Department Of Computer Science

Software Engineering – I (CSC291)

Lecture 03

Agile Process Models

Objectives

- To introduce Agile software process models

Agile SDLC’s Model

Agile SDLC’s

- Speed up or bypass one or more life cycle phases
- Usually less formal and reduced scope
- Used for time-critical applications
- Used in organizations that employ disciplined methods

What is Agility?

- **Effective response to change**
  - Software being built, changes to the team members, changes because of new technology,
- Changes of all kinds that may have an impact on the product or on project
- **Agility**
  - **Encourages team structures** and attitudes that make communication
  - **Rapid delivery** of operational software
  - **Deemphasizes the importance** of intermediate work products
  - **Adopts the customer** as a part of the development team

What is Agility?

- **Agility can be applied** to any software process by meeting following criteria
  - **The process be designed** in a way that allows the project team to adapt tasks and to streamline them
  - **Conduct planning** in a way that understands the fluidity of an agile development approach
  - **Eliminate all but the most essential work** products and keep them lean
  - **Emphasize an incremental delivery strategy** that gets working software to the customer as rapidly as feasible

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
  - 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
- **Prioritizing changes** can be difficult where there are multiple stakeholders.
- **Contracts may be a problem** as with other approaches to iterative development.
  - As requirement document is part of contract between customer and supplier
- **Organizations have spent year in formalizing their process** so it is difficult for them to adopt informal process model defined by development teams

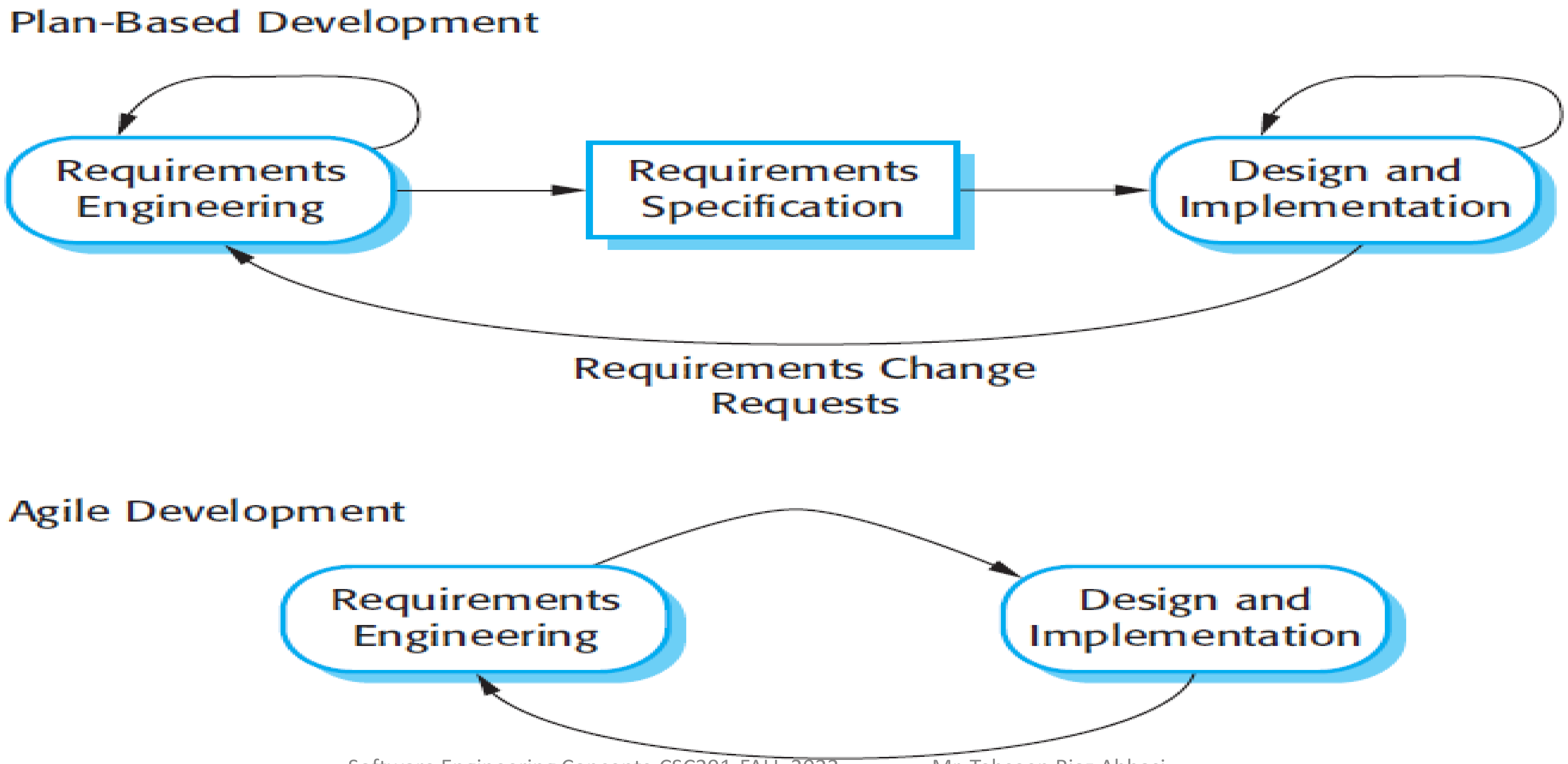
Agile Methods and Software Maintenance

- **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?**
  - Key to implementing maintainable software is to produce high quality and well-structured code instead of documentation
  - **Can agile methods be used effectively for evolving a system in response to customer change requests?**
    - Incremental delivery, design for change and maintaining simplicity all make sense when software is being changed which shows agile methods are used effectively
- **Problems may arise if original development team is broken up.**

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**
  - Specification, design, implementation and testing are inter-leaved
  - 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

Agile vs. Plan-driven Methods

Agile methods	Plan-driven methods
Low criticality	High criticality
Senior developers	Junior developers
Requirements change often	Requirements do not change often
Small number of developers	Large number of developers
Culture that responds to change	Culture that demands order

Agile Methods

- Well-known agile software development methods and/or process frameworks include:
  - **Rapid Application Development (RAD)**
  - **Rational Unify Process (RUP)**
  - **Extreme programming (XP)**
  - **Scrum**
  - Test Driven Development (TDD)
  - Feature Driven Development (FDD)
  - Dynamic systems development method (DSDM)
  - Agile Unified Process (AUP)
  - Adaptive Software Development (ASD)
  - Agile Modeling
  - Crystal Clear

• **Agile Process Model**

RUP

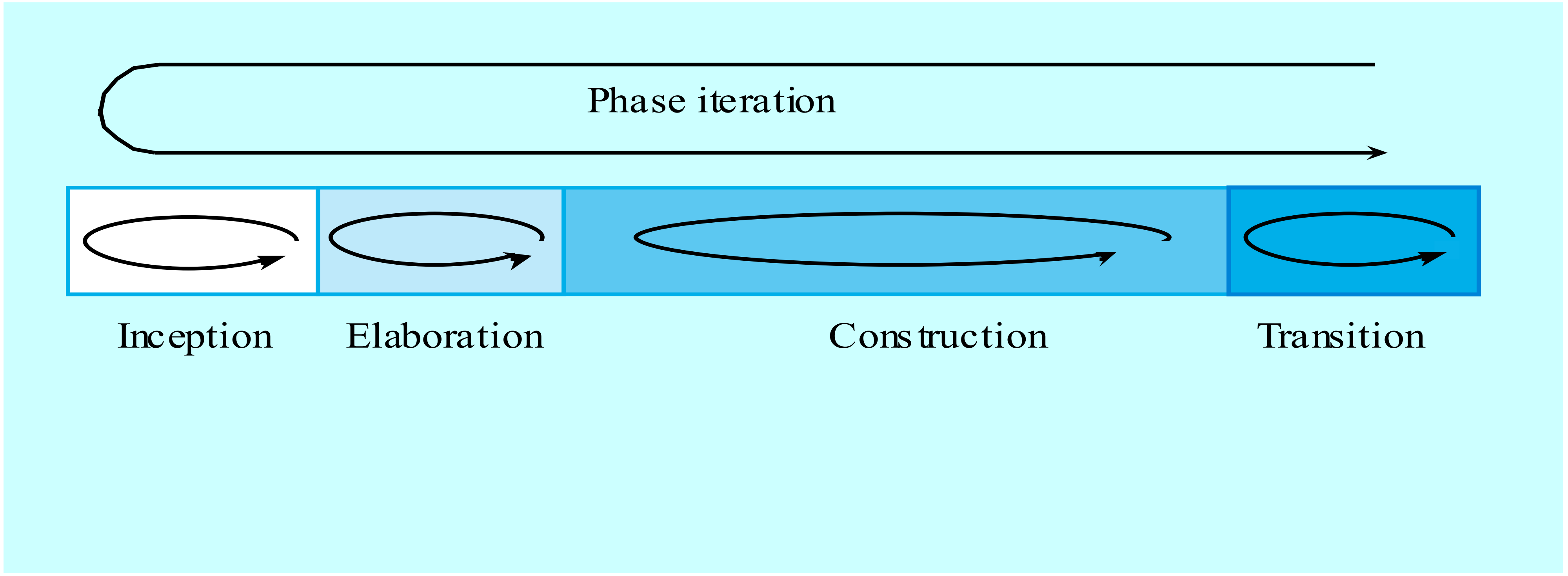
RAD

- **RUP**

‣ The Rational Unified Process

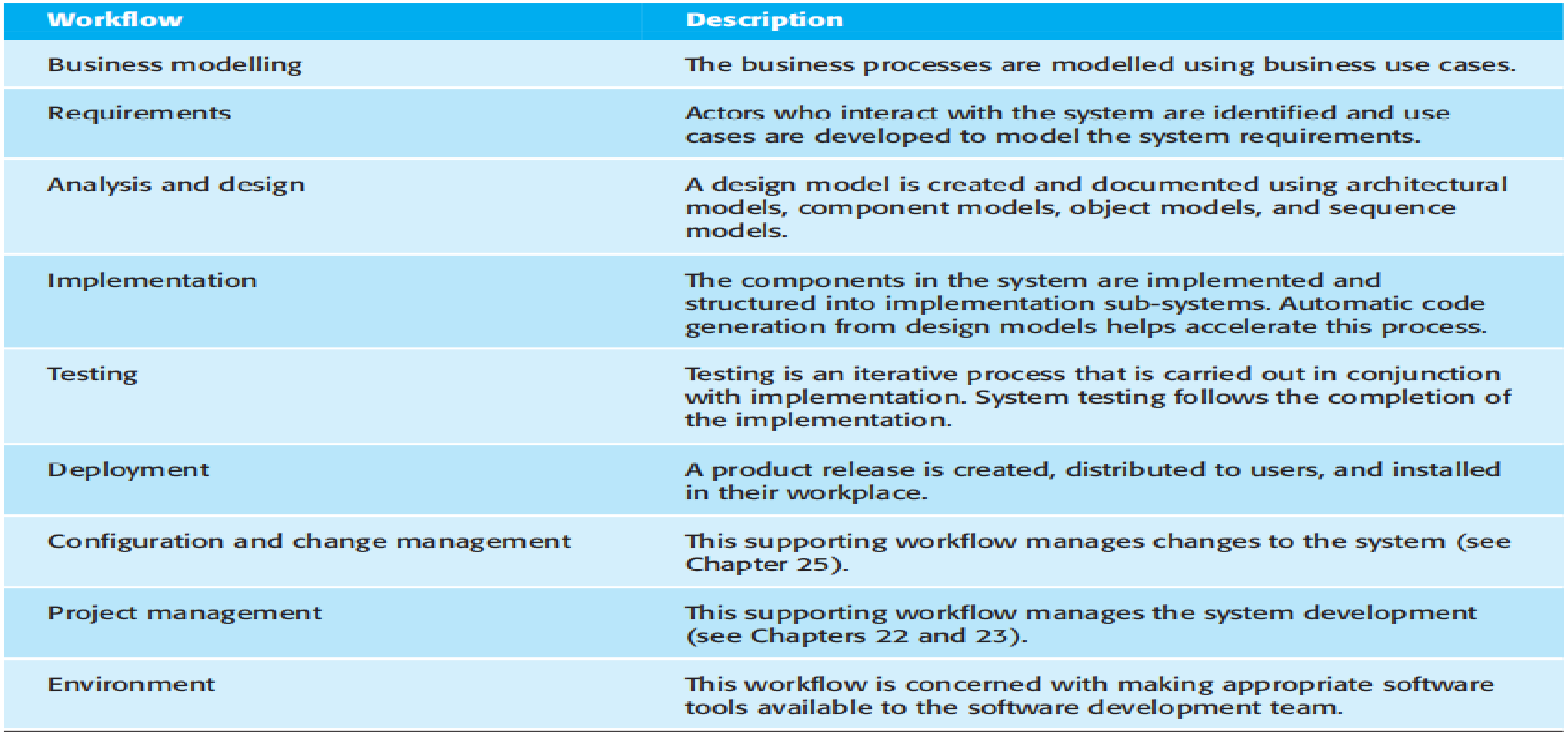
- A modern process model derived from the work on the UML and associated process.
- **Normally described from 3 perspectives**
  - A **dynamic perspective** that shows phases over time;
  - A **static perspective** that shows process activities;
  - A **proactive perspective** that suggests good practice.

