

# Chapter 1 – Agile Development

# Rapid software development

---

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Rapid software development
  - Specification, design and implementation are inter-leaved
  - System is developed as a series of versions with stakeholders involved in version evaluation
  - User interfaces are often developed using an IDE and graphical toolset.

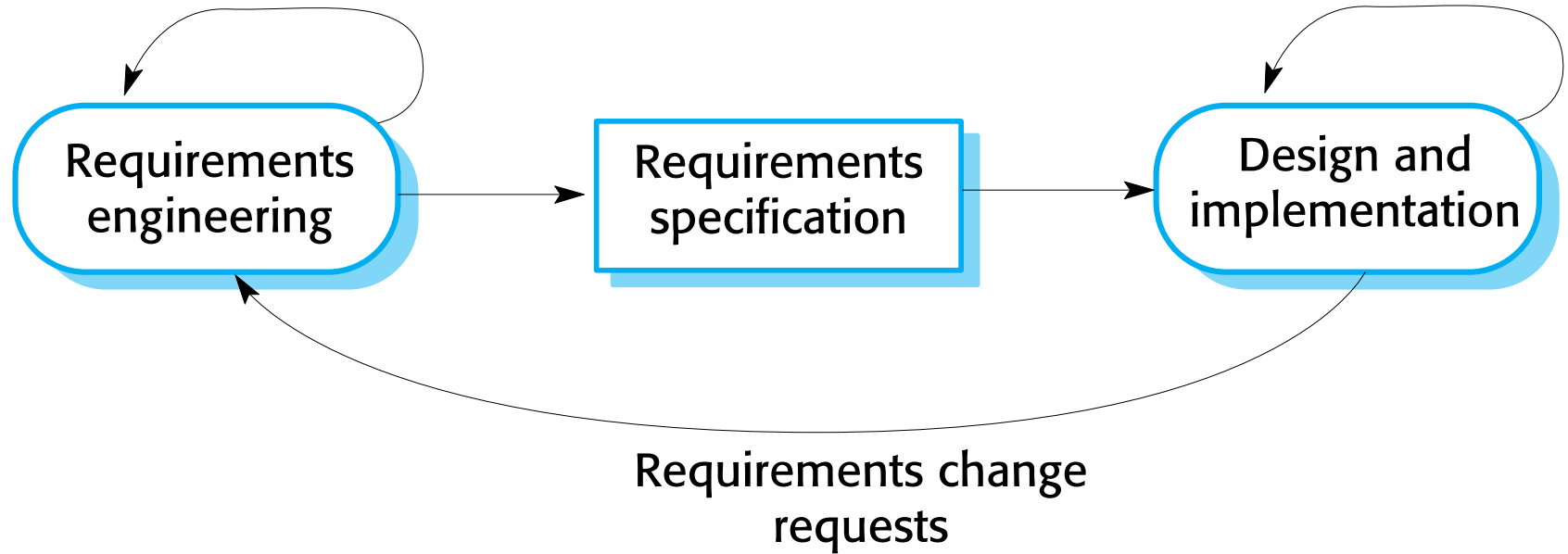
# Plan-driven and agile development

---

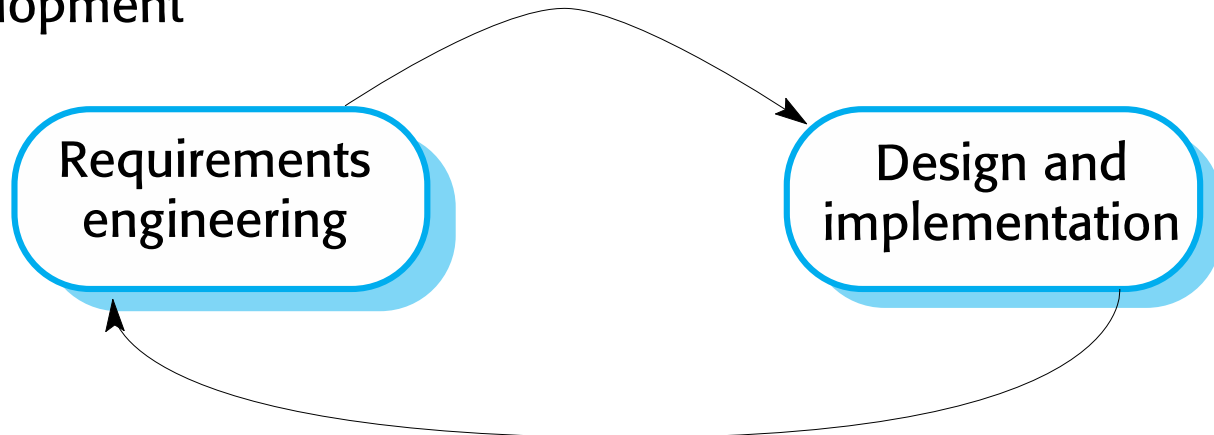
- **Plan-driven development**
  - A plan-driven approach to software engineering is **based around separate development stages with the outputs to be produced at each of these stages planned in advance.**
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.
- **Agile development**
  - **Agile is a set of values and principles**
  - Specification, design, implementation and testing are **inter-leaved and the outputs from the development process** are decided through a process of negotiation during the software development process.
  - **Build Short... Build Often ...** build short and small but times to time
  - E.g. Tally, Microsoft Office ... **has different versions after taking reviews**

# Plan-driven and agile specification

## Plan-based development



## Agile development



# Agile methods

---

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - **Focus on the code** rather than the design
  - Can **go back** to previous stage and make changes, product is **tested in every stages**
  - Are based on an **iterative approach or incremental approach** to software development, takes account a **client's evolving needs**
  - Are intended to **deliver working software** quickly and evolve this quickly to **meet changing requirements**.
- The aim of agile methods is to **reduce overheads in the software process** (e.g. by limiting documentation) and to be able to **respond quickly to changing requirements** without excessive rework.

