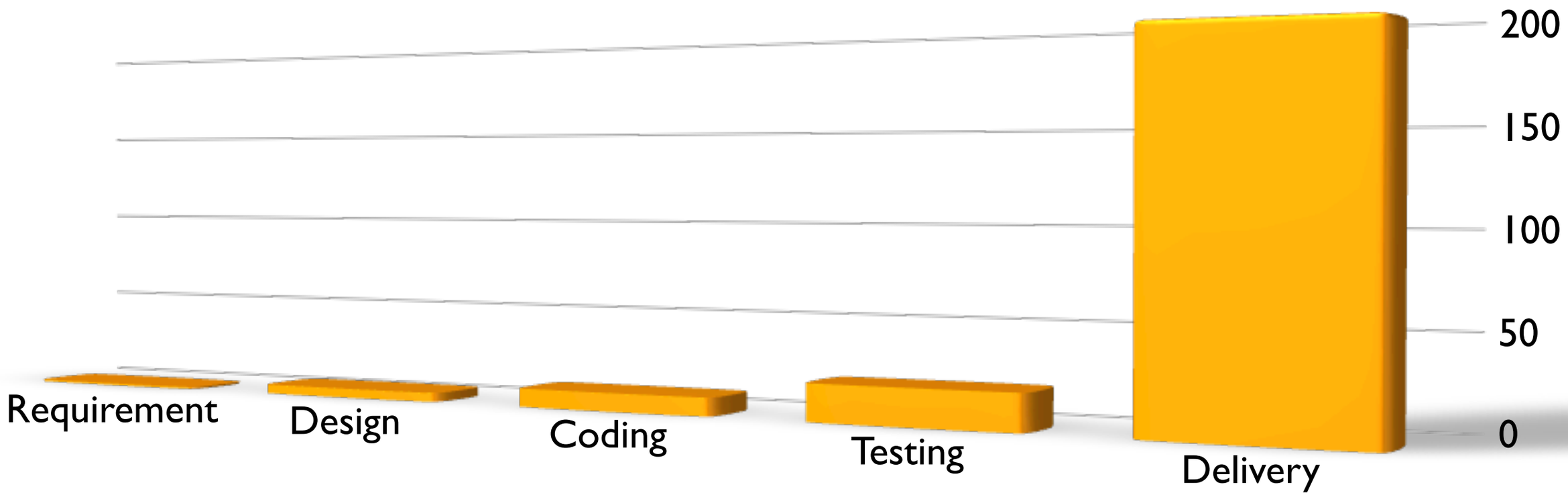
Wasserfallmodell, sequenziell. (Ausser für Kleinst-Projekte) Unrealistisch.

Der Software Lebenszyklus in der Realität



Iteratives Prozessmodell. Neue Anforderungen tauchen während der Entwicklung auf. Gute Software wird auch nach der ersten Auslieferung weiterentwickelt und mit neuer Funktionalität versehen werden.

Die Kosten der Fehlerbehebung



Relative costs of fixing mistakes [Davi95]

Die Behebung eines Fehler, der sich bei der Anforderungsanalyse einschleicht, kostet:
1x soviel wenn er bereits während der Requirements Engineering Phase entdeckt und behoben wird
5x soviel wenn er während der Design Phase entdeckt und behoben wird
10x soviel wenn er während der Coding Phase entdeckt und behoben wird
20x soviel wenn er während der Testing Phase entdeckt und behoben wird
200x soviel wenn er erst nach der Auslieferung an den Kunden entdeckt und behoben wird

Fazit: Anforderungsanalyse sorgfältig durchführen, nicht einfach darauf los programmieren!
Frühzeitig dem Kunden sog. Prototypen präsentieren.

Pareto-Prinzip* im Software Engineering

20% der Anforderungen bedingen 80% der Komplexität

80% des Systems sind in 20% der Zeit fertig gestellt

20% des Codes beinhalten 80% der Fehler

80% der Fehler werden in 20% der Zeit behoben

*Nach Vilfredo Pareto (1848-1923), Italienischer Ökonom und Gesellschaftstheoretiker.

