The Rise of Agile Methods

Week 2: Agile Software Development

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

The Rise of 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 <u>code</u> rather than the design
  - Are based on an <u>iterative approach</u> to software development
  - Are intended to deliver working software <u>quickly</u> and evolve this quickly to meet changing requirements.
- The goal of agile methods is to <u>reduce overhead</u> in the software process (e.g., by limiting documentation) and to be able to <u>respond quickly to changing requirements</u> without excessive rework.

Agile Manifesto

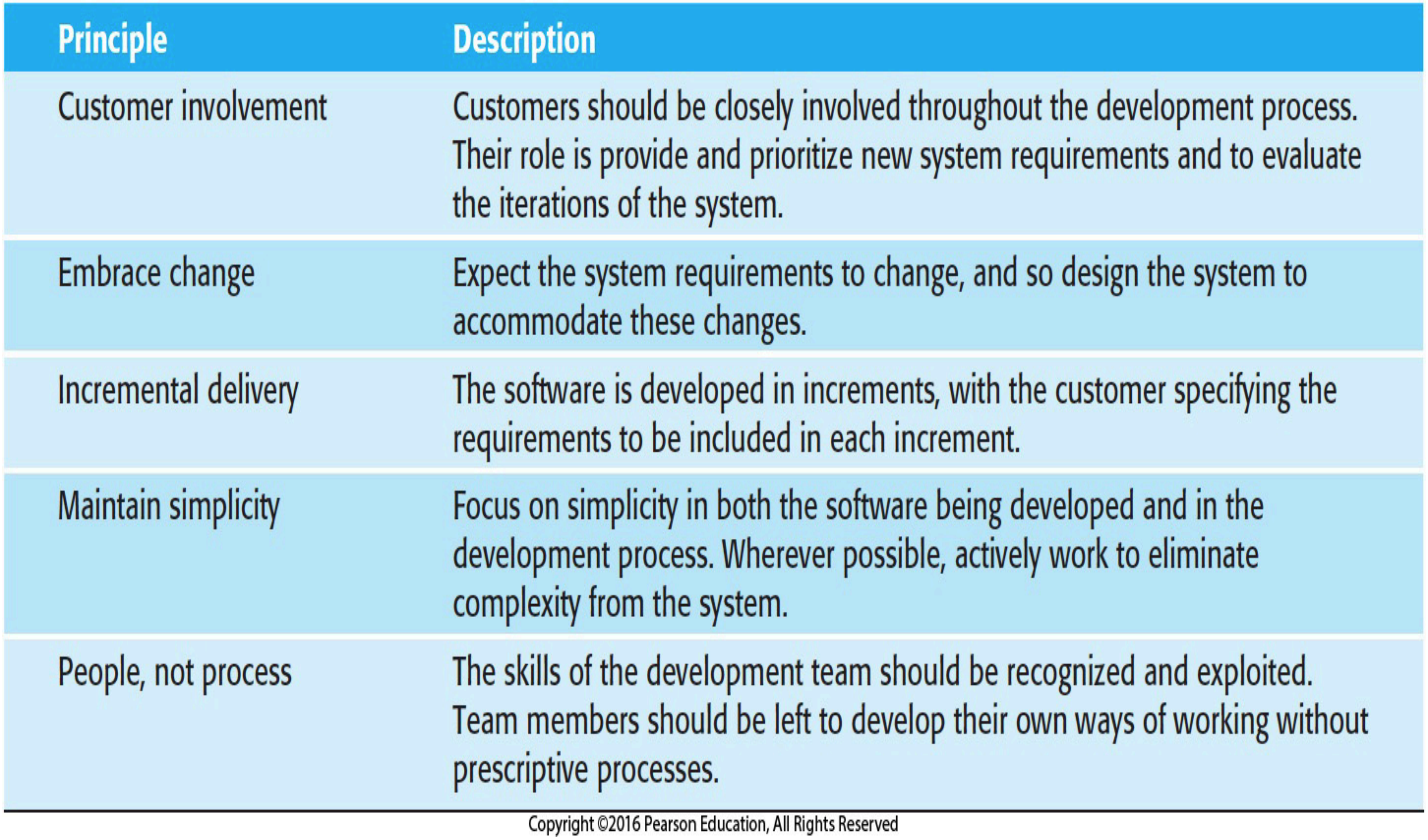
"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

<u>Individuals and interactions</u> over <u>processes and tools</u>
<u>Working software</u> over <u>comprehensive documentation</u>
<u>Customer collaboration</u> over <u>contract negotiation</u>
<u>Responding to change</u> over <u>following a plan</u>

That is, while there is value in the items on the right, we value the items on the left more."

http://agilemanifesto.org/

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.
Embrace change	Expect the system requirements to change, and so design the system to accommodate these changes.
Incremental delivery	The software is developed in increments, with the customer specifying the requirements to be included in each increment.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.
People, not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.

Plan-Based vs. Agile

Week 2: Agile Software Development

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

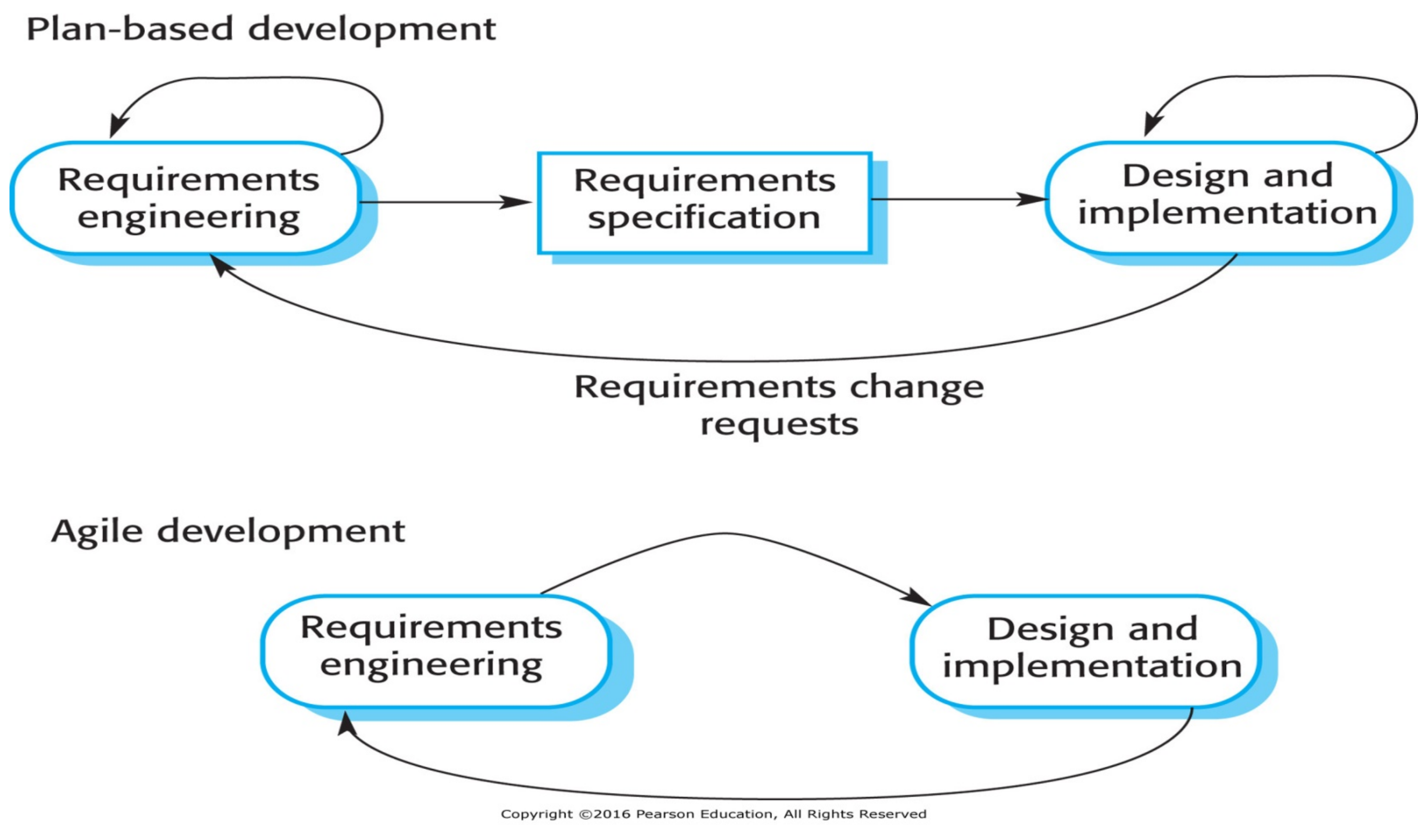
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Plan-Driven and Agile Processes

- There are <u>two</u> main categories of software processes:
  - Plan-driven processes:
    - Processes where <u>all</u> of the process activities are planned in advance and progress is measured against this plan.
  - Agile processes:
    - Processes where planning is incremental, and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- **There are no right or wrong software processes.**

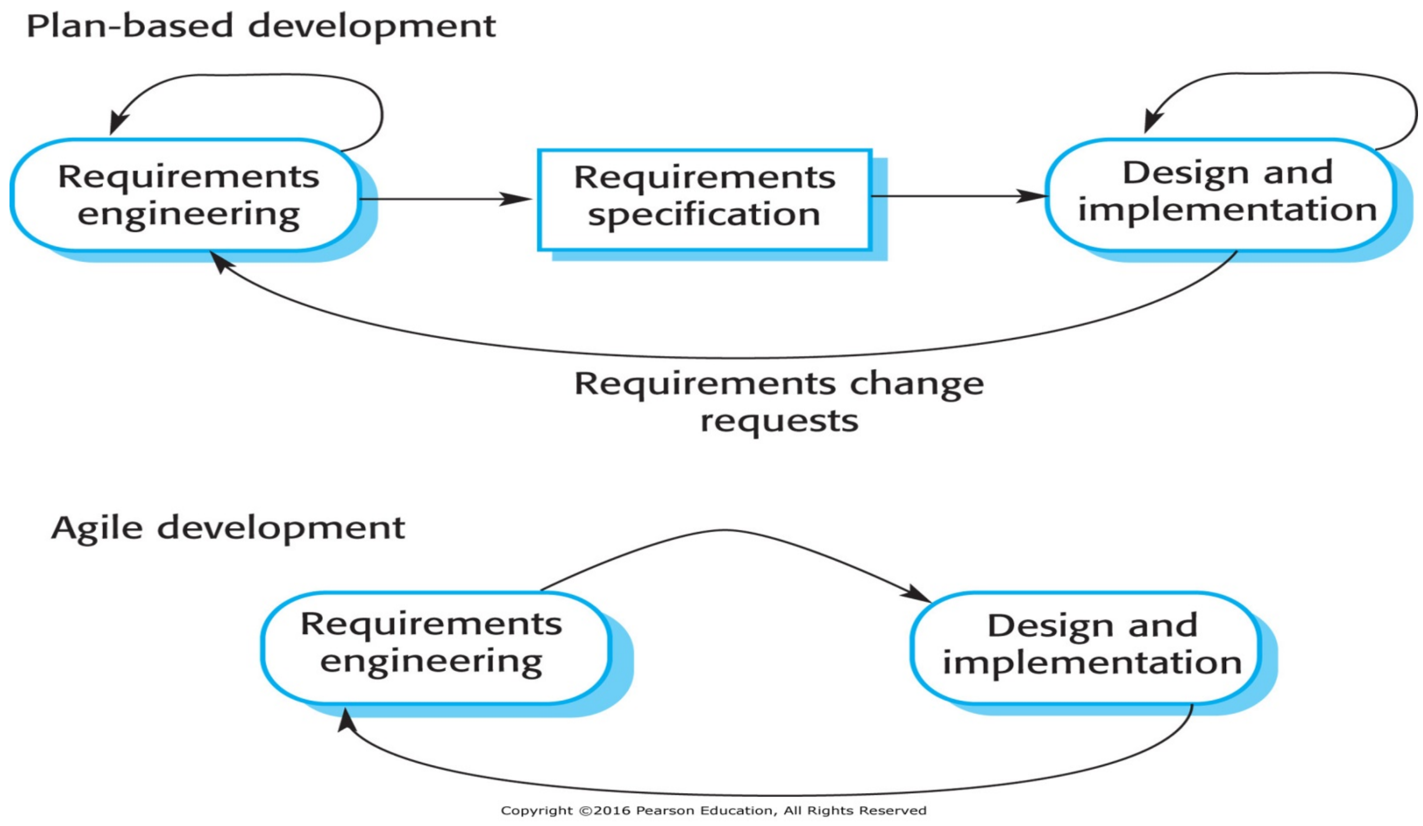
Plan-Based or Agile?

- Is it important to have a very detailed specification and design before moving to implementation?


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

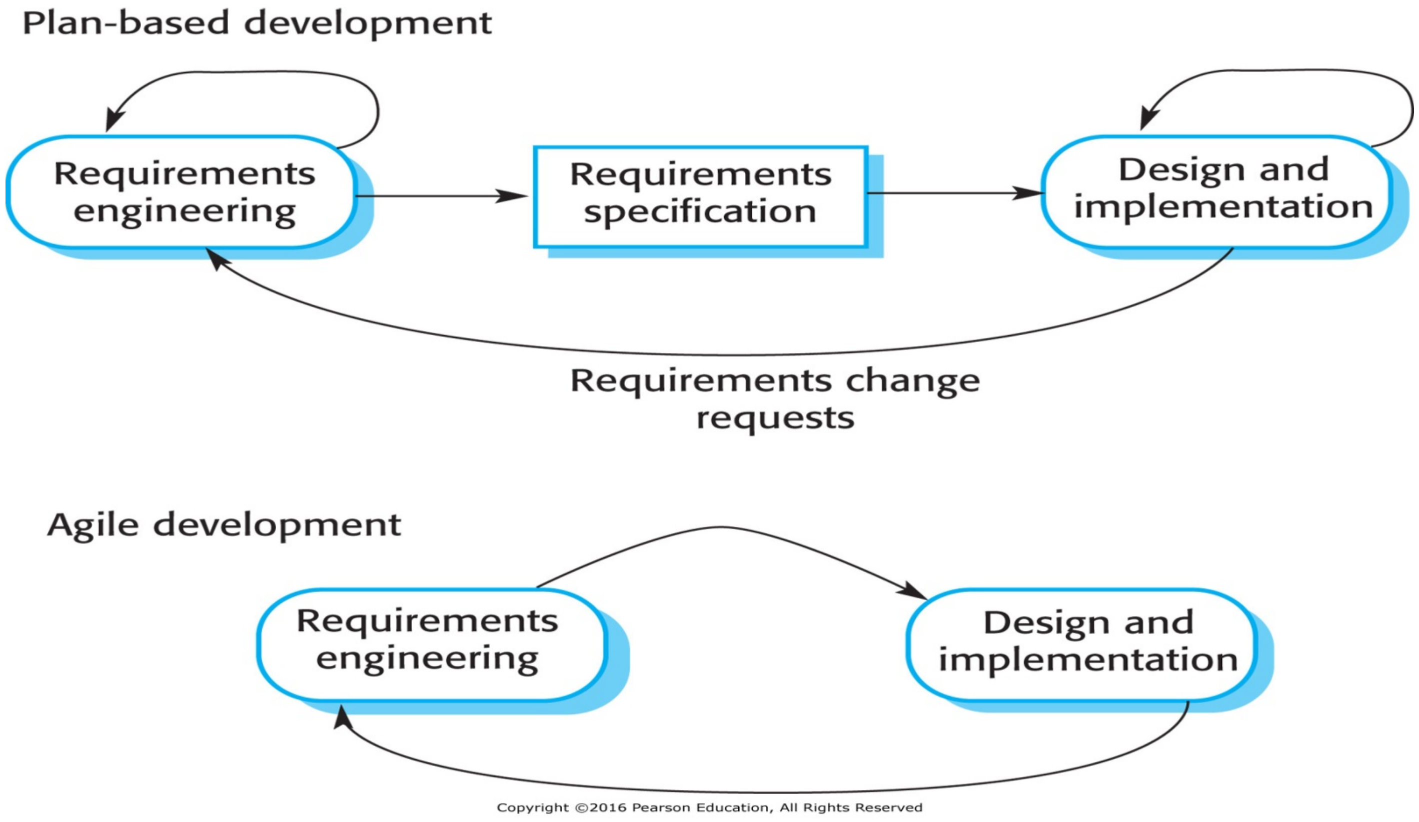
Plan-Based or Agile?

