Software in Modern World

National infrastructures and utilities are controlled by computer-based systems.

Most electrical products include a computer and controlling software.

Industrial manufacturing and distribution is completely computerized, as is the financial system.

Entertainment, including the music industry, computer games, and film and television, is software intensive.

Definitions

Software, Computer programs and associated documentation.

Software engineering is an engineering discipline that is concerned with all aspects of software production.

Computer science focuses on theory and fundamentals; **software engineering** is concerned with the practicalities of developing and delivering useful software.

Professional software, intended for use by someone apart from its developer, is usually developed by teams rather than individuals. It is maintained and changed throughout its life.

Software Failures;

Increasing Demands, -Systems have to be built and delivered more quickly. -Larger, even more complex systems are required. -Systems have to have new capabilities that were previously thought to be impossible.

Low Expectations, -It is relatively easy to write computer programs without using software engineering methods and techniques. -Many companies do not use software engineering methods. -Consequently, their software is often more expensive and less reliable than it should be.

Essential Attributes of Good Software

Maintainability; Software should be written in such a way so that it can evolve to meet the changing needs of customers.

Dependable software; should not cause physical or economic damage in the event of system failure.

Efficiency; Software should not make wasteful use of system resources such as memory and processor cycles.

Acceptability; means that it must be understandable, usable, and compatible with other systems that they use.

Kinds of Software Products

Generic products (MS Word), These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them.

Customized products (Submit System), These are systems that are commissioned by a particular customer. A software contractor develops the software especially for that customer.

Quality

When we talk about **the quality** of professional software, we have to take into account that the software is used and changed by people apart from its developers. **Quality** is therefore not just concerned with what the software does; it has to include the software's behavior while it is executing and the structure and organization of the system programs and associated documentation.

General Issues of Software; Heterogeneity, Business and social change & Security and trust.

Software engineering is important for two reasons:

- More and more, individuals and society rely on advanced software systems. We need to be able to **produce reliable and trustworthy** systems economically and quickly.
- It is usually cheaper, in the long run, to **use software engineering methods and techniques** for software systems rather than just write the programs as if it was a personal programming project.

Software Engineering vs. Computer Science

Computer science is concerned with the theories and methods that underlie computers and software systems, whereas **software engineering** is concerned with the practical problems of producing software.

Software Engineering vs. System Engineering

System engineering is concerned with all aspects of the development and evolution of complex systems where software plays a major role. **System engineering** is therefore concerned with hardware development, policy and process design and system deployment, as well as software engineering.

General Issues of Software

Heterogeneity; Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices. You often have to integrate new software with older legacy systems written in different programming languages. The challenge here is to develop techniques for building dependable software that is flexible enough to cope with this heterogeneity.

Business and social change; Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. Many traditional software engineering techniques are time consuming and delivery of new systems often takes longer than planned.

Security and trust; As software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface. We have to make sure that malicious users cannot attack our software and that information security is maintained.

Software Process

Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.

Software development, where the software is designed and programmed.

Software validation, where the software is checked to ensure that it is what the customer requires.

Software evolution, where the software is modified to reflect changing customer and market requirements.

Different types of systems need different development processes.

For example, **real-time software in an aircraft** has to be completely specified before development begins.

In e-commerce systems, the specification and the program are usually developed together.

Software Ethics

Confidentiality; You should normally respect the confidentiality of your employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.

Competence; You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.

Intellectual property rights; You should be aware of local laws governing the use of intellectual property such as patents and copyright.

Computer misuse; You should not use your technical skills to misuse other people's computers.

Software Process, a structured set of activities required to develop a software system.

Software Process Descriptions

Products, which are the outcomes of a process activity.

Roles, which reflect the responsibilities of the people involved in the process.

Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

Process Standardization

Software processes can be improved by process standardization where the **diversity in software processes** across an organization is reduced. This leads to improved communication and a reduction in training time, and makes automated process support more economical.

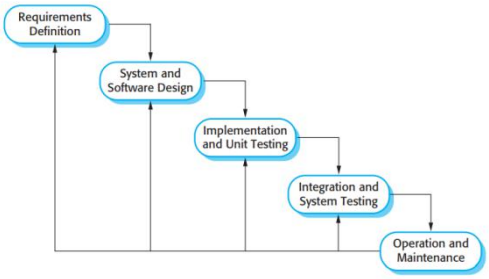
Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.

In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.

Software Process Models

The waterfall model;

- Plan-driven model. (Her şeyi önceden planla!)
- Separate and distinct phases of specification and development.



There are separate identified phases in the waterfall model:

Requirements analysis and definition, The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

System and software design, The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.

Implementation and unit testing, The software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

Integration and system testing, The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met.

Operation and maintenance, The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

Waterfall Model - Benefits

The waterfall model is **consistent with other engineering process models.**

As is **easier to use a common management model in other engineering projects** for the whole project, software processes based on the waterfall model are still commonly used.

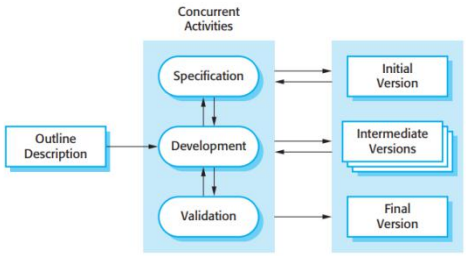
Waterfall Model - Problems

It is difficult to accommodate change after the process is underway. A phase has to be complete before moving onto the next phase.

Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements. Few business systems have stable requirements.

