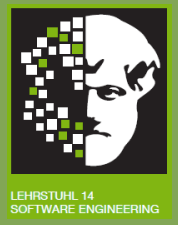


Introduction

Patterns

Components

References

Software-Konstruktion

Wintersemester 2010/2011

Prof. Dr. Jan Jürjens and Dr.-Ing. Holger Schmidt

TU Dortmund – Department of Computer Science
Software Engineering (LS 14)
http://ls14-www.cs.tu-dortmund.de/

Slides are based on the lecture "Muster- und Komponenten-basierte Softwareentwicklung" by Prof. Dr. Maritta Heisel

Organizational issues I

- Exercise sessions: Holger Schmidt and Gregor Kotainy
- Distribution of lectures and exercises as needed
- Dates
  - Freitags, 14:15-15:00, GB IV - 318
  - Freitags, 14:15-15:00, GB IV - 228
  - Freitags, 15:15-16:00, GB IV - 318
  - Freitags, 15:15-16:00, GB IV - 228
  - Freitags, 16:15-17:00, GB IV - 318
  - Freitags, 17:15-18:00, GB IV - 318
- Course material will be published under

  http://ls14-www.cs.tu-dortmund.de/main2/jj/teaching/index.html

  (check regularly!)

Organizational issues II

SWK
JJ+HS
Introduction
Patterns
Components
References

Prerequisite: basic knowledge of software engineering as taught in the course "Softwaretechnik"; knowledge of a programming language does not suffice!

Certificate

You have to pass the exam (60 minutes) to get a certificate for this course. The exam schedule will be announced soon on the webpage of this course.

Who studies in another program as the Bachelor?

Content

Introduction

Patterns

Components

References

- Patterns
  - Architectural styles (coarse-grained design)
  - Design patterns (fine-grained design)
  - Idioms (implementation)
- Components
  - Component definition and specification
  - Component models
    - Java Beans
  - Component-based software development process

Software engineering: definition

Software Engineering $\neq$ Programming!

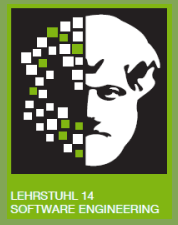
Software Engineering (Balzert):

Goal-oriented provision and systematic use of principles, methods, concepts, notations and tools for team-based development and application of large software systems according to engineering principles. Goal-oriented means e.g. taking costs, time and quality into account.

Software system

A system, whose system components and system elements consist of software.

Software: program + documentation



Phases of software engineering processes

- Analysis
  Goal: understand the problem
- Design
  Goal: obtain structure of software to be built
- Implementation
  Goal: obtain executable software solving the problem
- Testing
  Goal: find defects in implementation

Why do mere programming skills not suffice?

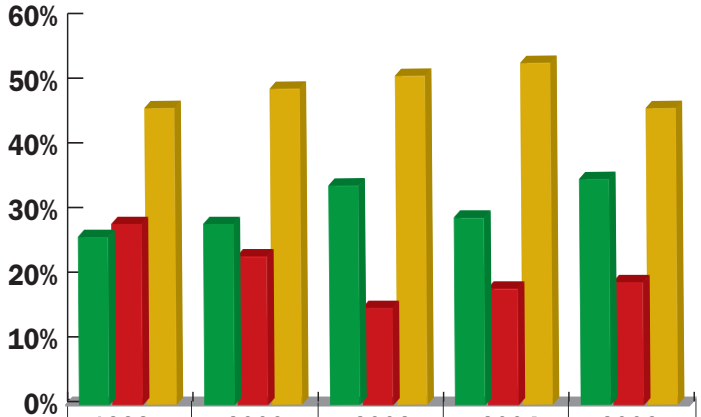
- (Practically) **all** software contains defects.

> *"Software and cathedrals are much the same:*
> *first we build them, then we pray."*
>
> Sam Redwine

- This leads to an immense economic loss and the endangering of human life.
- Why is that so?
- What are new promising areas of research?

Studies of the Standish-Group (CHAOS Research), 1994/1996/1998/2000/2002/2004/2006/2008

CHAOS PROJECT RESOLUTION



	1998	2000	2002	2004	2006	2008
Succeeded	26%	28%	34%	29%	35%	32%
Failed	28%	23%	15%	18%	19%	24%
Challenged	46%	49%	51%	53%	46%	44%

The above chart shows the results of project resolution over the last decade. This data is from our CHAOS Research project on project success and failure and covers more than 60,000 projects.

Why can't software be built in the same way as cars and houses?

Software is something special, because it

- is intangible
- does not wear off through use, but ages because of changes or no changes
- is not restricted through physical laws
- is easily alterable
- is difficult to measure, i.e. describe in a quantitative way
- does not exhibit a continuous but a discrete behavior (no safety margins possible, small causes can have great effects)

Therefore, we need specific methods for constructing software!

Summary

- Computer Science (and thus software engineering) is a very young science.
- Only a small part of software development is programming.
- Due to the special features of software, specific engineering methods are necessary, but have not reached maturity yet.
- An important goal is to develop software with fewer defects.
- For this, promising and exciting new approaches exist.

