Chapter 1: Introduction

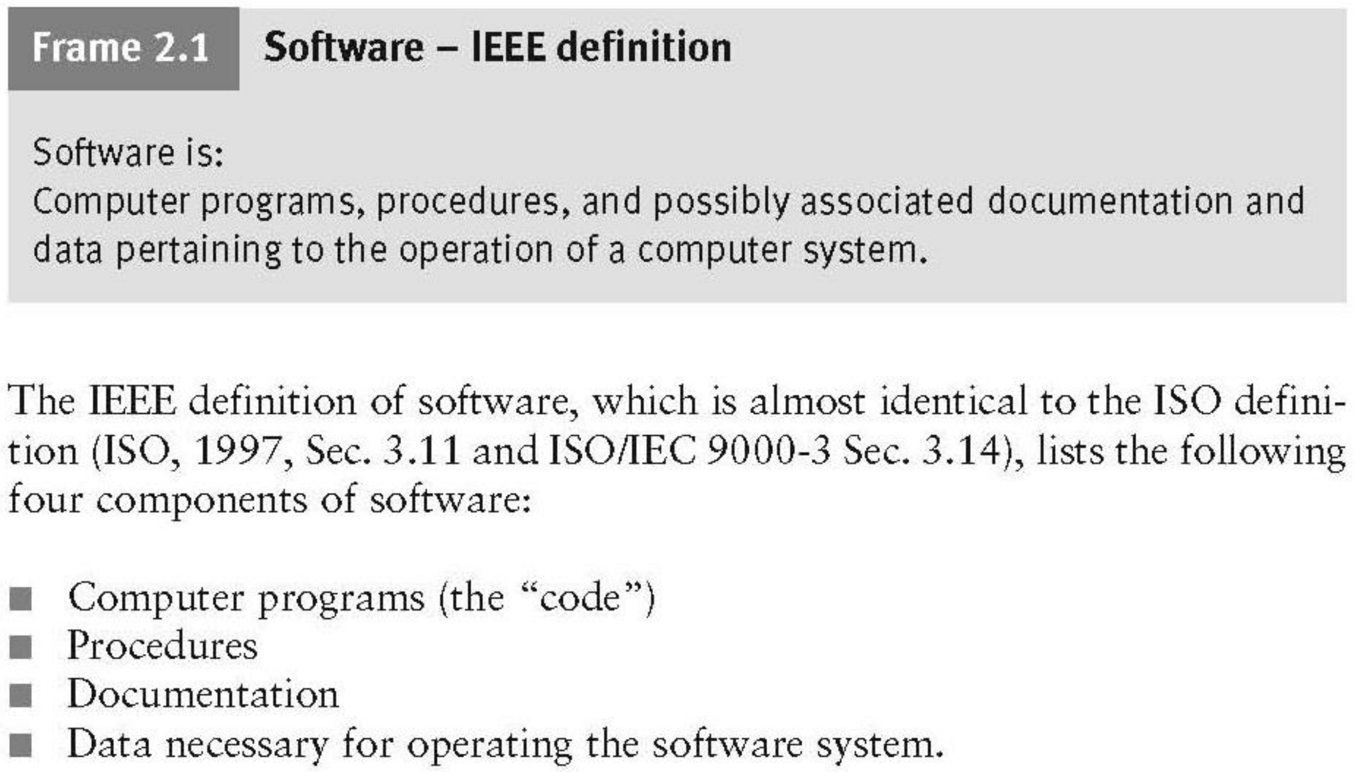
What is a computer program?

“A list of instructions, written in a specific programming language (Java, C, Fortran, etc.), which a computer follows in processing data, performing an operation, or solving a logical problem. “

What is Software?

Computer programs
Configuration files used to set up these programs
User documentation explaining how to use the software
Support service
System documentation describing the structure of the software

The Definition of Software



Frame 2.1 **Software – IEEE definition**

Software is:
Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

The IEEE definition of software, which is almost identical to the ISO definition (ISO, 1997, Sec. 3.11 and ISO/IEC 9000-3 Sec. 3.14), lists the following four components of software:

- Computer programs (the "code")
- Procedures
- Documentation
- Data necessary for operating the software system.

Hardware (Manufacturing) vs Software (Development)

Software is engineered, not manufactured.

Once a hardware product has been manufactured, it is difficult or impossible to modify. In contrast, software products are routinely modified and upgraded.

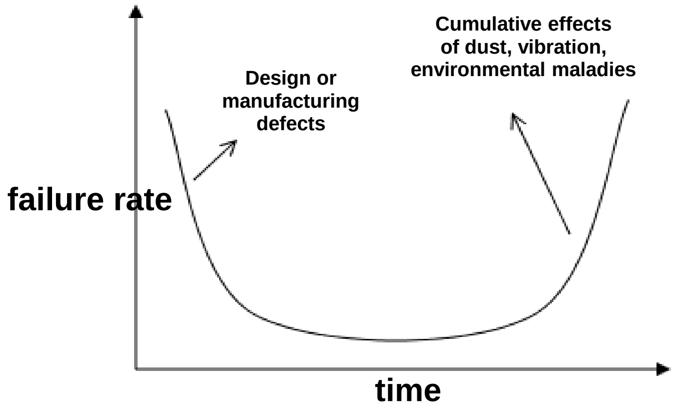
In hardware, hiring more people allows you to accomplish more work, but the same does not necessarily hold true in software engineering.

Unlike hardware, software costs are concentrated in design rather than production.

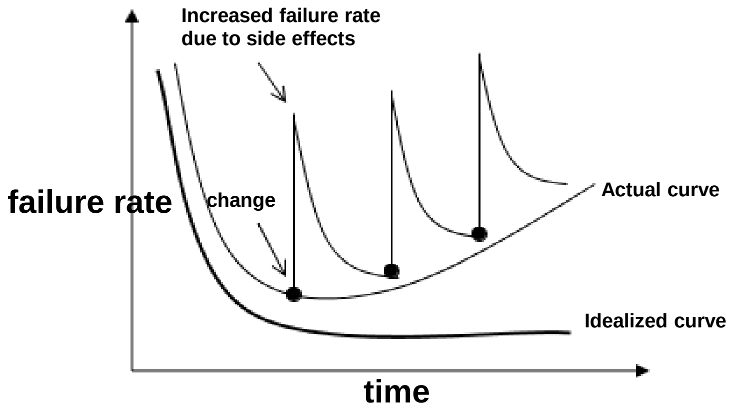
Software Deteriorates

Software does not wear-out, but it does deteriorate due to changes

Most software models a part of reality and reality evolves. If software does not evolve with the reality that is being modeled, then it deteriorates



Hardware



Software

What are the attributes of good Software?

