Welcome to the Module Foundations of Software Engineering (SWT-FSE-B)

Dr. Alexander Heußner
Lehrstuhl Softwaretechnik & Programmiersprachen
Fakultät WIAI, Universität Bamberg

SoSe 2016

About the Lehrstuhl SWT

- Teaching & research

  Programming languages,

  software engineering,

  concurrency, foundations of

  software specification, analysis,

  and verification,...

- Office hours

– Dr. A. Heußner	Tuesdays	12:30 – 13:30	WE5/03.012
– J. Gareis	Tuesdays	13:00 – 14:00	WE5/03.011
– O. Seddiki	Thursdays	12:30 – 13:30	WE5/03.081

Bachelor Teaching @ Lehrstuhl SWT

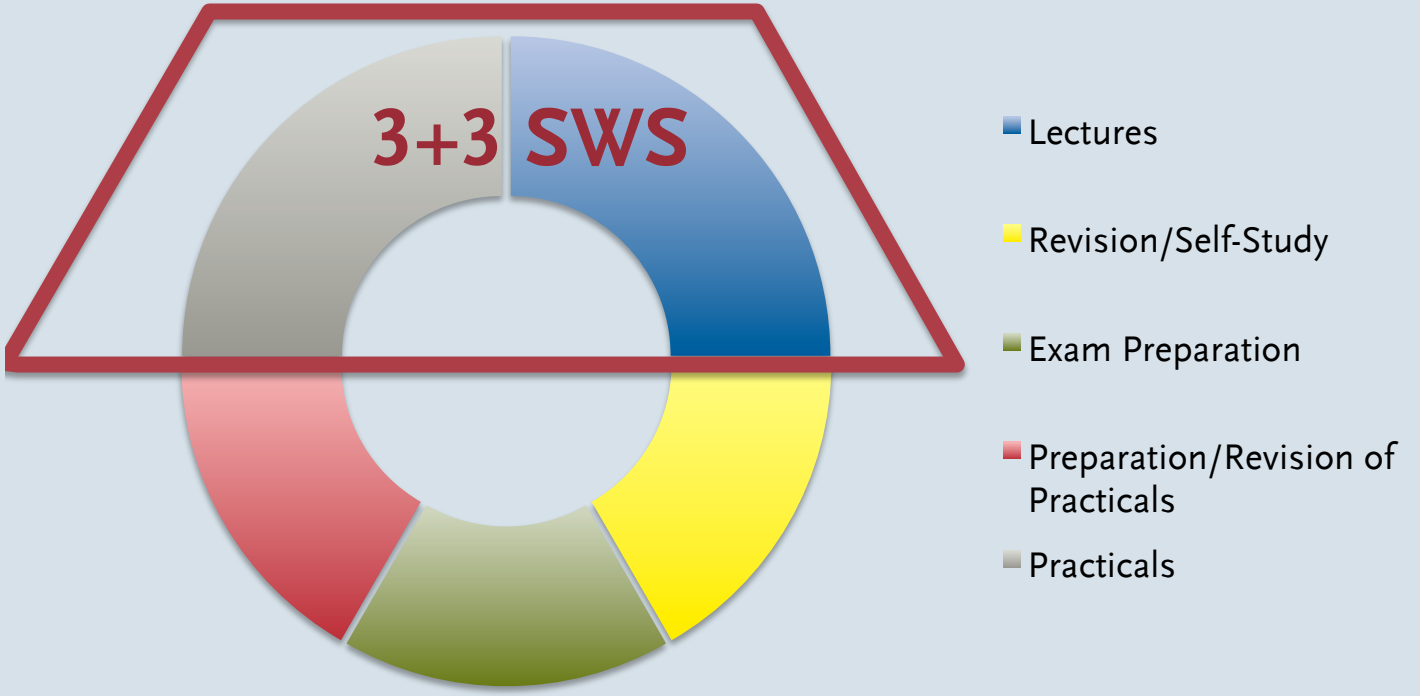
- Summer semester
  - *Soft Skills in IT Projekten* (SWT-SSP-B, 3 ECTS, 2 SWS)
  - *Bachelor Studies Project* (SWT-PR1-B, 6 ECTS, 4 SWS)
  - *Software Systems Science Project* (SWT-PR2-B, 2x 6 ECTS, 2x 4 SWS)
  - *SWT Seminar* (SWT-SEM-B, 3 ECTS, 2 SWS)

  *This semester's topic is "**Multi-Paradigm Programming with** Scala "*

- Winter semester
  - **Software Engineering Lab** **(SWT-SWL-B, 6 ECTS, 4 SWS)**
  - *Foundations of Software Analysis* (SWT-FSA-B, 6 ECTS, 4 SWS)
  - *Principles of Compiler Construction* (SWT-PCC-M, 6 ECTS, 4 SWS)

This Module: SWT-FSE-B

- About this module
  - Compulsory module for AI and SoSySc students
  - Principal module language is *English*
  - 6 ECTS = 180h of workload
  - 6 SWS = 3 SWS lectures + 3 SWS practicals



