SYSC-3120 —Software Requirements Engineering

Software Engineering Preview

Software Engineering Preview

- Definitions
- Software Failures
- History and Context
- Software Development Myths
- Principles
- Software Development Processes
- Software Development Tools
- Summary

Definitions of SW Engineering

- Software Engineering [IEEE-93]:
  - The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
  - Highlights the difference between programming and software engineering
- Canadian Standards Association:
  - “The systematic activities involved in the design, implementation and testing of software to optimize its production and support”

Definitions of SW Engineering (cont.)

- Parnas (1987):
  - “Multi-person construction of multi-version software”
  - Software Engineering means the construction of quality software with a limited budget and a given deadline in the context of constant change
- Lethbridge (2004):
  - “Software engineering is the process of solving the customers’ problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints.”

Definitions of SW Engineering: Conclusion

- Important notions to remember:
  - Systematic, disciplined, quantifiable
  - Development, operation, and maintenance (not only construction/coding)
  - Quality
  - Constraints

The Engineer in SW Engineering

- Engineers design products following well-accepted practices which normally involve the application of science, mathematics and economics

- As professionals, engineers assume a duty of personal responsibility to the public and society, and a code of ethics.
  - The society includes the customer (ie. meeting economic and time constraints)

Scope of SW Engineering

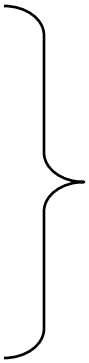
- SE is part of a much larger *system* design activity
  - Telephone switching systems, power plants, banking systems, hospital admin systems, aircraft
- Doing SE right requires a much larger look at *system* engineering issues
  - entities and activities within the *system*, its boundary and interface with other *systems* and users
- Understanding the application and user needs is key
  - Decide what activity should be supported by the *system* and how
  - Having a technical understanding of the *system* to be developed is not enough
  - Many domains, where very different software *systems* must be developed, with emphasis on different priorities:
    - time-to-market (telecom), safety (aerospace, NASA Shuttle), maintainability (telecom, banking)

Importance of SW Engineering

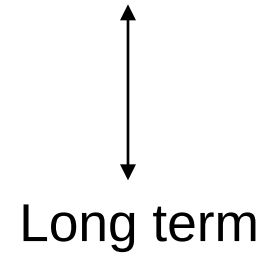
- Software is pervasive in our lives, in most systems surrounding us - we take it for granted!
- US $500 Billion world-wide in 1995
- This includes critical systems that affect our well-being and our lives
- Many reported stories of poor software engineering practices leading to catastrophes

Software Quality

- External Characteristics (of interest to stakeholders)
  - Usability
  - Efficiency
  - Reliability
  - Maintainability
  - Reusability



Engineering is tradeoffs



- Internal Characteristics (impact maintainability and reliability)
  - Comments
  - Code Complexity: Nesting depth, branches, complex programming
  - Modularity

Software Engineering Stakeholders

- Developers are only one of the stakeholders in a SE Project
- Users: Use the end-product
  - Appreciate software that is easy to learn, improves their working conditions
- Customers: Order and pay for the software
  - Increase profits or run business better
- Development Managers: Manage the developers
  - Please the customer while spending the least money.
- One person may take on multiple roles.

Activities Involved in SE

- *Knowledge acquisition*:
  - Understand the application domain, the system requirements
  - Knowledge acquisition is not sequential, as a single piece of additional information can invalidate complete models

- *Modeling* (the blue-print of the software engineer):
  - Way to cope with complexity by raising the level of abstraction, e.g., UML

- *Problem solving*:
  - Find an acceptable solution within constraints (budgets and timelines)
  - Find -> search -> experiment (i.e., compare alternatives, evaluate)

- *Documentation*:
  - The rationale behind decisions need to be captured, in order to be able to deal with change

Software Engineering Preview

- Definitions
- Software Failures
- History and Context
- Software Development Myths
- Principles
- Software Development Processes
- Software Development Tools
- Summary

Examples of SE Failures

- *Patient Protection and Affordable Care Act* (a.k.a. ObamaCare, 2013):
  - Incorrect functionality, unable to handle the load due to poor specifications, flawed design, poor coding, poor testing, security flaws, time constraints.
- *Soyuz spacecraft's descent from the ISS on May 3rd 2003*
  - Halfway back to Earth, for no apparent reason, the computer had suddenly begun searching for the ISS as if to dock with it.
- *Ariane 5 Flight 501*:
  - The space rocket was destroyed. Cause: poor specifications, usage testing, and exception handling.
- *Therac-25*:
  - Radiation therapy and X-ray machine killed several patients. Cause: unanticipated, non-standard user inputs.
- *NASA mission to Mars* (Mars Climate Orbiter Spacecraft, 1999):
  - Incorrect conversion from imperial→metric leads to loss of Mars satellite

US study (1995):

  - 81 billion US$ spend per year for failing software development projects
- Despite many success stories, there is much room for improvement.

