Aufgabe 1: Definition der Software-Architektur (10 Punkte)

Unter der Software Architektur eines Programms oder Computersystems versteht man die Struktur oder die Strukturen des Systems, die Softwarekomponenten, die von außen sichtbaren Eigenschaften dieser Komponenten und die Beziehungen zwischen ihnen. Die Architektur eines Softwaresystems definiert das System mittels Komponenten (computational components) und Interaktionen zwischen diesen Komponenten. Auf der Architekturebene sind bestimmte Eigenschaften relevant.

- Was gehört zu den von außen sichtbaren Eigenschaften? (Kennzeichnung mit S)
- Was gehört zu den Komponenten? (Kennzeichnung mit K)
- Was gehört zu den Interaktionen? (Kennzeichnung mit I)
- Welche Eigenschaften sind auf der Architekturebene relevant? (Kennzeichnung mit A)
- Was gehört nicht zu einer Software-Architektur? (Kennzeichnung mit N)

Betrachten Sie dazu die folgende Liste und kennzeichnen Sie die Listenelemente entsprechend den obigen Fragen.

[ ] Kapazität,

[ ] Prozeduraufrufe,

[ ] bereitgestellte Dienste (provided services),

[ ] Klienten und Server,

[ ] Kompatibilität der Komponenten,

[ ] Performance-Eigenschaften,

[ ] asynchrone Ereignisbehandlung (event multicast),

[ ] Datenbankzugangsprotokolle,

[ ] Zugriff auf gemeinsame Variablen,

[ ] Filter,

[ ] Durchsatz,

[ ] gemeinsame Nutzung von Ressourcen,

[ ] Schichten in einem hierarchischen System,

[ ] GOTO,

[ ] Fehlerbehandlung,

[ ] Konsistenz,

[ ] parametrischer Polymorphismus,

[ ] Client-Server-Protokolle,

[ ] offene Rekursion,

[ ] Streambehandlung (piped streams).

Lösung:

[A] Kapazität,

[I] Prozeduraufrufe,

[S] bereitgestellte Dienste (provided services),

[K] Klienten und Server,

[A] Kompatibilität der Komponenten,

[S] Performance-Eigenschaften,

[I] asynchrone Ereignisbehandlung (event multicast),

[I] Datenbankzugangsprotokolle,

[I] Zugriff auf gemeinsame Variablen,

[K] Filter,

[A] Durchsatz,

[S] gemeinsame Nutzung von Ressourcen,

[K] Schichten in einem hierarchischen System,

[N] GOTO,

[S] Fehlerbehandlung,

[A] Konsistenz,

[N] parametrischer Polymorphismus,

[I] Client-Server-Protokolle,

[N] offene Rekursion,

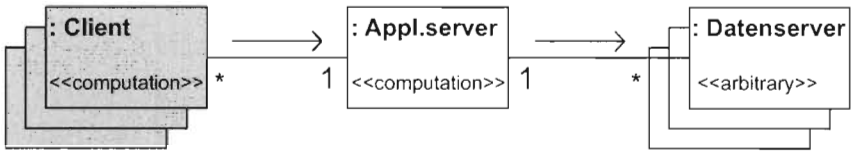
[I] Streambehandlung (piped streams).

Aufgabe 2: Architektur eines konkreten SWS (10 P = 7 + 3 P)

a.) Beschreiben Sie die Drei-Schichten-Architektur (Three Tier Architecture).
Fertigen Sie zur Illustration eine Skizze an. Geben Sie drei Einsatzbereiche der mittleren Schicht an.

Lösung:

Bei der Drei-Schichten-Architektur kommunizieren eine beliebige Anzahl von Clients auf der einen Seite und Datenservern auf der anderen Seite durch die mittlere Schicht, den Applikationsserver (häufig auch als „Prozessschicht" (process management) bezeichnet) miteinander.



Diese mittlere Schicht hat vielfältige Einsatzbereiche:

1. Sie kann einen Übersetzungsdienst bereitstellen, der eine Mainframe-Applikation an eine Client-Server-Umgebung anbindet;
2. sie kann zur Lastverteilung eingesetzt werden, um nur eine gewisse Maximalzahl von Clients auf demselben Server zuzulassen (häufig bei WWW-Servern eingesetzt);
3. sie kann intelligente Agentendienste bereitstellen, indem sie Anfragen eines Clients an verschiedene Server weiterleitet und deren Antworten zusammengefasst als eine Antwort an den Client zurückleitet.

b.) Ist Ihr Muster eine Spezialisierung eines der im Kurs vorgestellten Muster? Wo gibt es Unterschiede, wo Gemeinsamkeiten?