Recent Developments: From “Art” to “Engineering”

Which steps lead from "Art" to "Engineering"? I

- **Model-based development**
  - Develop sequence of models, each describing different aspects of the software system
  - Models can be analyzed and checked for coherence
- **Object Orientation**
  - Software architecture follows data, not functionality
  - Software as dynamic collection of communicating objects
  - Improved reusability through encapsulation of data
- **Patterns**
  - Templates for the different artifacts generated in software development
  - Useful in all phases of software development
  - Reuse through instantiation

Which steps lead from "Art" to "Engineering"? II

- **Component software**
  - Build software systems from ready-made parts
- **Aspect-oriented programming**
  - Write different programs, each covering different aspects (e.g. computation vs. graphical representation) of the software
  - Compilers combine the different aspect programs to one executable program
- **Software Engineering for special applications**
  e.g. Internet and multimedia applications

Why patterns and components?

- Both belong to the promising new developments
- Both are based on reuse:
  - Patterns allow re-use of **software development knowledge**
  - Components allow re-use of **pre-fabricated software**
- Both can be used in combination

Patterns: basic ideas

- **Templates** for documents set up during software development
- Serve to represent and re-use software development knowledge
- Represent **essence**, abstract from details
- Are used by **instantiation**
- Are available for (almost) all phases of software development

Components: basic ideas

- Assemble software from pre-fabricated parts
- Re-compilation not necessary
- Source code may be inaccessible
- Important: interface descriptions and component models
- Interoperability is an issue

Architectural Patterns

Software architecture

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

The software we construct for solving a software development problem must be structured further.
That structure is called architecture. It is the result of (coarse-grained) design. It structures the software in terms of

- Components
  These carry out computations. Examples: Filters, data bases, objects, abstract data types
- Connectors
  Means of interaction between components. Examples: procedure calls, pipes, event broadcast

Definition Bass et al. (1998)

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

> The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Important points of the definition

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

- Only properties of components that are externally visible are described.
- These (and only these) constitute the assumptions that can be made by components about one another.
- Internal details that are unimportant for the interaction of components are abstracted from.
- An architecture can define more than one structure. For example, assignment of “modules” to teams, set of parallel processes existing at runtime.

Why architectures?

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

All software has an architecture, even if nobody knows it!

However, that architecture should be *explicitly* designed and documented, because this entails that the software

- is better comprehensible
- can be analyzed more easily, e.g. for efficiency
- can be better maintained
- can be implemented systematically

Architectural styles

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

- Are patterns on the level of coarse-grained design, and are also called architectural patterns
- Classify software systems
- The architecture of a software should be an instance of some architectural style

Types of components used in architectures I

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

Passive components:

- process the requests sequentially and may return values
- used in call-and-return systems

Types of components used in architectures II

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

Independent components:

- can be implemented as communicating processes
- may initiate actions on their own (active)
- communicate using messages, pipes/streams, or shared memory
- different components can run on one computer with shared memory, or components can be distributed over a network
- exchange data, but do not control each other
- goal: modifiability of the software by decoupling different computations

Types of components used in architectures III

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

Reasons for independent components:

- Software should run on a multiprocessor-platform
- Software could be structured as a set of loosely coupled components, i.e. a component should be able to make reasonable progress while waiting for events from other components
- Performance is important. It can be improved by assigning tasks to the processes and assigning processes to processors.

Structure of software / components I

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

- function-oriented (FO, main program/subroutine)
- object-oriented (OO)

function-oriented software

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

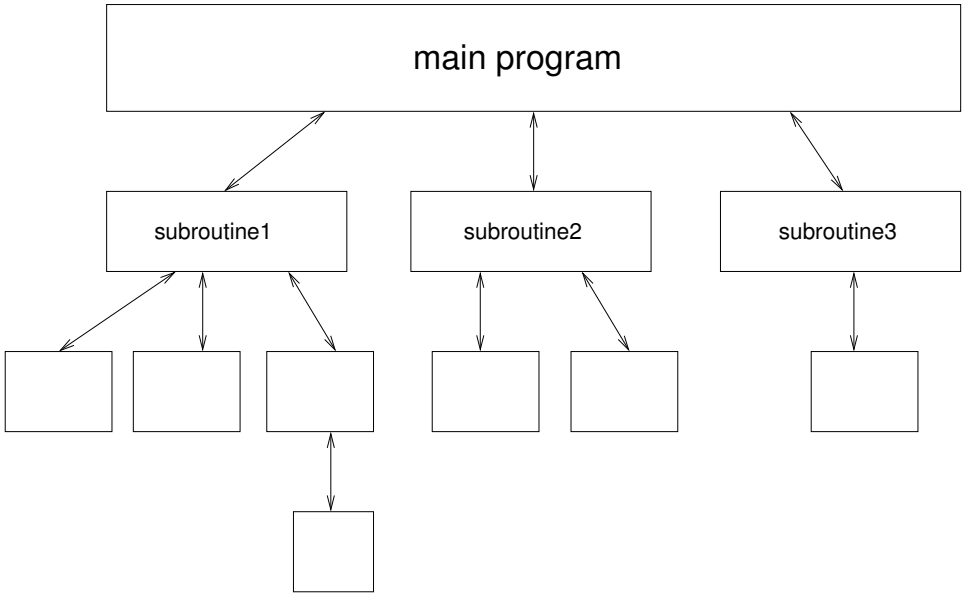
Design Patterns

Idioms

Patterns: Summary

Components

References



- Hierarchical decomposition of functionality, based on a *uses*-relation
- One single control-line, directly supported by programming languages
- Implicit subsystem structure: subprograms as modules
- Hierarchical deduction: correctness of a subprogram depends on the correctness of the subprograms it calls.

