CH1 - Introduction

- **Software:** Computer programs & associated documentation
- **Software engineering:** concerned with theories, methods, tools for software development
    - Fundamental activities: software specification, development, validation, evolution
    - A part of system engineering (hardware + software + process engineering)
- Often: Software costs > computer system / hardware cost
- Software maintenance cost > development cost
- Software products: "Generic" or "customized"
- Essential attributes of good software:
    - **Maintainability:** Critical attribute. To be able to evolve with changing needs
    - **Dependablity:** Security, reliability, safety. Should not cause damage in case of failure
    - **Efficiency**
    - **Acceptability:** Compatible with other systems, understandable, usable

---

Application Types

Stand-alone

- Run on a local machine (e.g a PC).
- Need not to be connected to a network
- Has all required functionality

Interactive transaction-based

- Accessed by users externally, runs on remote computer
- Web app.s (e.g e-commerce app.s)

Embedded control systems

- Controls and manages HW

Batch processing systems

- Processes individual inputs in large batches

Data collection systems

- Collect data using sensors, send to other systems to process

Entertainment systems

Modelling & simulation systems

- Developed by scientists & engineers for modeling
- Usually includes many separate interacting objects

Systems of systems

- Composed of other software systems

- Web-based systems: Distributed & complex. Need "agile development": impractical to specify all the requirements for such systems in advance.
- Service-oriened systems: All components considered as replaceable services. Allows rapid configurations & incremental updates as new services become available.

CH2 - Software Processes

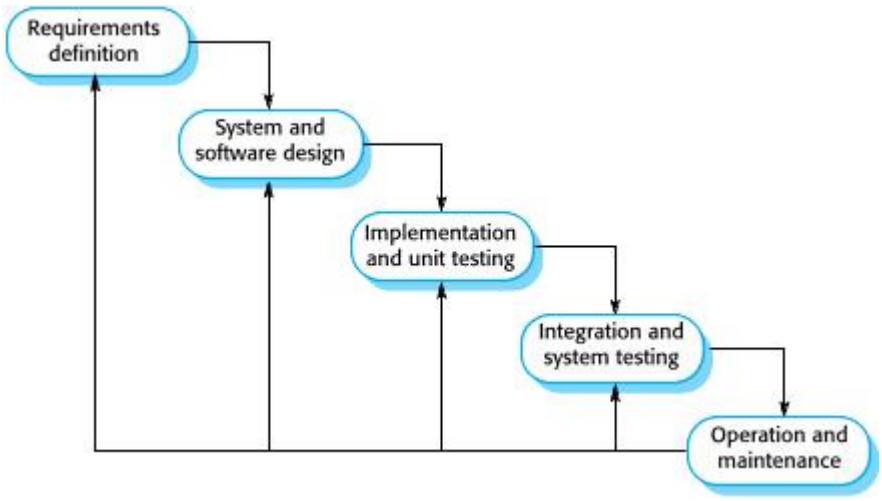
- "Software process": Structured set of activities required to develop a software system
- Many different ones but alll involve: Specification, design & implementation, validation, evolution
- "Plan driven process": all activities planned in advance, progress is measured against the plan

"Agile process": Incremental planning, easier to change provess to changing requirements

Often, elements from both approaches are used.

