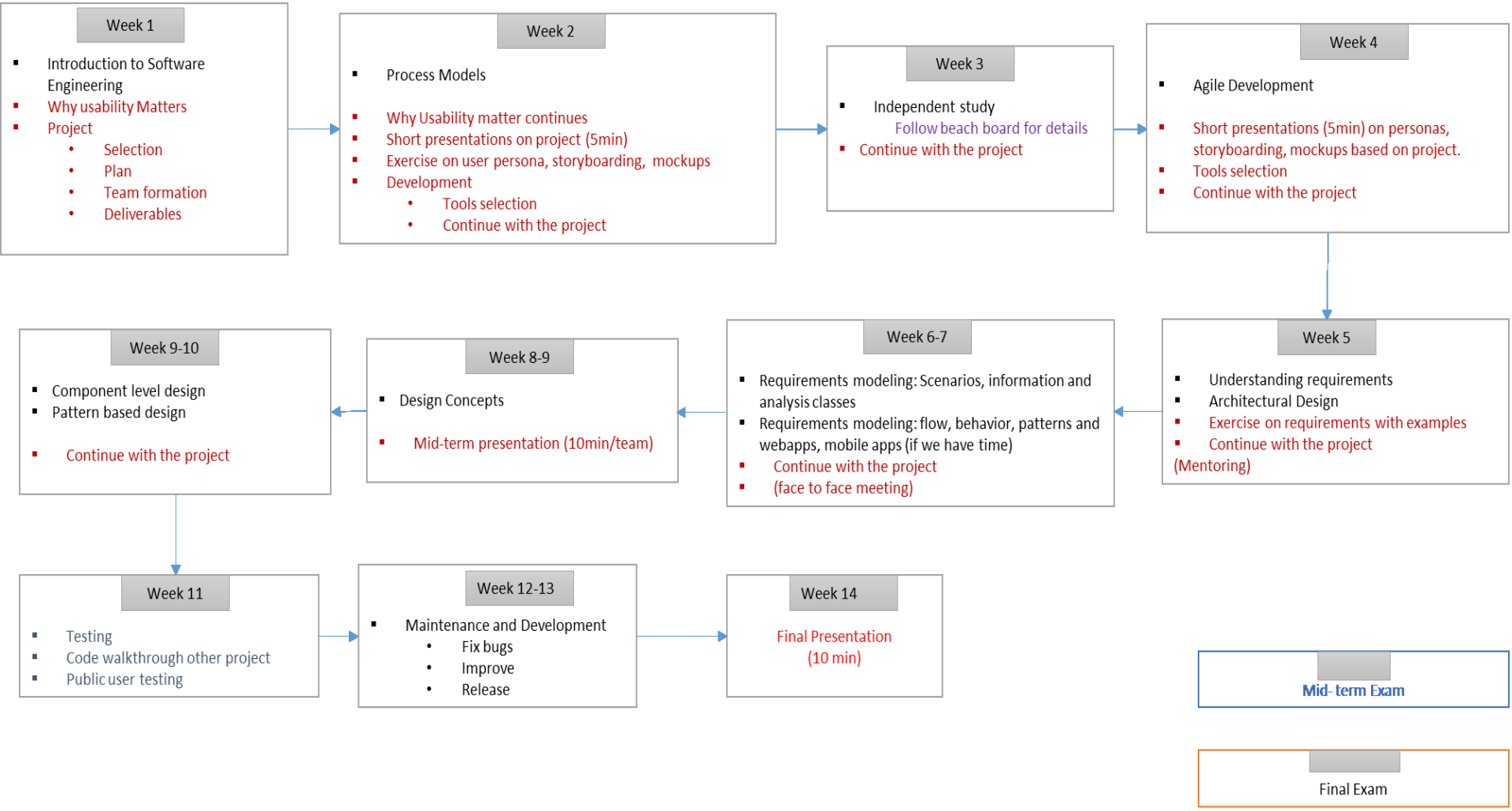
Intro to Software Engineering - Software and Software Engineering