Incremental development;

- Specification, development and validation are interleaved.
- May be plan-driven or agile.



Incremental development **reflects the way that we solve problems.**

We rarely work out a complete problem solution in advance but move toward a solution in a series of steps, backtracking when we realize that we have made a mistake.

By developing the software incrementally, **it is cheaper and easier to make changes in the software** as it is being developed.

Each **increment** or version of the system incorporates some of the functionality that is needed by the customer.

Generally, the early increments of the system include **the most important or most urgently required functionality.**

Incremental Development - Benefits

The cost of accommodating changing customer requirements is reduced.

The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.

It is easier to get customer feedback on the development work that has been done.

Customers can comment on demonstrations of the software and see how much has been implemented.

More rapid delivery and deployment of useful software to the customer is possible.

Incremental Development - Problems

The process is not visible. Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.

System structure tends to degrade as new increments are added. Unless time and money is spent on Re-factoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

Reuse-oriented software engineering (ROSE);

- The system is assembled from existing components.
- May be plan-driven or agile.

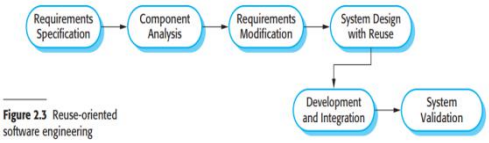


Figure 2.3 Reuse-oriented software engineering

In the majority of software projects, there is some **software reuse.**

This often happens informally when people working on the project know of designs or code that are similar to what is required. They look for these, modify them as needed, and incorporate them into their system.

Reuse-oriented approaches rely on a large base of reusable software components and **an integrating framework** for the composition of these components.

ROSE - Benefits & Problems

Benefits; ROSE has the obvious advantage of **reducing the amount of software to be developed and so reducing cost and risks.** It usually also leads to **faster delivery of the software.**

Problems; However, requirements compromises are inevitable and this **may lead to a system that does not meet the real needs of users.**

Furthermore, **some control over the system evolution is lost** as new versions of the reusable components are not under the control of the organization using them.

What is a Project?

A project is a temporary endeavour undertaken to create a unique product or service.

"Temporary", Has a start and an end date.

"Unique", Has specific attributes for its customers.

What is a Project Management?

Project management is the application of knowledge, skills, tools, and techniques to project activities in order to meet project requirements.

Project Management is accomplished through the appropriate application of five process groups which are: *Initiating, Planning, Executing, Monitoring and Controlling,* and *Closing.*

Software management distinctions

The product is intangible; Software cannot be seen or touched. Software project managers cannot see progress by simply looking at the artefact that is being constructed.

Many software projects are 'one-off' projects; Large software projects are usually different in some ways from previous projects. Even managers who have lots of previous experience may find it difficult to anticipate problems.

Software processes are variable & organization specific; We still cannot reliably predict when a particular software process is likely to lead to development problems.

Static work-flows in the Rational Unified Process

The business processes, are modeled using business use cases.

Requirements, Actors who interact with the system are identified and use cases are developed to model the system requirements.

Analysis and design, model is created and documented using architectural models, component models, object models, and sequence models.

Implementation, the components in the system are implemented and structured into implementation sub-systems.

Testing, is an iterative process that is carried out in conjunction with implementation.

Deployment, A product release is created, distributed to users, and installed in their workplace.

Configuration and change management, this supporting work-flow manages changes to the system.

Environment this work-flow is concerned with making appropriate software tools available to the software development team.

Management Activities

Project planning; Project managers are responsible for planning. estimating and scheduling project development and assigning people to tasks.

Reporting; Project managers are usually responsible for reporting on the progress of a project to customers and to the managers of the company developing the software.

Risk management; Project managers assess the risks that may affect a project, monitor these risks and take action when problems arise.

People management; Project managers have to choose people for their team and establish ways of working that leads to effective team performance.

Proposal writing; The first stage in a software project may involve writing a proposal to win a contract to carry out an item of work. The proposal describes the objectives of the project and how it will be carried out.

Cohesive Group

Group quality standards can be developed by the group members.

Team members learn from each other and get to know each others work; Inhibitions caused by ignorance are reduced.

Knowledge is shared. Continuity can be maintained if a group member leaves.

Re-factoring and continual improvement is encouraged. Group members work collectively to deliver high quality results and fix problems, irrespective of the individuals who originally created the design or program.

Risk

Risk management is concerned with identifying risks and drawing up plans to minimize their effect on a project.

Project risks affect schedule or resources;

Product risks affect the quality or performance of the software being developed;

Business risks affect the organization developing or procuring the software.

Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organizational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule.
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool underperformance	Product	CASE tools, which support the project, do not perform as anticipated.
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

Risk Management Process

Risk identification, Identify project, product and business risks. **Output:** List of Potential Risks

Risk analysis, Assess the likelihood and consequences of these risks. **Output:** Prioritized Risk List.

Risk planning, Draw up plans to avoid or minimize effects of the risk. **Output:** Risk Avoidance and Contingency Plans.

Risk monitoring, Monitor the risks throughout the project. **Output:** Risk Assessment.

Risk Examples

Technology	Reusable software components contain defects that mean they cannot be reused as planned.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unavailable at critical times.
Organizational	Organizational financial problems force reductions in the project budget.
Tools	Software tools cannot work together in an integrated way.
Requirements	Changes to requirements that require major design rework are proposed.
Estimation	The rate of defect repair is underestimated. The size of the software is underestimated.

