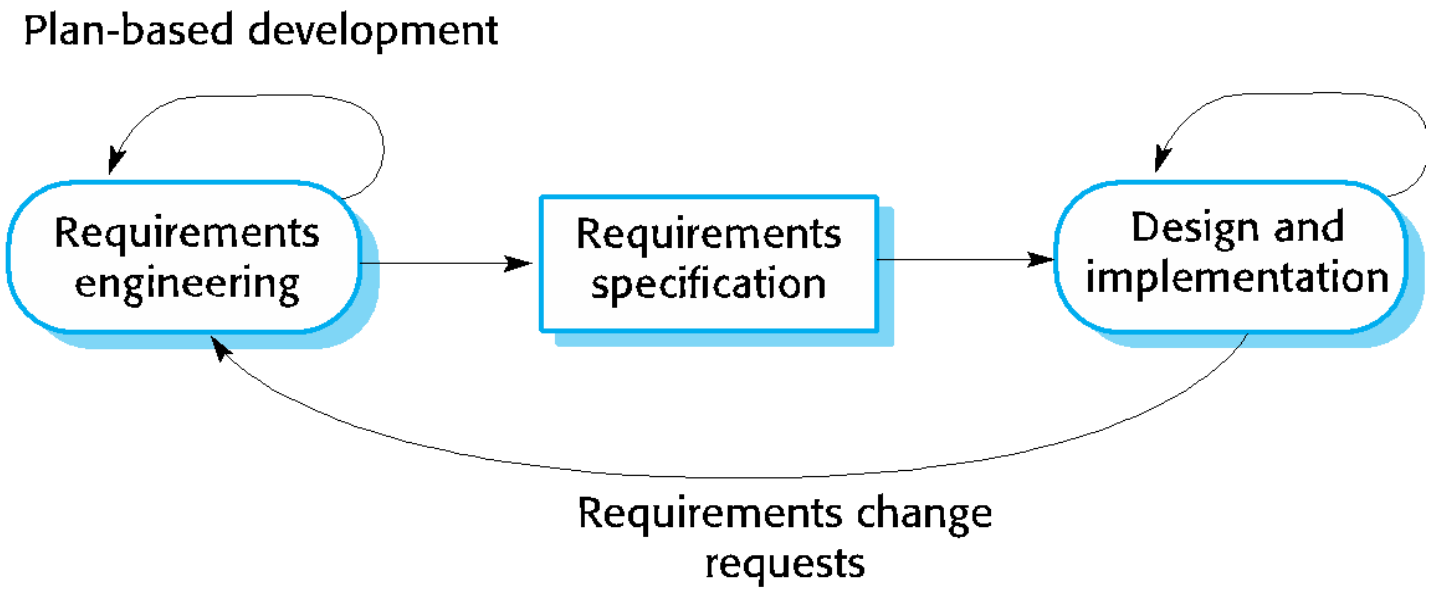
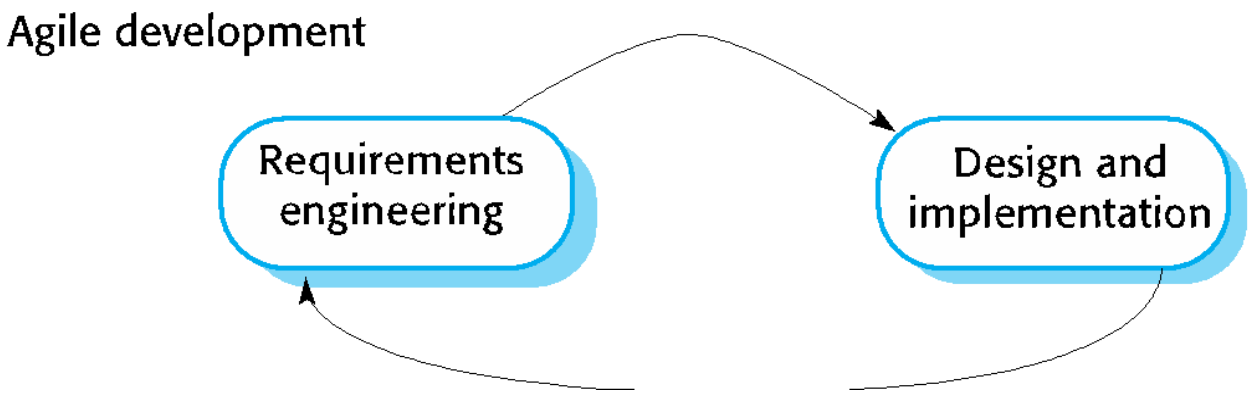
2. AGILE SOFTWARE DEVELOPMENT

Contents:

2.1 Plan-driven and agile development





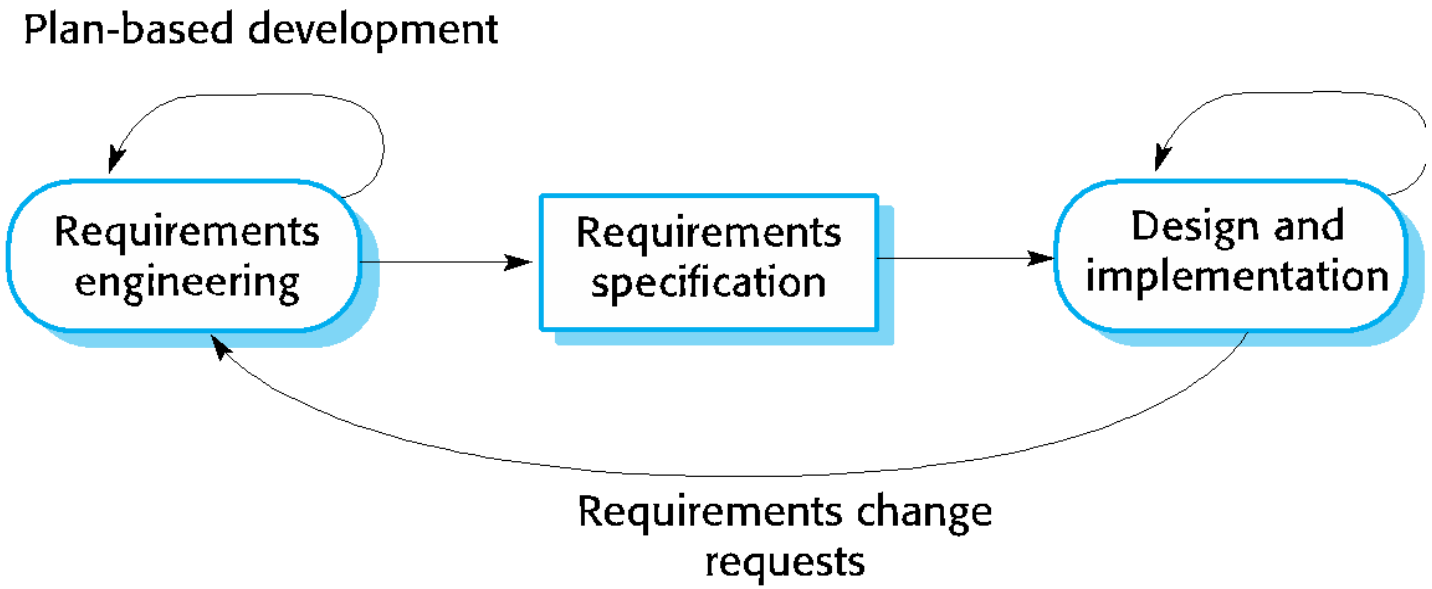
2.1 Plan-driven Development

- It is an approach to software development where the entire project is **planned in advance** and the **development process follows a linear and sequential path.**
- This approach is characterized by a **well-defined and rigid structure**, with distinct phases and activities that are carried out in a predetermined order.
- Each phase must be completed before moving on to the next, and changes to requirements or design are generally discouraged after the project has started.
- While plan-driven development has been widely used in the past, it has some limitations, such as inflexibility in accommodating changes during the development process and the potential for late discovery of issues.
- In contrast, more modern and iterative development methodologies, such as Agile, focus on flexibility, collaboration, and adaptability throughout the software development lifecycle.

Plan-driven 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.

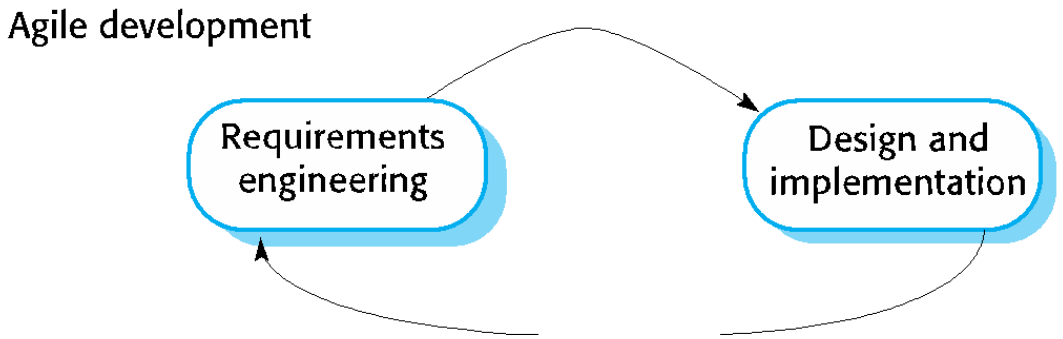


Rapid Software Development

- 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.
- Plan-driven development is essential for some types of system but does not meet these business needs.
- Agile development methods emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems

Agile Software Development

- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code



2.2 Agile methods

- Businesses are operating in a changing environment
- It is practically impossible to derive a complete set of stable software requirements.
- Requirements change because customers find it impossible to predict how a system will affect working practices, how it will interact with other systems, and what user operations should be automated.
- Plan-driven software development processes (like waterfall model) that completely specify the requirements and then design, build, and test a system are not geared to rapid software development.
- As the requirements change or as requirements problems are discovered, the system design or implementation has to be reworked and retested.