Software Engineering Preview

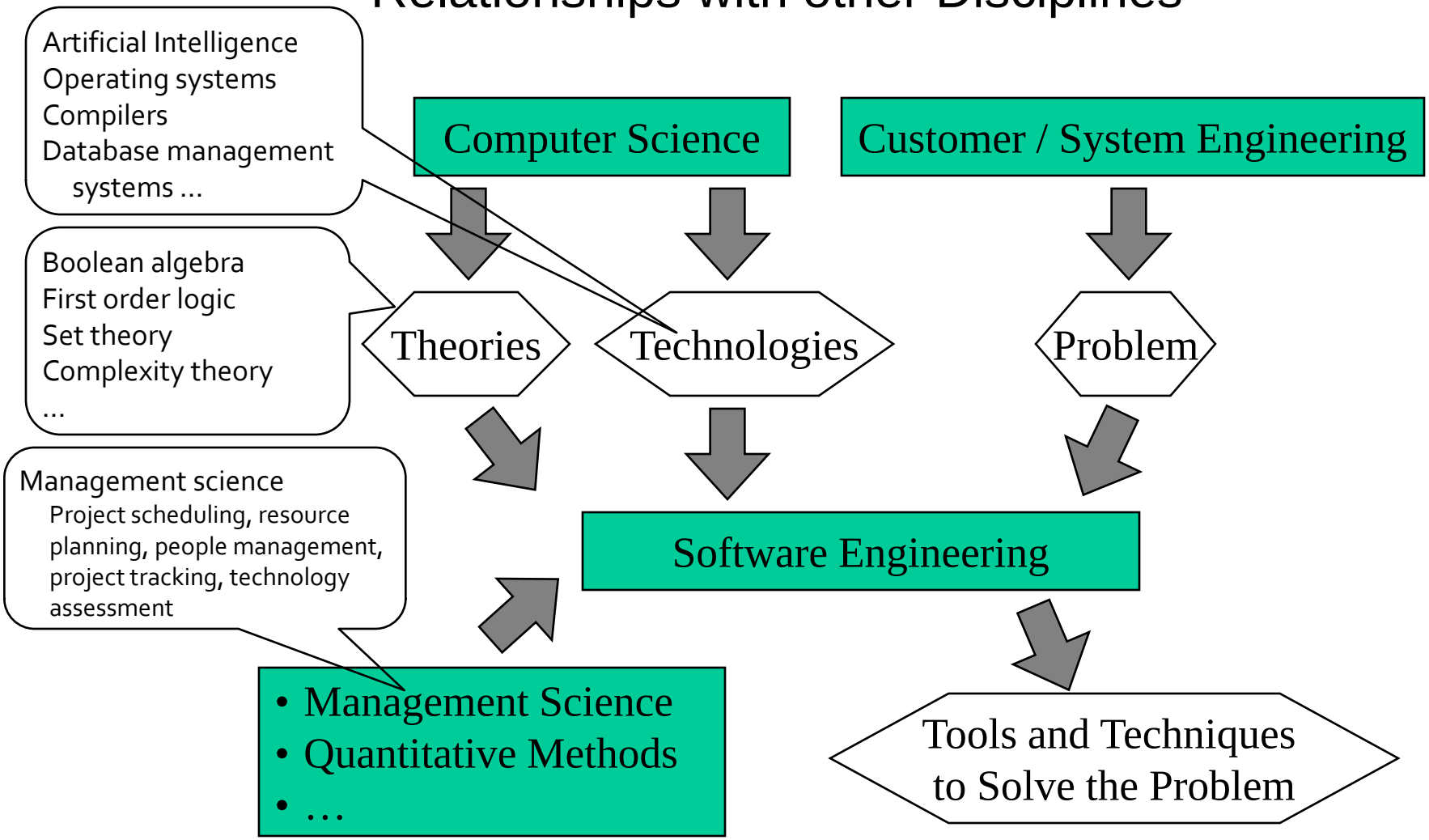
- Definitions
- Software Failures
- History and Context
- Software Development Myths
- Principles
- Software Development Processes
- Software Development Tools
- Summary

Historical perspective of SW Engineering

- Read the opening sentence in your textbook [Dutoit]
  - The term software engineering was coined in 1968 as a response to the desolate state of the art of developing quality software on time and within budget. …. More often that not, the moon was promised, lunar rover built and a pair of square wheels delivered.

- [Braude] The production of automobiles was revolutionized by Henry Ford's observation that parts could be standardized, so that cars of a given model could use any instance of each required part. The reduction in cost … made automobiles more affordable
  - We now expect to reuse ideas, architectures, designs or code from one application to build others. …
  - Only modular applications have potentially reusable parts.
  - Reusability of developer knowledge.

Relationships with other Disciplines



Pfleeger, 1998 (adapted)

Software Engineering Preview

- Definitions
- Software Failures
- History and Context
- Software Development Myths
- Principles
- Software Development Processes
- Software Development Tools
- Summary

Management Myths

- State-of-the-art tools are the solution

  “A fool with a tool is still a fool”

- Falling behind schedule is resolved by hiring additional programmers

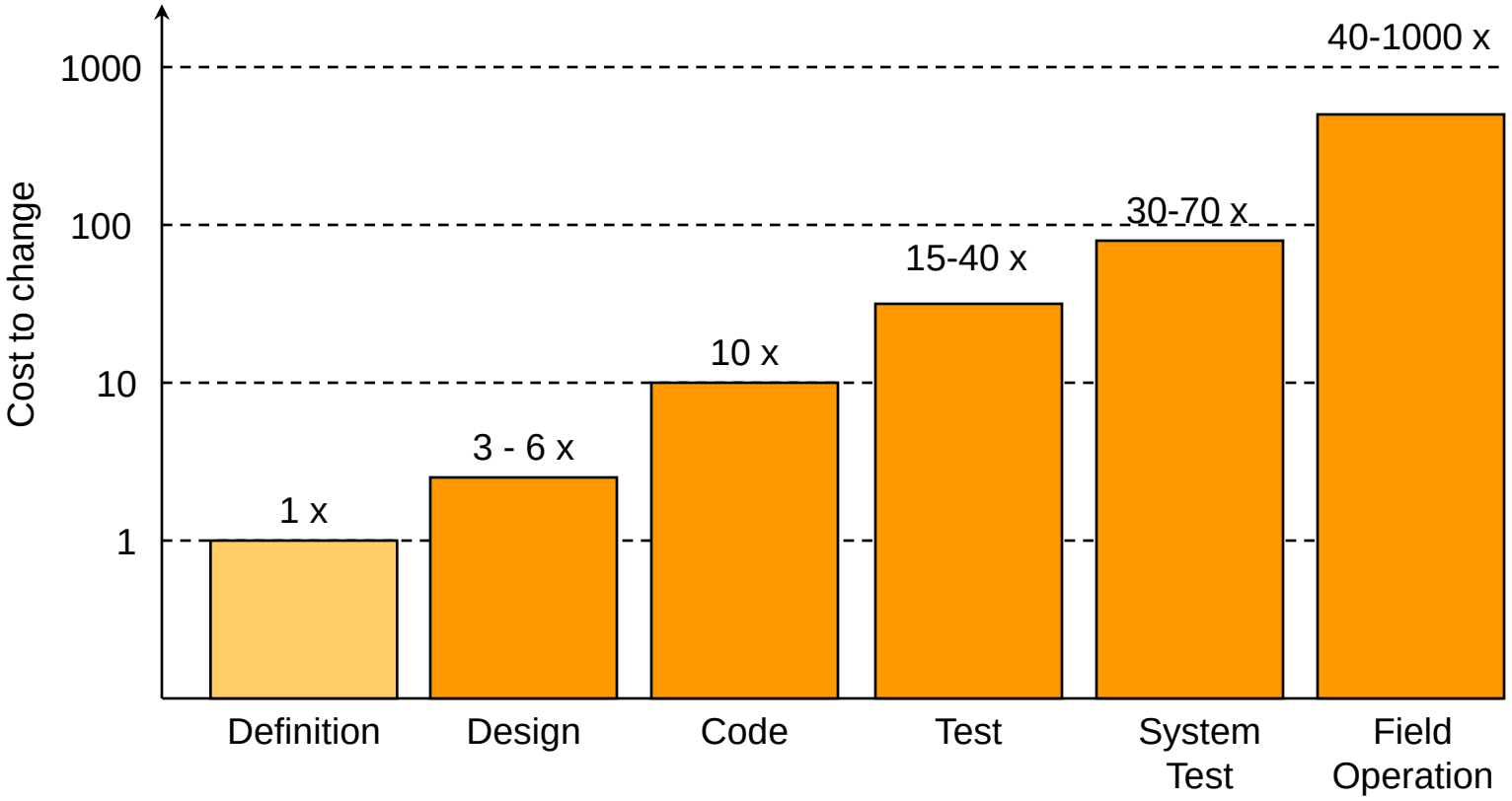
  “adding people to a late software project makes it later”

Customer myths

- A general statement of objectives is sufficient to begin writing programs - we can fill in the details later.
  - Problems:
    - Poor up-front definition is a major cause of failed software projects
    - Detailed description of function, performance, interfaces, design constraints and validation criteria are essential
  - Thorough communication between customer and developer needed
- Changes can be easily accommodated because software is flexible
  - Problem:
    - Impact of change grows throughout the lifecycle -> late changes are expensive
  - Changes happen as a fact of life, cannot avoid them

➢ Such myths lead to false expectations by the customer and result in dissatisfaction with the developer.

The impact of change



Practitioner's myths

- Once we write a program and get it to work, our job is done
  - 50-70% of all effort *after* first delivery
- Until I get the program "running", I really have no way in assessing its quality
  - inspections & reviews
- The only deliverable for a successful project is the working program
  - documentation (users, maintenance), e.g., UML Analysis and Design Models
- Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down
  - Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times

Software Engineering Preview

- Definitions
- Software Failures
- History and Context
- Software Development Myths
- Principles
- Software Development Processes
- Software Development Tools
- Summary

Characteristics of today's software development

- Development of large & complex systems
- Software systems must fulfill the requirements of many users (or usage conditions)
- Number of persons involved in the development >>>> 1
- Distributed development is now commonplace
  - Same place, same city (Kanata-Downtown)
  - Same country (Ottawa-Vancouver), same continent
  - Ottawa-Vancouver-England-India-Australia
- Software systems are expected to live long and be used by many people.

What are the problems?

- Increased quality demands on software products
- High cost and time pressure
- Shorter time to market
- Coordination problems within the projects
- Scarce resources (e.g., qualified personnel)

Software Engineering Principles

- There are a number of general principles underlying and driving all software engineering techniques
- They aim at dealing with the inherent complexity of software and help achieve quality goals, e.g., reliability, evolvability
- We will refer to these principles throughout the course.

- Rigor and formality
- Separation of concerns
- Modularity
- Abstraction
- Anticipation of change
- Generality
- Incrementality

Rigor and Formality

- More reliable products, control costs, increase our confidence in the product
- Rigor: well-defined, repeatable, technically sound steps (based on method, technique)
- Formality, the highest degree of rigor, require the software process to be driven by mathematical laws
- No need to be always formal -> tradeoff
  - Formality and Rigor have a cost (training, additional time) , so apply as long as the benefits are significantly larger
- Rigor and formality apply to both the SW process and product

- The UML notation is an example of a (semi-)formal notation. It brings rigor to the way we do analysis and design.

Separation of Concerns

- Decompose a complex problem (or concern) into simpler problems
  - Sub-problems (or concerns) should overlap as little as possible
- A concern is anything of interest
- Concerns may be separated
  - in time (e.g., life cycle phases),
  - qualities (the “alities”),
  - product views (e.g., UML diagrams),
  - product parts (subsystems, components)

Modularity

- Software systems are decomposed into simpler pieces: modules, components
- High cohesion and low coupling within/among components
- Allow reuse, easier understanding, team work, etc.
- Ideally, SW development could be based on composing reusable components

- Modularity vs. “separation of concerns”
  - Separation of concerns (for product parts) is often achieved through modularity

Abstraction

- Identify important aspects and ignore non-relevant details for the task at hand
- Equations, formalisms are forms of abstractions from reality, in all engineering disciplines
- Software specifications and design representations / models: abstract away from programming details
- Programming languages: abstract away from hardware details

Anticipation of Change

- Software undergoes change constantly
- How to account for potential change and limit the side effects?
- Impact on design strategy
  - Layered architecture

    e.g., user interface, business or application logic, database management system
  - Design patterns
- Manage versions and revisions (Configuration management)
- Process changes, e.g., personnel turnover: Analysis and Design documentation

Generality

- General solutions mean more software reuse
- General software solutions for a given application domain
- Different forms:
  - libraries, executable components, frameworks (e.g., JavaCC)
  - Database management systems, spreadsheets, text processing and numerical analysis libraries
- Overhead, acquisition cost versus reliability, reuse
- Large, expanding COTS market (Components/Commercial Off The Shelf) in the software industry

Incrementality

- Stepwise development => early subsets of an application
  - build the software in small increments; for example, adding one use case at a time
- Early feedback from customers, users
- Initial requirements often not stable and fully understood
  - start with parts that are clear
- Incrementality requires special care for managing documents, programs, test data, etc. of successive versions (configuration management)

Software Engineering Preview

- Definitions
- Software Failures
- History and Context
- Software Development Myths
- Principles
- Software Development Processes
- Software Development Tools
- Summary

The Software Process

- Software Engineering [IEEE-93]:
  - The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
- A Software Process is a series of predictable steps to follow to create a timely, high-quality result.
  - Provides stability, control, organization
  - Can be adapted to individual process needs (not rigid, can be agile)

