Chapter 3 – Agile Software Development

Agile Software Development: Agile methods, Plan driven and Agile Development, Introduction to Extreme Programming. Self Study: SCRUM

Topics covered

- Agile methods
- Plan-driven and agile development
- Extreme programming
- Agile project management
- Scaling agile methods

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**
  - **S**pecification, **d**esign and **i**mplementation 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.***

Agile methods

- ***Dissatisfaction with the overheads*** involved in ***software design methods*** of the 1980s and 1990s led to the creation of agile methods. These methods:
  - ***Focus on the code*** rather than the design
  - Are based on an ***iterative approach*** to software development
  - Are intended to ***deliver working software quickly*** and ***evolve this quickly*** to meet changing requirements.

- 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 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, whilethere is value in the items on the right, we value the items on the left more.*

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.***
Incremental delivery	The ***software is developed in increments*** with the ***customer specifying the requirements*** to be included in ***each increment.***
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.***
Embrace change	***Expect the system requirements to change*** and ***so design the system*** to accommodate these changes.
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.

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 intense involvement*** that characterises agile methods.

- ***Prioritising 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.

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**
  - **S**pecification, **d**esign, **i**mplementation and **t**esting are ***inter-leaved*** and the ***outputs from the development process*** are decided through ***a process of negotiation during the software development process.***