RUP phase model



RUP phase model

Static workflows in the Rational Unified Process



Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models, and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Testing	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users, and installed in their workplace.
Configuration and change management	This supporting workflow manages changes to the system (see Chapter 25).
Project management	This supporting workflow manages the system development (see Chapters 22 and 23).
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

RUP iteration

- **In-phase iteration**
  - Each phase is iterative with results developed incrementally.
- **Cross-phase iteration**
  - As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.

RUP phases

- **Inception**
  - Establish the business case for the system.
- **Elaboration**
  - Develop an understanding of the problem domain and the system architecture.
- **Construction**
  - System design, programming and testing.
- **Transition**
  - Deploy the system in its operating environment.

‣ RUP good practice

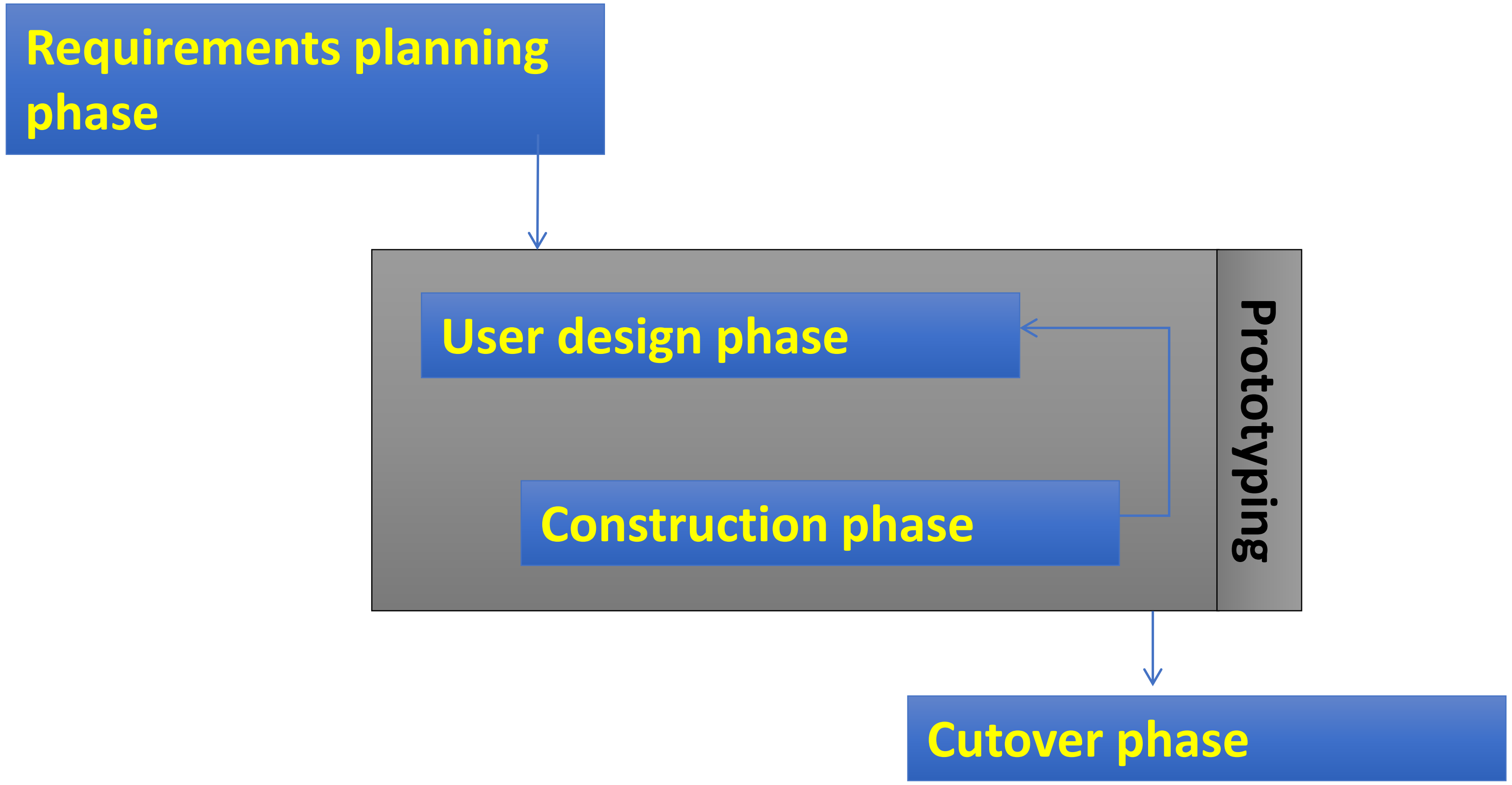
- **Develop** software iteratively
- **Manage** requirements
- **Use component-based** architectures
- **Visually model** software
- **Verify software quality**
- **Control changes** to software

RAD Model

Rapid Application Model (RAD)

- Requirements planning phase (a workshop utilizing structured discussion of business problems)
- User description phase – automated tools capture information from users
- Construction phase – productivity tools, such as code generators, screen generators, etc. inside a time-box. (“Do until done”)
- Cutover phase -- installation of the system, user acceptance testing and user training

Rapid Application Development Model (RAD)



RAD Strengths

- Reduced cycle time and improved productivity with fewer people means lower costs
- Time-box approach mitigates cost and schedule risk
- Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs
- Focus moves from documentation to code (WYSIWYG).
- Uses modeling concepts to capture information about business, data, and processes.

RAD Weaknesses

- Accelerated development process must give quick responses to the user
- Risk of never achieving closure
- Hard to use with legacy systems
- Requires a system that can be modularized
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

When to use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized

Topics Covered

- Agile Process Model
- **XP**
- **SCRUM**

▸ Extreme Programming (XP)

Extreme Programming - XP

- For small-to-medium-sized teams developing software with vague or rapidly changing requirements
- Coding is the key activity throughout a software project
- Communication among teammates is done with code
- Life cycle and behavior of complex objects defined in test cases – again in code

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.
  - Requirements are based on simple customer stories or scenarios
- **Customer involvement** means full-time customer engagement with the team.
  - Responsible for defining acceptance tests for the system
- **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.