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


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

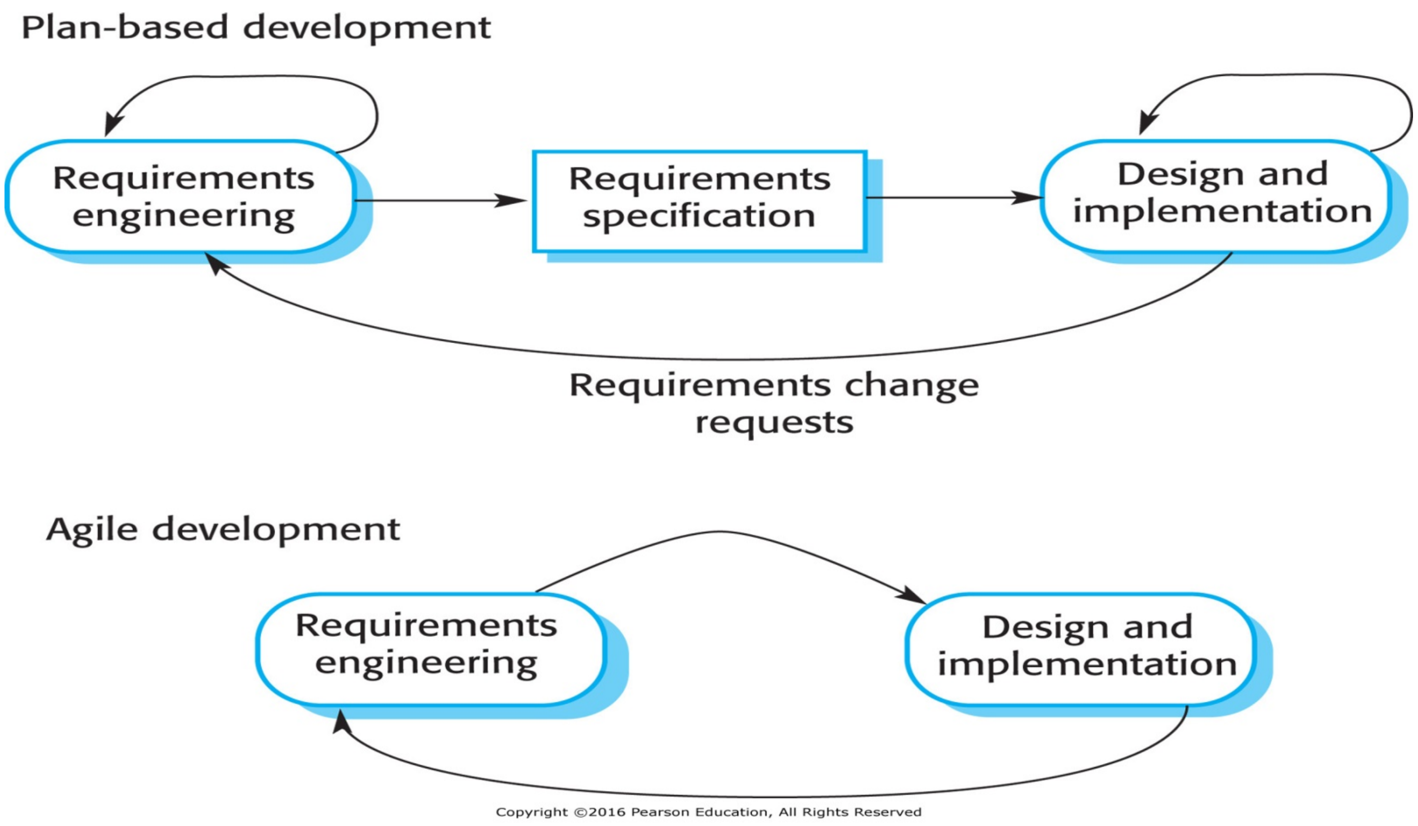
Plan-Based or Agile?

❖ How large is the system that is being developed?


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

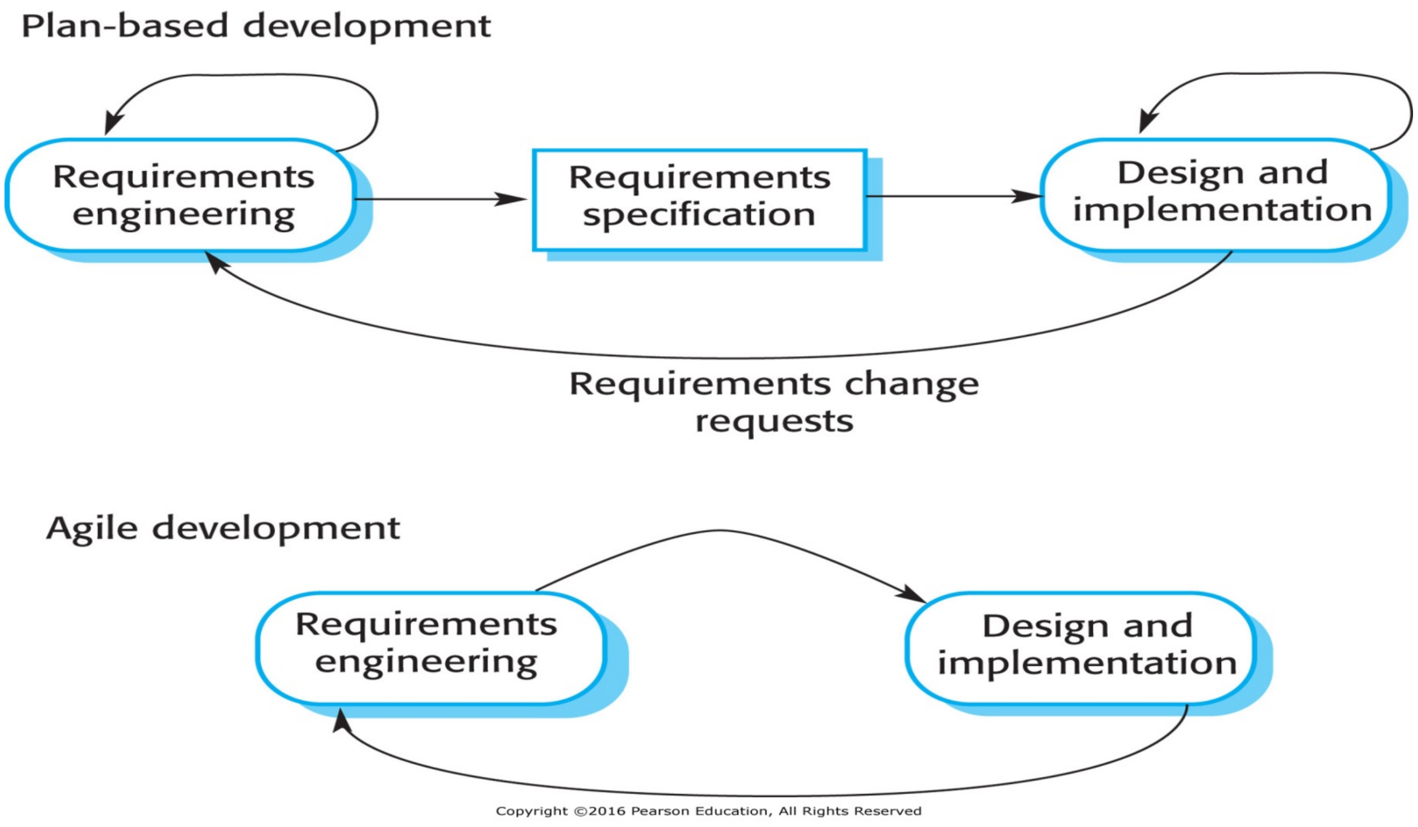
Plan-Based or Agile?

- What type of system is being developed? Or, does the system require a lot of analysis before implementation?


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

Plan-Based or Agile?

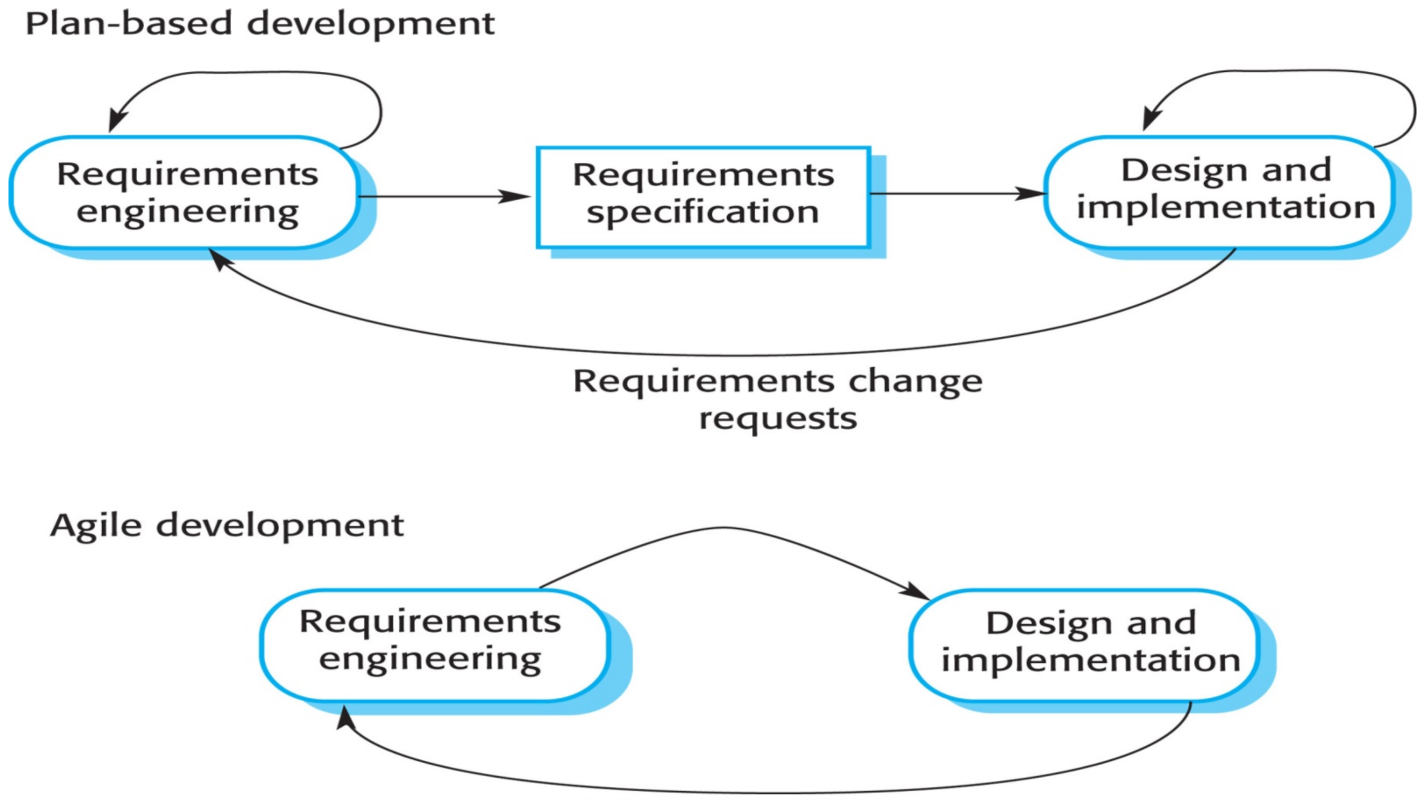
❖ What is the expected system lifetime?



Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation
Copyright ©2016 Pearson Education, All Rights Reserved

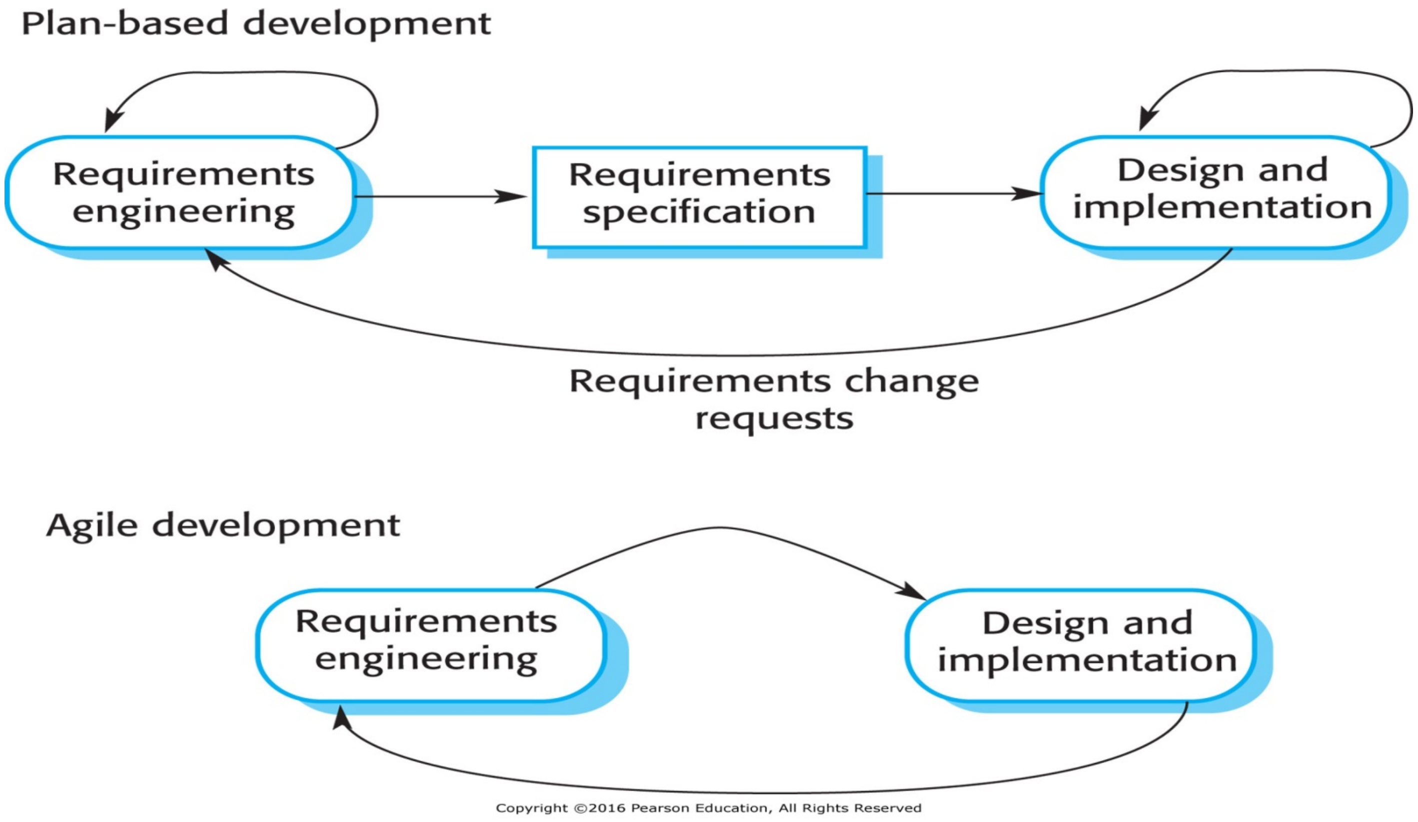
Plan-Based or Agile?