This Module: SWT-FSE-B

- Times & locations
  - *Lectures:* Mondays 12:15–13:45 in room WE5/04.004 *(Dr. A. Heußner)*
  - *Lectures/Plenum Practicals:* Tuesdays 10:15–11:45 in room WE5/04.004
  - *Group Practicals:*
    - Thursdays 14:15–15:45 in room WE5/04.004 (O. Seddiki)
    - Thursdays 14:15–15:45 in room WE5/04.003 (J. Gareis)
    - Fridays 8:15–9:45 in room WE5/02.029 (O. Seddiki)
  - *Distribution into groups at first practical session on 26th April*
  - *First week of semester: lectures only – in WE5/04.004*
  - *Several replacement dates for cancelled sessions due to holidays*

Calendar (preliminary)

KW 15: Lecture (L) + L + L *April*
16: L + Plenum Practical(PP)+ Group Practical (GP)
17: L + Programming in the Small Workshop + GP

KW 18: L + L + PP(on Wednesday at 16:00) *May*
19: L + L + GP
20: (Pfingsten) GP on Thursday + L (on Friday at 12:00)
21: L + L + PP (on Wednesday at 16:00)
22: L + Exam Preparation + GP

KW 23: L + L + GP *June*
24: L + Exam Preparation + GP
25: L + (guest) L + GP + Verification Workshop (floating, tba)
26: L + PP + GP (+ Verification Workshop)

KW 27: L + Scrum Workshop + GP *July*
28: L + L + Questions & Answers

This Module: SWT-FSE-B

- Legal note

  It is not permitted to record any lecture or practical of this module, via any audio, photo or video recording device.

Teaching Material

- All teaching material can be found on the corresponding VC page including but not limited to
  - *Lecture notes* in form of slides (≠ slides used in lecture, contain "holes")
  - References to chapters in standard text books (e.g. [Sommerville])
  - References to articles for further reading
  - Small quizzes
  - *Worksheets* for practicals (no solutions published!)
  - Material for exam preparation
  - Forums for discussing topics and contacting the teachers
- Material is sufficient for preparation & revision of the module...
- **...but preparation/revision is in your own responsibility!**

Examination: Written Exam

- Date & location
  - To be announced by the University's Exams Office
  - Registration via *FlexNow* – registration period set by the Exams Office
- What you should expect
  - Questions related to any topic taught in the lectures and practicals
  - Passing requires more than recalling knowledge of terminology
    - **You must be able to apply your knowledge to a small case study**
- Further information on teaching matters and etiquette
  - See the VC course *"Aktuelle Informationen des Lehrstuhls SWT"*
- **It is highly recommend that you attend all lectures and practicals**

Bonus Points

- Bonus points can be gained for a “reasonable” hand-in for the following four tasks:

1. *Programming in the Small I: Programming assignment*
   - Handed out today (details later)
   - Homework assignment: small individual programming task in Java
   - Submission until 25th April 12:00 CEST
2. *Test-Exam I*
   - Individual work in practical session on 31st May
   - Solving a small set of exercises that will be marked
3. *Test-Exam II*
   - Individual work in practical session on 14th June
4. *Programming in the Small II: Essay*
   - Hand out on 27th June (preliminary)
   - Submission until 11th July 12:00 CEST (preliminary)

Bonus Points II

- For each exercise you could *gain at most a 5% increment* to the final marking of the exam, e.g. additional 5 points of 100 total.
- The increment *is only awarded if the exam is passed*, i.e. if your mark would be at least 4.0 (without bonus points).
- **Submission deadlines are strict, i.e., everything not handed in in time will not be considered for bonus points.**
- **Any collusion, academic misconduct and cheating (e.g. "Unterschleif") will immediately lead to losing the chance for *any bonus points for the whole module.***

Etiquette

- Arrive in time. Avoid disturbing your colleagues when entering late / leaving earlier.
- No eating and drinking in the lecture rooms.
- Laptops are for note-taking only[1].
- Mobile phones, MP3 players etc. can wait until afterwards.

- Apply basic conversation etiquette when asking questions and try to "address" the whole audience when speaking.

[1] Facebook, Twitter, WhatsApp,... are no note taking apps!

Best Practices

- Take notes (“consuming” slides is not sufficient).
- Revise each lecture/practical in-time.
- Use the referred standard literature for revision.
- Use the supplied quizzes to question yourself.
- **Actively** participate in the practicals.
- Ask questions in lectures/practicals if something is unclear, contact your teachers in their office hours if needed.
- Don’t participate unprepared in the exam.
- Aim for bonus points.

PROGRAMMING IN THE SMALL

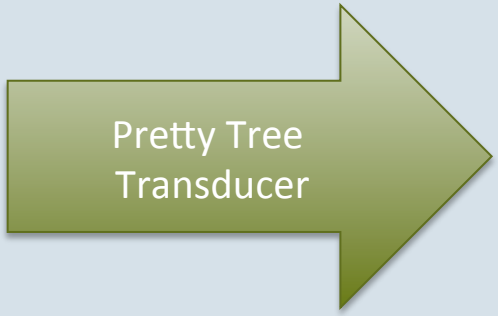
Firsts Homework Assignment

- Individually developing (no teams!) a small software tool in Java where
  - we supply a project brief describing the programming task;
  - we supply an Eclipse project including the basic code infrastructure;
  - you hand in the Eclipse project via the VC.
- Details can be found in the project brief (also including what language features you are allowed to use).
- Submission of your solution to the assignment via the VC **until MONDAY 25th April 2016 12:00 CEST (strict deadline).**