Risk Planning

Avoidance strategies, the probability that the risk will arise is reduced.

Minimization strategies, the impact of the risk on the project or product will be reduced.

Contingency plans, If the risk arises, contingency plans are plans to deal with that risk.

Project Planning

Project planning involves:

- breaking down the work into parts,
- assigning these parts to project team members,
- anticipating problems that might arise, and
- preparing tentative solutions to those problems.

The project plan, which is created at the start of a project, is used: to communicate how the work will be done to the project team and customers, and to help assess progress on the project.

Planning stages

At the proposal stage, when you are bidding for a contract to develop or provide a software system.

During the project startup phase, when you have to plan who will work on the project, how the project will be broken down into increments, how resources will be allocated across company, etc.

Periodically throughout the project, when you modify your plan in the light of experience gained and information from monitoring the progress of the work.

Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail. A project plan is created that records the work to be done, who will do it, the development schedule and the work products. Managers use the plan to support project decision making and as a way of measuring progress.

Milestones are points in the schedule against which you can assess progress, for example, the handover of the system for testing.

Deliverables are work products that are delivered to the customer, e.g. a requirements document for the system.

Project Scheduling

Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.

Project Plan

Quality plan, describes the quality procedures and standards that will be used in a project.

Validation plan, describes the approach, resources, and schedule used for system validation.

Configuration management plan, describes the configuration management procedures and structures to be used.

Maintenance plan, predicts the maintenance requirements, costs, and effort.

Staff development plan, describes how the skills and experience of the project team members will be developed.

Software Evolution

Software evolution processes depend on;

- The type of software being maintained,
- The development processes used,
- The skills and experience of the people involved.

Importance of Evolution

Organizations have huge investments in their software systems - they are critical business assets.

To maintain the value of these assets to the business, they must be changed and updated.

The majority of the software budget in large companies is devoted to changing and evolving existing software rather than developing new software.

Evolution & Servicing

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; At this stage, 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 software's environment. No new functionality is added.

Phase-out; The software may still be used but no further changes are made to it.

"Handover" Problems

Development team have **used an agile approach** but the evolution team **prefer a plan-based approach**.

The evolution team may expect detailed documentation to support evolution and this is not produced in agile processes.

A plan-based approach has been used for development but the evolution team **prefer to use agile methods.**

The evolution team may have to start from scratch developing automated tests and the code in the system may not have been re-factored and simplified as is expected in agile development.

Types of software systems:

S-Type	Conforms to an exact specification. **Specification based** → mathematical criteria.
P-Type	Consistent with a single external paradigm. **Paradigmatic domain** → stable assumptions. v.g. Unicode, Physics simulation
E-Type	Actively used and embedded in the real world. **Evolutionary systems** → constantly adapting to change.

Lehman's Laws of Software Evolution

Law	Description
Continuing change	A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases, and the number of reported errors is approximately invariant for each system release.
Organizational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will decline unless they are modified to reflect changes in their operational environment.
Feedback system	Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

Types of Maintenance

Adaptive, modifying the system to cope with changes in the software environment (DBMS, OS).

Perfective, implementing new or changed user requirements which concern functional enhancements to the software.

Corrective, diagnosing and fixing errors, possibly ones found by users.

Preventive, increasing software maintainability or reliability to prevent problems in the future.

Maintenance Cost Factors

Team stability; Maintenance costs are reduced if the same staff are involved with them for some time.

Contractual responsibility; The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change.

Staff skills; Maintenance staff are often inexperienced and have limited domain knowledge.

Program age and structure; As programs age, their structure is degraded and they become harder to understand and change.

Maintenance Prediction

Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs;

Change acceptance depends on the maintainability of the components affected by the change,

Implementing changes degrades the system and reduces its maintainability,

Maintenance costs depend on the number of changes and costs of change depend on maintainability.

Change Prediction

Predicting the number of changes requires and understanding of the relationships between a system and its environment.

Tightly coupled systems require changes whenever the environment is changed.

Factors influencing this relationship are;

Number and complexity of system interfaces,

Number of inherently volatile system requirements,

The business processes where the system is used.

System Re-engineering / Re-factoring

Re-structuring or re-writing part or all of a **legacy system** without changing its functionality.

Applicable where some but not all sub-systems of a larger system require frequent maintenance.

Re-engineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented. Also called **"re-factoring".**

Re-factoring is the process of making improvements to a program to slow down degradation through change.

You can think of **Re-factoring** as **'preventative maintenance'** that reduces the problems of future change.

Re-factoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand.

Advantages of Re-engineering / Re-factoring

Reduced risk; There is a high risk in new software development. There may be development problems, staffing problems and specification problems.

Reduced cost; The cost of re-engineering is often significantly less than the costs of developing new software.

Re-engineering Process Activities

Source code translation; Convert code to a new language.

Reverse engineering; Analyze the program to understand it.

Program structure improvement; Restructure automatically for understandability;

Program Modularization; Reorganize the program structure;

Data Re-engineering; Clean-up and restructure system data.

'Bad smells' in Program Code

Duplicate code; The same or very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.

Long methods; If a method is too long, it should be redesigned as a number of shorter methods.

Switch (case) statements; These often involve duplication, where the switch depends on the type of a value. The switch statements may be scattered around a program. In object-oriented languages, you can often use Polymorphism to achieve the same thing.

Data clumping; Data clumps occur when the same group of data items (fields in classes, parameters in methods) re-occur in several places in a program. These can often be replaced with an object that encapsulates all of the data.