Die Anforderungsanalyse

Perfektion ist erreicht, nicht, wenn sich nichts mehr hinzufügen lässt, sondern, wenn man nichts mehr wegnehmen kann.

- **Antoine de St. Exupery, Terre des Hommes, 1939**

Anforderungen?

Gut formulierte Anforderungen könnten wie folgt lauten:

- Eine Mitarbeiterakte kann lediglich von einer dazu speziell berechtigten Gruppe von Personen eingesehen werden.
- Der Texteditor soll Schlüsselwörter hervorheben, die abhängig vom gerade bearbeiteten Dateityp ausgewählt werden.

Ungenau formulierte Requirements führen zu Fehlern.
Aber auch zu spezifisch formulierte Requirements können zu unerwünschten Resultaten führen. Siehe Beispiel oben: “Nur Vorgesetzte und Angehörige der Personalabteilung können eine Mitarbeiterakte einsehen.” -> Dies könnte dazu führen, dass ein Programmierer explizite Überprüfungen à la “wenn der Benutzer der Personalabteilung angehört, dann...” einführt. Derartige Business Policies ändern jedoch stetig - allgemeiner formuliert führt die Anforderung vielleicht dazu, dass der Programmierer eine Art von Zugriffberechtigungssystem implementiert.

Wie erhebe ich Anforderungen?

Work with a User to Think Like a User

- **Tip 52, Andrew Hunt and David Thomas in ‘The Pragmatic Programmer’**

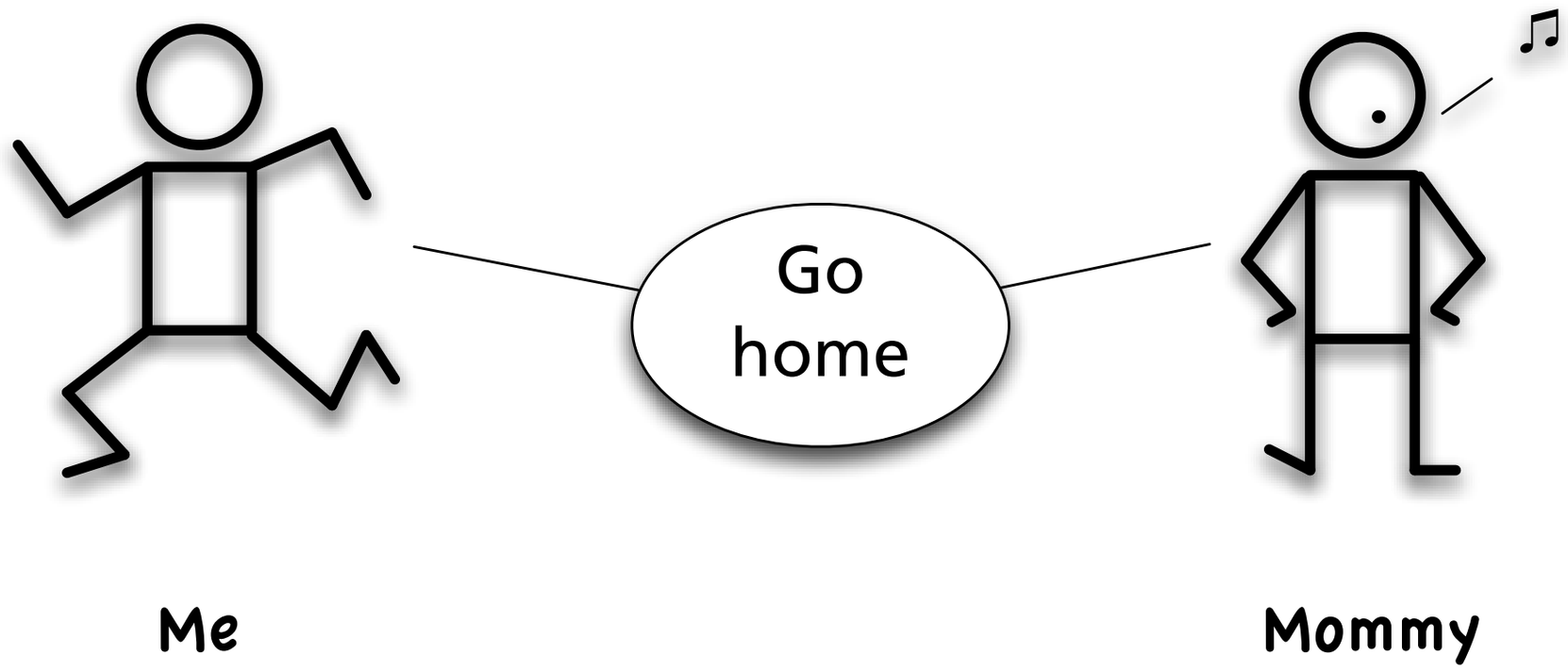
Gemeinsam mit den Stakeholders.

