# Chapter 2: Software Processes

1. Specification

* defining what the system should do

1. Design and implementation

* defining the organization of the system and implementing the system

1. Validation

* checking that the system does what the customer wants

1. Evolution

* changing the system in response to changing customer needs

1. Stage of testing
   * Component / Development testing
   * System / Release testing
   * Customer / User testing
2. Software prototype
   * A prototype is an initial version of a system used to demonstrate concepts and try out design options
3. Process metric
   * Used to measure the efficiency and effectiveness of various processes
4. Design activities
   * Architectural design
   * Database design
   * Interface design
   * Component selection and design
5. Stages of testing
   * Component
   * System
   * Acceptance
6. Reducing the costs of rework
   * Change anticipation
   * Change tolerance
7. Benefits of prototyping
   * Improved system usability.
   * A closer match to users’ real needs.
   * Improved design quality.
   * Improved maintainability.
   * Reduced development effort
8. Throw away prototypes
   * It may be impossible to tune the system to meet non-functional requirements;
   * Prototypes are normally undocumented;
   * The prototype structure is usually degraded through rapid change;
   * The prototype probably will not meet normal organizational quality standards.
9. Incremental delivery
   * the development and delivery is broken down into increments with each increment delivering part of the required functionality
   * User requirements are prioritised and the highest priority requirements are included in early increments
10. Process improvement
    * Way to enhance quality of software
    * Process maturity approach
    * Agile approach
11. Process improvement activities
    * Measure
    * Analysis
    * Change

# Chapter 3: Agile Software Development

1. Rapid software development

* often most important requirement of software

1. Agile development

* Program specification, design and implementation are inter-leaved
* Delivered system as a series of versions
* Frequent delivery 2 to 4 weeks
* Minimal documentation

1. Plan driven development

* Separate development stages with outputs at each stage
* Iteration occurs within activities

1. Agile development

* Specification, design, implement and testing interleaved
* Small to medium sized product
* Customer involvement

1. Aim of agile

* To reduce overhead in software process
* Reduced documentation
* Respond quickly to changing requirements

1. Extreme programming

* Incremental planning
* Small releases
* Simple design
* Test first
* Refactoring
* Pair programming
* Collective ownership
* Continuous integration
* Sustainable pace
* Customer involvement

1. User stories

* To capture requirements

1. Refactoring

* Constant code improvement to make change easier

1. Test-first
   * Write tests before code
   * Need test automation
2. Agile project management
   * Need different approach than plan driven
3. Scrum sprint

* A short (3-4 weeks) planning unit in which work to be done is assessed, features are selected for development, the software is implemented and delivered to system stakeholders.

1. Agile problems
   * Scaling up
   * Scaling out
   * Legal approach to contracts
   * Cost of maintenance

1. System issues with agile
   * System size
   * System types
   * System lifetime
   * External regulation
2. People and Teams
   * How good are programmers
   * Team organization
   * Supporting technologies
3. Organizational issues
   * Plan based culture
   * Customer access

# Chapter 8: Testing

1. Program testing
   * Show program does what is intended
   * Execute program using artificial data
   * Demonstrate sw meets requirements
   * Discover defects
2. Verification and validation
   * System is fit for purpose
3. Validation:

* Are we building the right product?

1. Verification:

* Are we building the product right?

1. Inspections
   * Analysis of the system to discover faults
   * Static verification
   * Code and document reviews
   * Manual process
   * Very effective to find hidden errors

1. Testing
   * Exercising and observing product behaviour
   * Dynamic verification
2. Stages of testing

* Development testing, where the system is tested to discover bugs and defects
* Release testing where the system is tested to check that it meets its requirements
* User testing where the system is tested in the user’s environment.

1. Development testing

* where the system is tested to discover bugs and defects

1. Release testing

* where the system is tested to check that it meets its requirements

1. User testing

* where the system is tested in the user’s environment.

1. Component testing

* where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.

1. System testing

* where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

1. Unit testing

* where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.

1. [Testing](http://iansommerville.com/software-engineering-book/web/test-planning/) strategies

* Partition testing
* Guideline-based testing

1. Partition Testing

* Groups of inputs with common characteristics

1. [Guideline based testing](http://iansommerville.com/software-engineering-book/web/path-testing/)

* Based on experience in knowing types of errors that occur and where

1. Equivalence partition

* A class of inputs or outputs where it is reasonable to expect that the system will behave the same way for all members of the class. For example, all strings with less than 256 characters

1. Test-driven development process

* Identify increment of functionality required
* Design tests for this functionality and implement as executable programs.
* Run test along with other implemented tests. The test will fail.
* Implement the functionality and re-run the test. Iterate until the test works.
* Move on to implement the next chunk of functionality

1. Alpha testing

* users work with the development team to test the software as it is being developed.