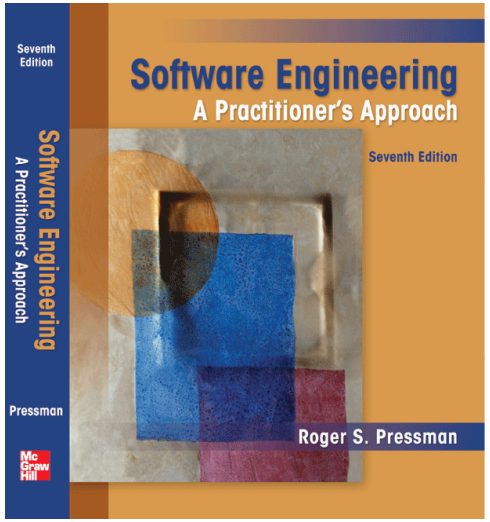
CECS 343

Timeline


Week 1
Introduction to Software Engineering
Why usability Matters
Project
Selection
Plan
Team formation
Deliverables
Week 2
Process Models
Why Usability matter continues
Short presentations on project (5min)
Exercise on user persona, storyboarding, mockups
Development
Tools selection
Continue with the project
Week 3
Independent study
Follow beach board for details
Continue with the project
Week 4
Agile Development
Short presentations (5min) on personas, storyboarding, mockups based on project.
Tools selection
Continue with the project
Week 5
Understanding requirements
Architectural Design
Exercise on requirements with examples
Continue with the project
(Mentoring)
Week 6-7
Requirements modeling: Scenarios, information and analysis classes
Requirements modeling: flow, behavior, patterns and webapps, mobile apps (if we have time)
Continue with the project
(face to face meeting)
Week 8-9
Design Concepts
Mid-term presentation (10min/team)
Week 9-10
Component level design
Pattern based design
Continue with the project
Week 11
Testing
Code walkthrough other project
Public user testing
Week 12-13
Maintenance and Development
Fix bugs
Improve
Release
Week 14
Final Presentation
(10 min)
Mid- term Exam
Final Exam

Course Outline

- Chapter 1 Software and Software Engineering
- Ch. 2 Process models
- Ch. 3 Agile development
- Ch. 4 Principles that guide practice
- Ch. 5 Understanding requirements
- Ch. 6 Requirements modeling: Scenarios, information, and analysis classes
- Ch. 7 Requirements modeling: flow, behavior, patterns and webapps
- Ch. 8 Design Concepts
- Ch. 9 Architectural Design
- Ch. 10 Component-level Design
- Ch. 12 Pattern-based Design
- Ch. 15 User Interface Design
- Ch. 17 Testing
- Ch. 29 Maintenance and Reengineering



Software and Software Engineering

1.1 The Nature of Software: Def, Domains, Legacy

1.2 Web Apps

1.3 Software Engineering

1.4 Software Process

1.5 Software Engineering Practice

1.6 Software Myths

1.7 How it all starts

1.8 Summary

Who of you has programmed?

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can’t we find all the errors before we give the software to our customers?
- Why do we spend so much time and effort maintaining existing programs?
- Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

→ These concerns are **why we do software engineering**.