2.2 Agile methods

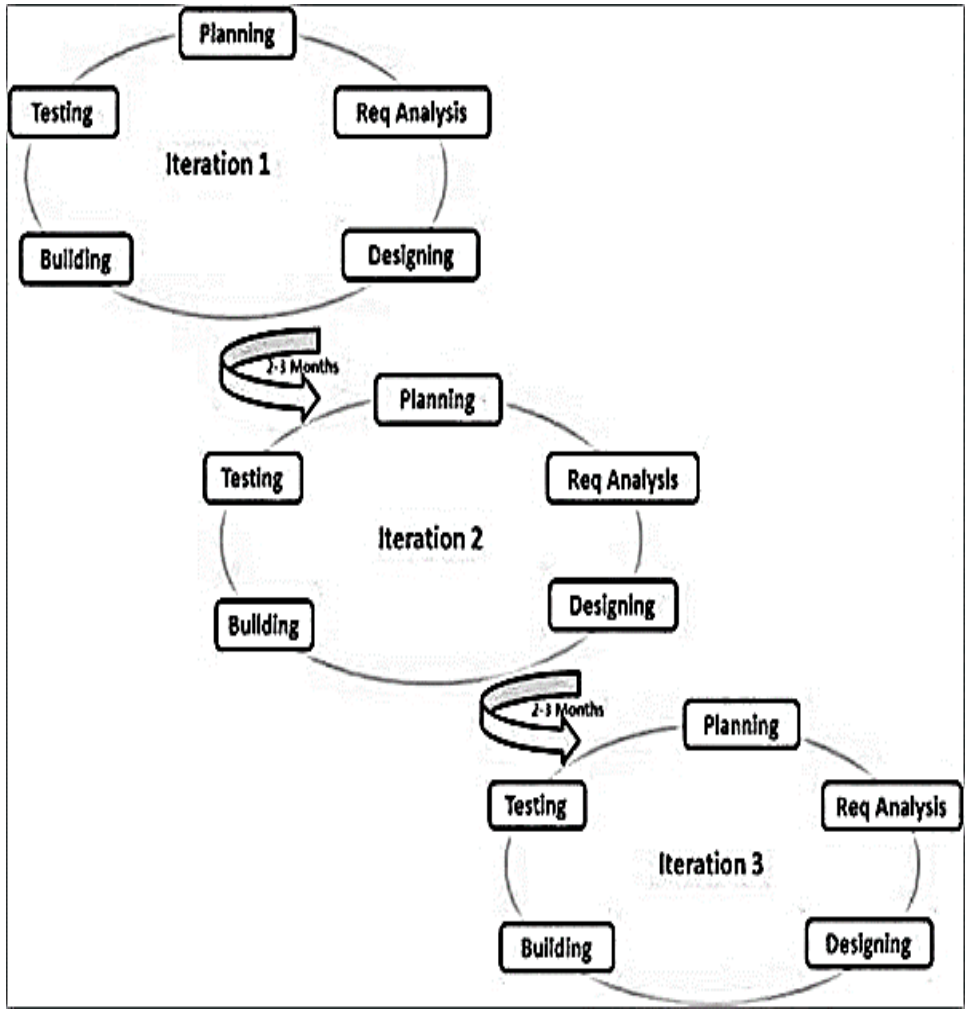
- *Agile methods are incremental development methods in which the increments are small, and, typically, new releases of the system are created and made available to customers every two or three weeks.*
- They **involve customers** in the development process to get rapid feedback on changing requirements.
- They **minimize documentation** by using informal communications rather than formal meetings with written documents.
- 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.

2.2 Agile methods

- 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.

2.2 Agile methods

- *Agile methods refer to a set of iterative and incremental software development approaches that prioritize flexibility, collaboration, and customer satisfaction.*
- These methods emerged as a response to the limitations of traditional, sequential project management methodologies, such as the Waterfall model, which often led to rigid processes and difficulties in adapting to changing requirements.



Agile manifesto

- The Agile Manifesto is a set of guiding values and principles for Agile software development, created by a group of software developers in 2001.
- The manifesto prioritizes flexibility, collaboration, and responsiveness to change.
- Here are the **four key values** and **12 principles** of the Agile Manifesto:

Four Key Values of Agile manifesto

1. **Individuals and Interactions over Processes and Tools:**
   - Emphasizes the importance of people and effective communication in the development process.
2. **Working Software over Comprehensive Documentation:**
   - Prioritizes delivering functional software over extensive documentation.
3. **Customer Collaboration over Contract Negotiation:**
   - Encourages close collaboration between developers and customers throughout the development process.
4. **Responding to Change over Following a Plan:**
   - Embraces the ability to adapt to changing requirements and circumstances over rigid planning.

Agile Principles

The Agile Alliance defines 12 agility principles for those who want to achieve agility:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects 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.

Agile Principles

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
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, then tunes and adjusts its behavior accordingly.

Not every agile process model applies these 12 principles with equal weight, and some models choose to ignore (or at least downplay) the importance of one or more of the principles. However, the principles define an *agile spirit* that is maintained in each of the process models.

Agile method applicability

- 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 rules and regulations that affect the software.

- Agile methods work well in these situations because it is possible to have continuous communications between the product manager or system customer and the development team.

Agile Development Techniques

1. **Extreme Programming (XP)**
2. **Scrum**
3. Adaptive Software Development
4. Dynamic System Development Method
5. Feature Driven Development
6. Lean Software Development

2.3 Extreme Programming (XP)

- is one of the most important software development framework of Agile models.
- It is used to improve software quality and **responsive to customer requirements.**
- The extreme programming model recommends taking the best practices that have worked well in the past in program development projects to extreme levels.

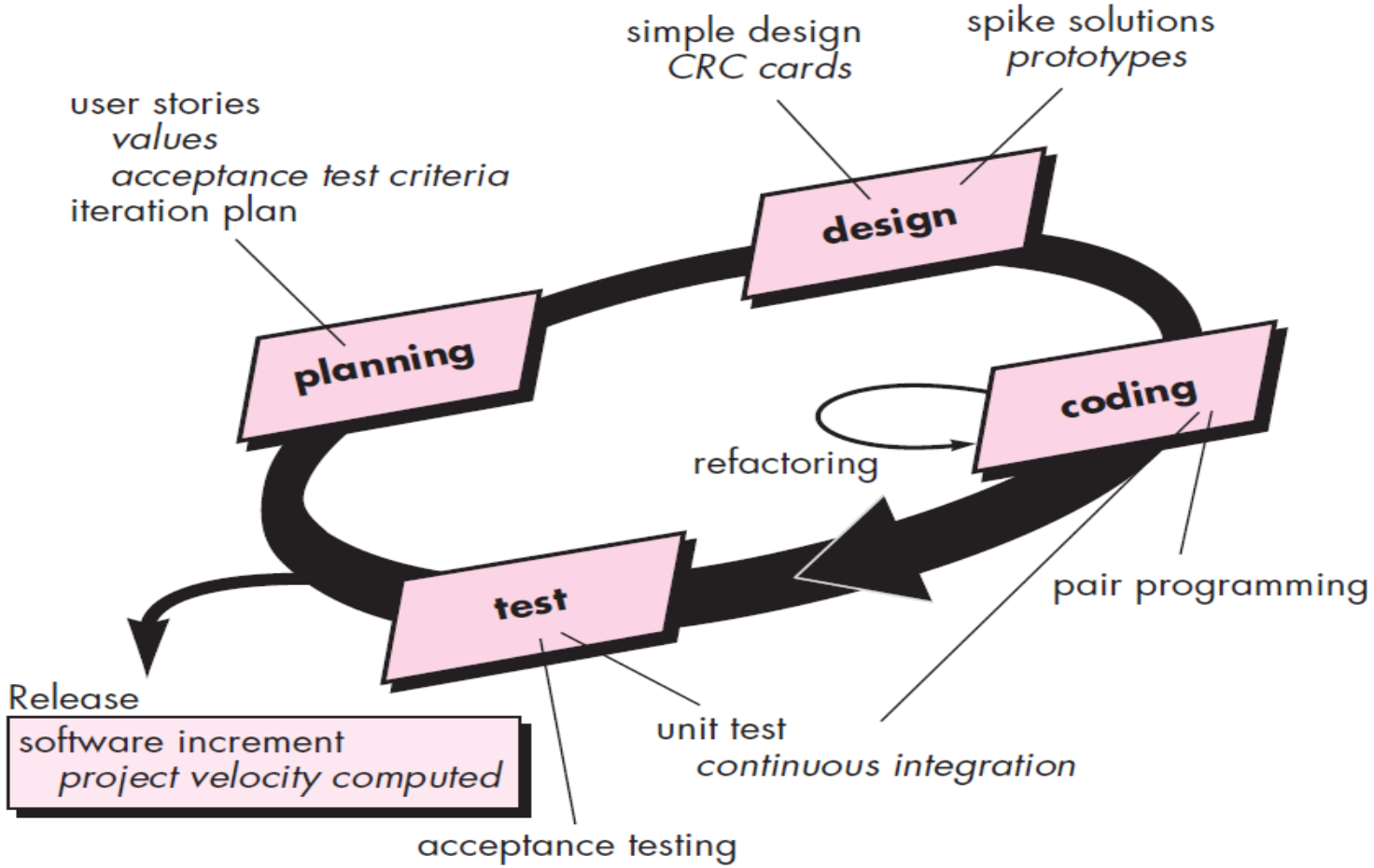


Fig. The Extreme Programming process

Extreme Programming (XP)

