Wahlpflichtfach Design Pattern

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- miwilhelm@hs-harz.de
- http://www.miwilhelm.de
- Raum 2.202
- Tel. 03943 / 659 338

Inhalt

Internet-Adressen

- www.dofactory.com/Patterns/
- www.patterndepot.com
- www.javapractices.com
- http://www.se.uni-hannover.de/documents/kurz-und-gut/creational-patterns_tliro.pdf

Literatur

- Design Patterns
  Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- Holub on Patterns: Learning Design Patterns by Looking at Code, 978-1590593882
- Entwurfsmuster, Klaus Quibeldey-Cirkel, 978-3-642-63632-5
- Software Design Pattern, 978-1-156-60763-3

Literatur

- Design Patterns Explained, 978-0-321-24714-8
- Pattern-Oriented Software Architecture
  Volume 1, ISBN 978-0471958697
- Pattern-Oriented Software Architecture
  Volume 2, ISBN: 978-0471606956

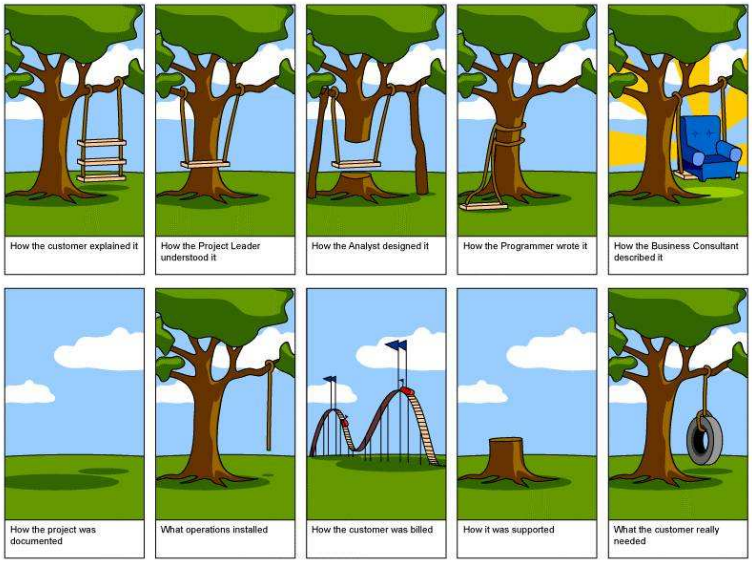
Stand der Software-Entwicklung

Krisis

„Softwarekrise geht zurück auf die mangelnde Ingenieurmäßigkeit im Software-Entwurf zurück“