- What technologies are available to support system development? Agile methods often rely on good tools to keep track of an evolving design.


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

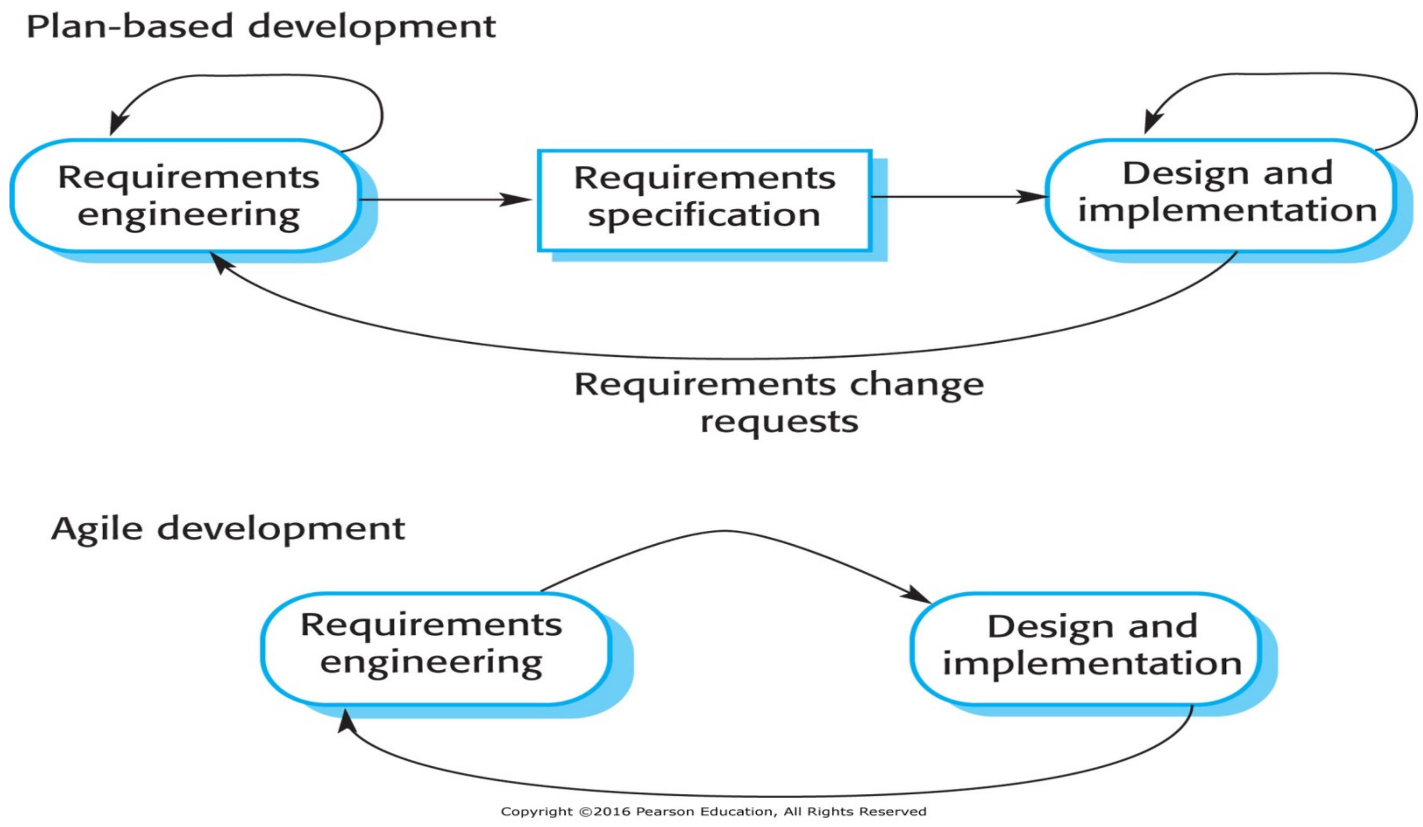
Plan-Based or Agile?

- How is the development team organized? If the development team is distributed, you may need to develop more design documents.


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation
Copyright ©2016 Pearson Education, All Rights Reserved

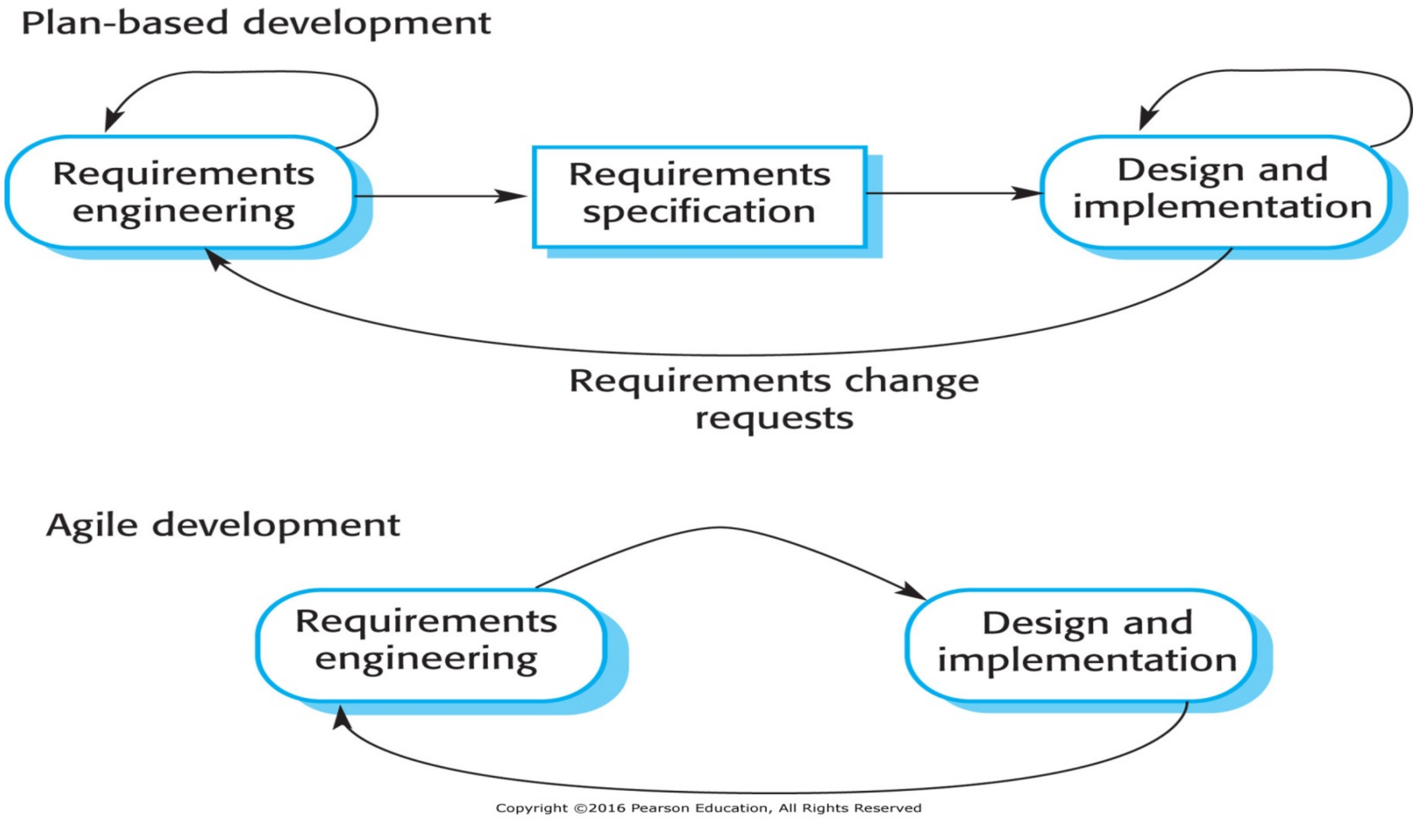
Plan-Based or Agile?

- Are there cultural issues that may affect the system development? Traditional engineering organizations have a culture of plan-based development.


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

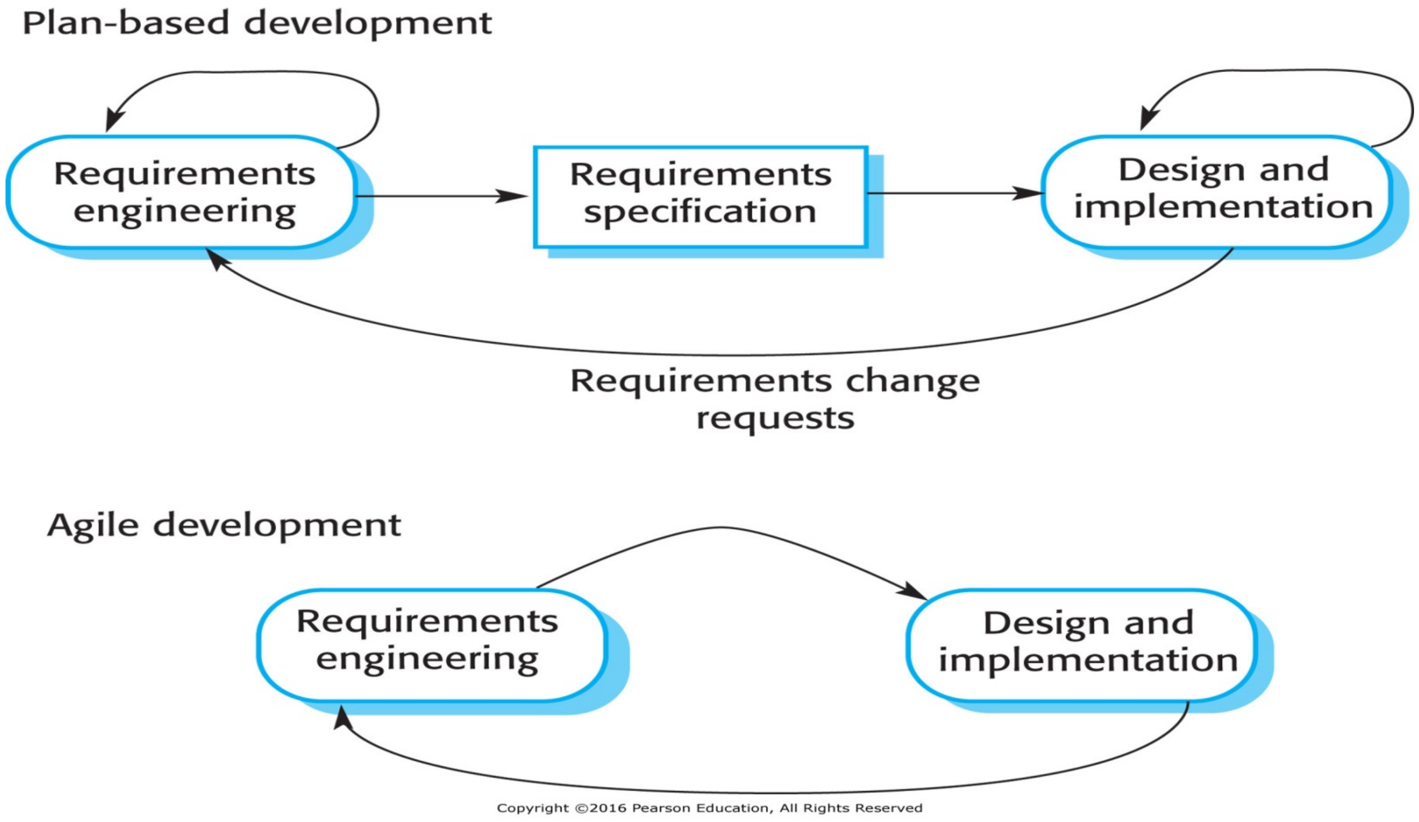
Plan-Based or Agile?

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


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

Plan-Based or Agile?

❖ Is the system subject to external regulation (e.g., Federal Aviation Authority has to approve critical aircraft software)?