Für wen dokumentiere ich Anforderungen?

Wir dokumentieren Anforderungen für Entwickler, die End-Benutzer, die Projektsponsoren, etc. - Das ist ein ziemlich breites Publikum!

Wie dokumentiere ich Anforderungen?

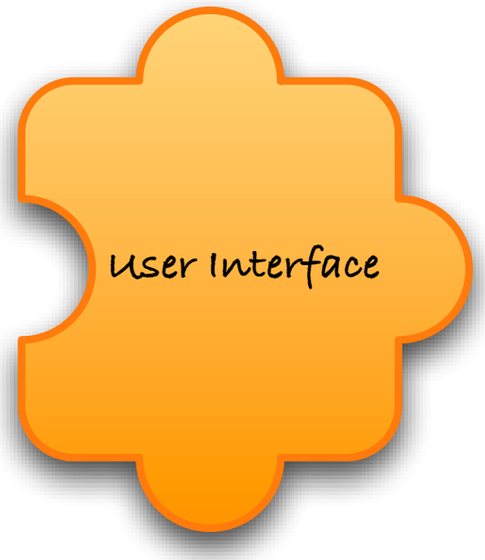
Use Cases und Use Case Diagramme



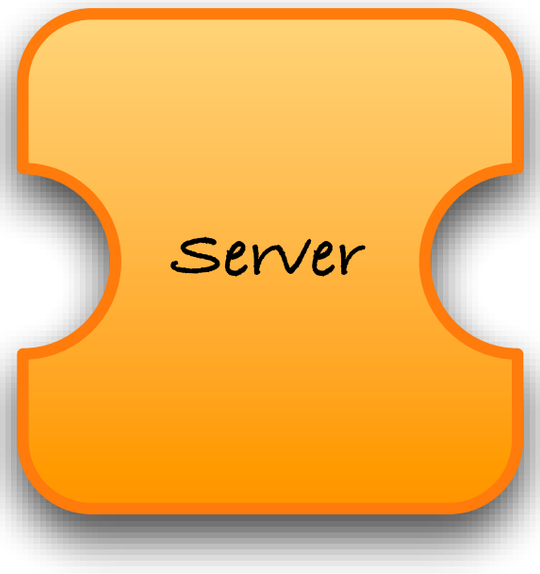
Use cases: Siehe Use Case Template/Beispiel.
Dokumentation so einfach wie möglich halten, damit alle Beteiligten sie verstehen, aber so präzise wie nötig sein, damit die Anforderungen auch aussagekräftig sind. Wichtig: Don’t be a slave to any notation!

Entwurf

- Daten laden
- Daten speichern
- ...



- Bild darstellen
- Liste mit Namen anzeigen
- ...



- Anfrage XY beantworten
- Passwortabfrage senden
- ...

- Berechnung XY durchführen
- ...