XP Practices (1-6)

1. Planning game – determine scope of the next release by combining business priorities and technical estimates
2. Small releases – put a simple system into production, then release new versions in very short cycle
3. Metaphor – all development is guided by a simple shared story of how the whole system works
4. Simple design – system is designed as simply as possible (extra complexity removed as soon as found)
5. Testing – programmers continuously write unit tests; customers write tests for features
6. Refactoring – programmers continuously restructure the system without changing its behavior to remove duplication and simplify

XP Practices (7 – 12)

7. Pair-programming -- all production code is written with two programmers at one machine
8. Collective ownership – anyone can change any code anywhere in the system at any time.
9. Continuous integration – integrate and build the system many times a day – every time a task is completed.
10. 40-hour week – work no more than 40 hours a week as a rule
11. On-site customer – a user is on the team and available full-time to answer questions
12. Coding standards – programmers write all code in accordance with rules emphasizing communication through the code

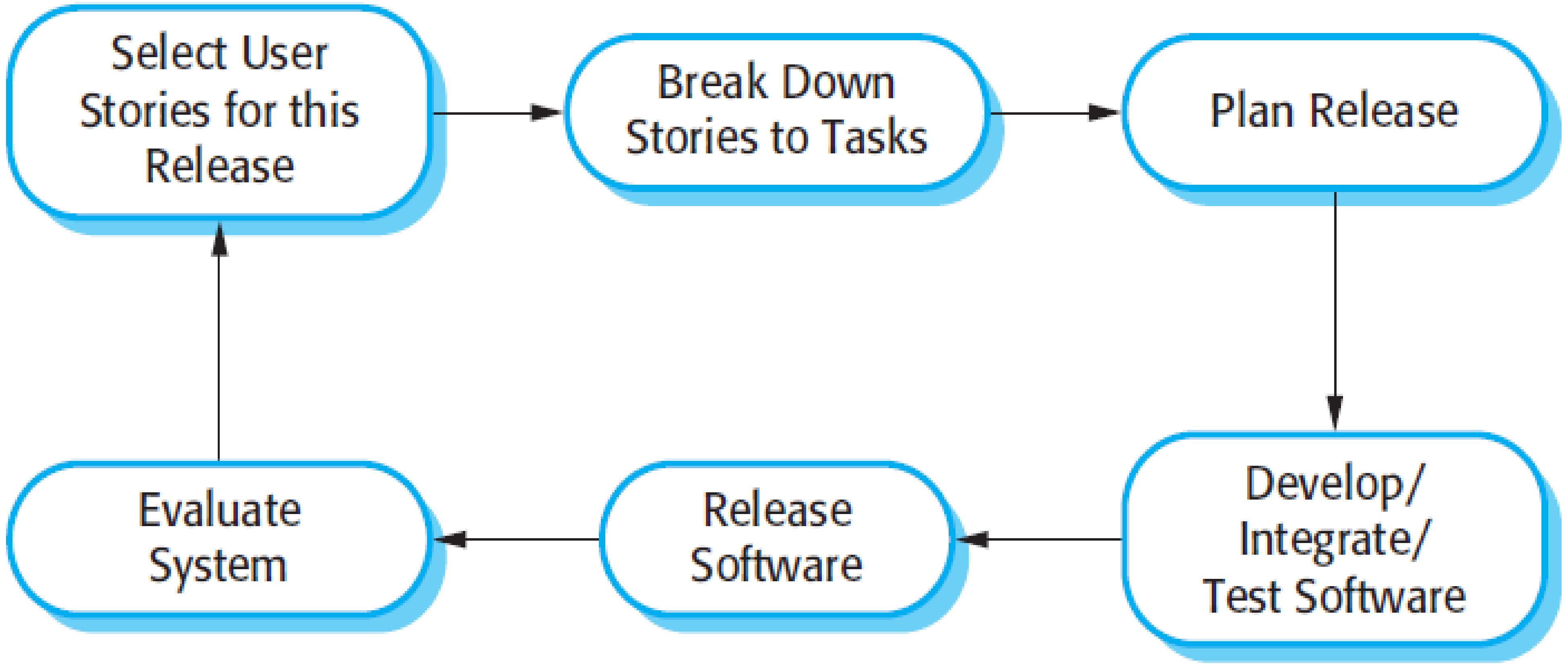
Extreme programming practices [1/2]

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.

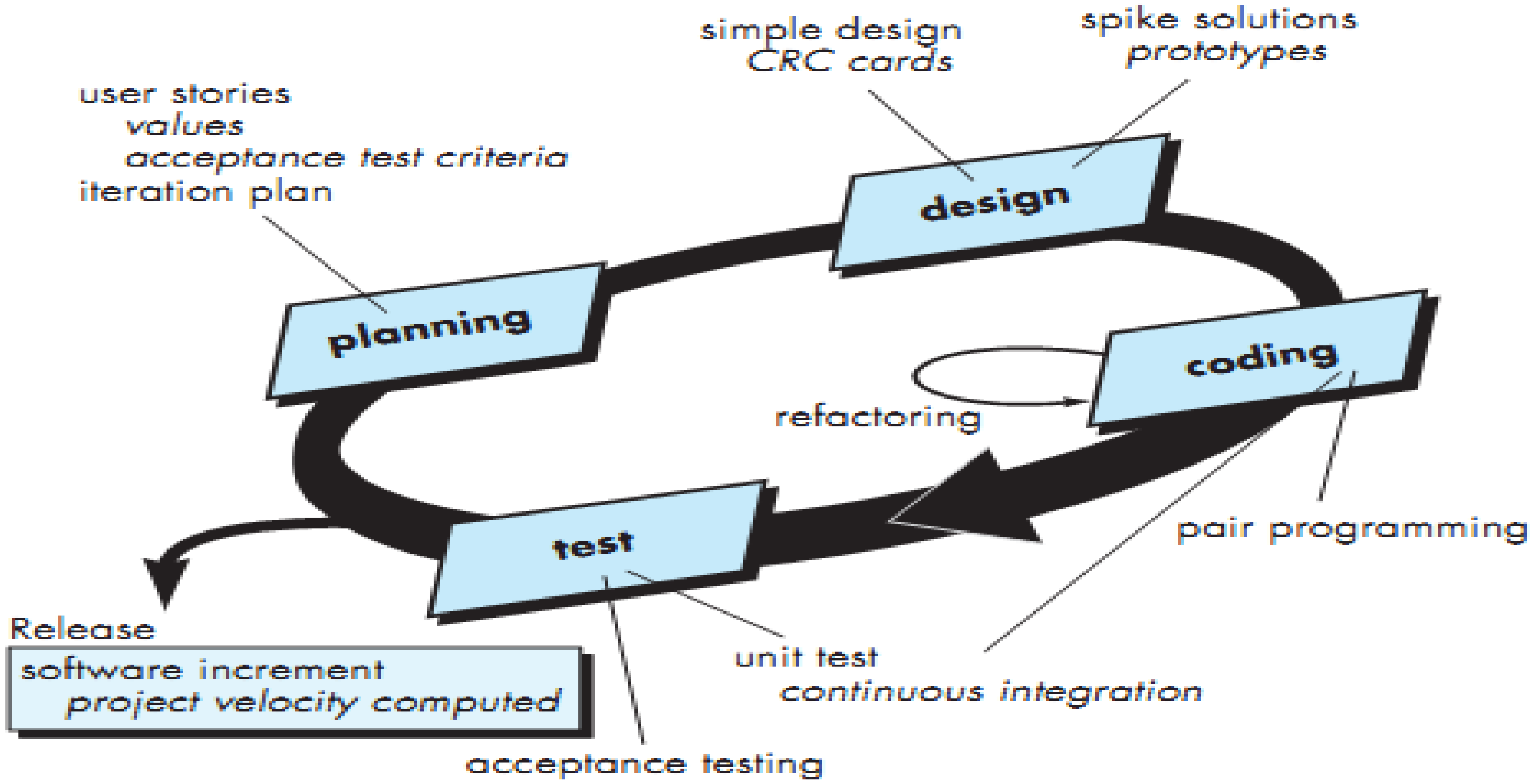
Extreme programming practices [2/2]