The Pretty Tree Transducer

- A tree encodes a calculation on a set of integers.
- Tool pretty prints this tree and the outcome of the calculation.
- Example:

```
1,,I
2,1,U
3,2,1,2
4,2,3,4
5,1,1,4
```




```
1 : I -> {1,4}
|
+- 2 : U -> {1,2,3,4}
|  |
|  +- 3 : {1,2} -> {1,2}
|  |
|  +- 4 : {3,4} -> {3,4}
|
+- 5 : {1,4} -> {1,4}
```

- Input:
  Tree representing a calculation (intersection/union) on lists of integers given as file of format comma-separated values.
- Output:
  Pretty printing the tree including the result of the computation.

Programming in the Small

- The Pretty Tree Transducer will serve as one of *the running examples of lectures/practicals*.
  Thus, you can re-use your code in later exercises.
- Explicit *workshop on "Programming in the Small"* in 3rd week (after the submission deadline) based on the assignment.
- The Pretty Tree Transducer will also be the topic of the *essay* at the end of the semester.

INTRODUCTION & MOTIVATION

[MAIN SOURCE: I. SOMMERVILLE. *SOFTWARE ENGINEERING*, 8TH ED. ADDISON-WESLEY, 2007]

- What is “Software Engineering”?
- Software development lifecycle
- Software development activities and processes

Software Today

- Is an indispensable part of our modern world
  - *"The fuel on which businesses run, governments rule, and societies become better connected"* [G. Booch, 1998]
- Plays a key role in almost all sectors of our economy
  - *Transport* – cars, airplanes, railways, ...
  - *Communication* – phone, email, radio & television, ...
  - *Manufacturing* – assembly lines, robots, ...
  - *Medicine* – scanners, heart pacemakers, radiation equipment, ...
  - *Finance markets* – banks, insurances, stock exchanges, ...
  - *Domestic products* – washing machines, microwaves, ...
- **Almost nothing works without software**

What is Software?

- First mentioned in 1958
  - *J. W. Tukey*. *The Teaching of Concrete Mathematics*. American Mathematical Monthly, 65(1):1-9, January 1958.
- Some Definitions
  - "Computer *programs*, procedures, rules, and possibly associated *documentation* and *data* pertaining to the operation of a computer system" [IEEE Standard Glossary of Software Engineering]
  - "A number of separate programs, *configuration files*, system documentation, and user documentation" [Sommerville]
- **Software is more than just a computer program**

Characteristics of Software

- Is *immaterial* and *easy to modify*
  - Defects cannot be seen but must be observed
  - Development progress is difficult to track
- *Does not wear off* with use, but still ages
  - System environment, e.g., hardware and operating system version, changes quite frequently
- Is *difficult to predict* in its behaviour
  - Even the smallest change can result in a crash
  - Changes require much oversight and detailed knowledge
- Is *complex* and costly to develop
  - Much more than non-experts would expect

Some Types of Software

- **System software vs. application software**
  - E.g., device drivers vs. social networking software
- **Generic software vs. bespoke software**
  - Shrink-wrapped vs. custom-built (e.g., MS Office® vs. UnivIS)
  - *Today's third way: component software* (customisable, e.g., SAP R/3®)
- **Embedded vs. stand-alone**
  - Embedded software is part of larger technical systems (e.g., an airbag controller) and often involves *real-time* aspects
  - Stand-alone software are typical PC programs (e.g., payroll, accounting)
- **Sequential vs. parallel vs. distributed software**

What is Software Engineering?

- Some definitions
  - "(1) *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software*; that is, the application of engineering to software. (2) The study of approaches as in (1)." [IEEE Computer Society]
  - "An *engineering discipline* that is concerned with *all aspects of software production* from the early stages of specification to maintaining the system after it has gone into use." [Sommerville]

- **Software engineering considers much more than just technical aspects!**
  - E.g., processes and methods carried out by humans

Software Engineering & Computer Science

- Intersect with each other
  - Computer scientists learn how to develop software
  - Software engineering involves programming but also non-technical disciplines, in particular management
- According to Ian Sommerville
  - "Software engineering is concerned with the *practical problems* of producing software."
  - "Computer science is concerned with the *theories and methods* that underlie computers and software systems."
  - **"Elegant theories of computer science cannot always be applied to real, complex problems that require a software solution."**

Software Engineering & Programming

- Software engineering is much more than programming
  - Is *conducted in teams* that, among many other things, also program
  - Addresses non-technical issues that cannot be solved by programming
- Programming knowledge benefits software engineers
  - Software cannot always be produced by gluing components together
  - Programming experience helps with, e.g., conducting *realistic* analyses, cost/time estimations and management processes
- Both disciplines require
  - A good knowledge of techniques and methods, e.g., *abstraction (hierarchy)* and *structuring (modularisation)*
  - The disciplined use of tools
  - **Lots of practice and experience!**

Historic Background: The Software Crisis

- Started in the late 1960s
  - "[...] as long as there were no machines, programming was not a problem at all; when we had a few weak computers, programming became a mild problem, and **now that we have gigantic computers, programming has become an equally gigantic problem**." [E.W. Dijkstra, ACM Turing Award Lecture, 1972]
  - *"Software engineering"* is suggested as a response to the problem [F.L. Bauer, 1968/69]
- And continues
  - Studies in the 1970s showed that *3/4th of developed software is never deployed*, and most of the remaining 1/4th only after major revisions
  - Further studies in the 1990s and 2000s showed that *1/2 of all projects exceed budget*, and *2/3rd are delivered late or not at all*

The Software Crisis Continues?!