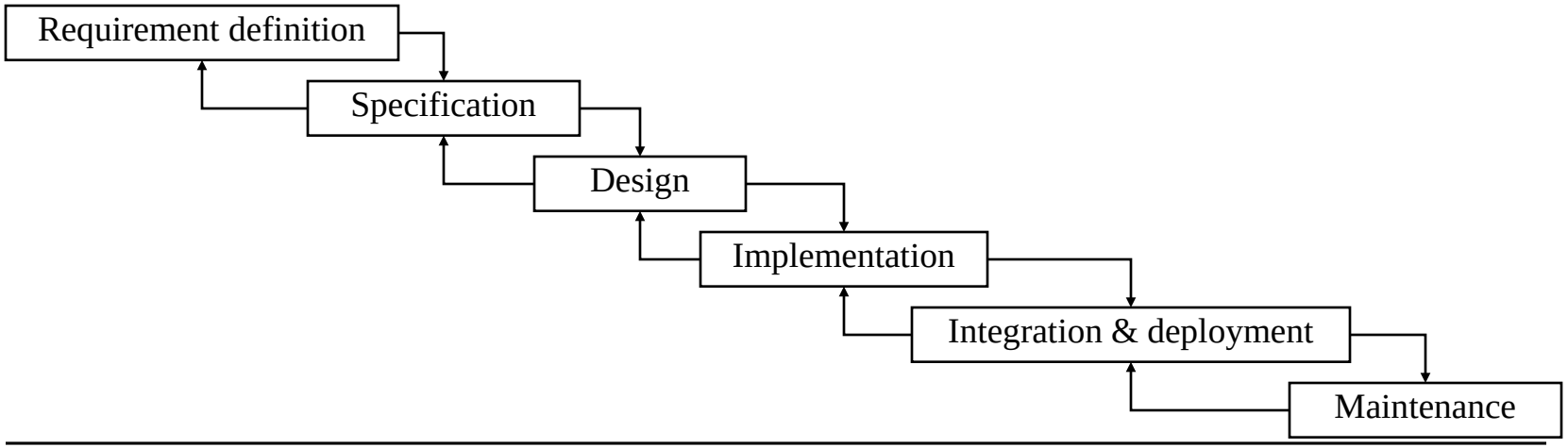
Survey of Some Process Models

- Waterfall Model
- Phased-Release Model
- Spiral Model
- Unified Process
- Agile Process
- Model-based Development Process

- All have the afore-mentioned activities and principles.

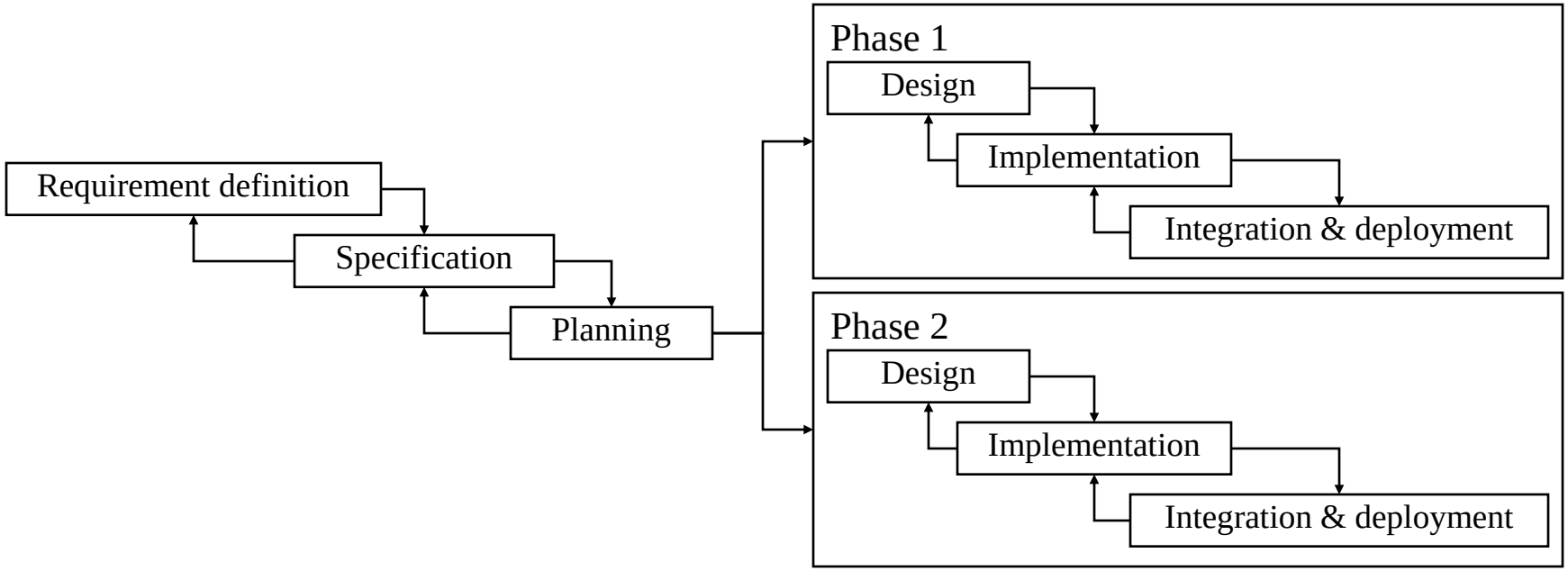
Waterfall Model

- In principle, a phase should not start until the previous phase has finished (has been approved).
- Problems:
  - Real projects rarely follow the sequential flow
  - Difficult for stakeholder to state all the requirements once and for all
  - Stakeholders must have patience: working version of software comes late in process.



Phased-Release Model

- Principle:
  - Linear sequences of the waterfall process, with each sequence producing an operational deliverable.
  - The incremental model delivers a series of releases, called increments.
  - Suggests that all requirements are finalized early in the process.



Spiral Model

Incremental and Iterative

Idea:

- start by developing a prototype following a mini-waterfall model.
- Prototype serves to gather requirements.
- Each increment is reviewed and evaluated.