Plan-driven and agile specification

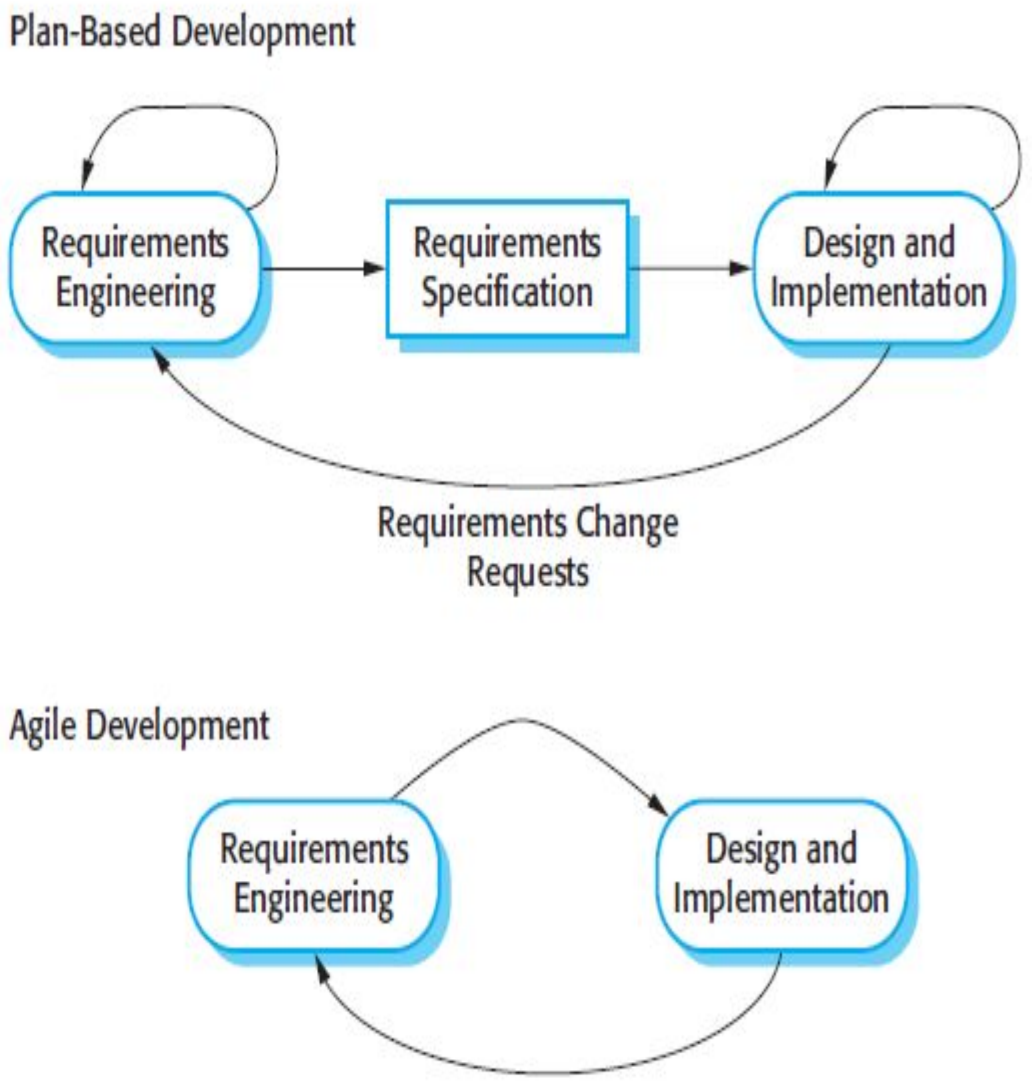

Plan-Based Development
Requirements Engineering
Requirements Specification
Design and Implementation
Requirements Change Requests
Agile Development
Requirements Engineering
Design and Implementation

Figure 3.2 Plan-driven and agile specification

Technical, human, organizational issues (10)

- **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 l***arger development teams*** so a ***plan-driven approach*** may have to be used.

Technical, human, organizational issues (cont..)

- **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 (cont..)

- **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***. (This usually requires extensive ***design documentation***, rather than the informal knowledge used in agile processes.)

- **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

- 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, are supported through ***pair programming***, ***collective ownership of the system code*** 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