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Extreme Programming: An Overview

Week 2: Agile Software Development

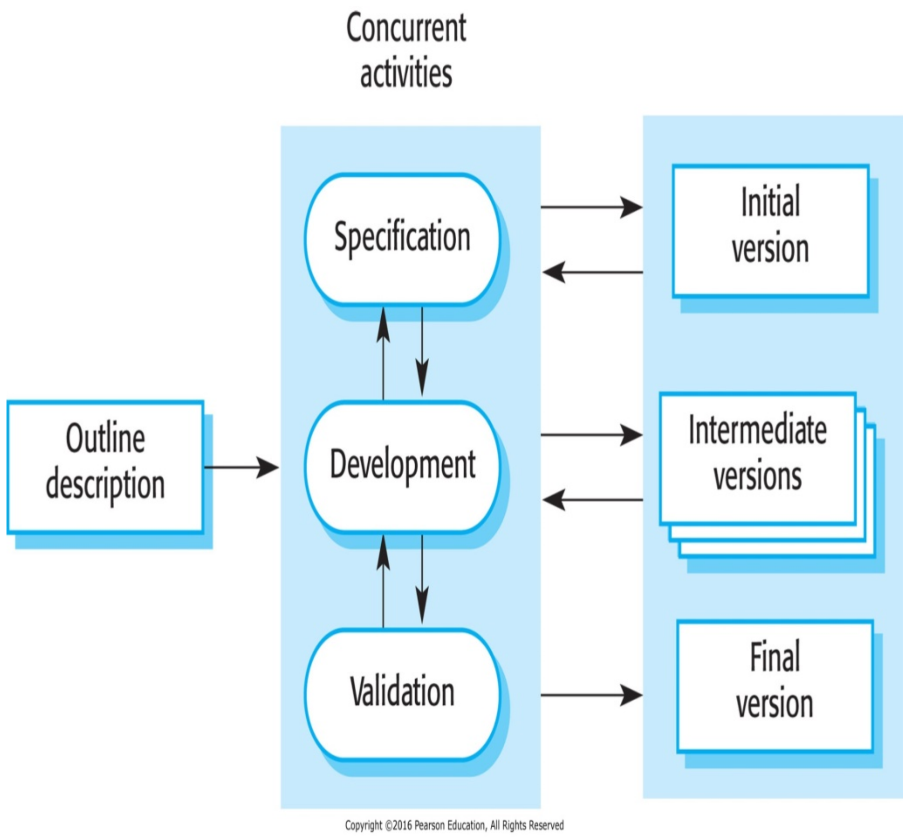
Edmund Yu, PhD
Associate Professor
esyu@syr.edu

Extreme Programming

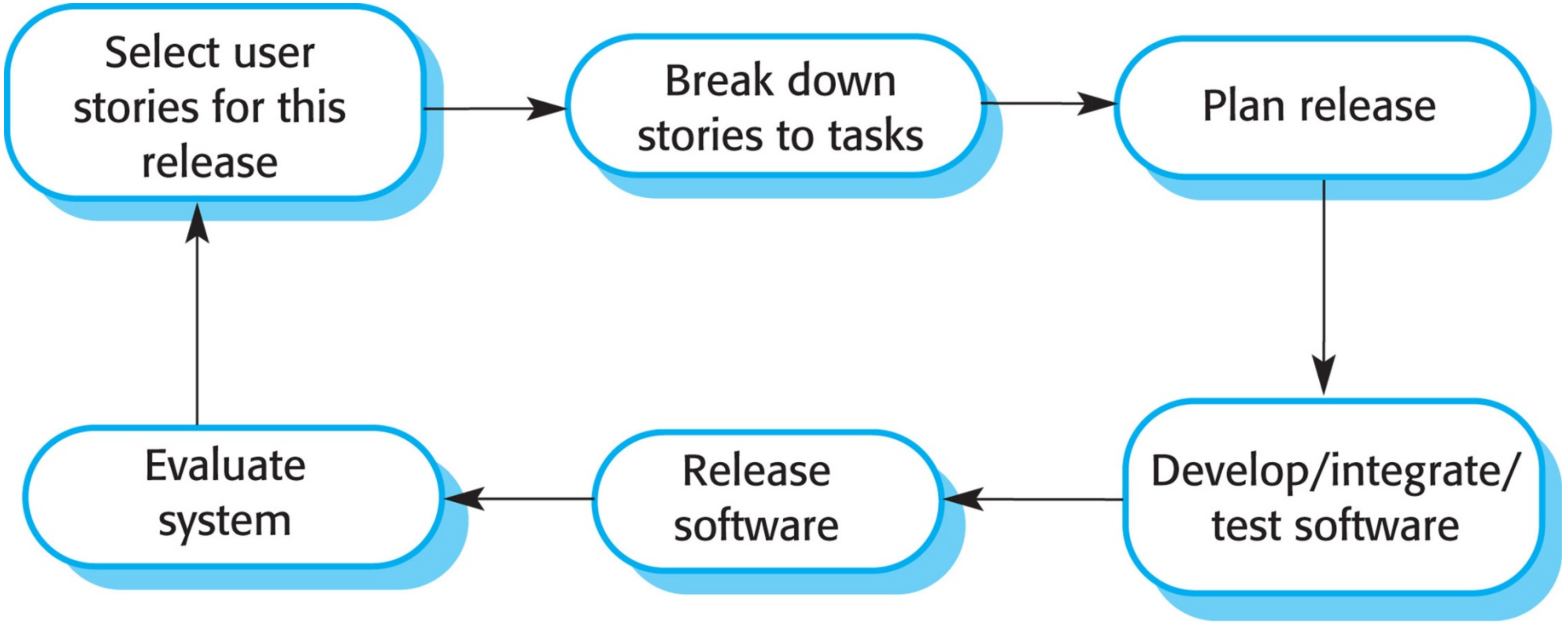
- A very influential agile method, developed in the late 1990.
  - The first extreme programming project was started March 6, 1996.
    - Kent Beck’s Chrysler Comprehensive Compensation System (C3) payroll project
  - It introduced a range of agile development techniques.
- Extreme programming (XP) is now one of the most popular agile processes.
- It has already been proven to be very successful at many companies of all different sizes and industries worldwide.

Extreme Programming (cont.)

- Extreme Programming (XP) takes an "extreme" approach to iterative/incremental development.
  - New versions may be built <u>several times per day.</u>
  - Increments are delivered to customers <u>every two weeks</u>.
  - All tests must be run for every build, and the build is only accepted if tests run successfully.



The XP Release Cycle


Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

Incremental Planning

Week 2: Agile Software Development

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

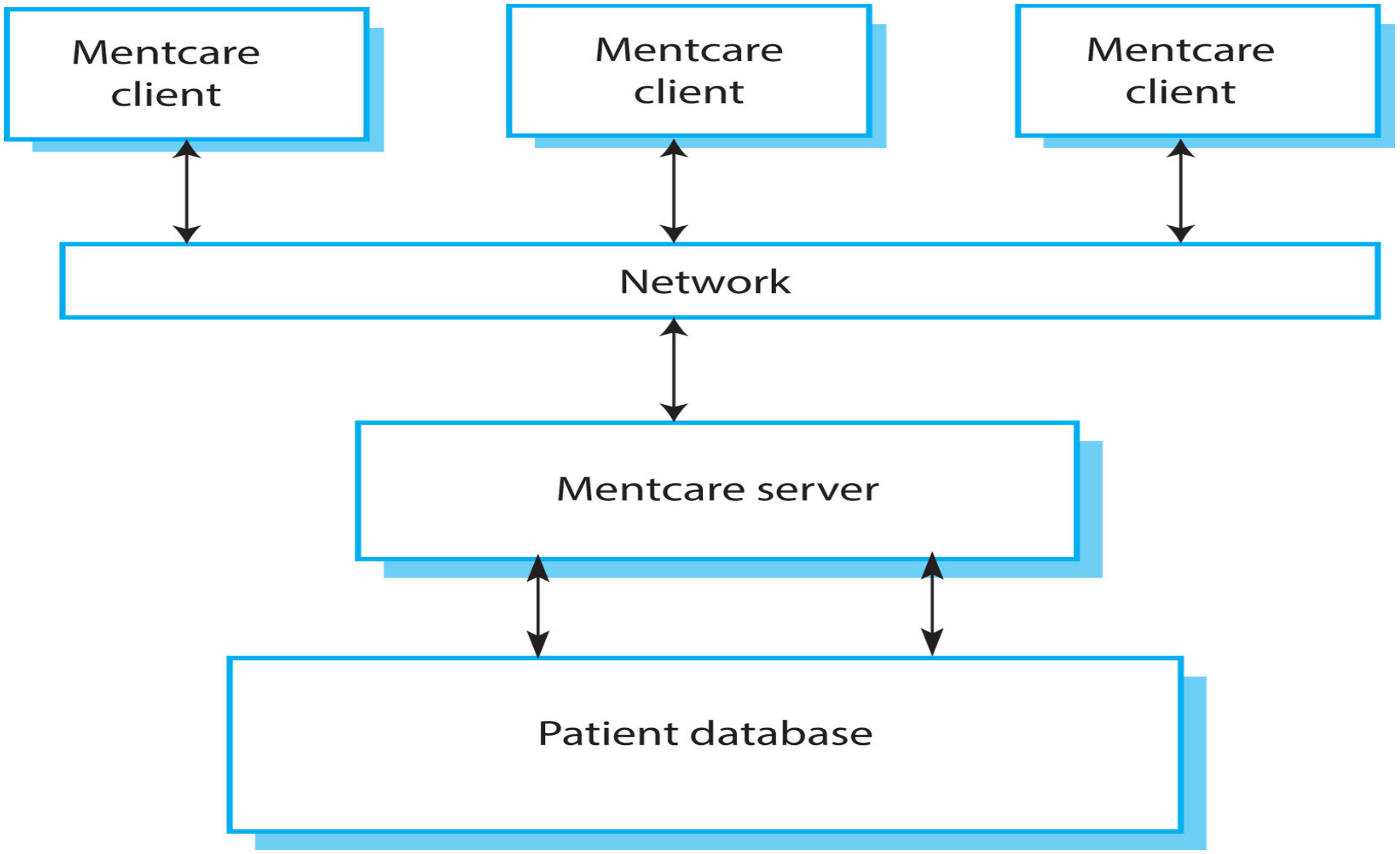
XP Practices

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.

Mentcare (Mental Health Care)

- The Mentare is a medical information system that is intended for use in clinics to support mental health care.
  - It maintains information about patients suffering from mental health problems and the treatments that they have received.
- It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database, but they can download and use local copies of patient records when they are disconnected.

The Organization of Mentcare

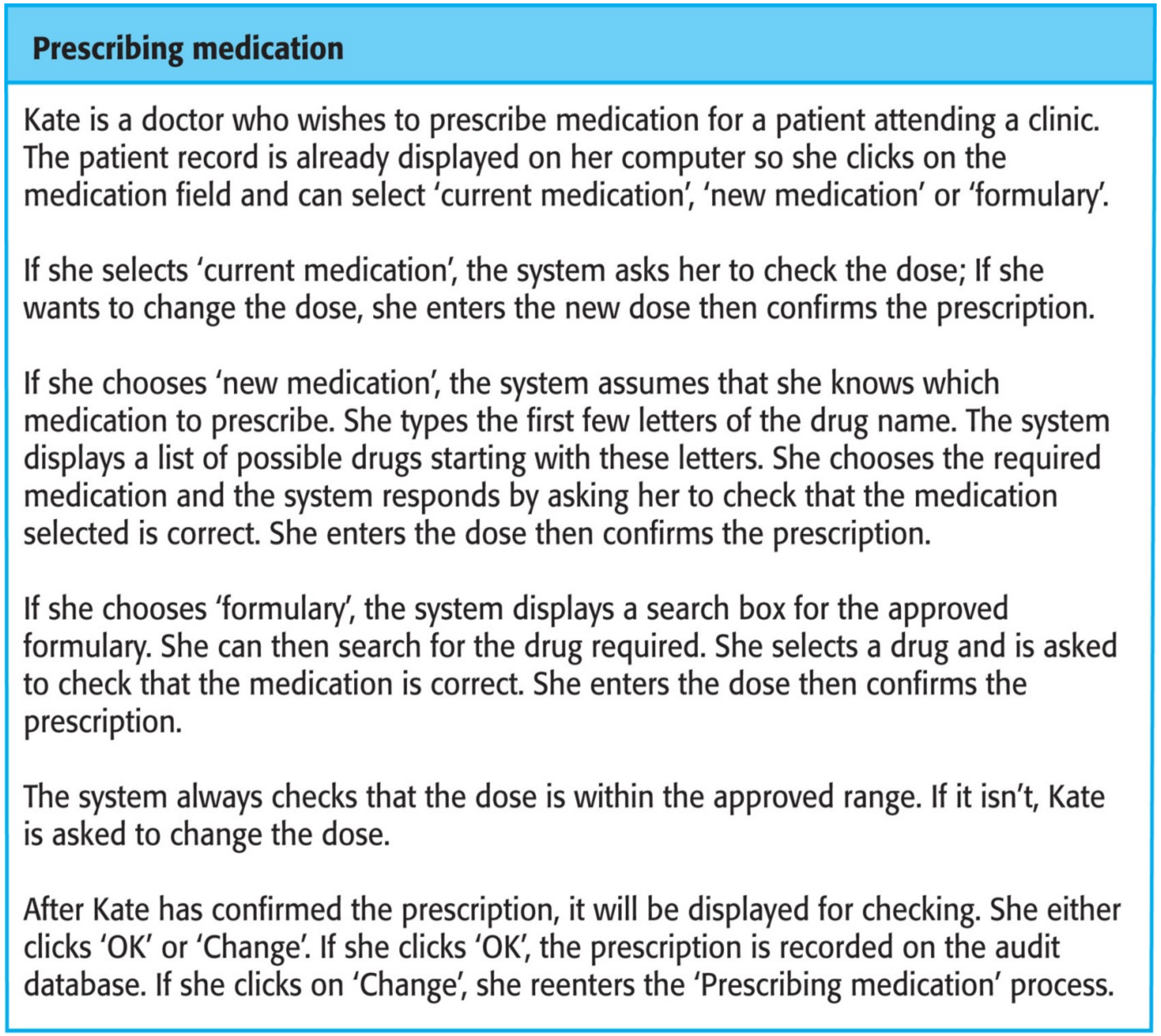


Key Features of Mentcare

- Individual care management
  - Clinicians can create records for patients, edit the information in the system, view patient history, and so on.
    - The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.
- Patient monitoring
  - The system monitors the records of patients who are involved in treatment and issues warnings if possible problems are detected.
- Administrative reporting
  - The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, and so on.

A "Prescribing Medication" Story

- 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.



Prescribing medication

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

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

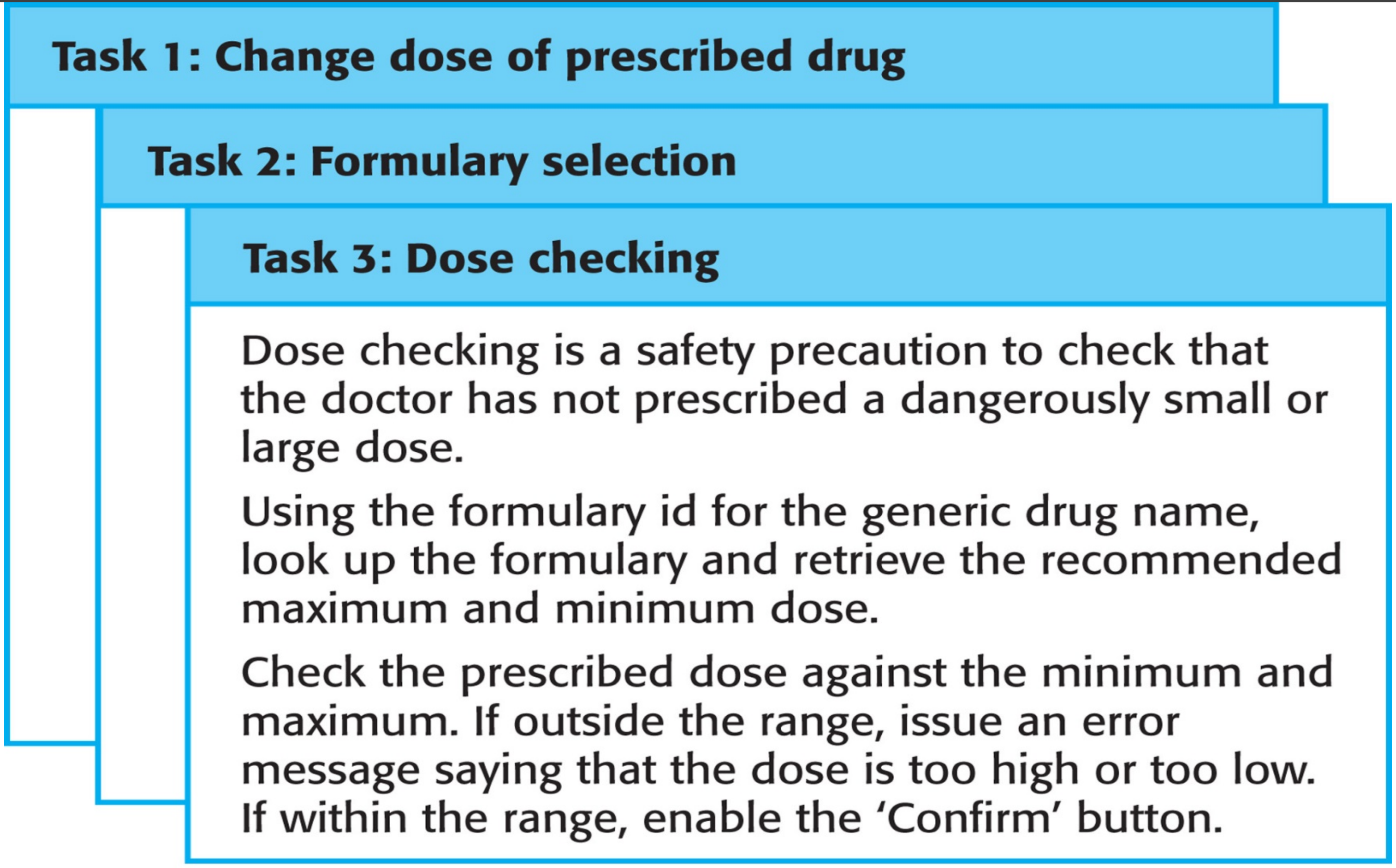
If she chooses 'new medication', the system assumes that she knows which medication to prescribe. She types the first few letters of the drug name. The system displays a list of possible drugs starting with these letters. She chooses the required medication and the system responds by asking her to check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

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

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

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

Examples of Task Cards



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.