Lösung:

Aus Sicht des Clients ist das Architekturmuster „Drei-Schichten-Architektur" gleich zur Client-Server-Architektur, da vom Client aus nur eine Schicht angesprochen wird und er die dahinterliegende Server-Schicht nicht wahrnimmt. Insgesamt ergibt sich also eine Spezialisierung der Client-Server-Architektur.

Drei-Schichten-Architektur ist eine Spezialisierung der Schichtenarchitektur, da die einzelnen Schichten nur mit den dahinterliegenden Schichten kommunizieren. Allerdings handelt es sich bei der Drei-Schichten-Architektur um genau drei Schichten, und die einzelnen Schichten können bei der Kommunikation nicht übersprungen werden (strenges Schichtenmodell). Mit den anderen Architekturmustern gibt es keine ausgeprägten Gemeinsamkeiten.

Aufgabe 3: Strukturen (10 P = 7 + 3 P)

a.) Der Kurstext nennt eine Reihe von Strukturen zur Beschreibung von Software-Architekturen:

- Conceptual Structure
- Data flow
- Control flow
- Hierarchical structure
- Uses/call structure
- Process structure
- Physical structure

Ordnen Sie diesen Strukturen die folgenden Beispielsysteme zu:

B1: Client-Server-Architektur eines Datenbankservers,

B2: UNIX-Shell mit Kommandozeilentools und Pipe,

B3: Compiler,

B4: Waschmaschinensteuerung,

B5: ein Programmgerüst, z. B. das AWT, das innerhalb eines Programms benutzt wird,

B6: World Wide Web und Internet-Dienste,

B7: Multitasking-Betriebssystem.

Lösung:

- Conceptual Structure: B6: World Wide Web und Internet-Dienste
- Data flow: B2: UNIX-Shell mit Kommandozeilentools und Pipe.
- Control flow: B4:Waschmaschinensteuerung
- Hierarchical structure: B3: Compiler
- Uses/call structure: B5: Ein Programmgerüst, z. B. das AWT, das innerhalb eines Programms benutzt wird.
- Process structure: B7: Multitasking-Betriebssystem
- Physical structure: B1:Client-Server-Architektur eines Datenbankservers

b.) Beschreiben Sie die Uses/call structure.

Lösung:

Benutzungs/Aufrufstruktur (uses/call structure):
Die Komponenten sind Programmmodule, Klassen, Prozeduren. Die Struktur drückt die Beziehung zwischen Moduln aus, die Benutzungsrelation bei Klassen, die Aufrufrelation bei Prozeduren.

Aufgabe 4: Umwandlung von Architekturen (10 P)

Diese Schichtenarchitektur ist aus der vorliegenden CompoundObject-Architektur entstanden.