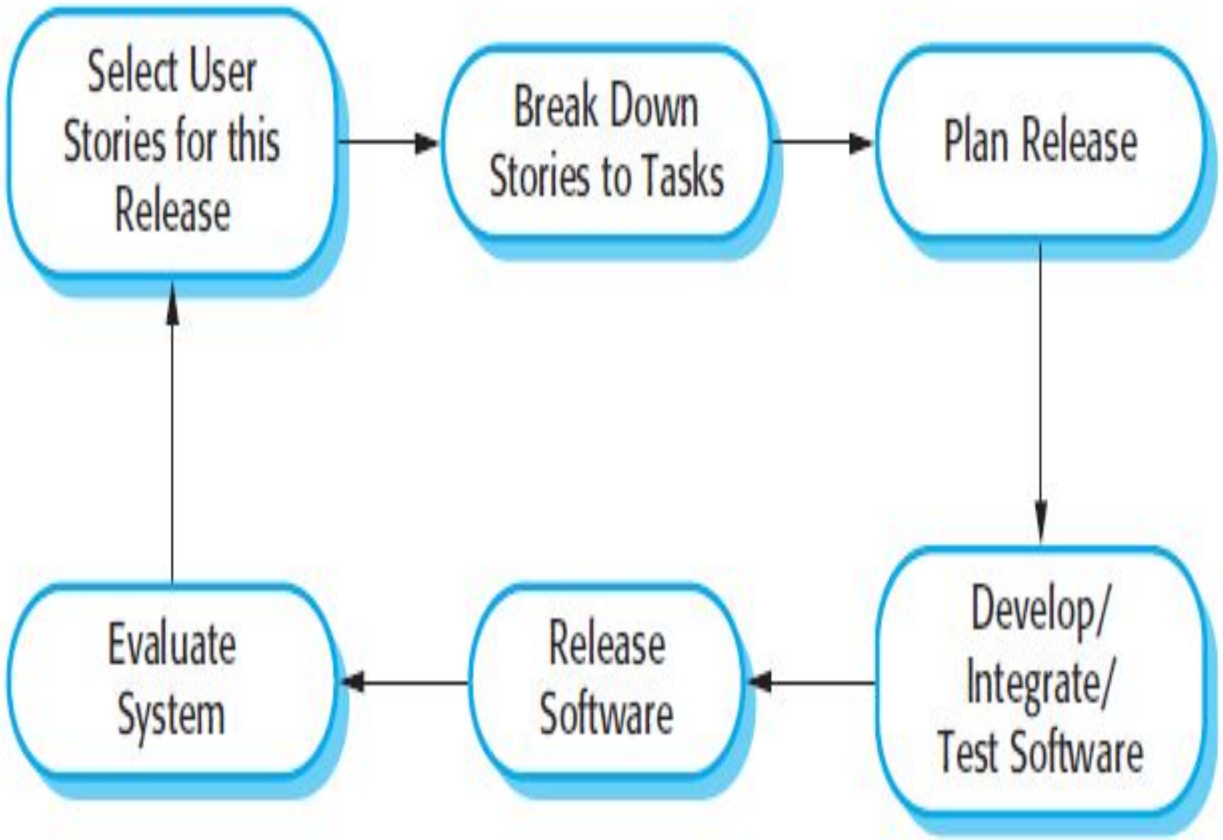

Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

Figure 3.3 The extreme programming release cycle

Extreme programming practices (a)

Principle or practice	**Description**
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'.***
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***.
Simple design	***Enough design is carried out to meet the current requirements*** and ***no more.***
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.***
Refactoring	All developers are expected to ***refactor the code continuously as soon as possible code improvements are found***. This keeps the code simple and maintainable.

Extreme programming practices (b)

Pair programming	***Developers work in pairs***, 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 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.***
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to ***reduce code quality and medium term productivity***
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.***

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.***

- In an XP process, customers are intimately involved in specifying and prioritizing system requirements.
- The requirements are not specified as lists of required system functions.
- Rather, the system customer is part of the development team and discusses scenarios with other team members. Together, they develop a ‘story card’ 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 mental health*** care patient management system is shown in Figure 3.5.
- This is a short description of a scenario for prescribing medication for a patient.

- Of course, 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.

- Extreme programming takes an ‘extreme’ approach to incremental development.
- New versions of the software may be built several times per day and releases are delivered to customers roughly every two weeks. Release deadlines are never slipped; if there are development problems, the customer is consulted and functionality is removed from the planned release.

- When a programmer builds the system to create a new version, he or she must run all existing automated tests as well as the tests for the new functionality.
- The new build of the software is accepted only if all tests execute successfully. This then becomes the basis for the next iteration of the system.

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 dose and 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 and then confirms the prescription.