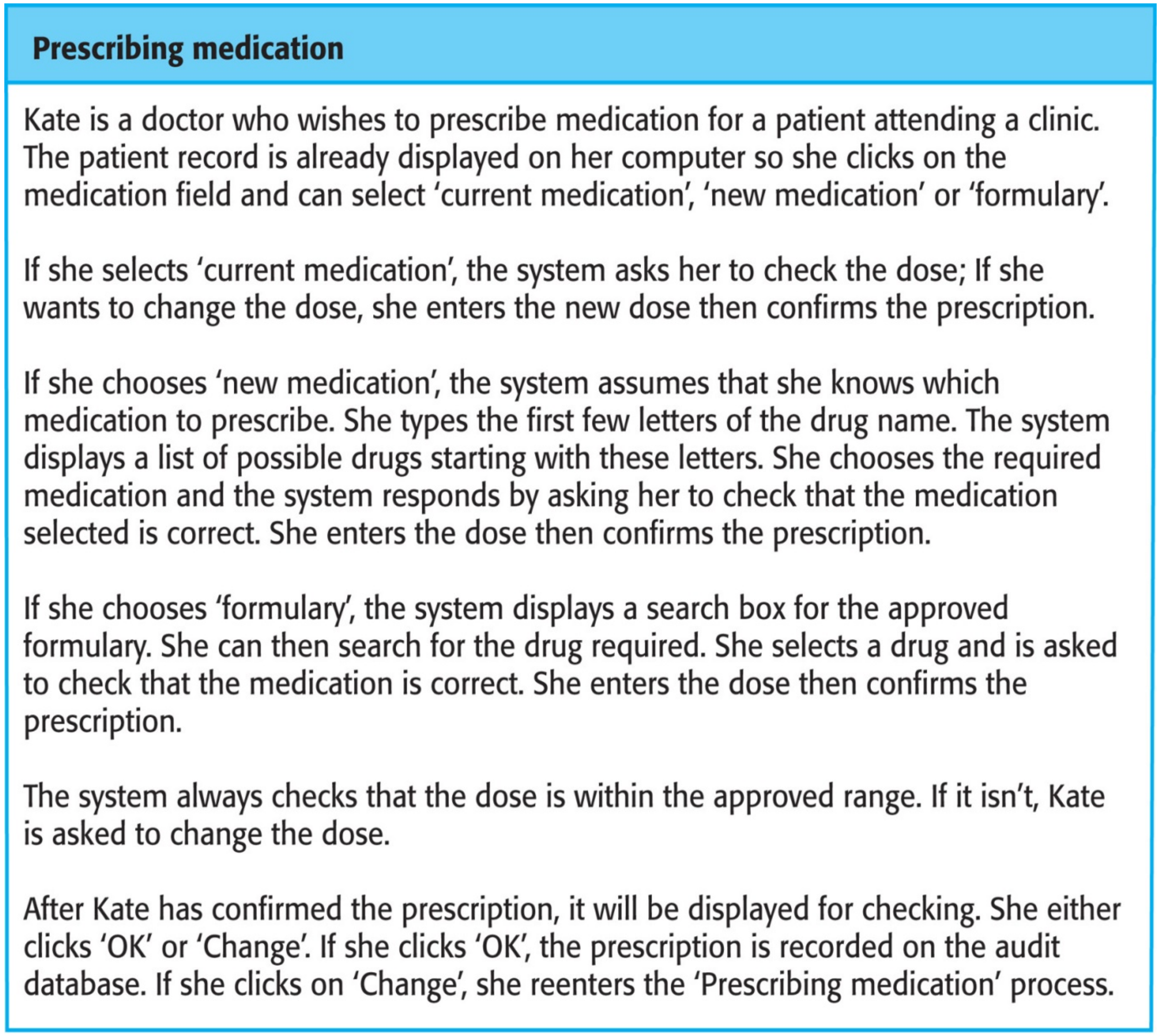
Using the formulary id for the generic drug name, look up the formulary and retrieve the recommended maximum and minimum dose.

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.

- The development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.

A “Prescribing Medication” Story

- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.



Prescribing medication

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

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

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

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

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

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

XP Practices

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.” See figures 3.5 and 3.6.
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.

XP Practices

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.” See figures 3.5 and 3.6.
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.

Test-First Development

Week 2: Agile Software Development

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

XP Practices

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.” See figures 3.5 and 3.6.
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.

Testing in XP

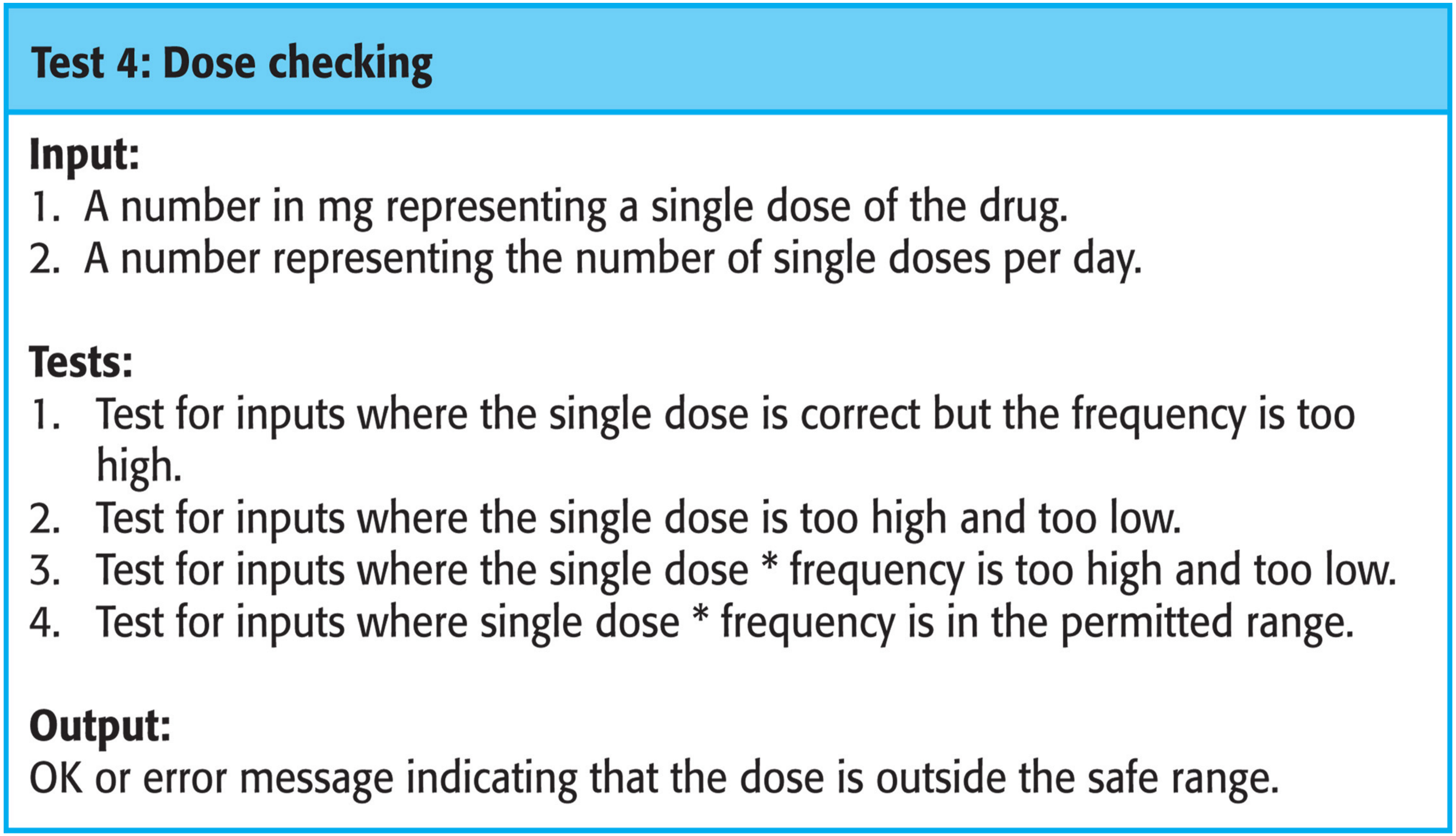
- Testing is central to XP.
  - XP has developed an approach where the program is tested after every change has been made.

- XP testing features:
  - Test-first development.
  - Incremental test development from “user stories.”
  - 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.

Customer Involvement in Testing

- 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.
- Problem: The customer has 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.

Figure 3.7 Test case description for dose checking

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.

Refactoring

Week 2: Agile Software Development

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

XP Practices

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.” See figures 3.5 and 3.6.
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.**

Refactoring

- A programming team looks 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 (self-documenting code).
  - Changes are easier to make because the code is well-structured and clear.
  - Problem: Some changes may require architecture refactoring, and this is much more expensive.

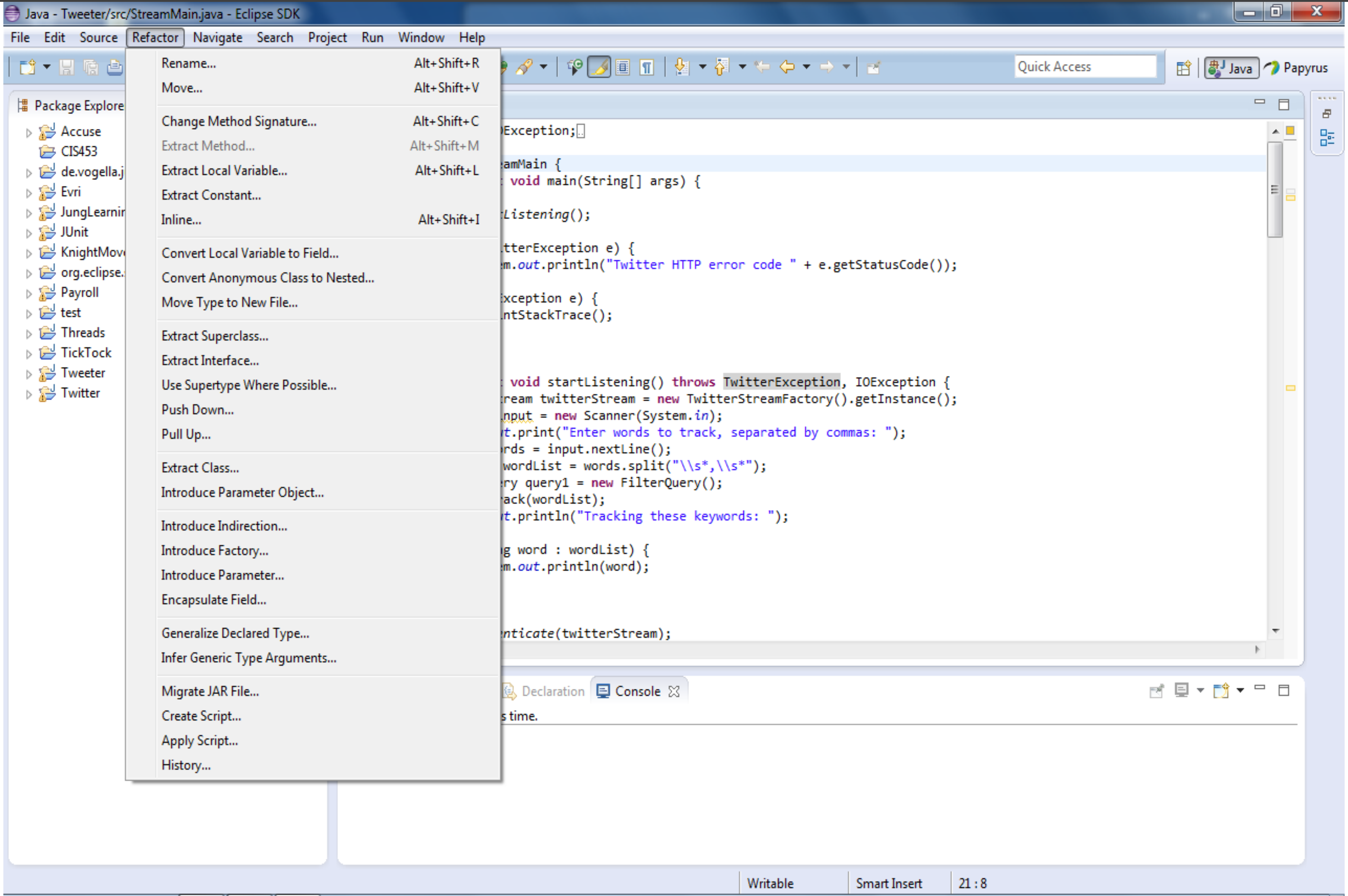
Common Examples of Refactoring

- Reorganization of a class hierarchy to remove duplicate code
- Tidying up and renaming variables/attributes and methods to make them easier to understand
  - Which of the following is easier to understand?

```
double tr = 0.0725;
double salesTaxRate = 0.0725;
```

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

More Examples of Refactoring—Eclipse



```
Exception;

amMain {
 void main(String[] args) {

Listening();

tterException e) {
m.out.println("Twitter HTTP error code " + e.getStatusCode());

xception e) {
ntStackTrace();



 void startListening() throws TwitterException, IOException {
ream twitterStream = new TwitterStreamFactory().getInstance();
nput = new Scanner(System.in);
t.print("Enter words to track, separated by commas: ");
rds = input.nextLine();
wordList = words.split("\\s*,\\s*");
ry query1 = new FilterQuery();
ack(wordList);
t.println("Tracking these keywords: ");

g word : wordList) {
m.out.println(word);



nticate(twitterStream);
```

```
s time.
```

Examples of Refactoring—Eclipse

Name	Description
Rename	Renames the selected element, and (if enabled) corrects all references to the elements (also in other files).
Move	Moves the selected elements, and (if enabled) corrects all references to the elements (also in other files).
Extract Method	Creates a new method containing the statements or expression currently selected, and replaces the selection with a reference to the new method.
Extract Local Variable	Creates a new variable assigned to the expression currently selected, and replaces the selection with a reference to the new variable.
Extract Constant	Creates a static final field from the selected expression, and substitutes a field reference, and optionally rewrites other places where the same expression occurs.
Extract Superclass	Extracts a common superclass from a set of sibling types. The selected sibling types become direct subclasses of the extracted superclass after applying the refactoring.
Extract Interface	Creates a new interface with a set of methods, and makes the selected class implement the interface.
Extract Class	Replaces a set of fields with a new container object. All references to the fields are updated to access the new container object.

