

Teil V

Implementierungsaspekte

- defensives Programmieren, da keine Integrationstests
- wiederverwendbar machen:
 - entferne anwendungsspezifische Methoden
 - generalisiere Namen
 - füge Methoden hinzu, um Funktionalität zu vervollständigen
 - führe konsistente Ausnahmebehandlung ein
 - füge Möglichkeiten hinzu, die Komponenten an verschiedene Benutzer anzupassen
 - binde benötigte Komponenten ein, um die Unabhängigkeit zu erhöhen

- Anpassen einer Komponente:
 - durch Parametrisieren und Verbinden mit anderen Komponenten
 - durch Ableiten
- Arten:
 - Implementierungsvererbung
 - Schnittstellenvererbung
- Zusage der Austauschbarkeit
- Mehrfachvererbung:
 - Schnittstellenvererbung: kein Problem
 - Implementierungsvererbung: problematisch

- Basisklasse wird geändert, sind erbende Klassen immer noch funktionsfähig?
- unvorhergesehene Aufrufgraphen → unerwarteter Wiedereintritt
- notwendig ist Schnittstellenvertrag zwischen Klasse und erbenden Klassen
- Lösungen:
 - Schutz (public, protected, private, finale, override)
 - disjunkte Gruppen von Methoden und Variablen
 - alle Instanzvariablen sind privat und die erbende Klasse fügt keine hinzu
 - Ableiten der Klasse verbieten

Aufrufe zwischen Komponenten in verteilten Systemen

- Idee des lokalen Aufrufes bleibt erhalten
- Generierung von Stubs
- Aktionen des Aufrufers bei Aufruf:
 - ① Kontrolle geht an Stub
 - ② Marshalling/Serialisierung
 - ③ Übertragung der Parameter
 - ④ Ausführen auf dem Ziel
 - ⑤ Übertragung des Ergebnisses/Ausnahme
 - ⑥ Unmarshalling
 - ⑦ Rückgabe an den Aufrufer
- Optimierung für lokalen Fall

- Wiedereintritt als Problem
- Vorkommen: Callbacks (von der unteren zur oberen Schicht) oder Multi-threading
- Problem: welche Funktionen dürfen benutzt werden
- Beispiel: Unix-Signal-Handler
- nicht mit Vor- und Nachbedingungen ausdrückbar

Garlan u. a. (1995): Komposition eines Case-Tools aus

- einer objektorientierten Datenbank
- Toolkit zur Konstruktion graphischer Benutzeroberflächen
- event-basiertem Tool-Integrations-Mechanismus
- RPC-Mechanismus

Alle Komponenten in C oder C++ geschrieben.

Schätzung:

- Dauer: 6 Monate
- Aufwand: 12 PM

Schätzung:

- Dauer: 6 Monate
- Aufwand: 12 PM

Tatsächlich:

- Dauer: 2 Jahre
- Aufwand: 60 PM für ersten Prototyp

Schätzung:

- Dauer: 6 Monate
- Aufwand: 12 PM

Tatsächlich:

- Dauer: 2 Jahre
- Aufwand: 60 PM für ersten Prototyp

Ergebnis:

- sehr großes System
- träge Performanz
- viele Anpassungen für die Integration notwendig
- existierende Funktionalität musste neu implementiert werden, weil sie nicht exakt den Anforderungen entsprach

Definition

Architektur-Mismatch: inkompatible Annahmen von wiederzuverwendenden Komponenten über das Systems, in dem sie eingesetzt werden sollen.

Meist spät entdeckt, weil die Annahmen in aller Regel nur implizit sind.

Komponenten betreffend

Annahmen von Komponenten über

- ihre Infrastruktur, die zur Verfügung gestellt werden soll bzw. vorausgesetzt wird
 - Komponenten stellten vieles zur Verfügung, was gar nicht gebraucht wurde
- exzessiver Code

Annahmen von Komponenten über

- ihre Infrastruktur, die zur Verfügung gestellt werden soll bzw. vorausgesetzt wird
 - Komponenten stellten vieles zur Verfügung, was gar nicht gebraucht wurde
 - exzessiver Code
- das Kontrollmodell: wer steuert den Kontrollfluss
 - jede Komponente nahm an, dass sie die Hauptkontrollschleife darstelle
 - aufwändige Restrukturierung notwendig

Komponenten betreffend

Annahmen von Komponenten über

- ihre Infrastruktur, die zur Verfügung gestellt werden soll bzw. vorausgesetzt wird
 - Komponenten stellten vieles zur Verfügung, was gar nicht gebraucht wurde
 - exzessiver Code
- das Kontrollmodell: wer steuert den Kontrollfluss
 - jede Komponente nahm an, dass sie die Hauptkontrollschleife darstelle
 - aufwändige Restrukturierung notwendig
- das Datenmodell: Art und Weise, wie die Umgebung Daten manipuliert, die von der Komponente verwaltet werden
 - hierarchische Datenstruktur erlaubte Änderung der Teile nur über das Ganze
 - für Anwendung zu unflexibel
 - teilweise Neuimplementierung

Annahmen von Konnektoren über

- Protokoll: Interaktionsmuster
 - Semantik des synchronen Aufrufs passte nicht
 - Ausweichung auf RPC des Betriebssystems hierfür
- Datenmodell: Art der Daten, die kommuniziert werden
 - RPC des Betriebssystems nahm an, C-Datenstrukturen zu transportieren
 - wiederzuverwendendes Event-Broadcast-System nahm an, ASCII zu transportieren
 - Konvertierungsroutinen wurden notwendig