1.1 The Nature of Software

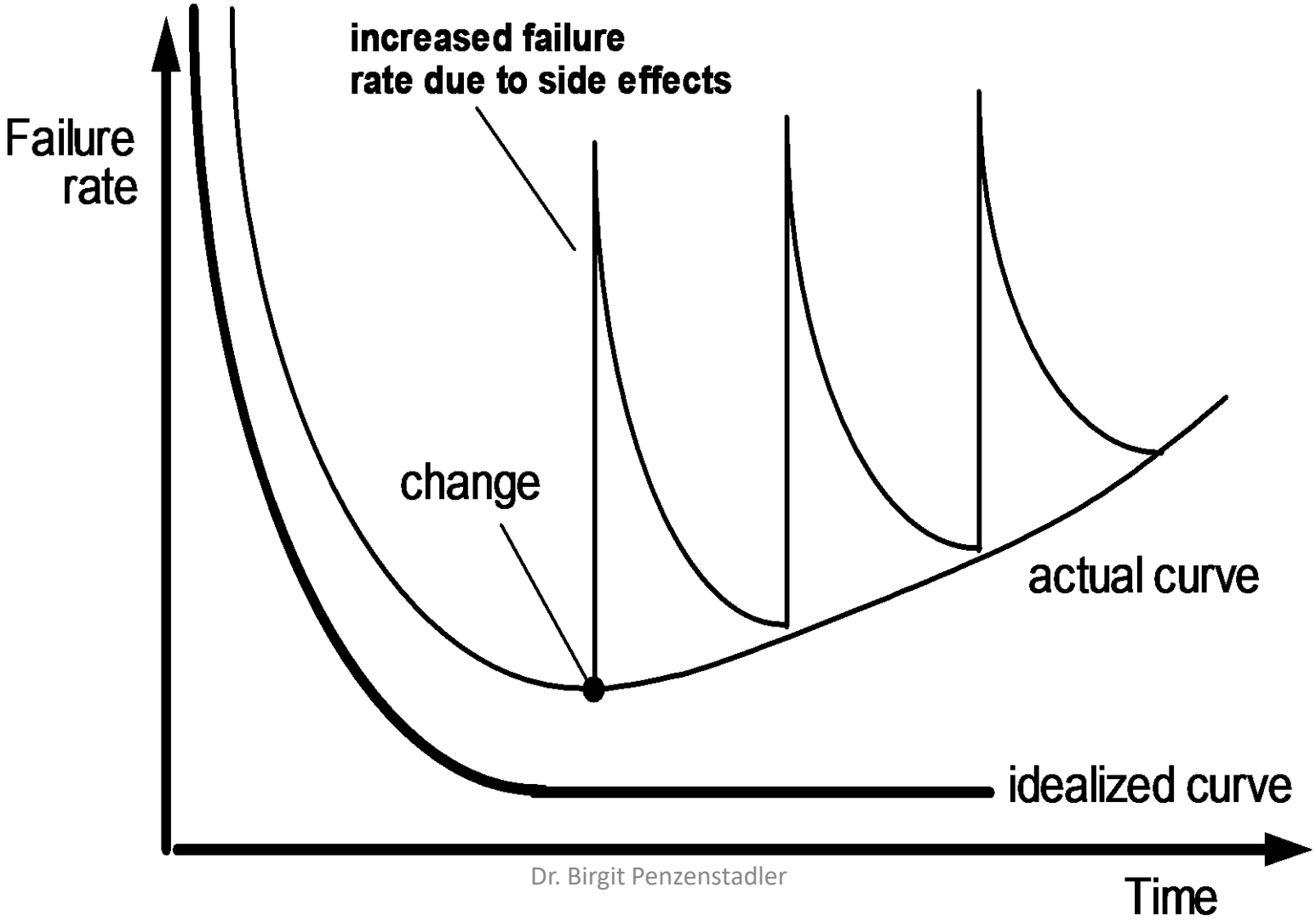
Definition: Software is

1. **instructions** (computer programs) that when executed provide desired features, function, and performance;
2. **data structures** that enable the programs to adequately manipulate information and
3. **documentation** that describes the operation and use of the programs.

What characterizes Software?

- Purpose: Software is an **information transformer**.
- Software is both a **product** and a **vehicle** that delivers a product.
- Software is developed or **engineered**, it is not manufactured in the classical sense.
- Software doesn't "wear out."
- Although the industry is moving toward component-based construction, most software continues to be custom-built.