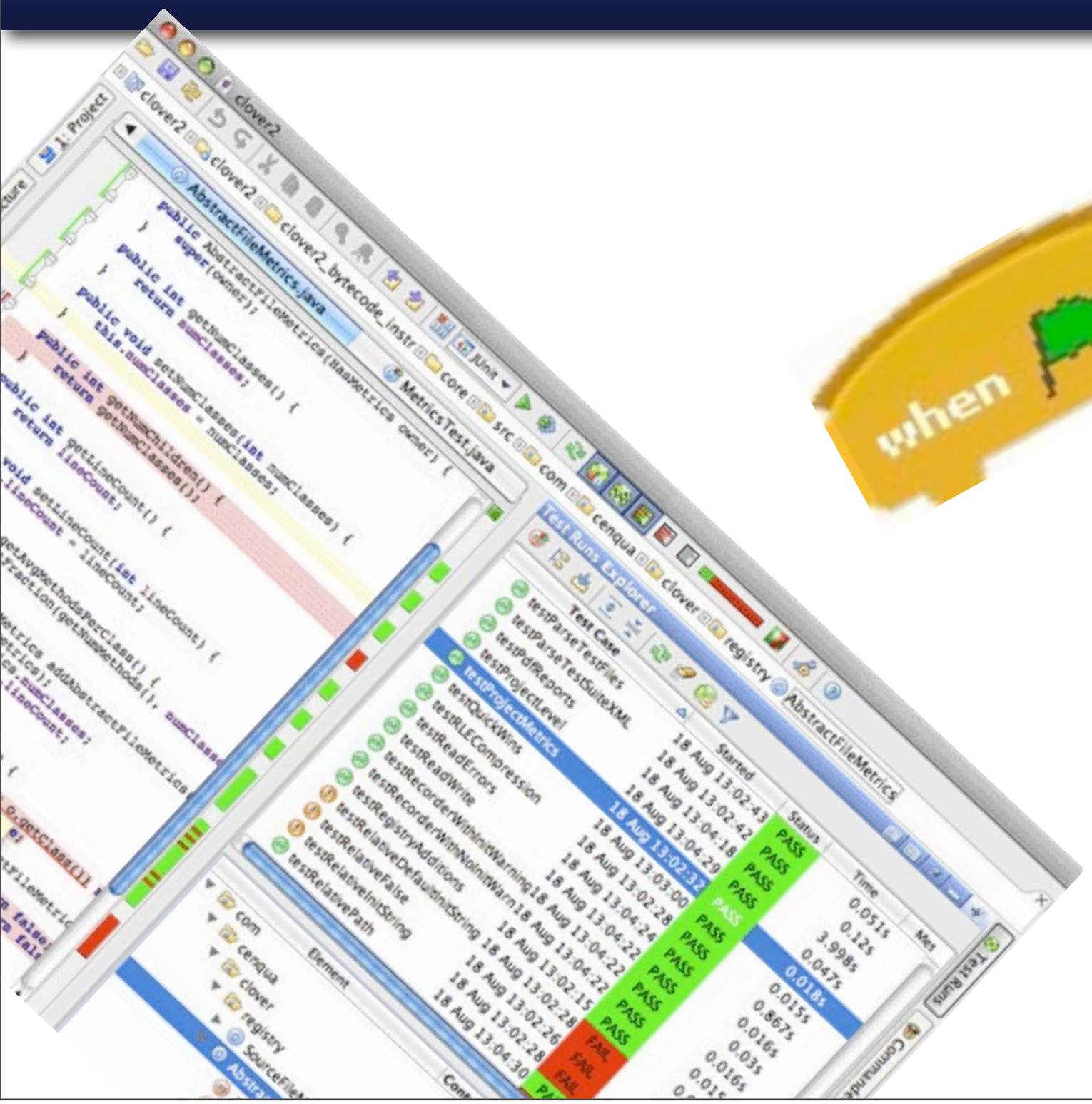
Die wichtigsten Bausteine der Applikation festlegen. Welche Funktionalität muss welcher Baustein erbringen?