- *1994 – New baggage handling system at Denver airport*
  - 9 months late at an extra cost of $1m/day
- *1996 – Self-destruction of Ariane 5 after launch*
  - Reuse of Ariane 4 code under wrong premises
- *1999 – NASA's Mars Climate Orbiter lost*
  - Different development teams interpreted measurement data in metric units and, respectively, imperial units
- *2004 – Germany's lorry road-toll system delayed*
  - Combination of various technical problems at hardware, software and infrastructure level
- *2008 – Baggage handling again! Heathrow's Terminal 5*

Rise of Complexity in Software

- Example – SAP's Resource Planning Software R/3®
  - 1994: approx. 07m lines of code (LOC) in 14k modules
  - 1997: approx. 30m LOC in 200k modules
  - 1999: approx. 50m LOC in 400k modules
  - *Many of these modules are connected to a computer network*
- Additional problem – maintenance of legacy code
  - Typically `grown' over two decades or more
  - Employs *outdated technologies* and often heterogeneous technologies
  - Frequently suffers from *missing documentation* and lack of access to engineers who developed the software
- One of today's software engineering challenges
  - **How to compensate for the sharp increase in complexity?**
  - How to develop high-quality, networked software?

Software Development follows a Process

- **Software processes are "sets of activities that lead to the production of a software product"** [Sommerville]
- Should always be selected according to, and be adapted to,
  - The *characteristics of the system* to be developed
  - The *capabilities of the people* in the development team
- Come in different flavours
  - *Very structured* approaches, e.g., for developing critical systems
  - *Flexible, agile* approaches, e.g., for developing software systems with rapidly changing requirements
- **There is no ideal process!**

