

Software Architecture Analysis Method (SAAM) (Kazman u. a. 1996):

- ① Lege wichtige Qualitätsaspekte fest
- ② Beschreibe alternative Modularisierungen
- ③ Entwickle Szenarien für die Qualitätsaspekte
- ④ Spiele die Szenarien durch und bewerte die Modularisierungen
- ⑤ Betrachte Wechselwirkungen zwischen den Qualitätsaspekten
- ⑥ Fasse die Evaluation zusammen

Beispiel: Key Word in Context (KWIC) (Parnas 1972)

westfälischer Friede

Friede von Osnabrück

Bistum Osnabrück

Beispiel: Key Word in Context (KWIC) (Parnas 1972)

zyklische Vertauschung
jeder Zeile



westfälischer Friede
Friede von Osnabrück
Bistum Osnabrück

westfälischer Friede
Friede westfälischer
Friede von Osnabrück
Osnabrück Friede von
von Osnabrück Friede
Bistum Osnabrück
Osnabrück Bistum

Beispiel: Key Word in Context (KWIC) (Parnas 1972)

zyklische Vertauschung
jeder Zeile



westfälischer Friede
Friede von Osnabrück
Bistum Osnabrück

Sortierung der
Zeilen



westfälischer Friede
Friede westfälischer
Friede von Osnabrück
Osnabrück Friede von
von Osnabrück Friede
Bistum Osnabrück
Osnabrück Bistum

Bistum Osnabrück
Friede von Osnabrück
Friede westfälischer
Osnabrück Bistum
Osnabrück Friede von
von Osnabrück Friede
westfälischer Friede

- Änderbarkeit
 - Eingabeformat ändert sich
 - größere Dateien müssen verarbeitet werden
 - riesige Dateien müssen verarbeitet werden
- separate Entwickelbarkeit
 - verteile Arbeit an Entwicklungsteams
- Performanz
 - berechne KWIC für FB3-Webseiten

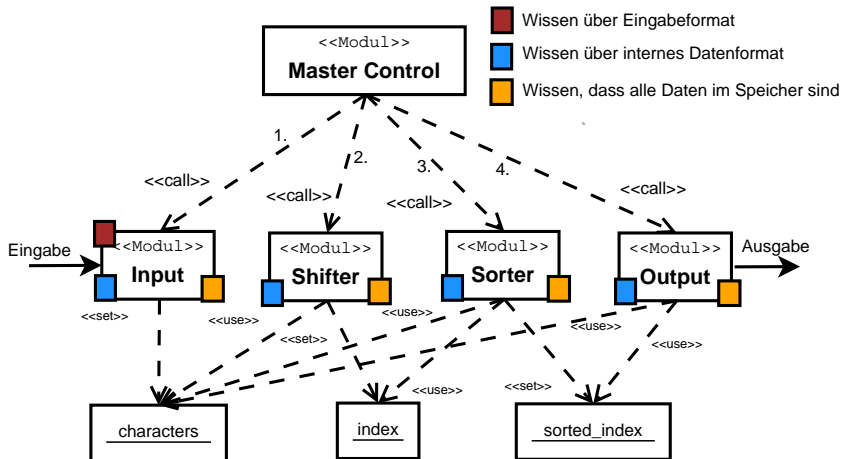
Ablauf

- ① Eingabe der Daten
- ② Interne Speicherung der Daten
- ③ Indizierung (Zeile, Wortanfang, Wortende)

W e s t f ä l i s c h e r F r i e d e	(1, 1, 13)
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20	(1, 15, 20)
B i s t u m O s n a b r ü c k	(2, 1, 6)
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16	(2, 8, 16)
F r i e d e v o n O s n a b r ü c k	(3, 1, 6)
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20	(3, 8, 10)
	(3, 12, 20)

- ④ Zyklische Rotation der Indizierung
- ⑤ Sortierung der Indizierung
- ⑥ Ausgabe der Indizierung

Erste Modularisierung (ablauforientiert)



Änderbarkeit:

	betroffene Module			
	Input	Shifter	Sorter	Output
Eingabeformat	×			
größere Dateien	×	×	×	×
riesige Dateien	×	×	×	×

Getrennte Entwicklung:

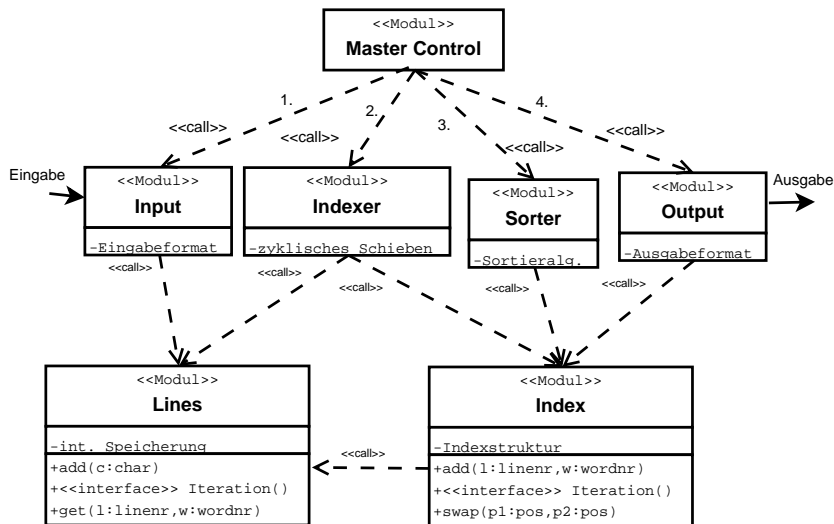
- alle Datenstrukturen müssen bekannt sein
- komplexe Beschreibung notwendig

Performanz:


- schneller Zugriff auf globale Variablen

Zweite Modularisierung (objektbasiert)

... nach dem Geheimnisprinzip (Information Hiding) (Parnas 1972)



Änderbarkeit:

	betroffene Module					
	Input	Lines	Index	Indexer	Sorter	Output
Eingabeformat	×					
größere Dateien		×				
riesige Dateien		×				

Getrennte Entwicklung:

- nur Schnittstellen müssen bekannt sein

Performanz:

- zusätzliche Aufrufe

Abschließendes Wort zur Modularisierung

Es gibt viele Modularisierungsmöglichkeiten.

Jede ist gut für einen Zweck, schlecht für einen anderen.

Mit gängigen Programmiersprachen muss man sich auf eine festlegen.

Abschließendes Wort zur Modularisierung

Es gibt viele Modularisierungsmöglichkeiten.

Jede ist gut für einen Zweck, schlecht für einen anderen.

Mit gängigen Programmiersprachen muss man sich auf eine festlegen.

Definition

Querschnittsbelange (Cross-Cutting Concerns):

Implementierungsaspekte, die eine große Zahl (z.B. alle) von Modulen betreffen.

Beispiele: Fehlerbehandlung, Logging-Mechanismen, Vermeidung von Speicherlöchern etc.

Abschließendes Wort zur Modularisierung

Es gibt viele Modularisierungsmöglichkeiten.

Jede ist gut für einen Zweck, schlecht für einen anderen.

Mit gängigen Programmiersprachen muss man sich auf eine festlegen.

Definition

Querschnittsbelange (Cross-Cutting Concerns):

Implementierungsaspekte, die eine große Zahl (z.B. alle) von Modulen betreffen.

Beispiele: Fehlerbehandlung, Logging-Mechanismen, Vermeidung von Speicherlöchern etc.

Aspektorientierte Programmiersprachen erlauben eine separate Beschreibung dieser Aspekte und ein “Einweben” des Aspekts in das System.