Wear vs. deterioration



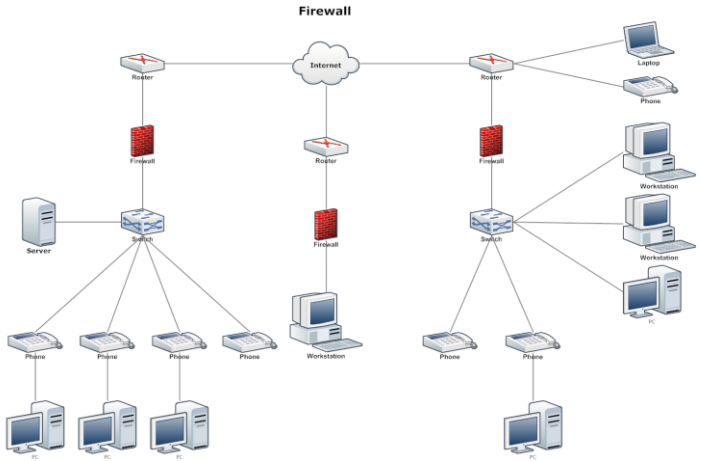
Software application domains

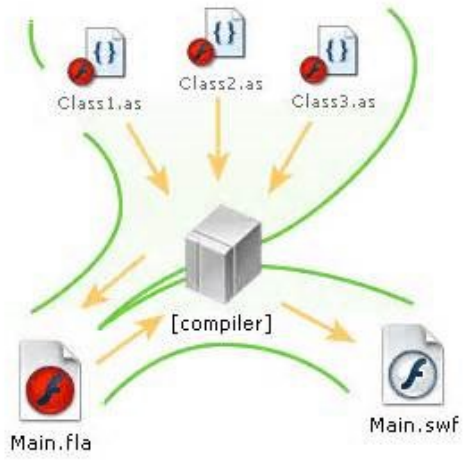
1. System software
2. Application software
3. Engineering/scientific software
4. Embedded software
5. Product-line software
6. Web, Mobile, wearable,… applications
7. Artificial intelligence software
8. Open-world computing
9. Netsourcing
10. Open source

1. System software

Operating systems, networking software, compilers, etc.