Speculative generality; This occurs when developers include generality in a program in case it is required in the future. This can often simply be removed.

Legacy System Management

Organizations that rely on **legacy systems** must choose a strategy for evolving these systems;

Scrap the system completely and modify business processes so that it is no longer required,

Continue maintaining the system,

Transform the system by re-engineering to improve its maintainability,

Replace the system with a new system.

System Quality Assessments

Business process assessment; How well does the business process support the current goals of the business?

Environment assessment; How effective is the system's environment and how expensive is it to maintain?

Application assessment; What is the quality of the application software system?

Configure Management

Definition: Interrelated functional and physical characteristics of a product defined in product configuration information.

Why Do We Need Configure Management?

Because **software changes frequently,** systems, can be thought of as a set of versions, each of which has to be maintained and managed.

CM Activities

CHANGE MANAGEMENT; Keeping track of requests for changes to the software from customers and developers, working out the costs and impact of changes, and deciding the changes should be implemented.

VERSION MANAGEMENT; Keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other.

SYSTEM BUILDING; The process of assembling program components, data and libraries, then compiling these to create an executable system.

RELEASE MANAGEMENT; Preparing software for external release and keeping track of the system versions that have been released for customer use.

Terminology

Software Configuration Item (SCI); Anything associated with a software project (design, code, test data, document, etc.) that has been placed under configuration control. There are often different versions of a configuration item. Configuration items have a unique name.

Version; An instance of a configuration item that differs, in some way, from other instances of that item. Versions always have a unique identifier, which is often composed of the configuration item name plus a version number.

Baseline; A baseline is a collection of component versions that make up a system. Baselines are controlled, which means that the versions of the components making up the system cannot be changed. This means that it should always be possible to recreate a baseline from its constituent components.

Release; A version of a system that has been released to customers (or other users in an organization) for use.

Term	Explanation
Mainline	A sequence of baselines representing different versions of a system.
Release	A version of a system that has been released to customers (or other users in an organization) for use.
Workspace	A private work area where software can be modified without affecting other developers who may be using or modifying that software.
Branching	The creation of a new codeline from a version in an existing codeline. The new codeline and the existing codeline may then develop independently.
Merging	The creation of a new version of a software component by merging separate versions in different codelines. These codelines may have been created by a previous branch of one of the codelines involved.
System building	The creation of an executable system version by compiling and linking the appropriate versions of the components and libraries making up the system.

Software Change Management 4 Stages

Change Request	a 'customer' completes and submits a change request describing the change required to the system
Change Assessment	The impact of the change on the rest of the system must be checked. A separate group decides if it is cost-effective from a business perspective to make the change.
Change Implementation	Accepted changes are passed back to the development group for implementation.
Change Validation	Implemented changes are validated by software testing (as any other requirements implementation), and QA people approves the implementation of the change.
Version:	An instance of a configuration item that differs, in some way, from other instances of that item.
Baseline:	A baseline is a collection of component versions that make up a system. Baselines are controlled, which means that the versions of the components making up the system cannot be changed.

Software Quality

Software quality management is a management process, the goal of which is to develop and manage the quality of a software to make sure the product satisfies the user.

Quality management provides an independent check on the software development process. It checks project deliverables to ensure they are consistent with organizational standards and goals.

Three principal concerns:

At the organizational level, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high-quality software.

At the project level, quality management involves the application of specific quality processes and checking that these planned processes have been followed.

At the project level, quality management is also concerned with establishing a quality plan for a project. The quality plan should set out the quality goals for the project and define what processes and standards are to be used.

Software Quality Attributes

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Process & Product Quality

The quality of a developed product is influenced by the quality of the production process.

This is important in software development as some product quality attributes are hard to assess.

However, **there is a very complex and poorly** understood relationship between software processes and product quality;

The application of individual skills and experience is particularly important in software development,

External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality.

Software Standards

Standards define the required attributes of a product or process. They play an important role in quality management. Standards may be international, national, organizational or project standards.

Product standards define characteristics that all software components should exhibit. e.g. a common programming style.

Process standards define how the software process should be enacted.

Product & Process Standards

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

How To Achieve Quality?

Traditional Approach: **Defect detection**

"Quality Control". Achieving quality by targeting the product/service that is produced.

Modern Approach: **Defect prevention**

"Quality Assurance". Achieving quality by targeting the system that produces the product/service.

Quality Through Product: Methods

Static Methods: Applied by not executing the code (please refer to IEEE 1028:2008)

Management review, Technical review, Inspection, Walk-through, Audit

Dynamic methods: Applied by executing the code

Unit testing, Integration testing, System testing, Acceptance testing

Program Inspections

These are **peer reviews** where engineers examine the source of a system with the aim of discovering anomalies and defects.

Inspections do not require execution of a system so may be used before implementation.

They may be applied to any representation of the system (requirements, design, configuration data, test data, etc.).

They have been shown to be an effective technique for

discovering program errors.

Scope of Quality Management

Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.

Quality, simplistically, means that a product should meet its specification.

The review process in agile software development is usually informal. In Scrum, for example, there is a review meeting after each iteration of the software has been completed (a sprint review), where quality issues and problems may be discussed.

Software measurement is concerned with deriving a numeric value for an attribute of a software product or process.

Importance of Standards;

Encapsulation of best practice - avoids repetition of past mistakes.

They are a framework for defining what quality means in a particular setting i.e. that organization's view of quality.

They provide continuity - new staff can understand the organization by understanding the standards that are used.

