**Specification** defines what the system should do

**Design and implementation** define the organization of the system and implementing the system

**Validation**: check that the system does what the customer wants

**Evolution:** change the system in response to changing customer needs

**Stage of testing**

Component / Development testing

System / Release testing

Customer / User testing

**Software prototype** is an initial version of a system used to demonstrate concepts and try out design options

**Process metric** is used to measure the efficiency and effectiveness of various processes

**Design activities**

Architectural design

Database design

Interface design

Component selection and design

Reducing the costs of rework

Change anticipation

Change tolerance

**Benefits of prototyping**

Improved system usability.

A closer match to users’ real needs.

Improved design quality.

Improved maintainability.

Reduced development effort

**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.

**Incremental delivery** the development and delivery is broken down into increments with each increment delivering part of the required functionality.

**Incremental development** the software is deployed to users in increments

\*User requirements are prioritised and the highest priority requirements are included in early increments

**Process improvement** is a way to enhance quality of software Process maturity approach agile approach

**Fundamental activities in all software processes:**

-**Software specification**. In this activity the functionality of the software and constraints on its operation must be defined.

-**Software design and implementation**. The software that meets the specification is produced.

-**Software validation**. The software must be validated to ensure that it has all the functionalities what the customer needs.

-**Software evolution**. The software must evolve to meet changing customer needs.

**A software process model** is an abstract representation of a process. It presents a description of a process from some particular perspective.eg**:**

**-Waterfall(**for large systems, inflexible**)**

**-Incremental (**flexible, customer feedback, rapid delivery, becomes complex as features are added**)**

**-Integration and configuration (**reuse elements and systems already developed reconfigured to your needs**)**

**A software process** is a structured set of activities required to develop a software system. E.g. **WATERFALL, AGILE, PROTOTYPES**

Plan driven development has separate development stages with outputs at each stage. Iteration occurs within activities

**Process improvement activities**

Measure

Analysis

Change

**Rapid software development** often most important requirement of software are developed first

**Agile development**:

- Program specification, design and implementation are inter-leaved

-Delivered system as a series of versions

-Frequent delivery with increments

-Minimal documentation

-Extensive tool support e.g CI, automated testing

**Agile development**

Specification, design, implement and testing interleaved. Suitable for:

-Small to medium sized product/firms

- Custom software development

**Agile Principles:**

**-**Customer Involvement

-Incremental delivery

-People not process

-Embrace change

-maintain simplicity

**Extreme Programming:**

-User stories for specification

-Refactoring

-Test-first development

-Pair programming

**Aim of agile**

-To reduce overhead in software process Reduced documentation

-Respond quickly to changing requirements

**Extreme programming**

-Incremental planning

-Small releases

-Simple design

-Test first

-Refactoring

-Pair programming

-Collective ownership

-Continuous integration

-Sustainable pace

-Customer involvement

**User stories**: To capture requirements

**Refactoring:** Constant code improvement to make change easier

**Test-first:** Write tests before code. Need test automation

**Scrum sprint:** The initial phase is where you establish the objectives for the project and design the software architecture. \*This is followed by a series of sprint cycles, where each cycle develops an increment of the system. \*The last phase completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

**Scrums:** The whole team attends short daily meetings (Scrums) where all team members share information

**Scrum Manager:** Ensures that the scrum properties are followed and protect developers from distractions

**Velocity:** how much backlog is covered in a single sprint?

**Agile problems**

Scaling up (for larger projects)

Scaling out (for larger organizations)

Legal approach to contracts

Cost of maintenance

**Agile dependencies**

People and Teams

How good are programmers

Team organization

Supporting technologies

Organizational issues

Plan based culture

Customer access

Communication and understanding in team

**Program testing**: Show program does what is intended. Execute program using artificial data (DYNAMIC)

**Validation:** Are we building the right product?

**Verification:** Are we building the product right?

**Inspections:** Analysis of the system to discover faults (STATIC).

**-**can be done on incomplete systems

-can find errors not possible in testing

-Inspections can consider broader quality attributes as well as program defects

-can’t check functional characteristics

**Static verification:** Code and document reviews

**Manual process:** Very effective to find hidden errors

**Testing:** Exercising and observing product behavior**: AKA** Dynamic verification

**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.

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

**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.

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

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

Partition testing

Guideline-based testing

**Partition Testing:** Groups of inputs with related groups of expected outputs are fed into unit tests

[**Guideline based testing**](http://iansommerville.com/software-engineering-book/web/path-testing/)**:** testing done in areas based on experience for knowing types of errors often made in development

**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

**Test-driven development process**

**-I**dentify 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

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

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

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

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

**Requirements based testing:** Verify functionality of system by executing test that correspond to system requirements

**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.

**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.

**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.

**Change:** Driven by system evolution

Continues throughout system lifetime

**Change implementation** done in iterations of design, implement and test

**Agile and evolution:** Incremental based so evolution is no different than development

**Legacy systems** are systems that rely on obsolete languages and technologies

**Legacy system replacement**

-Scrap system.