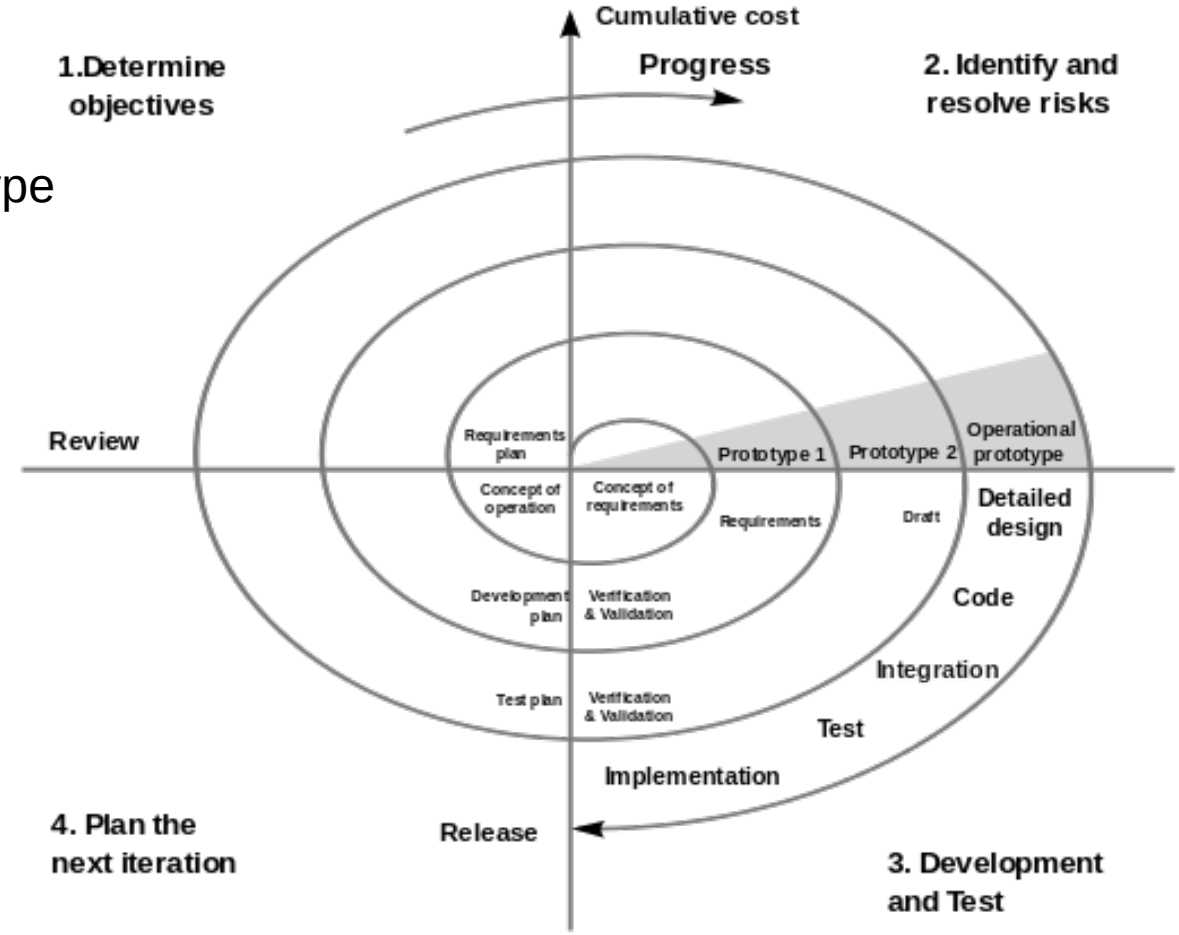


Image from creative commons

Unified Process

Iterative and Incremental, Use-case driven, Architecture centric

Phases:

- Inception: The core idea is developed into a product vision. We review and confirm our understanding of the core business drivers. We want to understand the business case for why the project should be attempted. Product feasibility and project scope.
- Elaboration: The majority of the Use Cases are specified in detail and the system architecture is designed. "Do-Ability" of the project. We identify significant risks and prepare a schedule, staff and cost profile for the entire project.
- Construction: Produces a system complete enough to transition to the user. The design is refined into code.
- Transition: The goal is to ensure that the requirements have been met to the satisfaction of the stakeholders. Other activities include site preparation, manual completion, and defect identification and correction. The transition phase ends with a postmortem devoted to learning and recording lessons for future cycles.

Unified Process (cont.)