Software Process Models

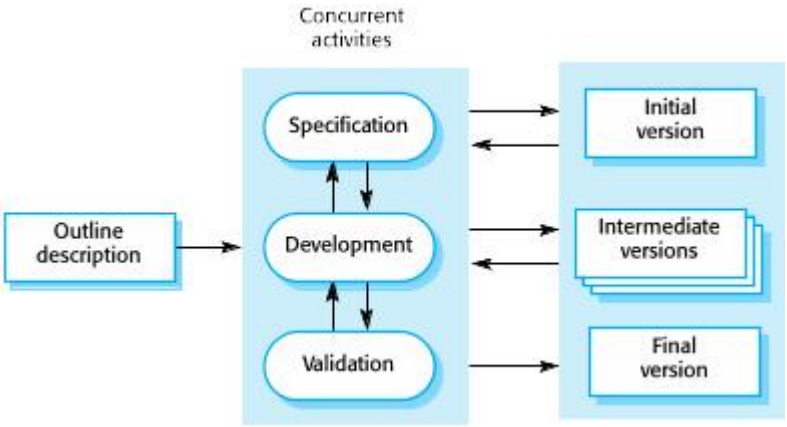
The Waterfall Model



- Plan-driven
- Separate identified phases

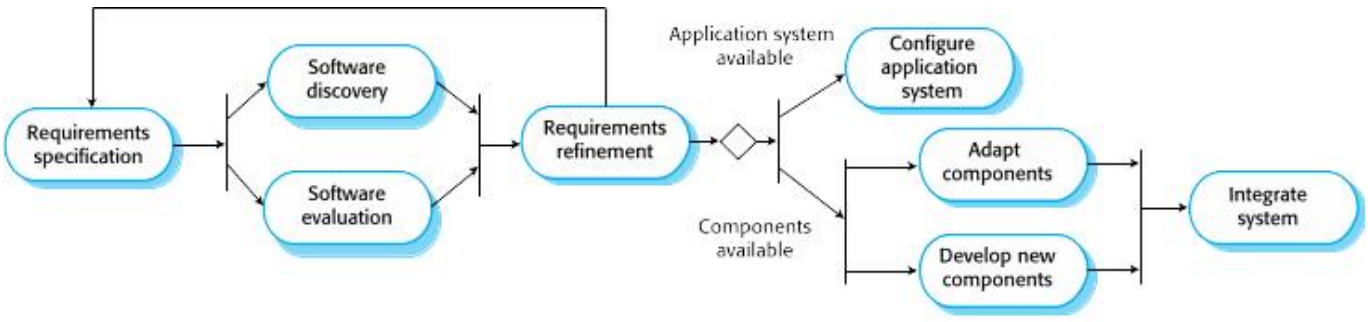
- Drawback: a phase has to be completed before moving onto next, hard to accomodate changes
    - Only appropriate when the requirements are well understood & changes will be limited
- Mostly used for large system engineering projects where a system is developed at several sites

Incremental Development



- Cost of making changes reduced
- Easier to get customer feedback
    - Customers can comment on demonstrations
    - Customers can be provided with useful software earlier
- Drawbacks:
    - Regular changes may corrupt project structure - incorporating changes becomes harder and costlier as time progresses
    - Cost-ineffective to document each system version -> Measuring progress is hard

Integration & Configuration (Reuse-oriented SW. Eng.)



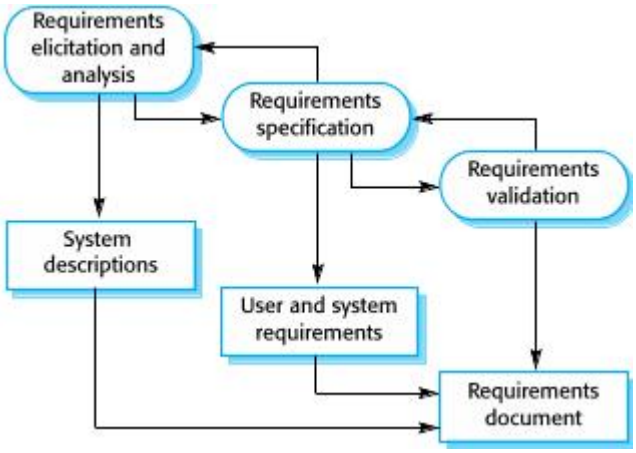
- Based on software reuse - reused elements can be configured according to requirements
- Standard approach for many business system types

- Faster delivery
- Less SW developed from scratch -> Reduced risks & costs
- Drawbacks:
    - Some requirements are likely to be unsatisfied
    - Lost control over the evolution of reused components