Annahmen über Architekturkonfiguration

- Topologie der Systemkommunikation
 - Datenbank nahm an, dass verbundene Tools nicht kooperieren wollen und blockierte sie, um Sequenzialisierung zu garantieren
 - Tools mussten aber kooperieren
 - eigener Transaktionsmonitor musste implementiert werden
- An- oder Abwesenheit bestimmter Komponenten und Konnektoren

Konstruktionsprozess betreffend

Annahmen über Konstruktionsprozess (wie Komponenten/Konnektoren aus generischen Einheiten erstellt werden – sowohl zur Übersetzungs- als auch zur Laufzeit)

Konstruktionsprozess betreffend

Annahmen über Konstruktionsprozess (wie Komponenten/Konnektoren aus generischen Einheiten erstellt werden – sowohl zur Übersetzungs- als auch zur Laufzeit)

Beispiele:

- Datenbank → Schema muss festgelegt werden
- Event-System → Menge der Ereignisse und Registration

Konstruktionsprozess betreffend

Annahmen über Konstruktionsprozess (wie Komponenten/Konnektoren aus generischen Einheiten erstellt werden – sowohl zur Übersetzungs- als auch zur Laufzeit)

Beispiele:

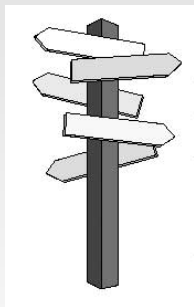
- Datenbank → Schema muss festgelegt werden
- Event-System → Menge der Ereignisse und Registration

Annahmen über die Reihenfolge, in der Teile erstellt und kombiniert werden.

- nicht zueinander passende Annahmen über den Konstruktionsprozess
→ Umwege machten Konstruktionsprozess aufwändig und kompliziert

2 Software-Architektur

- Was ist Software-Architektur?
- Zusammenfassung aus dem Software-Projekt: Hofmeister-Methode und -Blickwinkel
- Qualität von Software-Architekturen
- Taktiken



Teil I

Software-Architektur: kurze Zusammenfassung aus dem Software-Projekt

Was ist Architektur?

Architecture *is the human organization of empty space using raw material.*

Richard Hooker, 1996.

Was ist Architektur?

Architecture *is the human organization of empty space using raw material.*

Richard Hooker, 1996.

Definition

Software-Architektur ist die grundlegende Organisation eines Systems, verkörpert (IEEE P1471 2002)

- in seinen Komponenten,
- deren Beziehungen untereinander und zur Umgebung
- und die Prinzipien, die den Entwurf und die Evolution leiten.

Was ist Architektur?

Architecture *is the human organization of empty space using raw material.*

Richard Hooker, 1996.

Definition

Software-Architektur ist die grundlegende Organisation eines Systems, verkörpert (IEEE P1471 2002)

- in seinen Komponenten,
- deren Beziehungen untereinander und zur Umgebung
- und die Prinzipien, die den Entwurf und die Evolution leiten.

Weitere über 100 Definitionen unter
www.sei.cmu.edu/architecture/community_definitions.html.

Bedeutung von Software-Architektur

- Kommunikation zwischen allen Interessenten
 - hoher Abstraktionsgrad, der von vielen verstanden werden kann
- Frühe Entwurfsentscheidungen
 - nachhaltige Auswirkungen
 - frühzeitige Analyse
- Transferierbare Abstraktion des Systems
 - Beherrschung der Komplexität
 - Aufgabenverteilung
 - eigenständig wiederverwendbar
 - unterstützt Wiederverwendung im großen Stil (Software-Produktlinien)

Definition

Architektursicht (View): Repräsentation eines ganzen Systems aus der Perspektive einer kohärenten Menge von Anliegen (IEEE P1471 2002).

Architektursichten und -blickwinkel

Definition

Architektursicht (View): Repräsentation eines ganzen Systems aus der Perspektive einer kohärenten Menge von Anliegen (IEEE P1471 2002).

Definition

Architekturblickwinkel (Viewpoint): Spezifikation der Regeln und Konventionen, um eine Architektursicht zu konstruieren und zu benutzen (IEEE P1471 2002).

Ein Blickwinkel ist ein Muster oder eine Vorlage, von der aus individuelle Sichten entwickelt werden können, durch Festlegung von

- Zweck,
- adressierte Betrachter,
- und Techniken für Erstellung, Gebrauch und Analyse.

Architektursichten und -blickwinkel

Definition

Architektursicht (View): Repräsentation eines ganzen Systems aus der Perspektive einer kohärenten Menge von Anliegen (IEEE P1471 2002).

Definition

Architekturblickwinkel (Viewpoint): Spezifikation der Regeln und Konventionen, um eine Architektursicht zu konstruieren und zu benutzen (IEEE P1471 2002).

Ein Blickwinkel ist ein Muster oder eine Vorlage, von der aus individuelle Sichten entwickelt werden können, durch Festlegung von

- Zweck,
- adressierte Betrachter,
- und Techniken für Erstellung, Gebrauch und Analyse.

Unterschiedliche Sichten helfen der Strukturierung:
Separation of Concerns.

- **Konzeptioneller Blickwinkel:** beschreibt logische Struktur des Systems; abstrahiert weitgehend von technologischen Details
- **Modulblickwinkel:** beschreibt die statische logische Struktur des Systems
- **Ausführungsblickwinkel:** beschreibt die dynamische logische Struktur des Systems
- **Code-Blickwinkel:** beschreibt die „anfassbaren“ Elemente des Systems (Quelldateien, Bibliotheken, ausführbare Dateien etc.)