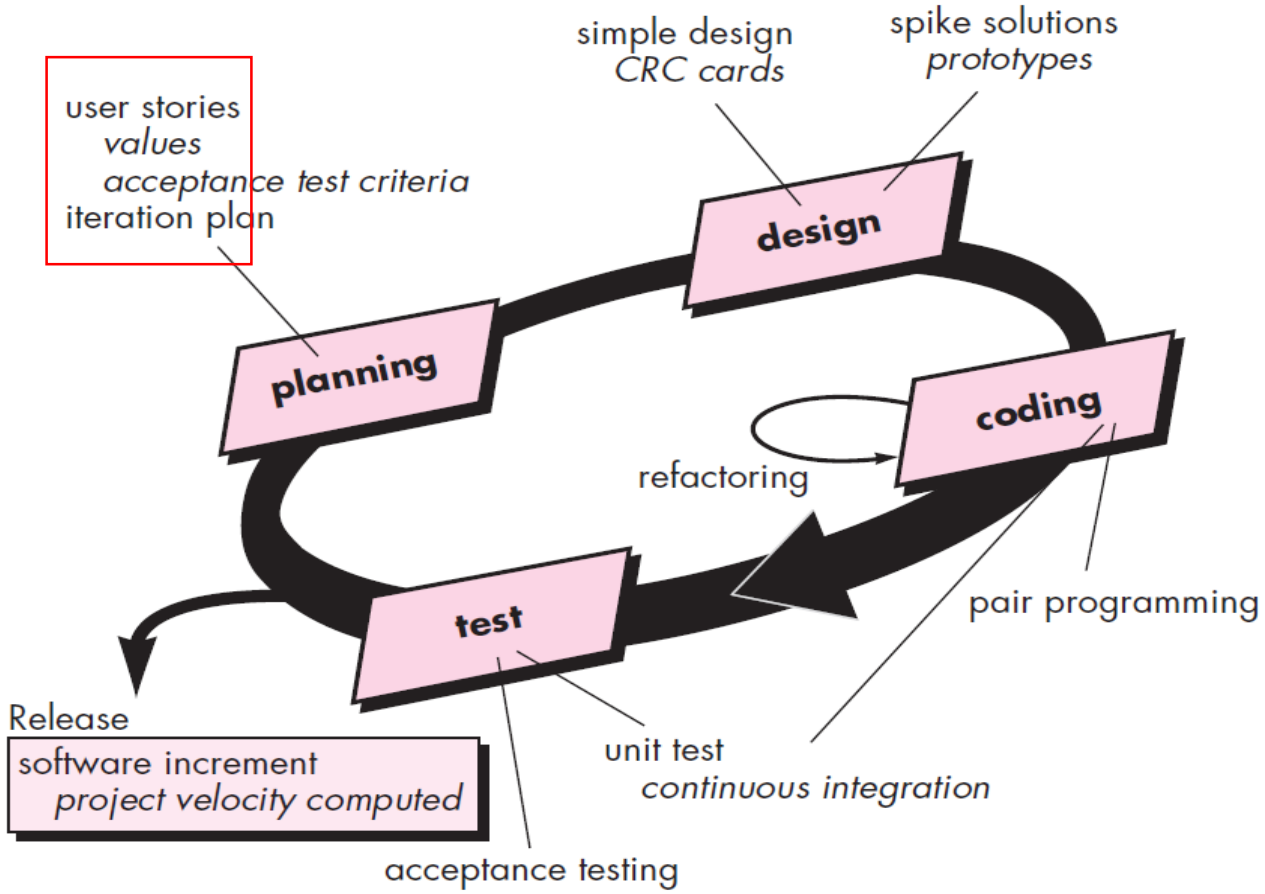


Fig. The Extreme Programming process

- XP is based on the frequent iteration through which the developers implement **User Stories.**
  - User stories are simple and informal statements of the customer about the functionalities needed.
  - A User story is a conventional description by the user about a feature of the required system.
  - It does not mention finer details such as the different scenarios that can occur.

Extreme Programming (XP)

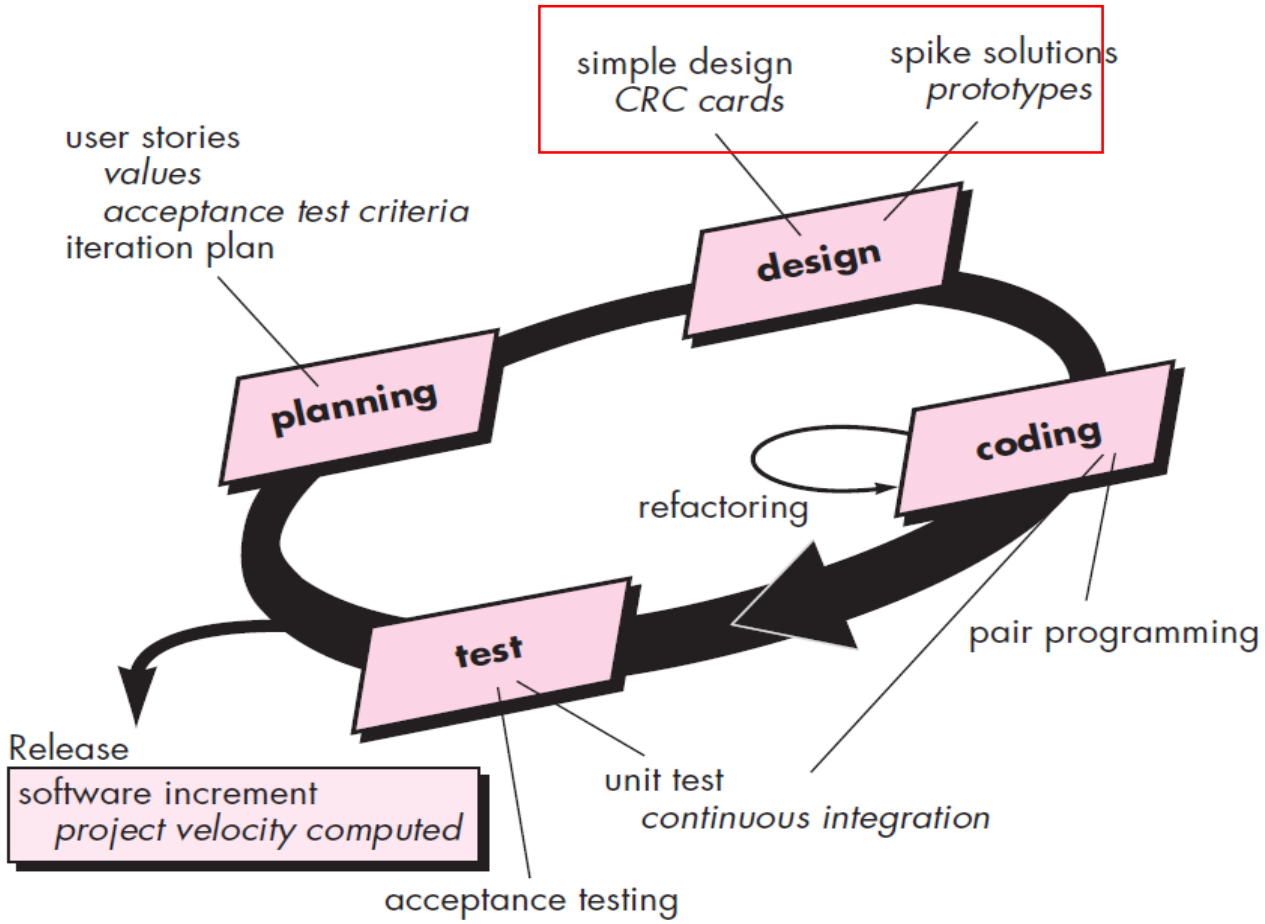


Fig. The Extreme Programming process

- On the basis of User stories, the project team proposes **Metaphors.**
  - Metaphors are a common vision of how the system would work.
- The development team may decide to build a **Spike** for some feature.
  - A Spike is a very simple program that is constructed to explore the suitability of a solution being proposed.
  - It can be considered similar to a prototype.

Extreme Programming (XP)

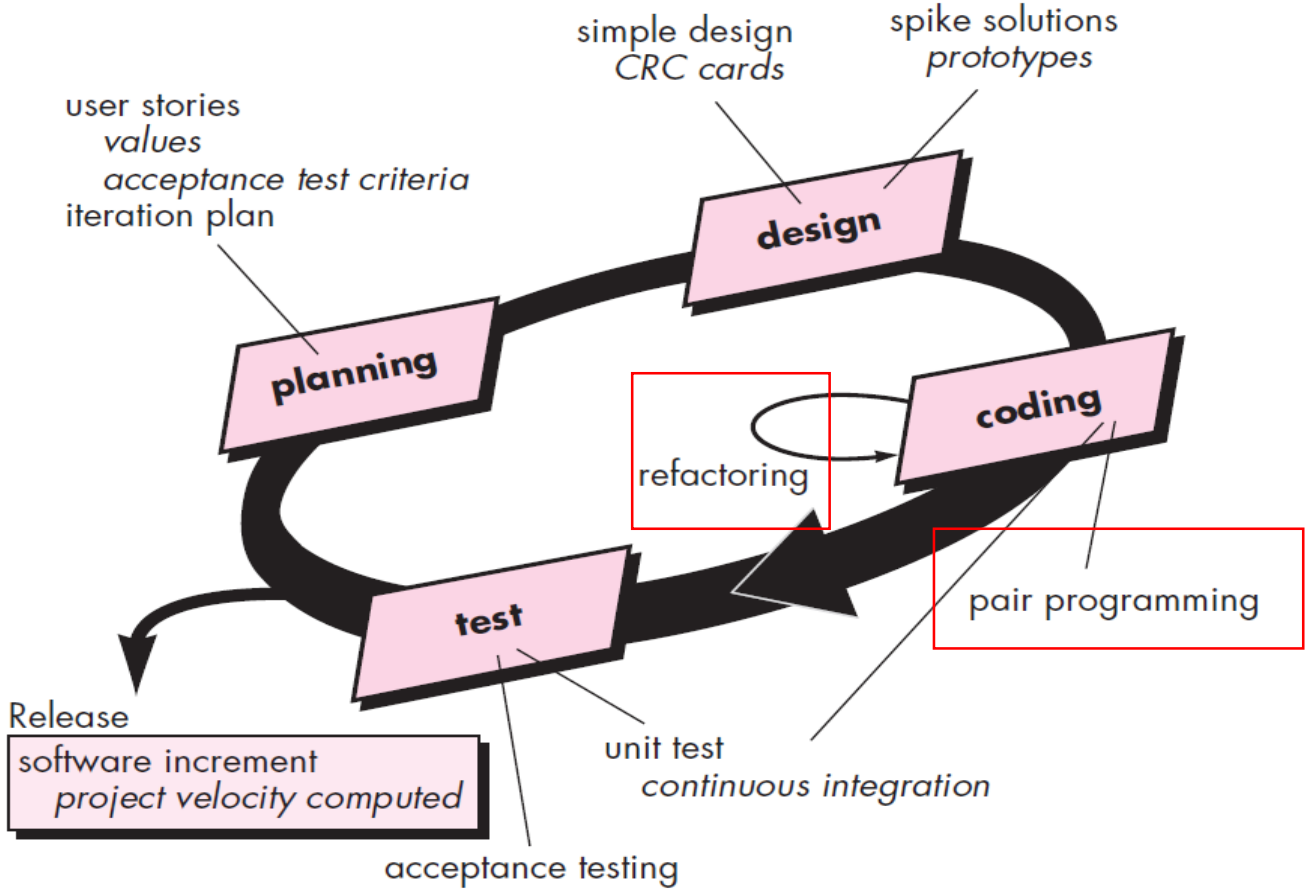


Fig. The Extreme Programming process

- A key concept of **"Pair Programming"** is done during the coding activity. Code review detects and corrects errors efficiently.
  - XP recommends two people work together at one computer workstation to provide better solution for the problem.
  - It suggests pair programming as coding and reviewing of written code carried out by a pair of programmers who switch their works between them every hour.
- **Refactoring is** always considered.
  - Refactoring is the technique of improving code without changing functionality.