The software should deliver the required functionality and performance to the user and

should be maintainable, dependable, efficient and usable.

- Maintainability
  - Software must (easily) evolvable to meet changing needs
- Dependability
  - Software must be trustworthy (work with all data)
- Efficiency
  - Software should not make wasteful use of system resources
- Usability
  - Software must be usable by the users for which it was designed

Where is Software?

- In computer systems
  - Operating systems (eg: Windows, Linux)
  - End-user programs (eg:Photoshop, dreamveawer)
  - Compilers (eg: javac, pascal, gcc)
- Aircrafts, Space Shuttles (Eg: F16, Discovery Space Shuttle)
- Cellular Phones (Eg: IOS, Android etc.)
- Education (Eg: Distance Learning)
- Entertainment, Transportation
- Health systems, Military
- And many more….

The economies of ALL developed nations are dependent on software.

More and more systems are software controlled

Granularity of Software

Trivial: 1 month, 1 programmer, 500 LOC

Ex: Intro programming assignments

Very small: 3 months, 1 programmer, 2000 LOC,

Ex: Course project

Small: 1 year, 3 programmers, 50K LOC,

Ex: Mobile App

Medium: 3 years, 10s of programmers, 100K LOC

Ex: Optimizing compiler

Large: 5 years, 100s of programmers, 1M LOC,

Ex: MS Word, Excel

Very large: 10 years, 1000s of programmers, 10M LOC

Ex: Air traffic control, Telecommunications, space shuttle

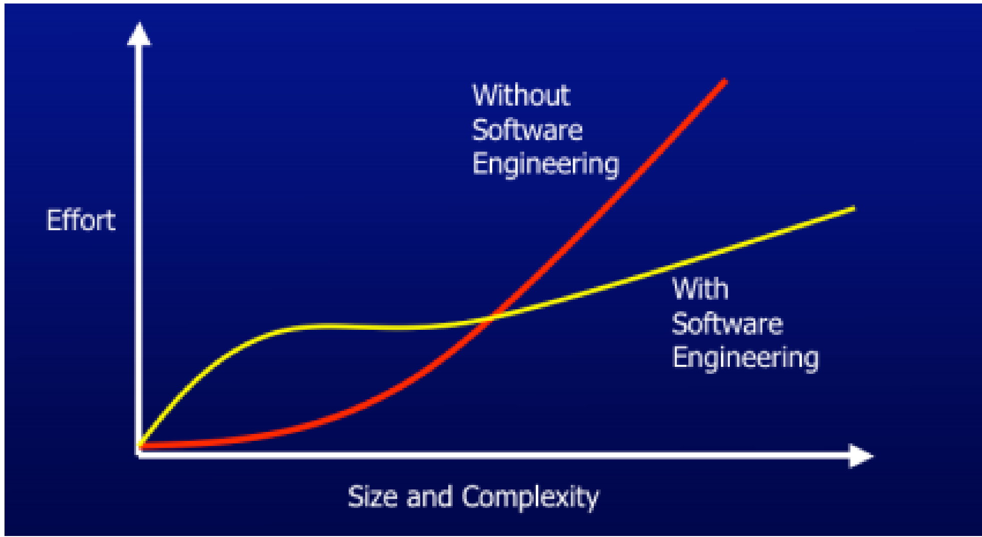
What type of software?

Small single-developer projects can typically get by without Software Engineering

- typically no deadlines, small budget (freeware), not safety-critical

Software Engineering is especially required for

- Medium to large projects (50,000 lines of code and up)
- multiple subsystems
- teams of developers (often geographically dispersed)
- safety-critical systems (software that can kill people…)



What is Software Engineering?

Software Engineering is a collection of techniques, methodologies and tools that help with the production of

A **high quality** software system developed
with a given **budget**
before a given **deadline**
while **chang**e occurs

Challenge: Dealing with complexity and change

Computer Scientist vs Software Engineer

- Computer Scientist
  - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
  - Has infinite time…
- Engineer
  - Develops a solution for an application-specific problem for a client
  - Uses computers & languages, tools, techniques and methods
- Software Engineer
  - Works in multiple application domains
  - Has only 3(?) months…
  - …while changes occurs in requirements and available technology

How successful have we been in Software Engineering?

Perform tasks more quickly and effectively

- Word processing, spreadsheets, e-mail

Support advances in medicine, agriculture, transportation, multimedia education, and most other industries

Many good stories

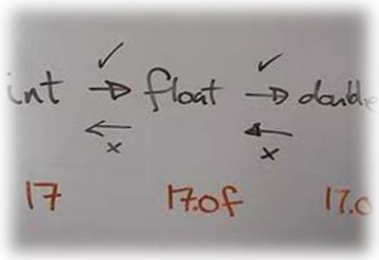
However, software is not without problems

If SW Engineering is so popular, why so many SWE disasters?

- 1985: Therac-25(radiation theraphy) lethal radiation overdose
  - Reused SW from machine with HW interlock on machine without it: SW bug => 3 died
- 1996: Ariane 5 rocket explosion
  - 1996 => wasted $370M
  - https://www.youtube.com/watch?v=gp_D8r-2hwk
- 1999: Mars Climate Orbiter disintegration
  - SW used wrong units (pound-seconds vs. metric-seconds) => wasted $325M
- 2005: FBI Virtual Case File project abandoned
  - give up after 5 years of work => wasted $170M

ARIANE Flight 501

- Disintegration after 39 sec
- Caused by wrong data being sent to On Board Computer
- Large correction for attitude deviation
- Software exception in Inertial Reference System after 36 sec.
  - Overflow in conversion of a variable from 64-bit floating point to 16-bit signed integer
  - Of 7 risky conversions, 4 were protected
  - Reasoning: physically limited, or large margin of safety
  - In case of exception: report failure and shut down



http://www.devtopics.com/20-famous-software-disasters-part-4/
http://en.wikipedia.org/wiki/List_of_software_bugs

Why Software Engineering ?

- Software failure can be very serious
  - – Software controls safety critical systems
  - – Software protects sensitive data
  - – Software is involved in systems which handle money
- Software Engineering has to
  - – Produce software which has a very low chance of faulting
  - – Be able to demonstrate/proof that software has very low chance of fault
    - Testing or program proving

Participants and Roles

- Developing software requires collaboration of many people with different backgrounds and interests.
- All the persons involved in the software project are called **participants** (stakeholders)
- The set of responsibilities in the project of a system are defined as **roles**.
- A role is associated with a set of **tasks** assigned to a participant.
- Role is also called **stakeholder**
- The same participant can fulfill multiple roles.
- Example of roles?...