Agile Method & Inspection

Agile processes rarely use formal inspection or peer review processes.

Rather, they rely on team members cooperating to check each others code, and informal guidelines, such as 'check before check-in', which suggest that programmers should check their own code.

Extreme programming practitioners argue that pair programming is an effective substitute for inspection as this is, in effect, a continual inspection process.

Two people look at every line of code and check it before it is accepted.

Product Metrics

Dynamic metrics are closely related to software quality attributes. It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute).

Static metrics have an indirect relationship with quality attributes. You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.

Coping With Change

Change is inevitable in all large software projects.

Business changes lead to new and changed system requirements.

New technologies open up new possibilities for improving implementations.

Changing platforms require application changes.

Reducing The Costs of Rework

Change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required. For example, **a prototype system** may be developed to show some key features of the system to customers.

Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost. This normally involves some form of **incremental development.**

1. Software Prototyping

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

May be based on **rapid prototyping languages or tools.**

A prototype can be used in;

The requirements engineering process to help with requirements elicitation and validation,

In design processes to explore options and develop a user interface (UI) design,

In the testing process to run back-to-back tests.

Benefits of Prototyping

Improved system usability, A closer match to users' real needs, Improved design quality, Improved maintainability, Reduced development effort.

2. Incremental Delivery

Rather than delivering the system as a single delivery, the development and delivery is broken down into increments with **each increment delivering part of the required functionality.**

User requirements are prioritized and the **highest priority requirements are included in early increments.**

Incremental Development & Delivery

Incremental development; Develop the system in increments and evaluate each increment before proceeding to the development of the next increment, normal approach used in agile methods.

Evaluation done by user/customer proxy.

Incremental delivery; Deploy an increment for use by end-users. More realistic evaluation about practical use of software. Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

Benefits of Incremental Delivery

Customer value can be delivered with each increment so **system functionality is available earlier.**

Early increments act as a prototype to help elicit requirements for later increments.

Lower risk of overall project failure.

The highest priority system services tend to receive the most testing.

Problems of Incremental Delivery

Most systems require a set of basic facilities that are used by different parts of the system.

As requirements are not defined in detail until an increment is to be implemented, it can be **hard to identify common facilities that are needed by all increments.**

3. Boehm's Spiral Model

Process is represented as **a spiral rather than as a sequence of activities with backtracking.**

Each loop in the spiral represents **a phase** in the process.

No fixed phases such as specification or design - **loops in the spiral are chosen depending on what is required.**

Risks are explicitly assessed and resolved throughout the process.

4. The Rational Unified Process (RUP)

A modern generic process derived from the work on the UML (Unified Modeling Language) and associated process.

Brings together aspects of the 3 generic process models discussed previously. Normally described from 3 perspectives:

A dynamic perspective that shows phases over time,

A static perspective that shows process activities.

A practice perspective that suggests good practice.

RUP Phases

Inception; Establish the business case for the system.

Elaboration; Develop an understanding of the problem domain and the system architecture.

Construction; System design, programming and testing.

Transition; Deploy the system in its operating environment.

RUP Iterations

In-phase iteration; Each phase is iterative with results developed incrementally.

Cross-phase iteration; As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.

Plan Driven vs. Agile Development

Plan-driven development

A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.

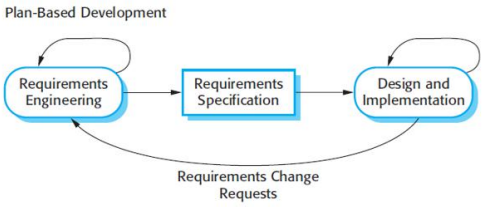
Not necessarily waterfall model – plan-driven,

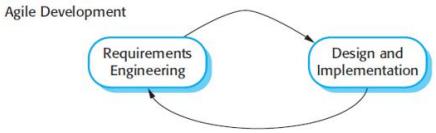
incremental development is possible.

Iteration occurs within activities.

Agile development

Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.





Agile Methods

Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods;

Focus on the code rather than the design,

Are based on an iterative approach to software development,

Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.

Extreme Programming

Perhaps the best-known and most widely used agile method.

Extreme Programming (XP) takes an 'extreme' approach to 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.

XP & Agile Principles

Incremental development is supported through small, frequent system releases.

Customer involvement means full-time customer engagement with the team.

People not process through pair programming, collective ownership and a process that avoids long working hours.

Change supported through regular system releases.

Maintaining simplicity through constant Re-factoring of code.

XP & Change

Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.

XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.

Rather, it proposes constant code improvement (Re-factoring) to make changes easier when they have to be implemented.

Re-factoring

Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.

Examples;

Re-organization of a class hierarchy to remove duplicate code,

Tidying up and renaming attributes and methods to make them easier to understand,

The replacement of inline code with calls to methods that have been included in a program library.

Agile Project Management

The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.

The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.

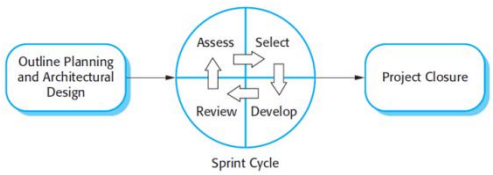
Agile project management requires a different approach, which is adapted to incremental development and the particular strengths of agile methods.

Scrum

The Scrum approach is a general agile method but its focus is on managing iterative development rather than specific agile practices.

There are three phases in Scrum.

- The initial phase is an outline planning phase where you establish the general 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 project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.