Extreme Programming (XP)

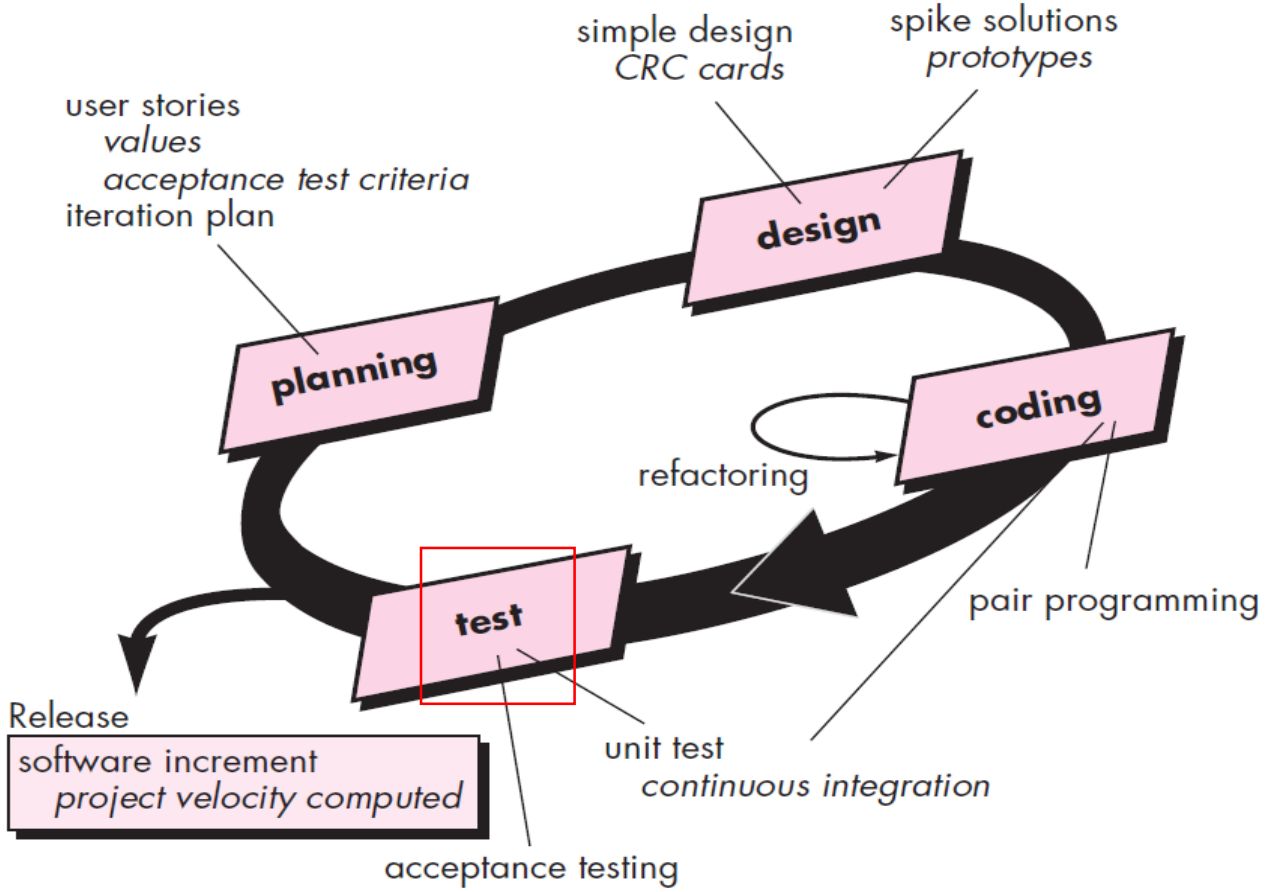
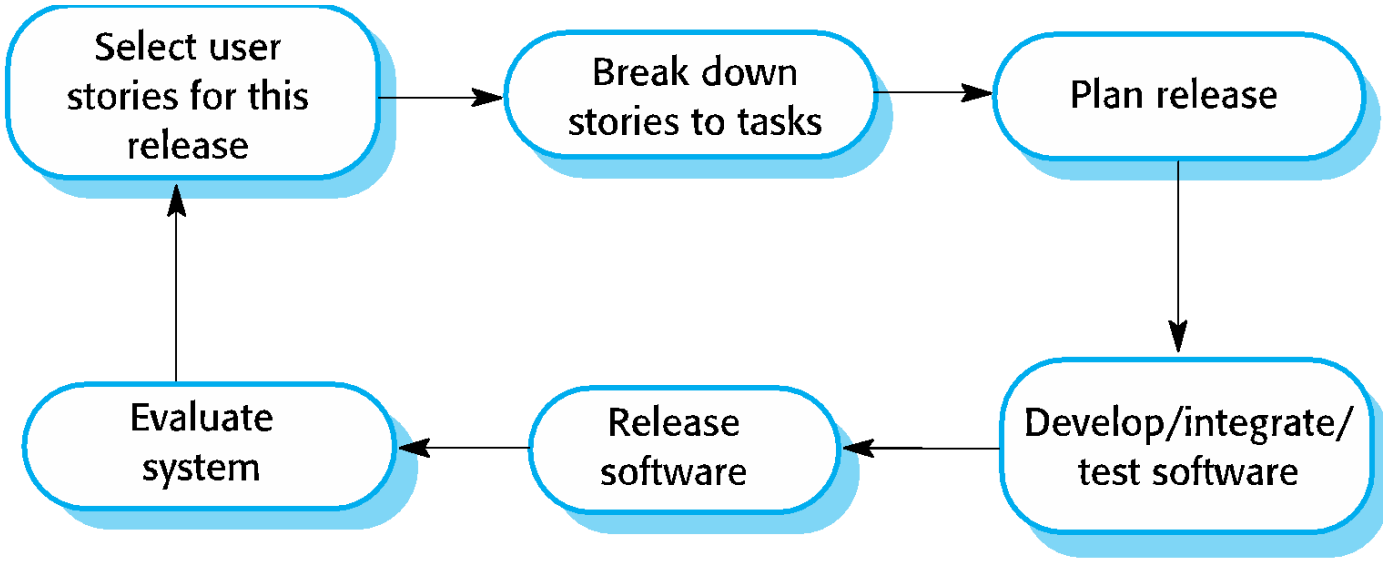


Fig. The Extreme Programming process

- XP model gives high importance on **testing** and considers it be the primary factor to develop a fault-free software.
  - XP suggests test-driven development (TDD) to continually write and execute test cases.
  - In the TDD approach test cases are written even before any code is written.
- **Incremental development** is very good because customer feedback is gained
- and based on this development team come up with new increments every few days after each iteration.

The extreme programming release cycle



Extreme programming practices

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.

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 for requirements

- 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. A

User stories for requirements

- 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 user stories or scenarios.
- 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.

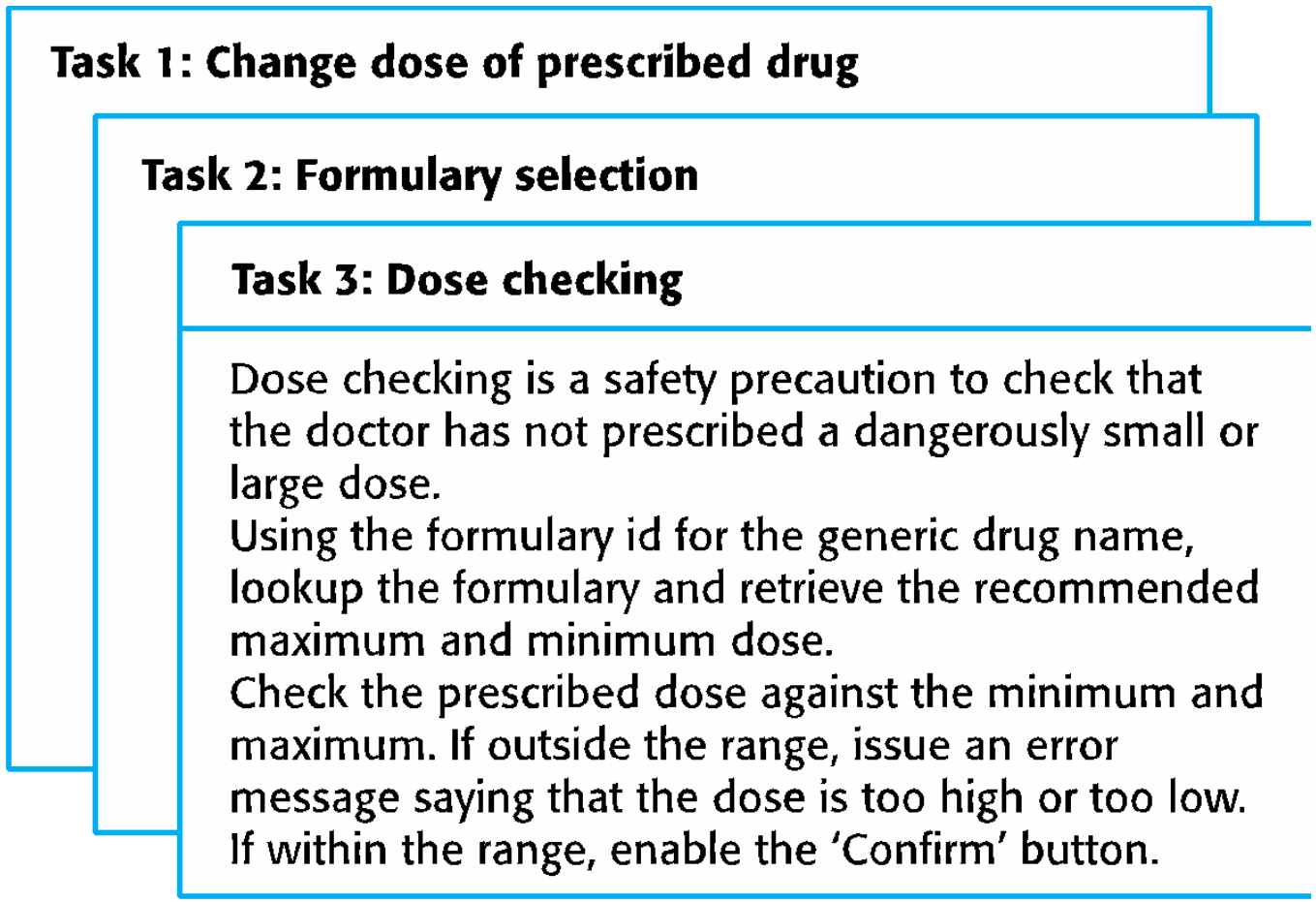
In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

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.