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.

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

Extreme Programming



Planning

- 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**
- **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.

Design

- **Provides implementation guidance** for a story as it is written
- **XP encourages the use of CRC** (class-responsibility-collaborator) cards
  - Identify and organize the object oriented classes that are relevant to the current software increment
- **If a difficult design problem is encountered**
  - Immediately creation of an operational prototype of that portion of the design **called *Spike***
  - Intent is to lower risk when true implementation starts
- Encourage ***refactoring***

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.
  - Eclipse include tools for refactoring which simplify the process of finding dependencies between code sections and making global code modifications
- **In principle**, software is easy to understand and change as new stories are implemented

<u>Coding</u>

- **After creating simple design team** move to developing tests instead of coding
  - **Develops unit test**; the developer is better able to focus on what must be implemented to pass the test
- **A key concept during the coding activity** is pair programming

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** between pairs reduces the overall risks to a project when team members leave.
- **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.
- 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.

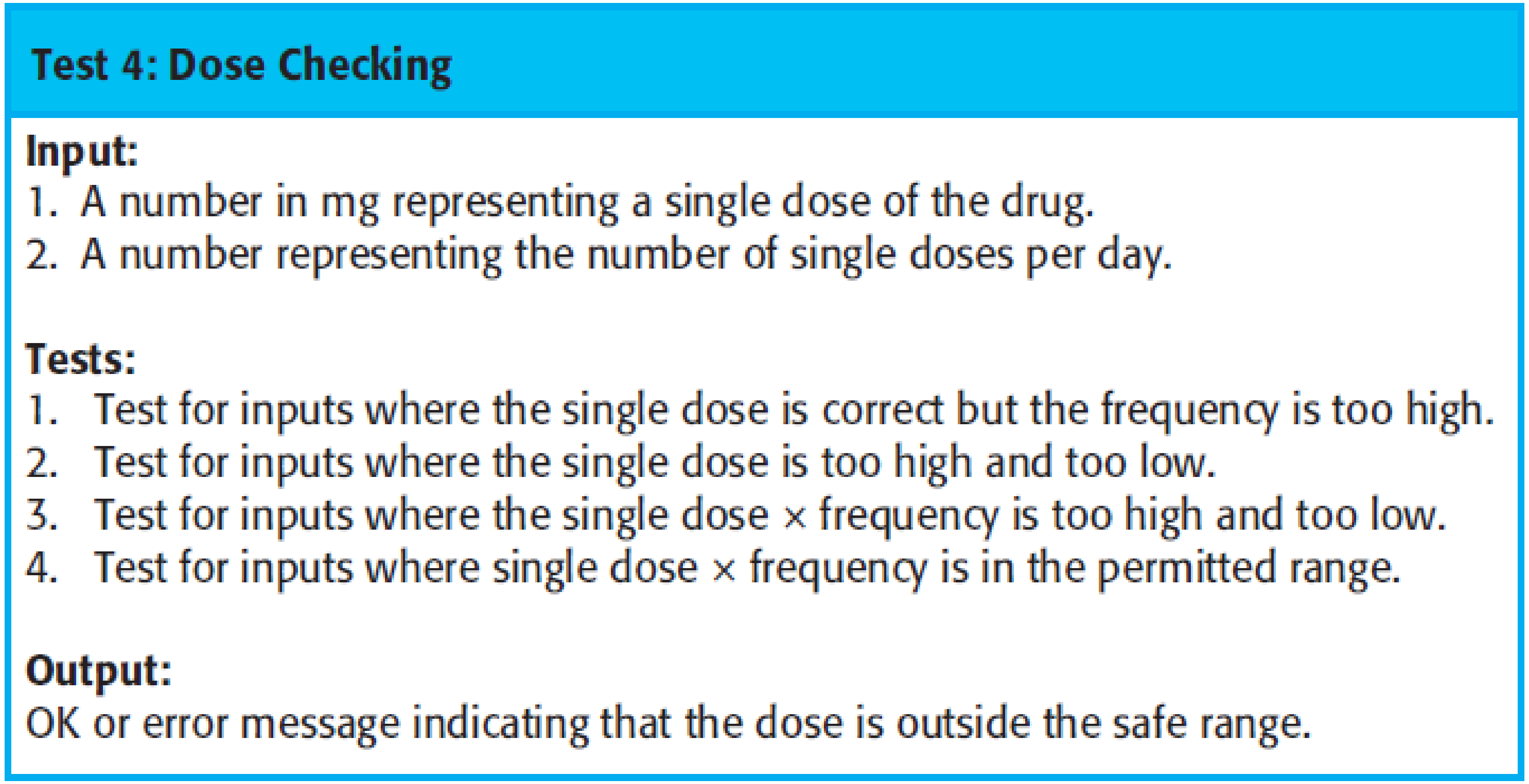
Testing in XP

- **XP testing features:**
  - **Test-first development.**
  - **Incremental test development** from scenarios.
  - **User involvement in test development** and validation.
  - **Automated test** 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.
  - Ambiguities have to clarify before actual implementation
- **Tests are written as programs** 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.

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.

