6 Implementieren

6.1 Grundlagen

Tätigkeiten

1) **Implementieren**
   - Erstellen von Programmcode (Sourcen) und ausführbaren Code (Binaries)
   - Scripte zur Installation, Erzeugen von Datenbank-Tabellen, Laden von Altdaten, ...
2) **Dokumentieren**
   - der erstellten Ergebnisse (in separaten Dokumenten)
   - des Source Codes (inline)
3) **Integrieren**
   - Integration der Sourcen zu Komponenten bzw. zu Subsystemen
   - Test der einzelnen Komponenten bzw. Subsysteme
4) **Installieren(Deployment)**
   - Verteilung auf verschiedene Rechner/ Systemsoftware
   - Konfigurierung Systemsoftware (z.B. IDE, Application Server, Web Server, DBMS etc.), ...

Vorbereitende Aufgaben

- vor Beginn der Implementierung müssen einige Querschnittskonzepte bereits geklärt sein

  1) **Implementierung**
     1.1. Programmierrichtlinien 1.2 Entwicklungsumgebung
  2) **Integration**
     2.1. Integrationsstrategie
     2.2. Versionsverwaltung
     2.3. **Build-Management (Deployment)**

- Querschnittskonzepte werden in Projekt-Wiki (Entwicklerhandbuch) dokumentiert (strukturelle und konzeptionelle Sicht auf Implementierung)

2.1. Integrationsstrategie

a. Reihenfolge der Implementierung der einzelnen Komponenten und Zusammenbau (= Integration) zu Subsystemen
b. Zeitliche Häufigkeit der Integration: **Continuous Integration** (CI)

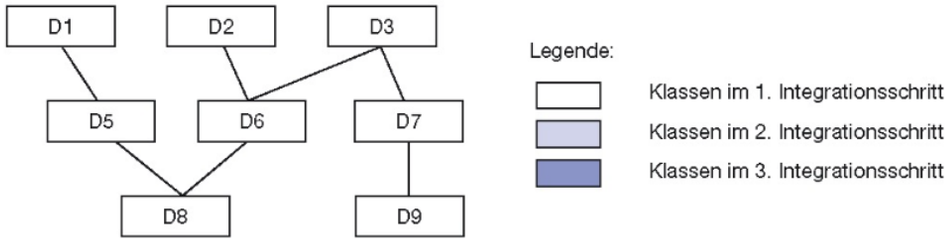
Voraussetzungen:

- vor einem Integrationsschritt:
  - jede einzelne **Komponente** ist bereits **getestet**
- nach jedem Integrationsschritt:
  - ein **Integrationstest** wird durchgeführt: stellt sicher, dass die Integration erfolgreich war

Reihenfolge der Integration

Big-Bang-Integration

- alle Teile werden zum selben Zeitpunkt zusammengefügt
- Nachteil: großes Risiko, deshalb nur noch selten eingesetzt, aber manchmal unumgänglich



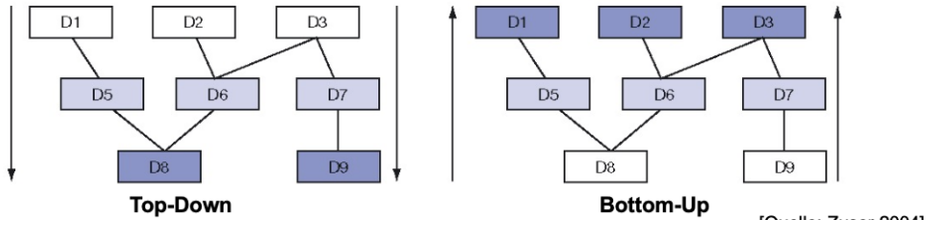
- Manchmal unumgänglich, falls keine sinnvollen Teilsysteme existieren oder die viel zu aufwändig wären, da zusätzliche Hilfsklassen notwendig wären

Top-Down-Integration

- von Benutzerschnittstelle nach und nach bis zur techn. Plattform
- Vorteil: Benutzerschnittstelle schnell verfügbar
- Nachteil: meist komplexe techn. Anbindung von systemnahen Teilen erst spät; Tests nur mit Hilfsklassen möglich

Bottom-Up-Integration

- von systemnaher Ebene bis zur Benutzerschnittstelle
- Vorteile/ Nachteile umgekehrt zu Top-Down

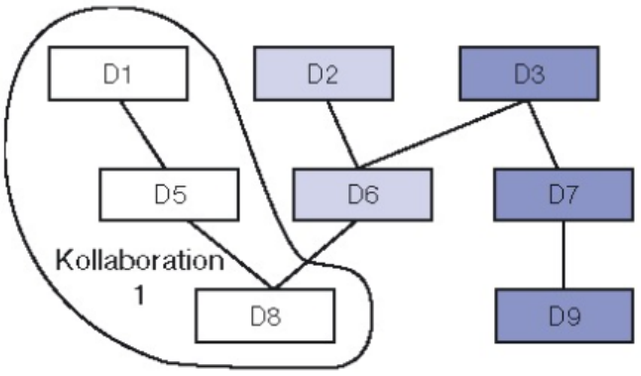


- Beide benötigen Hilfsklassen beim Testen. Wenn bei Platform bzw. technische Umsetzung unerfahren, dann Bottom-Up-Integration verwenden