User stories for requirements

- User stories may be used in planning system iterations.
- Once the story cards have been developed, the development team breaks these down into tasks (Figure 3.6) and estimates the effort and resources required for implementing each task.
- This usually involves discussions with the customer to refine the requirements.
- The customer then prioritizes the stories for implementation, choosing those stories that can be used immediately to deliver useful business support.
- The 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.

Examples of task cards for prescribing medication



Refactoring

- 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.

Test-first development

- Testing is central to XP.
- 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 frameworks are used to run all component tests each time that a new release is built.

Test-driven 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.

Problems with test-first development

- 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

- Pair programming involves programmers working in pairs, developing code together.
- In pair programming, programmers sit together at the same computer 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 some evidence that suggests that a pair working together is more efficient than 2 programmers working separately.

Advantages of Pair Programming

1. It supports the idea of collective ownership and responsibility for the 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.

2.4 Agile project management

- In any software business, managers need to know what is going on and whether or not a project is likely to meet its objectives and deliver the software on time with the proposed budget.
- Plan-driven approaches to software development evolved to meet this need.
- 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.

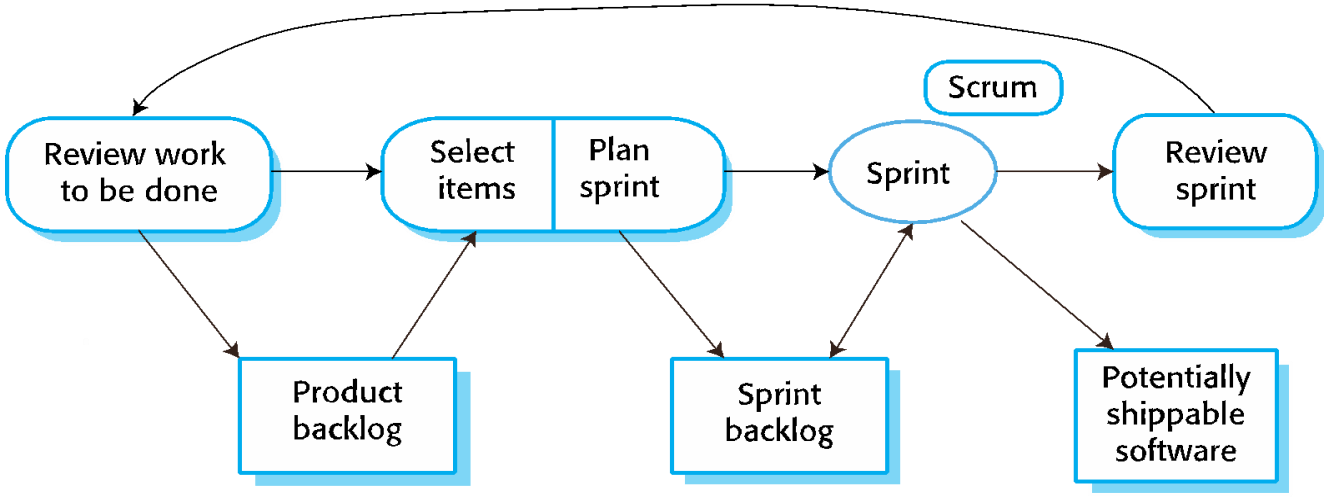
- The informal planning and project control that was proposed by the early adherents of agile methods clashed with this business requirement for visibility.
- Teams were self-organizing, did not produce documentation, and planned development in very short cycles.
- While this can and does work for small companies developing software products, it is inappropriate for larger companies who need to know what is going on in their organization.
- Like every other professional software development process, agile development has to be managed so that the best use is made of the time and resources available to the team.
- To address this issue, the Scrum agile method was developed.

2.5 Scrum

- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.
- There are three phases in Scrum.
  - The initial phase is an outline **planning phase** where you establish the general objectives for the project and design the software architecture.
  - This is followed by a **series of sprint cycles**, where each cycle develops an increment of the system.
  - The **project closure phase** wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

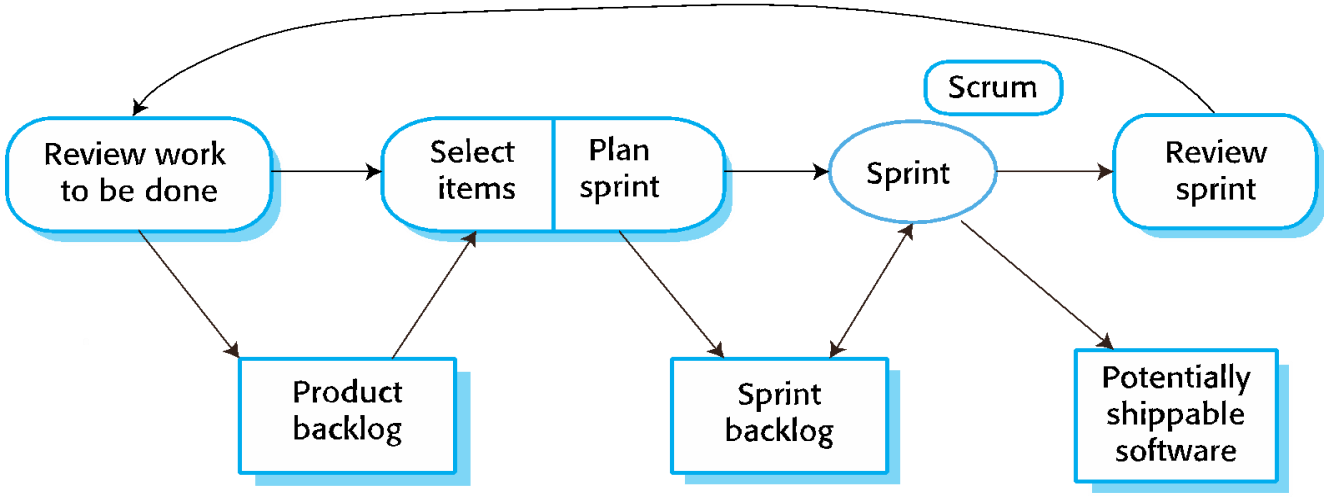
Scrum Sprint Cycle

- Sprints are fixed length, normally 2–4 weeks.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog to be developed during the sprint.



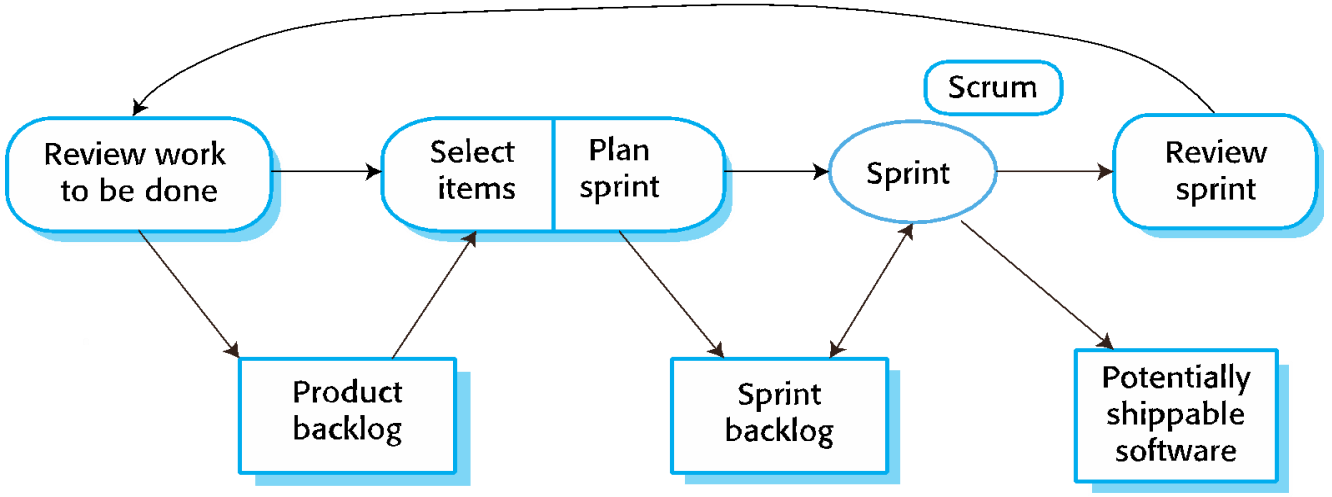
Scrum Sprint Cycle

- Once these are agreed, the team organize themselves to develop the software.
- During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called ‘Scrum master’.
- The role of the Scrum master is to protect the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.



Teamwork in Scrum

- The ‘Scrum master’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings (Scrums) where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
- This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.



3. Scrum

Scrum is the type of **Agile framework**. It is a framework within which people can address complex adaptive problem while productivity and creativity of delivering product is at highest possible values. Scrum uses **Iterative process**.

