Übung 08

Softwarequalität, -evolution und -wartung

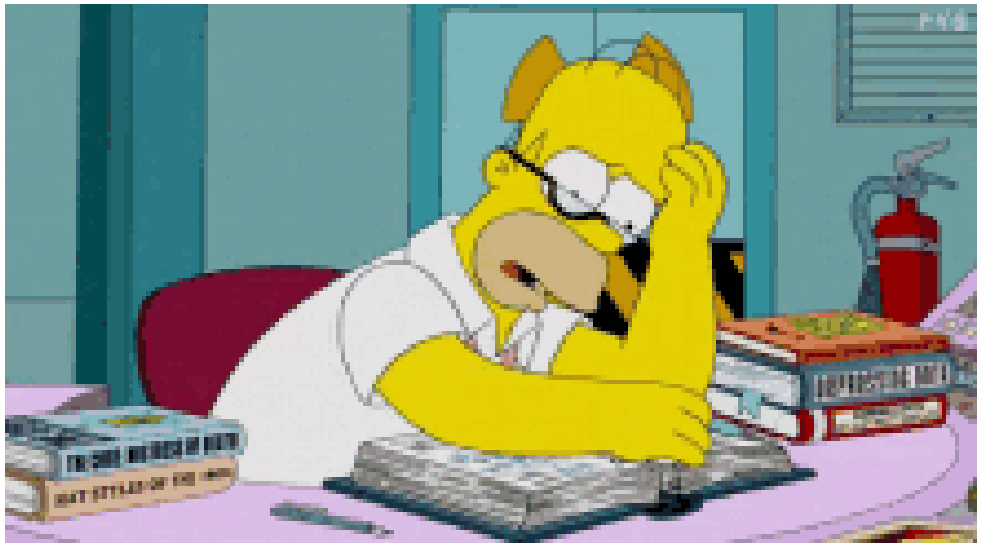
0. Organisatorisches

- Nur noch 2x Übung
  - 20.01.2017

    → Übung 09: Softwareentwicklungsprozesse
  - 27.01.2017

    → Übung 10: Projektmanagement
- **Gastvortrag**: 26.01.2017
- **Klausur**: Do. 09.02.2017, 13.00 Uhr,

  SR 015, B11



http://http://giphy.com/gifs/IPbS5R4fSUI5S

01. Softwarequalität: Softwaremetriken

- Metriken oft nicht sinnvoll zur Messung von Softwarequalität
- Metriken müssen stets auf das Einsatzgebiet angepasst werden, keine universellen Standards

- Lines of Code (LoC)
- Bugs per line of code
- Comment density
- Zyklomatische Komplexität (nach McCabe)
- Halstead complexity measures
- Program execution time
- Test Coverage

- Number of classes and interfaces
- Abstractness
- Entwicklungszeit
- Schulungsaufwand
- Kundenzufriedenheit
- ...

02. Softwareevoluation: Lehman‘s Laws

Gesetze bzw. Hypothesen, die beschreiben, wie und warum ein großes Softwaresystem sich weiterentwickelt.

Name of Law	Law
Continuing Change	Ein System muss kontinuierlich angepasst werden, sonst sinkt die Zufriedenheit mit der Zeit
Increasing Complexity	Die Komplexität des Systems steigt stetig, es sei denn das System wird in Stand gehalten (Maintainance) oder an der Reduzierung dessen gearbeitet
Self Regulation	Systementwicklungsprozesse sind selbstregulierend mit der Verwaltung von Produkt- und Prozessmaßen
Conservation of Organisational Stability	Die durchschnittliche „effective global activity rate“ in einem entwickelnden System bleibt über die Lebensdauer des Produktes unveränderlich
Conservation of Familiarity	Um eine zufriedenstellende Evolution zu erreichen, müssen alle Beteiligten (z.B. Entwickler, Vertriebsmitarbeiter und Kunden) ihre Verhaltensweisen beibehalten
Continuing Growth	Die Funktionalität eines System muss konstant erweitert werden, um die Zufriedenheit zu gewährleisten
Declining Quality	Die Qualität eines Systems wird sinken, sofern keine Adaption an veränderte Umweltfaktoren geschieht
Feedback System	Evolutionsprozesse Stellen ein mehrstufiges Multi-Loop-, Multi-Agent-Feedback-System dar

02. Softwareevoluation: Refactoring vs. Reengineering

Reengineering

- Rekonstruktion eines Systems, um Wartbarkeit und Verständlichkeit zu verbessern
- Portierung auf andere Plattformen
- Dabei können alte Schnittstellen aufgelöst und neu strukturiert werden