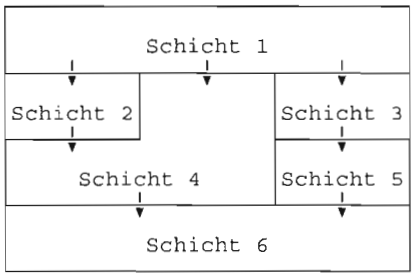


Abbildung 1: Schichtenarchitektur

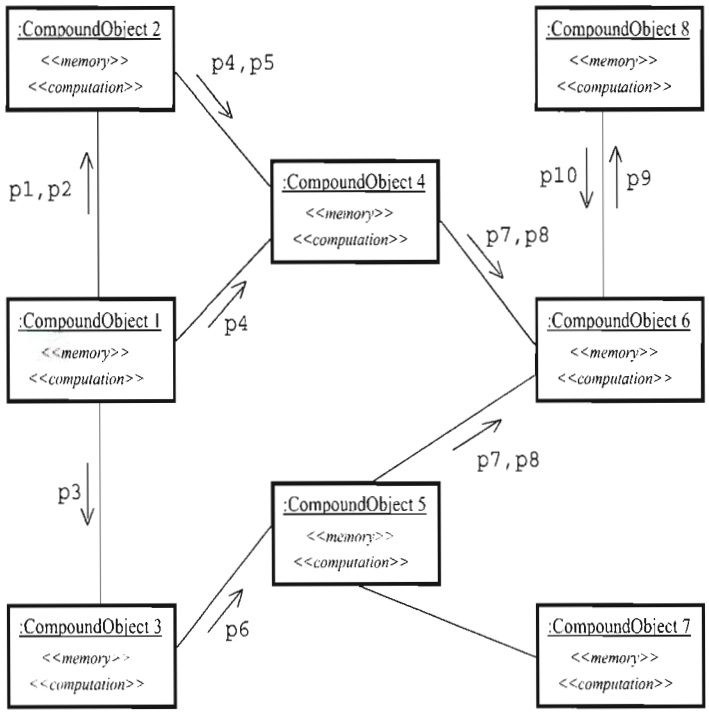


Abbildung 2: CompoundObject-Architektur

Welche Objekte und Prozeduren befinden sich in den Schichten?
Begründen Sie Ihre Aufteilung. Beachten Sie dabei, ob die Aufrufbeziehung hierarchisch oder zyklisch ist.

Lösung:

Aufteilung der Objekte und Prozeduren auf die Schichten:

Schicht	Comp.-Objekte	Prozeduren
1	1	–
2	2	p1,p2
3	3	p3
4	4	p4,p5
5	5,7	p6
6	6,8	p7,p8,p9,p10

Ist die Aufrufbeziehung hierarchisch, so liegt das aufrufende CompoundObject in einer Schicht, die möglichst dicht über derjenigen liegt, die das CompoundObject mit der aufgerufenen Methode enthält.
Ist die Aufrufbeziehung zyklisch, so liegen alle Compound-Objekte des Zyklus in einer Schicht.

Aufgabe 5: Architekturen und Evaluation, Aspekte eines Softwaresystems (10 P = 6 + 4)

a.) Beschreiben Sie die vier im Kurstext angegebenen allgemeinen Regeln, die man für das *Design der Module* jeder Software-Architektur beachten sollte.

Lösung:

1. Die Architektur sollte durch sinnvoll definierte Module ausgezeichnet sein, deren funktionale Verantwortlichkeiten auf den Prinzipien des Information Hiding und Separation of Concerns beruhen.
   Jedes Modul sollte eine wohldefinierte Schnittstelle haben, die änderbare Aspekte vor anderer Software, die seine Möglichkeiten nutzt, kapselt oder versteckt.
2. Die Module sollen die Trennung der Verantwortlichkeit so reflektieren, dass die Entwicklungsteams weitgehend unabhängig voneinander arbeiten können.
3. Zu den informationsverbergenden Modulen sollten diejenigen gehören, die spezielle Eigenschaften der Recheninfrastruktur kapseln, so dass ein Großteil der Software vom Wechsel der Plattform unbeeinflusst bleibt.
4. Module, die Daten produzieren, sollten von Modulen, die Daten verarbeiten, getrennt sein.

b.) Nennen Sie drei funktionale Aspekte, drei nichtfunktionale und zwei Aspekte, die beiden Kategorien zuzurechnen sind.

Lösung:

Funktionale Aspekte:

- Eigenschaften der partiellen Korrektheit
- Termination
- Lebensdauer (liveness)
- Freiheit von Deadlocks

Nichtfunktionale Aspekte:

- Sicherheit
- Verfügbarkeit (availability)
- Verlässlichkeit (reliability)
- Testbarkeit
- Modifizierbarkeit
- Wiederverwendbarkeit
- Skalierbarkeit
- Portierbarkeit
- Benutzerfreundlichkeit (usability)
- Fehlerbehandlung

Aspekte, die funktional und nicht funktional sind:

- Performance
- Fehlerbehandlung
- Garbage Collection

Aufgabe 6: Architekturrahmenwerk (10 Punkte = 2 + 2 + 2 + 2 + 2 P)

Was ist ein Architekturrahmenwerk (architecture framework)?

Lösung:

B1: Eine Sammlung von Sprachen zur Architekturbeschreibung mit wohldefinierten Semantiken. Die Semantiken sollen auch die Beziehungen zwischen den Sprachen erklären. Eine Sprache darf verschiedene Notationen unterstützen, z.B. textorientierte und graphische Notationen.

