Probeklausur

Lenz Belzner

January 26, 2015

Definieren Sie *Software Engineering* in Abgrenzung zu *Individual Programming*.

- Ingenieursdisziplin
- professionelle Softwareentwicklung
- alle Aspekte der Softwareproduktion
  - Requirements Engineering
  - Design
  - Programmierung
  - Validierung und Verifikation
  - Operations und Wartung

Nennen Sie drei typische Qualitätskriterien für Software je mit kurzer Beschreibung. Warum ist Softwarequalität so schwierig zu bestimmen?

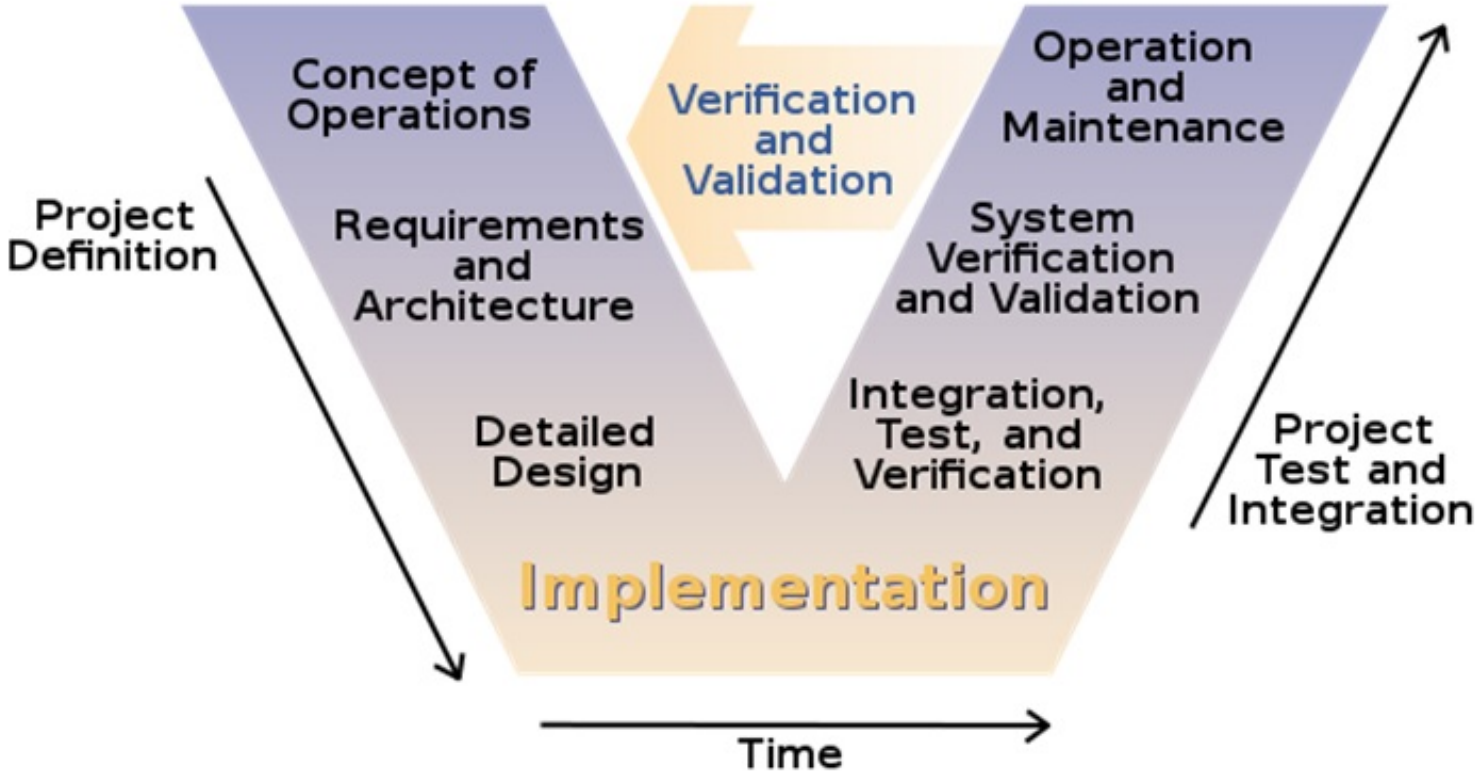
- Wartbarkeit: Flexibel bzgl. Änderung der Anforderungen
- Zuverlässigkeit: Robust bei fehlerhafter Eingabe/Angriffen
- Effizienz: Sparsamer Ressourcenbedarf (Zeit, Speicher, etc...)
- Kriterien sind nichtfunktionale Anforderungen
  - nur indirekt messbar

Was ist der zentrale Unterschied zwischen einem agilen und einem document-driven Prozess? Nennen Sie jeweils 2 Vor- und 2 Nachteile?