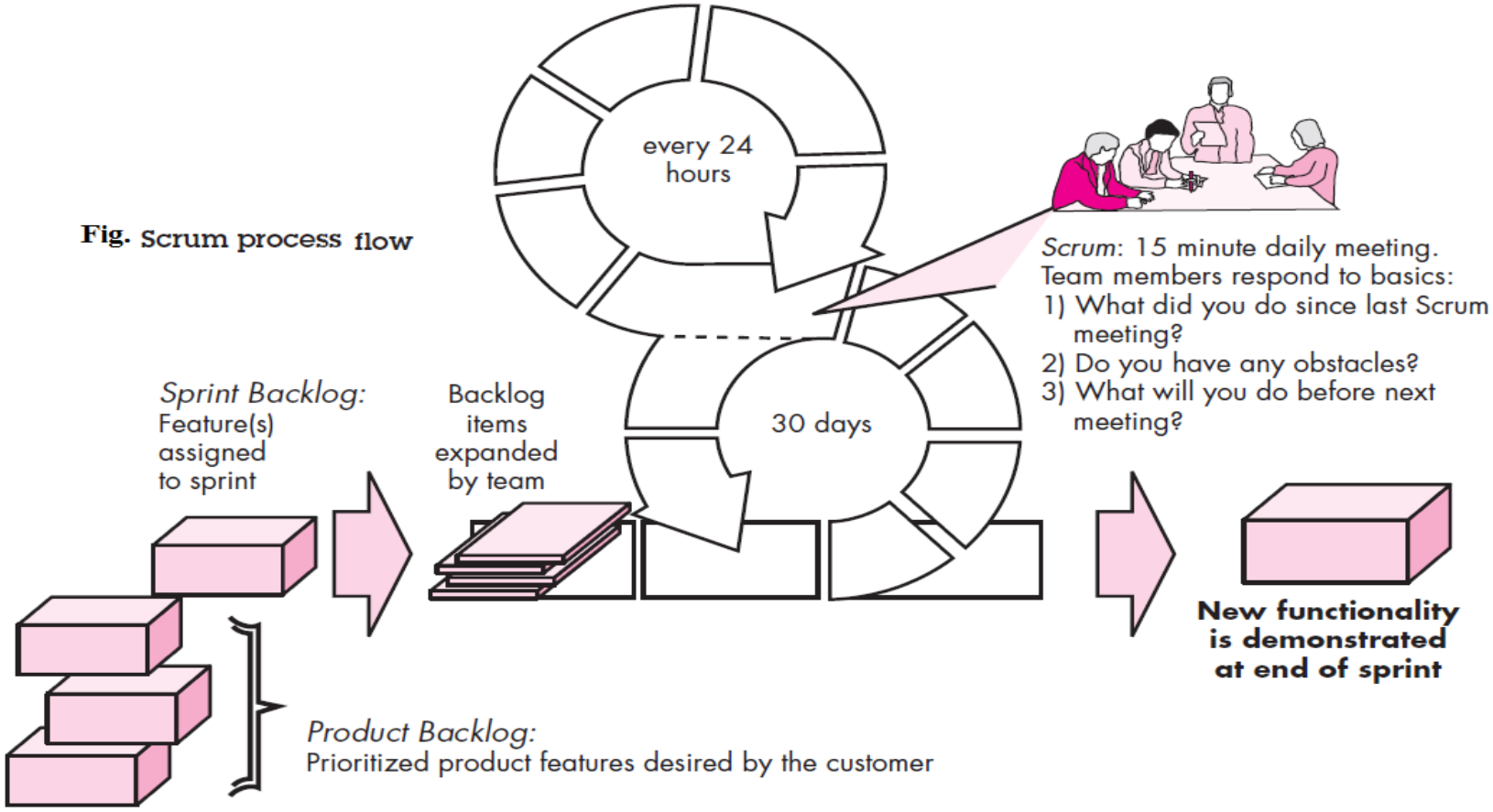
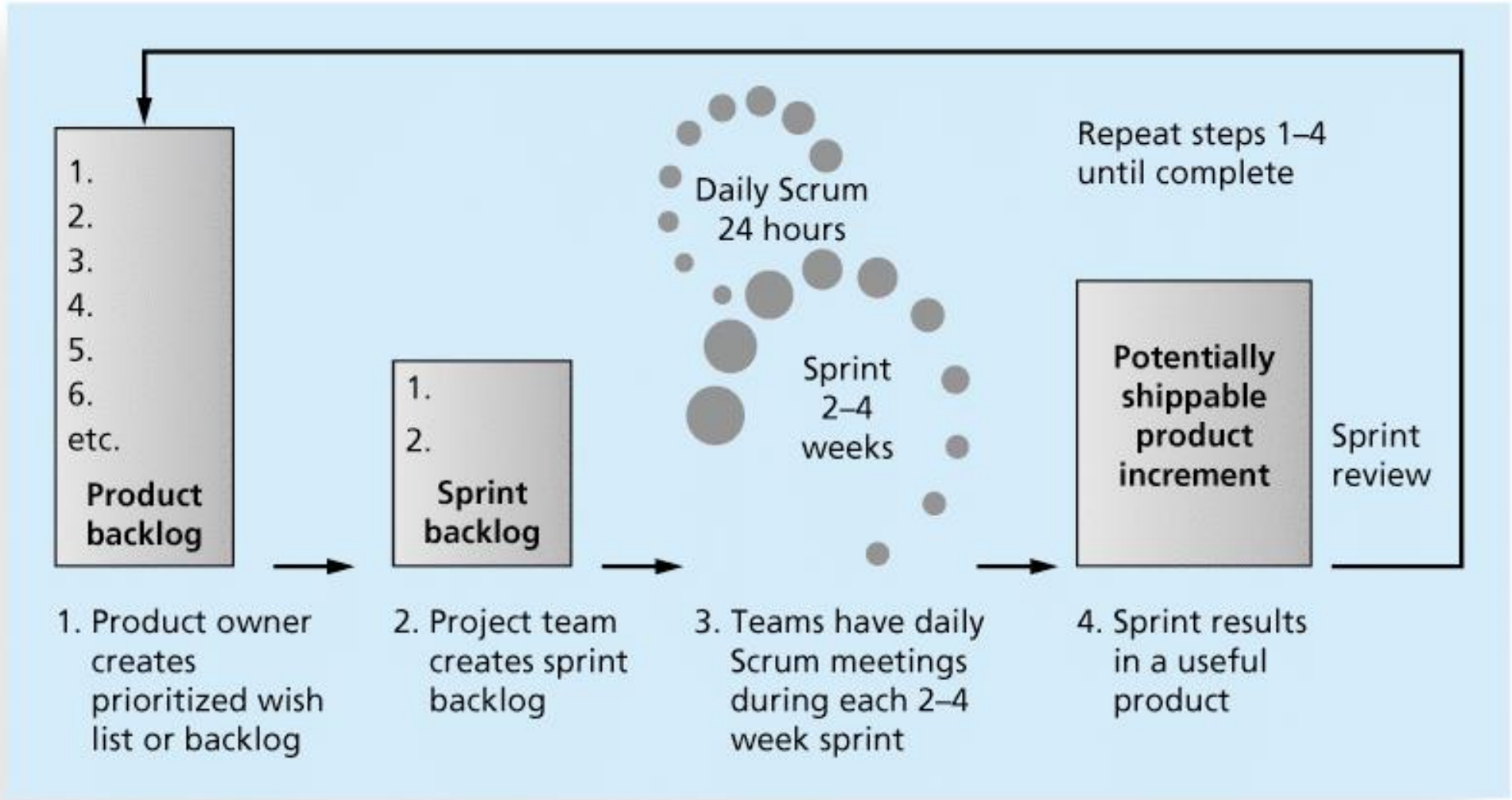


Fig. Scrum process flow


1.
2.
3.
4.
5.
6.
etc.
Product backlog
1.
2.
Sprint backlog
Daily Scrum
24 hours
Sprint
2–4
weeks
Repeat steps 1–4
until complete
Potentially shippable product increment
Sprint review
1. Product owner creates prioritized wish list or backlog
2. Project team creates sprint backlog
3. Teams have daily Scrum meetings during each 2–4 week sprint
4. Sprint results in a useful product

Scrum (Contd.)

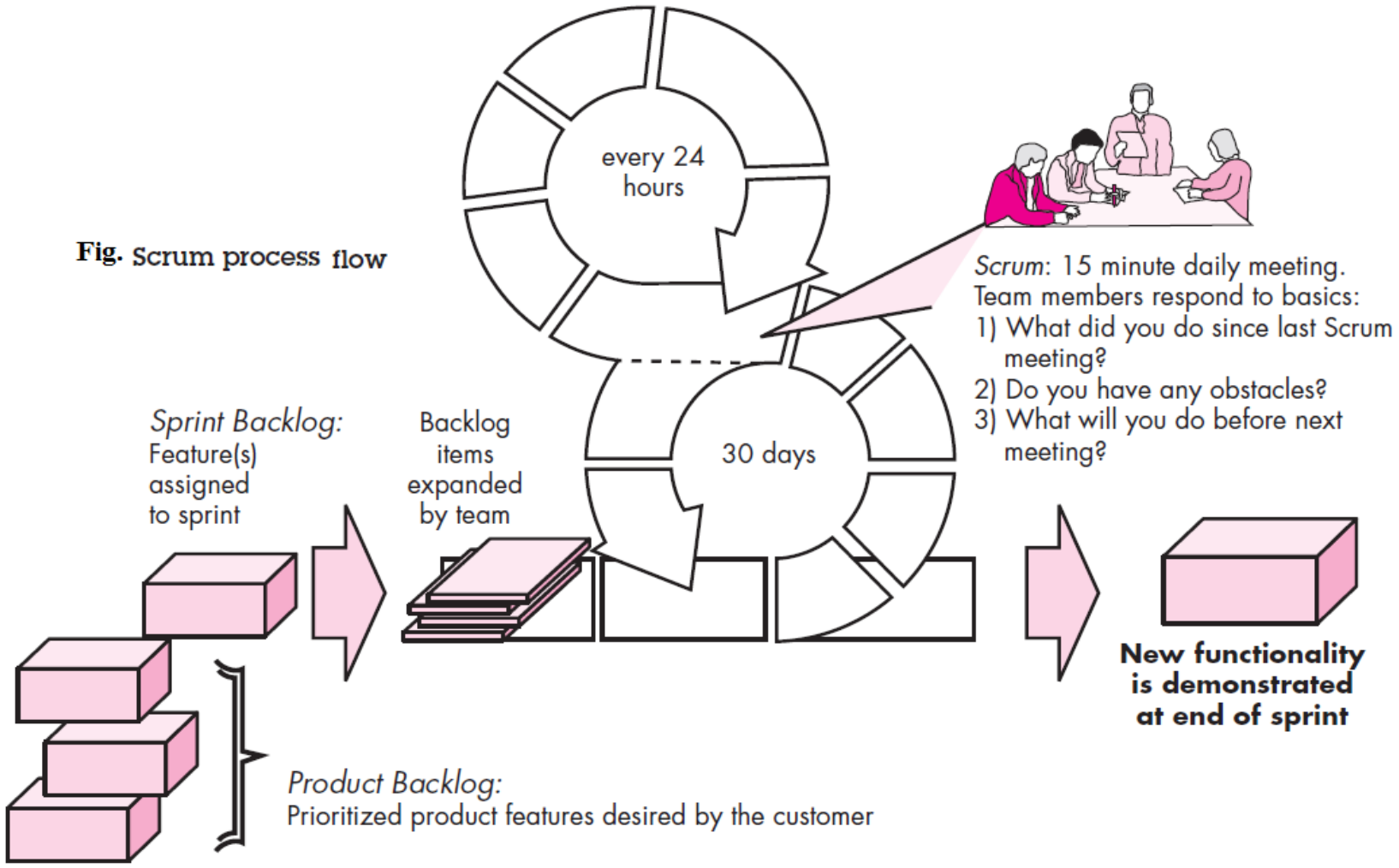


Fig. Scrum process flow

Product Backlog

- It is the master list of work that needs to get done maintained by the product owner or product manager.
- Items can be added to the backlog at any time.
- The product backlog is constantly revisited, re-prioritized and maintained by the Product Owner.