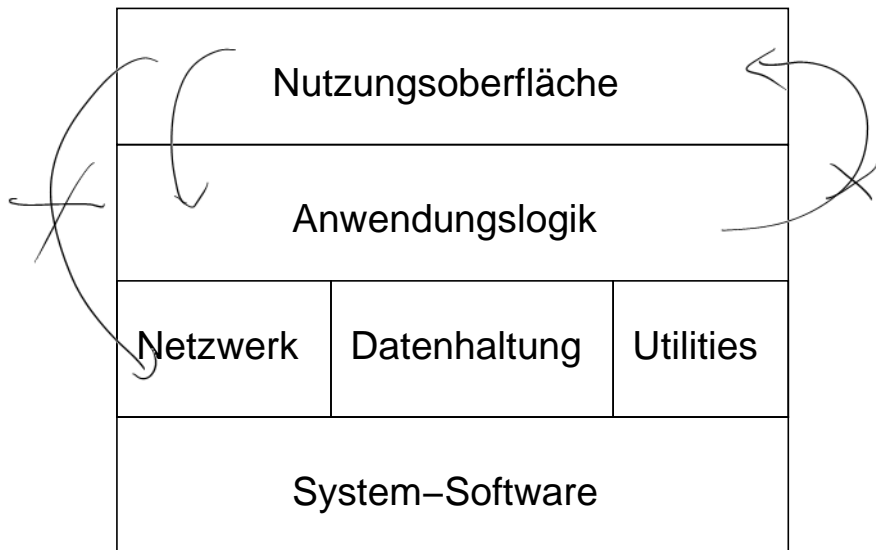
Definition

Architekturstil: beschreibt eine Familie von Architekturen/Systeme als ein Muster der strukturellen Organisation durch

- ein Vokabular (Komponenten- und Konnektorentypen)
- und eine Menge von Einschränkungen, wie Komponenten und Konnektoren verbunden werden dürfen.

Synonyme: Architekturmuster oder Architekturidiom.

Architekturstil: Schichtung



Architekturstil: Schichtung I

- Vokabular:
 - Komponenten: Module und Schichten
 - Konnektoren: Use-Beziehung
- Struktur:
 - Module sind eindeutig einer Schicht zugeordnet
 - Module einer Schicht dürfen nur auf Module derselben und der direkt darunter liegenden Schicht zugreifen
- Ausführungsmodell:
 - Aufruf von Methoden tieferer Schichten
 - Datenfluss in beide Richtungen (von der unteren Schicht zur oberen durch Rückgabeparameter)

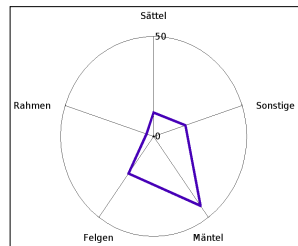
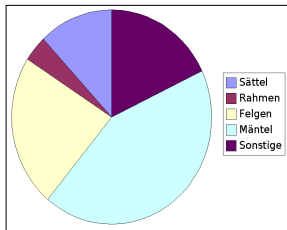
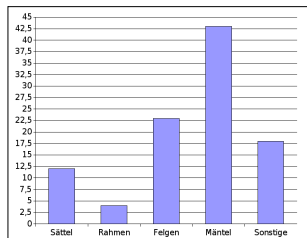
- Vorteile:

- Schicht implementiert virtuelle Maschine, deren Implementierung leicht ausgetauscht werden kann, ohne dass höhere Schichten geändert werden müssen

- Nachteile:

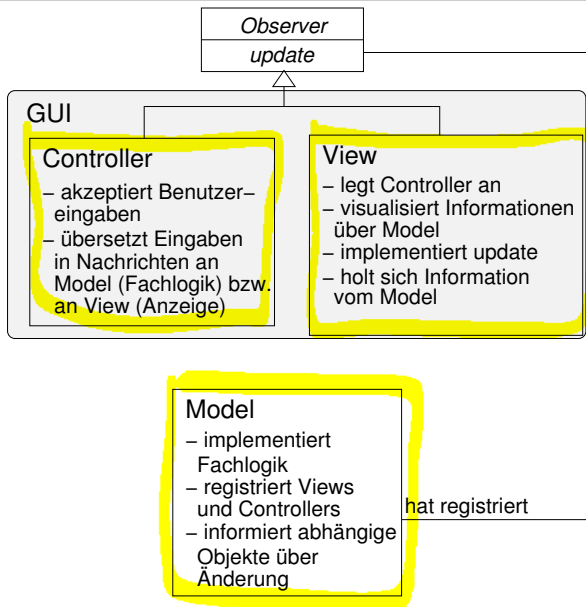
- höherer Aufwand durch das “Durchreichen” von Information
- Redundanz durch Dienste tieferer Schichten, die in hohen Schichten benutzt und auf allen Ebenen dazwischen repliziert werden