‣ **Scrum**

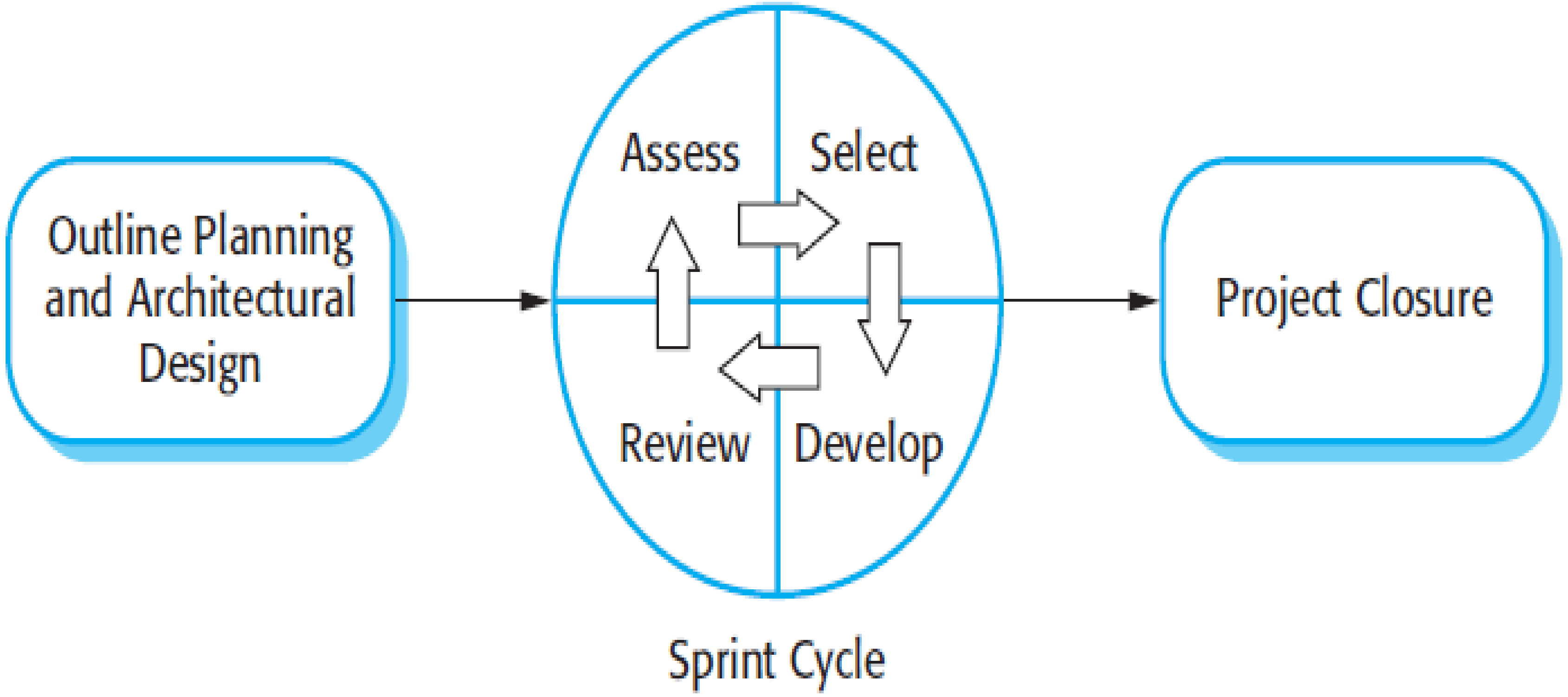
SCRUM

- **What is Scrum?**
- **'SCRUM' originally derives from a strategy in the game of rugby** where it denotes "getting an out of play ball back into the game" with teamwork.
- **SCRUM enables OO through advanced iterative**, incremental development within a controlled environment. It allows organizations to build the best possible OO systems possible.
- **SCRUM is one of many different types of Agile Software Development** Frameworks that you can use in the real world to deliver working software. It is all about people and not technological
- **SCRUM is an Agile process framework that allows organizations to continuously direct** the project toward early delivery of real business value through the frequent and regular delivery of high quality software.