Scrum (Contd.)

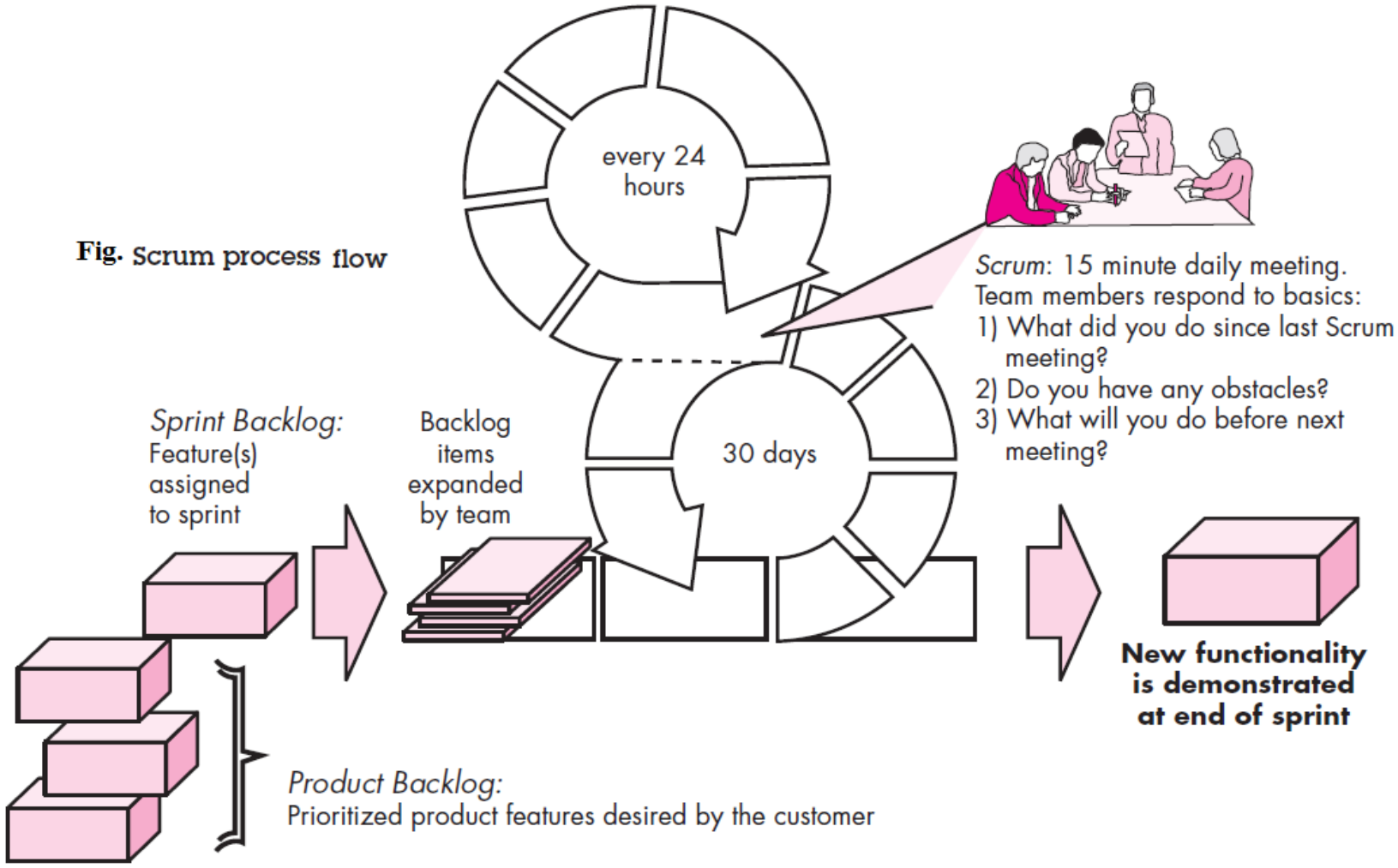


Fig. Scrum process flow

Sprint Backlog:

- It is the list of items, user stories, or bug fixes, selected by the development team for implementation in the current sprint cycle.
- Before each sprint, in the sprint planning meeting, the team chooses which items it will work on for the sprint from the product backlog.

Scrum (Contd.)

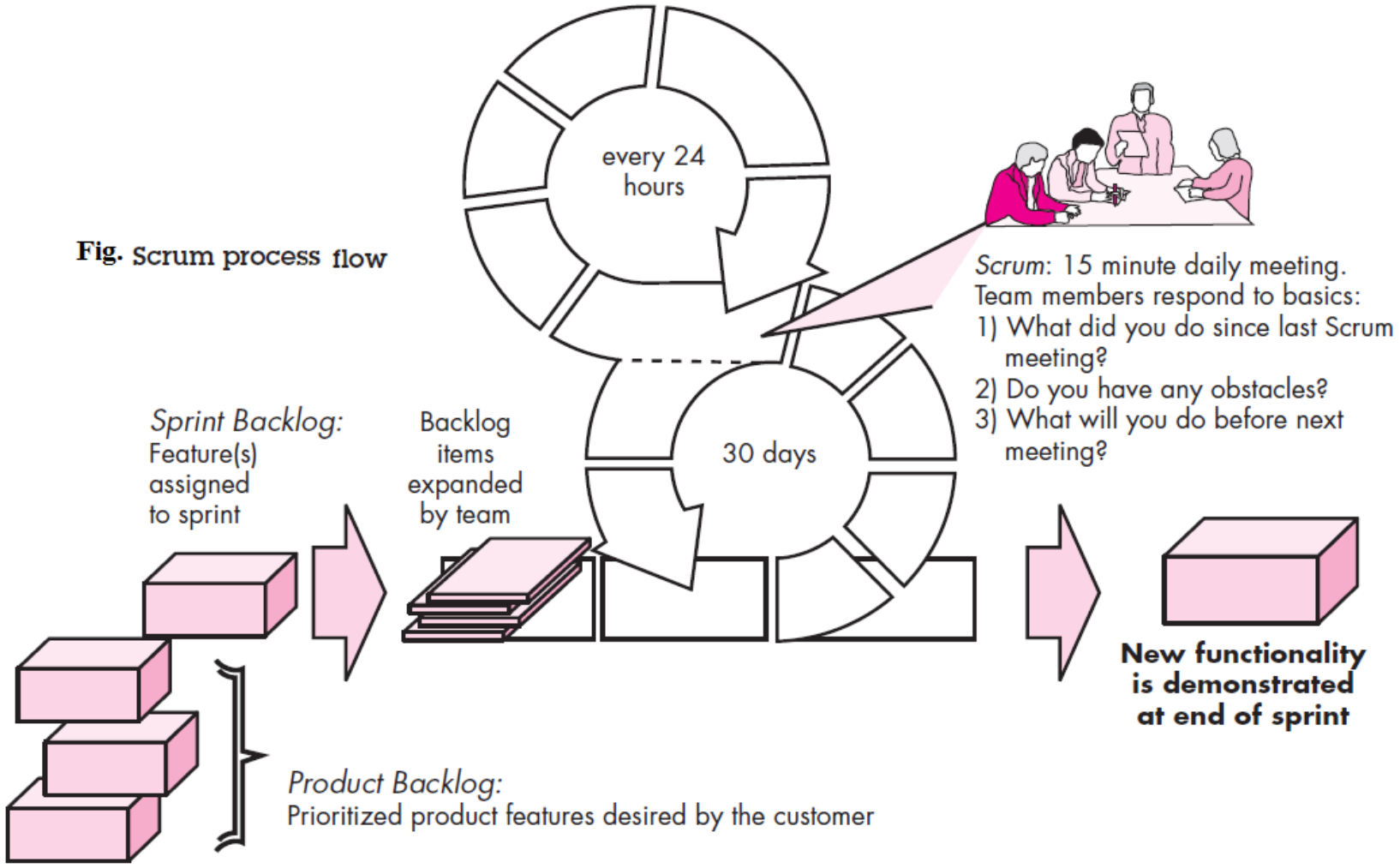


Fig. Scrum process flow

Scrum Meetings:

- These are short (typically 15 minutes) meetings held daily by the Scrum team.
- A team leader, called a Scrum master, leads the meeting and assesses the responses from each person.
- The Scrum meeting helps the team to uncover potential problems as early as possible.
- Also, these daily meetings lead to "knowledge socialization" and thereby promote a self-organizing team structure.

Scrum (Contd.)