aus Entwurfsmuster, Quibeldey-Cirkel

Stand der Software-Entwicklung

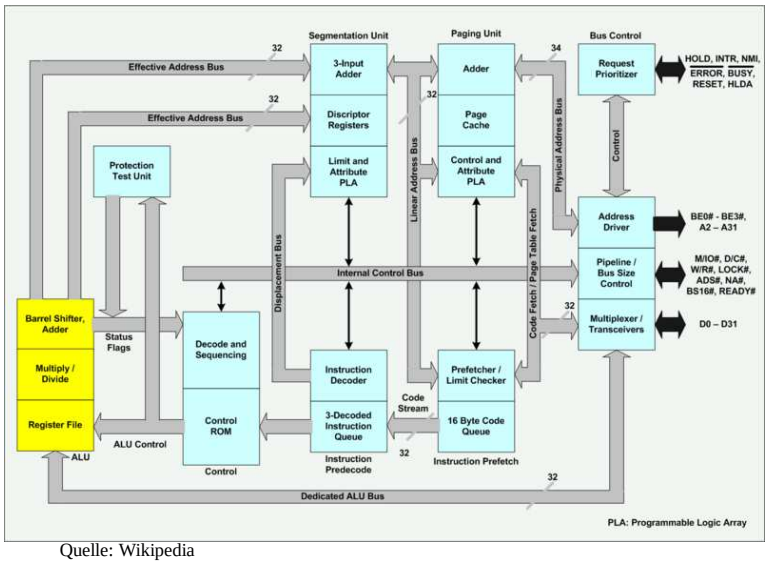


Quelle: thingsdesigner.com

Hardware-Entwicklung

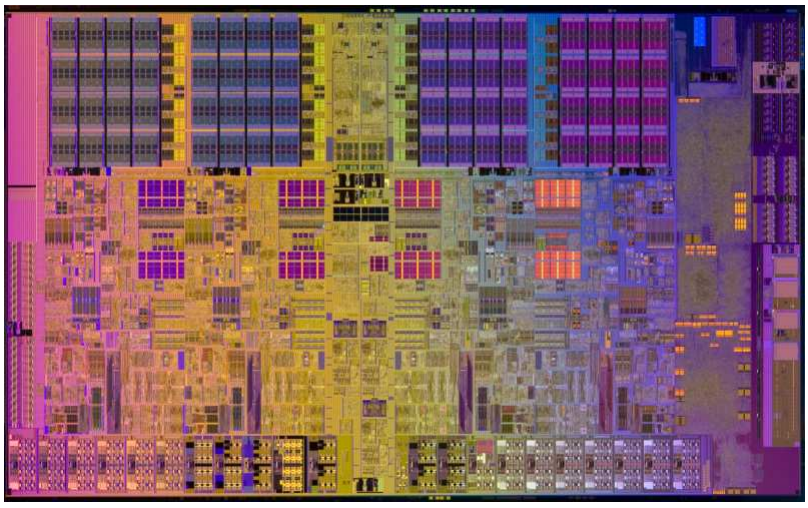
- CPU-Entwicklung
  - Aufbau mit Grundelementen
    - And-Gatter
    - Or-Gatter
    - Nand / Nor-Gatter
  - Aufbau mit Modulen
- Wiederverwendbarkeit
  - Mehrere Integer Units
  - Mehrere Floating-Point Units

Hardware-Entwicklung



Quelle: Wikipedia

Hardware-Entwicklung



Quelle: allround-pc.com

Software-Entwicklung

- Grundelemente der Software-Entwicklung
  - Sequenz
  - Iteration
  - Selektion
- Beispiele: Struktur vs. Code

```
if (a<10) {
  max=44;
}
else {
  max=22;
}
```

```
mov ax, 22
cmp ax, 10
jl less
   mov bx, 22
   jmp together
less:    mov bx,44
together:
```


Software-Entwicklung

- Wiederverwendung
  - Ab einer gewissen Komplexität werden Entwickler Elemente wiederverwenden.
  - Maximal 80 % des vorhandenen Quellcodes können wieder verwendet werden.
  - Strukturentwurf ist besser als Quellcode
- Wiederverwendung durch
  - Makros, Prozeduren
  - Objekte
  - Design Pattern
  - Frameworks, Module
  - Lehrbücher, Skripte, Laboraufgaben/Lösungen
  - Erfahrungen eines Experten

Software-Entwicklung

- Anforderungen
  - Entwickler-Bemerkungen:
    - Die Anforderungen sind nicht komplett
    - Die Anforderungen sind falsch
    - Die Anforderungen sind irreführend
    - Die Anwender sind irreführend
    - Die Anforderungen erzählen nicht alles
  - Selten:
    - Not only were our requirements complete, clear and understandable, but they laid out all functionality we were going to need für the next five years (Designed Patterns Explained)

Software-Entwicklung

- Optimaler Entwurf?
  - Einen optimalen Entwurf gibt es nicht.
  - Beschränkte Resourcen:
    - Budget
    - Entwicklungszeit
    - Rechenzeit des Anwenders
    - Personal:
      - Adding „man power“ to a late project, makes it later
      - (Brooks 75)
  - Versuch und Irrtum
  - Beispiele:Stundenplan, Schach

Software-Entwicklung

- **Hierachien**
  - **Konzeptuelle Schicht**
    - Repräsentiert das Konzept, die grobe Struktur. Quellcode darf nicht erscheinen.
    - Wofür bin ich zuständig?
    - Entwicklung der Klassen-Namen
  - **Spezifikations-Schicht**
    - Wie werden die „Module“, Methoden, benutzt?
    - Entwicklung der Klassen-Methoden (Interfaces)
  - **Implementierung**
    - Wie löse ich meine Anforderungen?
    - Quellcode in den Methoden