Anforderungen

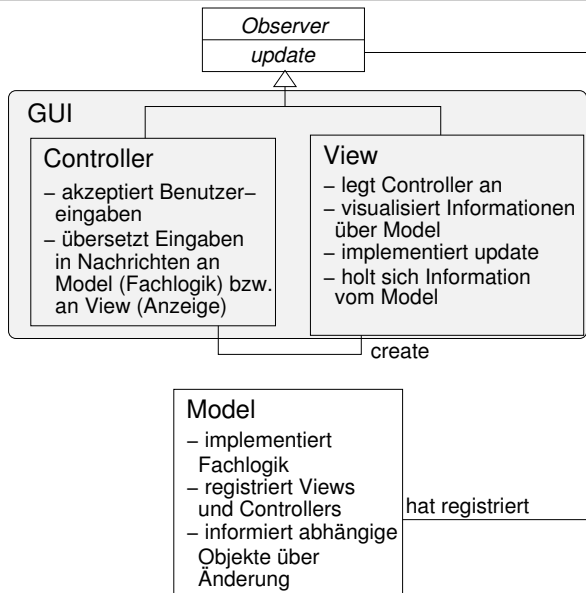


Sättel: 12%
Rahmen: 4%
Felgen: 23%
Mäntel: 43%
Sonstige: 18%

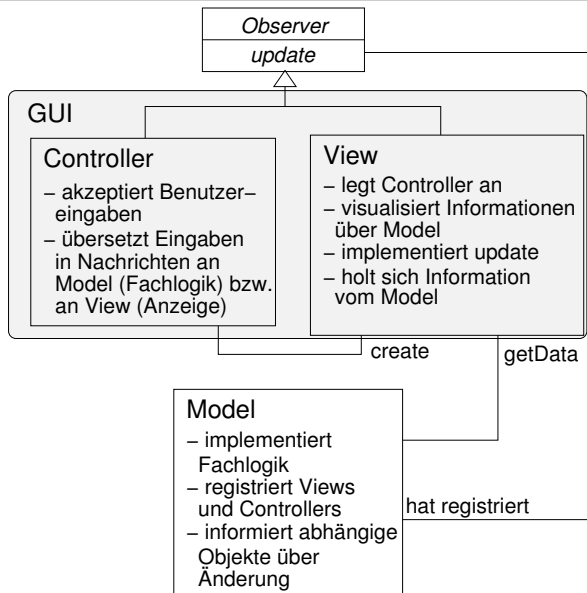
Model-View-Controller (Buschmann u. a. 1996)



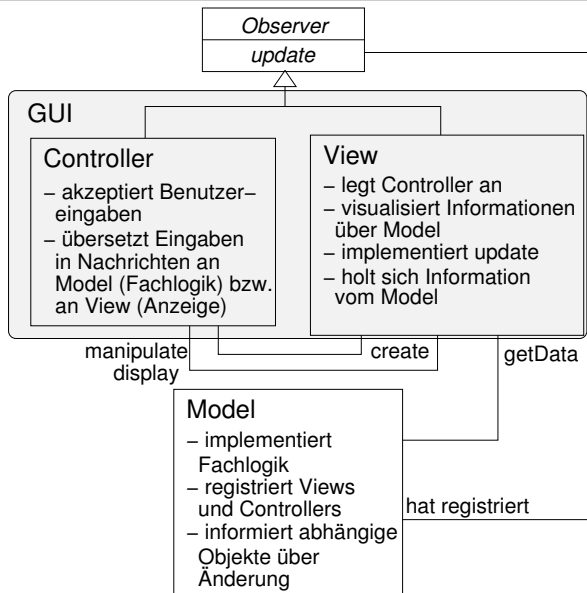
Model-View-Controller (Buschmann u. a. 1996)