Characterized by heavy hardware interaction



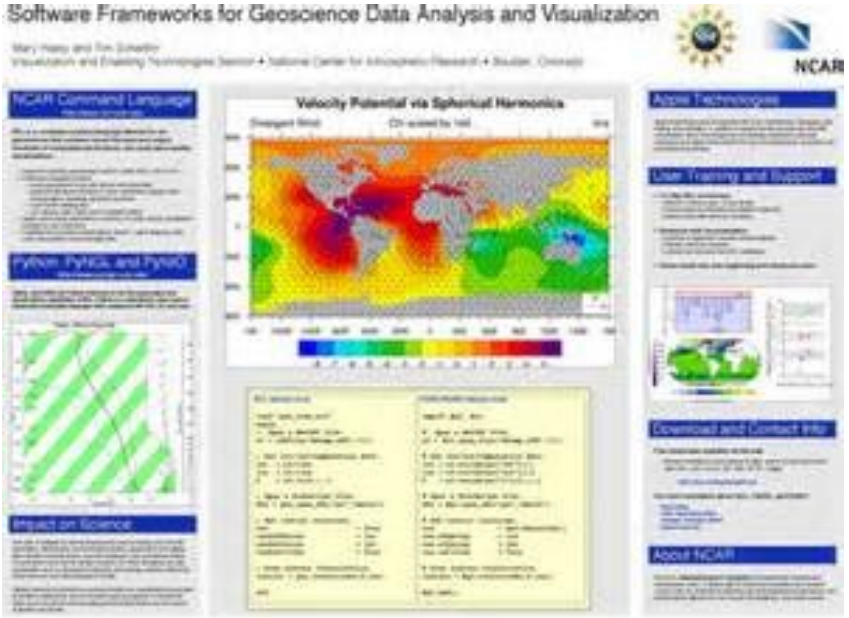


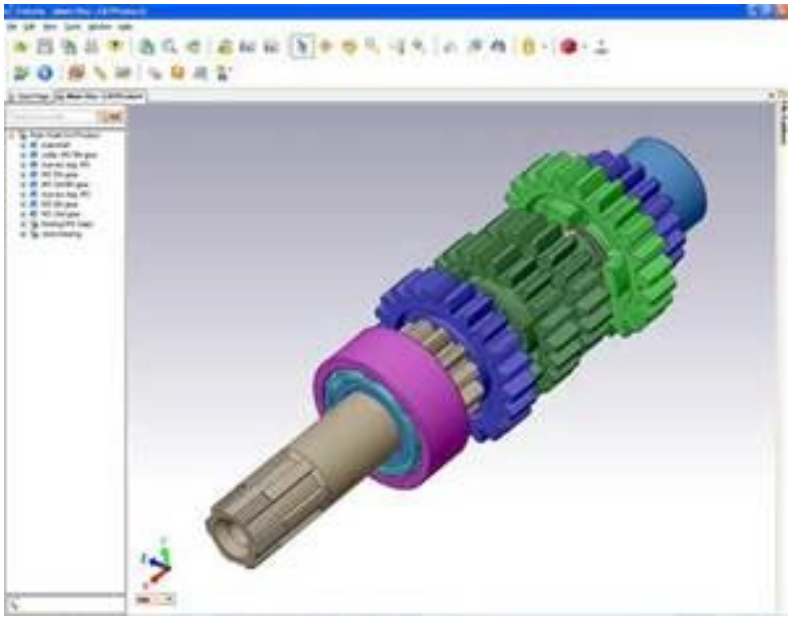
2. Application software

Stand-alone programs for a specific business need that facilitates business operations

3. Engineering/scientific software

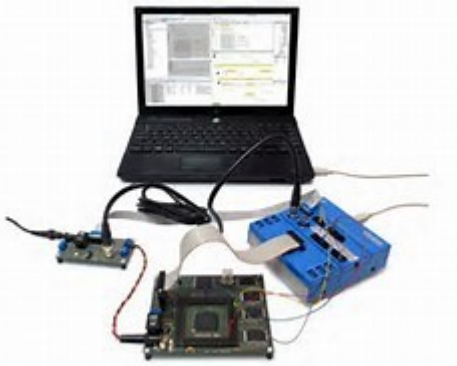
Characterized by number crunching from astronomy to volcanology, furthermore CAD and simulation.

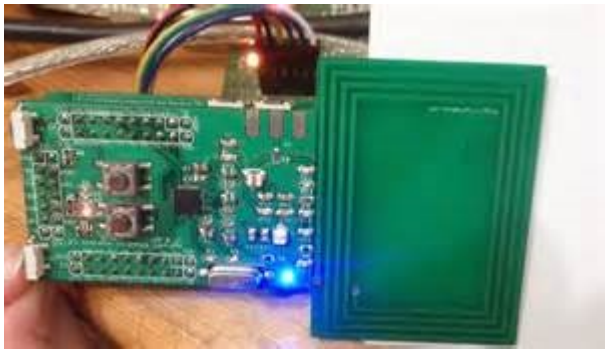




4. Embedded software

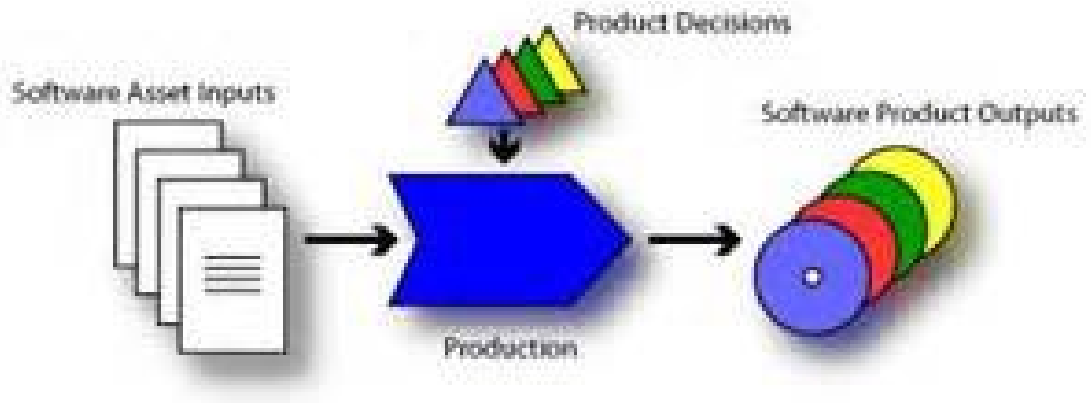
Embedded software: controls functions for end user and system, from microwave to automobile