Sprint Cycle

Sprints are fixed length, normally 2–4 weeks. They correspond to the development of a release of the system in XP.

The starting point for planning is the product backlog, which is the list of work to be done on the project.

The selection phase involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.

Scrum Benefits

The product is broken down into a set of **manageable and understandable chunks.**

Unstable requirements do not hold up progress.

The whole team have **visibility** of everything and consequently team communication is improved.

Customers see on-time delivery of increments and gain feedback on how the product works.

Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

ROSE Stages

Component analysis; Given the requirements specification, a search is made for components to implement that specification. Usually, there is no exact match and the components that may be used only provide some of the functionality required.

Requirements modification; During this stage, the requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components. Where modifications are impossible, the component analysis activity may be re-entered to search for alternative solutions.

System design with reuse; During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused and organize the framework to cater for this. Some new software may have to be designed if reusable components are not available.

Development and integration; Software that cannot be externally procured is developed, and the components and COTS systems are integrated to create the new system. System integration, in this model, may be part of the development process rather than a separate activity.

The Rational Unified Process (RUP) is a modern generic process model that is organized into phases (inception, elaboration, construction and transition) but separates activities (requirements, analysis and design, etc.) from these phases.

RUP Good Practices

Develop software iteratively; Plan increments based on customer priorities and deliver highest priority increments first.

Manage requirements; Explicitly document customer requirements and keep track of changes to these requirements.

Use component-based architectures; Organize the system architecture as a set of reusable components.

Visually model software; Use graphical UML models to present static and dynamic views of the software.

Verify software quality; Ensure that the software meet's organizational quality standards.

Control changes to software; Manage software changes using a change management system and configuration management tools.

Problems With Agile Methods

It can be difficult to keep the interest of customers who

are involved in the process.

Team members may be unsuited to the intense involvement **that characterizes agile methods.**

Prioritizing changes can be difficult where there are multiple stakeholders.

Maintaining simplicity requires extra work.

Contracts may be a problem as with other approaches to iterative development.

Test Automation; As testing is automated, there is always a set of tests that can be quickly and easily executed.

Scaling Agile Methods

Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team.

It is sometimes argued that the success of these methods comes because of improved communications which is possible when everyone is working together.

Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations.

Scaling-out & Scaling-up

'Scaling up' is concerned with using agile methods for developing large software systems that cannot be developed by a small team.

'Scaling out' is concerned with how agile methods can be introduced across a large organization with many years of software development experience.

When scaling agile methods it is essential to maintain agile fundamentals;

Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

Q - Software Architecture Type

Software Application	(Map)	Architecture Type	Description of Architecture Type & Mapping Rationale
MS Windows Operating System		Model-View-Controller	Separates presentation and interaction from the system data. Used when there are multiple ways to view and interact with data. / Students, supervisors, and administrators interact in different ways with data in different times (registration period vs. in-term)
Spotify Web Player		Layered	Organizes the system into layers with related functionality associated with each layer. Used when building new facilities on top of existing systems. / OS architecture (kernel and shell, each with its sub-layers) fits well with layered architecture.
HU Course Registration System		Repository	All data in a system is managed in a central repository that is accessible to all system components. You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. / Developement data are various and large in volumes and need storage for a long time.
Github		Client-Server	The functionality of the system is organized into services, with each service delivered from a separate server. Used when data in a shared database has to be accessed from a range of locations. / Music database and its features are accessible by any client user at any geographic location via replicated servers.

Q - Types of Review (V Development Model)

Reviews		Tests
Requirements review		System testing
Architectural (technical) review		Integration testing
Detailed design (technical) review or inspection		Component testing
Code review or inspection		Unit testing

Q - Iterative vs Waterfall Model

Your selection (Waterfall / Iterative ?)	Rationale of your selection (Why did you choose it?)	Rationale of discard for the other (Why did you discard the other?)
Iterative	*(Any two answers will be accepted)* - Allows interaction with customer during development - Customer requirements can be prioritized and be developed by increments - Any change requests can always be included in the next iterations - Customer satisfaction is more important than obeying to contract requirements	*(Any two answers will be accepted)* - Customers typically do not interact with development team during development - Requires complete specification of requirements before development - Does not welcome changes after requirements stage - Typically requires a contract, which is not the case with in-house development

Q - Agile vs Plan-Driven Model

Agile Methodology (values these factors)	(over these in) **Plan-Driven Methodology**
1) Individuals and interactions	processes and tools
2) Working software	comprehensive documentation
3) Customer collaboration	contract negotiation
4) Responding to change	following a plan

1. INTRODUCTION

1.2 What is the most important difference between generic software product development and custom software development? What might this mean in practice for users of generic software products?

The essential difference is that in generic software product development, the specification is owned by the product developer. For custom product development, the specification is owned and controlled by the customer. The implications of this are significant – the developer can quickly decide to change the specification in response to some external change (e.g. a competing product) but, when the customer owns the specification, changes have to be negotiated between the customer and the developer and may have contractual implications.For users of generic products, this means they have no control over the software specification so cannot control the evolution of the product. The developer may decide to include/exclude features and change the user interface. This could have implications for the user's business processes and add extra training costs when new versions of the system are installed. It also may limit the customer's flexibility to change their own business processes.

1.3 What are the four important attributes that all professional software should have? Suggest four other attributes that may sometimes be significant.