The Development Lifecycle

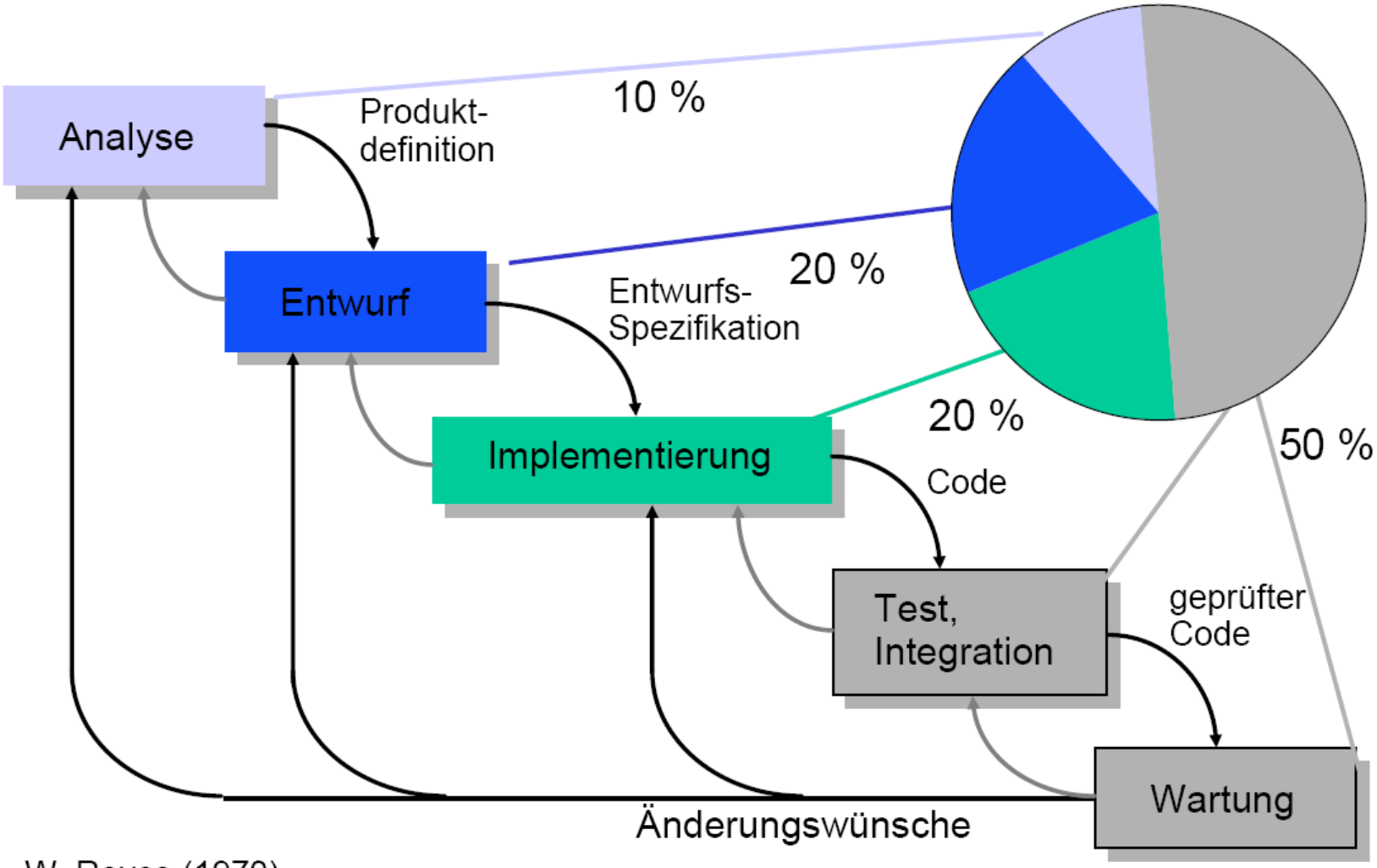
Generic software process

1. *Specification* (**Analyse**) – requirements
2. *Architecture & design* (**Entwurf**)
3. *Implementation* (**Implementierung**)
4. *Testing* (**Test**) – including integration
5. *Maintenance* (**Wartung**) – deployment, support & evolution

Each activity follows a process, too

[IBM]

Typical Distribution of Effort

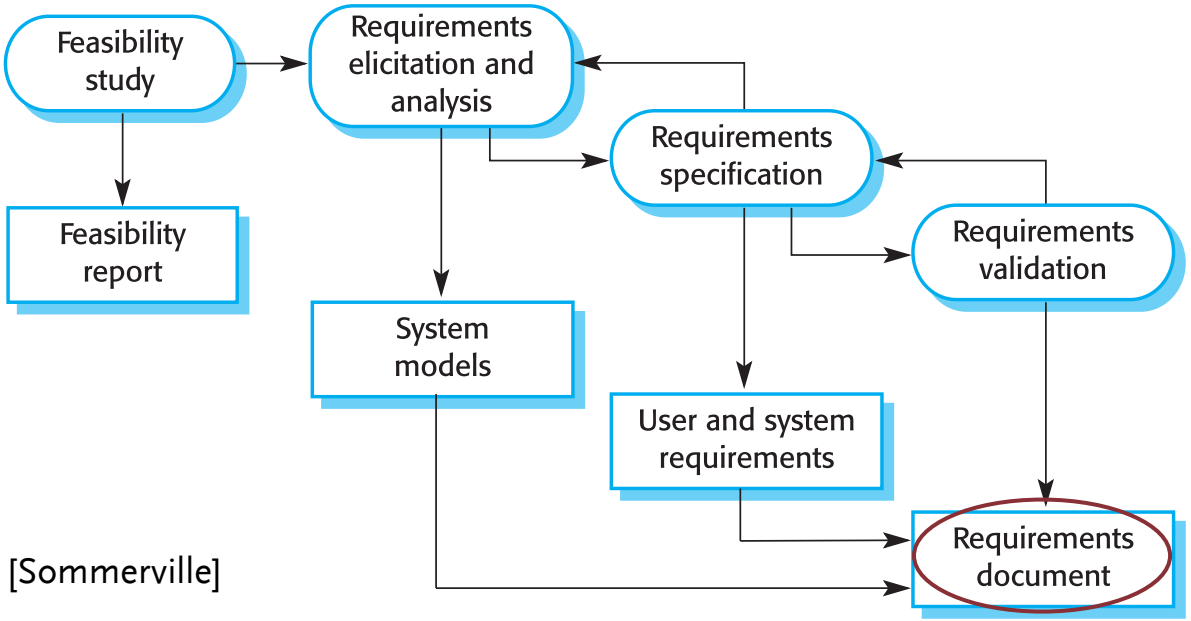


W. Royce (1970)

[Rumpe]

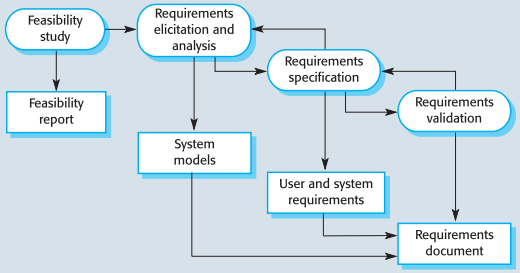
Software Specification: “What to Build?”

- Understanding and defining
  - The **services required** of the system and their desired quality
  - The **constraints** on the system’s operation and development (e.g., platform, programming language, database to be used)
- Requirements engineering process



[Sommerville]

