CSCD306-DCIT208: SOFTWARE ENGINEERING

Mark Atta Mensah | Department of Computer Science | 2020-2021

SESSION 1

Introduction to Software Development in Practice | Software Engineering | Software Process

Reference: Sommerville, Software Engineering, 10 ed., Chapters 1 & 2

AIM OF THE COURSE

- We assume you know a lot about computers, can program reasonably, and can learn more on the job.
  - ✓*But there's more to software development than just writing good code.*
  - ✓*After UG, you may work on production projects worth millions of dollars.*
- *You may soon be boss. May be your cash.*
  - ✓*We want you to make mistakes and learn from them now.*

PERSONAL EXPERIENCE

- **Career:** Teaching, IT Manager, General Manger/CTO, Researcher
- **Projects Managed:**
  - Agency Banking for financial Institution
  - Assembly Revenue Collection application

What is Software Engineering?

“In theory, there is no difference between theory and practice. But, in practice, there is.”

Why do we need software engineering?

- Software and software systems are everywhere.
- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled.
- Software engineering is concerned with *theories*, *methods* and *tools* for professional software development.
- Expenditure on software represents a significant fraction of GNP in all developed countries

Why do we need software engineering?

Software comes in *many forms*;

- **Applications:** web sites, smartphones
- **System software:** operating systems, compilers
- **Communications:** routers, telephone switches
- **Data processing:** telephone billing, pensions
- **Real time:** air traffic control, autonomous vehicles
- **Embedded software:** device drivers, controllers
- **Mobile devices:** digital camera, GPS, sensors
- **Information systems:** databases, digital libraries
- **Offices:** word processing, video conferences
- **Scientific:** simulations, weather forecasting
- **Graphical:** film making, design

Why do we need software engineering?

- **Client** requirements are very different
- There is no standard process for software engineering
- There is no best language, operating system, platform, database system, development environment, etc.
- A skilled software developer knows about a wide variety of approaches, methods, tools.
- The craft of software development is to select appropriate methods for each project and apply them effectively

Why do we need software engineering?

Applying software engineering principles helps address the two main factors of software failures:

- **Increasing demands:** As new software engineering techniques help us to build larger, more complex systems, the demands change. 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. Computer programming is NOT software engineering.

Clients, Customers, and Users

- **Client –**
  - ✓The person (or group of people) for whom the software development team creates the software.
  - ✓The client provides resources, such as money, and expects some product in return.
  - ✓The client's job success may depend on the success of the software project.

Client satisfaction is a primary measure of success in a so(ware project.

Who is the client for a general-purpose package (e.g., Microsoft Excel)?

Clients, Customers, and Users

- **Customer –**
  - ✓The customer is the person who buys the so(ware or selects it for use by an organisation.
- **User –**
  - ✓A person who actually uses the software
- With *personal software*, the user may be the same person as the customer.
- In organisations, the customers and the users are usually different.

Many people in UG admin offices use Microsoft Excel? Who is the customer? Who are the users?

Risks

- Many (probably most) software development projects run into difficulties.
- **Problems:**
  - Does not work as expected *(FUNCTION)*
  - Over budget *(COST)*
  - Late delivery *(TIME)*
- **Competing goals**
  - Every software project has a trade-off between ***function***, ***cost***, and ***time***.
  - Extra function adds extra costs for development, testing, maintenance, etc.

What is important to the person who is paying?

Risks

- Failures of software development projects can bankrupt businesses.
- Failures in software development projects frequently result in the dismissal of senior executives. **Example**: *Apple's mapping app*.
- A lot of software is thrown away (perhaps 50 percent is never used).
- Many software projects fail because the software developers build the wrong software.