Case

- TicketDistributor is a machine that distributes tickets for trains.
- Travelers have the option of selecting a ticket for a single trip or for multiple trips, or selecting a time card for a day or a week.
- The TicketDistributor computes the price of the requested ticket based on the area in which the trip will take place and whether the traveler is a child or an adult.
- The TicketDistributor must be able to handle several exceptions, such as travelers who do not complete the transaction, travelers who attempt to pay with large bills, and resource outages, such as running out of tickets, change, or power.

Roles Example

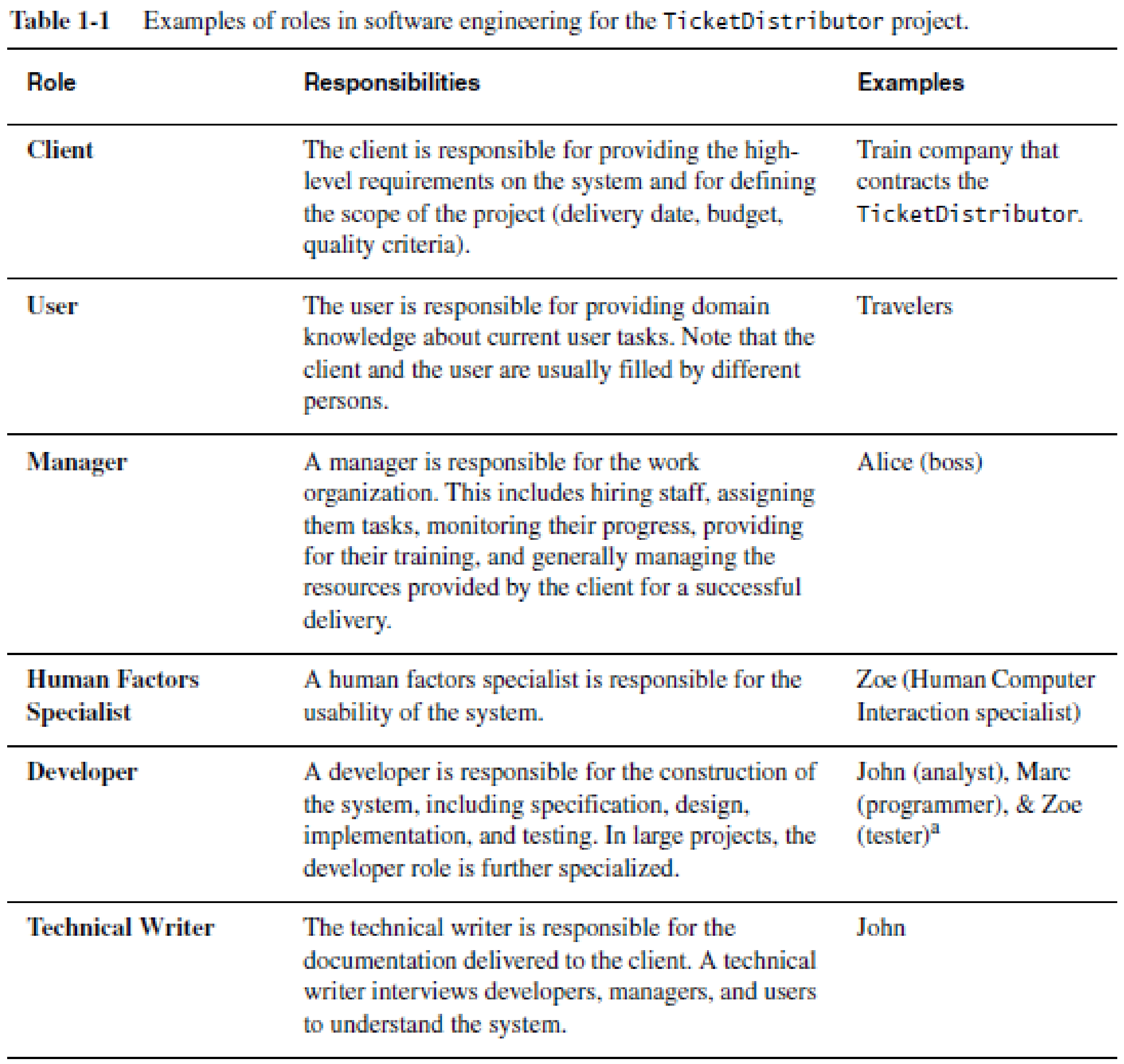


Table 1-1 Examples of roles in software engineering for the `TicketDistributor` project.

Role	Responsibilities	Examples
Client	The client is responsible for providing the high-level requirements on the system and for defining the scope of the project (delivery date, budget, quality criteria).	Train company that contracts the `TicketDistributor`.
User	The user is responsible for providing domain knowledge about current user tasks. Note that the client and the user are usually filled by different persons.	Travelers
Manager	A manager is responsible for the work organization. This includes hiring staff, assigning them tasks, monitoring their progress, providing for their training, and generally managing the resources provided by the client for a successful delivery.	Alice (boss)
Human Factors Specialist	A human factors specialist is responsible for the usability of the system.	Zoe (Human Computer Interaction specialist)
Developer	A developer is responsible for the construction of the system, including specification, design, implementation, and testing. In large projects, the developer role is further specialized.	John (analyst), Marc (programmer), & Zoe (tester)[a]
Technical Writer	The technical writer is responsible for the documentation delivered to the client. A technical writer interviews developers, managers, and users to understand the system.	John

Work Products

- Workproduct is an artifact that is produced during the development, such as documentation of software
- **Internal work product -** work product that is defined for the project’s internal use.
- **Deliverable** – work product that must be delivered to the client.
  - Deliverable are usually defined prior to the start of the project and specified in the contract

Examples?