Software Specification Phases

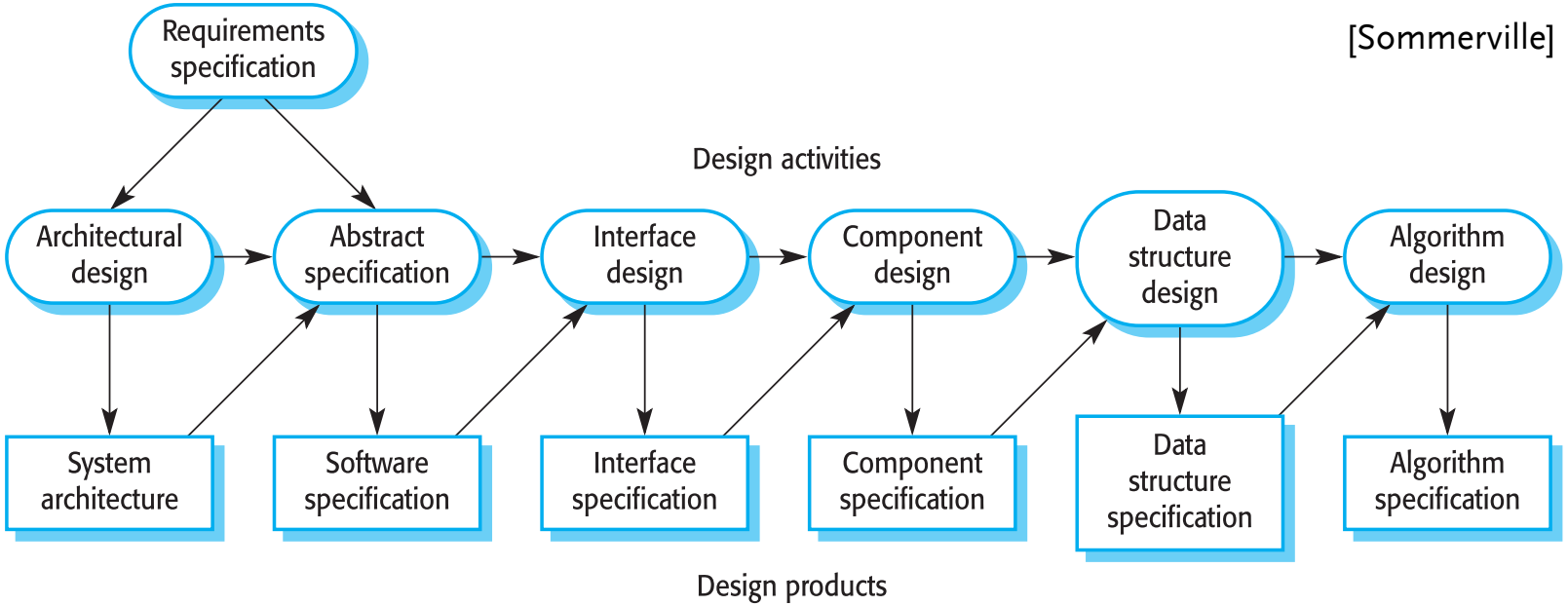


- Feasibility study, or blast-off
  - A **cheap and quick** assessment of the user needs, whether they can be satisfied, the estimated costs, identified risks, ...
- Requirements elicitation & analysis
  - **Deriving the requirements** through discussions, task analyses, observations of existing systems, prototyping, modelling, ...
- Requirements specification
  - Translate the gathered information into a **document**
- Requirements validation
  - Check for consistency, completeness and **realism**
- *More on "Requirements Engineering" in the next lecture*

Software Design: From "What?" to "How?"

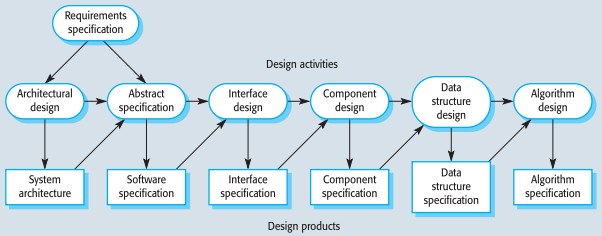
- Typically a description of the
  - **Structure** and **behaviour** of the software to be implemented
  - **Data** that is part of the system
  - **Interfaces** between system components
- Design process (often interleaved & iterated)

[Sommerville]



Software Design Phases

- *Architectural design*
  - Identify and document **subsystems** and their relationships



- *Abstract specification*
  - Decide on the **services and constraints** of each subsystem
- *Interface design*
  - Design an unambiguous **interface** for each subsystem
- *Component design*
  - Allocate services to **components** within a subsystem
- *Data structure design*
  - Design and specify the **data structures** to be used
- *Algorithm design*
  - Design and specify the **algorithms** to be used

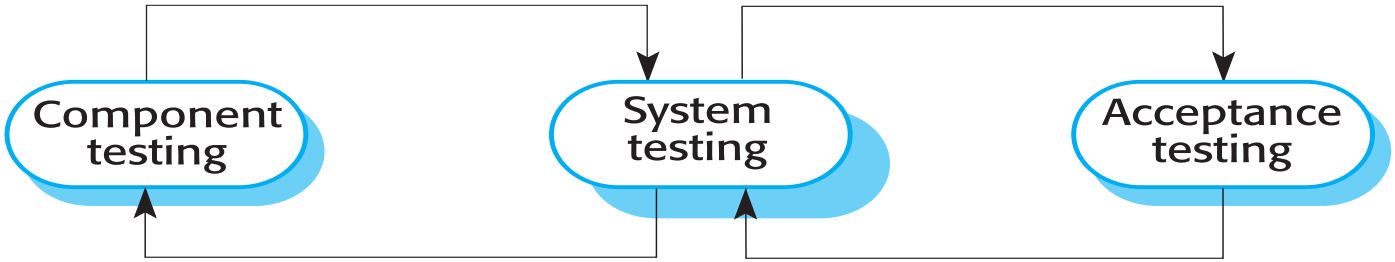
Software Implementation

- System specification is converted into an executable system, using the system's design as a sign post
- Extra designing is often needed
  - *Before and during coding, unless the design is very detailed*
  - Change the design in case of a recognised design flaw, rather than coding around it
  - **Make sure to keep implementation and design synchronized**
- Some guidelines
  - Choose a programming language according to the task at hand and the development tools and expertise available
  - **Write clear, adaptable and documented (i.e., changeable) code** (efficiency is usually less important)