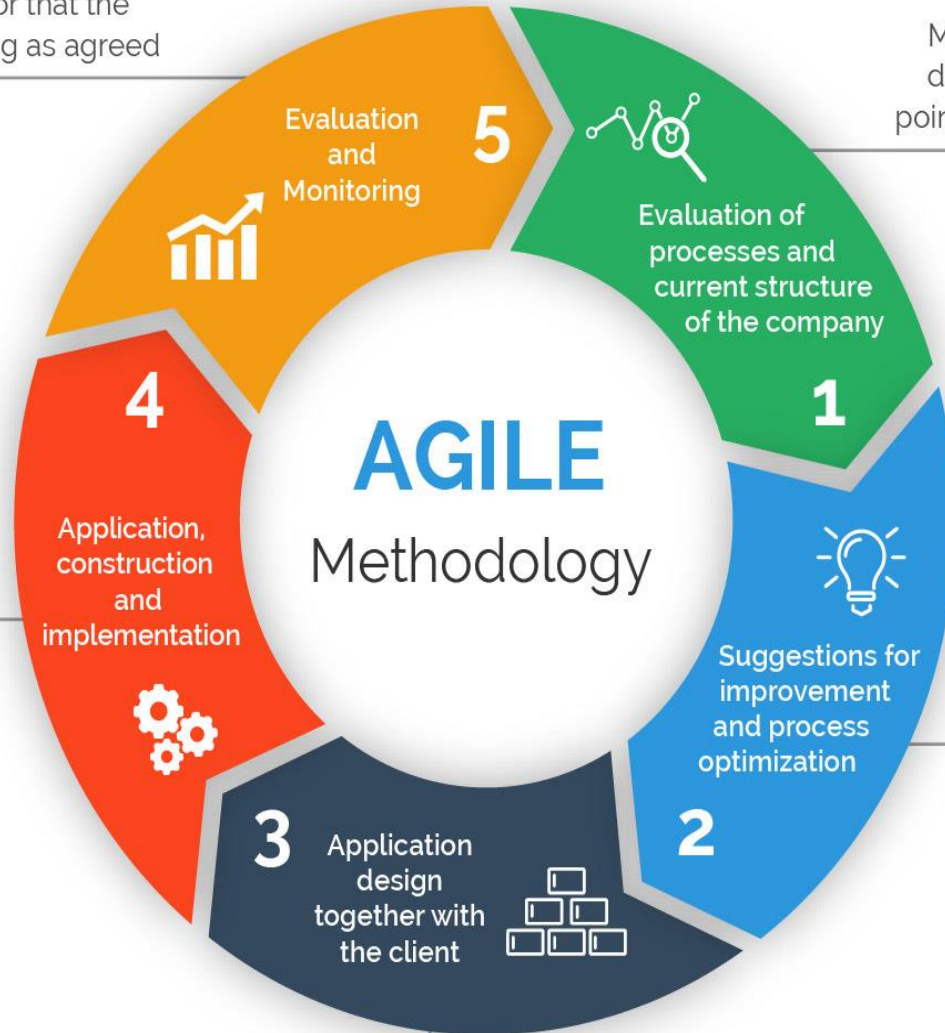
# Agile strategy

---

- 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.

These determine KPIs, reports and other metrics, per user, in order to monitor that the process is working as agreed

Mapping processes to determine the starting point and the "Actual State"



Weekly delivery of developments for the end user to operate and request for changes. It is IDEAL that the end user is part of the process of development and testing

Optimize a process with a right combination of technologies

we choose technologies and review options with the client. The client is part of the project from the very beginning and feedback is VITAL

# The 12 principles of agile

---

1. Highest priority is to **satisfy the customer** through early and continuous delivery of valuable software
2. **Accept changing requirements**, even delay in development
3. Deliver working software frequently from **a couple of weeks to months** with preference to the shorter timescale
4. Business people and developers **must work together or coordination** each other with daily throughout the project.

# The 12 principles of agile... contd

---

5. Builds the project around **motivated individuals**. Give them the environment and support they need and trust them to get the job done.

6. The most efficient and effective method of conveying information **to and within a development team is face-to-face conversation**.

7. **Working software is the primary measure of progress**, should work in any platform.

8. Agile process **promote sustainable development**. The sponsors, developers, and users **should know easily how to use, when to use**, so that, able to maintain easily and quickly.

# The 12 principles of agile...contd

---

9. Continuous attention to technical excellence and good design enhance liveliness and know how to use?
10. **Simplicity**- the art of maximizing the amount of work done for implementation and make popular.
11. The best **architectures, requirements, and designs emerge** from self-organizing teams.
12. At regular intervals, the team **reflects on how to become more effective and easily use**, then tunes and adjusts its behaviour accordingly.

# Advantages and Disadvantages of Agile Methodology

---

## Advantages:-

- Allows for changes to be made **after planning and rewrite to the program** as requirement change
- **Easier to add features** that will keep us up to date with the latest developments in industry
- Allows clients to add their **feedback**

## Disadvantages:-

- Project is likely to come in **late and over budget**
- **As initial project, does not** have a definite plan

# When should use Agile Methodology

---

- When rapid production is **more important than the quality of the product.**
- When clients will be **able to change the scope** of the project.
- When there **isn't a clear picture of what the final product** should look like.
- When have **skilled developers** who are adaptable and able to **think independently**
- When product is proposed for an industry with **rapidly changing standards.**

# Agile method applicability

---

- Product development where a **software company** is developing a **small or medium-sized product** for sale.
- 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 not a lot of external rules and regulations** that affect the software.
- Because of their focus on small, tightly-integrated teams, there are **problems in scaling agile methods to large systems**.

# Problems with agile methods

---

- It can be **difficult to keep the interest of customers** who are involved in the process.
- Team members may be **unsuited to the extreme involvement** that characterizes agile methods.
- **Prioritizing changes** can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires **extra work**.
- Contracts may be a **problem as with other approaches to iterative development**.