Risks

- The software development team must:
  - Learn what the client expects from the software.
  - Recognize what the client's organization expects from the client.
  - Recognize what customers and users expect from the software.
- The software development team will frequently provide technical insights and suggestions, but ***keep in mind that client satisfaction is the primary measure of success in a software project.***

Minimising Risk: Communicate with the Client

- **Feasibility studies** (whether to begin a project).
- **Separation** of **requirements** (what the client wants) from **design** (how the developers meet the requirements).
- **Milestones** (how the developers report or demonstrate progress to the clients) and **releases**.
- **Acceptance** (how the client tests that the so(ware meets the requirements) and **user testing.**
- **Handover** (ensuring that the client receives a package that can be used and maintained for an extended period of time).

Minimising Risk: Visibility

Visibility

The people who take the responsibility must know what is happening.

The problem (as seen by a manager)

- Must rely on others for reports of progress or difficulties.
- ... but software developers
  - Have difficulty evaluating progress.
  - Are usually optimistic about progress.
  - Consider reporting a waste of time.
  - etc.
- You will make regular progress reports on your projects.
- *Working software provides excellent visibility.*

Minimising Risk: Short Development Cycles

Risk is minimized by frequent delivery of working software (weeks rather than months).

- Client, customers, and users can evaluate the developers' work.
- Opportunities to adapt to changing circumstances.

This is one of the basic principles of agile software development.

FAQ about software engineering

- **What is software?**
  - Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
- **What are the attributes of good software?**
  - Good software should deliver the required functionality and performance to the user and should be maintainable, dependable, and usable.
- **What is software engineering?**
  - Software engineering is an engineering discipline that is concerned with all aspects of software production.
- **What are the fundamental software engineering activities?**
  - Software specification, software development, software validation and software evolution.

FAQ about software engineering

- **What is the difference between software engineering and computer science?**
  - Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
- **What is the difference between software engineering and system engineering?**
  - System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.
- **What are the key challenges facing software engineering?**
  - Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.

FAQ about software engineering

- **What are the costs of software engineering?**
  - Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
- **What are the best software engineering techniques and methods?**
  - While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.

FAQ about software engineering

- **What differences has the web made to software engineering?**
  - The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

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. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
- **Dependability and security**
  - Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
- **Efficiency**
  - Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.

Essential attributes of good software

- **Acceptability**
  - Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

Software engineering: a definition

- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
- It is an ***engineering discipline*** because it uses appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
- It focuses on ***all aspects of software production*** and not just on the technical process of development; it includes project management and the development of tools, methods etc. to support software production.
- 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. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

Software engineering: a definition

Any software process includes four types of activities:

- 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.

General issues affecting most software

- **Heterogeneity**

  Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.

- **Business and social change**

  Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.

- **Security and trust**

  As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

Software engineering fundamentals

These software engineering fundamentals that apply to all types of software system:

- **Software process**
  - Systems should be developed using a managed and understood development process. The organization developing the software should plan the development process and have clear ideas of what will be produced and when it will be completed. Of course, different processes are used for different types of software.
- **Focus on reliability**
  - Dependability and performance are important for all types of systems. Software should behave as expected, without failures and should be available for use when it is required. It should be safe in its operation and, as far as possible, should be secure against external attack. The system should perform efficiently and should not waste resources.

Software engineering fundamentals

These software engineering fundamentals that apply to all types of software system:

- **Importance of requirements**
  - Understanding and managing the software specification and requirements (what the software should do) are important. You have to know what different customers and users of the system expect from it and you have to manage their expectations so that a useful system can be delivered within budget and to schedule.
- **Leverage software reuse**
  - You should make as effective use as possible of existing resources. This means that, where appropriate, you should reuse software that has already been developed rather than write new software.

Software engineering ethics

Some issues of professional responsibility:

- **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. You should be careful to ensure that the intellectual property of employers and clients is protected.

Software engineering ethics

- **Computer misuse**

  You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious

Software Processes: The Big Picture

A software process is a structured set of activities required to develop a software system. Note that we are talking about a "**software process**" -- not a "**software *development* process**."

There are many different kinds of software processes, but each and every one of them involve these four types of fundamental activities:

- Software **specification** - defining what the system should do;
- Software **design and implementation** - defining the organization of the system and implementing the system;
- Software **validation** - checking that it does what the customer wants;
- Software **evolution** - changing the system in response to changing customer needs.