Software Testing

- Context – **Verification and Validation (V&V)**
  - Ensuring that the system *conforms to its specification* and *meets the customer's expectations*
- Terminology
  - **Verification:** Are we building the product right?
  - **Validation:** Are we building the right product?
- Some popular verification and validation techniques
  - *Reviews* – of development documents and code
  - *Testing* – of code at various stages of development
  - *Model checking* – for critical system components

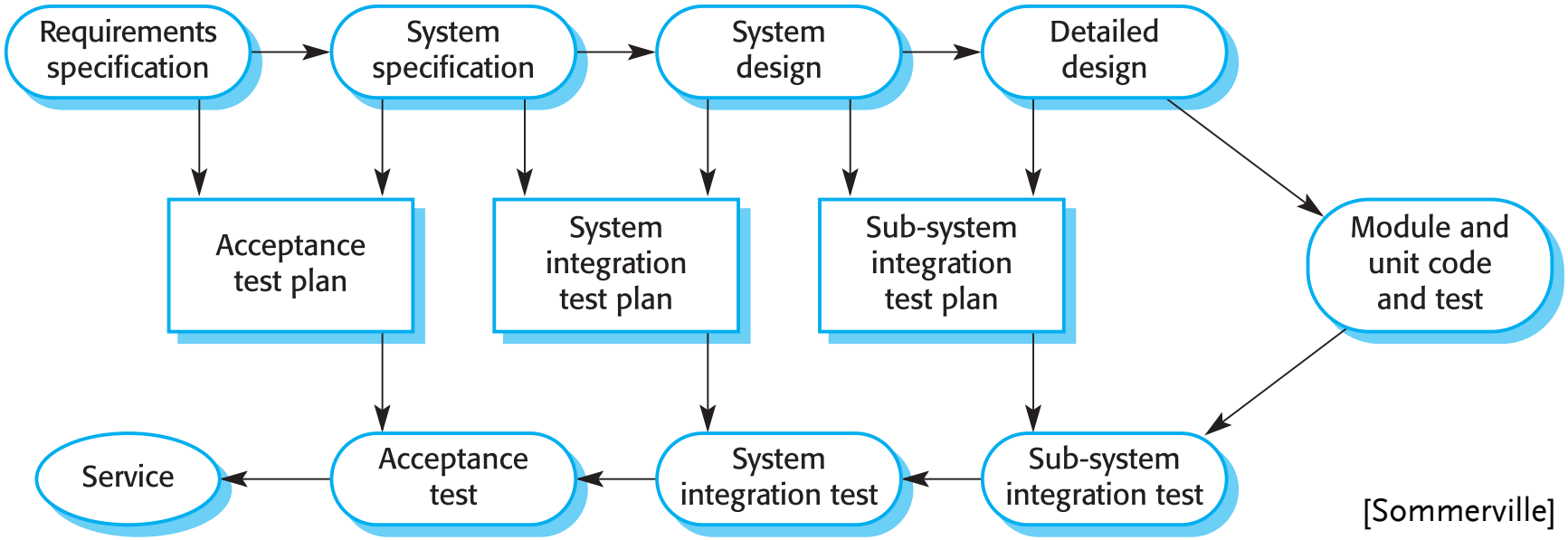
Testing Stages



[Sommerville]

- Component (or unit) testing
  - Test each component (e.g., class) **individually** to ensure its correct operation
- System (or integration) testing
  - Test **system integration** to reveal *unanticipated interactions* between components and *component interface problems*
- Acceptance testing
  - Test system using the *customer's data* rather than simulated test data