B2: Eine Sammlung von Stilen, Regeln, Invarianten und Eigenschaften, die die Verwendung der Sprache und ihrer Sätze regelt.

B3: Eine Definition der wesentlichen Elemente einer Architekturbeschreibung, d.h. das Rahmenwerk soll uns mitteilen, was wir liefern müssen, um eine vollständige Architekturbeschreibung zu geben.

B4: Eine Definition, was eine Implementierung einer Architekturbeschreibung ausmacht.

B5: Optional: Eine Sammlung von vordefinierten Elementen, Diensten, Standards und Werkzeugen (oder ihren Spezifikationen).

Aufgabe 7: Frameworks (10 Punkte = 1 + 4 + 4 + 1 P)

Im Zusammenhang mit Bibliotheken oder Frameworks kommen die folgenden Begriffe vor:

- Call-down-Prinzip,
- Callback- oder Hollywood-Prinzip.

In welchem Kontext werden sie verwendet?
Welches Prinzip beschreiben sie?
Gilt das jeweilige Prinzip nur für Bibliotheken, nur für Frameworks oder für beide?

Lösung:

Die Begriffe werden im Kontext der *Steuerung des Kontrollflusses* derjenigen Anwendungen, die Bibliotheken oder Frameworks verwenden, gebraucht.

Call-down-Prinzip:
Konventionell entwickelte Anwendungen nutzen typischerweise eine Reihe von Klassenbibliotheken, um die eigenen Funktionalitäten möglichst einfach zu realisieren. Der Kontrollfluss solcher Anwendungen wird typischerweise von ihnen selbst gesteuert. Die Klassenbibliotheken werden lediglich von der sie nutzenden Anwendung aufgerufen, übernehmen jedoch niemals den Hauptkontrollfluss der Anwendung.

Callback oder Hollywood-Prinzip:
Das charakterisierende Prinzip eines Programmrahmenwerks ist die Umkehrung der Ausführungskontrolle, die nicht mehr beim Programmierer der Anwendungsklassen liegt, sondern beim Rahmenwerk: das Hollywood-Prinzip ("don't call us, we call you").
Der Hauptkontrollfluss einer Anwendung, die mit Hilfe eines Frameworks entwickelt wurde, wird vom Framework gesteuert: Zentrale Bestandteile der Anwendungsarchitektur werden bereits vom Framework vorgegeben. Die anwendungsspezifischen Funktionalitäten einer Anwendung, die das Framework nutzt, werden aus dem Framework heraus aufgerufen. So hat das Framework den Hauptkontrollfluss über die Anwendung inne.

Aufgabe 8: Architekturbeschreibungssprachen (10 P = 1.5 + 7.5 + 1 P)

a) Mit welchen Mitteln (außer UML) könnten Sie die Architektur eines Softwaresystems beschreiben?
Nennen Sie drei Darstellungsmöglichkeiten.

Lösung:

- Mittel 1: Architekturbeschreibungssprachen
- Mittel 2: Verschiedene Sichten
- Mittel 3: Softwarearchitekturmuster

b.) UML wird oft zur Beschreibung von Architekturen verwendet, obwohl es dafür nicht entwickelt wurde. Welche Unzulänglichkeiten weist es in diesem Zusammenhang auf?

Lösung:

U1: Das Zusammenspiel der verschiedenen Notationen ist nicht klar definiert. Es gibt kein zugrundeliegendes Architekturmodell, das als semantischer Hintergrund für verschiedene Notationen dienen kann.

U2: UML unterstützt keine wohldefinierten Abstraktionsmechanismen zwischen verschiedenen Beschreibungen. Insbesondere unterstützt UML keine Abstraktion zur Kommunikation zwischen Komponenten.

U3: UML liefert nur schwache Unterstützung für hierarchische Beschreibungen. (Die Strukturierung einer Architekturbeschreibung auf verschiedenen Detailebenen muss außerhalb von UML vorgenommen werden.)

U4: UML stellt keine Techniken zur Parametrisierung - d.h. zur Beschreibung von Familien von Systemen - bereit.

U5: UML bietet nur informelle Mechanismen an, um globale Bedingungen und nichtfunktionale Eigenschaften zu beschreiben.

c.) Zu welcher Sicht (statische oder dynamische Sicht) gehören die folgenden UML-Diagrammarten??