Entwurf objektorientierter Software ist nicht leicht

- Was sind Objekte ?
- Was sind Attribute / Felder ?
- Definition der Schnittstellen (get/set)
- Bestimmen der Methoden
- Abhängigkeit zwischen Objekten
- Verallgemeinerung
- Wiederverwendbarkeit

Definition von Muster mit Lösungsschemen

Der Begriff des Musters wurde vom Architekten Christopher Alexander 1977 wie folgt definiert:

„Jedes Muster beschreibt ein in unserer Umwelt beständiges, wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass Sie diese Lösung beliebig oft anwenden können, ohne sie jemals ein zweites Mal gleich auszuführen."

Ein Muster stellt also eine bewährte Lösung nach Art einer Schablone für ein häufiges Probleme bereit. So gesehen sind bereits die länger existierenden Algorithmensammlungen wie z.B. „The Art of Computer Programming" von Donald Knuth eine Katalogisierung von Mustern (vergl. [Gamma97, S.392]).

Die Beschreibung eines Musters enthält im allgemeinen folgende Abschnitte:

- Name: Im Idealfall Charakterisierung der Auswirkungen des Musters in einem Wort.
- Zweck: Was macht das Muster? Was ist das Grundprinzip, was ist sein Zweck? Welches Problem, in welchem Umfeld macht den Einsatz des Musters sinnvoll?
- Auch bekannt als
- Struktur: Beschreibung des eigentlichen Musters, textuell und Klassen-, eventuell ein Sequenzdiagramm.
- Konsequenzen: Vor- und Nachteile, die durch den Einsatz des Musters in einem Entwurf entstehen.
- Implementierung: Der Abschnitt präsentiert die Fallen, Tipps oder Techniken, die man kennen sollten, wenn man das Muster einsetzt.

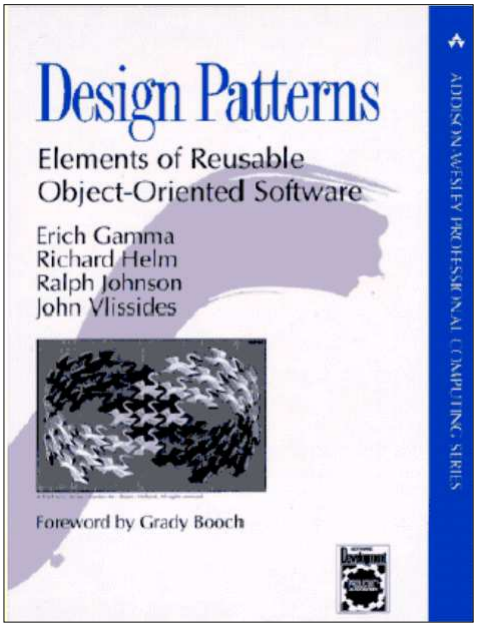
Klassifikation von Patterns

- Architektur Patterns
  - Beschreiben die grundlegende Struktur eines Softwaresystems
  - Client- Schicht, Server- Schicht und Daten- Schicht
- Design Patterns
  - Beschreiben auch Strukturen, allerdings auf einer niedrigeren Ebene als Architektur Patterns
  - Typischer Weise mit Klassen und deren Beziehung zueinander
  - Kann somit zur Ausgestaltung von Teilsystemen führen
- Idioms
  - Behandeln Detailprobleme die z.B. bei der Umsetzung von Design Pattern entstehen können
  - Implementierungsaspekte
  - Programmiersprachenspezifische Probleme (Sockets/Java)

Entwurfsmuster - Konsequenzen

- (+) Robustheit des Entwurfes
- (+) Höherer Wiederverwendungsgrad
- (+) Kommunikation
  - gemeinsames Vokabular
  - höhere Abstraktion
  - leichtere Dokumentation und Verständnis
- (-) höhere „Kosten“