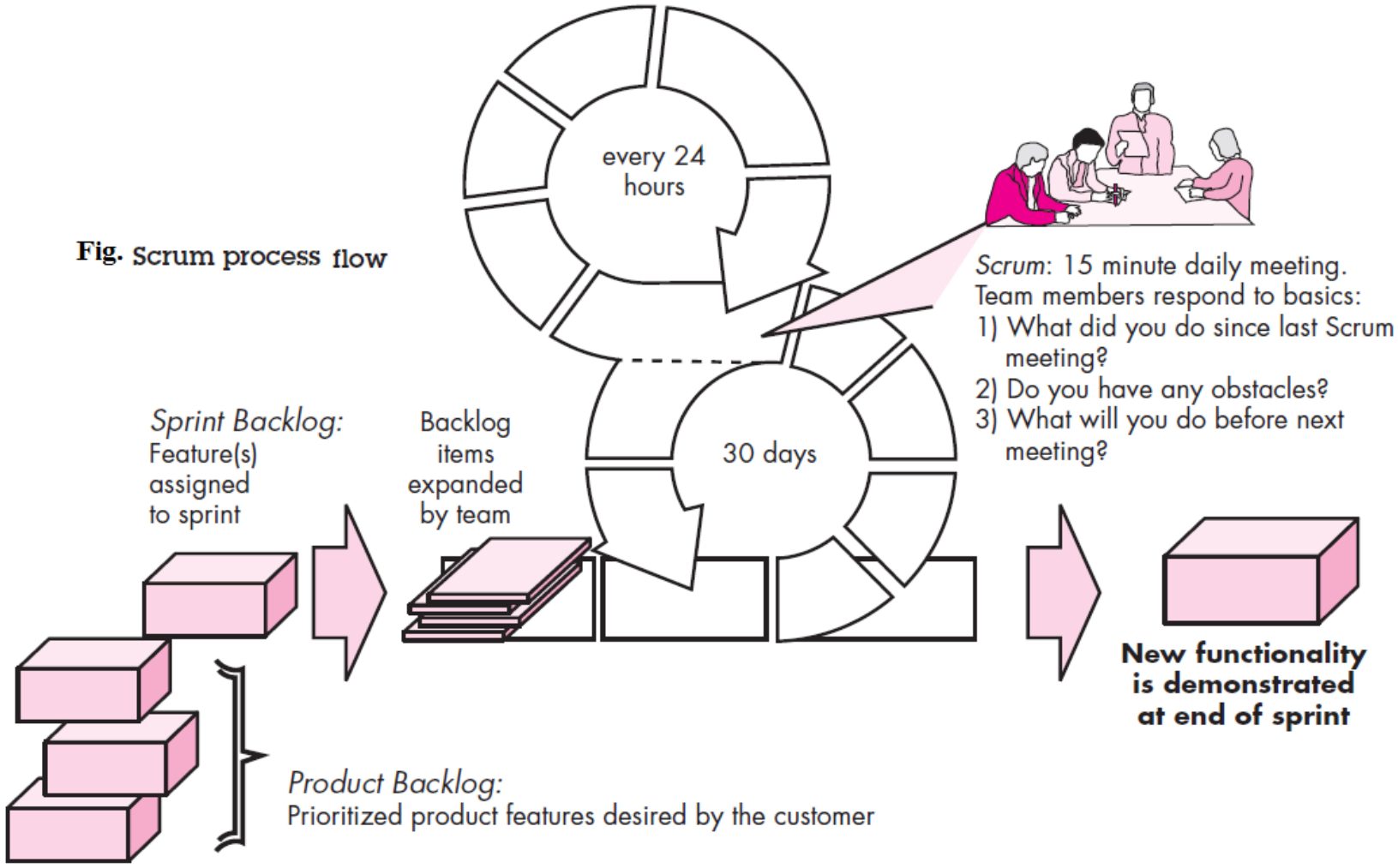


Fig. Scrum process flow

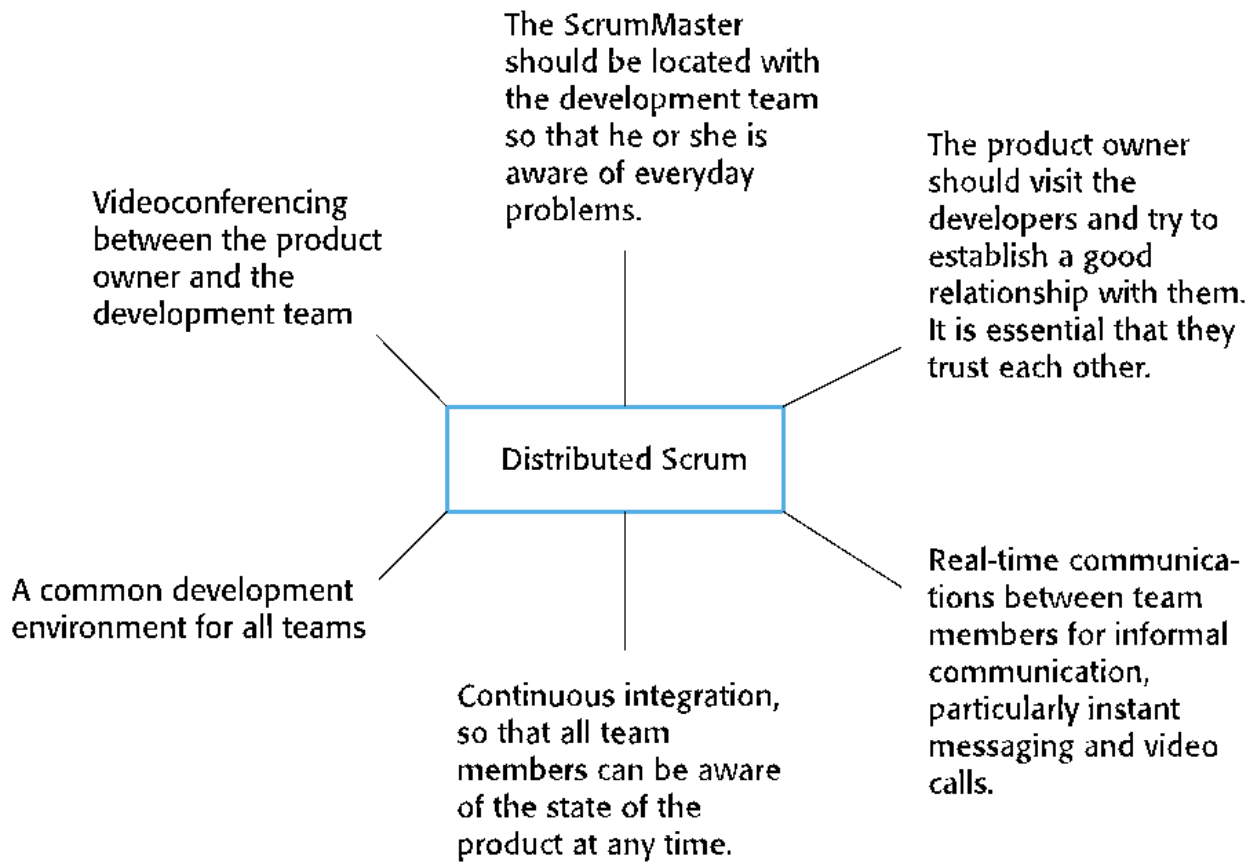
Demos:

- Demos deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.

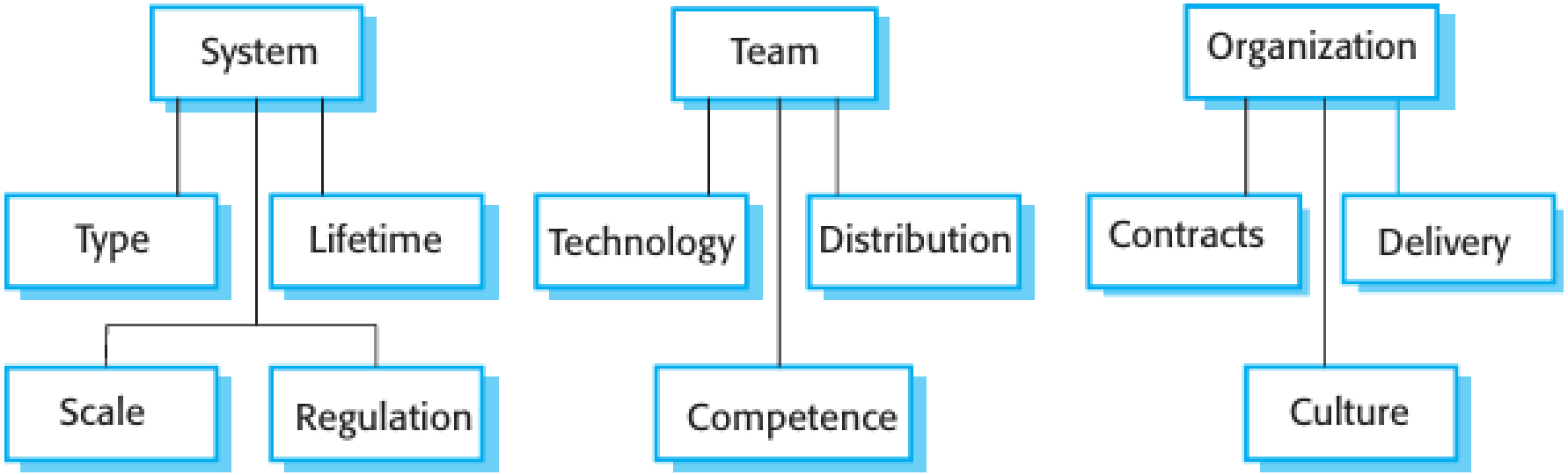
Benefits of Scrum

1. The product is broken down into a set of manageable and understandable chunks.
2. Unstable requirements do not hold up progress.
3. The whole team have visibility of everything and consequently team communication is improved.
4. Customers see on-time delivery of increments and gain feedback on how the product works.
5. Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

Distributed Scrum



Factors influencing the choice of Plan-based or Agile development



1. **How large is the system that is being developed?**
   - Agile methods are most effective when the system can be developed with a relatively 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.
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. However, supporters of agile methods rightly argue that documentation is frequently not kept up to date and is not of much use for long-term system maintenance.
4. **Is the system subject to external regulation?**
   - If a system has to be approved by an external regulator (e.g., the Federal Aviation Administration approves 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.

KEY POINTS

- Agile methods are iterative development methods that focus on reducing process overheads and documentation and on incremental software delivery. They involve customer representatives 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. In practice, a mix of agile and plan-based techniques may be used.
- Agile development practices include requirements expressed as user stories, pair programming, refactoring, continuous integration, and test-first development.
- Scrum is an agile method that provides a framework for organizing agile projects. It is centered around a set of sprints, which are fixed time periods when a system increment is developed. Planning is based on prioritizing a backlog of work and selecting the highest priority tasks for a sprint.
- To scale agile methods, some plan-based practices have to be integrated with agile practice. These include up-front requirements, multiple customer representatives, more documentation, common tooling across project teams, and the alignment of releases across teams.

Brief Answer Questions:

1. What is the main difference between Plan-driven development and Agile Development.
2. What is Rapid software development?
3. What are the main characteristics of Agile methods? List them..
4. Write the four key values of Agile methodology.
5. Mention any two Agile development techniques.
6. What is pair programming in XP agile model?
7. Define these terms in XP model: refactoring, spike solution
8. What is test automation?
9. Define these terms in Scrum model: Sprint, Product Backlog, Sprint Backlog, Sprint.
10. What are sprint cycles in Scrum model?
11. How scrum meeting are conducted?
12. What do you mean by refactoring?

Short Answer Questions:

1. Differentiate between Plan-driven and Agile Development.
2. What do you mean by Agile Software Development? Mention the key characteristics of such model.
3. Why Agile models are more popular nowadays? Explain.
4. Why do we prefer agile development rather than plan development? Support your answer with brief explanation of any one agile methods.
5. Explain the Extreme Programming model.
6. What are the user stories, metaphors, spikes, pair programming and refactoring in XP model?
7. How user stories can be used for requirement gathering?
8. How scrum method Is used for Agile project management? Explain.
9. Explain the scrum sprint cycle with suitable diagram.
10. Explain sprint and backlog.

End of Chapter