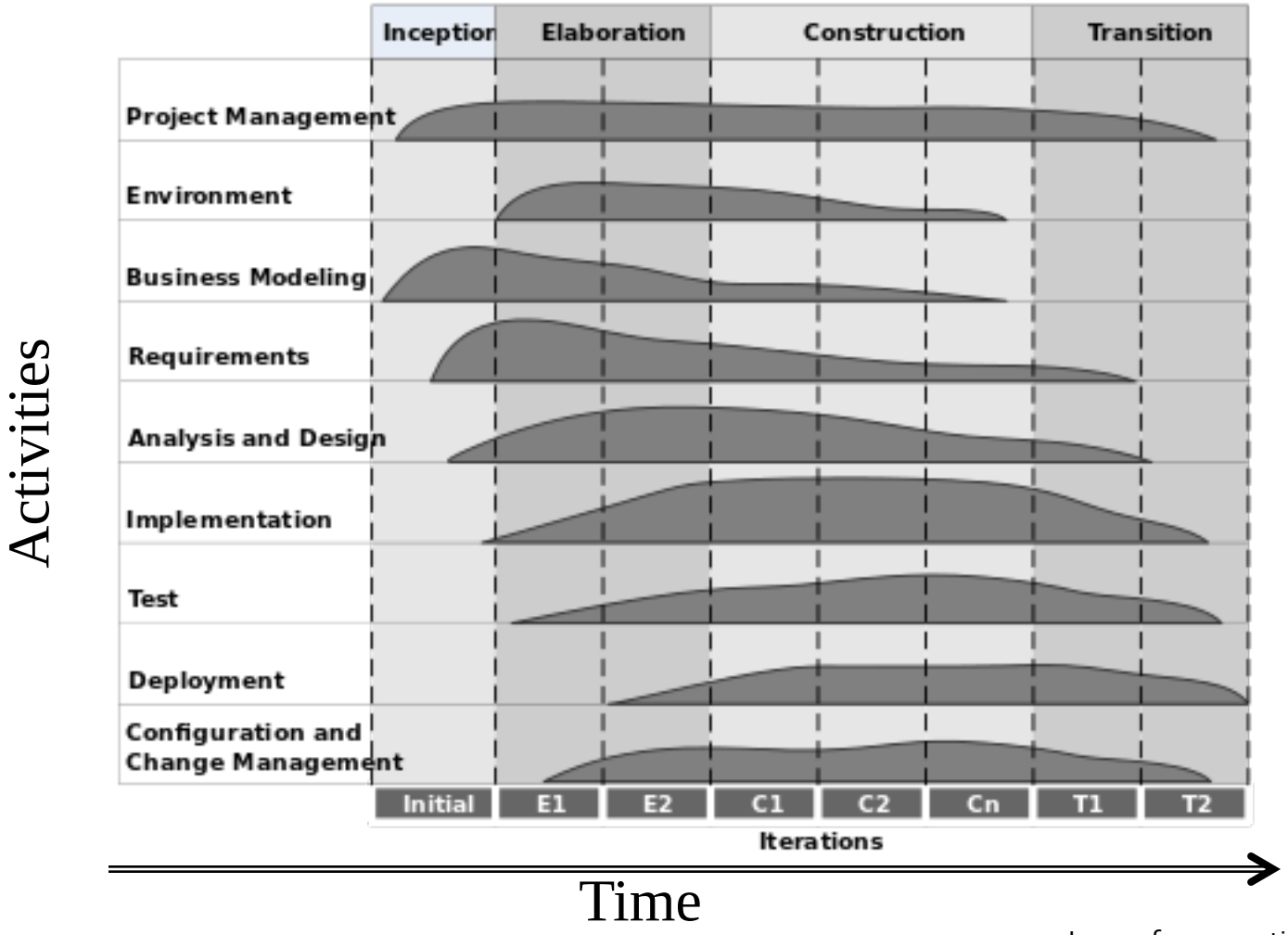
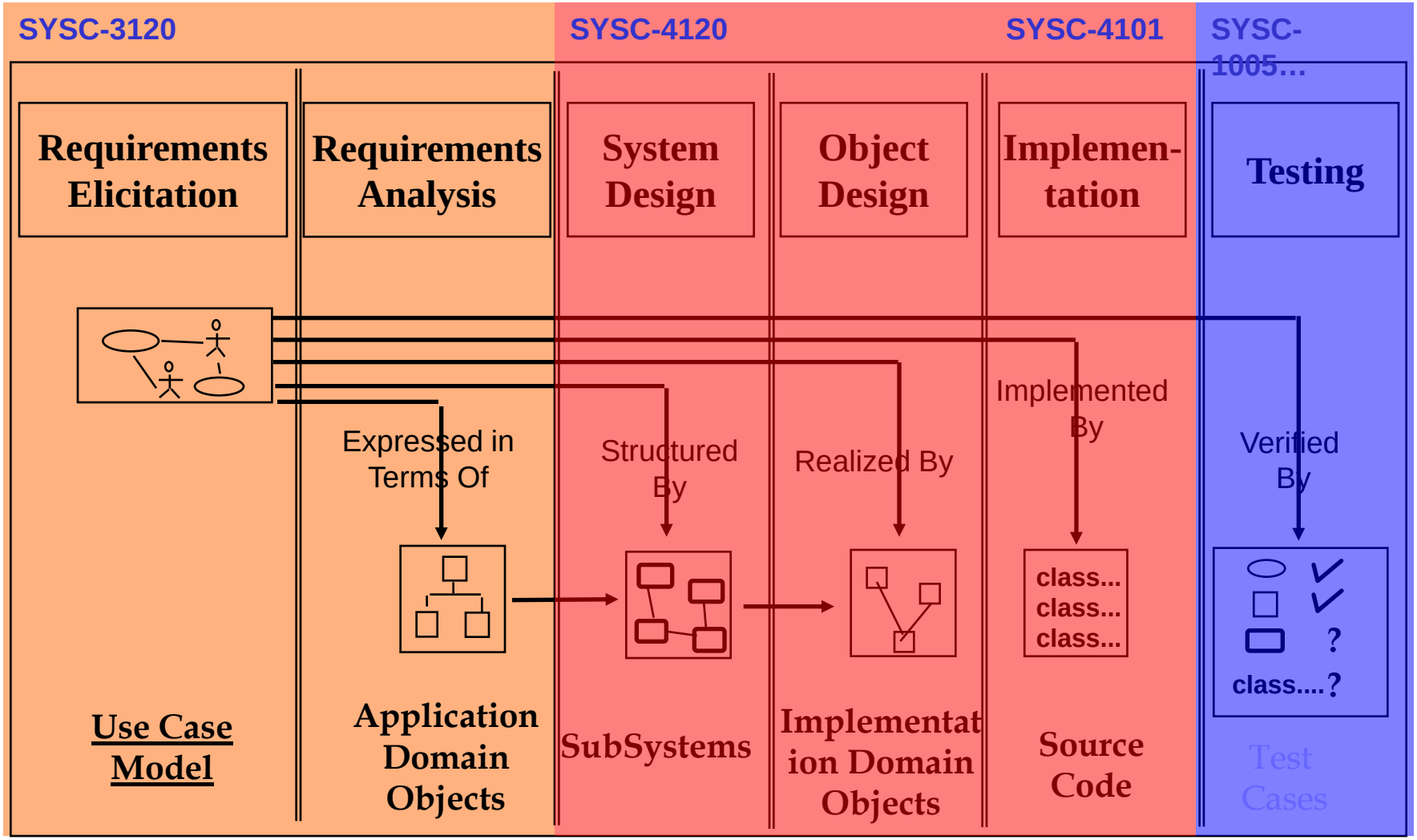


Image from creative commons

Agile Model

- Key Assumptions
  - Difficult to predict software requirements
  - Difficult to predict analysis, design, construction, and testing
  - Design and construction should be interleaved
- How can we design a process that can manage unpredictability?
  - ➢ Process adaptability.
- Example: Extreme Programming (XP)
  - 4 phases: Planning (stories), Design (prototype solutions), Coding (pair programming, re-factoring), Test
  - The tests are the specification
  - Communication paramount (small team, knowledgeable programmers)

Software Lifecycle in Textbook



Software Engineering Preview

- Definitions
- Software Failures
- History and Context
- Software Development Myths
- Principles
- Software Development Processes
- Software Development Tools
- Summary

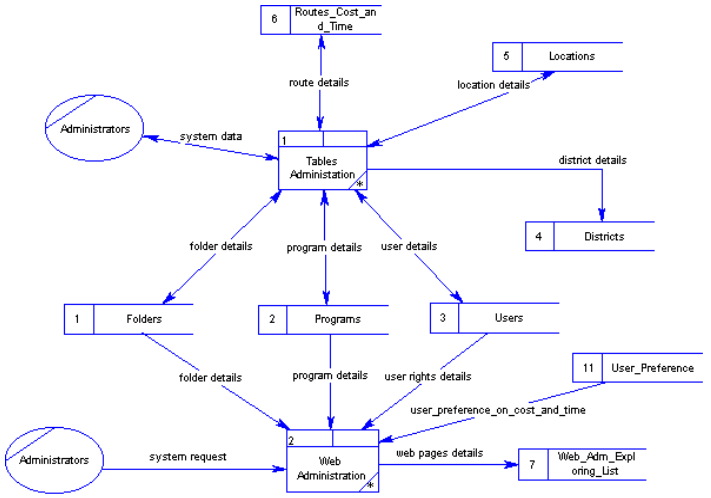
Some Modeling Tools

- Dataflow diagrams
- Entity-Relationship diagrams
- Finite state machines
- Petri nets, queueing networks, fault tree
- Descriptive Formal specification languages: Z, B, Alloy, SPIN, TRIO, VDM, LOTOS, RT-LOTOS …

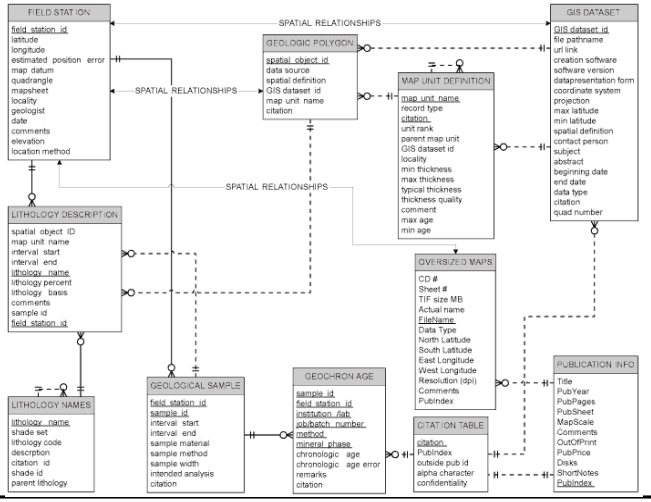
- Which notation is used depends on the type of system, familiarity of analysts, organization decisions, etc.
- UML combines/adapts many notations into a consistent framework

