RV College of Engineering®

AI Integrated Software Engineering

UNIT - I

By
Prof. Narasimha Swamy S
Department of AIML
R V College of Engineering®, Bengaluru-59

Go, change the world®

Outline

→ Introduction

- > Professional Software Development
- > Software Engineering Ethics
- > Case Studies

→ Process

- > Models
- > Process activities
- > Coping with Change
- > Process improvement
- > The Rational Unified Process
- > Computer Aided Software Engineering

→ Agile Software Development

- > Introduction to agile methods
- > Agile development techniques
- > Agile project management and scaling agile methods.

Introduction

Introduction

- Software Engineering (SE) is essential for the functioning of government, society, and national and international businesses and institutions.
- National infrastructures and utilities are controlled by computer-based systems, and 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.
- More than 75% of the world's population have a software-controlled smart phone, and by 2016, almost all of these will be Internet-enabled.
- Software systems are abstract and intangible. They are not constrained by the properties of materials, nor are they governed by physical laws or by manufacturing processes.
- This simplifies software engineering, as there are no natural limits to the potential of software.
- However, because of the lack of physical constraints, software systems can quickly become extremely complex, difficult to understand, and expensive to change.
- There are many different types of software system, ranging from simple embedded systems to complex, worldwide information systems.

Introduction (Contd.)

- There are no universal notations, methods, or techniques for software engineering because different types of software require different approaches.
- Developing an organizational information system is completely different from developing a controller for a scientific instrument. Neither of these systems has much in common with a graphics-intensive computer game.
- All these applications need software engineering; they do not all need the same software engineering methods and techniques.
- Here are still many reports of software projects going wrong and of "software failures." Software engineering is criticized as inadequate for modern software development.
- However, in my opinion, many of these so-called software failures are a consequence of two factors:
  - Increasing system complexity
  - Failure to use software engineering methods

Introduction (Contd.)

- **Increasing system complexity**
  - As new software engineering techniques help us to build larger, more complex systems, the demands change. Systems must be built and delivered more quickly; larger, even more complex systems are required; and systems must have new capabilities that were previously thought to be impossible.
  - New software engineering techniques must be developed to meet new the challenges of delivering more complex software.
- **Failure to use software engineering methods**
  - It is easy to write computer programs without using software engineering methods and techniques. Many companies have drifted into software development as their products and services have evolved.
  - They do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be.

Introduction (Contd.)

History of Software Engineering

- The notion of software engineering was first proposed in 1968 (Naur and Randell 1969).
- Throughout the 1970s and 1980s, a variety of new software engineering techniques and methods were developed, such as structured programming, information hiding, and object-oriented development.
- Tools and standard notations were developed which are the basis of today's software engineering.

http://software-engineering-book.com/web/history/

Professional Software Development

Professional Software Development (PSD)

- Lots of people write programs.
    - People in business write spreadsheet programs to simplify their jobs;
    - Scientists and engineers write programs to process their experimental data;
    - Hobbyists write programs for their own interest and enjoyment.
- However, most software development is a professional activity in which software is developed for business purposes, for inclusion in other devices, or as software products such as information systems and computer-aided design systems.
- The key distinctions are that professional software is intended for use by someone apart from its developer and that teams rather than individuals usually develop the software. It is maintained and changed throughout its life.
- Software Engineering (SE) is intended to support professional software development rather than individual programming.
- It includes techniques that specification, design, and evolution, none of which are normally relevant for personal software development.

PSD (Contd.)

Professional Software Development

- Many people think that software is simply another word for computer programs.
- However, when we are talking about software engineering, software is not just the programs themselves but also all associated documentation, libraries, support websites, and configuration data that are needed to make these programs useful.
- A professionally developed software system is often more than a single program. A system may consist of several separate programs and configuration files that are used to set up these programs.
- It may include system documentation, which describes the structure of the system, user documentation, which explains how to use the system, and websites for users to download recent product information.
- Software engineers are concerned with developing software products, that is, software that can be sold to a customer.
- There are two kinds of software product: Generic products and Customized software.

PSD (Contd.)

Professional Software Development

- **Generic Products:** These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who can buy them. Examples of this type of product include apps for mobile devices, software for PCs such as databases, word processors, drawing packages, and project management tools. This kind of software also includes “vertical” applications designed for a specific market such as library information systems, accounting systems, or systems for maintaining dental records.
- **Customized Software:** These are systems that are commissioned by and developed for a particular customer. A software contractor designs and implements the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

PSD: Software Engineering

- 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.
- In this definition, there are two key phrases:

1. Engineering discipline: Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work within organizational and financial constraints, and they must look for solutions within these constraints.
2. All aspects of software production Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software development.

PSD: Software Engineering (Contd.)

Frequently asked questions about Software Engineering

Question	Answer
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 from initial conception to operation and maintenance.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
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.

PSD: Software Engineering (Contd.)

Frequently asked questions about Software Engineering

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.
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. There are no methods and techniques that are good for everything.
What differences has the Internet made to software engineering?	Not only has the Internet led to the development of massive, highly distributed, service-based systems, it has also supported the creation of an "app" industry for mobile devices which has changed the economics of software.

PSD: Software Engineering (Contd.)

- Essential *attributes of good software*

Product characteristic	Description
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.
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. Software has to be secure so that malicious users cannot 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, resource utilization, etc.
Maintainability	Software should be written in such a way 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.

PSD: Software Engineering (Contd.)

Software Engineering

- Software Engineering is important for two reasons:
  1. 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.
  2. It is usually cheaper, in the long run, to use software engineering methods and techniques for professional software systems rather than just write programs as a personal programming project. Failure to use software engineering method leads to higher costs for testing, quality assurance, and long-term maintenance.
- The systematic approach that is used in software engineering is sometimes called a software process. A software process is a sequence of activities that leads to the production of a software product.
- Four fundamental activities are common to all software processes.
  1. Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
  2. Software development, where the software is designed and programmed.

PSD: Software Engineering (Contd.)

Software Engineering

3. Software validation, where the software is checked to ensure that it is what the customer requires.
4. 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 must be completely specified before development begins. In e-commerce systems, the specification and the program are usually developed together.
- Software engineering is related to both computer science and systems engineering
  - 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. Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers. Computer science theory, however, is often most applicable to relatively small programs.

PSD: Software Engineering (Contd.)

Software 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. System engineers are involved in specifying the system, defining its overall architecture, and then integrating the different parts to create the finished system.

PSD: Software Engineering (Contd.)

- There are many different types of software. There are no universal software engineering methods or techniques that may be used.
- However, there are four related issues that affect many different types of software:

1. ***Heterogeneity*** Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices. As well as running on general-purpose computers, software may also have to execute on mobile phones and tablets. You often must 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.
2. ***Business and social change*** Businesses 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. Many traditional software engineering techniques are time consuming, and delivery of new systems often takes longer than planned. They need to evolve so that the time required for software to deliver value to its customers is reduced.

PSD: Software Engineering (Contd.)

3. ***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 successfully attack our software, and that information security is maintained.
4. ***Scale Software*** must be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.

- To address these challenges, we will need new tools and techniques as well as innovative ways of combining and using existing software engineering methods.

PSD: Software Engineering Diversity (Contd.)

- Software engineering is a systematic approach to the production of software that considers practical cost, schedule, and dependability issues, as well as the needs of software customers and producers.
- The specific methods, tools, and techniques used depend on the organization developing the software, the type of software, and the people involved in the development process.
- There are no universal software engineering methods that are suitable for all systems and all companies.
- Rather, a diverse set of software engineering methods and tools has evolved over the past 50 years.
- However, the Software Engineering Method and Theory (SEMAT) initiative proposed by Jacobson et al. 2013.
- Its main goal is to address the challenges and shortcomings in software engineering by establishing a solid theoretical and practical foundation that is universal, scalable, and adaptable across different projects, methodologies, and organizations.

PSD: Software Engineering Diversity (Contd.)

- There are ***many different types of application***, including:
  1. Stand-alone applications These are application systems that run on a personal computer or apps that run on a mobile device. They include all necessary functionality and may not need to be connected to a network. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, travel apps, productivity apps, and so on.
  2. Interactive transaction-based applications These are applications that execute on a remote computer and that are accessed by users from their own computers, phones, or tablets.
  3. Embedded control systems These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system. Examples of embedded systems include the software in a mobile (cell) phone, software that controls antilock braking in a car, and software in a microwave oven to control the cooking process.
  4. Batch processing systems These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs. Examples of batch systems are periodic billing systems, such as phone billing systems, and salary payment systems.

PSD: Software Engineering Diversity (Contd.)

5. Entertainment systems These are systems for personal use that are intended to entertain the user. Most of these systems are games of one kind or another, which may run on special-purpose console hardware. The quality of the user interaction offered is the most important distinguishing characteristic of entertainment systems.
6. Systems for modeling and simulation These are systems that are developed by scientists and engineers to model physical processes or situations, which include many separate, interacting objects. These are often computationally intensive and require high-performance parallel systems for execution.
7. Data collection and analysis systems Data collection systems are systems that collect data from their environment and send that data to other systems for processing. The software may have to interact with sensors and often is installed in a hostile environment such as inside an engine or in a remote location. "Big data" analysis may involve cloud-based systems carrying out statistical analysis and looking for relationships in the collected data.

PSD: Software Engineering Diversity (Contd.)

8. Systems of systems These are systems, used in enterprises and other large organizations, that are composed of several other software systems. Some of these may be generic software products, such as an ERP system. Other systems in the assembly may be specially written for that environment.

- Each type of system requires specialized software engineering techniques because the software has different characteristics.
- For example, an embedded control system in an automobile is safety-critical and is burned into ROM (read-only memory) when installed in the vehicle. It is therefore very expensive to change. Such a system needs extensive verification and validation so that the chances of having to recall cars after sale to fix software problems are minimized. User interaction is minimal (or perhaps nonexistent), so there is no need to use a development process that relies on user interface prototyping.
- For an interactive web-based system or app, iterative development and delivery is the best approach, with the system being composed of reusable components.

PSD: Software Engineering Diversity (Contd.)

- Nevertheless, there are software engineering fundamentals that apply to all types of software systems:
  1. They 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, the specific process that you should use depends on the type of software that you are developing.
  2. Dependability and performance are important for all types of system. 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.
  3. 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.

PSD: Software Engineering Diversity (Contd.)

4. Make effective use of existing resources. This means that, where appropriate, you should reuse software that has already been developed rather than write new software.

- These fundamental notions of process, dependability, requirements, management, and reuse are important themes of this book. Different methods reflect them in different ways, but they underlie all professional software development.
- These fundamentals are independent of the program language used for software development. For example, a dynamic language, such as Ruby, is the right type of language for interactive system development but is inappropriate for embedded systems engineering.

PSD: Software Engineering Ethics (Contd.)

- Like other engineering disciplines, software engineering is carried out within a social and legal framework. As a software engineer, you must accept that your job involves wider responsibilities than simply the application of technical skills.
- You must also behave in an ethical and morally responsible way if you are to be respected as a professional engineer.
  1. Confidentiality You should normally respect the confidentiality of your employers or clients regardless of whether a formal confidentiality agreement has been signed.
  2. Competence You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
  3. 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.
  4. 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) to extremely serious (dissemination of viruses or other malware).

Introduction (Contd.)

Case Studies

1. An embedded system This is a system where the software controls some hardware device and is embedded in that device. Issues in embedded systems typically include physical size, responsiveness, and power management, etc. The example of an embedded system that I use is a software system to control an insulin pump for people who have diabetes.
2. An information system The primary purpose of this type of system is to manage and provide access to a database of information. Issues in information systems include security, usability, privacy, and maintaining data integrity. The example of an information system used is a medical records system.
3. A sensor-based data collection system This is a system whose primary purposes are to collect data from a set of sensors and to process that data in some way. The key requirements of such systems are reliability, even in hostile environmental conditions, and maintainability. The example of a data collection system that I use is a wilderness weather station.
4. A support environment. This is an integrated collection of software tools that are used to support some kind of activity. Programming environments, such as Eclipse (Vogel 2012) will be the most familiar type of environment for readers of this book. I describe an example here of a digital learning environment that is used to support students' learning in schools.

Introduction (Contd.)