5. Product-line software

Niche markets, e.g. inventory control, and mass consumer markets, e.g. word processing, databases, etc.

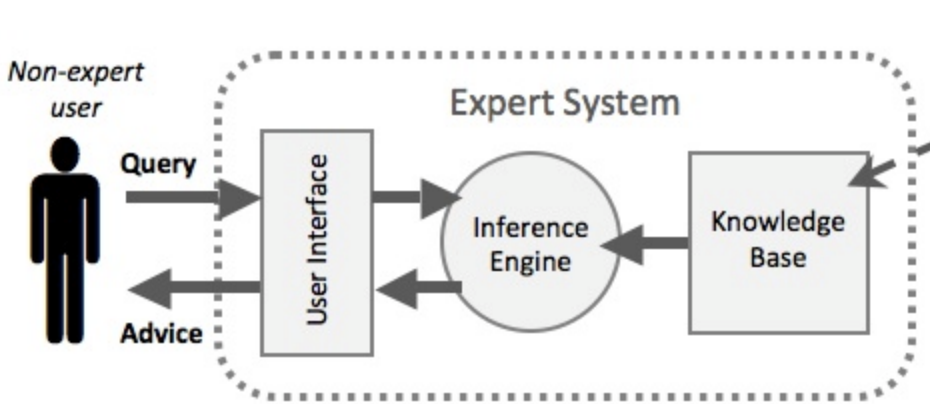


6. Web applications

Hypertext and graphics, interactive content, database and business application access