Link Between Testing & Development



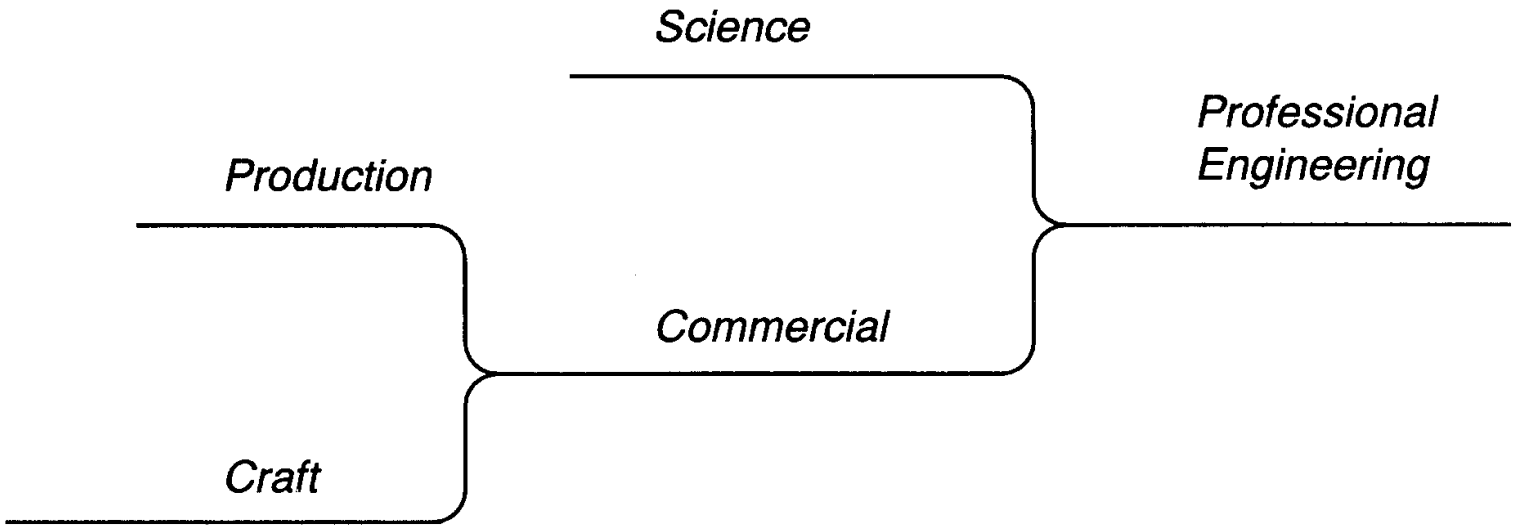
[Sommerville]

- Initially programmers test their code as it is developed, making up their own test data (see also *test-driven development*)
- Later stages of testing involve *integrating* work from a number of programmers and must be planned in advance

Deployment & Maintenance

- Deployment
  - Final validation at customer's site and with customer's data
  - *Migration of customer's data* onto the new software system
  - Sometimes parallel operation of the old and new systems
  - *Training* of users, administrators, possibly maintainers, ...
- Maintenance – the longest phase
  - Adaptation to new environments, addition of features, fixing of faults and shortcomings, ...
  - Maintenance costs are typically several times higher than the initial development costs
  - **Distinction between development and maintenance is often not relevant**

Software Engineering as a Professional Discipline



- Virtuosos and talented amateurs
- Intuition and brute force
- Haphazard progress
- Casual transmission
- Extravagant use of available materials
- Manufacture for use rather than sale

- Skilled craftsmen
- Established procedure
- Pragmatic refinement
- Training in mechanics
- Economic concern for cost and supply of materials
- Manufacture for sale

- Educated professionals
- Analysis and theory
- Progress relies on science
- Educated professional class
- Enabling new applications through analysis
- Market segmentation by product variety

[Zuser et al.]

Professional & Ethical Responsibility

- Software engineering is a responsible activity
  - More than simply the application of technical skills
  - **Imposes high demands and standards**
  - Requires a practice-oriented education
- ACM/IEEE joint code of ethics and professional practice
  - *ACM* – Association of Computing Machinery
  - *IEEE* – Institute of Electrical and Electronics Engineers, Inc.
  - "Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software *a beneficial and respected profession*. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following **eight principles** …"

ACM/IEEE Code of Conduct

PUBLIC – Software engineers shall act consistently with the public interest.

CLIENT AND EMPLOYER – Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.

PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

JUDGEMENT – Software engineers shall maintain integrity and independence in their professional judgment.

MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

ACM/IEEE Code of Conduct (cont'd)

PROFESSION – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

COLLEAGUES – Software engineers shall be fair to and supportive of their colleagues.

SELF – Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

[IEEE/ACM]

Nevertheless

You will likely be faced with ethical dilemmas in your career, e.g., whether to deliver an ill-tested but critical software component.

Plan & Summary

- Plan for this module
  - **To explore all phases and many facets of software development by means of a real-world example that will be discussed in the lectures and practicals**
  - The lectures will often be interactive, highlighting the important points
    - *Key terminology, techniques and methods will be introduced, i.e., a toolbox for developing complex software along with some suggestions on how to use it*
    - *Students are expected to deepen and supplement their knowledge by consulting additional material and textbooks*
  - The practicals will help students to check and broaden their understanding by applying the taught material to examples
- Today's summary
  - **Software engineering is a discipline that is concerned with all aspects of software production**
  - The software process includes all activities that are involved in modern software development
  - Software engineers have responsibilities to their profession and society

Selected Literature

- A. Cockburn. *Writing Effective Use Cases*. Addison Wesley, 2000.
- G. Kotonya and I. Sommerville. *Requirements Engineering – Processes and Techniques*. John Wiley, 1998.
- R.S. Pressman. *Software Engineering: A Practitioner's Approach*, 7th ed. McGraw-Hill, 2009.
- *S. Robertson and J. Robertson. Mastering the Requirements Process, 2nd ed. Addison Wesley, 2006.*
- B. Rumpe. *Modellierung mit UML*. Springer, 2004.
- **I. Sommerville. *Software Engineering*, 8th ed. Addison-Wesley, 2007.**
- *P. Stevens and R. Pooley. Using UML – Software Engineering with Objects and Components, 2nd ed. Addison-Wesley, 2005.*
- W. Zuser, T. Grechenig and M. Köhle. *Software Engineering mit UML und dem Unified Process*, 2nd ed. Pearson Studium, 2004.