Examples

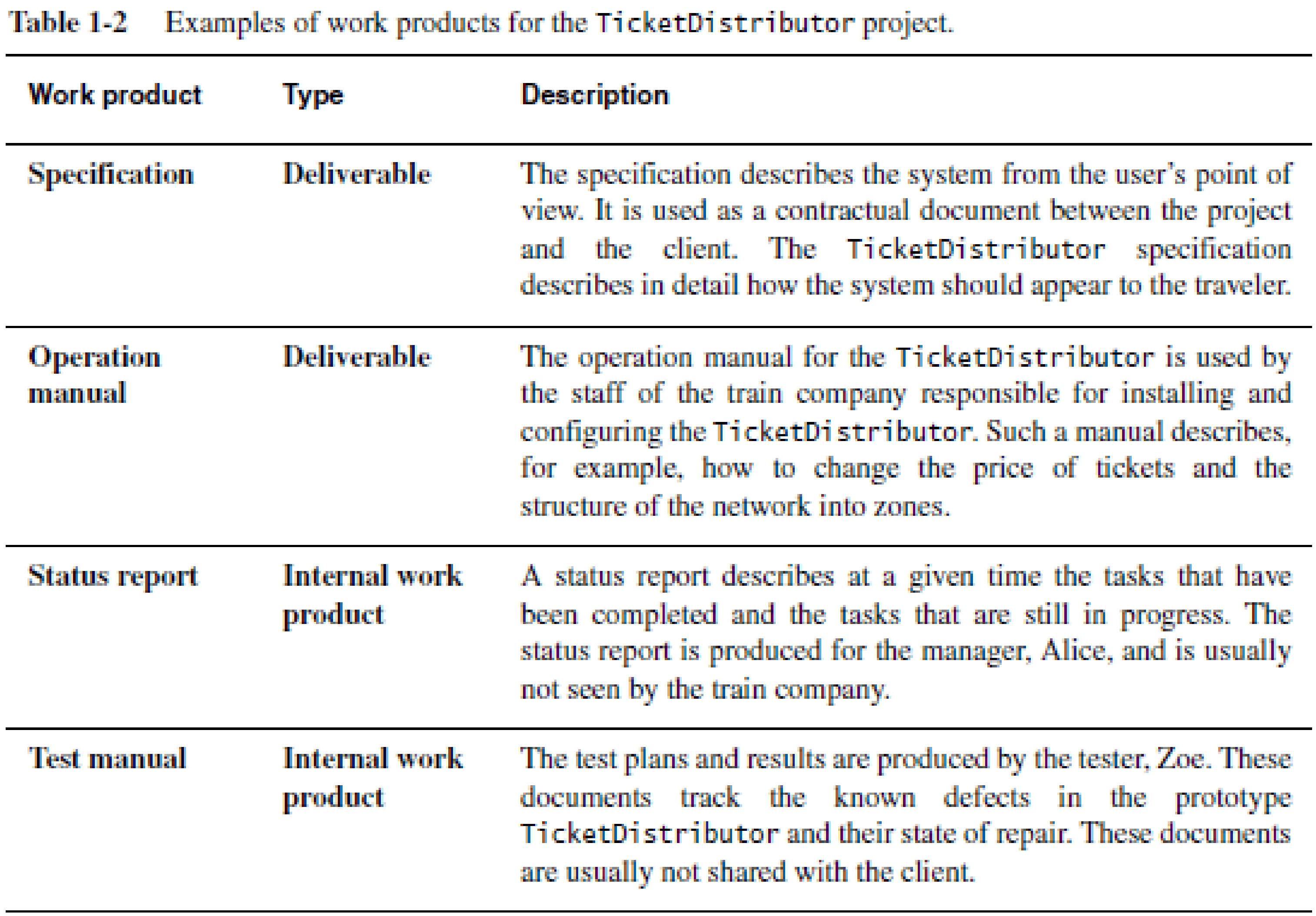
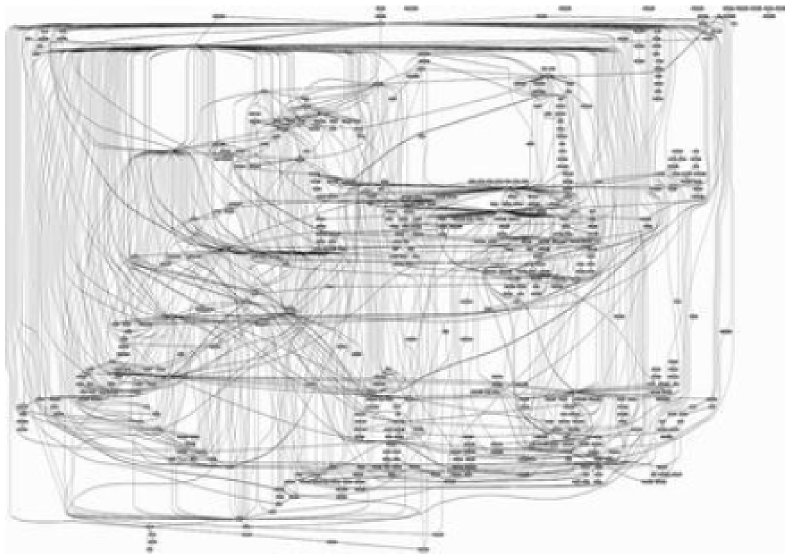


Table 1-2 Examples of work products for the TicketDistributor project.

Work product	Type	Description
Specification	**Deliverable**	The specification describes the system from the user's point of view. It is used as a contractual document between the project and the client. The `TicketDistributor` specification describes in detail how the system should appear to the traveler.
Operation manual	**Deliverable**	The operation manual for the `TicketDistributor` is used by the staff of the train company responsible for installing and configuring the `TicketDistributor`. Such a manual describes, for example, how to change the price of tickets and the structure of the network into zones.
Status report	**Internal work product**	A status report describes at a given time the tasks that have been completed and the tasks that are still in progress. The status report is produced for the manager, Alice, and is usually not seen by the train company.
Test manual	**Internal work product**	The test plans and results are produced by the tester, Zoe. These documents track the known defects in the prototype `TicketDistributor` and their state of repair. These documents are usually not shared with the client.

Why is software development difficult?

- Accidental Complexity
    - Complexity caused by the approach chosen to solve the problem
- Essential Complexity
    - Complexity that is inherent and unavoidable
- much of the accidental complexity is cleaned up with evolving approaches, however,…
- The complexity of software is an essential property, not an accidental one…
- A proper ***engineering approach is necessary***



"No Silver Bullet - Essence and Accidents of Software Engineering", Brooks, F. P., *IEEE Computer* 20, 4 (April 1987), pp. 10-19.

Why is software development difficult?

- **Change (in requirements & technology):**
  - What kind of changes?
  - The "Entropy" of a software system increases with each change: Each implemented change erodes the structure of the system which makes the next change even more expensive ("Second Law of Software Dynamics").
  - As time goes on, the cost to implement a change will be too high, and the system will then be unable to support its intended task. This is true of all systems, independent of their application domain or technological base.