Go to Page 1 > *Essential Attributes of Good Software*

1.4 Apart from the challenges of heterogeneity, business and social change and trust and security, identify other problems and challenges that software engineering is likely to face in the 21st century (hint: think about the environment) Go to Page 1 > *General Issues of Software*

There are many possible challenges that could be identified. These include:

1. Developing systems that are energy-efficient. This makes them more usable on low power mobile devices and helps reduce the overall carbon footprint of IT equipment.
2. Developing validation techniques for simulation systems (which will be essential in predicting the extent and planning for climate change).
3. Developing systems for multicultural use.
4. Developing systems that can be adapted quickly to new business needs.
5. Designing systems for outsourced development.
6. Developing systems that are resistant to attack.
7. Developing systems that can be adapted and configured by end-users.
8. Finding ways of testing, validating and maintaining end-user developed systems.

2. SIFTWARE PROCESS

2.1 What are the advantages of providing static and dynamic views of the software process as in the Rational Unified Process?

An approach to process modeling which is simply based on static activities, such as requirements, implementation, etc. forces these activities to be set out in a sequence which may not reflect the actual way that these are enacted in any one organization. In most cases, the static activities shown in Figure 2.13 are actually interleaved so a sequential process model does not accurately describe the process used. By separating these from the dynamic perspective i.e. the phases of development, you can then discuss how each of these static activities may be used at each phase of the process. Furthermore, some of the activities that are required during some of the system phases are in addition to the central static activities shown in Figure 2.13. These vary from one organization to another and it is not appropriate to impose a particular process in the model.

2.2 Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software process model that might be used as a basis for managing the development of the following systems:

- A system to control anti-lock braking in a car.
- A virtual reality system to support software maintenance.
- A university accounting system that replaces an existing system.
- An interactive travel planning system that helps users plan journeys with the lowest environmental impact.

1. Anti-lock braking system This is a safety-critical system so requires a lot of up-front analysis before implementation. It certainly needs a plan-driven approach to development with the requirements carefully analyzed. A waterfall model is therefore the most appropriate approach to use, perhaps with formal transformations between the different development stages.

2. Virtual reality system This is a system where the requirements will change and there will be an extensive user interface components. Incremental development with, perhaps, some UI prototyping is the most appropriate model. An agile process may be used.

3. University accounting system This is a system whose requirements are fairly well-known and which will be used in an environment in conjunction with lots of other systems such as a research grant management system. Therefore, a reuse-based approach is likely to be appropriate for this.

4. Interactive travel planning system System with a complex user interface but which must be stable and reliable. An incremental development approach is the most appropriate as the system requirements will change as real user experience with the system is gained.

3. AGILE SOFTWARE DEVELOPMENT

3.1 Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.

The principles underlying agile development are:

1. Individual and interactions over processes and tools. By taking advantages of individual skills and ability and by ensuring that the development team know what each other are doing, the overheads of formal communication and process assurance are avoided. This means that the team can focus on the development of working software.

2. Working software over comprehensive documentation. This contributes to accelerated development because time is not spent developing, checking and managing documentation. Rather, the programmer's time is focused on the development and testing of code.

3. Customer collaboration over contract negotiation. Rather than spending time developing, analyzing and negotiating requirements to be included in a system contract, agile developers argue that it is more effective to get feedback from customer's directly during the development about what is required. This allows useful functionality to be developed and delivered earlier than would be possible if contracts were required.

4. Responding to change over following a plan. Agile developers argue (rightly) that being responsive to change is more effective than following a plan-based process because change is inevitable whatever process is used. There is significant overhead in changing plans to accommodate change and the inflexibility of a plan means that work may be done that is later discarded.

3.2 When would you recommend against the use of an agile method for developing a software system?

Agile methods should probably not be used when the software is being developed by teams who are not co-located. If any of the individual teams use agile methods, it is very difficult to coordinate their work with other teams. Furthermore, the informal communication which is an essential part of agile methods is practically impossible to maintain.

Agile methods should probably also be avoided for critical systems where the consequences of a specification error are serious. In those circumstances, a system specification that is available before development starts makes a detailed specification analysis possible. However, some ideas from agile approaches such as test first development are certainly applicable to critical systems.

3.3. Extreme programming expresses user requirements as stories, with each story written on a card. Discuss the advantages and disadvantages of this approach to requirements description.

Advantages of stories:

1. They represent real situations that commonly arise so the system will support the most common user operations.
2. It is easy for users to understand and critique the stories.
3. They represent increments of functionality – implementing a story delivers some value to the user.

Disadvantages of stories:

1. They are liable to be incomplete and their informal nature makes this incompleteness difficult to detect.
2. They focus on functional requirements rather than non-functional requirements.
3. Representing cross-cutting system requirements such as performance and reliability is impossible when stories are used.
4. The relationship between the system architecture and the user stories is unclear so architectural design is difficult.

4. SOFTWARE EVOLUTION

4.1 Explain why a software system that is used in a real-world environment must change or become progressively less useful.

Systems must change or become progressively less useful for a number of reasons:

1. The presence of the system changes the ways of working in its environment and this generates new requirements. If these are not satisfied, the usefulness of the system declines.

2. The business in which the system is used changes in response to market forces and this also generates new system requirements.

3. The external legal and political environment for the system changes and generates new requirements.

4. New technologies become available that offer significant benefits and the system must change to take advantage of them.

4.2 Briefly describe the three main types of software maintenance. Why is it sometimes difficult to distinguish between them?

The three main types of software maintenance are:

1. Corrective maintenance or fault repair. The changes made to the system are to repair reported faults which may be program bugs or specification errors or omissions.