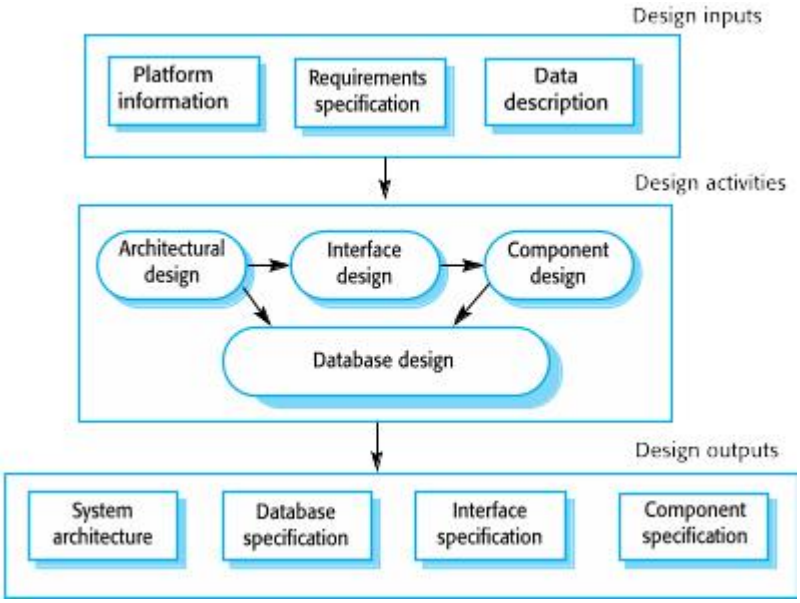
Process Activities

- Can be ordered sequentially or interleaved according to the model

Requirements Engineering Process



Software Design & Implementation

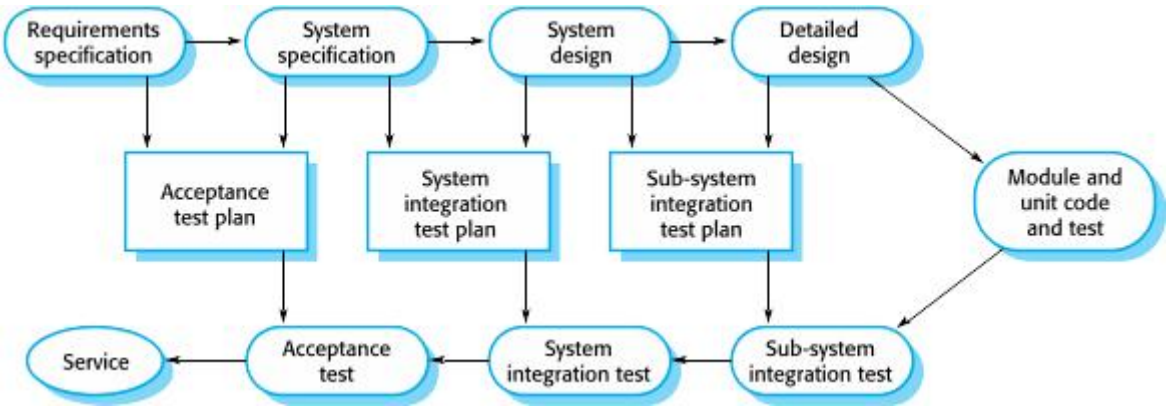


- Design: Creating software structure
    - Architectural, database, interface (between components), component selection & design
- Implementation: translating the structure into an executable
    - Design & implementation can be closely related or interleaved

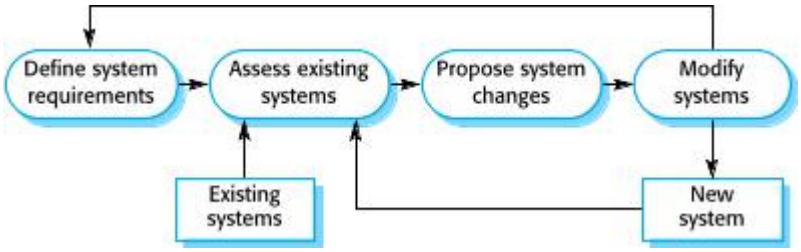
Software Validation

- Testing: Most common verification & validation activity

- V-model: Testing phases in a plan-driven software process:

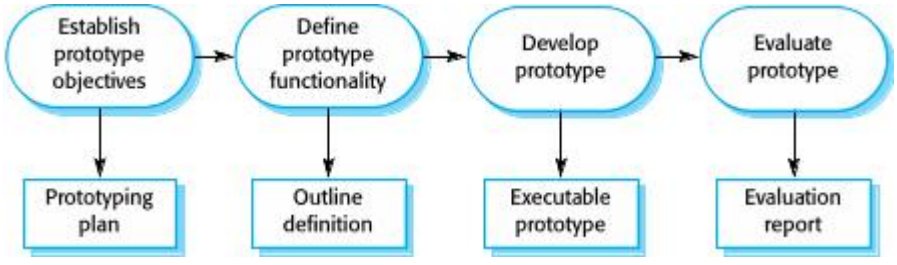


System Evolution



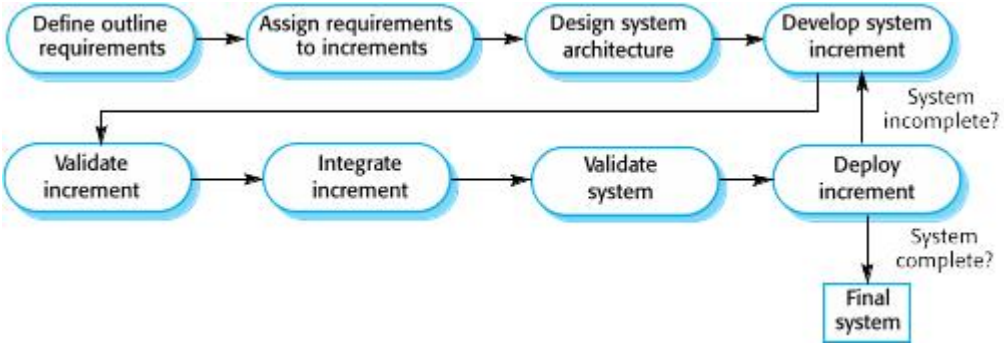
- Change anticipation: Possible changes can be anticipated without much rework
- Change tolerance: Process is designed so that changes are applied without much cost