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Pair Programming

Week 2: Agile Software Development

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

XP Practices

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.

Pair Programming

- Two programmers work together at one machine.
- **Driver** enters code, while **navigator** critiques it.
- Periodically switch roles.

- Research results:
  - Measurements suggest that development productivity with pair programming is similar to that of two people working independently.
  - Williams, L., R. Kessler, W. Cunningham, and R. Jeffries, R. “Strengthening the Case for Pair Programming.” *IEEE Software* 17 no. 3 (July/August 2000).
    - Higher quality code (15 percent fewer defects) in about half the time (58%).
  - Requires proximity in lab or work environment.

Pair Programming (cont.)

- Experiment at NC State
  - CS1—programming in Java
  - Two sections, same instructor, same exams
  - 69 in solo programming section, 44 in paired section
- Results:
  - 68 percent of paired students got C or better vs. 45 percent of solo students.
  - Paired students performed much better (16 to 18 points better) on first two projects.
  - No difference on third project (perhaps because lower-performing solo students had dropped before the third project).
  - Midterm exam: 65.8 vs. 49.5   Final exam: 74.1 vs. 67.2.
  - Course and instructor evaluations were higher for paired students.
  - Similar results at UC Santa Cruz (86 vs. 67 on programs).

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 an important 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.

XP Practices

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.

More XP Practices

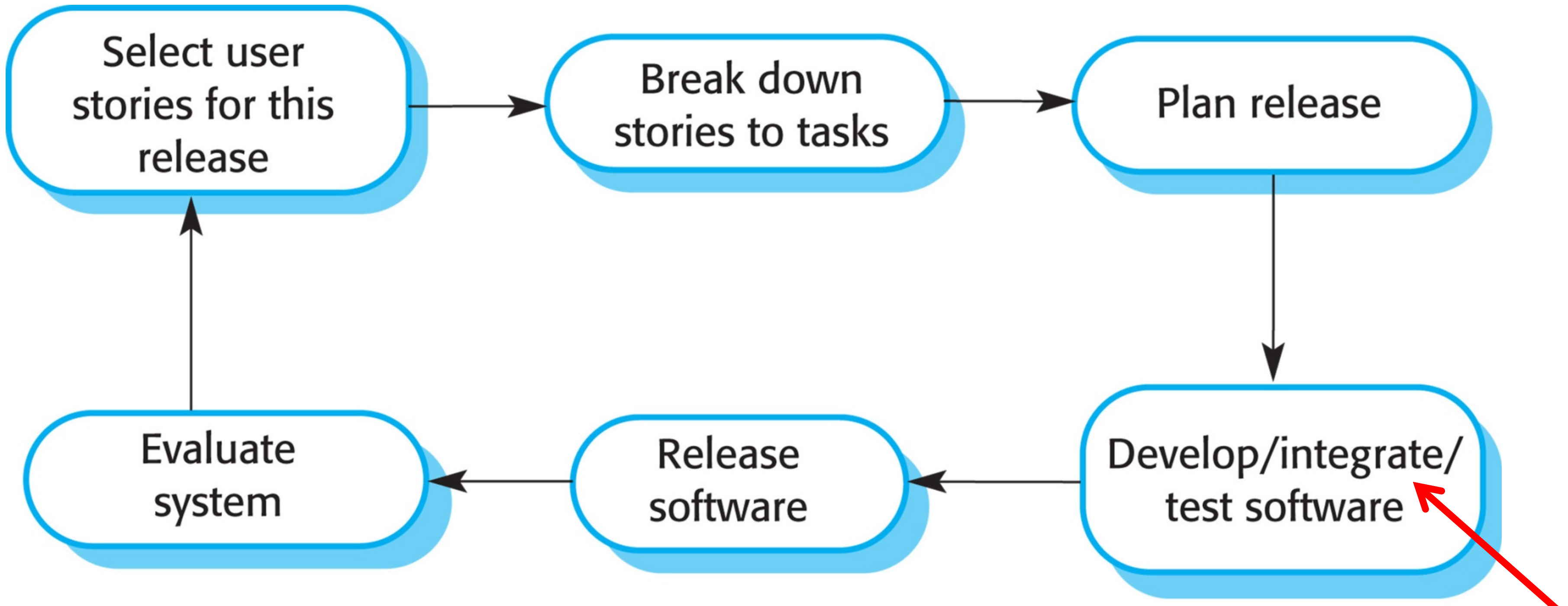
Week 2: Agile Software Development

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

XP Practices

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 XP Release Cycle

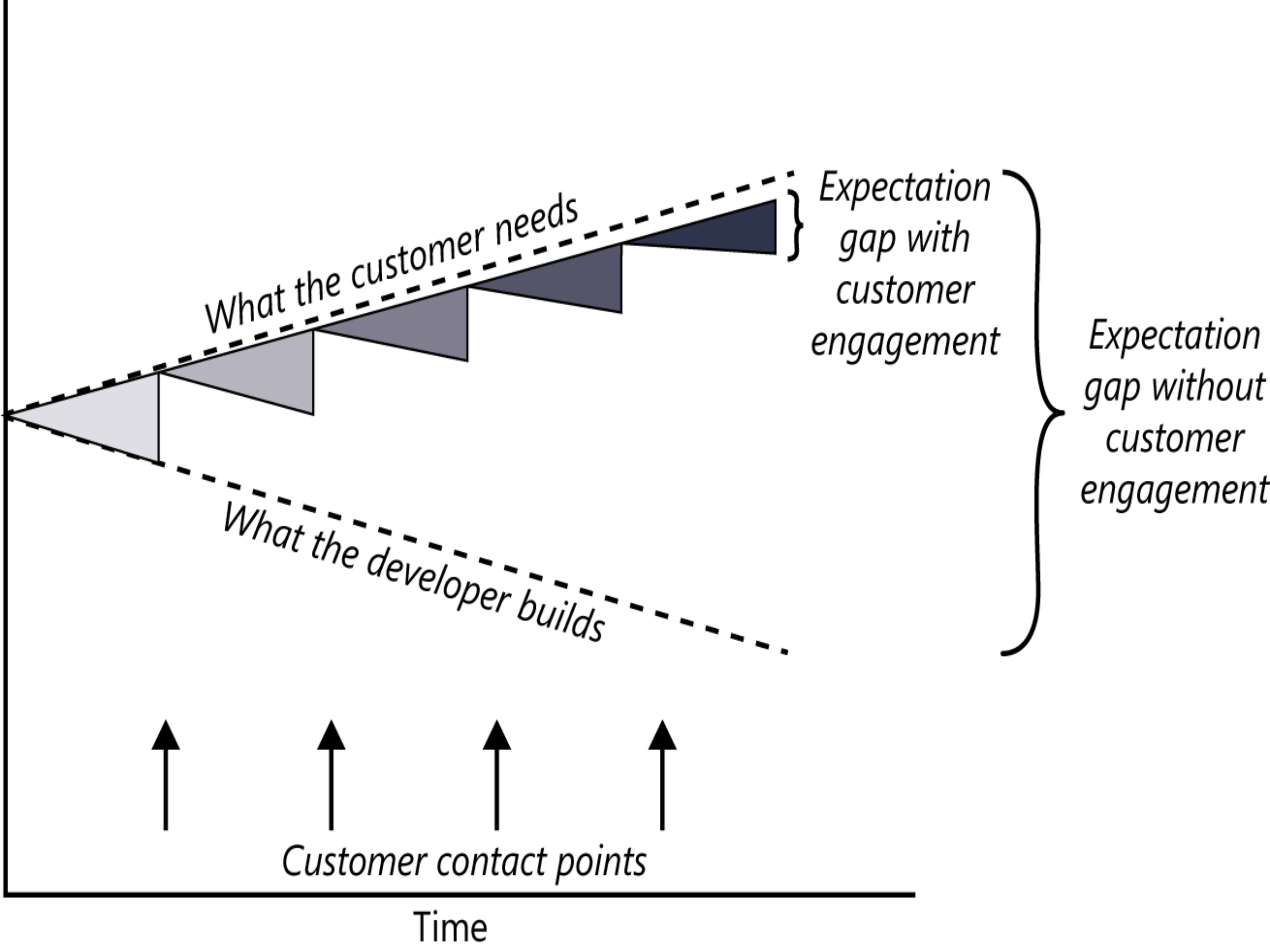

Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

XP Practices

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.

XP Practices

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.


What the customer needs
What the developer builds
Expectation gap with customer engagement
Expectation gap without customer engagement
Customer contact points
Time

Scrum

Week 2: Agile Software Development

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY ENGINEERING & COMPUTER SCIENCE

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 last phase wraps up the project, completes required documentation such as system help frames and user manuals, and assesses the lessons learned from the project.

Figure 3.8 The Scrum process

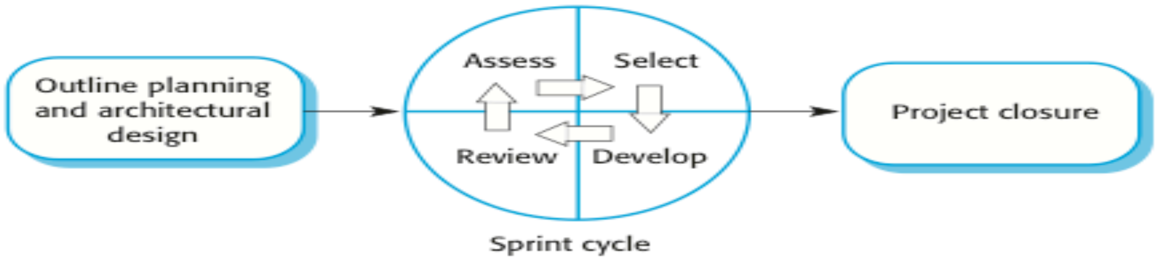


Figure 3.3 The XP release cycle (for comparison)

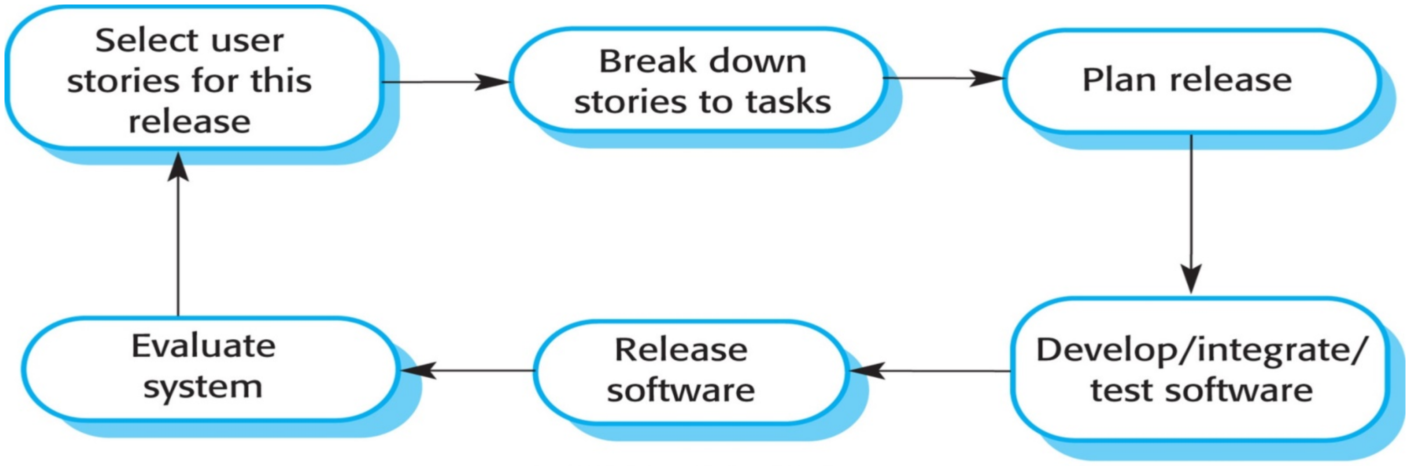
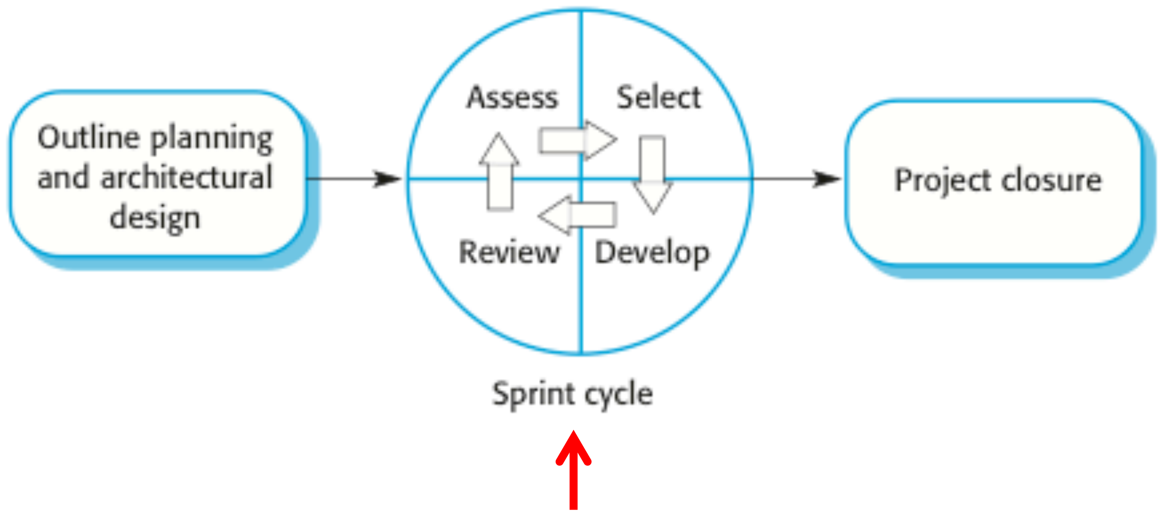

Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

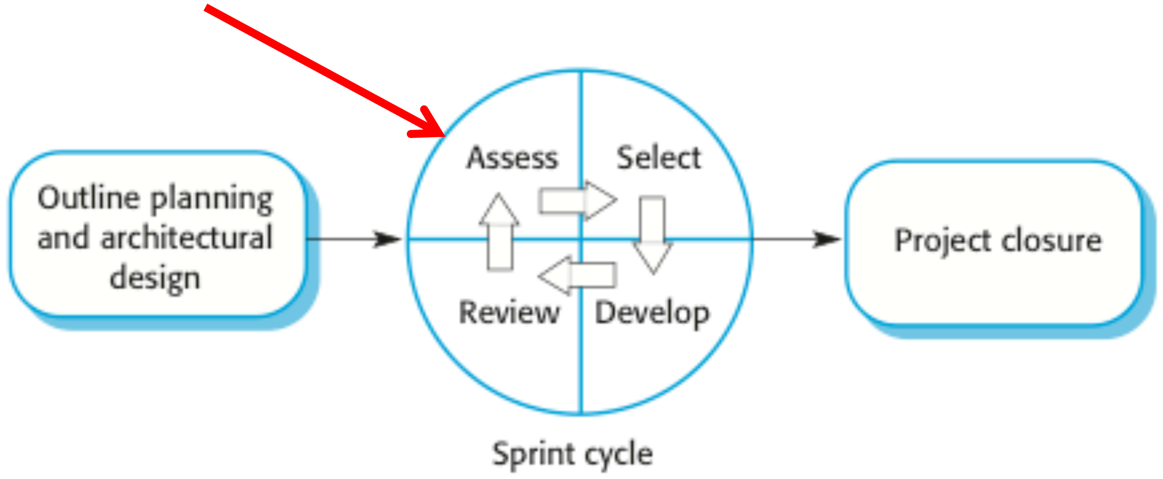
Figure 3.8 The Scrum process



- ❖ Sprints are fixed length, normally two to four weeks.
- ❖ They correspond to the development of a release of the system in XP.