Why is software development difficult?

- The problem domain (also called application domain) is difficult
- The solution domain is difficult
- The development process is difficult to manage
- Software offers extreme flexibility

Dealing with Complexity

1. Abstraction
2. Decomposition
3. Hierarchy

1. Abstraction

- Inherent human limitation to deal with complexity
  - **The 7 +- 2 phenomena**
  - Our short term memory cannot store more than 7+-2 pieces at the same time -> limitation of the brain
  - 12062156875
- Chunking: Group collection of objects
  - Group collection of objects to reduce complexity
  - 4 chunks:
    - Country-code, city-code, phone-number, Office-Part
- Ignore unessential details => Models

Model

- A model is an abstraction of a system
  - **A system that no longer exists**
  - **An existing system**
  - **A future system to be built.**

We use Models to describe Software Systems

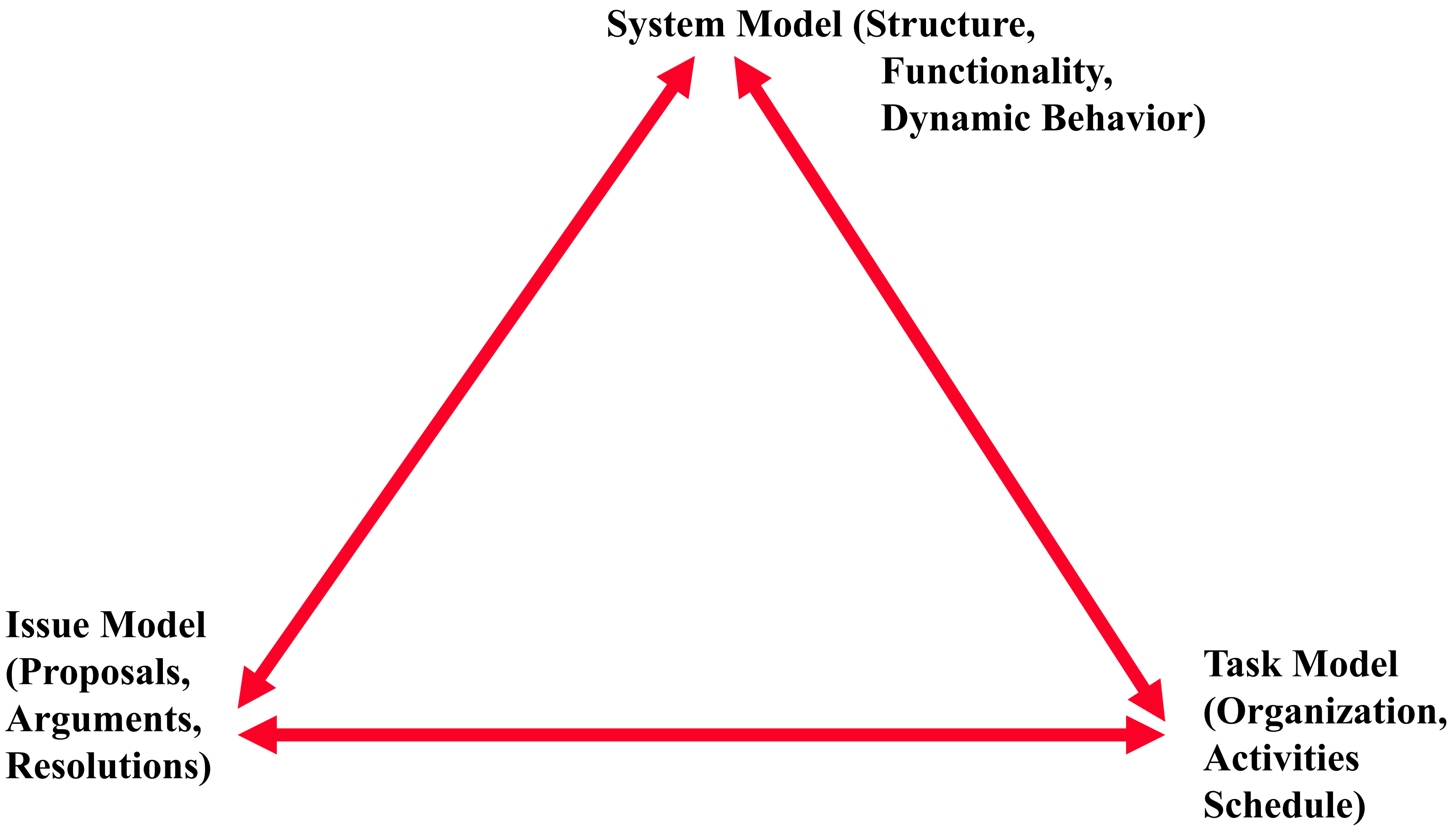
- Object model: What is the structure of the system?
- Functional model: What are the functions of the system?
- Dynamic model: How does the system react to external events?