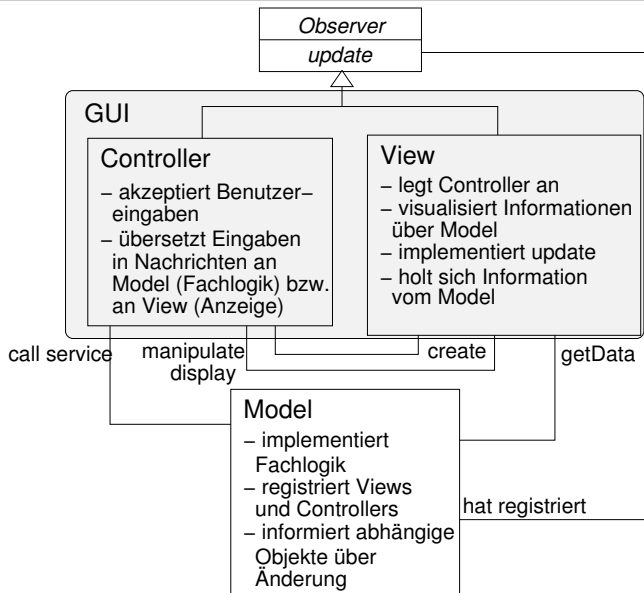
Model-View-Controller (Buschmann u. a. 1996)



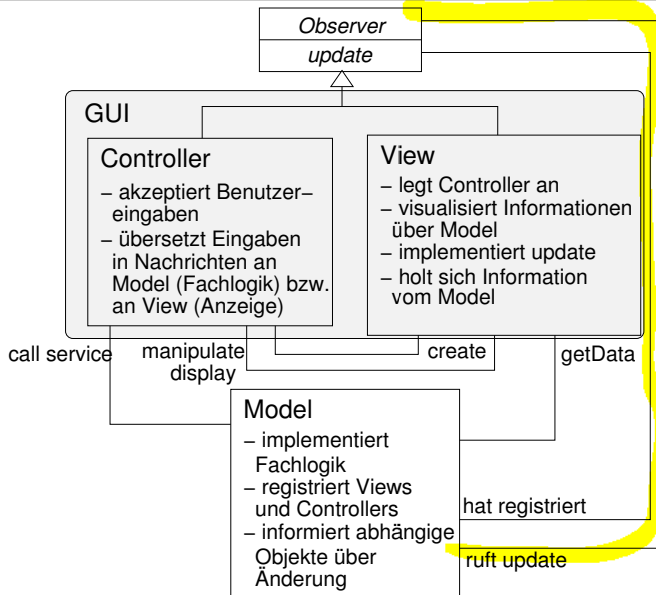
Model-View-Controller (Buschmann u. a. 1996)



Model-View-Controller (Buschmann u. a. 1996)



Model-View-Controller (Buschmann u. a. 1996)



2 Software-Test

- Begriffe des Testens
- Soziologie des Testens
- Testaktivitäten
- Testfall
- Teststümpfe und -treiber
- Komponententests
- Äquivalenztest
- Komponententest mit JUnit
- Grenztest
- Pfadtest
- Maße der Testabdeckung
- Zustandsbasiertes Testen
- Integrationstest
- Leistungstests
- Zusammenfassung der Testarten

- Notwendigkeit und Grenzen des Tests verstehen
- Arten und Varianten des Software-Tests kennen
- eine geeignete Test-Strategie auswählen können
- Testpläne erstellen können
- Tests durchführen können

N.B.: Diese Darstellung folgt in weiten Teilen Kapitel 11 des Buchs von Brügge und Dutoit (2004).

Ein korrektes Programm?

```
class Kalender {
    public static class MonatUngueltig extends Exception {};
    public static class JahrUngueltig extends Exception {};
    public static boolean istSchaltJahr(int jahr)
        {return (jahr % 4) == 0;}
    public static int TageProMonat (int monat, int jahr)
        throws MonatUngueltig, JahrUngueltig {
        int anzTage;
        if (jahr < 1) { throw new JahrUngueltig(); }
        if (monat in {1, 3, 5, 7, 10, 12}) { anzTage = 32;}
        else if (monat in {4, 6, 9, 11}) { anzTage = 30; }
        else if (monat == 2) {
            if (istSchaltJahr (jahr)) anzTage = 29;
            else anzTage = 28;
        } else throw new MonatUngueltig();
        return anzTage;
    }
}
```

Definition

Zuverlässigkeit: Maß für den Erfolg, inwieweit das beobachtete Verhalten mit dem spezifizierten übereinstimmt.