Figure 3.8 The Scrum process

- The starting point for planning is the **product backlog**, which is the list of work to be done on the project.

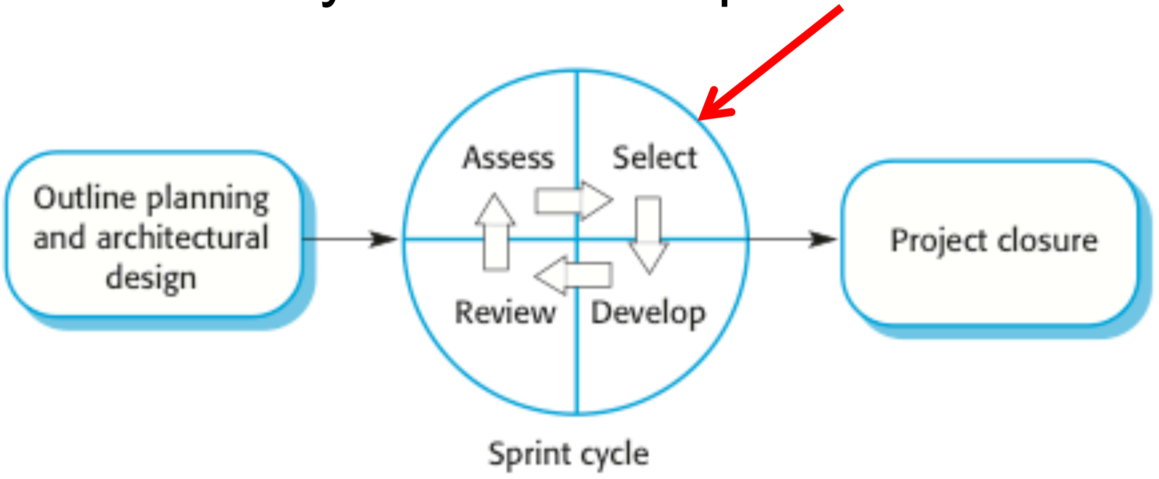


Scrum Terminology (A)

Scrum Term	**Definition**
Development team	A self-organizing group of software developers, which should be no more than seven people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be "potentially shippable," which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of "to-do" items which the scrum team must tackle. They may be feature definitions for the software, software requirements, user stories, or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The product owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

Figure 3.8 The Scrum process

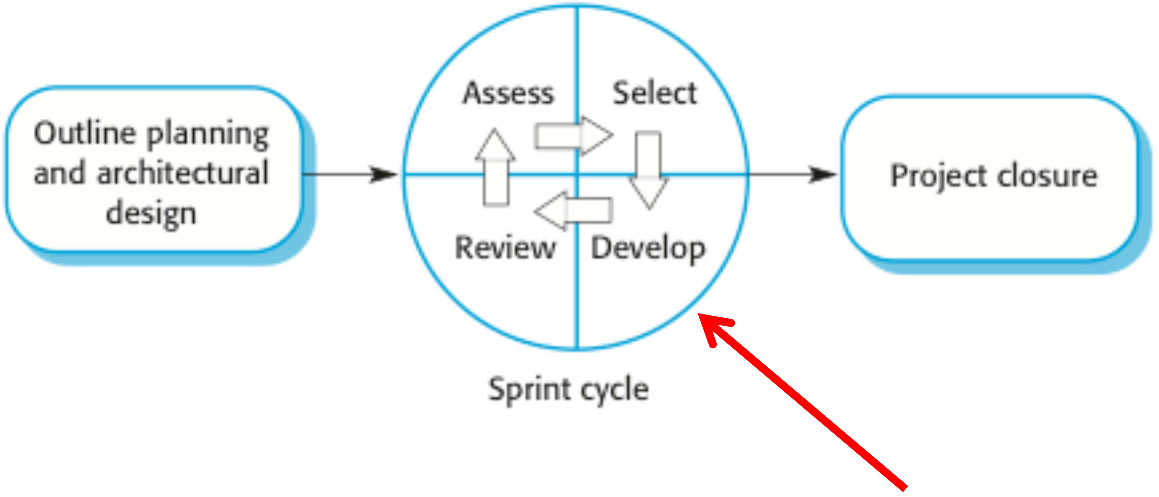
- The selection phase involves all of the **development team** who work with the customer to select the features/functionality to be developed.



Scrum Terminology (A)

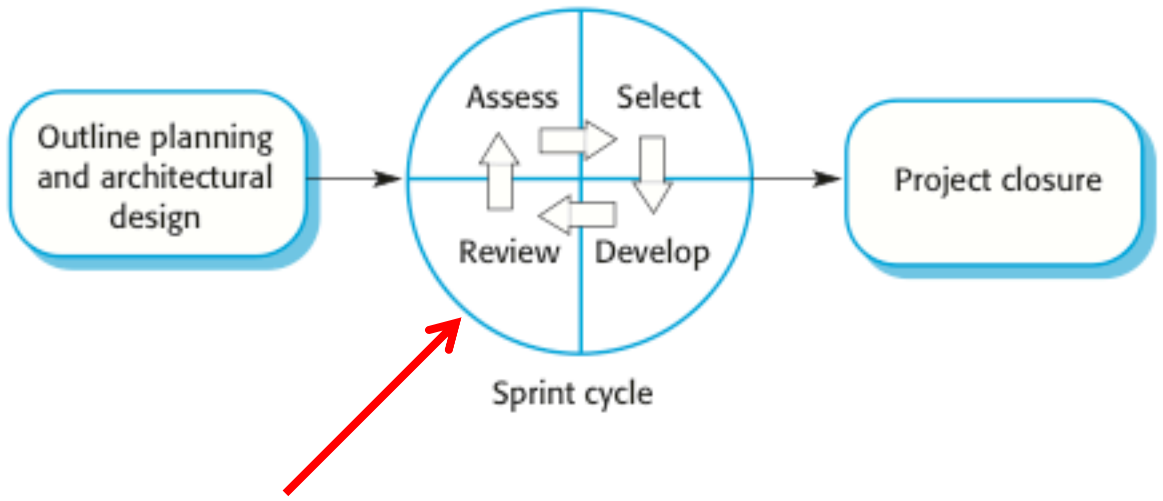
Scrum Term	Definition
Development team	A self-organizing group of software developers, which should be no more than seven people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be “potentially shippable,” which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of “to-do” items which the scrum team must tackle. They may be feature definitions for the software, software requirements, user stories, or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The product owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

Figure 3.8 The Scrum process



- ❖ Once these are agreed, the team organize themselves to develop the software.

Figure 3.8 The Scrum process



- At the end of the sprint, the work done is reviewed and presented to stakeholders.
- The next sprint cycle then begins.

Sprint Cycle: A More Detailed View

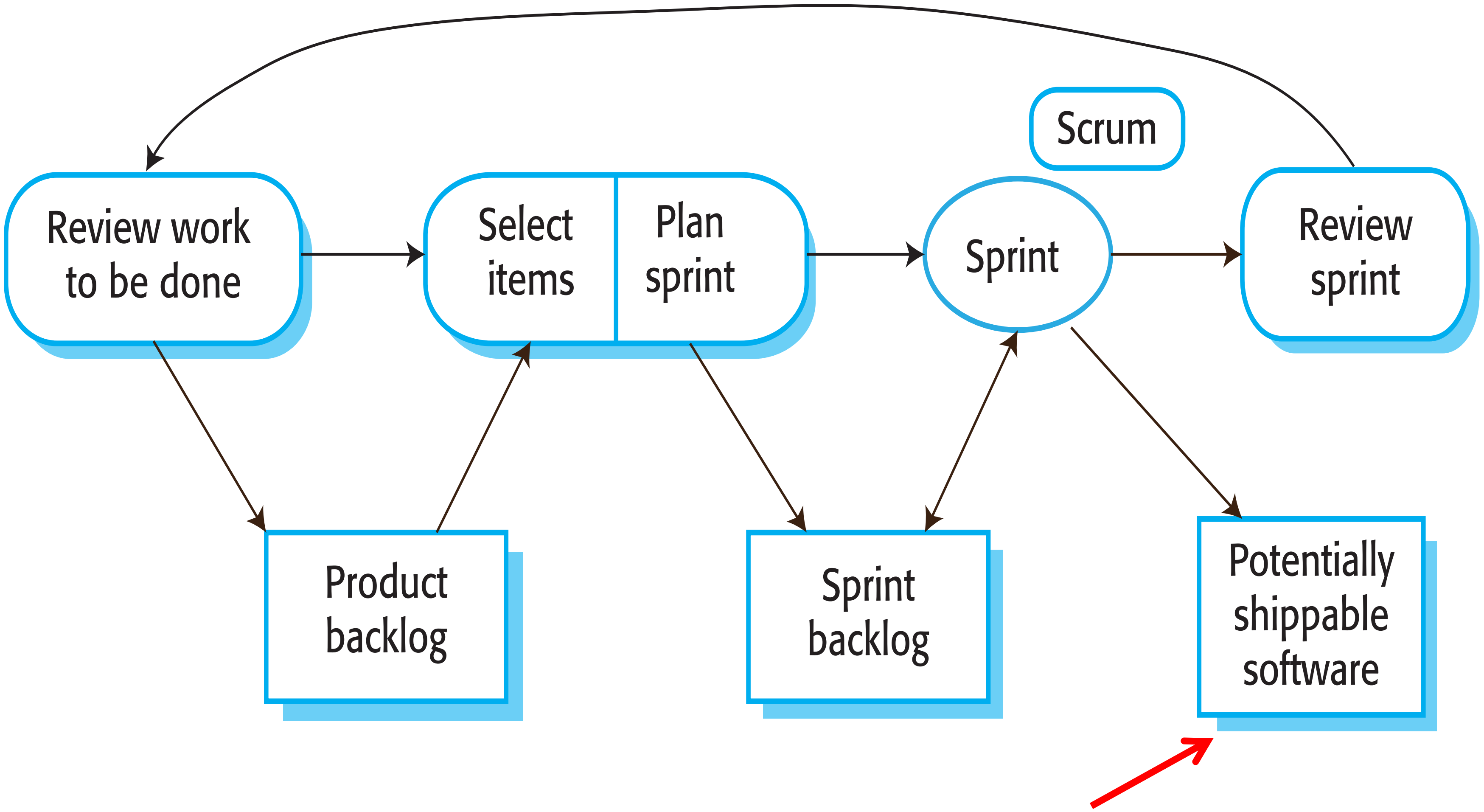


Figure 3.9 The Scrum sprint cycle

Scrum Terminology (A)

Scrum Term	**Definition**
Development team	A self-organizing group of software developers, which should be no more than **seven** people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be “potentially shippable,” which means that **it is in a finished state** and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of “to-do” items which the scrum team must tackle. They may be feature definitions for the software, software requirements, user stories, or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The product owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

Sprint Cycle: A More Detailed View

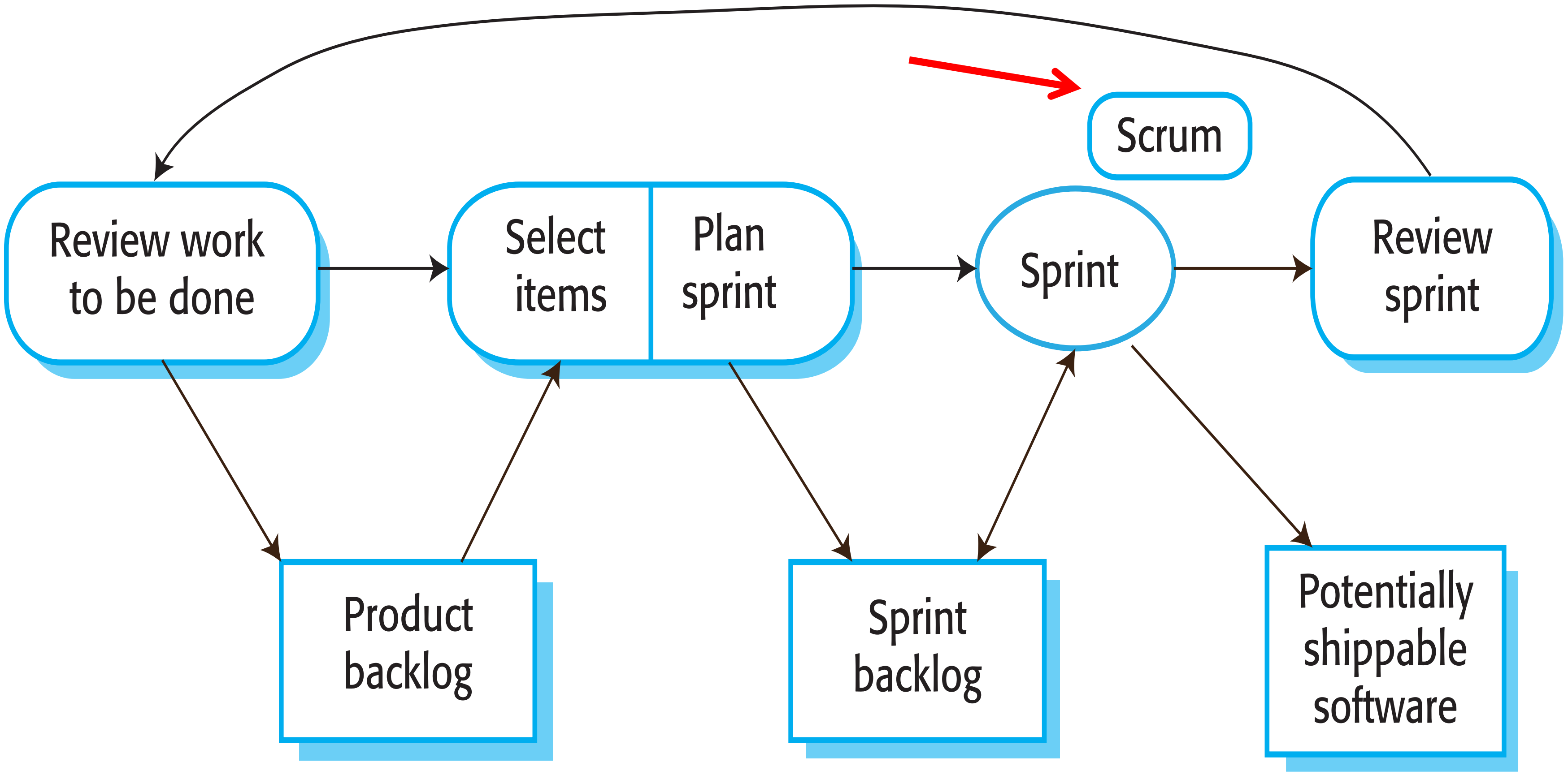


Figure 3.9 The Scrum sprint cycle

Scrum Terminology (B)