Case Study 1: An Insulin Pump Control System

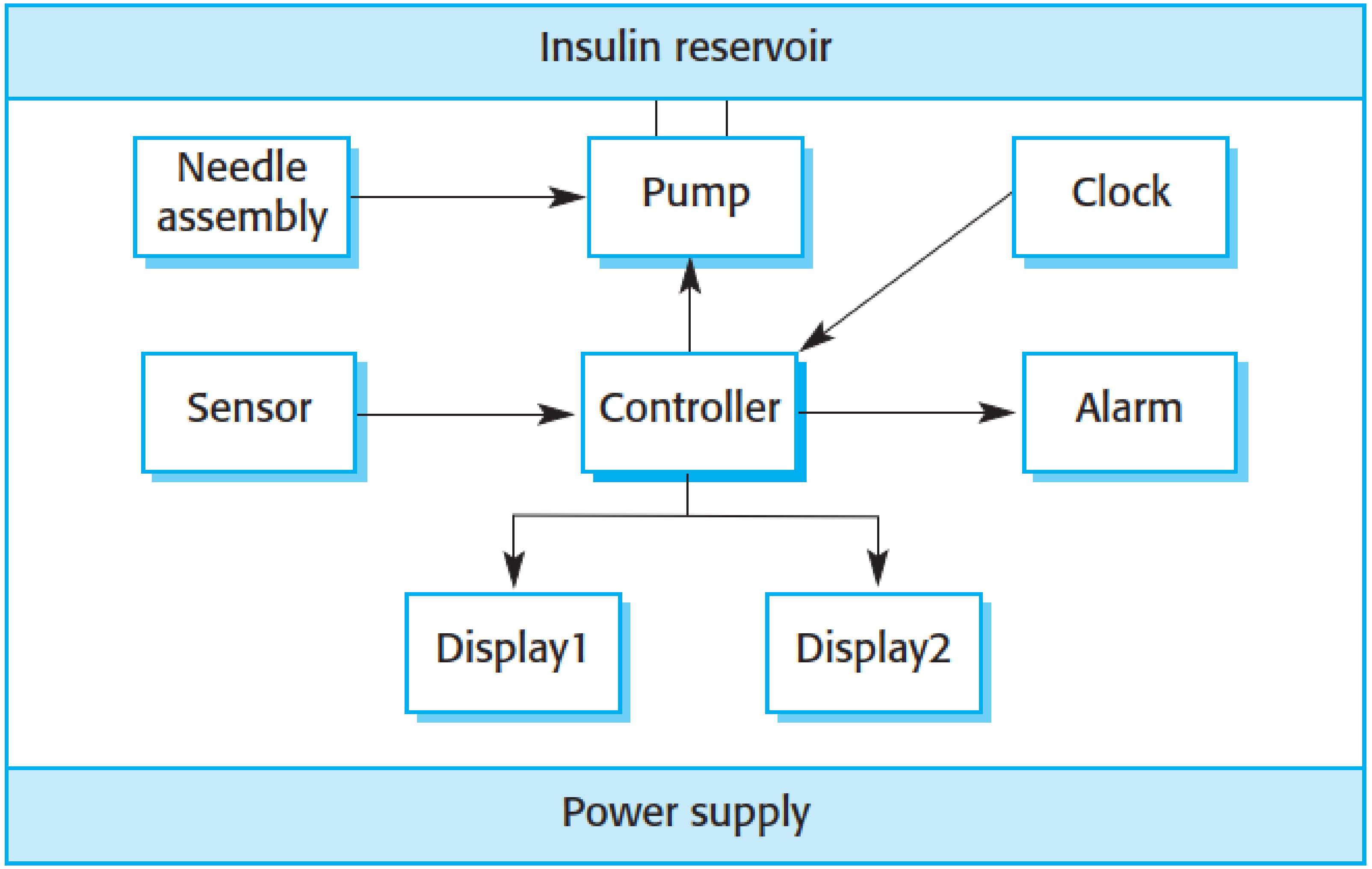


Figure. Insulin Pump Hardware Architecture

Introduction (Contd.)

Case Study 1: An Insulin Pump Control System

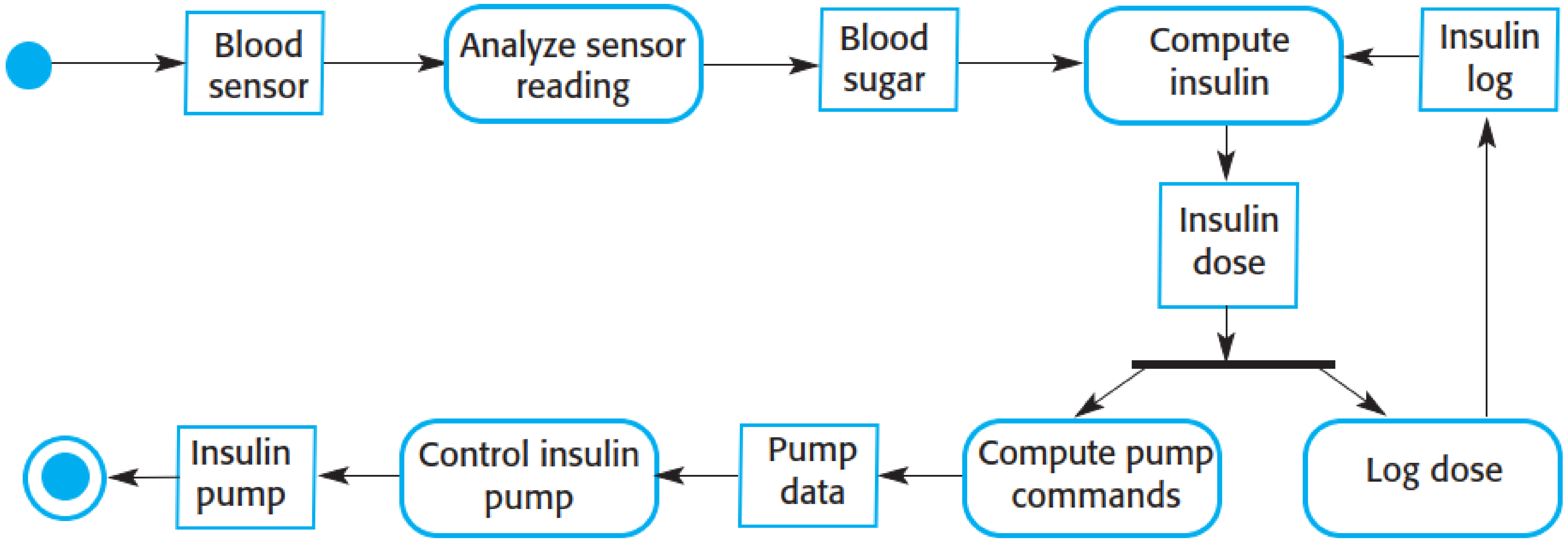


Figure. Activity Model of the Insulin Pump

Introduction (Contd.)

Case Study 2: A Patient Information System for Mental Health Care

- A patient information system to support mental health care (the Mentcare system) is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- This system has two purposes
  - To generate management information that allows health service managers to assess performance against local and government targets.
  - To provide medical staff with timely information to support the treatment of patients

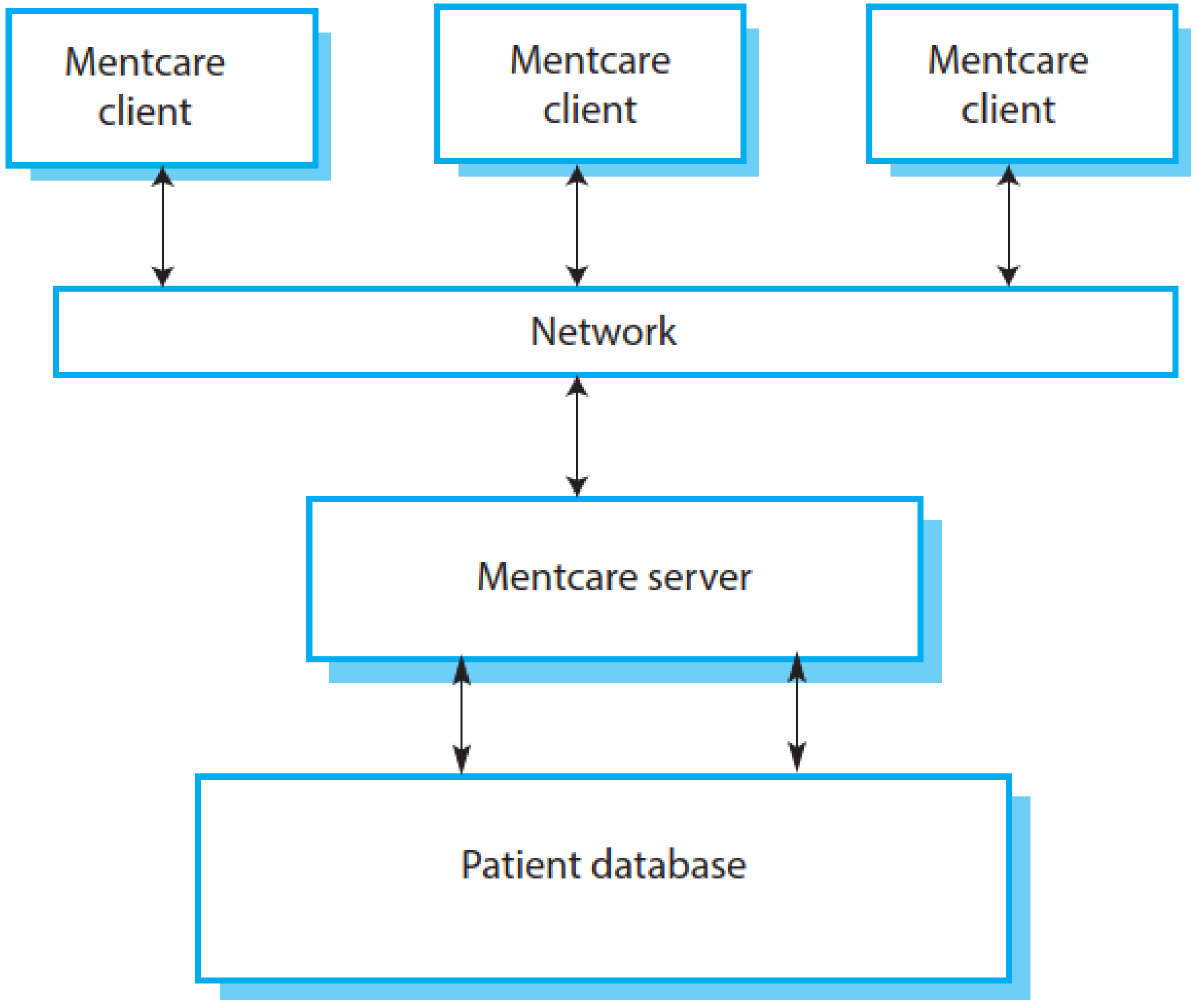


Figure. The Organization of the Mentcare system

Introduction (Contd.)

Case Study 3: A Wilderness Weather Station

- To help monitor climate change and to improve the accuracy of weather forecasts in remote areas, the government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.
- These weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.

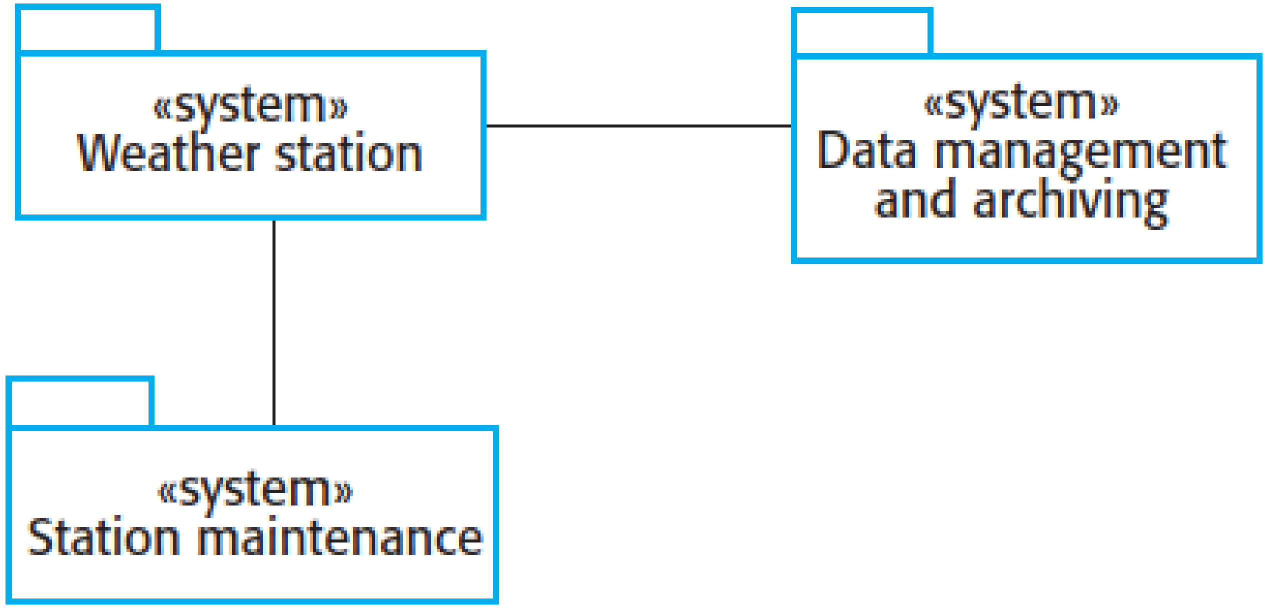

«system» Weather station
«system» Data management and archiving
«system» Station maintenance

Figure. The Weather Station's Environment

Introduction (Contd.)

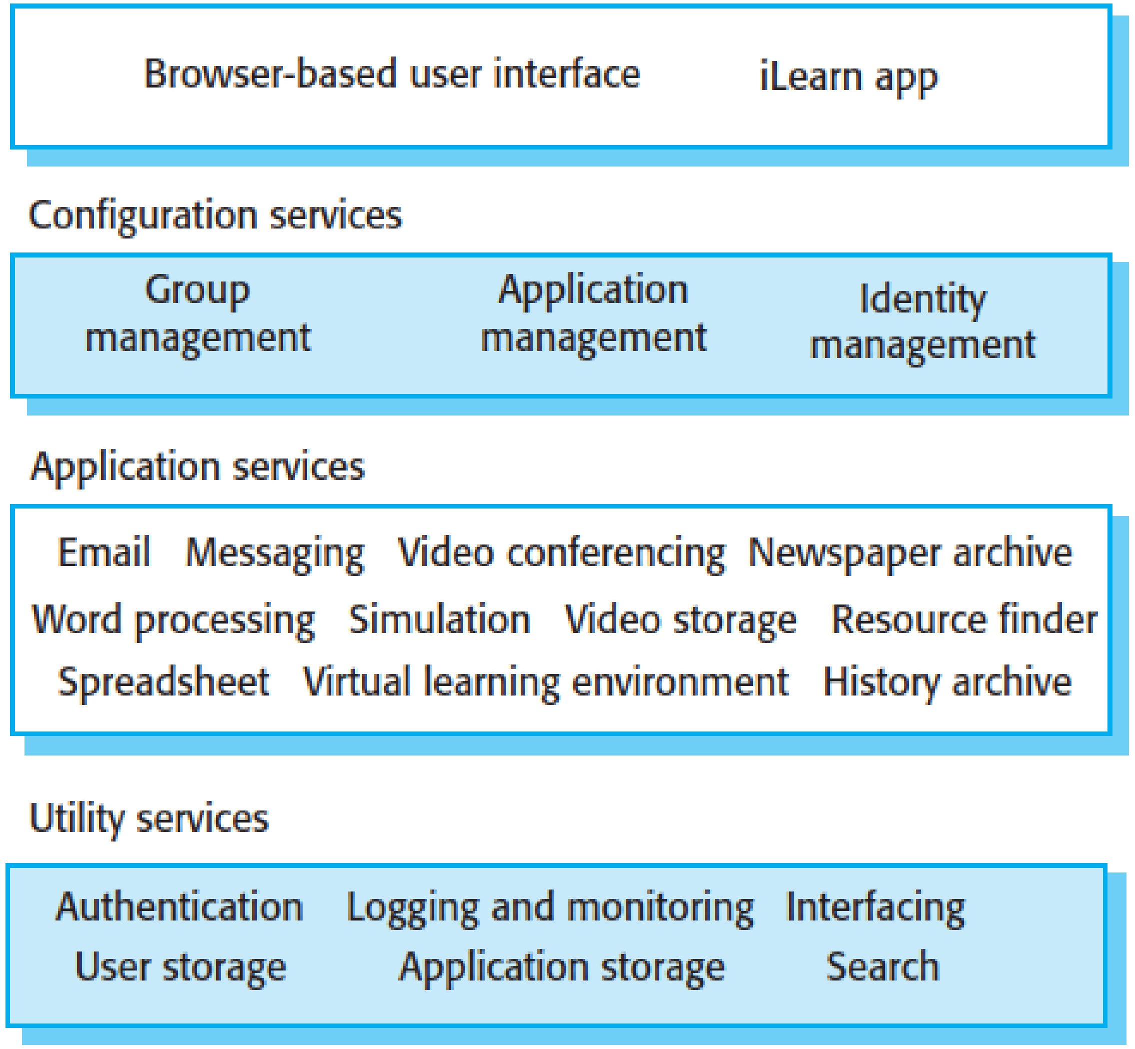
Case Study 3: A Wilderness Weather Station

- The weather station system
  - This system is responsible for collecting weather data, carrying out some initial data processing, and transmitting it to the data management system.
- The data management and archiving system
  - This system collects the data from all of the wilderness weather stations, carries out data processing and analysis, and archives the data in a form that can be retrieved by other systems, such as weather forecasting systems.
- The station maintenance system
  - This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.
  - It can update the embedded software in these systems. In the event of system problems, this system can also be used to remotely control the weather station.

Introduction (Contd.)

Case Study 4: A Digital Learning Environment for Schools

- Figure high-level architectural model of a digital learning environment (iLearn) that was designed for use in schools for students from 3 to 18 years of age.
- The approach adopted is that this is a distributed system in which all components of the environment are services that can be accessed from anywhere on the Internet.
- There are three types of service in the system
  - Utility Services
  - Application Services
  - Configuration Services



Introduction (Contd.)

Case Study 4: A Digital Learning Environment for Schools

- Utility services that provide basic application-independent functionality and that may be used by other services in the system. Utility services are usually developed or adapted specifically for this system.
- Application services that provide specific applications such as email, conferencing, photo sharing, etc., and access to specific educational content such as scientific films or historical resources. Application services are external services that are either specifically purchased for the system or are available freely over the Internet.
- Configuration services that are used to adapt the environment with a specific set of application services and to define how services are shared between students, teachers, and their parents.

Software Process

Software Process

- A software process is a set of related activities that leads to the production of a software system.
- There are many different types of software systems, and there is no universal software engineering method that is applicable to all of them.
- Consequently, there is no universally applicable software process. The process used in different companies depends on the type of software being developed, the requirements of the software customer, and the skills of the people writing the software.
- However, although there are many different software processes, they all must include, in some form, the four fundamental software engine
  1. Software specification The functionality of the software and constraints on its operation must be defined.
  2. Software development The software to meet the specification must be produced.
  3. Software validation The software must be validated to ensure that it does what the customer wants.
  4. Software evolution The software must evolve to meet changing customer needs

Software Process (Contd.)

- These activities are complex activities in themselves, and they include sub activities such as requirements validation, architectural design, and unit testing.
- Processes also include other activities, such as software configuration management and project planning that support production activities.
- When we describe and discuss processes, we usually talk about the activities in these processes, such as specifying a data model and designing a user interface, and the ordering of these activities.
- However, when describing processes, it is also important to describe who is involved, what is produced, and conditions that influence the sequence of activities:
    1. Products or deliverables are the outcomes of a process activity. For example, the outcome of the activity of architectural design may be a model of the software architecture.
    2. Roles reflect the responsibilities of the people involved in the process. Examples of roles are project manager, configuration manager, and programmer.
    3. Pre- and postconditions are conditions that must hold before and after a process activity has been enacted or a product produced. For example, before architectural design begins, a precondition may be that the consumer has approved all requirements; after this activity is finished, a postcondition might be that the UML models describing the architecture have been reviewed.

Software Process (Contd.)

Software Process Model

- A Software Process Model (sometimes called a Software Development Life Cycle or SDLC model) is a simplified representation of a software process.
- Each process model represents a process from a particular perspective and thus only provides partial information about that process. For example, a process activity model shows the activities and their sequence but may not show the roles of the people involved in these activities.
- The waterfall model

  Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development

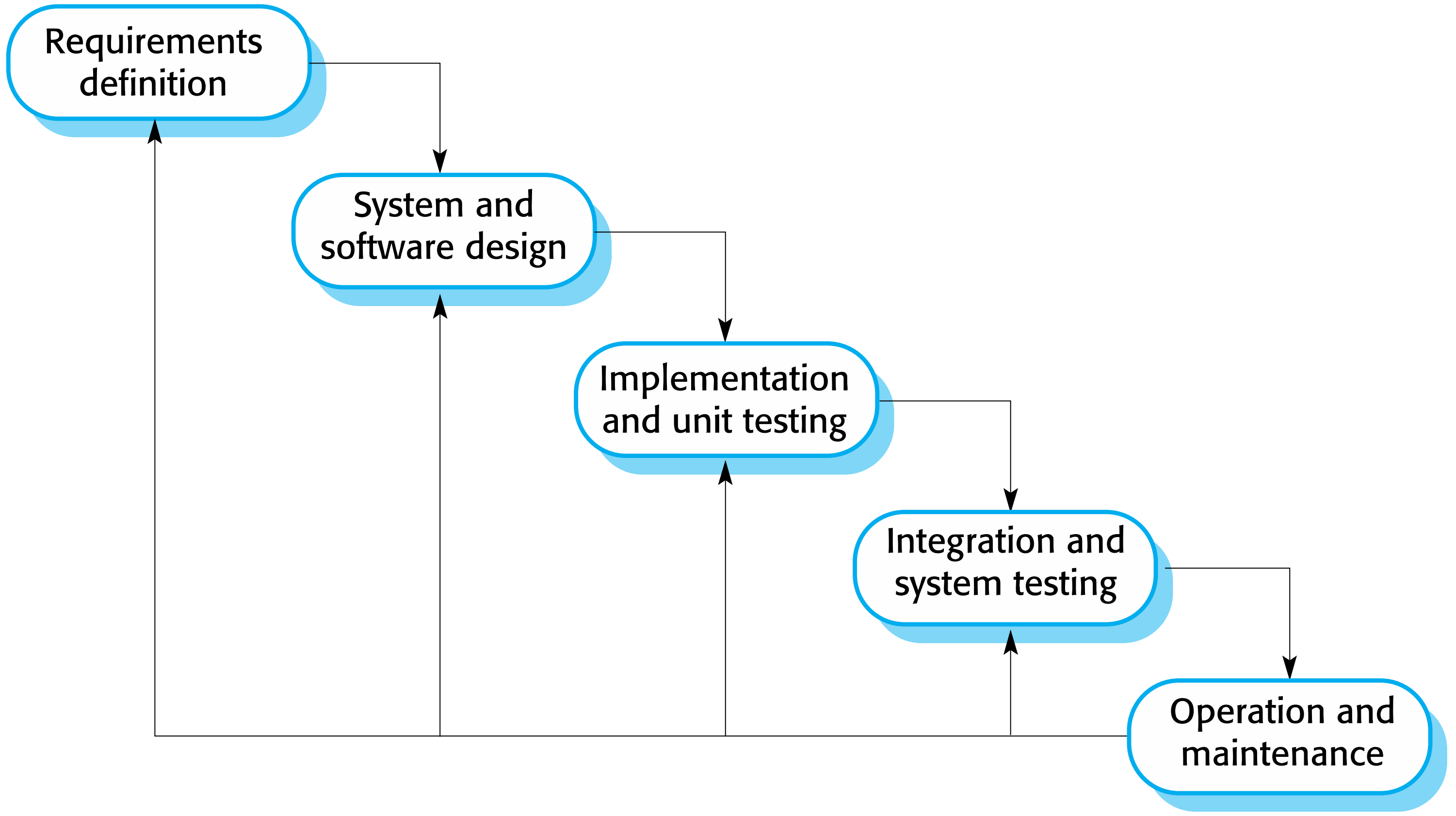
  Specification, development and validation are interleaved. May be plan-driven or agile.
- Integration and configuration

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

Software Process (Contd.)

Software Process Model: Waterfall Model

- The first published model of the software development process was derived from engineering process models used in large military systems engineering (Royce 1970).
- It presents the software development process as a few stages, as shown


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Software Process (Contd.)

Software Process Model: Wate fall Model

- The stages of the waterfall model directly reflect the fundamental software development activities:
    1. 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.
    2. System and software design The systems design process allocates the requirements to either hardware or software systems. It establishes an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.
    3. 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.
    4. 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.

Software Process (Contd.)

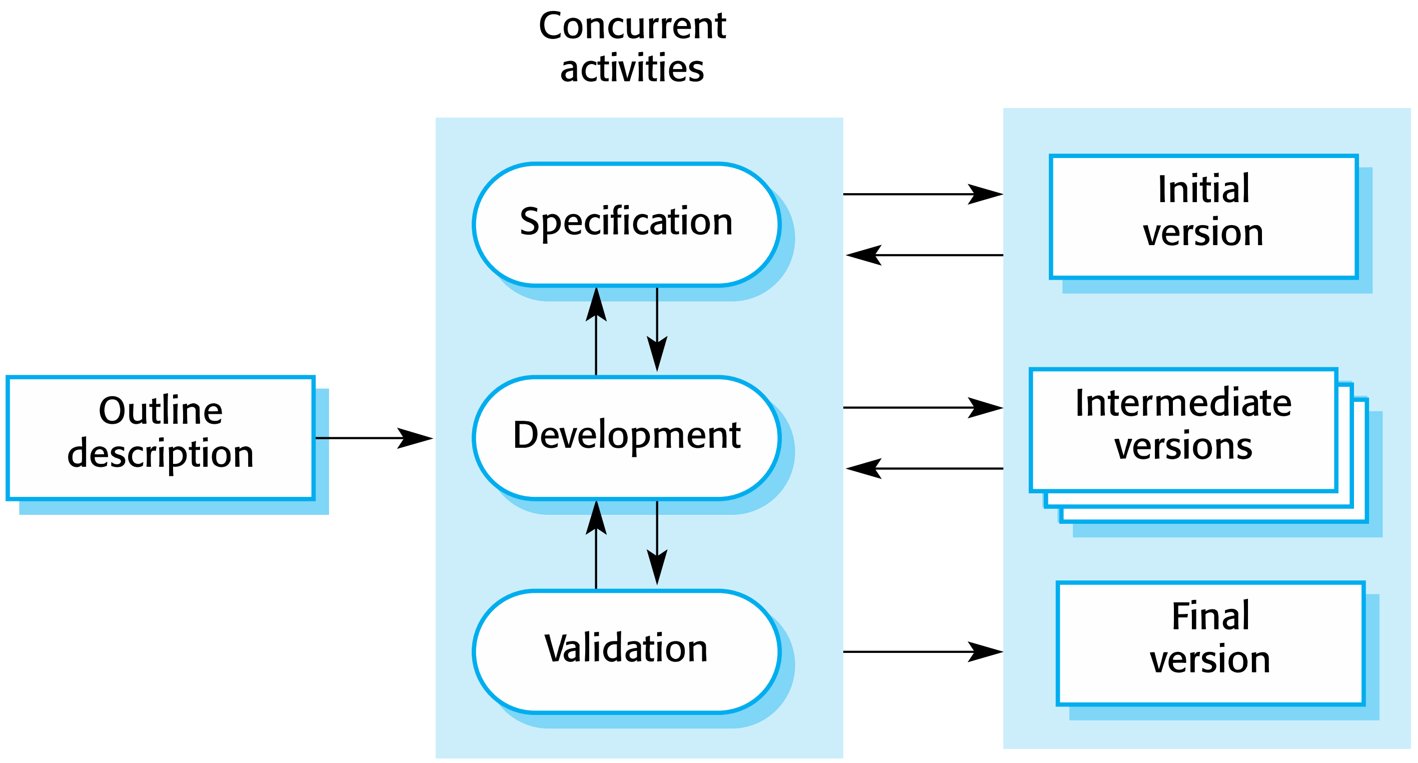
Software Process Model: Waterfall Model

- Operation and maintenance Normally, this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors that 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 main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase must 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.

Software Process (Contd.)

Software Process Model: Incremental Development Model

- Incremental development in some form is now the most common approach for the development of application systems and software products.
- This approach can be either plan-driven, agile or, both, a mixture of these approaches.
- In a plan-driven approach, the system increments are identified in advance; if an agile approach is adopted, the early increments are identified, but the development of later increments depends on progress and customer priorities



Software Process (Contd.)

Software Process Model: Incremental Development Model

- Incremental development has three major advantages over the waterfall model:
  1. The cost of implementing requirements changes is reduced. The amount of analysis and documentation that must be redone is significantly less than is required with the waterfall model.
  2. 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.
  3. Early delivery and deployment of useful software to the customer is possible, even if all the functionality has not been included. Customers can use and gain value from the software earlier than is possible with a waterfall process.

Software Process (Contd.)

Software Process Model: Incremental Development Model

- From a management perspective, the incremental approach has two problems:
  1. 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.
  2. The System structure tends to degrade as new increments are added. Regular change leads to messy code as new functionality is added in whatever way is possible.
  3. It becomes increasingly difficult and costly to add new features to a system. To reduce structural degradation and general code messiness, agile methods suggest that you should regularly refactor (improve and restructure) the software.

Software Process (Contd.)

Software Process Model: Integration and configuration (Software Re-use)

- In most software projects, there is some software reuse.
- This often happens informally when people working on the project know of or search for code that is like what is required. They look for these, modify them as needed, and integrate them with the new code that they have developed.
- Three types of software components are frequently reused
  - Stand-alone application systems that are configured for use in a particular environment. These systems are general-purpose systems that have many features, but they must be adapted for use in a specific application
  - Collections of objects that are developed as a component or as a package to be integrated with a component framework such as the Java Spring framework
  - Web services that are developed according to service standards and that are available for remote invocation over the Internet

Software Process (Contd.)

Software Process Model: Integration and Configuration

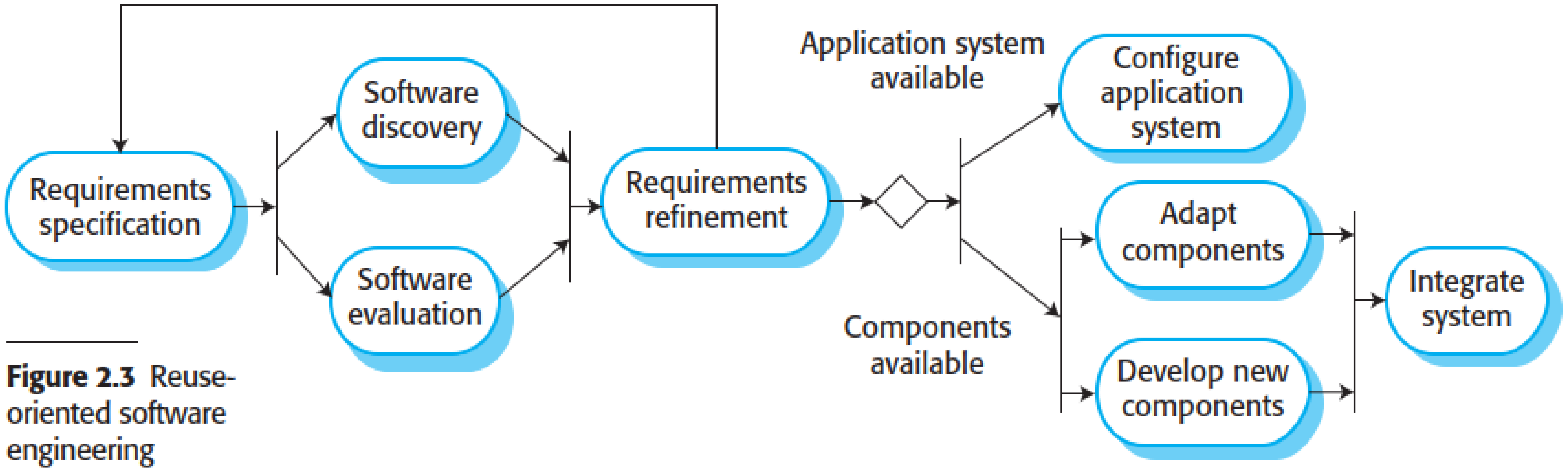


Figure 2.3 Reuse-oriented software engineering

Software Process (Contd.)

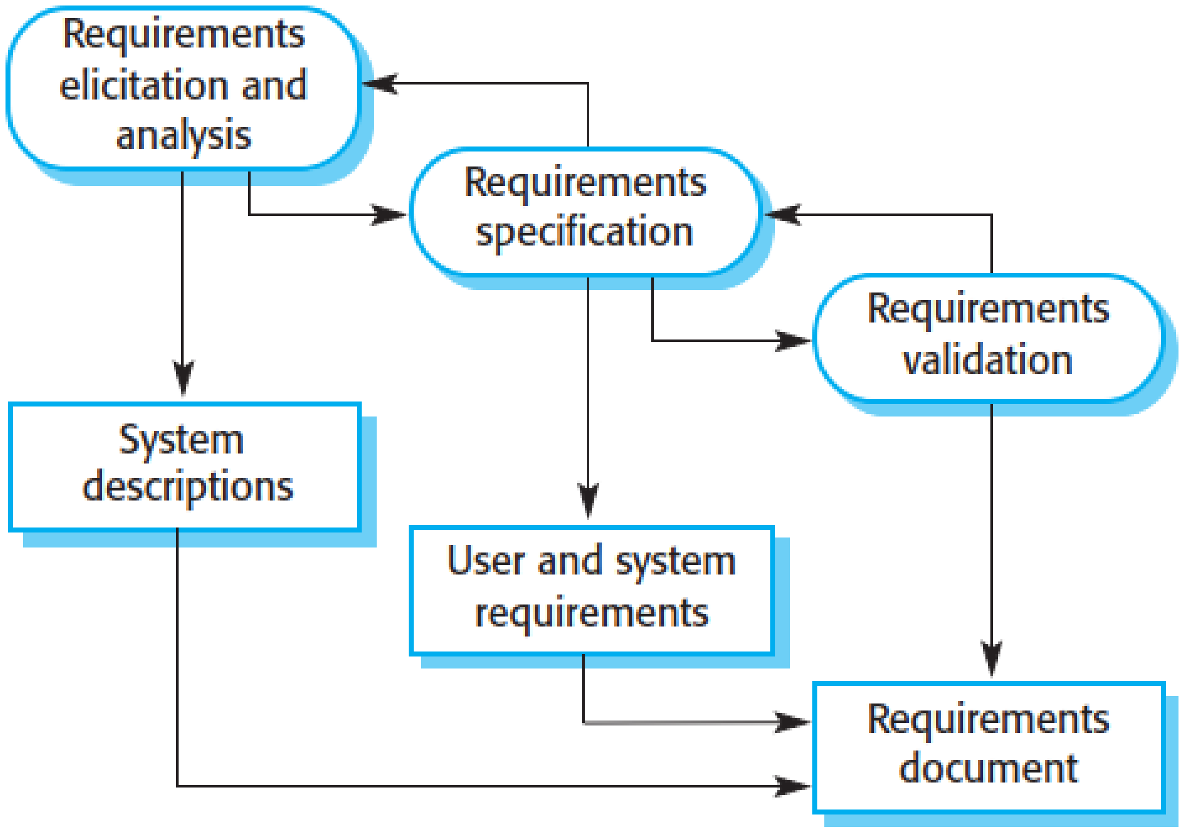
Software Process Model: Integration and Configuration

- Requirements specification The initial requirements for the system are proposed. These do not have to be elaborated in detail but should include brief descriptions of essential requirements and desirable system features.
- Software discovery and evaluation Given an outline of the software requirements, a search is made for components and systems that provide the functionality required. Candidate components and systems are evaluated to see if they meet the essential requirements and if they are generally suitable for use in the system
- Requirements refinement During this stage, the requirements are refined using information about the reusable components and applications that have been discovered. The requirements are modified to reflect the available components, and the system specification is re-defined. Where modifications are impossible, the component analysis activity may be reentered to search for alternative solutions.
- Application system configuration If an off-the-shelf application system that meets the requirements is available, it may then be configured for use to create the new system.
- Component adaptation and integration If there is no off-the-shelf system, individual reusable components may be modified and new components developed. These are then integrated to create the system.

Software Process : Process Activity

1. Software specification

- Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the systems operation and development.
- Requirements engineering is a particularly critical stage of the software process, as mistakes made at this stage inevitably lead to later problems in the system design and implementation



Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

The requirements engineering process

Software Process : Process Activity

1. Software specification

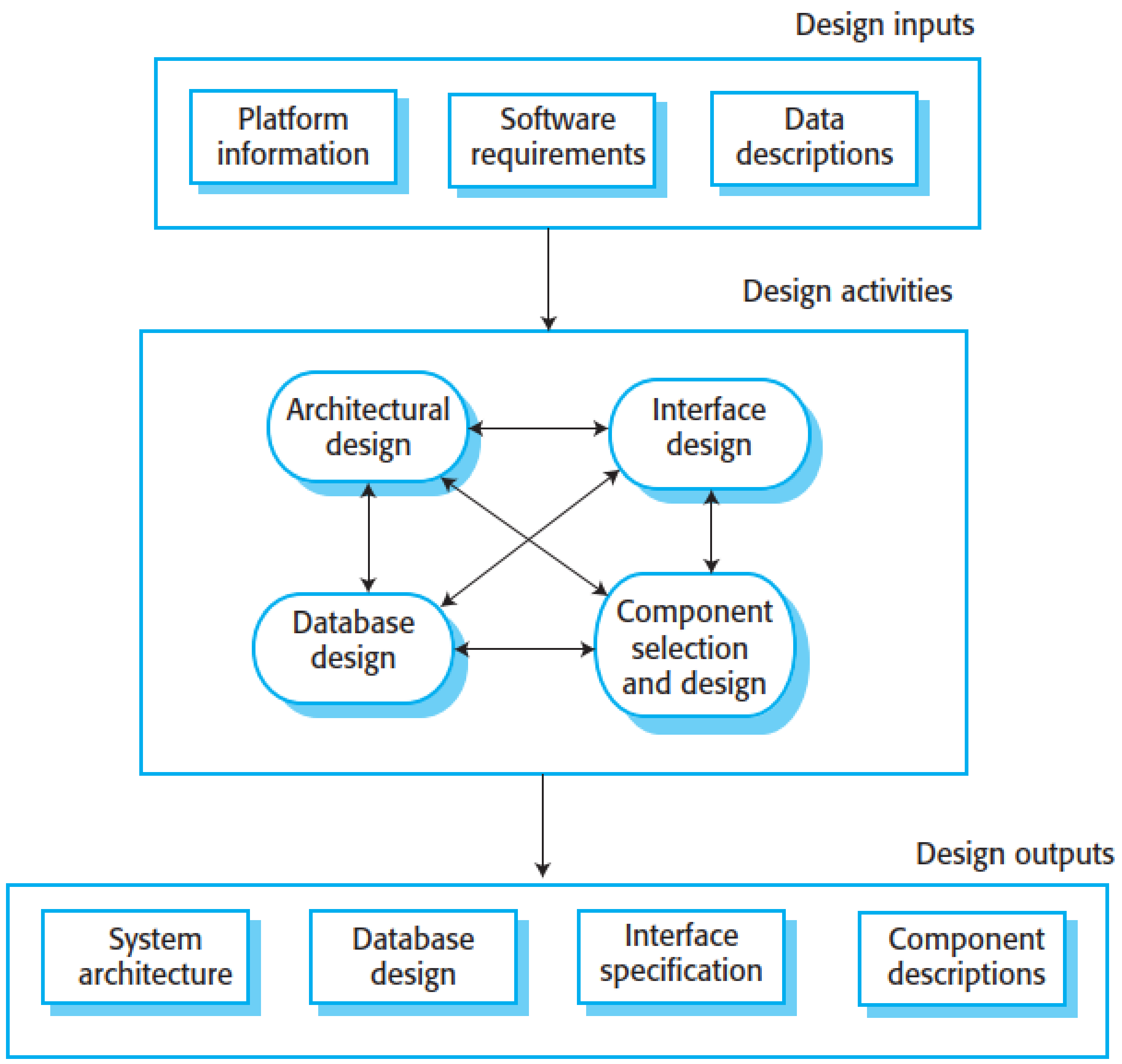
- There are three main activities in the requirements engineering process:

  1. Requirements elicitation and analysis This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.
  2. Requirements specification Requirements specification is the activity of translating the information gathered during requirements analysis into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; system requirements are a more detailed description of the functionality to be provided.
  3. Requirements validation This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

Software Process : Process Activity

2. Software design and implementation

- A software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used.
- Figure shows four activities that may be part of the design process for information systems
- Architectural design, where you identify the overall structure of the system, the principal components, their relationships, and how they are distributed.



Software Process : Process Activity

2. Software design and implementation

- Database design, where you design the system data structures and how these are to be represented in a database. Again, the work here depends on whether an existing database is to be reused, or a new database is to be created.
- Interface design, where you define the interfaces between system components. This interface specification must be unambiguous. With a precise interface, a component may be used by other components without them having to know how it is implemented. Once interface specifications are agreed, the components can be separately designed and developed.
- Component selection and design, where you search for reusable components and, if no suitable components are available, design new software components. The design at this stage may be a simple component description with the implementation details left to the programmer.

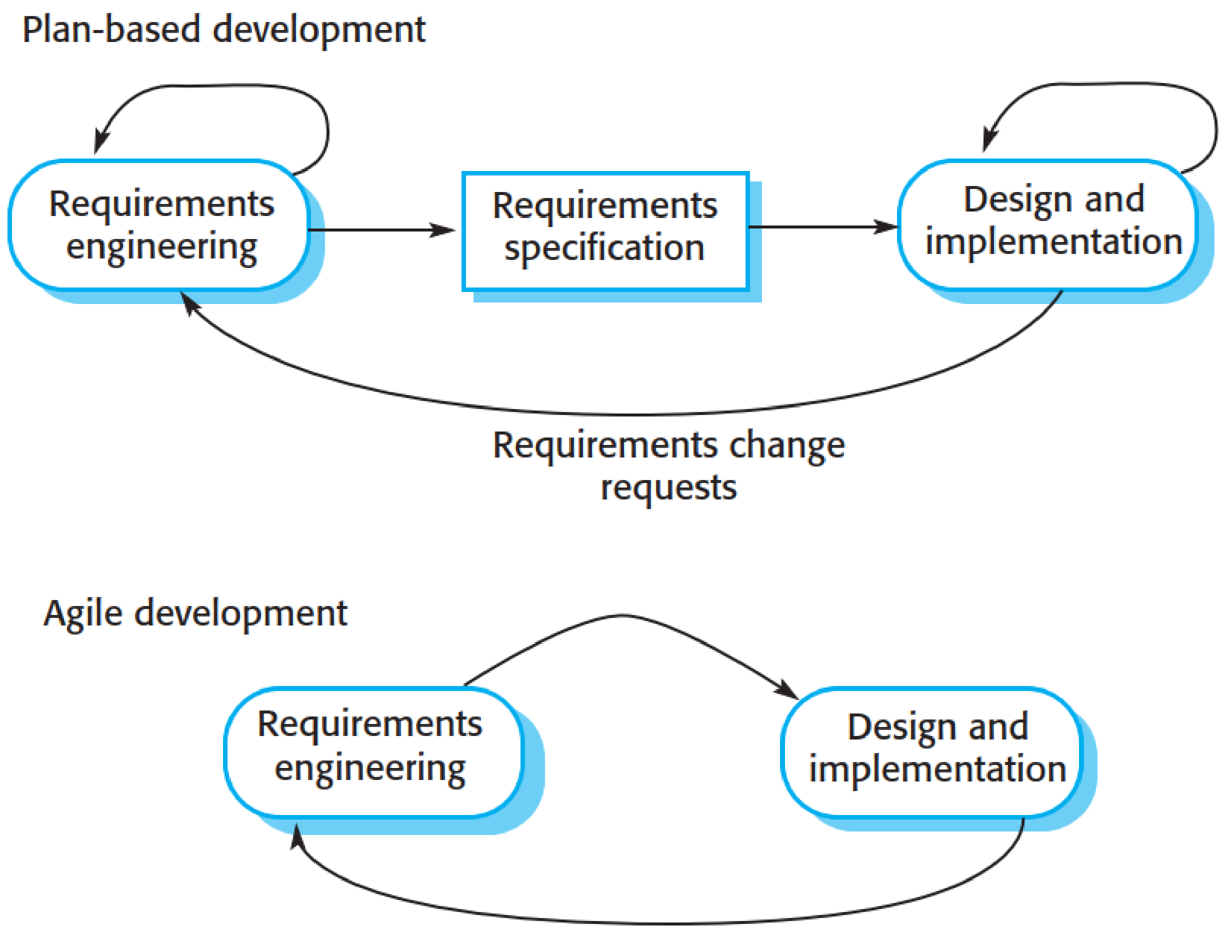
Agile Software Development (ASD)

Agile Software Development

- Businesses now operate in a global, rapidly changing environment.
- They have to respond to new opportunities and markets, changing economic conditions and the emergence of competing products and services.
- Software is part of almost all business operations, so new software must be developed quickly to take advantage of new opportunities and to respond to competitive pressure.
- Rapid software development and delivery is therefore the most critical requirement for most business systems.
- It is practically impossible to derive a complete set of stable software requirements.
  - Requirements change because customers find it impossible to predict how a system will affect working practices, how it will interact with other systems, and what user operations should be automated.
  - It may only be after a system has been delivered and users gain experience with it that the real requirements become clear. Even then, external factors drive requirements change.

Agile Software Development (Contd.)

- The need for rapid software development and processes that can handle changing requirements has been recognized for many years (Larman and Basili 2003).
- However, faster software development really took off in the late 1990s with the development of the idea of “agile methods” such as Extreme Programming (Beck 1999), Scrum (Schwaber and Beedle 2001), and DSDM (Stapleton 2003).
- Rapid software development became known as agile development or agile methods. These agile methods are designed to produce useful software quickly.


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

ASD: Agile Methods

- The need for rapid software development and processes that can handle changing requirements has been recognized for many years (Larman and Basili 2003).
- However, faster software development really took off in the late 1990s with the development of the idea of "agile methods" such as Extreme Programming (Beck 1999), Scrum (Schwaber and Beedle 2001), and DSDM (Stapleton 2003).
- Rapid software development became known as agile development or agile methods. These agile methods are designed to produce useful software quickly.
- All agile methods suggest that software should be developed and delivered incrementally. These methods are based on different agile processes, but they share a set of principles, based on the agile manifesto, and so they have much in common. I have listed these principles
- Agile methods have been particularly successful for two kinds of system development.
  1. Product development where a software company is developing a small or medium-sized product for sale. Virtually all software products and apps are now developed using an agile approach.

ASD: Agile Methods

2. Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external stakeholders and regulations that affect the software.

- Agile methods work well in these situations because it is possible to have continuous communications between the product manager or system customer and the development team. The software itself is a stand-alone system rather than tightly integrated with other systems being developed at the same time

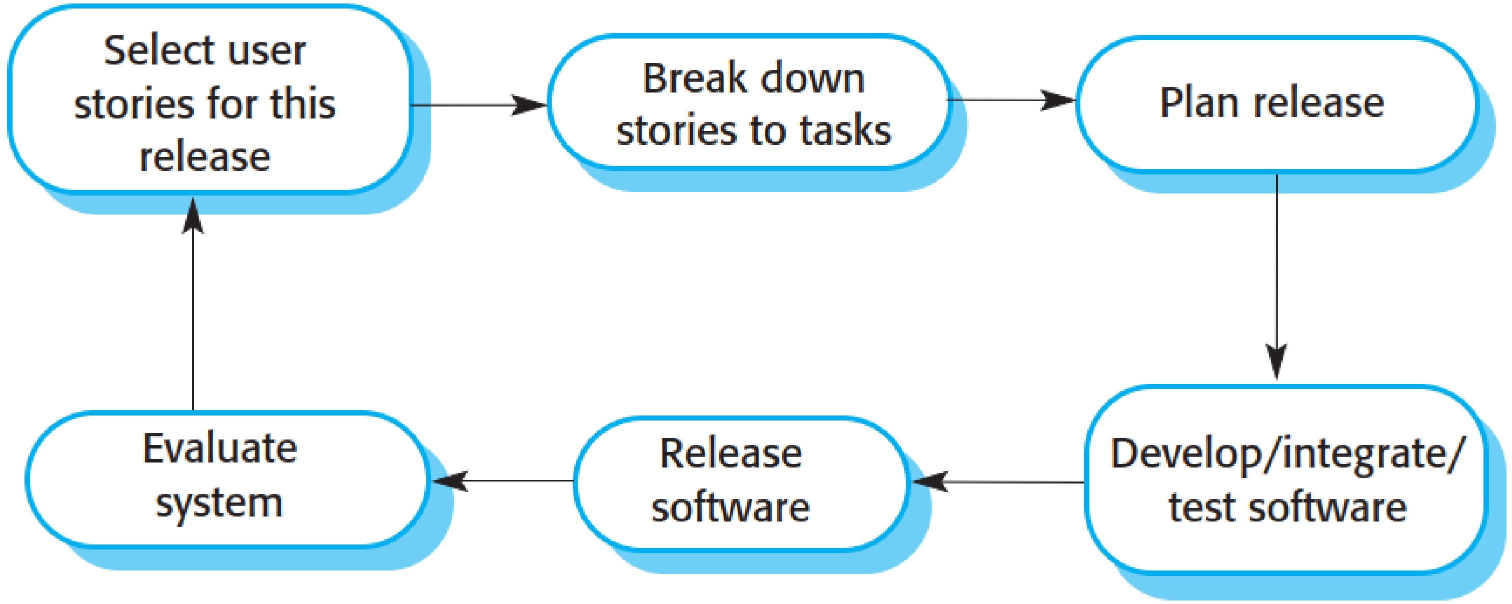
ASD: Agile Methods

The Principles of Agile Methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Embrace change	Expect the system requirements to change, and so design the system to accommodate these changes.
Incremental delivery	The software is developed in increments, with the customer specifying the requirements to be included in each increment.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.
People, not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.

ASD: Agile Development Techniques

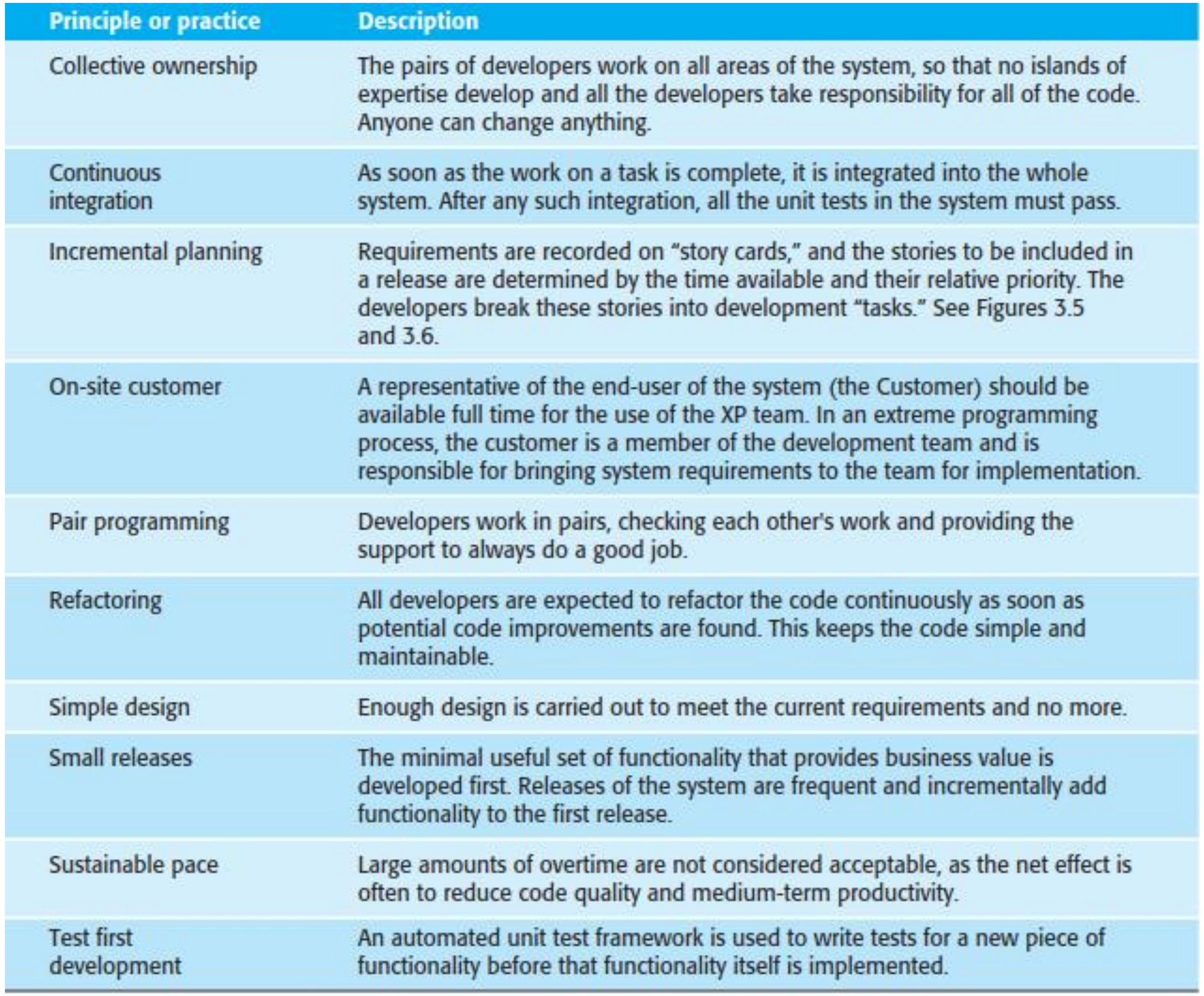
- The ideas underlying agile methods were developed around the same time by a number of different people in the 1990s.
- However, perhaps the most significant approach to changing software development culture was the development of Extreme Programming (XP). The name was coined by Kent Beck (Beck 1998) because the approach was developed by pushing recognized good practice, such as iterative development, to “extreme” levels. For example, in XP, several new versions of a system may be developed by different programmers, integrated, and tested in a day.
- Figure 3.3 illustrates the XP process to produce an increment of the system that is being developed.


Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

ASD: Agile Development Techniques (Contd.)

- In XP, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks. Programmers work in pairs and develop tests for each task before writing the code. All tests must be successfully executed when new code is integrated into the system. There is a short time gap between releases of the system.
- Extreme programming was controversial as it introduced a few agile practices that were quite different from the development practice of that time.
- These practices are summarized in Figure 3.4 and reflect the principles of the agile manifesto:
  1. Incremental development is supported through small, frequent releases of the system. Requirements are based on simple customer stories or scenarios that are used as a basis for deciding what functionality should be included in a system increment.
  2. Customer involvement is supported through the continuous engagement of the customer in the development team. The customer representative takes part in the development and is responsible for defining acceptance tests for the system.
  3. People, not process, are supported through pair programming, collective ownership of the system code, and a sustainable development process that does not involve excessively long working hours.

ASD: Agile Development Techniques (Contd.)



Principle or practice	Description
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Incremental planning	Requirements are recorded on "story cards," and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development "tasks." See Figures 3.5 and 3.6.
On-site customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Refactoring	All developers are expected to refactor the code continuously as soon as potential code improvements are found. This keeps the code simple and maintainable.
Simple design	Enough design is carried out to meet the current requirements and no more.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Sustainable pace	Large amounts of overtime are not considered acceptable, as the net effect is often to reduce code quality and medium-term productivity.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.

ASD: Agile Development Techniques (Contd.)

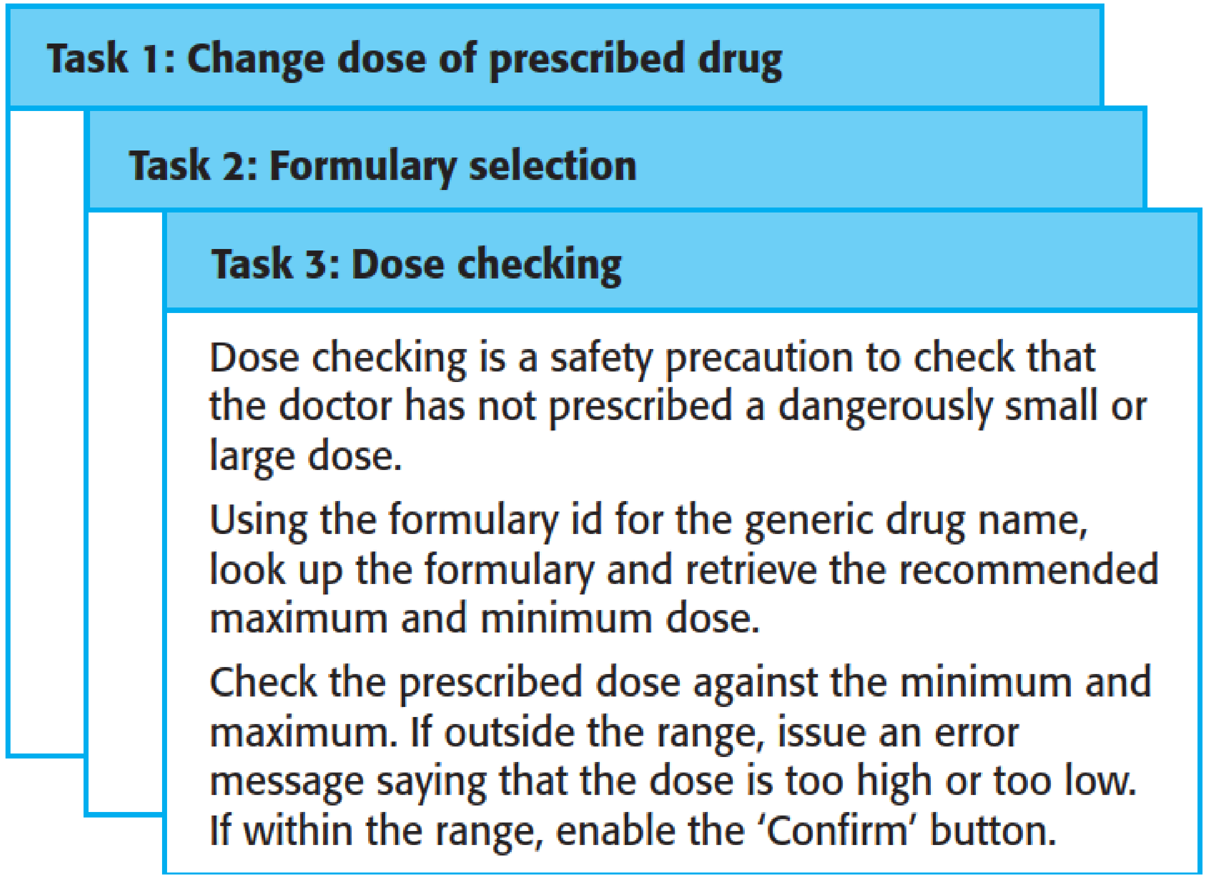
User Stories

- Software requirements always change. To handle these changes, agile methods do not have a separate requirements engineering activity.
- Rather, they integrate requirements elicitation with development. To make this easier, the idea of "user stories" was developed where a user story is a scenario of use that might be experienced by a system user.
- As far as possible, the system customer works closely with the development team and discusses these scenarios with other team members.
- Together, they develop a "story card" that briefly describes a story that encapsulates the customer needs. The development team then aims to implement that scenario in a future release of the software.
- An example of a story card for the Mentcare system is shown in Figure 3.5. This is a short description of a scenario for prescribing medication for a patient.
- User stories may be used in planning system iterations. Once the story cards have been developed, the development team breaks these down into tasks (Figure 3.6) and estimates the effort and resources required for implementing each task.

ASD: Agile Development Techniques (Contd.)

User Stories

- This usually involves discussions with the customer to refine the requirements. The customer then prioritizes the stories for implementation, choosing those stories that can be used immediately to deliver useful business support
- The idea of user stories is a powerful one—people find it much easier to relate to these stories than to a conventional requirements document or use cases.
- User stories can be helpful in getting users involved in suggesting requirements during an initial predevelopment requirements elicitation activity.
- The principal problem with user stories is completeness. It is difficult to judge if enough user stories have been developed to cover all the essential requirements of a system.



Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.

Using the formulary id for the generic drug name, look up the formulary and retrieve the recommended maximum and minimum dose.

Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low. If within the range, enable the 'Confirm' button.

ASD: Agile Development Techniques (Contd.)

Refactoring

- A fundamental precept of traditional software engineering is that you should design for change.
- That is, you should anticipate future changes to the software and design it so that these changes can be easily implemented.
- Extreme programming, however, has discarded this principle on the basis that designing for change is often wasted effort.
- It isn’t worth taking time to add generality to a program to cope with change. Often the changes anticipated never materialize, or completely different change requests may be made.
- In practice, changes will always have to be made to the code being developed. To make these changes easier, the developers of XP suggested that the code being developed should be constantly refactored.
- Refactoring (Fowler et al. 1999) means that the programming team look for possible improvements to the software and implements

ASD: Agile Development Techniques (Contd.)

Refactoring

- A fundamental precept of traditional software engineering is that you should design for change.
- That is, you should anticipate future changes to the software and design it so that these changes can be easily implemented.
- Extreme programming, however, has discarded this principle on the basis that designing for change is often wasted effort.
- It isn’t worth taking time to add generality to a program to cope with change. Often the changes anticipated never materialize, or completely different change requests may be made.
- In practice, changes will always have to be made to the code being developed. To make these changes easier, the developers of XP suggested that the code being developed should be constantly refactored.
- Refactoring (Fowler et al. 1999) means that the programming team look for possible improvements to the software and implements them immediately

ASD: Agile Development Techniques (Contd.)

Refactoring

- When team members see code that can be improved, they make these improvements even in situations where there is no immediate need for them.
- A fundamental problem of incremental development is that local changes tend to degrade the software structure. Consequently, further changes to the software become harder and harder to implement.
- Essentially, the development proceeds by finding workarounds to problems, with the result that code is often duplicated, parts of the software are reused in inappropriate ways, and the overall structure degrades as code is added to the system.
- Refactoring improves the software structure and readability and so avoids the structural deterioration that naturally occurs when software is changed.
- When refactoring is part of the development process, the software should always be easy to understand and change as new requirements are proposed.
- Sometimes development pressure means that refactoring is delayed because the time is devoted to the implementation of new functionality.

ASD: Agile Development Techniques (Contd.)

Test-First Development

- One of the important differences between incremental development and plan-driven development is in the way that the system is tested.
- With incremental development, there is no system specification that can be used by an external testing team to develop system tests.
- Therefore, some approaches to incremental development have a very informal testing process, in comparison with plan-driven testing.
- Extreme Programming developed a new approach to program testing to address the difficulties of testing without a specification.
- Testing is automated and is central to the development process, and development cannot proceed until all tests have been successfully executed.
- The key features of testing in XP are:

  1. test-first development, 2. incremental test development from scenarios, 3. user involvement in the test development and validation, and 4. the use of automated testing frameworks

ASD: Agile Development Techniques (Contd.)

Test-First Development

- One of the important differences between incremental development and plan-driven development is in the way that the system is tested.
- With incremental development, there is no system specification that can be used by an external testing team to develop system tests.
- Therefore, some approaches to incremental development have a very informal testing process, in comparison with plan-driven testing.
- Extreme Programming developed a new approach to program testing to address the difficulties of testing without a specification.
- Testing is automated and is central to the development process, and development cannot proceed until all tests have been successfully executed.
- The key features of testing in XP are:

  1. test-first development, 2. incremental test development from scenarios, 3. user involvement in the test development and validation, and 4. the use of automated testing frameworks

ASD: Agile Development Techniques (Contd.)

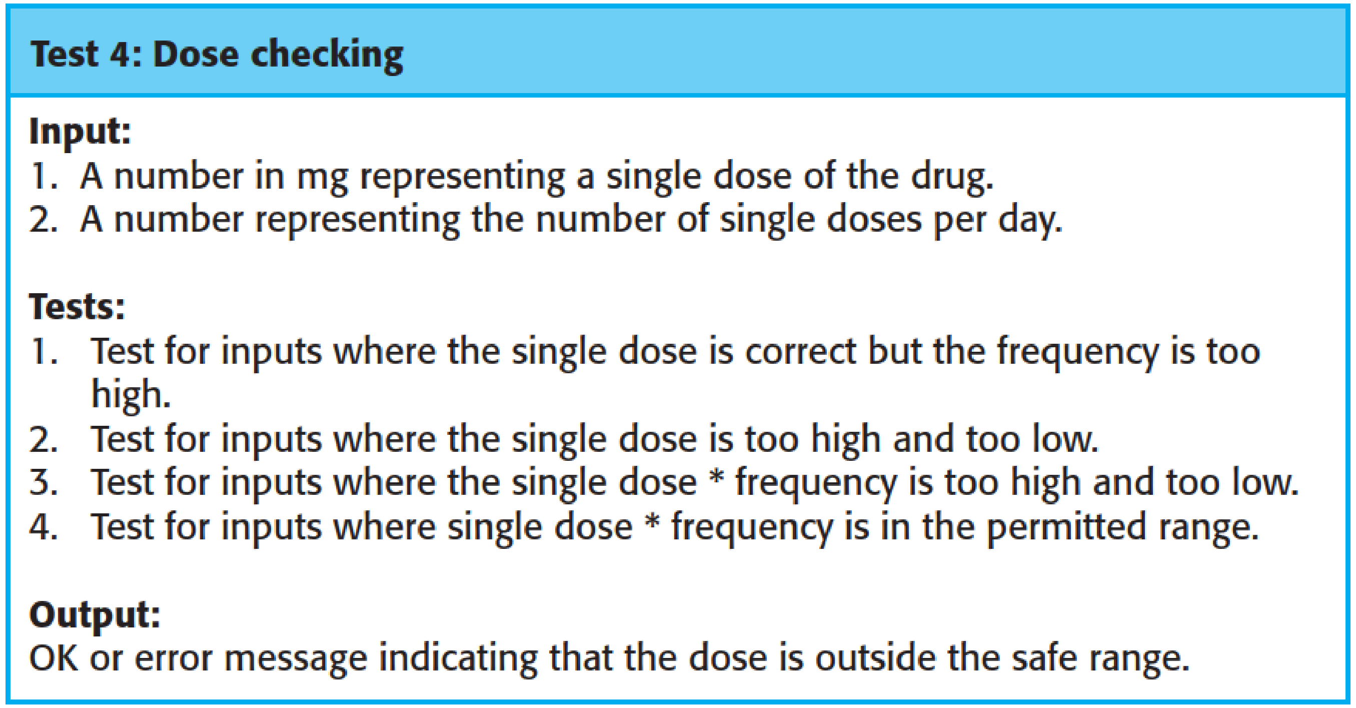
Test-First Development

- XP’s test-first philosophy has now evolved into more general test-driven development techniques (Jeffries and Melnik 2007).
- I believe that test-driven development is one of the most important innovations in software engineering. Instead of writing code and then writing tests for that code, you write the tests before you write the code.
- In test-first development, the task implementers must thoroughly understand the specification so that they can write tests for the system.
- XP’s test-first approach assumes that user stories have been developed, and these have been broken down into a set of task card
- Each task generates one or more-unit tests that check the implementation described in that task. Figure 3.7 is a shortened description of a test case that has been developed to check that the prescribed dose of a drug does not fall outside known safe limits.
- Test automation is essential for test-first development. Tests are written as executable components before the task is implemented.

ASD: Agile Development Techniques (Contd.)

Test-First Development

- These testing components should be stand-alone, should simulate the submission of input to be tested, and should check that the result meets the output specification.
- An automated test framework is a system that makes it easy to write executable tests and submit a set of tests for execution.
- Junit (Tahchiev et al. 2010) is a widely used example of an automated testing framework for Java programs.



Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

- As testing is automated, there is always a set of tests that can be quickly and easily executed. Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

ASD: Agile Development Techniques (Contd.)

Test-First Development

- Test-first development and automated testing usually result in many tests being written and executed. However, there are problems in ensuring that test coverage is complete:
  1. Programmers prefer programming to testing, and sometimes they take shortcuts when writing tests. For example, they may write incomplete tests that do not check for all possible exceptions that may occur
  2. Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the "display logic" and workflow between screens.
- It is difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage. Crucial parts of the system may not be executed and so will remain untested.
- Therefore, although a large set of frequently executed tests may give the impression that the system is complete and correct, this may not be the case.

ASD: Agile Development Techniques (Contd.)

Pair Programming

- Another innovative practice that was introduced in XP is that programmers work in pairs to develop the software. The programming pair sits at the same computer to develop the software. However, the same pair do not always program together.
- Rather, pairs are created dynamically so that all team members work with each other during the development process.
- Pair programming has several advantages
  1. It supports the idea of collective ownership and responsibility for the system. This reflects Weinberg's idea of egoless programming (Weinberg 1971) where the software is owned by the team as a whole and individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
  2. It acts as an informal review process because each line of code is looked at by at least two people. Code inspections and reviews are effective in discovering a high percentage of software errors

ASD: Agile Development Techniques (Contd.)

Pair Programming

3. It encourages refactoring to improve the software structure. The problem with asking programmers to refactor in a normal development environment is that effort involved is expended for long-term benefit. A developer who spends time refactoring may be judged to be less efficient than one who simply carries on developing code. Where pair programming and collective ownership are used, others benefit immediately from the refactoring, so they are likely to support the process.

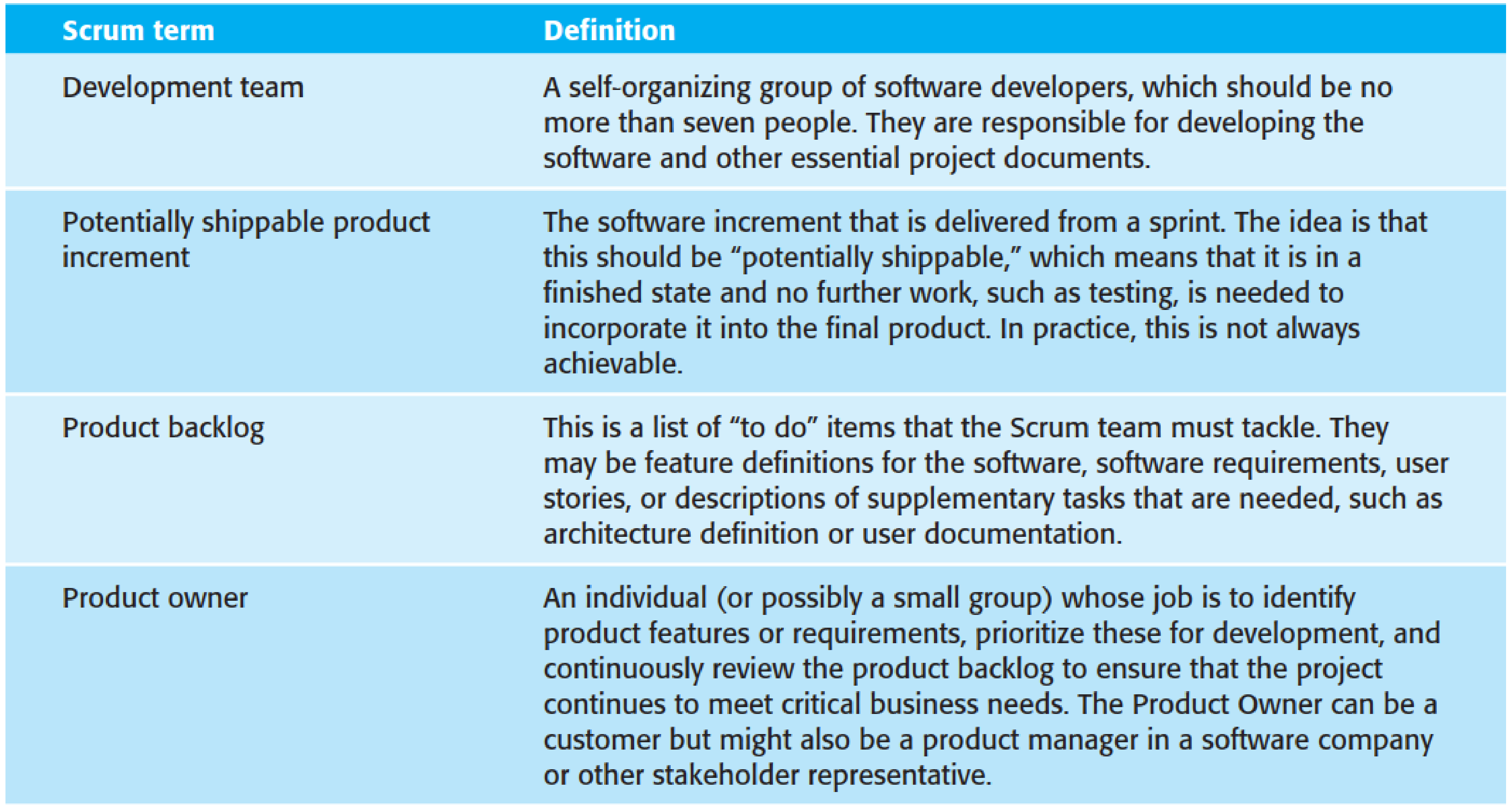
- Formal studies of the value of pair programming have had mixed results. Using student volunteers, Williams and her collaborators (Williams et al. 2000) found that productivity with pair programming seems to be comparable to that of two people working independently.
- The reasons suggested are that pairs discuss the software before development and so probably have fewer false starts and less rework. Furthermore, the number of errors avoided by the informal inspection is such that less time is spent repairing bugs discovered during the testing process.

ASD: Agile Project Management

- In any software business, managers need to know what is going on and whether or not a project is likely to meet its objectives and deliver the software on time with the proposed budget. Plan-driven approaches to software development evolved to meet this need.
- A plan-based approach requires a manager to have a stable view of everything that has to be developed and the development processes.
- The informal planning and project control that was proposed by the early adherents of agile methods clashed with this business requirement for visibility.
- Teams were self-organizing, did not produce documentation, and planned development in very short cycles.
- While this can and does work for small companies developing software products, it is inappropriate for larger companies who need to know what is going on in their organization.
- Like every other professional software development process, agile development has to be managed so that the best use is made of the time and resources available to the team. To address this issue, the Scrum agile method was developed (Schwaber and Beedle 2001; Rubin 2013) to provide a framework for organizing agile projects and, to some extent at least, provide external visibility of what is going on.

ASD: Agile Project Management

- The developers of Scrum wished to make clear that Scrum was not a method for project management in the conventional sense, so they deliberately invented new terminology, such as ScrumMaster, which replaced names such as project manager.
- Figure 3.8 summarizes Scrum terminology and what it means.



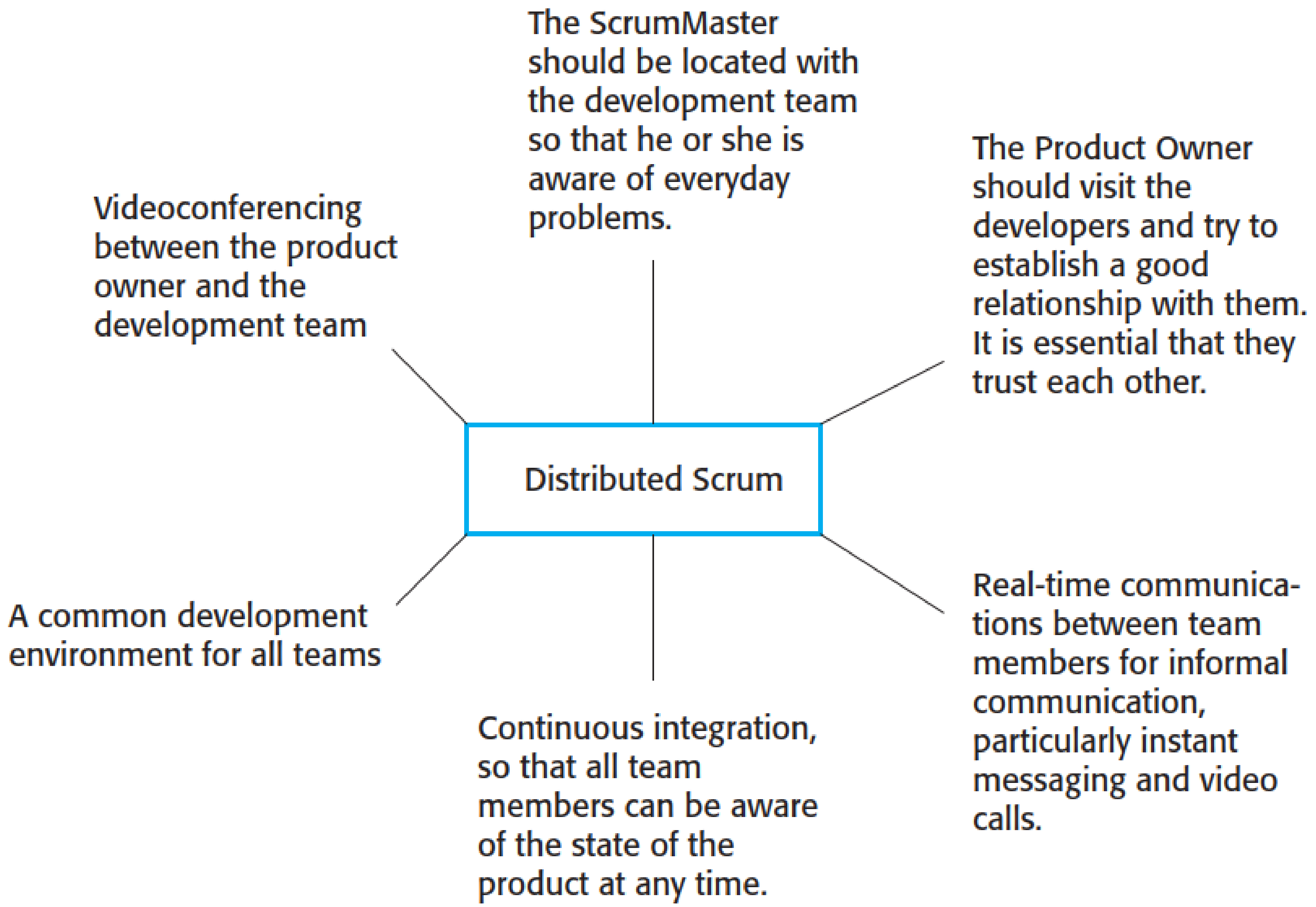
Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than seven people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be "potentially shippable," which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of "to do" items that the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories, or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development, and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

ASD: Agile Project Management

ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2 to 4 weeks long.
Velocity	An estimate of how much product backlog effort a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

- Scrum focuses on providing a framework for agile project organization, and it does not mandate the use of specific development practices such as pair programming and test-first development.
- This means that it can be more easily integrated with existing practice in a company. Consequently, as agile methods have become a mainstream approach to software development, Scrum has emerged as the most widely used method.
- The Scrum process or sprint cycle is shown in Figure. The input to the process is the product backlog. Each process iteration produces a product increment that could be delivered to customers.

ASD: Agile Project Management


The ScrumMaster should be located with the development team so that he or she is aware of everyday problems.
Videoconferencing between the product owner and the development team
The Product Owner should visit the developers and try to establish a good relationship with them. It is essential that they trust each other.
Distributed Scrum
A common development environment for all teams
Real-time communications between team members for informal communication, particularly instant messaging and video calls.
Continuous integration, so that all team members can be aware of the state of the product at any time.

ASD: Scaling Agile Methods

- Of course, the need for faster delivery of software, which is more suited to customer needs, also applies to both larger systems and larger companies. Consequently, over the past few years, a lot of work has been put into evolving agile methods for both large software systems and for use in large companies.
- Scaling agile methods has closely related facets:
  1. Scaling up these methods to handle the development of large systems that are too big to be developed by a single small team.
  2. Scaling out these methods from specialized development teams to more widespread use in a large company that has many years of software development experience.

ASD: Scaling Agile Methods

Practical Problems with Agile Methods

- For large, long-lifetime systems that are developed by a software company for an external client, using an agile approach presents a few problems.
  1. The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies.
  2. Agile methods are most appropriate for new software development rather than for software maintenance. Yet the majority of software costs in large companies come from maintaining their existing software systems.
  3. Agile methods are designed for small co-located teams, yet much software development now involves worldwide distributed teams.

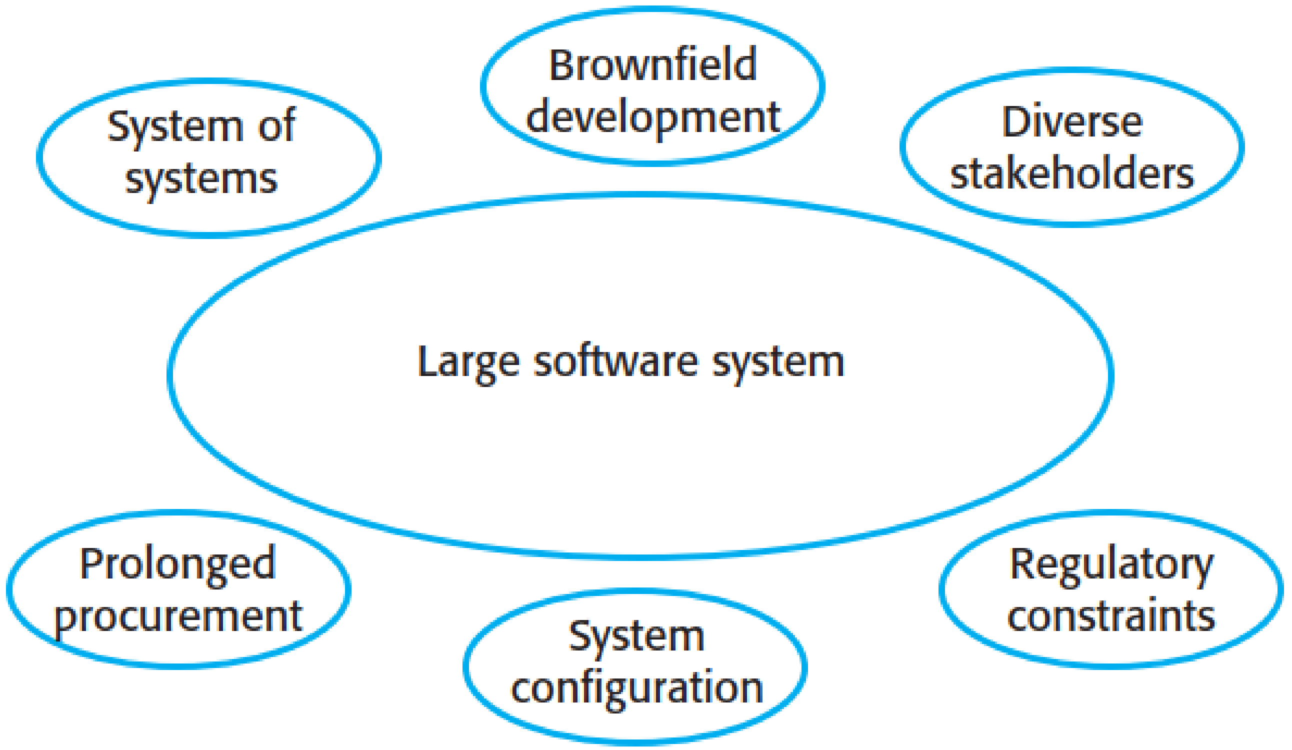
ASD: Scaling Agile Methods

Agile and Plan-Driven Methods

ASD: Scaling Agile Methods

Agile Methods for Large systems

- Agile methods must evolve to be used for large-scale software development. The fundamental reason for this is that large-scale software systems are much more complex and difficult to understand and manage than small-scale systems or software products.
- Six principal factors (Figure 3.13) contribute to this complexity:



ASD: Scaling Agile Methods

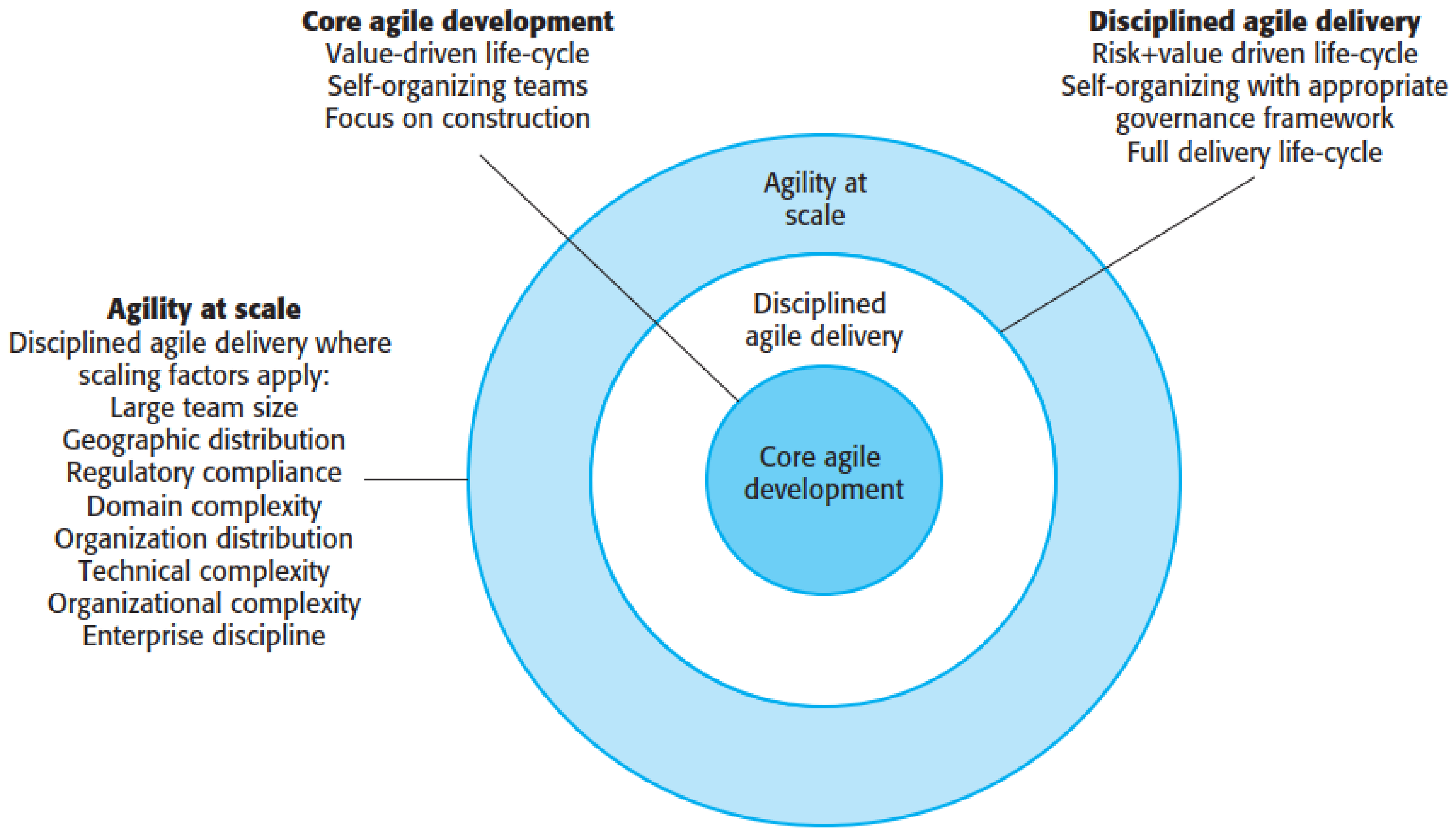
- Large systems are usually systems of systems—collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones. It is practically impossible for each team to have a view of the whole system. Consequently, their priorities are usually to complete their part of the system without regard for wider systems issues.
- Large systems are brownfield systems (Hopkins and Jenkins 2008); that is, they include and interact with several existing systems. Many of the system requirements are concerned with this interaction and so don’t really lend themselves to flexibility and incremental development. Political issues can also be significant here.
- Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development. This is not necessarily compatible with incremental development and frequent system integration.
- Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed, that require certain types of system documentation to be produced, and so on

ASD: Scaling Agile Methods

- Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.
- Large systems usually have a diverse set of stakeholders with different perspectives and objectives. For example, nurses and administrators may be the end-users of a medical system, but senior medical staff, hospital managers, and others, are also stakeholders in the system. It is practically impossible to involve all these different stakeholders in the development process.

ASD: Scaling Agile Methods

- IBM has also developed a framework for the large-scale use of agile methods called the Agile Scaling Model (ASM). Figure 3.14, taken from Ambler's white paper that discusses ASM (Ambler 2010), shows an overview of this model.


Core agile development
Value-driven life-cycle
Self-organizing teams
Focus on construction
Disciplined agile delivery
Risk+value driven life-cycle
Self-organizing with appropriate governance framework
Full delivery life-cycle
Agility at scale
Disciplined agile delivery where scaling factors apply:
Large team size
Geographic distribution
Regulatory compliance
Domain complexity
Organization distribution
Technical complexity
Organizational complexity
Enterprise discipline
Agility at scale
Disciplined agile delivery
Core agile development

ASD: Scaling Agile Methods

- IBM has also developed a framework for the large-scale use of agile methods called the Agile Scaling Model (ASM). Figure, taken from Ambler's white paper that discusses ASM (Ambler 2010), shows an overview of this model.
- The ASM recognizes that scaling is a staged process where development teams move from the core agile practices discussed here to what is called Disciplined Agile Delivery. Essentially, this stage involves adapting these practices to a disciplined organizational setting and recognizing that teams cannot simply focus on development but must also consider other stages of the software engineering process, such as requirements and architectural design.
- No single model is appropriate for all large-scale agile products as the type of product, the customer requirements, and the people available are all different. However, approaches to scaling agile methods have several things in common:

1. A completely incremental approach to requirements engineering is impossible. Some early work on initial software requirements is essential. You need this work to identify the different parts of the system that may be developed by different teams and, often, to be part of the contract for the system development.

ASD: Scaling Agile Methods

2. There cannot be a single product owner or customer representative. Different people must be involved for different parts of the system, and they have to continuously communicate and negotiate throughout the development process.
3. It is not possible to focus only on the code of the system. You need to do more up-front design and system documentation. The software architecture has to be designed, and there has to be documentation produced to describe critical aspects of the system, such as database schemas and the work breakdown across teams.
4. Cross-team communication mechanisms have to be designed and used. This should involve regular phone and videoconferences between team members and frequent, short electronic meetings where teams update each other on progress. A range of communication channels such as email, instant messaging, wikis, and social networking systems should be provided to facilitate communications
5. Continuous integration, where the whole system is built every time any developer checks in a change, is practically impossible when several separate programs have to be integrated to create the system. However, it is essential to maintain frequent system builds and regular releases of the system. Configuration management tools that support multi-team software development are essential.

ASD: Scaling Agile Methods

Scrum has been adapted for large-scale development. The key characteristics of multi-team Scrum are:

1. Role replication Each team has a Product Owner for its work component and ScrumMaster. There may be a chief Product Owner and ScrumMaster for the entire project.
2. Product architects Each team chooses a product architect, and these architects collaborate to design and evolve the overall system architecture.
3. Release alignment The dates of product releases from each team are aligned so that a demonstrable and complete system is produced.
4. Scrum of Scrums There is a daily Scrum of Scrums where representatives from each team meet to discuss progress, identify problems, and plan the work to be done that day. Individual team Scrums may be staggered in time so that representatives from other teams can attend if necessary.

ASD: Scaling Agile Methods

Agile Methods across Organizations

- Small software companies that develop software products have been among the most enthusiastic adopters of agile methods.
- These companies are not constrained by organizational bureaucracies or process standards, and they can change quickly to adopt new ideas.
- Of course, larger companies have also experimented with agile methods in specific projects, but it is much more difficult for them to “scale out” these methods across the organization.
- It can be difficult to introduce agile methods into large companies for several reasons:
  1. Project managers who do not have experience of agile methods may be reluctant to accept the risk of a new approach, as they do not know how this will affect their projects.
  2. Large organizations often have quality procedures and standards that all projects are expected to follow, and, because of their bureaucratic nature, these are likely to be incompatible with agile methods. Sometimes, these are supported by software tools (e.g., requirements management tools), and the use of these tools is mandated for all projects.

ASD: Scaling Agile Methods

Agile Methods across Organizations

3. Agile methods seem to work best when team members have a relatively high skill level. However, within large organizations, there are likely to be a wide range of skills and abilities, and people with lower skill levels may not be effective team members in agile processes.
4. There may be cultural resistance to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.

Thank You