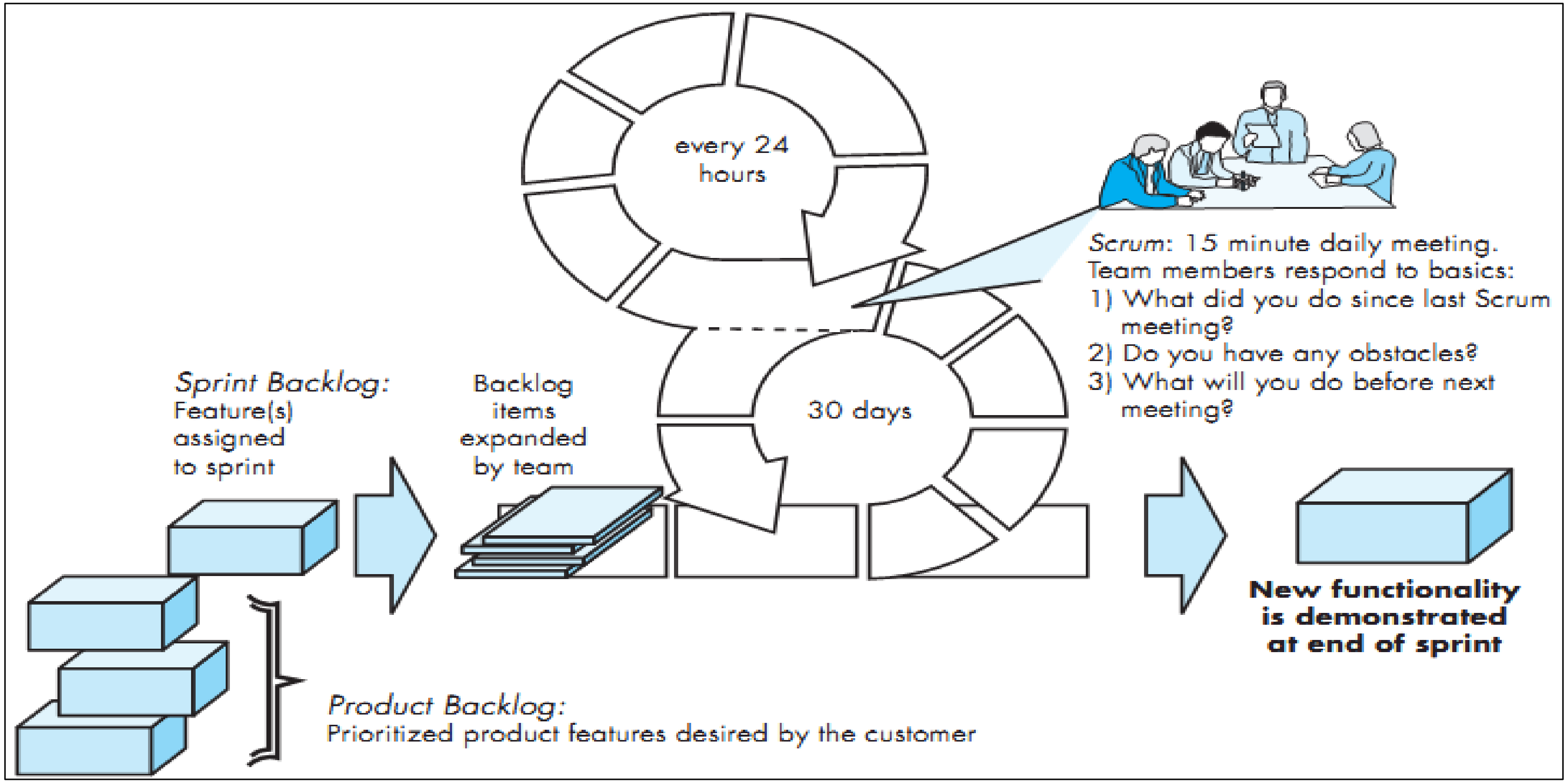
Scrum

- **The Scrum approach is a general agile method but its focus is on <u>managing iterative development</u>**
- **There are three phases in Scrum.**
  - **<u>Outline planning phase</u>** where you establish the **<u>general objectives for the project</u> and <u>design the software architecture</u>.**
  - This is followed by a series of **<u>sprint cycles</u>,** where each cycle develops an increment of the system.
  - **The <u>project closure phase</u>** wraps up the project, completes required documentation
    - such as user manuals and assesses the lessons learned from the project.

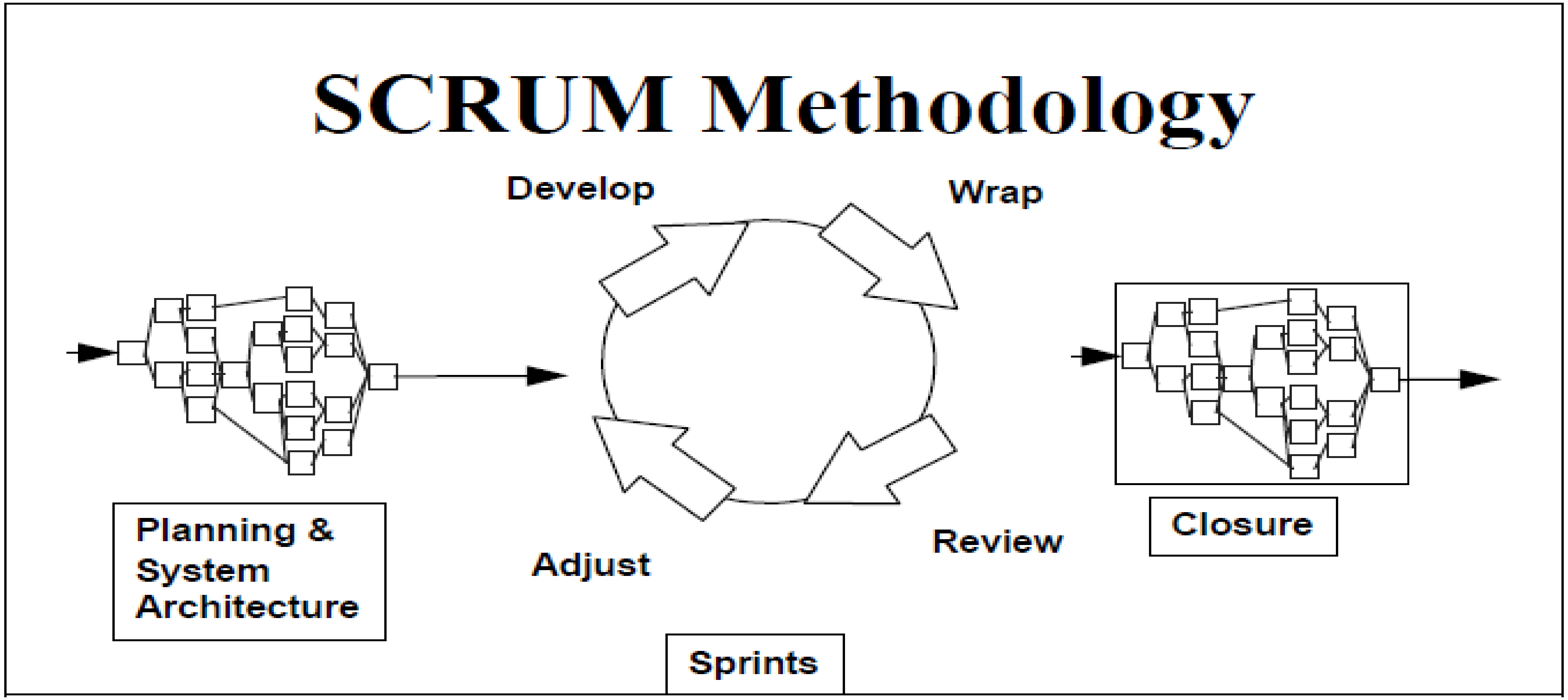
The Scrum Process



The Scrum Process


every 24 hours
Scrum: 15 minute daily meeting.
Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
30 days
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

The Scrum Process



The Sprint cycle

- **Sprints are fixed length, normally 2–4 weeks**. They correspond to the development of a release of the system in XP.
- **The Assess Phase**
  - **Product backlog is reviewed**, and priorities and risks are assigned
  - Customer is involved to introduce new requirements or tasks at beginning of each sprint
- **The Selection Phase**
  - **Involves all of the project team who work with the customer** to select the features and functionality to be developed during the sprint.

The Sprint cycle

- **The Develop Phase**
  - **The team organize themselves to develop the software**.
  - The team is isolated from the customer and the organization
  - All communication is channelled through the so-called ‘Scrum master’.
  - The role of the Scrum master is to protect the development team from external distractions.
- **The Review Phase**
  - **At the end of the sprint, the work done is reviewed and presented to stakeholders.**
  - The next sprint cycle then begins.

INTRODUCTION TO TERMS

- **Scrum Roles**
  - Product Owner
  - Scrum Master
  - Scrum Team
- Sprint
- Daily Scrum
- Product Backlog
- Sprint Backlog
- Scrum Meeting Rules

Teamwork in Daily Scrum

- **The whole team attends short daily meetings**
  - What did you do since the last team meeting?
  - What obstacles are you encountering?
  - What do you plan to accomplish by the next team meeting?
- 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.