Refactoring

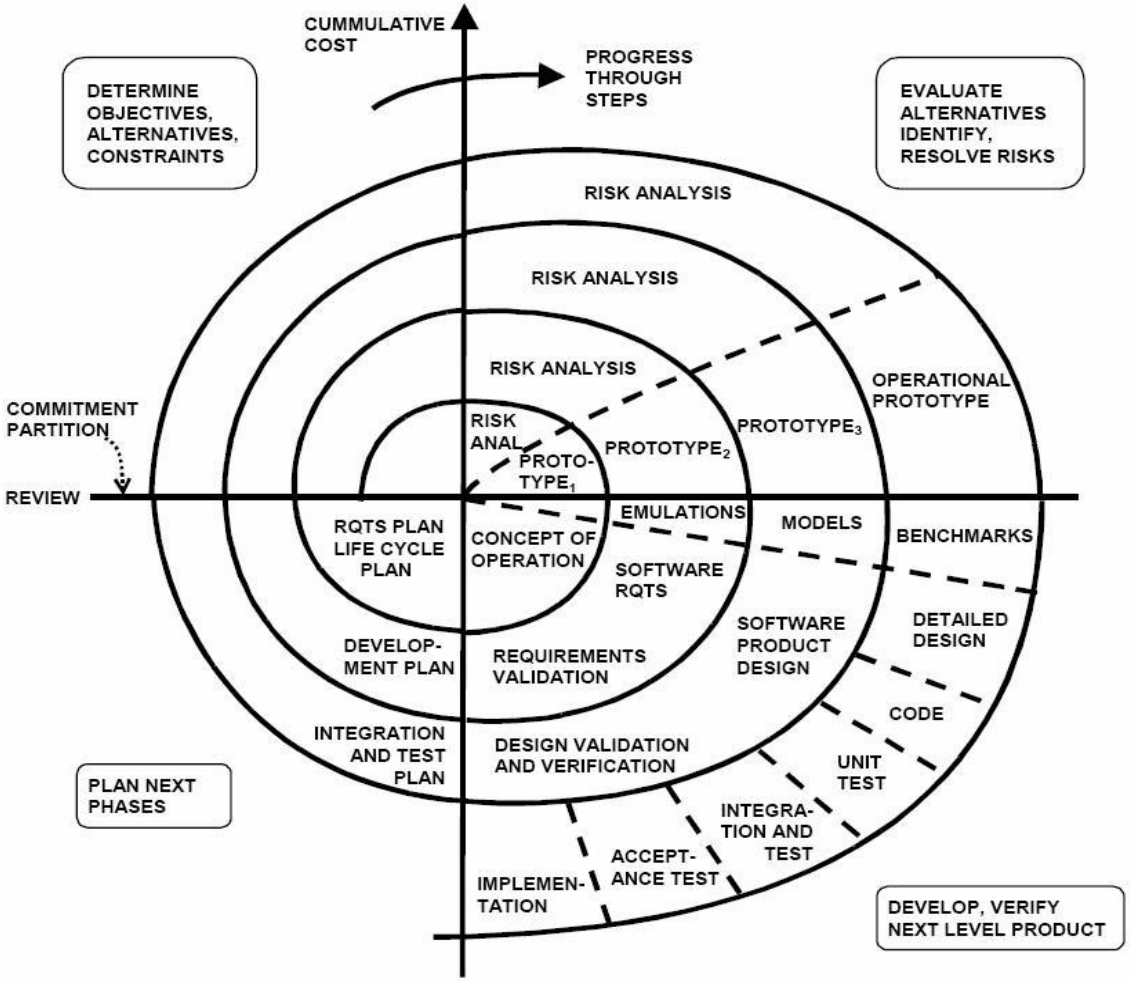
- Veränderung interner Strukturen mit gleichbleibender Funktionalität
- z.B. Auflösen von „God“-Klassen in mehrere Module
- Verbesserung der Lesbarkeit, Verständlichkeit, Wartbarkeit, Erweiterbarkeit

02. Softwareevoluation: Steigende Komplexität

- Ausbau des ursprünglichen Systems → Evolution der Software
- Bug-Fixes, Implementierung neuer Funktionalitäten u.ä. müssen in bestehendes System integriert werden
- Wartbarkeit der Software essentiell
  - Sollte schon in der Design-Phase mit berücksichtigt werden
- Refactoring & Reengineering
- Löschen von nicht mehr verwendeten Code (Feature wurde anderweitig implementiert oder gar gelöscht)
- ...