# Agile methods and software maintenance

---

- Most organizations **spend more on maintaining existing software** than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.
- Two key issues:
  - Are systems that are developed using an agile approach maintainable, given the emphasis in the **development process of minimizing formal documentation?**
  - Can agile methods be used effectively for evolving a system in **response to customer change requests?**
- **Problems may arise if original development team cannot be maintained.**

# Technical, human, organizational issues

- Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:
  - Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.
  - Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.
  - How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

# Technical, human, organizational issues

## – What type of system is being developed?

- Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements).

## – 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.

## – What technologies are available to support system development?

- Agile methods rely on good tools to keep track of an evolving design

## – How is the development team organized?

- If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.

# Technical, human, organizational issues

---

- Are there cultural or organizational issues that may affect the system development?
  - Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
- How good are the designers and programmers in the development team?
  - It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code
- Is the system subject to external regulation?
  - If a system has to be approved by an external regulator (e.g. the FAA approve software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.

# Extreme programming

Agile Method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement and customer participation

# XP Strength

---

A particular strength of extreme programming is the **development of automated tests before a program feature is created**. All tests must successfully execute when an increment is integrated into a system.

# XP (Extreme programming)

---

- Perhaps the best-known and most widely used agile method.
- Extreme Programming (XP) takes an **'extreme' approach to iterative development.**
  - **New versions** may be built several times per day;
  - **Increments** are delivered to customers every 2 weeks;
  - All tests must be run for **every build and the build is only accepted** if tests run successfully.