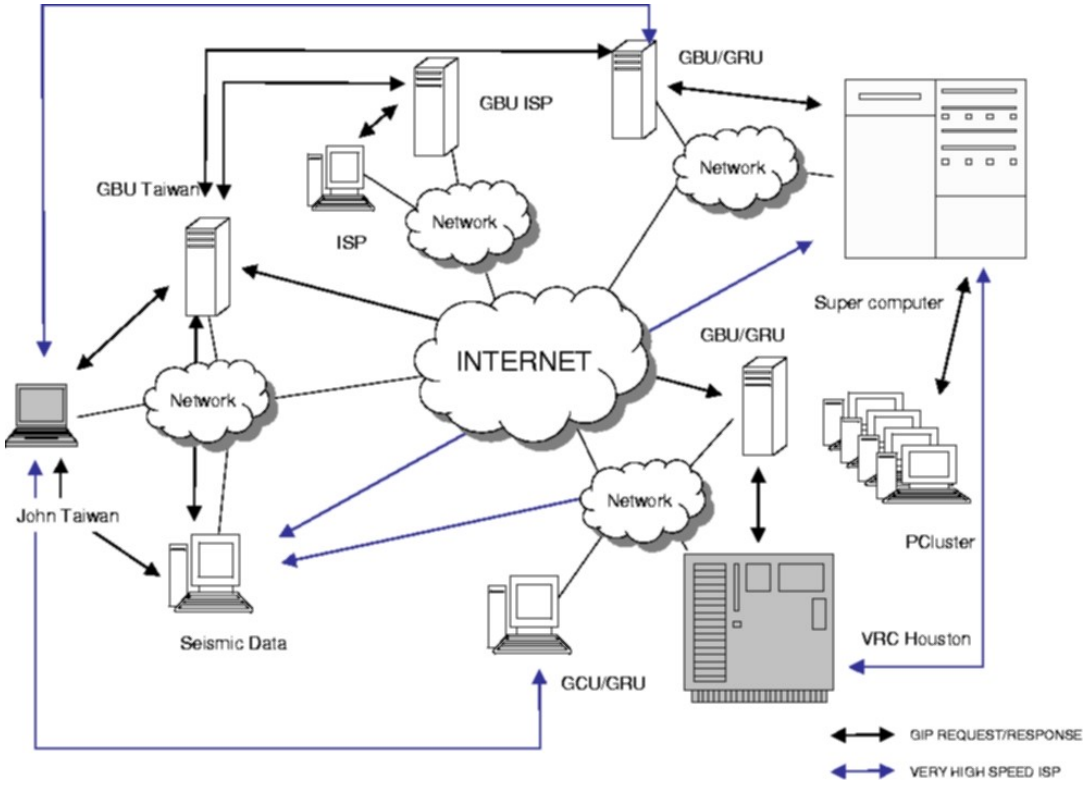
7. Artificial intelligence software

Non-numerical algorithms to solve complex tasks, e.g. robotics, expert systems, pattern recognition, game playing



8. Open-world computing

Pervasive, distributed computing that connects mobile devices, PCs, and enterprise systems via vast networks



9. Netsourcing

WWW as computing engine and content provider

10. Open source

Distribution of source code for systems applications, self-descriptive with high quality change management

Legacy Software

Legacy software systems were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms.

The proliferation of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve. [Dayani-Fard'99]

→ Why must software change?

Legacy software – why change it?

Software must be

- **adapted** to meet the needs of new computing environments or technology.
- **enhanced** to implement new business requirements.
- **extended** to make it interoperable with other more modern systems or databases.
- **re-architected** to make it viable within a network environment.

1.2 Web Apps

- “Early days” ‘90-’95: hypertext and graphics
- Today: web-based systems and applications
- Characteristics:
  - Network intensiveness
  - Concurrency
  - Unpredictable load
  - Performance
  - Availability
  - Data driven
  - Content sensitive
  - Continuous evolution
  - Immediacy
  - Security
  - Aesthetics

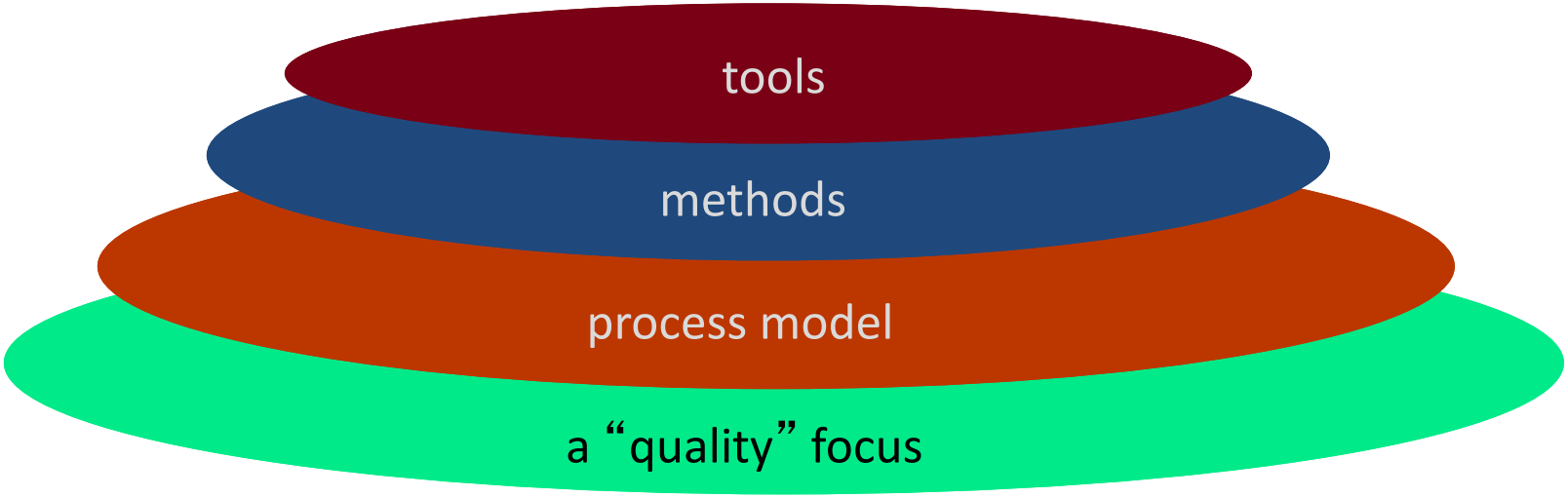
1.3 Software Engineering (SE)