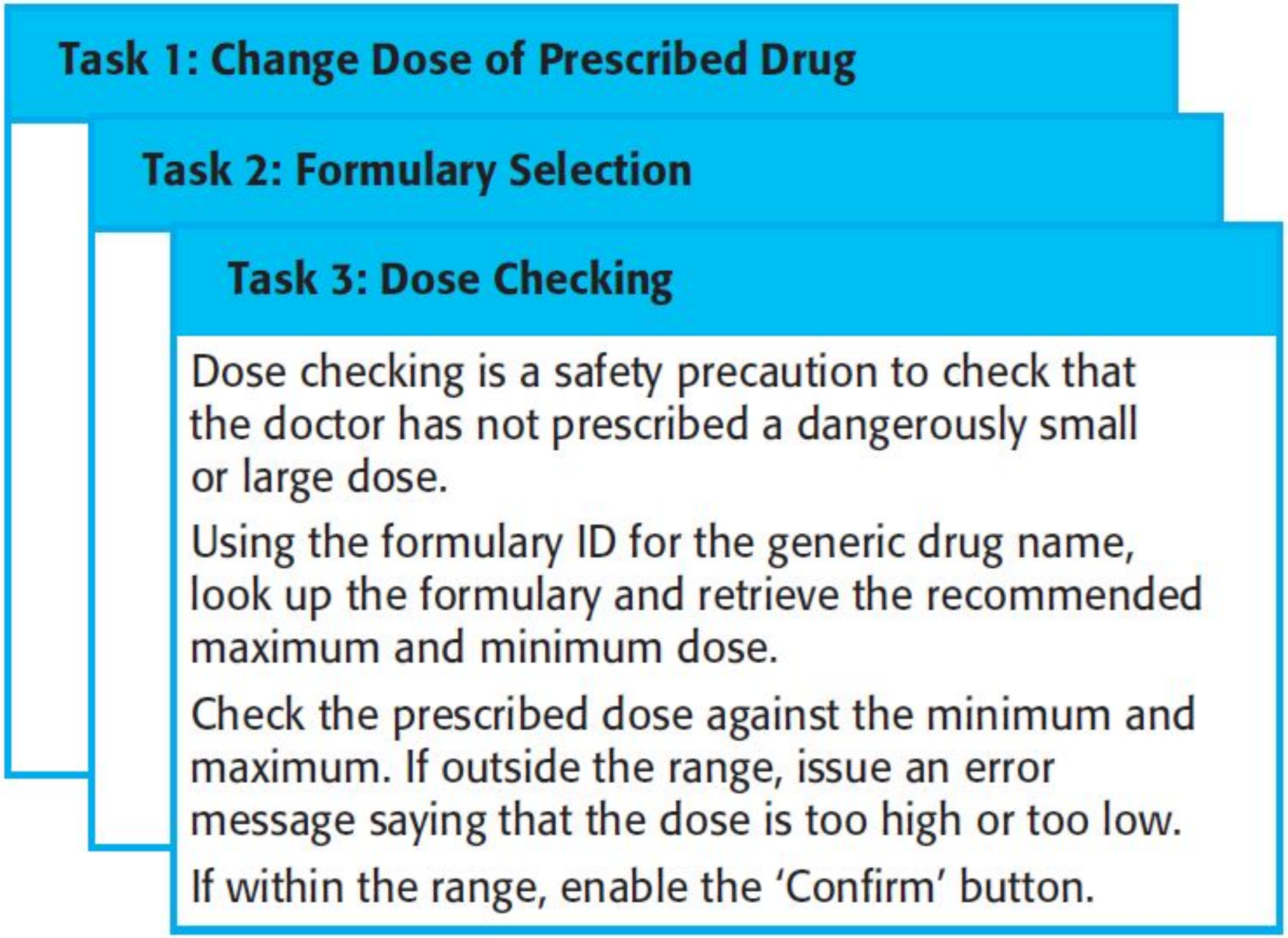
If she chooses ‘formulary’, the system displays a search box for the approved formulary. She can then search for the drug required. She selects a drug and is asked to check that the medication is correct. She enters the dose and then confirms the prescription.

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.

A ‘prescribing medication’ story

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, 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.

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 worthwhile** 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 there is no immediate need for them***.
- 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.***

- ***Tidying up*** and ***renaming attributes*** and ***methods*** to make them **easier to understand**.

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

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:

  1. **Test-first development.**

  2. **Incremental test development from scenarios.**

  3. **User involvement in test development and validation.**

  4. **the use of automated testing frameworks.**

     (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.