-Continue to maintain system

-Transform system

-Replace system

**Decision making factors**

-System replacement analysis

-Asset system quality

-Asset business value

-Maintenance costs

**Legacy Environment Assessment**

-Supplier Stability

-Failure Rate

-Age

-Performance

-Maintenance costs

-Programming tools

-People availability to maintain/develop

**Legacy Application Assessment**

-Documentation

-understanding of code and business rules

-Data

-Performance

-Maintenance costs

-Operation with other systems

**Software Maintenance:**

-Fault repairs-2

-Environmental Adaptation-3

-Functionality Addition-1

**Complexity metrics** determine maintainability by complexity of system components

**Software Reengineering:** Restructuring or rewriting part or all of a legacy system without changing its functionality.

-Source code translation

-Reverse engineering

-Program structure improvement

-Program modularisation

-Data reengineering

**Refactoring:** Improving program to slow down degradation.

**Bad smells in program code**

-Duplicate code

-Long methods

-Data Clumping

-Speculative generality

**Most important aspect of many systems**

-Dependability of the system

-Dependable system

-Reliability

-Availability

-Security

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

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

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

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

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

**Dependability costs** Increase exponentially with dependability requirement

**Holistic system design** caters for Interaction between all components and layers in a system

**Dependable process is Explicitly defined**

**Repeatable**

**-**Auditable(Modifiable and understood by all participants)

-Diverse

-Documented(Defined and followed)

-Robust(Recover from failures)

-Standardized(Applicable across multiple systems)

**Formal methods:** Approaches to software development based on mathematical representation and analysis. Formal approaches are:

**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.

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

**Issues with Formal Methods**

-Problem owners cannot understand a formal specification and so cannot assess if it is an accurate representation of their requirements.

-It is easy to assess the costs of developing a formal specification but harder to assess the benefits. Managers may therefore be unwilling to invest in formal methods.

-Software engineers are unfamiliar with this approach and are therefore reluctant to propose the use of FM.

-Formal methods are still hard to scale up to large systems.

-Formal specification is not really compatible with agile development methods.

**Application security** the application is designed to resist attacks

**Infrastructure security** the software is configured to resist attacks

**Operational security**, which is concerned with the secure operation and use of the organization’s systems.

**controls to enhance system security**

**-**Vulnerability avoidance

-Attack detection and neutralization

-Exposure limitation and recovery

**Stages of preliminary risk assessment**

-Asset identification

-Asset value assessment

-Exposure assessment

-Threat identification

-Attack assessment

-Control identification

-Security requirements definition

**Design guidelines for secure systems engineering**

-Base security decisions on an explicit security policy applied to all systems

-Avoid a single point of failure. – Distributed systems

-Fail securely- Encrypt or protect sensitive information incase system fails

-Use redundancy and diversity to reduce risk.- Backup and distributed data servers

-Validate all inputs

-log user actions

-Balance security and usability

-Modularize system so that each module is independent

-Design to avoid deployment problems

-Design to be recoverable

**Experience-based testing,** test where the system is analysed against known types of attack.

**Penetration testing** iswhere an external team is contracted to discover security flaws in a system.

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

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

**Interception threats** Allows attacker to gain access to an asset

**Interruption threats** Make part or all of a system unavailable

**Modification threats** Attacker tampers with a system asset

**Fabrication threats** Insert false information in the system

**Security Concerns/Dimensions**

***Confidentiality*** Information in a system may be disclosed or made accessible to people or programs that are not authorized to have access to that information.

***Integrity****:* Information in a system may be damaged or corrupted making it unusual or unreliable.

***Availability***: Access to a system or its data that is normally available may not be possible.

**Preliminary Risk Assessment (High Level):** Identify generic risks applicable to the system to assess the cost and level of security.

**Design Risk Assessment:** Occurs during development and is solved by implementation decisions and system design

**Operational Risk Assessment:** Focus on risks that arise from humans using the system

**Risk avoidance** requirements set out the risks that should be avoided by designing the system so that these risks simply cannot arise.

**Risk detection** requirements define mechanisms that identify the risk if it arises and neutralise the risk before losses occur.

**Risk mitigation** requirements set out how the system should be designed so that it can recover from and restore system assets after some loss has occurred.

**Platform-level protection:** Top-level controls on the platform on which a system runs.

**Application-level protection:** Specific protection mechanisms built into the application itself e.g. additional password protection.

**Record-level protection:** Protection that is invoked when access to specific information is requested

**Why are agile methods rarely used for dependable software development?**

A requirements specification is not available before implementation so up-front requirements analysis is impossible

Process documentation may be required to demonstrate that a dependable process has been followed

Formal change analysis is needed to assess the impact of change on safety and dependability and this conflicts with the constant refactoring and incremental development in agile methods

**Security requirements.**

-Identification requirements

-authentication requirements

-authorization requirements

-immunity requirements

-Integrity requirements

-intrusion detection requirements

-non-repudiation requirements

-privacy requirements

-Security auditing requirements

-system maintenance

-security requirements

**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

**Why is it expensive and risky to replace legacy systems with new systems?**

-There is rarely a complete specification of the legacy system

-Business processes may be tightly integrated with the legacy system and so have to change with a new system

-Business rules may be embedded in the code of the legacy system but not externally documented

-New software development is inherently risky and may go over budget.

**What are the two different approaches to process improvement and change that have been proposed**

**-**Agile Approach: which has focused on iterative development and the reduction of overheads in the software process

**-**Maturity Approach: level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes