Process activities

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 system's operation & 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 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.

Requirements specification Requirements specification is the activity of translating the information gathered during requirements analysis into a document that defines a set of requirements.

Requirements validation This activity checks the requirements for realism, consistency, and completeness.

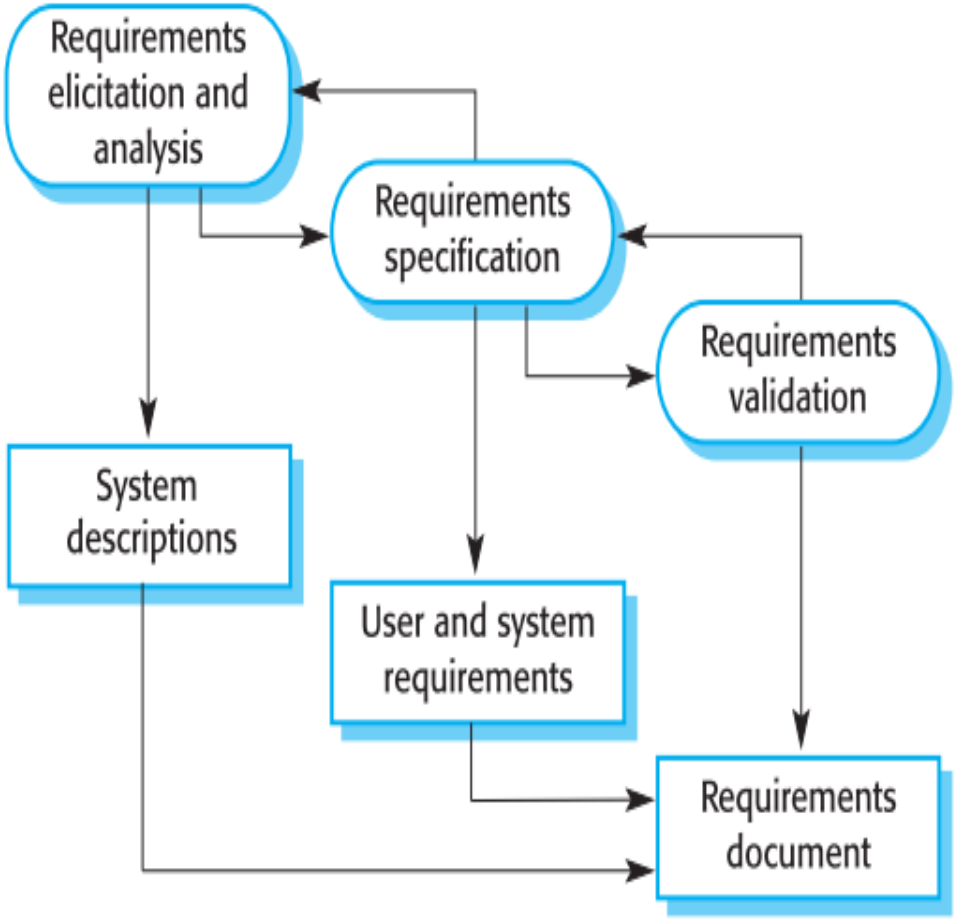


Figure 2.4 The requirements engineering process

Process activities

Software Design and implementation The implementation stage of software development is the process of developing an executable system for delivery to the customer.

Sometimes this involves separate activities of software design and programming. However, if an agile approach to development is used, design and implementation are interleaved, with no formal design documents produced during the process. Of course, the software is still designed, but the design is recorded informally on whiteboards and programmer's notebooks.

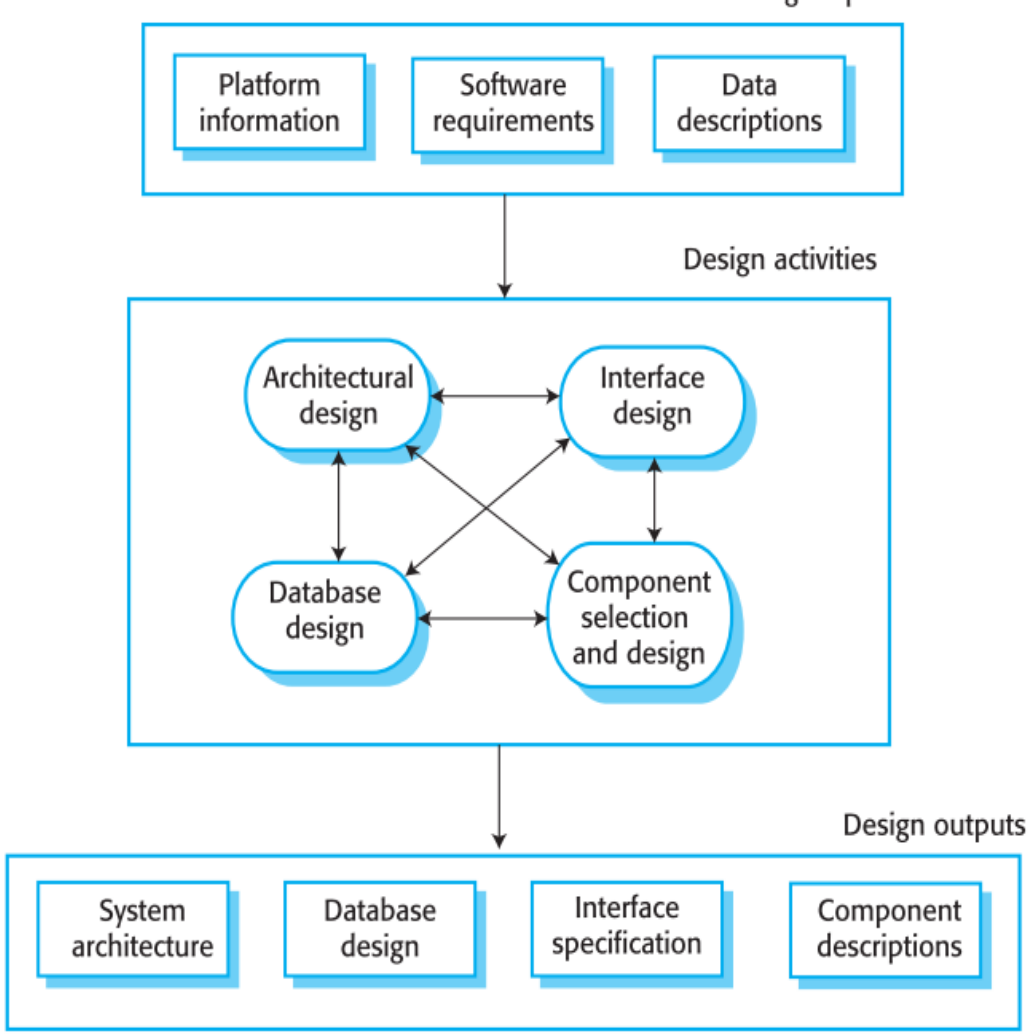


Figure 2.5 A general model of the design process

Architectural design, where you identify the overall structure of the system, the principal components (sometimes called subsystems or modules), their relationships, and how they are distributed.

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.

Alternatively, it may be a list of changes to be made to a reusable component or a detailed design model expressed in the UML.

Then be used to automatically generate an implementation

Process activities

Software validation or, more generally, verification and validation (V & V) is intended to show that a system both conforms to its specification and meets the expectations of the system customer.

Program testing, where the system is executed using simulated test data, is the principal validation technique.

Validation may also involve checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development.

However, most V & V time and effort is spent on program testing.

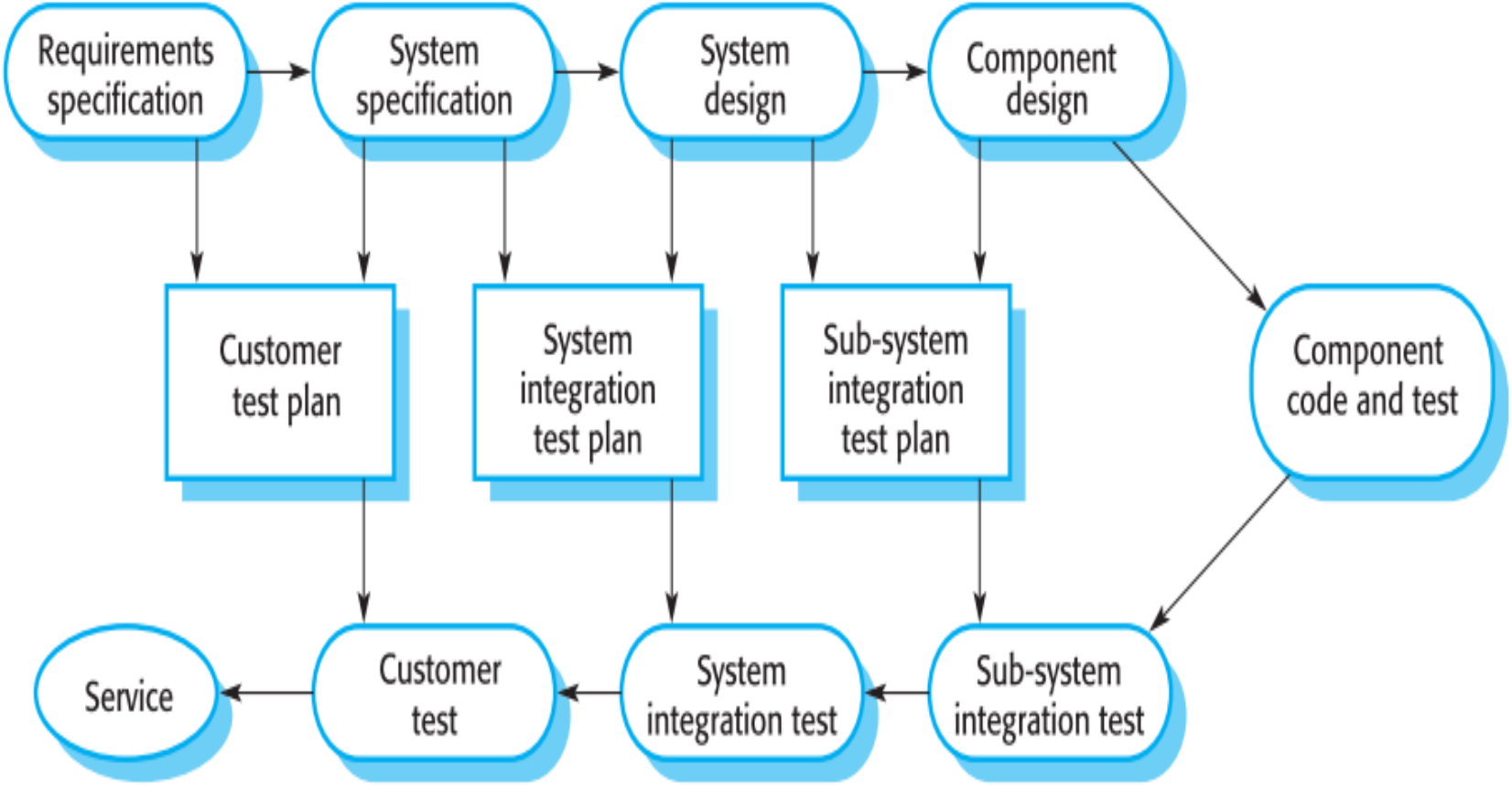
The stages in the testing process are:

1. Component testing The components making up the system are tested by the people developing the system.
2. Each component is tested independently, without other system components.

2. System testing System components are integrated to create a complete system.

This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems.

3. Customer testing This is the final stage in the testing process before the system is accepted for operational use.



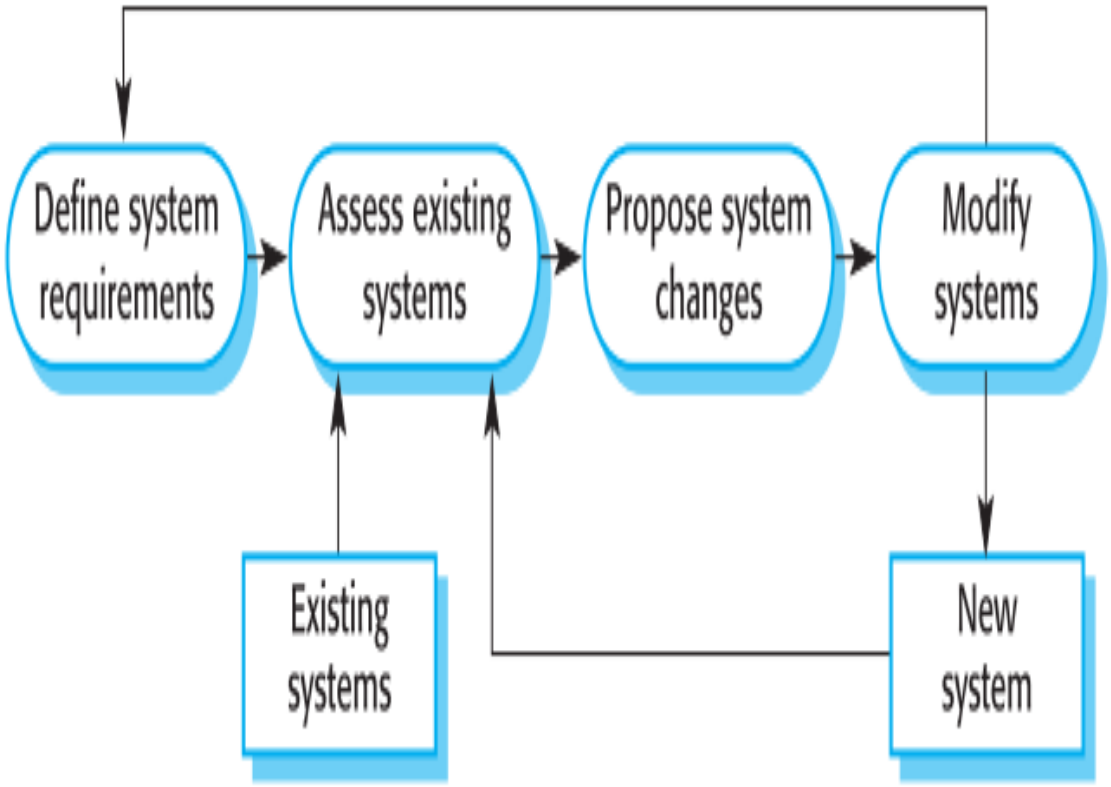


Figure 2.8 Software system evolution

Process activities

Software Evaluation The flexibility of software is one of the main reasons why more and more software is being incorporated into large, complex systems.

Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design.

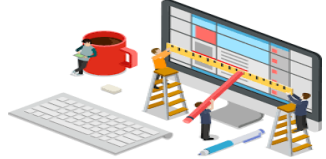
However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware.

Software Engineering

Summarized & Presented By Dr.Engineer

IBRAHIM ESKANDAR

School of Human Development and Techno-communication
UNIVERSITY MALAYSIA PERLIS



- **Understand the rationale for agile software development methods the agile manifesto, and the differences between agile and plan-driven development.**

- **Know about important agile development practices such as user stories, refactoring, pair programming and test-first development.**

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 has to be developed quickly to take advantage of new opportunities and to respond to competitive pressure.

Businesses may be willing to trade off software quality and compromise on requirements if they can deploy essential new software quickly.

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.

software development

Businesses are operating in a changing environment, it is practically impossible to derive a complete set of stable software requirements.

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.

Plan-driven software development processes that completely specify the requirements and then design, build, and test a system are not geared to rapid software development.

safety-critical control systems ->plan-driven

Agile: XP DSDM Scrum

2-3weeks customer no doc incorporate and not formal meetings

Agile Characteristics

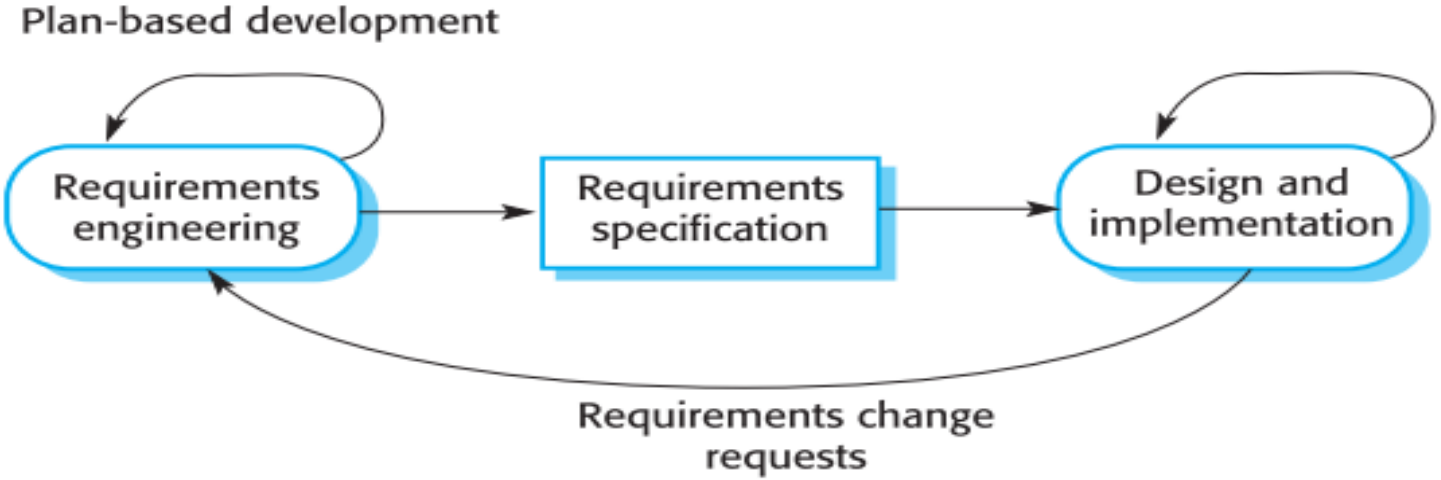
The processes of specification, design and implementation are interleaved.

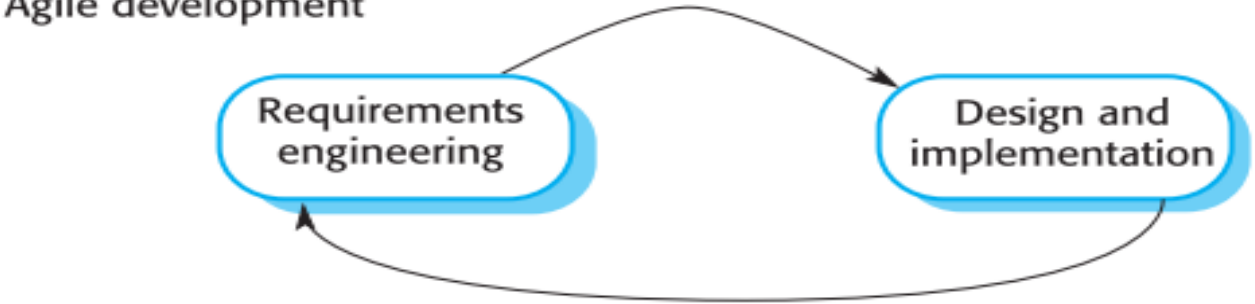
There is no detailed system specification, and design documentation is minimized or generated automatically by the programming environment used to implement the system. The user requirements document is an outline definition of the most important characteristics of the system.

The system is developed in a series of increments. End-users and other system stakeholders are involved in specifying and evaluating each increment. They may propose changes to the software and new requirements that should be implemented in a later version of the system.

Extensive tool support is used to support the development process. Tools that may be used include automated testing tools, tools to support configuration management, and system integration and tools to automate user interface production.

Development: Agile-Plan Based





Agile Method:Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more[†].

Agile

Agile methods have been particularly successful for two kinds of system development.

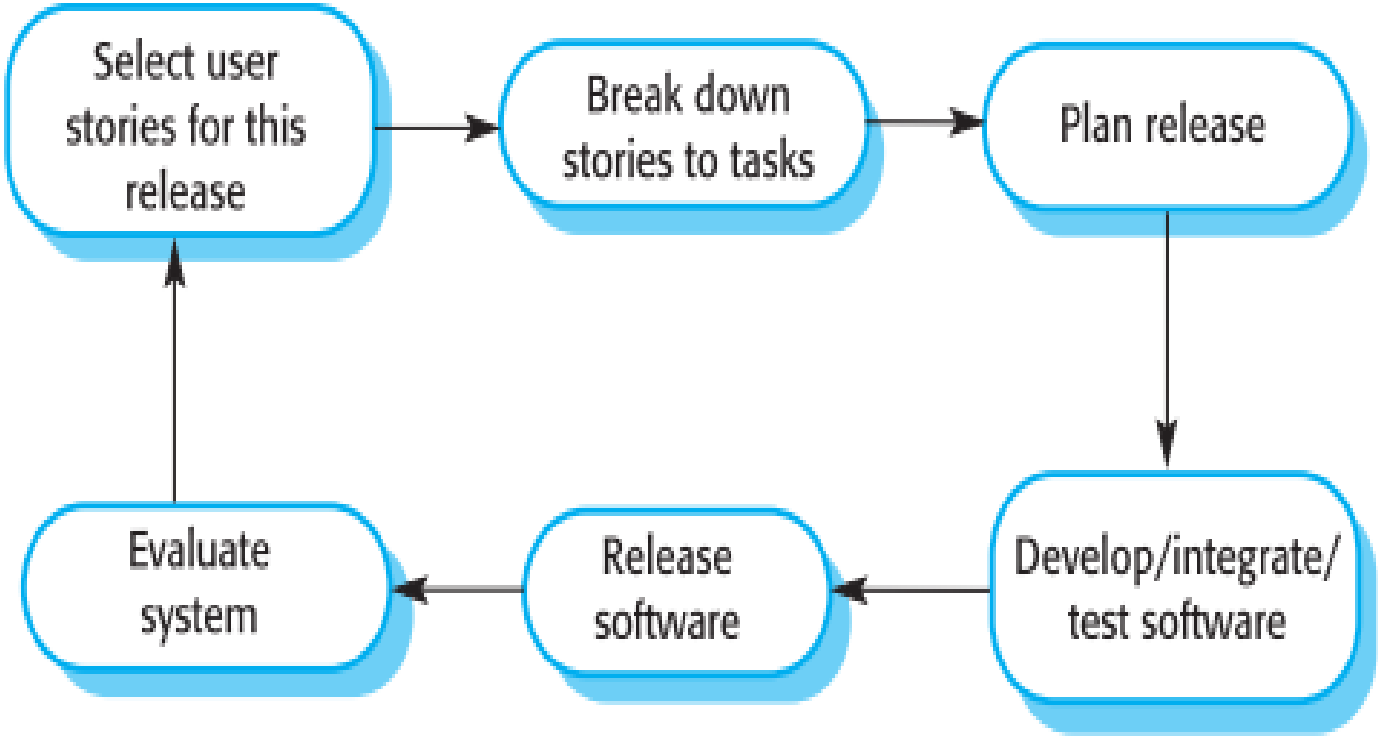
Product development where a software company is developing a small or medium-sized product for sale. Virtually all software products and apps are now developed using an agile approach.

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.

continuous communications - standalone system -

informal communications

Agile: XP



Agile Method

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.

Agile

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 owner-ship of the system code, and a sustainable development process that does not involve excessively long working hours.**

Agile

4. Change is embraced through regular system releases to customers, test-first development, refactoring to avoid code degeneration, and continuous integration of new functionality.

5. Maintaining simplicity is supported by constant refactoring that improves code quality and by using simple designs that do not unnecessarily anticipate future changes to the system.

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.

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.

Prescribing medication

Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer so she clicks on the medication field and can select 'current medication', 'new medication' or 'formulary'.

If she selects 'current medication', the system asks her to check the dose; If she wants to change the dose, she enters the new dose then confirms the prescription.

If she chooses 'new medication', the system assumes that she knows which medication to prescribe. She types the first few letters of the drug name. The system displays a list of possible drugs starting with these letters. She chooses the required medication and the system responds by asking her to check that the medication selected is correct. She enters the dose then confirms the prescription.

If she chooses 'formulary', the system displays a search box for the approved formulary. She can then search for the drug required. She selects a drug and is asked to check that the medication is correct. She enters the dose then confirms the prescription.

The system always checks that the dose is within the approved range. If it isn't, Kate is asked to change the dose.

After Kate has confirmed the prescription, it will be displayed for checking. She either clicks 'OK' or 'Change'. If she clicks 'OK', the prescription is recorded on the audit database. If she clicks on 'Change', she reenters the 'Prescribing medication' process.

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.

User Stories

The intention is to identify useful functionality that can be implemented in about two weeks, when the next release of the system is made available to the customer.

As requirements change, the unimplemented stories change or may be discarded.

If changes are required for a system that has already been delivered, new story cards are developed and again, the customer decides whether these changes should have priority over new functionality.

User Stories

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.

Completeness judge about activity or the requirements.

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 actually be made.

"Deterioration"

To make these changes easier "code change", the developers of XP suggested that the code being developed should be constantly refactored.

CLASS- structure & readability-duplicate code & use in other..

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 a number of advantages.

1. It supports the idea of collective ownership and responsibility for t he system.
2. It acts as an informal review process because each line of code is looked at by at least two people.
3. It encourages refactoring to improve the software structure. The problem with asking programmers to refactor in a normal development environment is that effort.

Pair-Programming

You might think that pair programming would be less efficient than individual programming. In a given time, a pair of developers would produce half as much code as two individuals working alone.

Many companies that have adopted agile methods are suspicious of pair programming and do not use it. Other companies mix pair and individual programming with an experienced programmer working with a less experienced colleague when they have problems.

Formal studies of the value of pair programming have had mixed results.

Pair-Programming

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.

They found that there was a significant loss of productivity compared with two programmers working alone. There were some quality benefits, but these did not fully compensate for the pair-programming overhead.

Nevertheless, the sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.

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.

As a consequence, 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:

Test-First Development

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.

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.

Test-First Development

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.

Test-First Development

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.

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.

Software Engineering

Summarized & Presented By Dr.Engineer

IBRAHIM ESKANDAR

School of Human Development and Techno-communication
UNIVERSITY MALAYSIA PERLIS

The Aims

- **Understand the Scrum approach to agile project management.**
- **Understand the issues of scaling agile development methods and combining agile approaches with plan-driven approaches in the development of large software systems.**

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 pro-posed budget. Plan-driven approaches to software development evolved to meet this need.

Managers draw up a plan for the project showing what should be delivered, when it should be delivered, and who will work on the development of the project deliverables. A plan-based approach requires a manager to have a stable view of everything that has to be developed and the development processes.

Informal planning and project control that was proposed by the early adherents of agile methods clashed with this business requirement for visibility.

Agile project management

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.

Scrum is an agile method insofar as it follows the principles from the agile manifesto. However, it 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.

Agile project management Scrum

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.
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.

Agile project management Scrum

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.

Agile project management Scrum

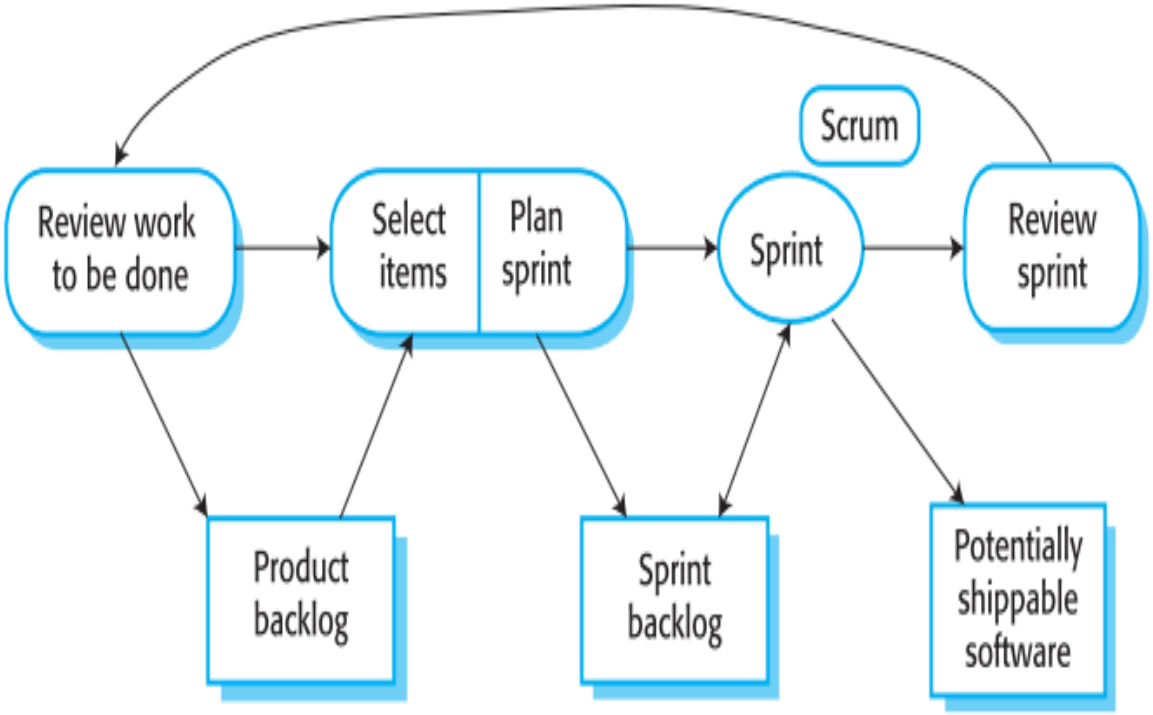


Figure 3.9 The Scrum sprint cycle

Agile project management Scrum

Each sprint cycle lasts a fixed length of time, which is usually between 2 and 4 weeks. At the beginning of each cycle, the Product Owner prioritizes the items on the product backlog to define which are the most important items to be developed in that cycle. Sprints are never extended to take account of unfinished work. Items are returned to the product backlog if these cannot be completed within the allocated time for the sprint.

The whole team is then involved in selecting which of the highest priority items they believe can be completed. They then estimate the time required to complete these items. To make these estimates, they use the velocity attained in previous sprints, that is, how much of the backlog could be covered in a single sprint. This leads to the creation of a sprint backlog—the work to be done during that sprint. The team self-organizes to decide who will work on what, and the sprint begins.

Agile project management Scrum

During the sprint, the team holds short daily meetings (Scrums) to review progress and, where necessary, to re-prioritize work. During the Scrum, all team members share information, describe their progress since the last meeting, bring up problems that have arisen, and state what is planned for the following day. Thus, everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them. Everyone participates in this short-term planning; there is no top-down direction from the Scrum Master.

The daily interactions among Scrum teams may be coordinated using a Scrum board. This is an office whiteboard that includes information and post-it notes about the Sprint backlog, work done, unavailability of staff, and so on . This is a shared resource for the whole team, and anyone can change or move items on the board. It means that any team member can, at a glance, see what others are doing and what work remains to be done.

Agile project management Scrum

At the end of each sprint, there is a review meeting, which involves the whole team. This meeting has two purposes.

First, it is a means of process improvement. The team reviews the way they have worked and reflects on how things could have been done better.

Second, it provides input on the product and the product state for the product backlog review that precedes the next sprint.

Agile project management Scrum

things that users like about the Scrum method are:

1. **The product is broken down into a set of manageable and understandable chunks that stakeholders can relate to.**
2. **Unstable requirements do not hold up progress.**
3. **The whole team has visibility of everything, and consequently team communication and morale are improved.**
4. **Customers see on-time delivery of increments and gain feedback on how the product works. They are not faced with last-minute surprises when a team announces that software will not be delivered as expected.**
5. **Trust between customers and developers is established, and a positive culture is created in which everyone expects the project to succeed.**

Agile project management Scrum Distributed

Scrum, as originally designed, was intended for use with co-located teams where all team members could get together every day in stand-up meetings. However, much software development now involves distributed teams, with team members located in different places around the world. This allows companies to take advantage of lower cost staff in other countries, makes access to specialist skills possible, and allows for 24-hour development, with work going on in different time zones.

Consequently, there have been developments of Scrum for distributed development environments and multi-team working. Typically, for offshore development, the product owner is in a different country from the development team, which may also be distributed. Figure 3.10 shows the requirements for Distributed Scrum (Deemer 2011).

Agile project management Scrum Distributed

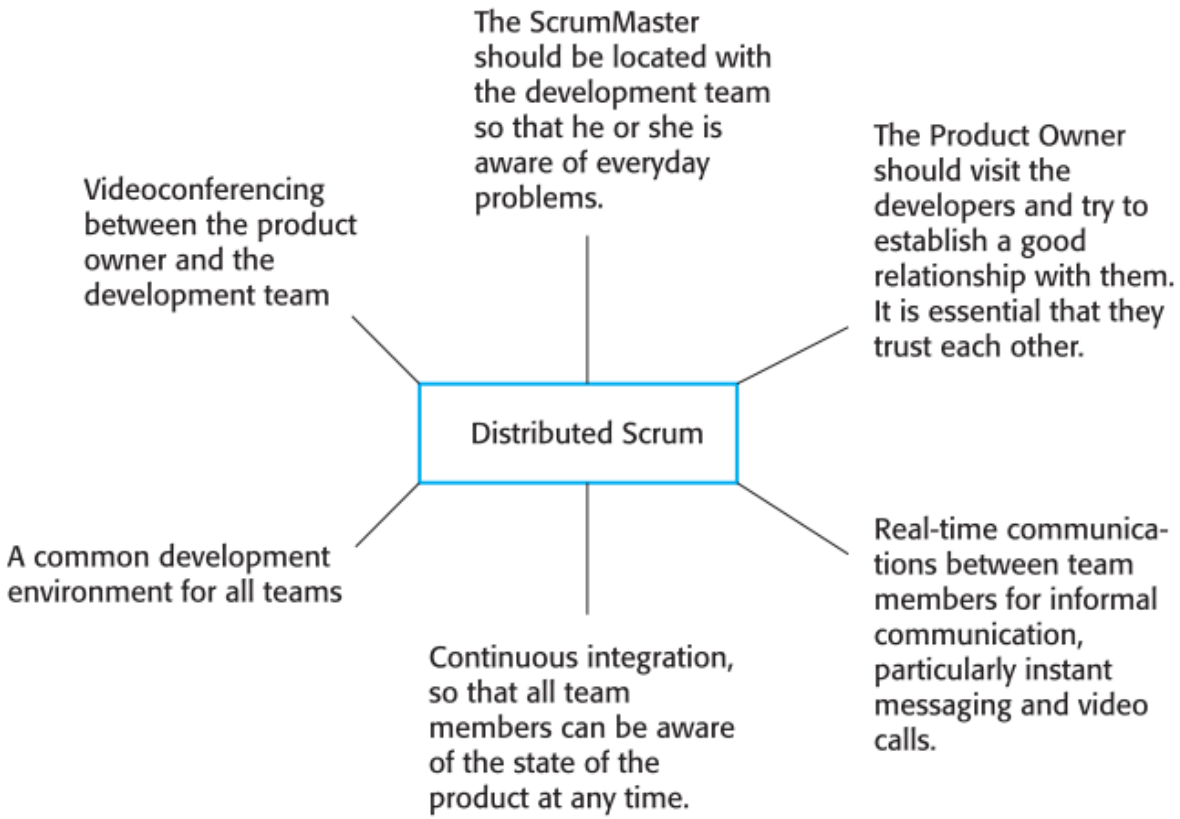


Figure 3.10 Distributed Scrum

Scaling Agile Methods

Agile methods were developed for use by small programming teams that could work together in the same room and communicate informally. They were originally used by for the development of small and medium-sized systems and software products. Small companies, without formal processes or bureaucracy, were enthusiastic initial adopters of these methods.

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.**

Practical problems with agile methods

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.

Contractual issues can be a major problem when agile methods are used. When the system customer uses an outside organization for system development, a contract for the software development is drawn up between them.

Practical problems with agile methods

However, where maintenance involves a custom system that must be changed in response to new business requirements, there is no clear consensus on the suitability of agile methods for software maintenance (Bird 2011; Kilner 2012) . Three types of

problems can arise:

- **Lack of product documentation**
- **Keeping customers involved**
- **Development team continuity**

Agile and plan-driven methods

A fundamental requirement of scaling agile methods is to integrate them with plan-driven approaches. Small startup companies can work with informal and short-term planning, but larger companies have to have longer-term plans and budgets for investment, staffing, and business development. Their software development must support these plans, so longer-term software planning is essential.

To address these problems, most large "agile" software development projects combine practices from plan-driven and agile approaches. Some are mostly agile, and others are mostly plan-driven but with some agile practices. To decide on the balance between a plan-based and an agile approach, you have to answer a range of technical, human and organizational questions. These relate to the system being developed, the development team, and the organizations that are developing and procuring the system.

Agile and plan-driven methods

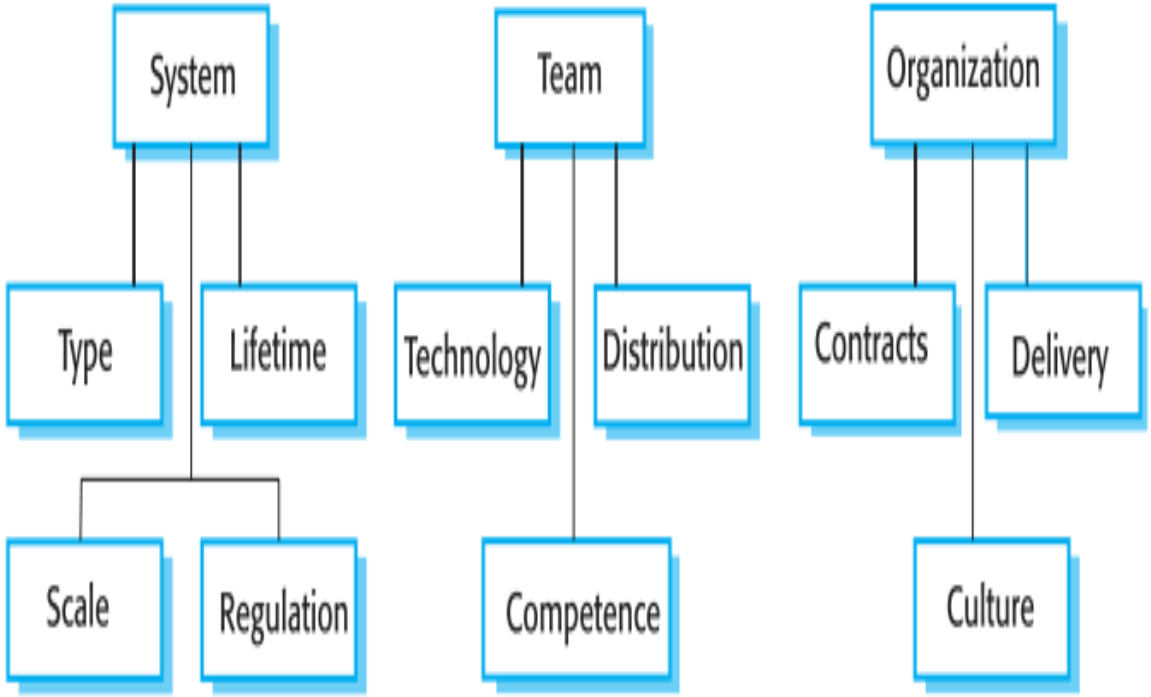


Figure 3.12 Factors influencing the choice of plan-based or agile development

Agile and plan-driven methods

Some of the key issues of mapping (agile & plan driven) are as follows:

1. **How large is the system that is being developed?**

2. **What type of system is being developed? Systems that require a lot of analysis before implementation (e.g., real-time system with complex timing requirements) usually need a fairly detailed design to carry out this analysis. A plan-driven approach may be best in those circumstances.**

3. **What is the expected system lifetime? Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.**

4. **Is the system subject to external regulation? If a system has to be approved by an external regulator.**

Agile methods across organizations

It can be difficult to introduce agile methods into large companies for a number of reasons:

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 particular projects.

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.

Agile methods seem to work best when team members have a relatively high skill level.

There may be cultural resistance to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.

Agile and plan-driven methods

Agile methods place a great deal of responsibility on the development team to cooperate and communicate during the development of the system. They rely on individual engineering skills and software support for the development process.

However, in reality, not everyone is a highly skilled engineer, people do not communicate effectively, and it is not always possible for teams to work together. Some planning may be required to make the most effective use of the people available. Key issues are:

How good are the designers and programmers in the development-team?

How is the development team organized?

What technologies are available to support system development?

Agile methods for large systems

Television and films have created a popular vision of software companies as informal organizations run by young men (mostly) who provide a fashionable working environment, with a minimum of bureaucracy and organizational procedures.

This is far from the truth. Most software is developed in large companies that have established their own working practices and procedures. Management in these companies may be uncomfortable with the lack of documentation and the informal decision making in agile methods. Key issues are:

- Is it important to have a very detailed specification and design before moving to implementation, perhaps for contractual reasons?

- Is an incremental delivery strategy, where you deliver the software to customers or other system stakeholders and get rapid feedback from them, realistic?

- Are there cultural issues that may affect system development?

Agile methods for large systems

Agile methods have to 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:

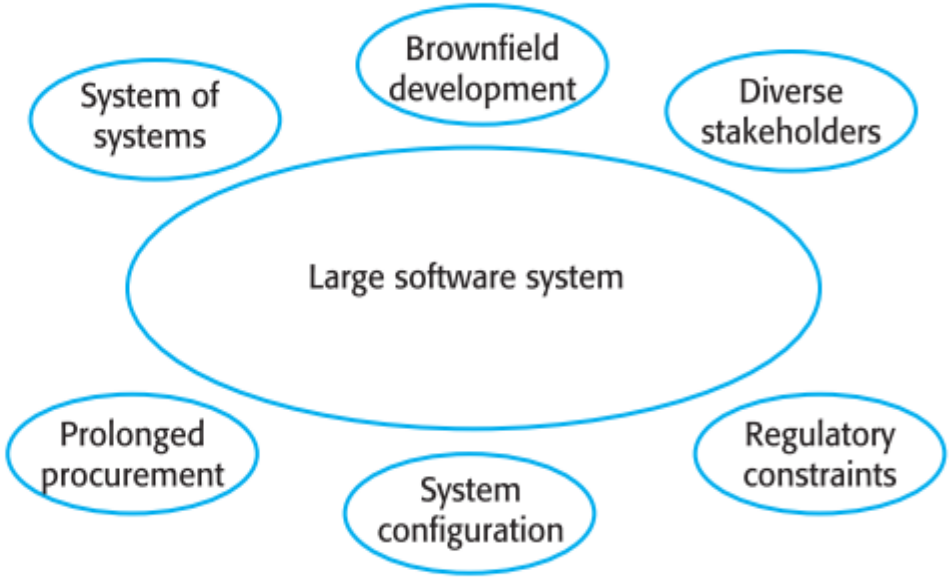


Figure 3.13 Large project characteristics