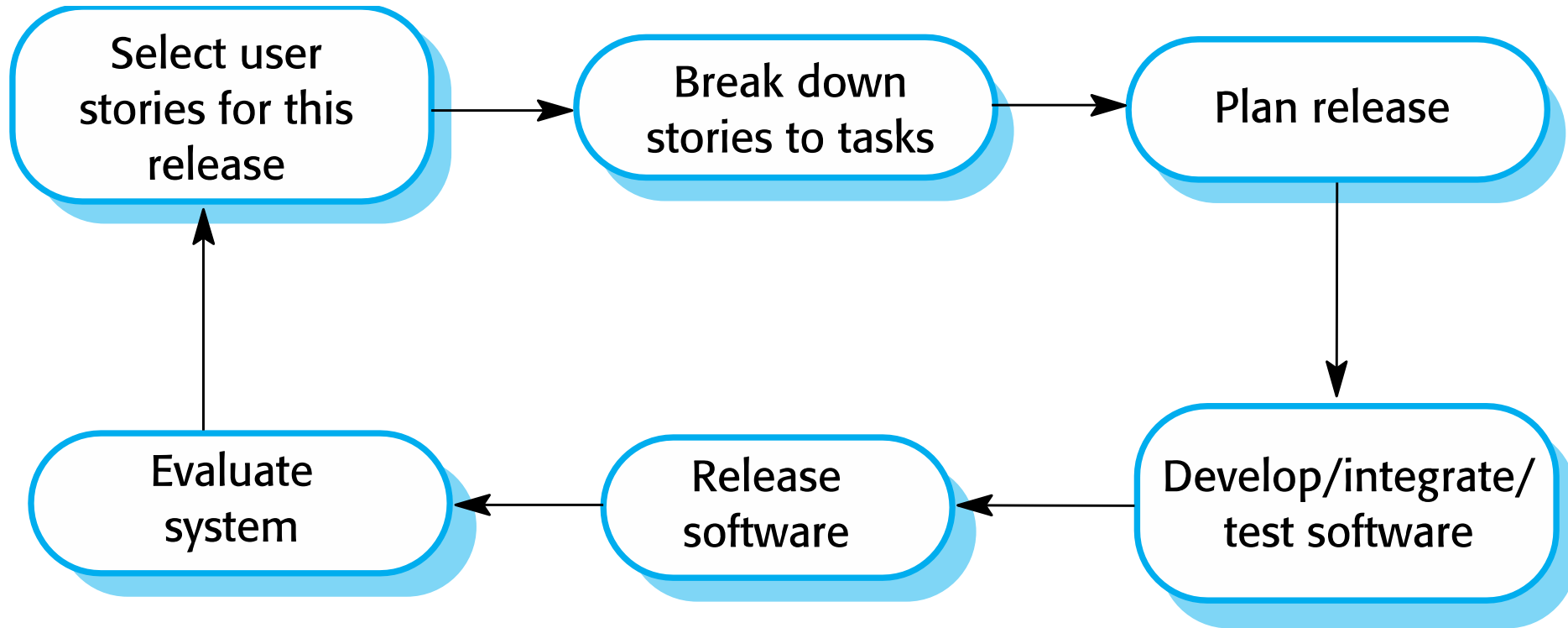
# XP and agile principles

---

- **Incremental development** is supported through small, frequent system releases.
- **Customer involvement** means full-time customer engagement with the team.
- **People not process through pair programming**, collective ownership and a process that avoids long working hours.
- Change supported through **regular system releases**.
- Maintaining simplicity through **constant refactoring** of code.

# The extreme programming release cycle

---



# Extreme programming practices (a)

---

Principle practice	or	Description
Incremental planning		Requirements are recorded on story cards and the stories to be included in a <b>release are determined by the time available and their relative priority</b> . The developers break these stories into development 'Tasks'.
Small releases		The minimal useful set of functionality that provides business value is developed first. <b>Releases of the system are frequent and incrementally add functionality</b> to the first release.
Simple design		Enough design is carried out to meet the current requirements and no more.
Test-first development		<b>An automated unit test framework</b> is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring		All developers are expected to <b>refactor the code continuously as soon as possible code improvements are found</b> . This keeps the <b>code simple and maintainable</b> .

# Extreme programming practices (b)

Pair programming	<b>Developers work in pairs</b> , checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that <b>no islands of expertise develop and all the developers take responsibility for all of the code</b> . 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, <b>all the unit tests in the system must pass</b> .
Sustainable step	Large amounts of overtime are not considered acceptable as the <b>net effect is often to reduce code quality and medium term productivity</b>
On-site customer	A representative of the <b>end-user of the system (the customer) should be available full time for the use of the XP team</b> . 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.

# Requirements scenarios

---

- In XP, a customer or user is part of the XP team and is **responsible for making decisions on requirements.**
- User requirements are **expressed as scenarios or user stories.**