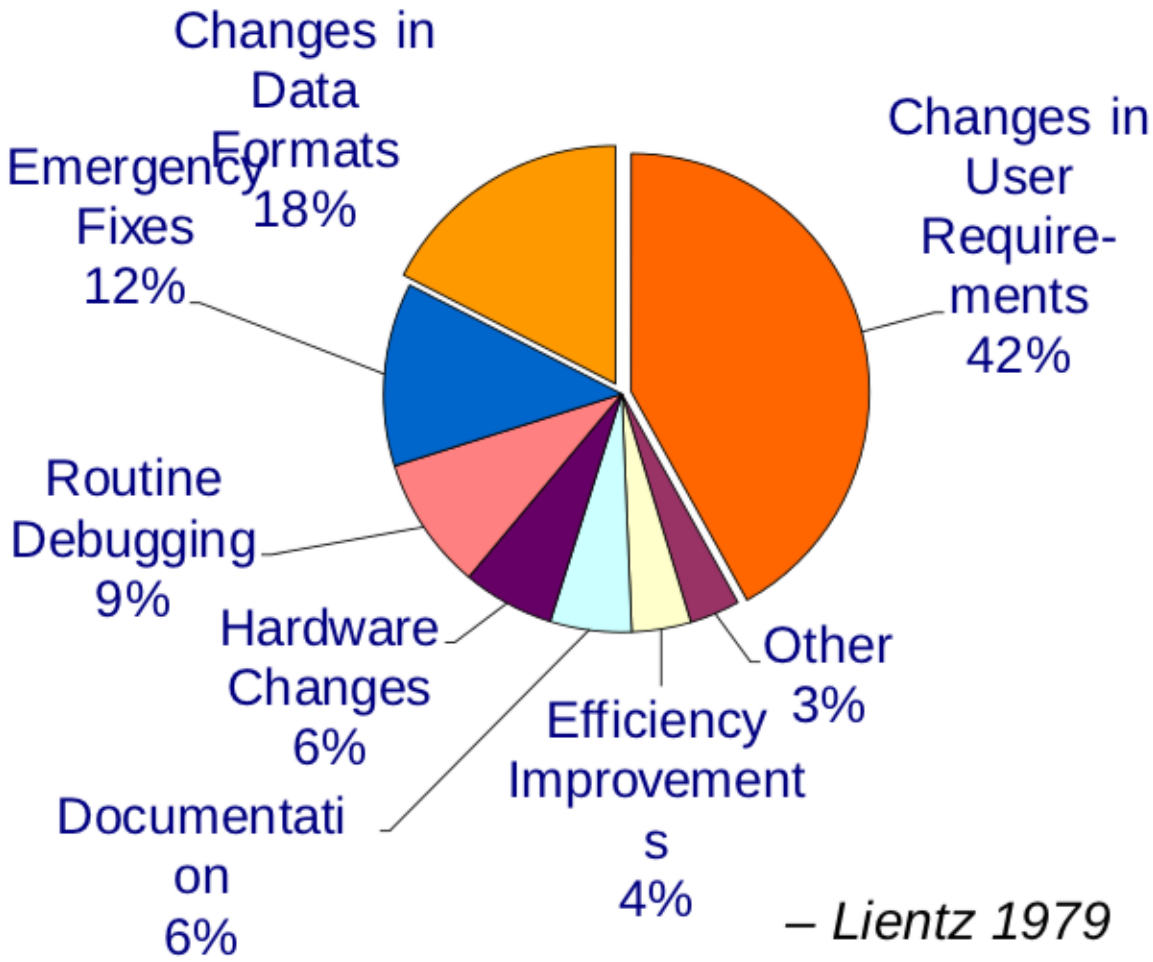
03. Softwarewartung: Wartung

- beinhaltet alle Änderungen des Systems ab erster Installation
- Spiralmodell: eigentlich ab Installation des ersten Prototyps
- umfasst:
  - Konfiguration und Versionsverwaltung
  - Reengineering mit Redesign und Refactoring
  - Aktualisierung aller Dokumente aus Analyse- und Designphase
  - Aktualisierung der gesamten Dokumentation



03. Softwarewartung: Wartung

- Softwarewartung macht bis zu 80% der Projektkosten aus
- Wie hoch ist z.B. Anteil für:
  - echte Fehlerbehebung
  - neue user-requirements
  - Folgeänderungen
- Kosten können dem Kunden in Rechnung gestellt werden



03. Softwarewartung: Wartungsarten

- **Korrektive Wartung:**
  - Auffinden und Korrigieren von Fehlern der Software
- **Adaptive Wartung:**
  - Anpassung der Software an neue Systemumgebung, Hardware, Änderungen von Standards, ...
- **Präventive Wartung:**
  - Vermeidung von potntiellen Problemen
- **Perfektionierende Wartung:**
  - Reengineering mit Redesign und Refactoring
  - Optimierung der Performanz
  - Verbesserung der Wartbarkeit

03. Softwarewartung: Gutes Design vs. Refactoring

- sehr schwer auf Anhieb einen korrekten Systementwurf zu erstellen
  - unbekannte Problemdomäne
  - unverständliche Anforderungen
  - unvorhersehbar wie sich System weiterentwickeln wird
- Software Evolution ist nur bis zu gewissem Grad zu berücksichtigen
  - keine zu frühen Optimierungen
  - Abstraktion ist nicht immer sinnvoll
- gutes Design nicht zwingend gleichbedeutend mit gutem Code

- inkrementelles Refactoring
- TDD: Refactoring als fester Bestandteil des Entwicklungsprozesses

http://www.hwk-chemnitz.de/fileadmin/user_upload/ausbildung/Pruefungen/Gesellenpruefungen/Fragezeichen.JPG