Scrum Term	Definition
Scrum	**A daily meeting** of the scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
Scrum master	The scrum master is responsible for ensuring that the scrum process is followed and guides the team in the effective use of scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the scrum team is not diverted by outside interference. The scrum developers are adamant that **the scrum master should not be thought of as a project manager**. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually two to four weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team’s velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

Scrum Terminology (B)

Scrum Term	**Definition**
Scrum	**A daily meeting** of the scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
Scrum master	The scrum master is responsible for ensuring that the scrum process is followed and guides the team in the effective use of scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the scrum team is not diverted by outside interference. The scrum developers are adamant that **the scrum master should not be thought of as a project manager**. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually two to four weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

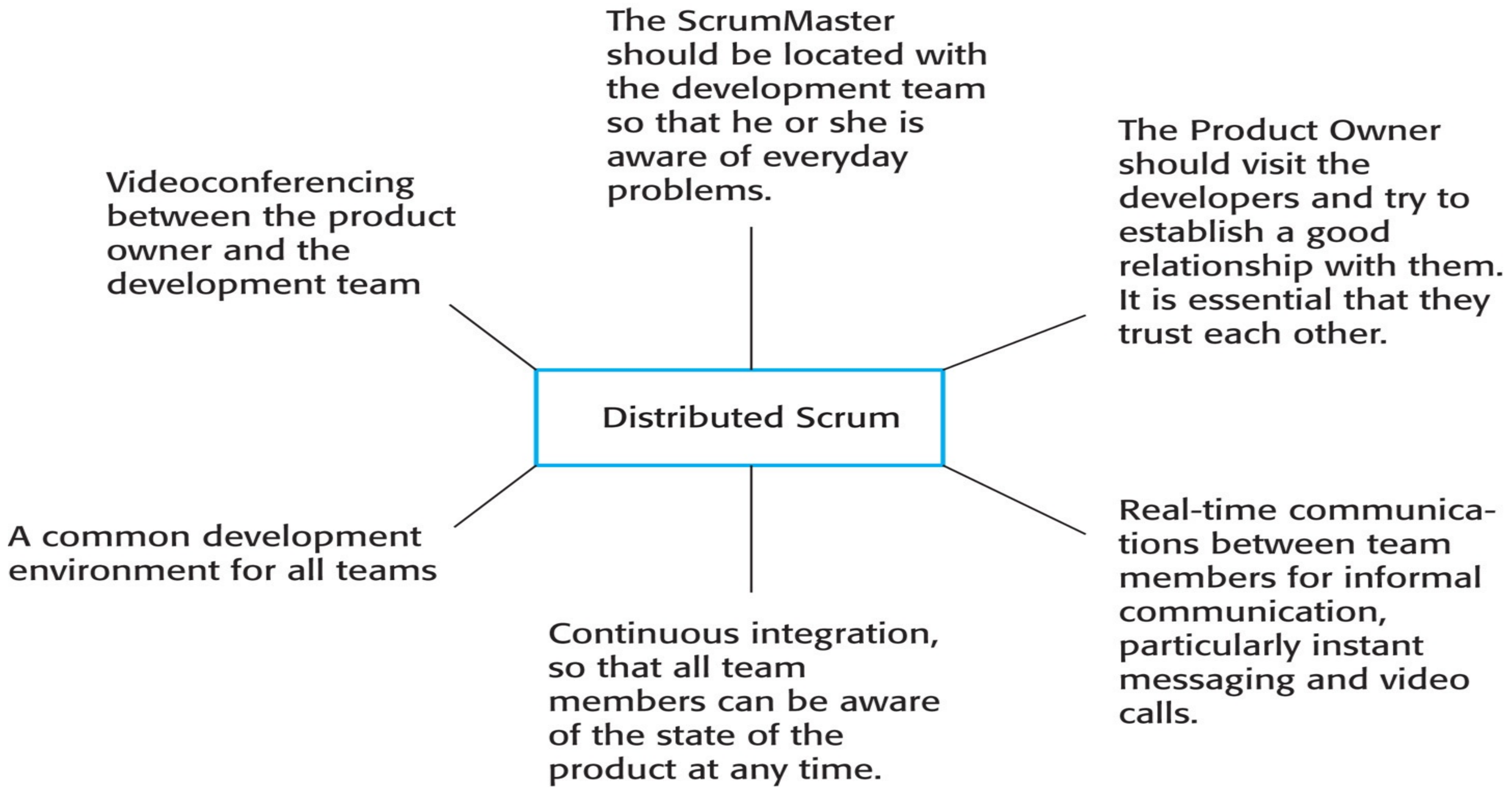
The Scrum Master

- During the development 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.
    - The scrum master also 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.

Teamwork in Scrum: Daily Meetings

- The whole team attends <u>short daily meetings</u>.
- In these meetings, all team members share information and 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 replan short-term work to cope with them.

Distributed Scrum


The ScrumMaster should be located with the development team so that he or she is aware of everyday problems.
Videoconferencing between the product owner and the development team
The Product Owner should visit the developers and try to establish a good relationship with them. It is essential that they trust each other.
Distributed Scrum
A common development environment for all teams
Real-time communications between team members for informal communication, particularly instant messaging and video calls.
Continuous integration, so that all team members can be aware of the state of the product at any time.

Scaling Agile Methods

Week 2: Agile Software Development

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

Scaling Agile Methods

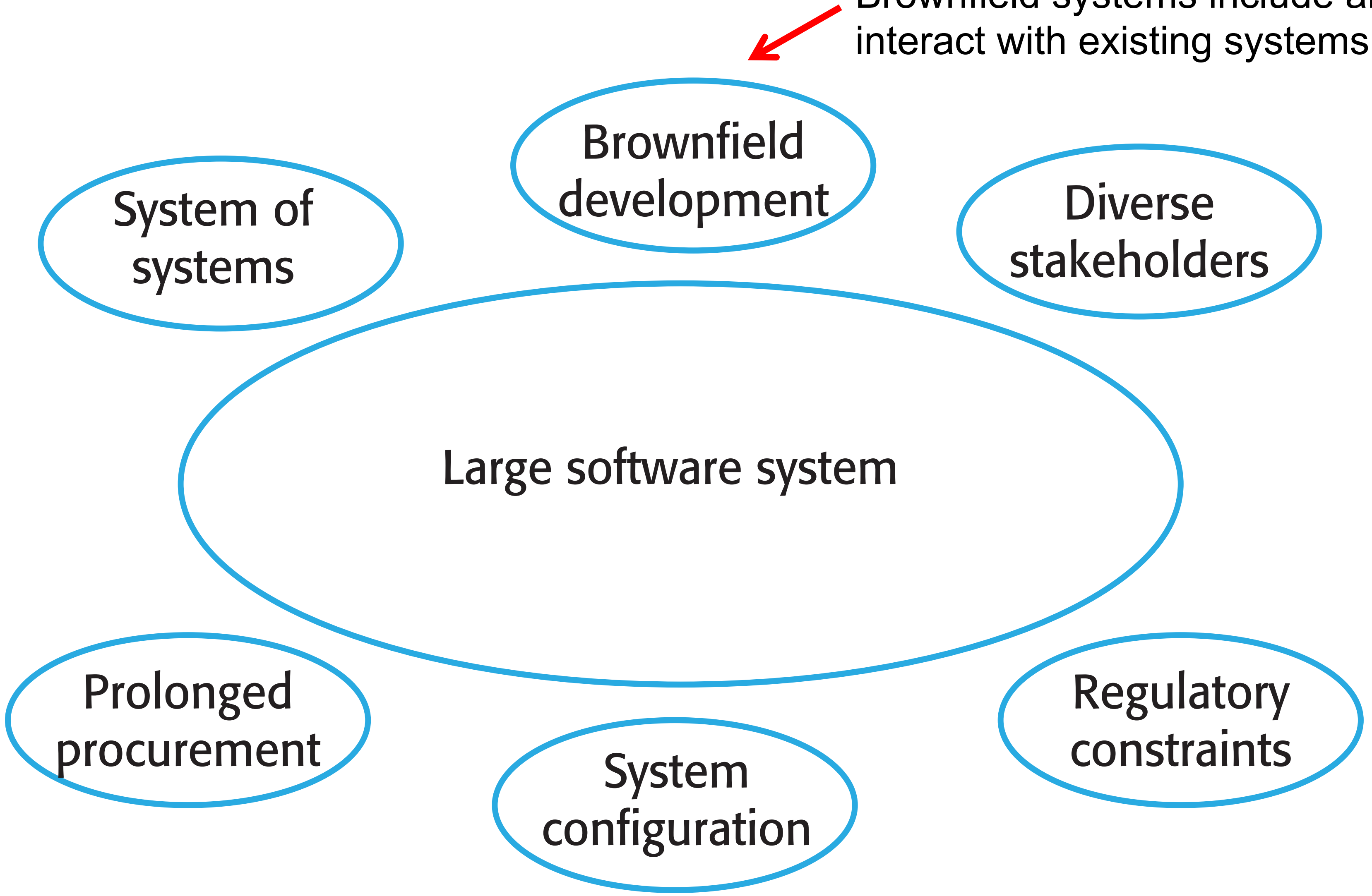
- Agile methods have proved to be successful for small and medium-sized projects that can be developed by a small co-located team.
- It is sometimes argued that the success of these methods comes because of improved communications, which is possible when everyone is working together.
- Scaling up agile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

Scaling Out vs. 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 important to maintain agile fundamentals.
  - Flexible planning
  - Frequent (small) system releases
  - Continuous integration
  - Test-driven development
  - Good team communications.

Factors in Large Systems

Brownfield systems include and interact with existing systems.



IBM's Agility at Scale Model

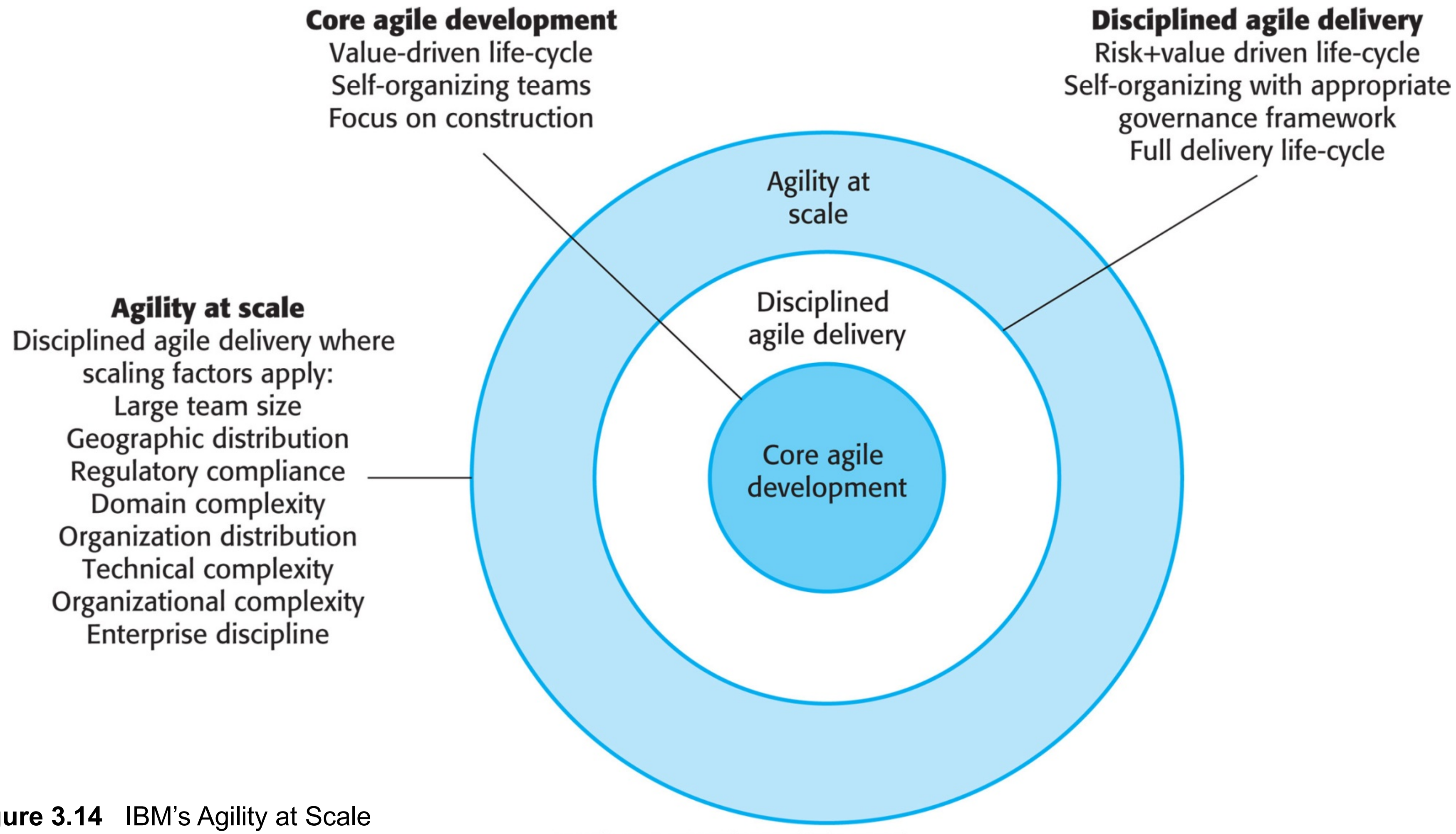

Core agile development
Value-driven life-cycle
Self-organizing teams
Focus on construction
Disciplined agile delivery
Risk+value driven life-cycle
Self-organizing with appropriate governance framework
Full delivery life-cycle
Agility at scale
Disciplined agile delivery
Core agile development
Agility at scale
Disciplined agile delivery where scaling factors apply:
Large team size
Geographic distribution
Regulatory compliance
Domain complexity
Organization distribution
Technical complexity
Organizational complexity
Enterprise discipline

Figure 3.14 IBM's Agility at Scale model. *Source*: © IBM 2010

Scaling Up to Large Systems

- A completely incremental approach to requirements engineering is impossible.
- There cannot be a single product owner or customer representative.
- For large systems development, it is not possible to focus only on the code of the system.
- Cross-team communication mechanisms have to be designed and used.
- Continuous integration is practically impossible. However, it is essential to maintain frequent system builds and regular releases of the system.

Multiteam Scrum

- **Role replication**
  - Each team has a product owner for their work component and scrum master.
- **Product architects**
  - Each team chooses a product architect, and these architects collaborate to design and evolve the overall system architecture.
- **Release alignment**
  - The dates of product releases from each team are aligned so that a demonstrable and complete system is produced.
- **Scrum of scrums**
  - There is a daily scrum of scrums where representatives from each team meet to discuss progress and plan work to be done.

Agile Methods across Organizations

- ❖ Project managers who do not have experience with 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.