- These are **written on cards and the development team break them down into implementation tasks.** These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the **next release based on their priorities and the schedule estimates.**

# A 'prescribing medication' story

## Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

# Examples of task cards for prescribing medication

---

## **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, lookup 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.

# XP and change

---

- Conventional wisdom in software engineering is to design for change. **It is worth spending time and effort anticipating changes as this reduces costs later** in the life cycle.
- XP, however, maintains that this is not valuable as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes **easier when they have to be implemented.**

# Refactoring

---

- Programming team look for possible software improvements and **make these improvements even where they are not available immediate.**
- This **improves the understandability** of the software and so **reduces** the need for **documentation.**
- **Changes are easier** to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is **much more expensive.**

# Examples of refactoring

---

- **Re-organization** of a class hierarchy to remove duplicate code.
- **Cleanup and renaming attributes** and methods to make them easier to understand.
- The **replacement of inline code** with calls to methods that have been included in a program library.

# Key points

---

- Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.
- The decision on whether to use an agile or a plan-driven approach to development should depend on the type of software being developed, the capabilities of the development team and the culture of the company developing the system.
- Extreme programming is a well-known agile method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement and customer participation in the development team.

# Testing in XP

---

- **Testing is central to XP** and XP has developed an approach where the program **is tested after every change has been made.**
- XP testing features:
  - **Test-first** development.
  - **Incremental test** development from scenarios.
  - **User involvement** in test development and validation.
  - **Automated test harnesses** are used to run all component tests each time that a new release is built.