Software Processes: The Big Picture

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective. When we describe and discuss software processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities. Process descriptions may also include:

- **Products (what),** which are the outcomes of a process activity;
- **Roles (who),** which reflect the responsibilities of the people involved in the process;
- Pre- and post-**conditions (how)**, which are statements that are true before and after a process activity has been enacted or a product produced.

Software Processes: The Big Picture

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. In practice, most practical processes include elements of both plan-driven and agile approaches.

Software process models

- **The waterfall model**

  Plan-driven model. Separate and distinct phases of specification, software design, implementation, testing, and maintenance.

- **Incremental development**

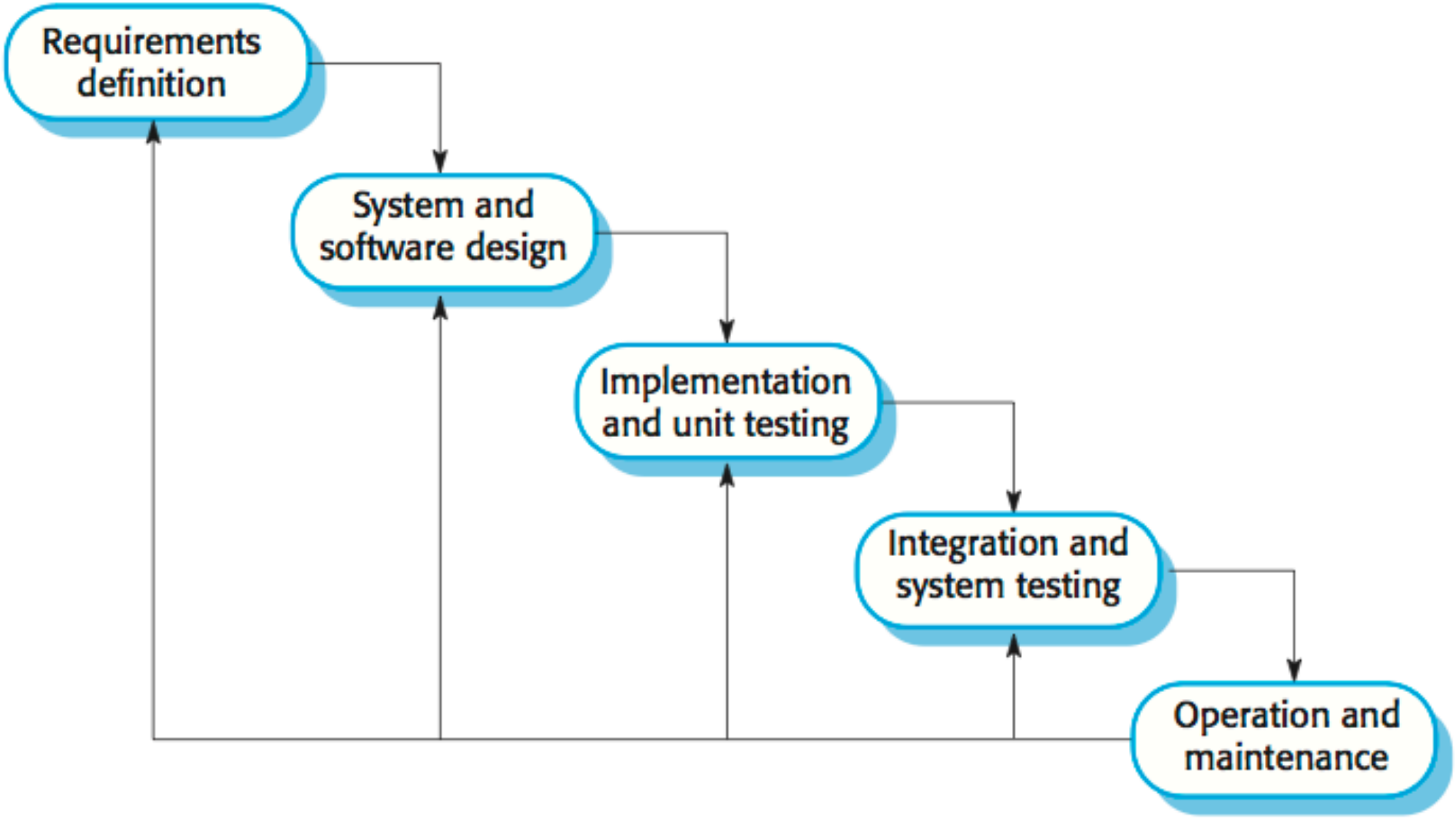
  Specification, development and validation are interleaved. The system is developed as a series of versions (increments), with each version adding functionality to the previous version. May be plan-driven or agile.