- For building software it is important to recognize:
  1. Understand the problem before you build a solution
  2. Design is a pivotal SE activity
  3. Both quality and maintainability are an outgrowth of good design

Definition of Software Engineering

- Fritz Bauer (1969): [Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.
- IEEE (1993): Software Engineering:
  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.

Software Engineering Layers



Software Engineering

1.4 The Software Process

- Informal: A process defines who is doing what when and how to reach a certain goal.
- A little more formal: A process is a collection of activities, actions, and tasks that are performed when a work product is to be created.
  An activity strives to achieve a broader objective, an action encompasses a set of tasks for a work product, and a task focuses on a small, well-defined objective.

Process framework

- Framework activities (engineering activities)
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment
- Umbrella activities (management activities)
  - Software project management
  - Risk management
  - Software quality assurance
  - Formal technical reviews
  - Measurement
  - Software configuration management
  - Reusability management
  - Work product preparation and production

1.5 Software Engineering Practice

The essence of practice on **problem solving** from "How to solve it" by George Polya (1945):

1. Understand the problem (communication and analysis).
2. Plan a solution (modeling and design).
3. Carry out the plan (generation).
4. Examine the result for accuracy (quality assurance).

Understand the Problem

- *Who has a stake in the solution to the problem?* That is, who are the stakeholders?
- *What are the unknowns?* What data, functions, and features are required to properly solve the problem?
- *Can the problem be compartmentalized?* Is it possible to represent smaller problems that may be easier to understand?
- *Can the problem be represented graphically?* Can an analysis model be created?

Plan the Solution

- *Have you seen similar problems before?* Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- *Has a similar problem been solved?* If so, are elements of the solution reusable?
- *Can subproblems be defined?* If so, are solutions readily apparent for the subproblems?
- *Can you represent a solution in a manner that leads to effective implementation?* Can a design model be created?

Carry Out the Plan

- *Does the solution conform to the plan?* Is source code traceable to the design model?
- *Is each component part of the solution provably correct?* Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

Examine the Result

- *Is it possible to test each component part of the solution?* Has a reasonable testing strategy been implemented?
- *Does the solution produce results that conform to the data, functions, and features that are required?* Has the software been validated against all stakeholder requirements?

General principles [Hooker'96]

1. The reason it all exists
2. KISS: Keep it simple, stupid!
3. Maintain the vision.
4. What you produce, others will consume.
5. Be open to the future.
6. Plan ahead for reuse.
7. Think!

1.6 Software myths

1. We have a book of standards… all we need?
2. If we get behind schedule, add more coders…
3. A general objective is enough to start, we'll fill in details later.
4. Once the code works, our job is done.
5. SE will slow us down with lots of documentation.

1.7 How it all starts

- Every software project is precipitated by some business need—
  - the need to correct a defect in an existing application;
  - the need to the need to adapt a ‘legacy system’ to a changing business environment;
  - the need to extend the functions and features of an existing application, or
  - the need to create a new product, service, or system.

1.8 Summary

- What characterizes software?
- What categories of software are there?
- What is Software Engineering?
- How to define a process?
- What is the essence of problem solving?
- What are general principles for SE?

Lab

Are you ready?