- Dokumenterstellung vs. direkte persönliche Kommunikation
- Dokumentenbasiert
  - Gut: Nachvollziehbar, beständige Dokumentation Wissen
  - Schlecht: Overhead, unflexibel/langsam bei Änderung
- Agil
  - Gut: Schnell/flexibel, direkter Kundenbezug
  - Schlecht: Degradierendes Design, implizite Dokumentation von Wissen

Beschreiben Sie kurz die prinzipielle Idee der Systementwicklung (inklusive der Aktivitäten) im V-Modell.

- Gegenüberstellung von Spezifikation und Realisierung in V-Form
- Jeder Realisierungsmeilenstein wird gegenüber einem Spezifikationsmeilenstein überprüft



Für welche Art von Software würden Sie RUP als Entwicklungsprozess vorschlagen? Warum?

- Inception $\rightarrow$ Erfassung eines Geschäftsprozesses
- Software zur Implementierung von Geschäftsprozessen

Beschreiben Sie das Architectural Pattern *SOA*. Für welchen Anwendungsbereich eignen sich SOA-Architekturen? Nennen Sie einen Vor- und einen Nachteil der Architektur.

- Komposition abgeschlossener, wiederverwenbarer Dienste über Netzwerk
- Integration von Enterprise-Systemen
- Vorteil: Ermöglicht komplexe Systeme durch Integration
- Nachteil: Hoher Anfangsaufwand

Was versteht man unter einem *Design Pattern*? Geben Sie ein Beispiel für solches Pattern an.

- beschreibt eine (wohlbekannte) Lösung zu einem typischen Designproblem
- Observer-Pattern
  - erlaubt, verschiedene nicht unbedingt vorab bekannte Teile der Anwendung über Änderungen zu informieren

Wofür steht das Akronym *KISS*? Welche Eigenschaften hat eine *KISS-Lösung*? Welche 3 Vorteile hat es, dieses Prinzip anzuwenden?

- Keep it Simple, Stupid
- vermeidet unnötige Komplexität im Design
- ist möglichst minimal
- schnell fertig, leicht kommunizierbar, einfacher zu testen

Welche Design Pattern finden Sie in folgendem Code? Wo? Beschreiben Sie kurz das Pattern.

- Klasse Transformer: Command Pattern
  - Kapselung einer Aktion als Objekt, um diese z.B. abspeichern zu können und eine History zu unterstützen
- Klasse XMLElement: Composite Pattern
  - Abbildung einer Teil-Ganzes-Struktur mit der Möglichkeit, gewisse Methoden uniform anwenden zu können

Erläutern Sie den Begriff *Refactoring*. Warum wird Refactoring angewendet?

- Änderung der internen Struktur
- Verhalten bleibt gleich
- Verbesserung der Struktur (Lesbarkeit, Wartbarkeit, ...)

Sie wenden das Refactoring *Extract Method* auf die drei Zeilen zwischen //from und //to in folgendem Programm an. Wie sieht die Signatur der extrahierten Methode aus?

```
public void doSomething() {

        int x= 0;
        int y= 2;
        int z= 0;

        // from
        for (int i= x; x < y; i++) {
                z+= i;
        }
        // to

        System.out.println(z);
}
```

- (*visibility*) int extracted(int x, int y, int z)

Was versteht man unter Validation? Was versteht man unter Verification?

- Validation: Bauen wir das richtige Produkt? Sind die Anforderungen korrekt?
- Verification: Bauen wir das Produkt richtig? Sind die Anforderungen erfüllt?

Was versteht man unter Testing? Welche beiden Ziele kann Testing verfolgen? Kann man durch Testing ein Programm für fehlerfrei erklären? Wenn ja, wie? Wenn nein, warum nicht?

- Ausführung eines Programms mit künstlichen (Test-)Daten
- Ziele
  - Überprüfen der Spezifikationserfüllung (Validation Testing)
  - Fehler finden (Defect Testing)
- Testen kann nur Fehler aufzeigen
  - sehr große Zustandsräume, komplette Testabdeckung nicht realisierbar

Erläutern Sie die Begriffe Unit, Component, und System Testing.

- Unit: Test einzelner kleinster Einheiten im Code
- Component: Test der Zusammensetzung von Units, Fokus öffentliche Schnittstellen
- System: Test des integrierten Systems, Fokus Interaktion von Komponenten

Nennen Sie je zwei Vorteile von Code-Inspektionen und Testing.

- Inspektion
  - Möglich bei nicht-vollständigen Systemen
  - Zusätzliche Code-Eigenschaften überprüfbar (z.B. Codestyles)
- Testing
  - Deckt unerwartete Interaktionen zwischen verschiedenen Programmteilen auf
  - Erkennen von Performance-Problemen