Object-oriented software

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

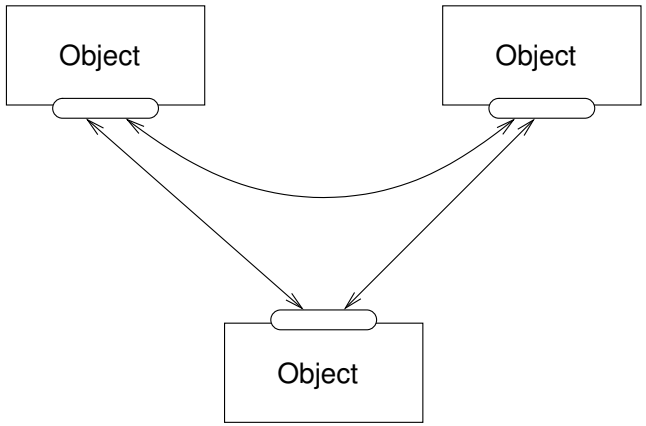
Design Patterns

Idioms

Patterns: Summary

Components

References



- Orientation of the architecture on the data
- If modifiability and the ability of integration (through well-defined interfaces) is important, consider using an object-oriented design.
- Encapsulation: access only possible through defined operations. The user of a service does not need to know, how it is implemented. The implementation can be changed.
- Disadvantage: object identities must be known.

Interfaces between components / coupling

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

Types of component coupling:

- call-and-return (for passive components)
- messages / events (for independent components)
- pipes / streams (for all, even network pipes exist)
- shared memory (for all local components)

Call-and-return

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

Call-and-return (control flow with some associated data):

- Used to connect passive components
- Corresponds to the call of an operation in a programming language
- Technically, a shared memory (stack or processor registers) is used to pass parameters and return values
- Works for function-oriented (main program/subroutine) and object-oriented software
- If the sequence of computations is fixed and components cannot make reasonable progress while waiting for the results of other components, consider using synchronous calls.

Messages / Events I

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

Messages or Events (control flow with some associated data):

- Asynchronous vs. synchronous (call-and-return) communication
  - between active components usually asynchronous communication is used
  - message queues are used to implement asynchronous communication
  - asynchronous messages cannot have a return value - an additional message in the opposite direction is necessary

Messages / Events II

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

- Sender-/Receiver relationship
  - 1:1 - The sending component sends a message to one known receiver component
  - 1:1U - The sending component sends a message to a unknown receiver component - The receiver has to register to get the message
  - 1:N - The sending component sends a message to all components that are registered to receive the message (see *Observer* pattern)

Messages / Events III

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

- Distribution, several components of a software may run:
  - within a single task (asynchronous messages are not necessary)
  - within a single process in separate tasks, but the same memory region
  - locally on one computer in separate processes (communication is between different processes)
  - remote on different computers (communication is over a network connection)

Messages / Events IV

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

- Data
  - no parameters, only single event (only control flow)
  - only limited data (e.g., a pointer)
  - only simple data types
  - serializable data
  - any object

Messages / Events V

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

- Implementations:
  - Events in Java (synchronous, 1toN, within single process, any object, OO)
  - Delegates in .NET (asynchronous, 1toN, within single process, any object, OO)
  - Signals and Slots in C++ with QT (asynchronous, 1toN, within single process, any object, OO)
  - Remote procedure calls (Windows RPC/Sun RPC) (synchronous or asynchronous, 1to1, remote, serializable data, FO)
  - Remote Method Invocation (Java RMI) (synchronous, 1to1, remote, serializable data, OO)

Messages / Events VI

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

- Corba (synchronous/asynchronous since 2000, 1to1, remote, serializable object, OO)
- Windows Events/Mutex/Semaphore received with *WaitForSingleObject* command (asynchronous, 1toN, within single process, no parameters, FO)
- Unix Signals sent with *kill -x* (asynchronous, 1to1, local, no parameters, FO)
- Windows Message Queues (asynchronous, 1to1, remote, serializable data, FO)

Pipes / Streams

SWK

JJ+HS

Introduction

Patterns

Architectual Patterns

Design Patterns

Idioms

Patterns: Summary

Components

References

Streams (data flow with necessary control messages):

- Network sockets
- Unix pipes (only between components of one computer)
- Windows named pipes (only between components of one computer)