- **Integration and configuration**

  Based on the existence of a significant number of reusable components/systems. The system development process focuses on integrating these components into a system rather than developing them from scratch. May be plan-driven or agile.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall Model



The Waterfall Model

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. Software design involves identifying and describing the fundamental software system abstractions and their relationships.
- **Implementation and unit testing**
  - During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

The Waterfall Model

There are separate identified phases in the waterfall model:

- **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. After testing, the software system is delivered to the customer.
- **Operation and maintenance**
  - Normally (although not necessarily), this is the longest life cycle phase. 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.

The Waterfall Model

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase. Waterfall model **problems** include:

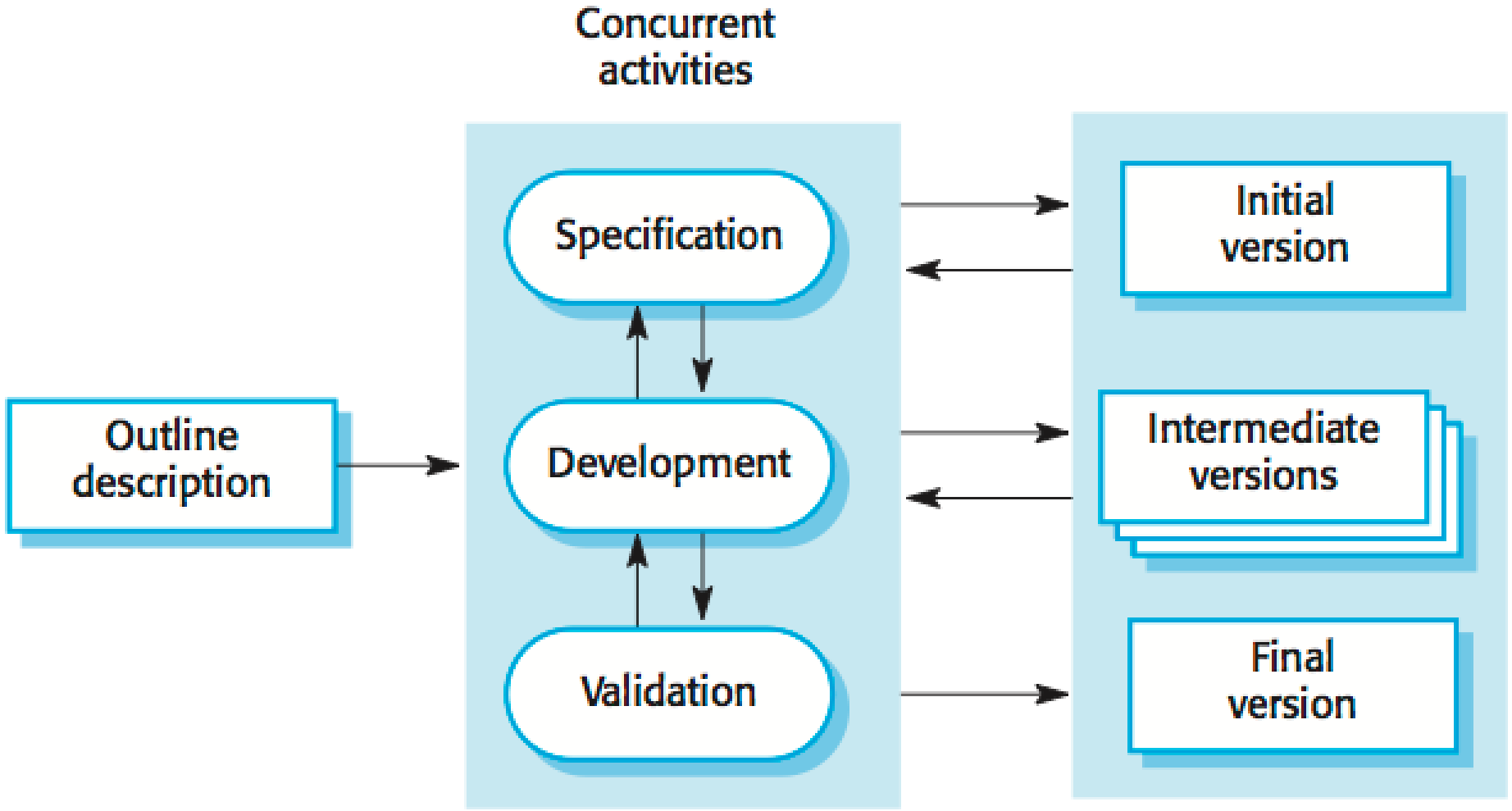
- **Difficult to address change**

  Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements. Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process. Few business systems have stable requirements.

- **Very few real-world applications**

  The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites. In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Incremental development model



Incremental development model: Benefits

- **Lower cost of changes**

  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.

- **Frequent feedback**

  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.

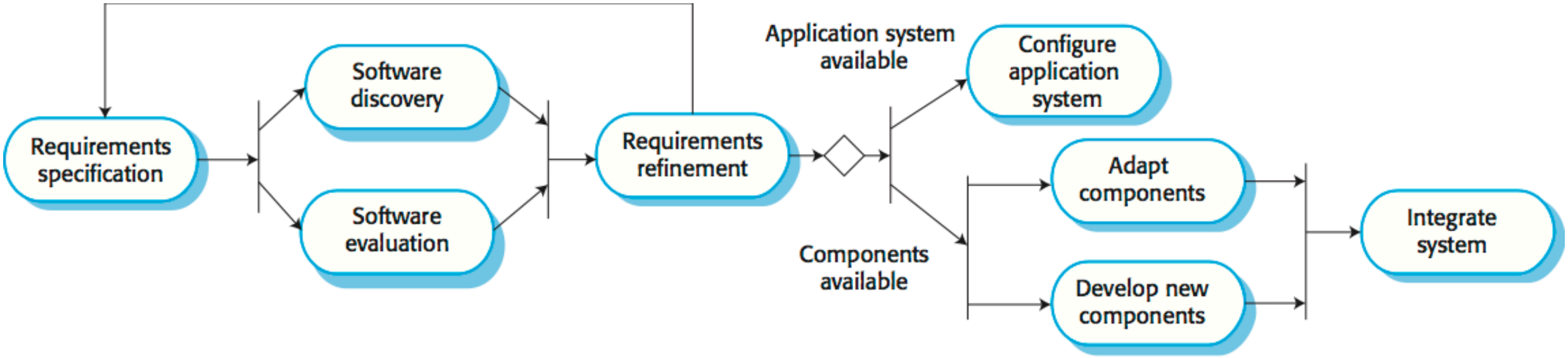
- **Faster delivery**

  More rapid delivery and deployment of useful software to the customer is possible. Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development model: Problems (Mgt)

- **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 refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

Integration and configuration



Integration and configuration

- This approach is based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems. Process stages include:
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.

Reuse is now the standard approach for building many types of business system.