Software-Zuverlässigkeit: Wahrscheinlichkeit, dass ein Software-System während einer festgelegten Zeit unter festgelegten Bedingungen keinen Systemausfall verursachen wird (IEEE Std. 982-1989 1989).

Definition

Störfall (Ausfall, Failure): jegliche Abweichung des beobachteten Verhaltens vom spezifizierten.

Im Nicht-Schaltjahr 200 wird für den Monat Februar 29 ausgegeben.

Definition

Störfall (Ausfall, Failure): jegliche Abweichung des beobachteten Verhaltens vom spezifizierten.

Im Nicht-Schaltjahr 200 wird für den Monat Februar 29 ausgegeben.

Fehlerhafter Zustand (Error): Zustand, in dem ein Weiterlaufen des Systems zu einem Störfall führen würde.

Die Funktion `istSchaltjahr(200)` liefert `true`.

Definition

Störfall (Ausfall, Failure): jegliche Abweichung des beobachteten Verhaltens vom spezifizierten.

Im Nicht-Schaltjahr 200 wird für den Monat Februar 29 ausgegeben.

Fehlerhafter Zustand (Error): Zustand, in dem ein Weiterlaufen des Systems zu einem Störfall führen würde.

Die Funktion `istSchaltjahr(200)` liefert `true`.

Fehler (Defekt, Fault): mechanische oder algorithmische Ursache eines fehlerhaften Zustands.

Die Prüfung in `istSchaltjahr` ist falsch.

Definition

Test: systematischer Versuch, in der implementierten Software Defekte zu finden.

Erfolgreicher (positiver) Test: Test, der Defekt aufgedeckt hat.

Erfolgloser (negativer) Test: Test, der keinen Defekt aufgedeckt hat.

Definition

Test: systematischer Versuch, in der implementierten Software Defekte zu finden.

Erfolgreicher (positiver) Test: Test, der Defekt aufgedeckt hat.

Erfolgloser (negativer) Test: Test, der keinen Defekt aufgedeckt hat.

- Tests sind Experimente zur Falsifikation der Hypothese “System ist korrekt”.
- Aus negativem Test folgt noch lange nicht, dass kein Defekt vorhanden ist.

Tests sind nur *ein* Mittel, die Zuverlässigkeit zu steigern:

- Fehlervermeidung (z.B. Entwicklungsmethoden, Verifikation, statische Analyse)
- Fehlerentdeckung (z.B. Tests, assert, “Quality-Feedback-Agent”, Flugschreiber)
- Fehlertoleranz: Behandlung von Fehlern zur Laufzeit, um Programm fortzusetzen

- Testen wird häufig (aber zu unrecht) als niedere Arbeit angesehen
- Frischlinge werden zu Testern
- Tester brauchen jedoch ein umfassendes Systemverständnis (Anforderungen, Entwurf, Implementierung)
 - Tester brauchen darüber hinaus, Wissen über Prüftechniken

- Testen wird häufig (aber zu unrecht) als niedrigere Arbeit angesehen
- Frischlinge werden zu Testern
- Tester brauchen jedoch ein umfassendes Systemverständnis (Anforderungen, Entwurf, Implementierung)
- Tester brauchen darüber hinaus, Wissen über Prüftechniken
- Autoren haben eine Lesart der Spezifikation verinnerlicht
- Denkfehler in der Implementierung werden sich bei Erstellung von Testfällen wiederholen
- Tester sollten nicht gleichzeitig Autoren sein

- Testen wird häufig (aber zu unrecht) als niedrigere Arbeit angesehen
- Frischlinge werden zu Testern
- Tester brauchen jedoch ein umfassendes Systemverständnis (Anforderungen, Entwurf, Implementierung)
- Tester brauchen darüber hinaus, Wissen über Prüftechniken
- Autoren haben eine Lesart der Spezifikation verinnerlicht
- Denkfehler in der Implementierung werden sich bei Erstellung von Testfällen wiederholen
- Tester sollten nicht gleichzeitig Autoren sein
- Tester versuchen, Fehler zu finden
- das Produkt wird kritisiert, dann fühlen sich Autoren selbst kritisiert
- Egoless-Programming (eine schöne Illusion)