Build-Integration (vertikal)

- vertikal **integrieren und einige Anwendungsfälle voll funktionsfähig umsetzen**

- Integrationsstrategie bei inkrementeller Entwicklung
- Vorteil: fachliche und technische Risiken früh erkannt
- Nachteil: sehr früh müssen auf allen Ebenen Basisdienste implementiert werden; ggf. hoher Änderungsaufwand
- Ist wie bei UP. Auch die empfehlenswerte Strategie



2.1 Zeitl. Häufigkeit: Continuous Integration

- Hat sich etabliert

Continuous Integration (CI)

- Idee: fortlaufende Integration
  - **daily build**, **multiple builds** per day
  - CI ist Voraussetzung für **Frequent Deployment (DevOps)**

- Vorteile: **Risikoreduzierung**
  - **weniger Integrationsprobleme**, weil das **Inkrement** zwischen zwei aufeinanderfolgenden **Iterationen** klein ist
  - entsprechend treten weniger Fehler (bugs) auf, die somit schneller lokalisiert (und beseitigt) werden können

- **Voraussetzungen: hoher Automatisierungsgrad**
  - Source code control system
  - Build/ deployment tools
  - automatisiertes Testen

2.2 Versionsverwaltung

- **Versionsverwaltung (Konfigurationsmanagement)**
  - Versionsstände von Artefakten (Sourcen, Dokumenten, Modellen etc.) verwalten und bereitstellen
  - Repository (DB, Dateisystem) speichert Versionsstände persistent

- Vorteile (u.a.):
  - Historie verwalten (Änderungen nachvollziehen)
  - Rollback von Veränderungen (alte Versionen wiederherstellen)
  - Ergebnisse mehrerer Personen integrieren
  - Release Management (passende Versionen mehrerer Artefakte zu einem Stand zusammenfassen)

→ Versionsverwaltung bereits für frühe Arbeitsschritte (Anforderungen, Analyse, Design) nutzen

→Versionsverwaltung ist unerlässlich für die Entwicklung und Pflege großer, langlebiger Software-Systeme!

2.3 Build-Management (Deployment) – Ant

- Software-Architekturen werden stetig komplexer
  - verteilte Multi-Tier-Systeme auf heterogenen Plattformen
  - Einsatz diverser Systemsoftware-Bausteine (DBMS, Web Server, Application Server, Treiber etc.)
  - Wiederverwendung von Komponenten (Klassenbibliotheken, Frameworks, fachl./ techn. Services etc.)
- Verringerung des Implementierungsaufwands, aber Verlagerung der Komplexität von Programmierung zur Konfigurierung
- Deployment (Verteilung bzw. Auslieferung) umfasst
  - Konfiguration der SW-Bausteine und verwendeten Systemsoftware (passende System-Versionen beachten!)
  - Verteilung der Software auf die verschiedenen Systemteile
  - Compilieren, Installieren, Einspielen von Daten, ...
    → Ergebnis ist lauffähiges System (bzw. Build)

- Einsatz von Werkzeugen (Build-Management-Tool) zum Deployment unabdingbar
  - Durchführung der Schritte muss automatisiert werden
  - make, Shell-Skripte; aktuelles Standardwerkzeug: ant von Apache (alternativ: Maven, Gradle, ...)

6.2 Ergebnisse

1) **Integrationsplanung**
   - Beschreibung des Implementierungsprozesses
   - Reihenfolgen für Implementierung und Integrationstest
2) **Implementierungsmodell**
   - Beschreibung aller Software-Artefakte des Systems
     - Packages mit Sourcen
     - Dokumentation der Softwarekomponenten
     - Entwicklungsumgebung
3) **Build-Management(Deployment)**
   - Software/ Systeme installieren und konfigurieren

6.3 Vorgehen

Schritte in der Implementierung

1) **Querschnittskonzepte bereitstellen**
   - Programmierrichtlinien, Entwicklungsumgebung, Versionsverwaltung etc.
2) **Infrastrukturbereitstellen**
   - Einrichten der Entwicklungs-, Test-, Integrations-, Produktivumgebungen
   - Installation und Konfiguration der Systemsoftware (z.B. DBMS, Web Server, Application Server, ...)
3) **Implementierungs-und Integrationsplanung**
4) **Implementierung (von Architektur, Subsystemen und Klassen) und Test der Software-Artefakte**
5) **Deployment der Software-Artefakte**
6) **Sukzessive Integration**