2. Adaptive maintenance or environmental adaptation. Changing the software to adapt it to changes in its environment e.g. changes to other software systems.

3. Perfective maintenance or functionality addition. This involves adding new functionality or features to the system.

4.3 What are the strategic options for legacy system evolution? When would you normally replace all or part of a system rather than continue maintenance of the software?

The strategic options for legacy system evolution are:

1. Abandon maintenance of the system and replace it with a new system.

2. Continue maintaining the system as it is.

3. Perform some re-engineering (system improvement) that makes the system easier to maintain and continue maintenance.

4. Encapsulate the existing functionality of the system in a wrapper and add new functionality by writing new code which calls on the existing system as a component.

5. Decompose the system into separate units and wrap them as components.

This is similar to the solution above but gives more flexibility in how the system is used. You would normally choose the replacement option in situations where the hardware platform for the system is being replaced, where the company wishes to standardize on some approach to development that is not consistent with the current system, where some major sub-system is being replaced (e.g. a database system) or where the technical quality of the existing system is low and there are no current tools for re-engineering.

5. SOFTWARE RE-USE

5.1 Suggest why the savings in cost from reusing existing software are not simply proportional to the size of the components that are reused.

If savings from reuse were proportional to the amount of code reused, then reusing 2000 lines of code would save twice as much as reusing 1000 lines of code. However, to reuse 2000 lines of code, that code must be understood and the costs of program understanding are not linear – the more code to be understood, the more effort it takes to understand it. Furthermore, more changes may be required, the larger the amount of code reused so this also adds to the costs of reusing more code. Of course, all this is only true if the code has to be understood before it is reused. If it can be reused without change, then savings from reusing large chunks of code tend to be proportionally greater than savings from reusing small code fragments.

5.2 Give four circumstances where you might recommend against software reuse.

Circumstances where software reuse is not recommended:

1. If the business status of the code provider is dubious. If the provider goes out of business, then no support for the reused code may be available.

2. In critical applications where source code is not available. Testing the code to the required standards may be very difficult.

3. In small systems where the costs of reuse are comparable to the savings that result if code is reused.

4. In systems where performance is a critical requirement – specially developed code can usually be made more efficient.

6. PROJECT MANAGEMENT

6.1 Explain why the best programmers do not always make the best software managers.

Management activities such as proposal writing, project planning and personnel selection require a set of skills including presentation and communication skills, organizational skills and the ability to communicate with other project team members.Programming skills are distinct from these (indeed, it is a common criticism of programmers that they lack human communication skills) so it does not follow that good programmers can re-orient their abilities to be good managers.

6.2 What problems do you think might arise in extreme programming teams where many management decisions are devolved to the team members?

While the notion of devolving management decisions to the team is attractive in terms of motivation, there are two types of problem that can arise:

1. Decisions are liable to be primarily influenced by technical considerations rather than business decisions. This is natural given the type of people on an XP team – it is difficult for them to take a business perspective.

2. Because of the focus on rapid iteration, management decisions tend to be short-term and pay insufficient attention to long-term issues. While this is in keeping with the XP philosophy, there is sometimes a need for a more detached, longer-term perspective which can be taken by a manager.

I assume here that management decisions on e.g. the performance of team members are not taken by the team. Given the close knit nature of XP teams, it is difficult for the team to take decisions that censure individual team members.

7. PROJECT PLANNING

7.1 Explain why the process of project planning is iterative and why a plan must be continually reviewed during a software project.

Project planning can only be based on available information. At the beginning of a project, there are many uncertainties in the available information and some information about the project and the product may not be available. As the project develops, more and more information becomes available and uncertainties are resolved. The project plan therefore must be reviewed and updated regularly to reflect this changing information environment.

8. QUALITY MANAGEMENT

8.1 Explain how standards may be used to capture organizational wisdom about effective methods of software development. Suggest four types of knowledge that might be captured in organizational standards.

Standards encapsulate organizational wisdom because they capture good practice that have evolved over the years. Knowledge that might be captured in organizational standards include:

1. Knowledge of specific types of fault that commonly occur in the type of software developed by an organization. This might be encapsulated in a standard review checklist.

2. Knowledge of the types of system model that have proved useful for software maintenance. This can be encapsulated in design documentation standards.

3. Knowledge of tool support that has been useful for a range of projects. This can be encapsulated in a standard for a development environment to be used by all projects.

4. Knowledge of the type of information that is useful to include as comments in code. This can be encapsulated in a code commenting standard.

8.2 Explain why program inspections are an effective technique for discovering errors in a program. What types of error are unlikely to be discovered through inspections?

Program inspections are effective for the following reasons:

1. They can find several faults in one pass without being concerned about interference between program faults.

2. They bring a number of people with different experience of different types of errors. Hence, the team approach offers greater coverage than any individual can bring.

3. They force the program author to re-examine the program in detail - this often reveals errors or misunderstandings.

The types of errors that inspections are unlikely to find are specification errors or errors that are based on a misunderstanding of the application domain (unless there are domain experts in the team).

9. CONFIGURATION MANAGEMENT

9.1 Describe 6 essential features that should be included in a tool to support change management processes.

The idea here is that the student should look at the change management process and propose tool features to support system activities. Examples include:

1. Change request form definition.

2. Change request validation.

3. Work-flow definition – showing the flow of the change request form.

4. Change costing support.

5. Change request trace-ability – ie maintaining information about the changes made to the software to implement the change request.

6. Change notification and reminders – notification to people involved of work done and reminders to team members who have work to do.