- System Model: Object model + functional model + dynamic model

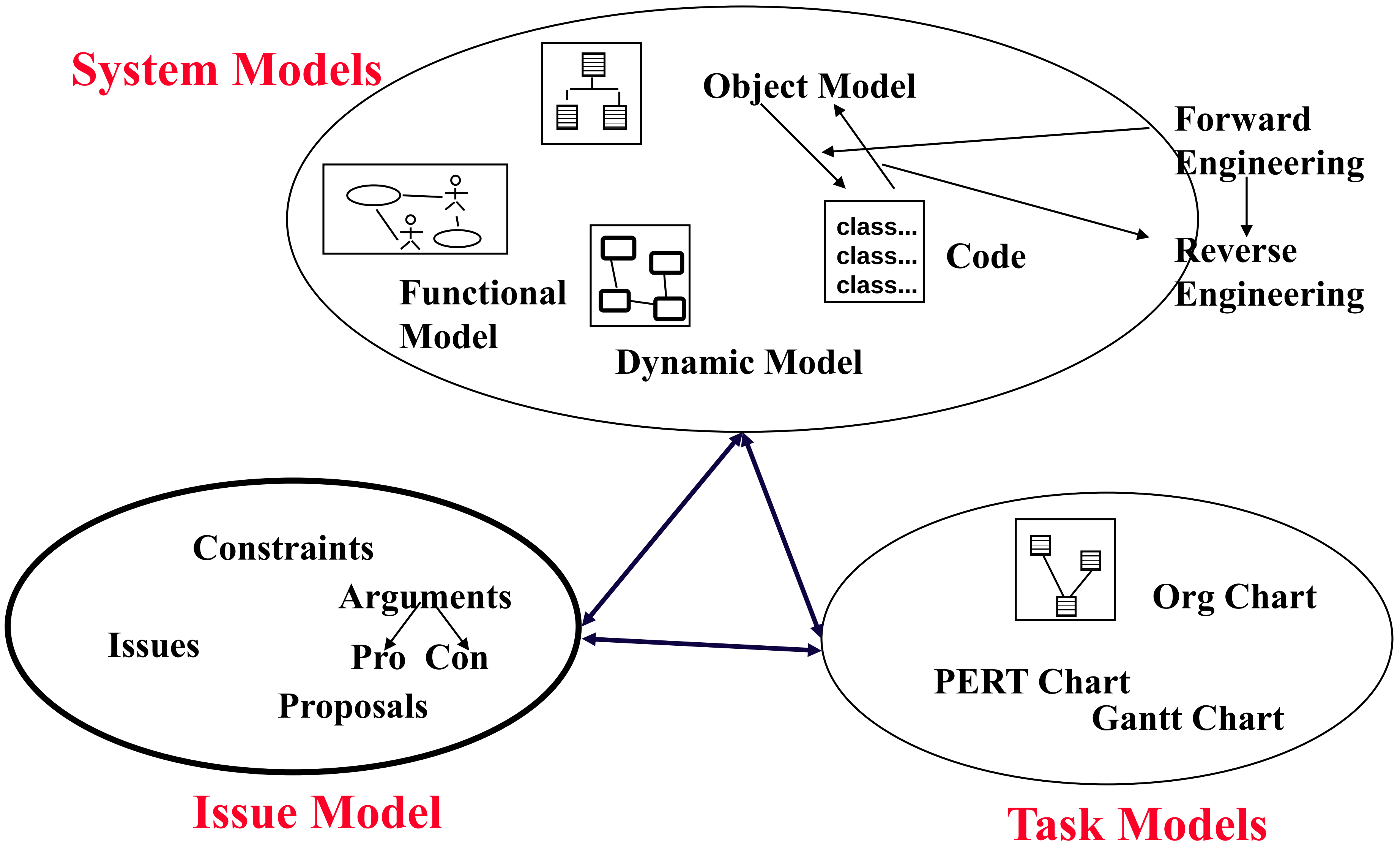
Other models used to describe Software System Development

- Task Model:
  - **PERT Chart: What are the dependencies between tasks?**
  - **Schedule: How can this be done within the time limit?**
  - **Organization Chart: What are the roles in the project?**
- Issues Model:
  - **What are the open and closed issues?**
    - **What blocks me from continuing?**
  - **What constraints were imposed by the client?**
  - **What resolutions were made?**
    - **These lead to action items**

Interdependencies of the Models


System Model (Structure, Functionality, Dynamic Behavior)
Issue Model (Proposals, Arguments, Resolutions)
Task Model (Organization, Activities Schedule)

The "Bermuda Triangle" of Modeling



2. Decomposition

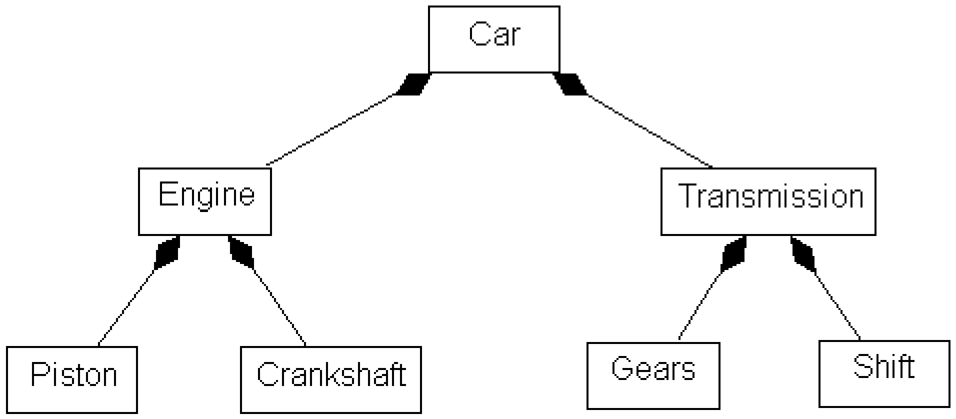
- A technique used to master complexity (“divide and conquer”)
- Functional decomposition
  - **The system is decomposed into modules**
  - **Each module is a major processing step (function) in the application domain**
  - **Modules can be decomposed into smaller modules**
- Object-oriented decomposition
  - **The system is decomposed into classes (“objects”)**
  - **Each class is a major abstraction in the application domain**
  - **Classes can be decomposed into smaller classes**

Which decomposition is the right one?

3. Hierarchy

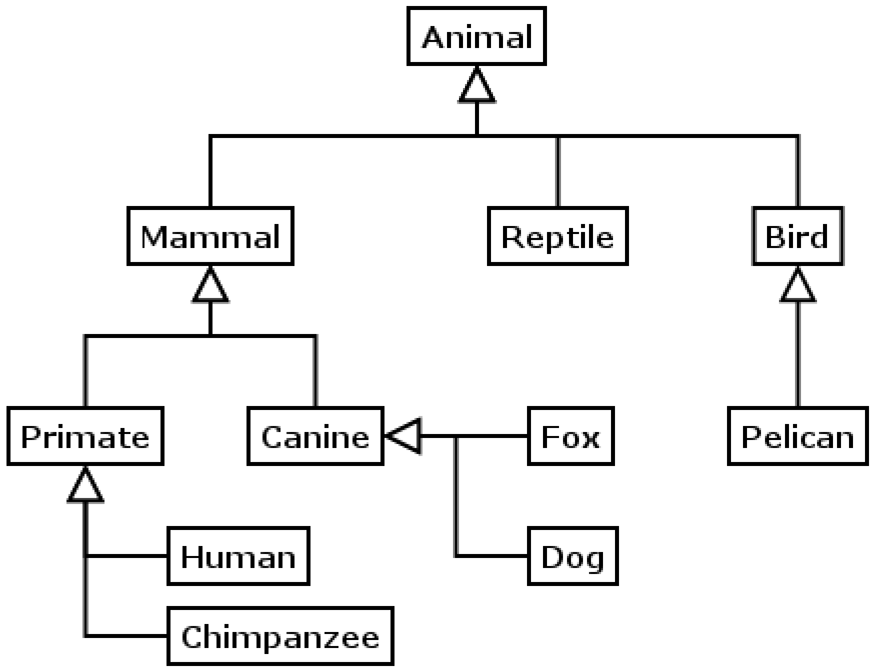
- We got abstractions and decomposition
  - **This leads us to chunks (classes, objects) which we view with object model**
- Another way to deal with complexity is to provide simple relationships between the chunks
- One of the most important relationships is hierarchy
- 2 important hierarchies
  - **"Part of" hierarchy**
  - **"Is-kind-of" hierarchy**

Part of Hierarchy



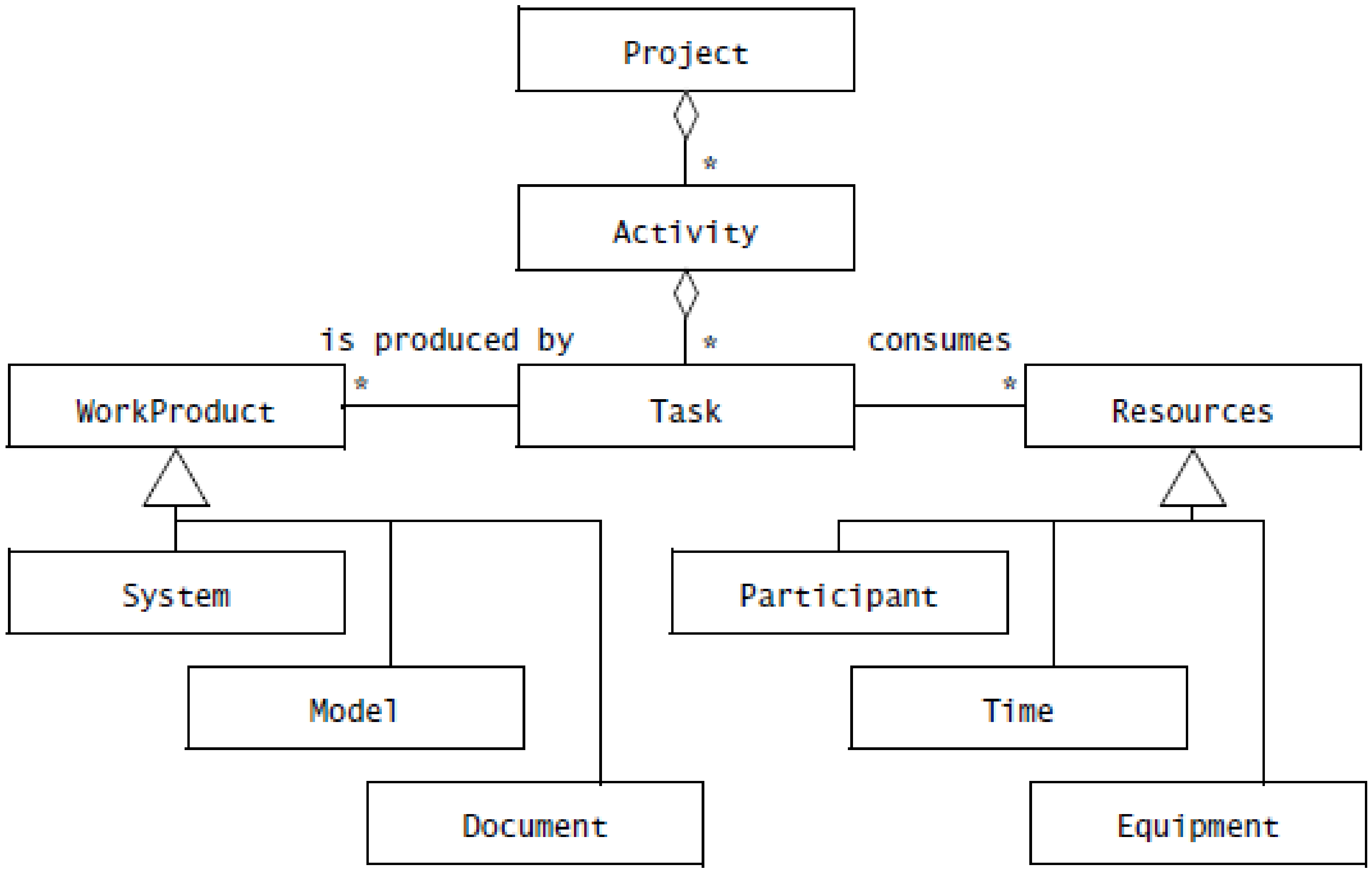
http://www.conradbock.org/relation4.html

Is-Kind-of Hierarchy (Taxonomy)

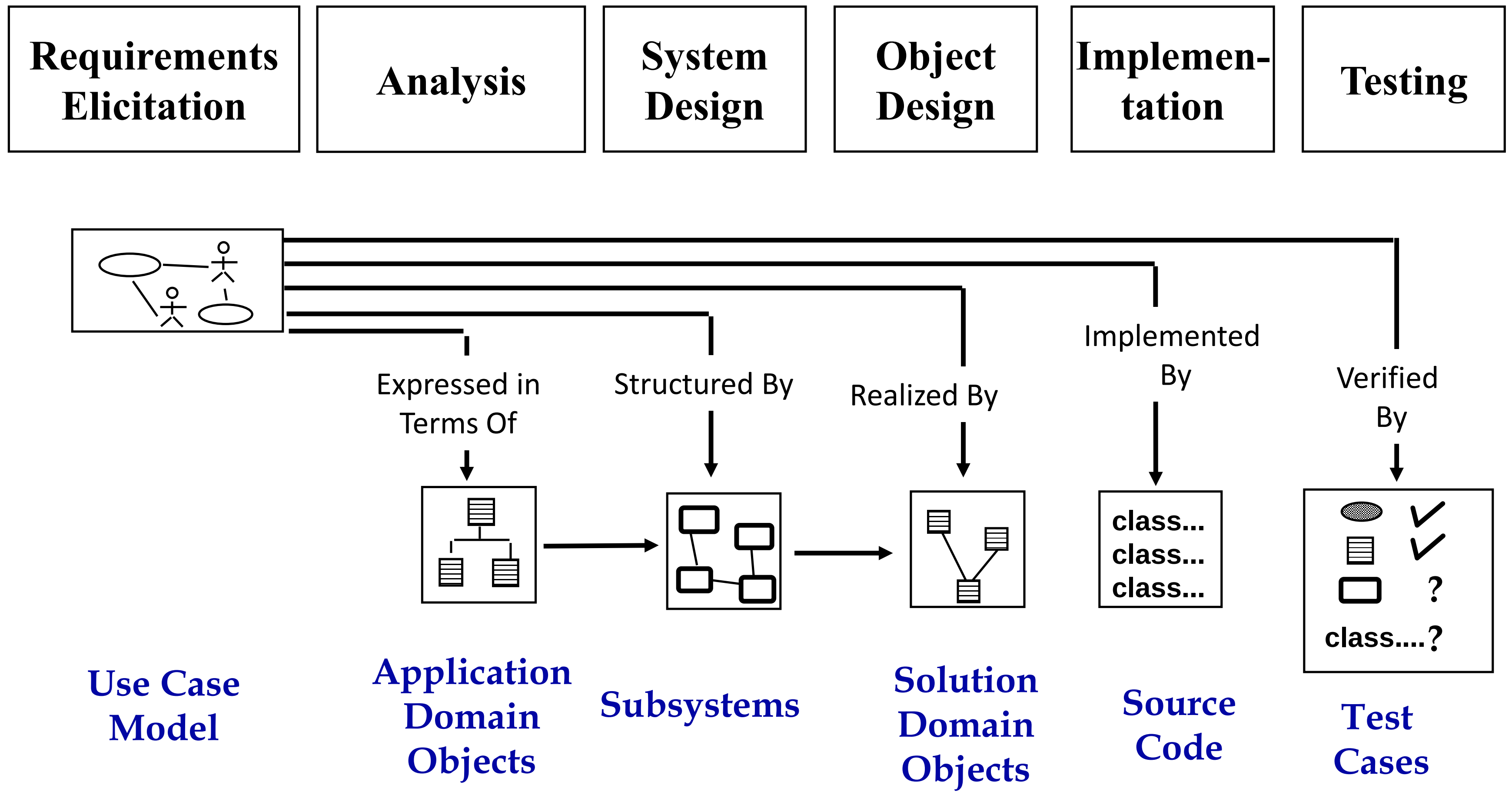


http://cs.lmu.edu/~ray/notes/devel/

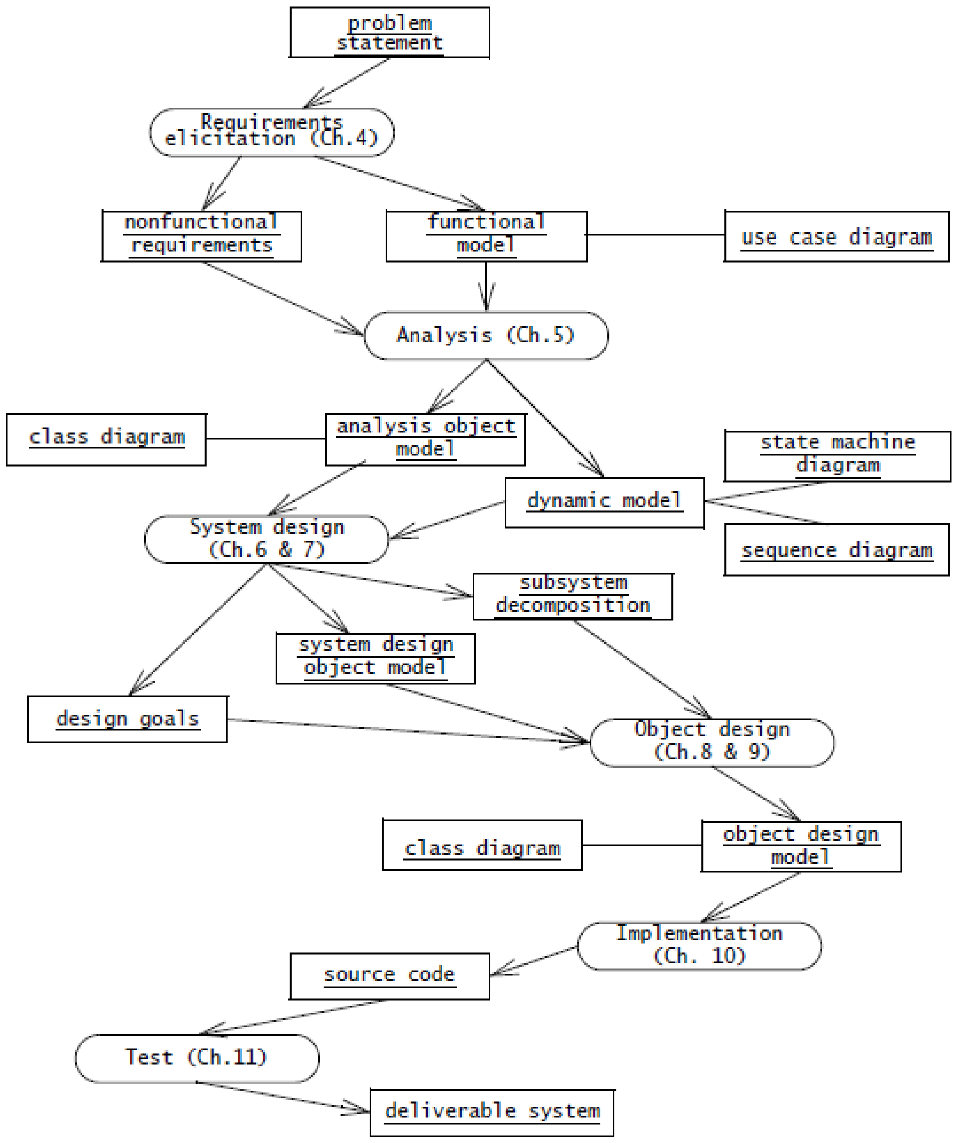
Software Engineering Concepts



Software Lifecycle Activities ...and their models



Software Lifecycle Activities



Summary

- Software engineering is a problem solving activity
  - **Developing quality software for a complex problem within a limited time while things are changing**
- There are many ways to deal with complexity
  - **Modeling, decomposition, abstraction, hierarchy**
  - **Issue models: Show the negotiation aspects**
  - **System models: Show the technical aspects**
  - **Task models: Show the project management aspects**
  - **Use patterns/styles: Reduce complexity even further**
- Many ways to deal with change
  - **Tailor the software lifecycle to deal with changing project conditions**
  - **Use a nonlinear software lifecycle to deal with changing requirements or changing technology**
  - **Provide configuration management to deal with changing entities**

Reminders

- Reading assignment
  - **Chapter 1 Introduction to Software Engineering**
  - **Chapter 2 Modelling with UML**
- Term Project
  - **Direct project related questions to your TA**
  - **Choice of projects**