Das Standardwerk



Entwurfsmuster: Gang of Four

		Aufgabe		
		Erzeugungsm.	**Strukturm.**	**Verhaltensmuster**
Gültigkeits-bereich	**klassenbasiert**	Fabrik	Adapter	Interpreter Schablonenmethode
	objektbasiert	Abstrakte Fabrik Erbauer Prototyp Singleton	Adapter Brücke Dekorierer Fassade Fliegengewicht Kompositum Proxy	Befehl Beobachter (MVC) Besucher Iterator Memento Strategie Vermittler Zustand Zuständigskette

Entwerfen von Systemen:

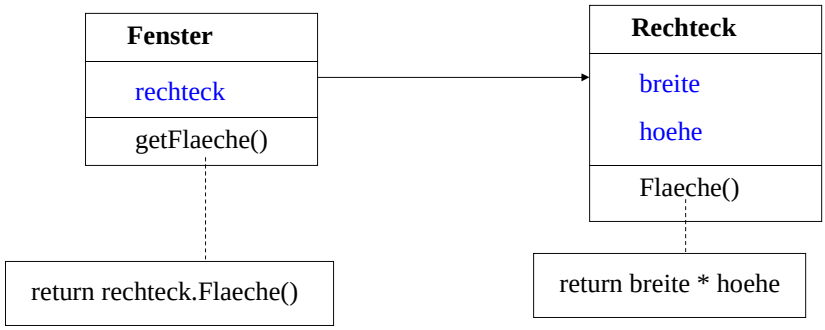
- Klassenvererbung
- Schnittstellenvererbung
- Delegation bzw.
- Komposition (Spezialfall der Delegation)
- Templates (Parametisierbare Typen)
- Klassenbibliotheken
- Frameworks

Verbung vs. Aggregation / Komposition

- Klassenvererbung nennt man auch White-Box-Wiederverwendung.
- Objektkomposition: Neue komplexe Funktionalität wird durch das Zusammenführen oder der Komposition erreicht (hat vs. ist)
- Die Komposition erwartet von Objekten, dass sie Schnittstellen der anderen Objekte respektieren.
- Es sind keine internen Details bekannt (Black-Box-Wiederverwendung).
- Klassenhierarchien bleiben klein.
- Es gibt eine größere Anzahl von Objekten. Das Verhalten hängt von ihren Beziehungen untereinander ab.
- Bei der Aggregation können die "Teile" des "Ganzen" auch einzeln existieren, bei der Komposition nur, wenn auch das "Ganze" existiert (z.B. UNummer mit Student).

Komposition

- Bei der Komposition sind zwei Objekte mit der Abarbeitung einer Anfrage beteiligt.
- Ein empfangenes Objekt delegiert Operationen an ein **Kompositionsobjekt** (Ähnlich einer Unterklassen).



Komposition:

- Der Hauptvorteil der Komposition besteht darin, dass es die Zusammensetzung von Verhalten zur Laufzeit vereinfacht.
- Das Fenster kann zur Laufzeit rechteckig oder kreisförmig sein.
- Nachteile:
  - Dynamische, hochgradig parametrische Software ist schwieriger zu verstehen.
  - Laufzeitineffzienten

Entwurfsmuster: **Klassifizierung**

1. Aufgabe: (was macht das Muster)

- **Erzeugungsmuster** betreffen den Prozess der Objekterzeugung.
- **Strukturmuster** befassen sich mit der Zusammensetzung von Klassen und Objekten.
- **Verhaltensmuster** charakterisieren die Art und Weise, in der Klassen und Objekte zusammenarbeiten und Zuständigkeiten aufteilen

Entwurfsmuster: **Klassifizierung**

2. Gültigkeitsbereich:

- **Klassenbasierte Muster** befassen sich mit Klassen und ihren Unterklassen. Beziehungen durch Vererbung (statisch).

- **Objektbasierte Muster** befassen sich mit Objektbeziehungen, die zur Laufzeit geändert werden können (dynamisch).

Vorgehen:

- Finden passender Objekte
- Bestimmen von Objektgranularität
- Spezifizieren von Objektschnittstellen
- Spezifizieren von Objektimplementierungen
- Wiederverwendungsmechanismen anwenden
- Strukturen der Laufzeit- und Übersetzungszeit aufeinander beziehen
- Veränderungen in Entwürfen vorhersehen