Integration and configuration

Types of software components:

- **Web services** that are developed according to service standards and which are available for remote invocation.
- Collections of objects that are developed as a **package** to be integrated with a component framework such as .NET or J2EE.
- Stand-alone **commercial-off-the-shelf** systems (COTS) that are configured for use in a particular environment.

Software process activities

Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.

The four basic process activities of specification, development, validation and evolution are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

Software process activities

Software specification

The process of establishing what services are required and the constraints on the system's operation and development.

Requirements engineering process:

- **Feasibility study:** is it technically and financially feasible to build the system?
- Requirements **elicitation and analysis**: what do the system stakeholders require or expect from the system?
- Requirements **specification**: defining the requirements in detail
- Requirements **validation**: checking the validity of the requirements

Software process activities

Software design and implementation

The process of converting the system specification into an executable system.

- **Software design:** design a software structure that realizes the specification;
- **Implementation:** translate this structure into an executable program;

Software process activities

Software design and implementation

The activities of design and implementation are closely related and may be interleaved.

Design activities include:

- **Architectural design:** identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- **Interface design:** define the interfaces between system components.
- **Component design:** take each system component and design how it will operate.
- **Database design:** design the system data structures and how these are to be represented in a database.

Software process activities

Software validation

Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.

- **Validation:** are we building the right product (what the customer wants)?
- **Verification:** are we building the product, right?

V & V involves checking and review processes and system testing. System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Software process activities

Software validation

Testing is the most commonly used V & V activity and includes the following stages:

- **Development or component testing:** individual components are tested independently; components may be functions or objects or coherent groupings of these entities.
- **System testing:** testing of the system as a whole, testing of emergent properties is particularly important.
- **Acceptance testing:** testing with customer data to check that the system meets the customer's needs.

Software process activities

Software evolution

Software is inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

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. Change leads to rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality.

Two strategies to reduce the costs of rework:

- **Change avoidance**

  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.

Coping with change

- **Change tolerance**

  The process is designed so that changes can be accommodated at relatively low cost. This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

Software prototyping

A prototype is an initial version of a system used to demonstrate concepts and try out design options. 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 UI design;
- In the testing process to run back-to-back tests.

Software prototyping: Benefits

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

Software prototyping

Prototypes may be based on rapid prototyping languages or tools. They may involve **leaving out functionality**:

- Prototype should focus on areas of the product that are not well-understood;
- Error checking and recovery may not be included in the prototype;
- Focus on functional rather than non-functional requirements such as reliability and security.

Software prototyping

Prototypes should be **discarded** after development as they are not a good basis for a production system:

- 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 development/delivery

Rather than deliver 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. Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental development/delivery: Advantages

Advantages 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.

Incremental development/delivery: Problems

Incremental delivery problems:

- 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.
- The essence of iterative processes is that the specification is developed in conjunction with the software. However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

Process improvement

Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes. Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

Process maturity approach

Focuses on improving process and project management and introducing good software engineering practice. The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes.

Process improvement

Agile approach

Focuses on iterative development and the reduction of overheads in the software process. The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.

Process improvement activities form a continuous cycle with a feedback loop:

- **Measure** one or more attributes of the software process or product. These measurements forms a baseline that help decide if process improvements have been effective.
- **Analyze** the current process and identify any bottlenecks.
- **Change** the process to address some of the identified process weaknesses. These are introduced and the cycle resumes to collect data about the effectiveness of the changes.

Process improvement

Process measurement

- Wherever possible, quantitative process data should be collected.
- Process measurements should be used to assess process improvements.
- Metrics may include:
  - Time taken for process activities to be completed, e.g. calendar time or effort to complete an activity or process.
  - Resources required for processes or activities, e.g. total effort in person-days.
  - Number of occurrences of a particular event, e.g. number of defects discovered.

Process improvement

The SEI capability maturity model

- **Initial:** Essentially uncontrolled
- **Repeatable:** Product management procedures defined and used
- **Defined:** Process management procedures and strategies defined and used
- **Managed:** Quality management strategies defined and used
- **Optimizing:** Process improvement strategies defined and used

END OF SESSION