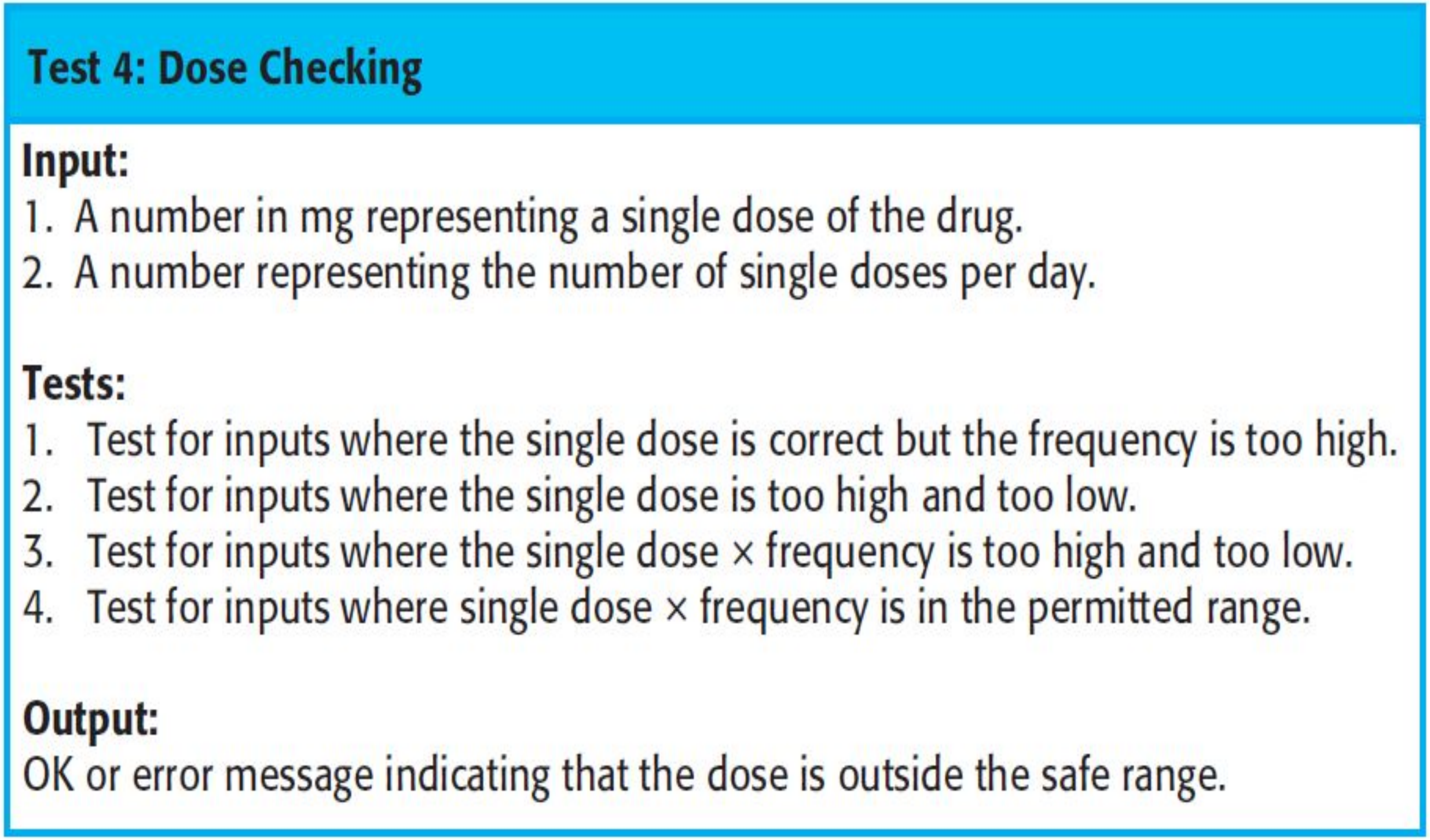
Customer involvement

- 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 (cont..)

- 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.

Review Questions

- With a neat diagram explain the flow of requirement model into the design model.
- What are the quality guidelines and quality attributes of software engineering design process.
- Give brief overview of important software design concepts that span both traditional and object-oriented software development.
- Explain Context model with an example.
- Define process model and design a process model (activity diagram) for MHC-PMS involuntary detention, Bank ATM, Library system.
- Develop a set of Use Cases that would serve as bases for understanding the requirement for an ATM system. Note: Actors : Bank, Customer, ATM
- Design a Use-case diagram and sequence diagram for ATM withdrawal system or library system.

- What are Agile methods? State Agile manifesto.
- List and explain the principles of Agile methods? Give its applicability and problems involved?
- With a neat diagram explain the difference between plan driven development and Agile Development.
- Briefly discuss the extreme programming release cycle with a neat diagram.
- Explain the practices followed in extreme programming
- Explain refactoring and give some examples.
- Discuss the advantages of pair programming
- With a neat diagram explain scrum process? Also discuss its benefits.