# Test-first development

---

- **Writing tests** before code clarifies the requirements to be implemented.
- **Tests are written as programs** rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as Junit.
- All previous and new tests are run automatically when new functionality is added, thus **checking that the new functionality has not introduced errors.**

# Customer involvement

---

- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

# Test case description for dose checking

---

## **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 automation

---

- Test automation means that **tests are written as executable components** before the task is implemented
  - 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 (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- 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.

# XP testing difficulties

---

- Programmers prefer programming to testing and sometimes they take short cuts 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 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.

# Pair programming

---

- In XP, **programmers work in pairs, sitting together** to develop code.
- This helps **develop common ownership of code and spreads knowledge** across the team.
- It serves as an **informal review process** as each line of **code** is looked at by more than 1 person.
- It **encourages refactoring** as the whole team can benefit from this.
- Measurements suggest that **development productivity** with pair programming is similar to that of two people working independently.

# Pair programming

---

- In pair programming, **programmers sit together** at the same workstation to develop the software.
- **Pairs are created dynamically** so that all team members work with each other during the development process.
- 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.
- Pair programming is not necessarily inefficient and there is evidence that a **pair working together is more efficient than 2 programmers** working separately.

# Advantages of pair programming

---

- It supports the **idea of collective ownership and responsibility for the system**.
  - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
- It acts as an **informal review process** because each line of code is looked at by at least two people.
- It helps **support refactoring**, which is a process of software improvement.
  - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

# Agile project management

---

- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- Agile project management requires a different approach, which is adapted to incremental development and the particular strengths of agile methods.

# Scaling agile methods

---

- Agile methods have proved to be successful for **small and medium sized projects** that can be developed **by a small co-located team**.
- It is sometimes argued that the success of these methods comes **because of improved communications** which is possible when everyone is working together.
- **Scaling up agile methods involves changing these to cope with larger, longer projects** where there are multiple development teams, perhaps working in different locations.

# Large systems development

---

- Large systems are usually **collections of separate, communicating systems, where separate teams develop each system.** Frequently, these teams are working in different places, sometimes in different time zones.
- Large systems are ‘brownfield systems’, that is they **include and interact with a number of existing systems.** Many of the system requirements are concerned with this interaction and so don’t really lend themselves to flexibility and incremental development.
- 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.

# Large system development

---

- Large systems and their development processes are often **constrained by external rules and regulations limiting** the way that they can be developed.
- 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**. It is practically impossible to involve all of these different stakeholders in the development process.

# Scaling out and scaling up

---

- ‘Scaling up’ is concerned with using agile methods for developing large software systems that cannot be developed by a small team.
- ‘Scaling out’ is concerned with how agile methods can be introduced across a large organization with many years of software development experience.
- When scaling agile methods it is essential to maintain agile fundamentals
  - Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

# Scaling up to large systems

---

- For large systems development, **it is not possible to focus only on the code of the system**. You need to do more up-front design and system documentation
- **Cross-team communication mechanisms have to be designed and used**. This should involve regular phone and video conferences between team members and frequent, short electronic meetings where teams update each other on progress.
- **Continuous integration**, where the whole system is built every time any developer checks in a change, is practically impossible. However, it is essential to **maintain frequent system builds and regular releases** of the system.

# Scaling out to large companies

---

- Project managers who do not have experience of agile methods may be **reluctant to accept the risk of a new approach.**
- 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. However, within large organizations, there are likely to be a **wide range of skills and abilities.**
- There may be **cultural resistance** to agile methods, especially in those organizations that have **a long history of using conventional systems** engineering processes.

# Key points

---

- A particular strength of extreme programming is the **development of automated tests before a program feature is created**. All tests must successfully execute when an increment is integrated into a system.
- The Scrum method is an agile method that provides a project management framework. It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Scaling agile methods for large systems is difficult. **Large systems need up-front design and some documentation.**