1. Beta testing

* the software is released to selected users for testing before the formal system release

1. Acceptance testing

* customers test a system to check that it is ready for deployment.

1. Release testing

* the software is tested by a team different than development
* purpose is to show system meets its requirements

1. Requirements based testing

* Verify functionality of system by executing test that correspond to system requirements

# Chapter 9: Software Evolution

1. Evolution

* The stage in a software system’s life cycle where it is in operational use and is evolving as new requirements are proposed and implemented in the system.

1. Software change

* Key problem to organize and manage in an organization

1. Servicing

* The stage in a software system’s life cycle where the software remains useful but the only changes made are those required to keep it operational i.e. bug fixes and changes to reflect changes in the software’s environment. No new functionality is added.

1. Phase-out

* The stage in a software system’s life cycle where the software may still be used but no further changes are made to it.

1. Change
   * Driven by system evolution
   * Continues throughout system lifetime
2. Change implementation

* Iteration of design, implement and test

1. Agile and evolution

* Incremental based so evolution is no different than development

1. Legacy systems

* systems that rely on obsolete languages and technologies

1. Legacy system replacement

* Scrap system.
* Continue to Maintain system
* Transform system
* Replace system

1. System replacement analysis

* Access system quality
* Access business value

1. Maintenance costs
   * 2 to 100 times development costs
2. Complexity metrics

* determine maintainability by complexity of system components

1. Software Reengineering

* recreate new system from legacy system.

1. Refactoring

* Improving program to slow down degradation.

1. Bad smells in program code

* Duplicate code
* Long methods
* Data Clumping
* Speculative generality

# Chapter 10: Dependable Systems

1. Most important aspect of many systems

* dependability of the system

1. Dependable system

* Reliability
* Availability
* Security

1. Hardware failure

* Hardware fails because of design and manufacturing errors or because components have reached the end of their natural life.

1. Software failure

* Software fails due to errors in its specification, design or implementation.

1. Operational failure

* Human operators make mistakes. Now perhaps the largest single cause of system failures in socio-technical systems.

1. Redundancy

* the inclusion of spare capacity in a system that can be used in the event of failure of part of the system.

1. Diversity

* the use of different types of redundant component so that the probability of a common failure that affects all redundant components is reduced.

1. Formal methods

* Formal specification
* Specification analysis and proof
* Transformational development
* Program verification

1. Verification-based approaches

* Different representations of a software system such as a specification and a program implementing that specification are proved to be equivalent.
* This demonstrates the absence of implementation errors.

1. Refinement-based approaches

* A representation of a system is systematically transformed into another, lower-level represention e.g. a specification is transformed automatically into an implementation.
* This means that, if the transformation is correct, the representations are equivalent.

1. Causes of failures

* A Hardware
* Software
* Operational failure

1. Dependability costs

* Increase exponentially with dependability requirement

1. Holistic system design

* Interaction between all components and layers in a system

1. Dependability process

* Explicitly defined
* Repeatable

1. Formal methods

* Approaches to software development based on mathematical representation and analysis

# Chapter 13: Security Engineering

1. Application security

* the application is designed to resist attacks

1. Infrastructure security

* the software is configured to resist attacks

1. Security dimensions

* Confidentiality
* Integrity
* Availability

1. Three controls to enhance system security

* Vulnerability avoidance
* Attack detection and neutralization
* Exposure limitation and recovery

1. Stages of preliminary risk assessment

* Asset value assessment, Exposure assessment
* Threat identification, Attack assessment
* Control identification
* Security requirements definition

1. Operational security

* Primarily human and social issue

1. Security trade off

* More secure system, less usable

1. Protection issues in system design

* How should the system be organised so that critical assets can be protected against an external attack?

1. Distribution issues in system design.

* How should system assets be distributed so that the effects of a successful attack are minimised?

1. Design guidelines for secure systems engineering

* Base security decisions on an explicit security policy.
* Avoid a single point of failure.
* Use redundancy and diversity to reduce risk.
* Validate all inputs

1. Experience-based testing,

* where the system is analyzed against known types of attack.

1. Penetration testing

* where an external team is contracted to discover security flaws in a system.

1. Tool-based testing

* where tools are used to exhaustively test some features of a system, such as the strength of passwords.

1. Formal verification

* where a system is formally verified against a formal security specification.

1. Interception threats

* Allows attacker to gain access to an asset

1. Interruption threats

* Make part or all of a system unavailable

1. Modification threats

* Attacker tampers with a system asset

1. Fabrication threats

* Insert false information in the system

1. Security specification

* Avoid something bad happening