Roles and Responsibilities

- **Scrum Master (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.
- **Product Owner**
  - Responsible for project, managing, controlling and making visible Product Backlog list
  - Makes final decisions of the tasks related to product Backlog
- **Scrum Team**
  - It is project team
  - Organize itself in order to achieve the goals of each Sprint
  - Involved in effort estimation, creating Sprint Backlog, reviewing Product Backlog list

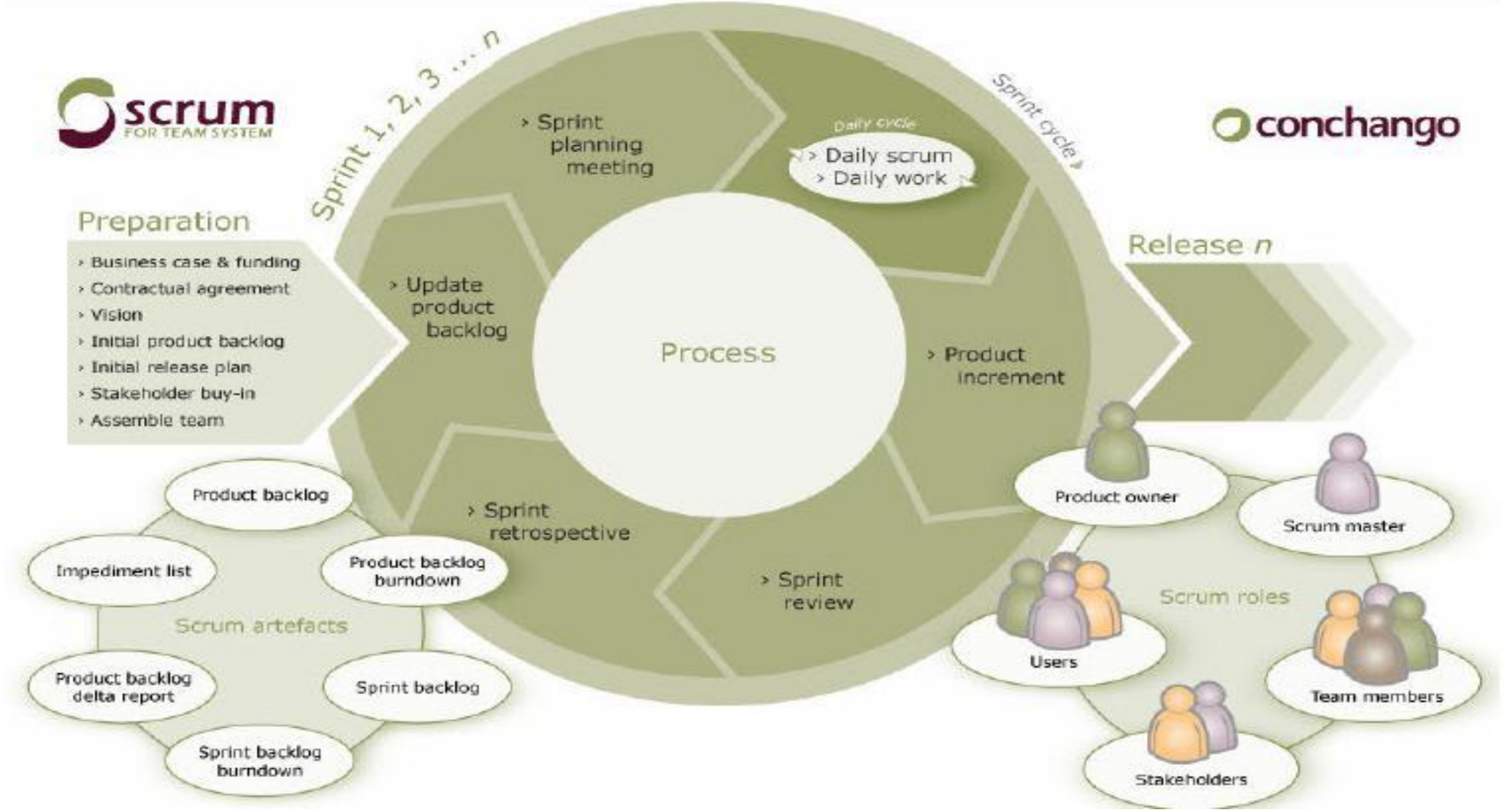
Roles and Responsibilities

- **Customer**
  - Participate in tasks related to product Backlog items
- **Management**
  - In charge of final decision making
  - Participate in setting of goals and requirements
  - E.g. management is involved in selecting Product Owner

Scrum benefits

- Rising and Janoff (2000) discuss its advantages as follows
  - **The product is broken down** into a set of manageable and understandable chunks.
  - **The whole team have visibility** of everything and consequently team communication is improved.
  - **Customers see on-time delivery of increments** and gain feedback on how the product works.
  - **Trust between customers and developers is established** and a positive culture is created in which everyone expects the project to succeed.

SCRUM


scrum FOR TEAM SYSTEM
conchango
Sprint 1, 2, 3 ... n
Sprint cycle
Preparation
› Business case & funding
› Contractual agreement
› Vision
› Initial product backlog
› Initial release plan
› Stakeholder buy-in
› Assemble team
› Sprint planning meeting
Daily cycle
› Daily scrum
› Daily work
› Update product backlog
Process
› Product increment
› Sprint retrospective
› Sprint review
Release n
Product backlog
Impediment list
Product backlog burndown
Scrum artefacts
Product backlog delta report
Sprint backlog
Sprint backlog burndown
Product owner
Scrum master
Users
Scrum roles
Team members
Stakeholders

SCRUM'S BENEFITS/ CHARACTERISTICS

- **Rising and Janoff (2000) discuss its advantages as follows**
  - The **product is broken down** into a set of manageable and understandable chunks.
  - The **whole team have visibility** of everything and consequently team communication is improved.
  - **Customers see on-time delivery of increments** and gain feedback on how the product works.
  - **Trust between customers and developers** is established and a positive culture is created in which everyone expects the project to succeed.
  - It enables **project to proceed systematically**
  - **Constant testing and documentation**
  - A **deliverable product is always ready**
  - **Frequent demonstrations** for early feedback from stakeholders
  - **Team spirit**
  - **Sense off accomplishment**
  - **Quality off product**

WHY USE SCRUM?

- **Advantages**
- Productivity increases
- Continuous improvement
- Leverages the chaos

- **Disadvantages**
- Scrum requires constant monitoring both quantitatively and qualitatively
- People are resistant to change
- Costly

CONCLUSION

- Scrum is many things. **Scrum is a management process** Scrum accepts the empirical nature of software development as a given, working within those constraints to enable the team to deliver as much business value as possible as quickly as possible..

Conclusions

- **Different life-cycle models**
  - Each with its own strengths
  - Each with its own weaknesses
- **Criteria for deciding on a model include:**
  - The organization
  - Its management
  - Skills of the employees
  - The nature of the product
- **Best suggestion**
  - “Mix-and-match” life-cycle model

Thank you