Some Modeling Tools

- Data Flow Diagram

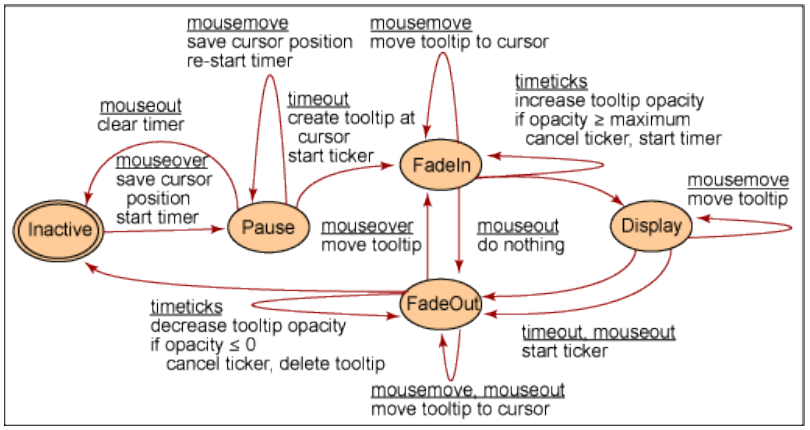


- Entity Relationship Diagram

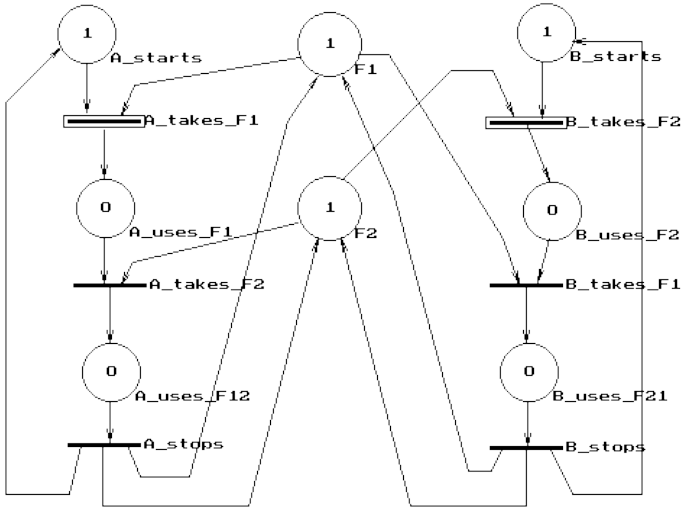


Some Modeling Tools

- Finite State Machine

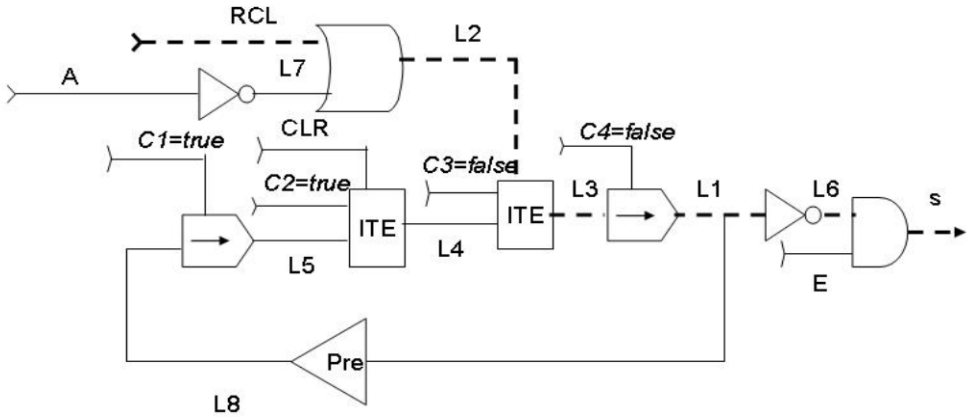


- Petri Net



Some Modeling Tools

- Synchronous (SCADE) diagram



- Z notation

$max : \mathbb{N}$

Buffer

$items : \operatorname{seq} MSG$

$\#items \leqslant max$

Buffer$_{INIT}$

Buffer

$items = \langle\rangle$

Join

$\Delta Buffer$

$msg? : MSG$

$\#items < max$

$items' = items \frown \langle msg? \rangle$

Leave

$\Delta Buffer$

$msg! : MSG$

$items \neq \langle\rangle$

$items = \langle msg! \rangle \frown items'$

Some Modeling Tools

- Unified Modeling Language (UML)
  - Now the de-facto standard for OO software development
  - 70% of IT shops use the UML in one way or another
  - 90% of the Fortune 500 companies use the UML in one way or another

UML diagrams

Two categories of UML diagrams (out of 14 diagrams types)

- Structure diagrams
  - class, component, composite structure, package, profile, deployment, object
- Behaviour diagrams
  - use case, state machine, activity, sequence, communication, interaction overview, timing

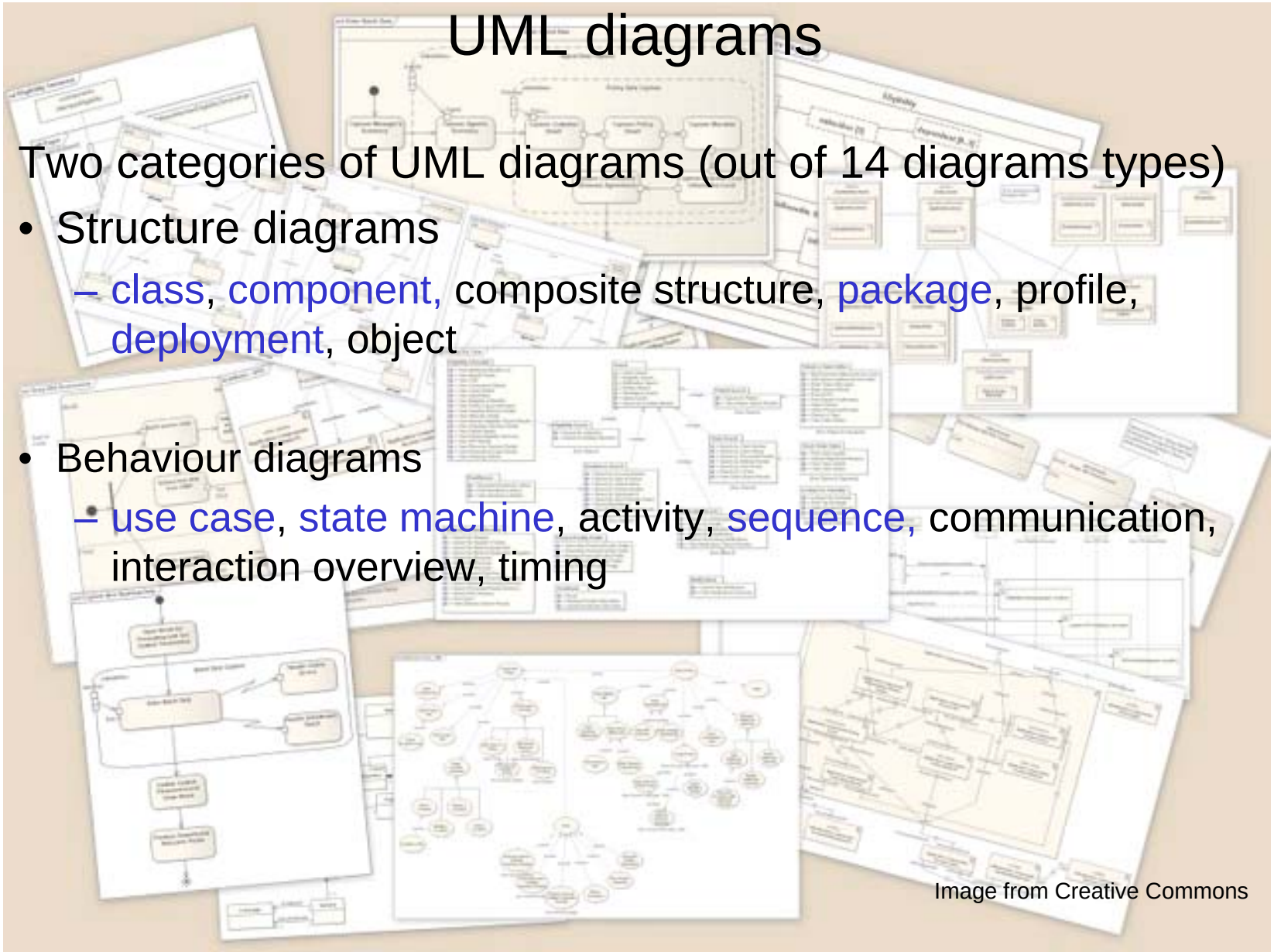


Image from Creative Commons

UML CASE Tools

- CASE = Computer-Aided Software Engineering
  - Using automated tools (software) in software engineering
- UML CASE tool
  - Software to support the use of the UML
  - Many, many such tool (incomplete list):
    - http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools
    - with varying capabilities
  - Software engineering activities supported?

Drawing	Modeling
Different versions of UML	Varying support for notation
Code generation	Documentation generation
Individual and team work	
Software development phases (analysis, design, …)	
Measurement	Simulation
Forward/Reverse/Round-trip engineering	Traceability …

Software Engineering Preview

- Definitions
- Software Failures
- History and Context
- Software Development Myths
- Principles
- Software Development Processes
- Software Development Tools
- Summary

Summary

- Software engineering is
  - engineering discipline focused on the development of software systems
  - as the solution to a user's problem
  - applying the right techniques / methods / tools
- Software engineering is necessary for developing complex but reliable software
- A good software engineering practice help to develop software in large teams.

SE goal is to avoid this