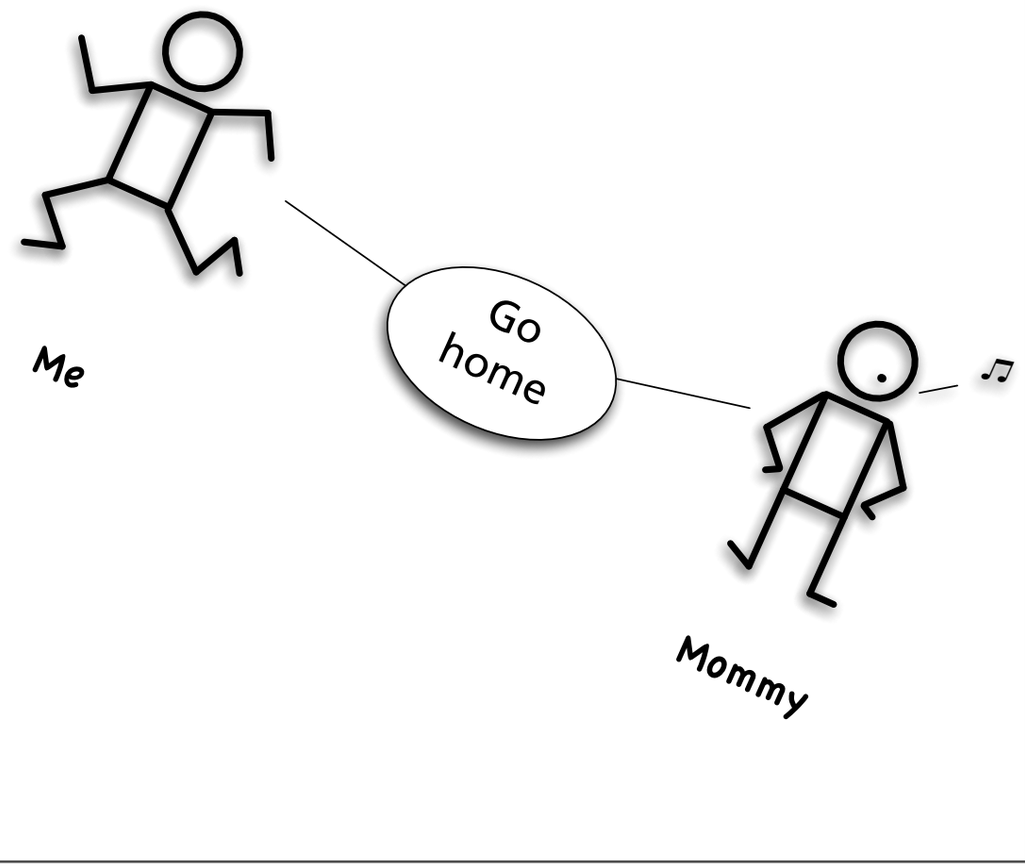
In Scratch: Was gibt es für Sprites? Was müssen diese “können”? Auf welche Events reagieren sie wie?

Implementierung

Die Umsetzung der Anforderungen und des Designs auf konkreten Programmcode.

Testen





Automatisierte Tests (kommt später im Studium), Use Cases durchspielen. Ausnahmefälle überprüfen (z.B. Benutzer gibt ungültige Werte ein, Überläufe, Divisionen durch 0, etc.). Ergebnisse protokollieren.

Aufgabe: Ein erstes Projekt mit Scratch

Programmiert ein Spiel in Scratch!

1. Zweierteams bilden.
2. Zwei Teams erarbeiten gemeinsam Use Cases und ein grobes Design.
3. Implementierung im Team!
4. Kurzpräsentation morgen früh im Plenum.

Mögliche Spiele: Space Invaders, Jump’n’Run, etc.

Relevante Punkte im Use Case Template: Goal in Context, Preconditions, Trigger, Main Success Scenario.

Zum Design gehört: Was für Sprites brauchen wir? Was müssen diese können? Auf was für Events sollen diese unter welchen Bedingungen reagieren?

Umfang der Kurzpräsentation: 2-4 Folien mit Powerpoint oder Keynote (empfohlen), maximal 5 min pro Gruppe. Inhalt: Beispiel-Use-Case, Übersicht über den Aufbau der Applikation, Kurzdemo.