- class diagram
- use case diagram
- component diagram
- collaboration diagram

Lösung:

UML-Diagrammarten der statischen Sicht:class diagram, component diagram

UML-Diagrammarten der dynamischen Sicht: use case diagram, collaboration diagramm

Aufgabe 9: .NET-Framework (10 P = 1 + 4 + 1 + 4 P)

Für das .NET-Framework sind CTS (common type system) und MSIL (Microsoft intermediate language) wichtig. Wozu benötigt man das CTS, wozu das MSIL? Wie funktionieren sie?

Lösung:

CTS: *Programmiersprachunabhängige Interoperabilität*
Um Programmiersprachunabhängigkeit zu erreichen, werden im .NET-Framework Varianten von Programmiersprachen definiert, die auf einem gemeinsamen Typsystem namens CTS (*common type system*) basieren. Beispiele dieser Sprachen sind C#, J#, Eiffel# und VB.NET. Diese Sprachen werden in die prozessor- und programmiersprachunabhängige Zwischensprachen MSIL übersetzt. Auf der MSIL-Ebene können in verschiedenen Sprachen geschriebene Programmteile dann leicht zusammenarbeiten.

MSIL: *Plattformunabhängigkeit*
Um Plattformunabhängigkeit auf niedrigem Abstraktionsniveau zu erreichen, werden Anwendungen und Anwendungsteile vom Entwickler in eine prozessor- und programmiersprachunabhängige Zwischensprache MSIL übersetzt anstatt in die Maschinensprache eines Prozessors. Dieser MSIL-Code wird an die Endanwender ausgeliefert und erst auf dem Zielsystem in die Maschinensprachen des entsprechenden Prozessors übersetzt.
Für den Zeitpunkt dieser Übersetzung gibt es zwei Möglichkeiten:
Entweder erfolgt die Übersetzung während der Installation der Software oder für jede Methode einzeln erst unmittelbar vor der erstmaligen Ausführung des Codes der Methode. Letzteres ist der Default und wird *just-in-time compilation* genannt.
Der MSIL-Code liegt im Abstraktionsniveau zwischen Assemblercode und Hochsprachen. Zum Beispiel ist der MSIL-Code noch in Methoden strukturiert und beschreibt auch Methodensignaturen (im .NET-Framework wird dies als *Meta-Informationen* bezeichnet). Komplexe Kontrollstrukturen sind aber bereits in einfache Befehlsfolgen aufgelöst.

Aufgabe 10: Komponenten-Framework (10 P = 2 + 2 + 1 + 5 P)

a.) Woraus besteht das Komponentenmodell (component model)?

Lösung:

Es besteht im Wesentlichen aus syntaktischen und semantischen Regeln, die spezifizieren, welche Arten von Komponenten unterstützt werden, was eine Komponente darstellt und was ihre Eigenschaften sind.

b.) Was sind die Aufgaben der Komponenteninfrastruktur (component infrastructure)?

Lösung:

Sie erklärt, wie Komponenten zusammengesetzt und weitergegeben werden.
Sie beschreibt die Regeln, wie eine Komponente andere Komponenten findet, wie Komponenten verlinkt werden und wie sie kommunizieren können.

c.) Wofür bieten Komponenten-Frameworks noch Unterstützung an? Nennen Sie zwei Beispiele.

Lösung:

Beispiel 1: Persistierung
Beispiel 2: Sicherheitsmechanismen.

d.) Die Komponenten der Eclipse Rich Client Platform sind die sogenannten Plug-ins.
Was versteht man unter einem Plug-in?
Wie fügt es Funktionalität zu einem System hinzu?

Lösung:

Plug-ins sind strukturierte Bündel von Code und/oder Daten, die Funktionalität zu dem System hinzufügen.

Funktionalität kann in Form von Code-Bibliotheken (Java-Klassen mit öffentlicher Anwendungsschnittstelle), Plattform-Erweiterungen oder auch Dokumentation ergänzt werden.
Plug-ins können Erweiterungspunkte definieren, klar definierte Orte, an denen Plug-ins Funktionalität hinzufügen können.
Ein Plug-in kapselt Logik und/oder Daten und stellt seine Funktionalität über ein Application Programming Interface (API) ggf. anderen Plug-ins zur Verfügung oder erweitert die Funktionalität bestehender Plug-ins durch sogenannte Extensions.