Software Prototyping



- Discarded after development: usually unstructured, undocumented and not standard-compliant

Incremental Development & Delivery

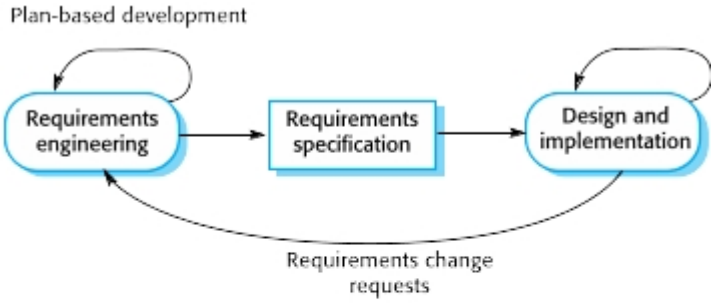


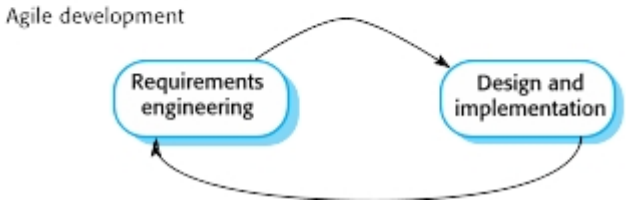
- Early increments serve as prototypes
- Requirements are not changed throughout the development of an increment
- Highest priority services receive most testing
- Specification is developed toghether with the SW itself
    - May contradict procurement models of some organizations

Process Improvement

- Process maturity approach: ? Agile approach: focus on iterative development & reduction of overheads. Emphasis on rapid functionality delivery & adapting to requirement changes
- improvement cycle: Change -> Measure -> Analyze -> Change ...
- Process metrics:
    - Taken time to complete activities
    - Required resources for activities
    - Number of occurrences of a specific event (e.g. an error)
- SEI capability maturity model:
    1. Initial: uncontrolled
    2. Repeatable: Defined & used product management procedures
    3. Defined: Defined & used process management procedures
    4. Managed: Defined & used quality management strategies
    5. Optimising: Defined & used process improvement procedures

CH3 - Agile SW Development





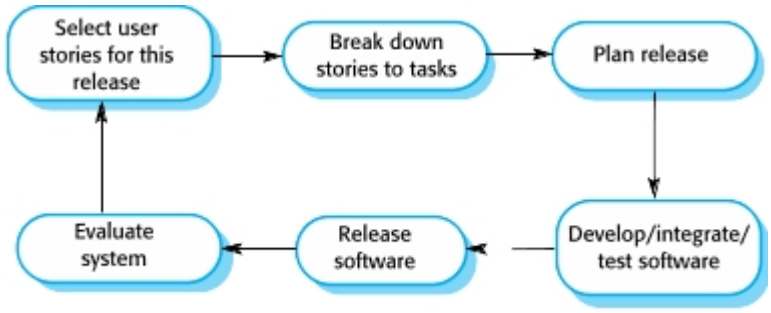
- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- The aim is to reduce overheads in the SW process, to be able do respond quickly to changing requirements without excessive rework.
    - e.g Minimal documentation - focus on working code

Principles

- Customer involvement: Evaluating iterations, provide and prioritize requirements
- Incremental delivery
- People over processes: Skill of the team should be recognized - members should develop their own ways of working without prescripted processes
- Embrace change
- Maintain simplicity

Agile development techniques

Extreme programming (XP)

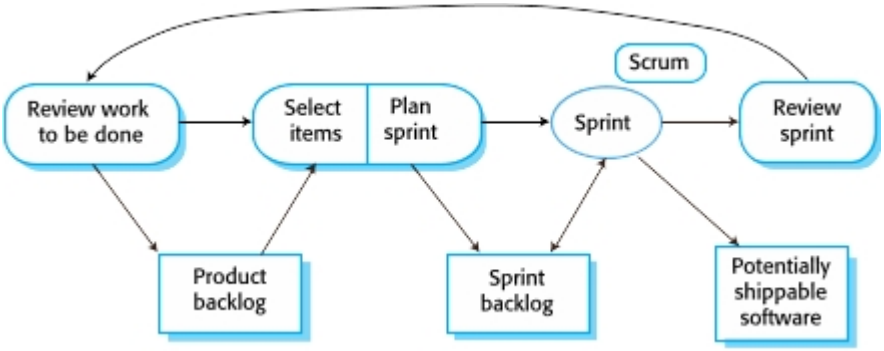


- "Extreme" iterative development:
    - New versions may be built several times per day
    - Increments are delivered to customers every 2 weeks
    - All tests must be run for every build and the build is only accepted if tests run successfully.
- Has a technical focus - usually not easy to integrate with management practice
- XP practices:
    - Incremental planning: Requirements recorded onto "story cards" -> Broken down into "Tasks"
    - Minimal useful set of functionality is developed first, frequent small releases. Functionality added incrementally.
    - Minimum design to satisfy current requirements is carried out
    - Test-first development: A functionality's test framework is implemented earlier than the functionality itself
    - Constant refactoring: All developers are expected to refactor code as soon as possible, even if there is no immediate need
        - Improves software understandability & reduces need for documentation
        - Well-structured code -> Changes are easier to make
        - Changes requiring architecture refactoring are much more expensive
    - Collective ownership: Everyone works on everything, no islands of expertise develop
    - Pair programming
        - Helps develop collective ownership
        - Informal review process
        - Encourages refactoring & sharing of knowledge
    - Continuous integration: After each task is completed, it is integrated & tested (must pass the test)
        - Test-driven development clarifies the requirements to be implemented
        - Tests are written as programs rather than data -> they can be executed automatically after integrating each new functionality
    - The customer should be available full-time as a member of the development team

        - Customer may be reluctant - may feel that providing requirements was enough
    - Large amounts of overtime are not acceptable: Reduces productivity & quality on long term

Agile Project Management

Scrum



- Agile method focusing on managing iterative development rather than agile practices
- "product backlog": a "TODO list", may be feature definitions, SW. requirements, user stories, supplementary tasks (e.g user documentation, architecture definition), etc.
- "scrum"s: daily meetings reviewing progress and daily tasks, ideally short f2f meeting including whole team
- "velocity": estimate of backlog covering rate of the team
- 3 phases:
    - Initial: plan outline, establish objectives (choose from backlog), design architecture
    - "Sprint cycles" of developing increments
        - fixed length (~2-4 weeks)
    - Project closure: wrap-up & complete documentation (e.g system help frames)
- Team is isolated from distractions & customer communication. Only the "scrum master" handles communication with the customer.
- Benefits:
    - Breaks down product into managable chunks
    - Unstable requirements do not hold up
    - Whole team has visibility on everything
    - Customers see on-time increment delivery, can provide feedback
        - Establishes customer-developer trust

Scaling Up Agile Methods

- Agile methods are successful for small & medium sized projects with small teams
    - "Scaling up": Agile methods for large SW systems with large teams
    - "Scaling out": Introducing agile methods to a large, experienced organization

Practical problems

- Agile is informal -> incompatible with legal approach to contracts in large companies
- Agile is more approporiate for new software rather than maintenance
    - In large companies usually: Maintenance cost > development cost
- Agile is more approporiate for small co-located teams
    - SW development involves worldwide distributed teams
    - Design documents may be needed for distributed teams
        - If not available: IDE support for visualisation & program analysis is essential
    - Cross-team communication mechanisms are needed
- Agile works best when team has high skill level, but large companies have a wide range of skill level
- Software contracts are based around specifications; however, agile interleaves specification and development
    - Contract based on time rather than functionality is required for agile -> considered risky
- Maintenance problems:
    - Lack of SW documentation
        - Also needed if system is subject to external regulation
    - Customers are kept involved in development
        - Diverse set of stakeholders in large systems -> hard to involve them all in development
    - Need to keep the original team
        - Problem for long-lifetime systems
- Prioritizing changes may be difficult: Multiple stakeholders have different priorities
- Incremental delivery: Can be hard for business planning and marketing

---

- Several integrated systems: Significant amount of development on system configuration rather than development
- Completely incremental approach to requirements engineering is impossible
- Continuous integration practically impossible. However, frequent system builds & regular releases are essential
- Multi-team Scrum:
    - Releases are aligned
    - Each team has their own scrum masters and product owners
    - Each team chooses "product architect"s to design & evolve overall system architecture
    - "scrum of scrums" are done where representatives of each team meet and discuss progress