軟體工程
(Software Engineering)

測試、DevOps和程式碼管理：程式碼管理和DevOps自動化

(Testing, DevOps and Code Management: Code management and DevOps automation)

1101SE10
MBA, IM, NTPU (M6131) (Fall 2021)
Thu 11, 12, 13 (19:25-22:10) (209)

Min-Yuh Day
戴敏育
Associate Professor
副教授
Institute of Information Management, National Taipei University
國立臺北大學 資訊管理研究所
https://web.ntpu.edu.tw/~myday

2021/12/23

課程大綱 (Syllabus)

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
1	2021/09/23	軟體工程概論 (Introduction to Software Engineering)
2	2021/09/30	軟體產品與專案管理：軟體產品管理，原型設計 (Software Products and Project Management: Software product management and prototyping)
3	2021/10/07	敏捷軟體工程：敏捷方法、Scrum、極限程式設計 (Agile Software Engineering: Agile methods, Scrum, and Extreme Programming)
4	2021/10/14	功能、場景和故事 (Features, Scenarios, and Stories)
5	2021/10/21	軟體工程個案研究 I (Case Study on Software Engineering I)
6	2021/10/28	軟體架構：架構設計、系統分解、分散式架構 (Software Architecture: Architectural design, System decomposition, and Distribution architecture)

課程大綱 (Syllabus)

週次 (Week) 日期 (Date) 內容 (Subject/Topics)

7 2021/11/04 基於雲的軟體：虛擬化和容器、軟體即服務
(Cloud-Based Software: Virtualization and containers, Everything as a service, Software as a service)

8 2021/11/11 期中報告 (Midterm Project Report)

9 2021/11/18 雲端運算與雲軟體架構
(Cloud Computing and Cloud Software Architecture)

10 2021/11/25 微服務架構：RESTful服務、服務部署
(Microservices Architecture, RESTful services, Service deployment)

11 2021/12/02 安全和隱私 (Security and Privacy);
可靠的程式設計 (Reliable Programming)

12 2021/12/09 軟體工程個案研究 II
(Case Study on Software Engineering II)

課程大綱 (Syllabus)

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
13	2021/12/16	軟體工程產業實務 (Industry Practices of Software Engineering)
14	2021/12/23	測試：功能測試、測試自動化、測試驅動的開發、程式碼審查 (Testing: Functional testing, Test automation, Test-driven development, and Code reviews); DevOps和程式碼管理：程式碼管理和DevOps自動化 (DevOps and Code Management: Code management and DevOps automation)
15	2021/12/30	期末報告 I (Final Project Report I)
16	2022/01/06	期末報告 II (Final Project Report II)
17	2022/01/13	學生自主學習 (Self-learning)
18	2022/01/20	學生自主學習 (Self-learning)

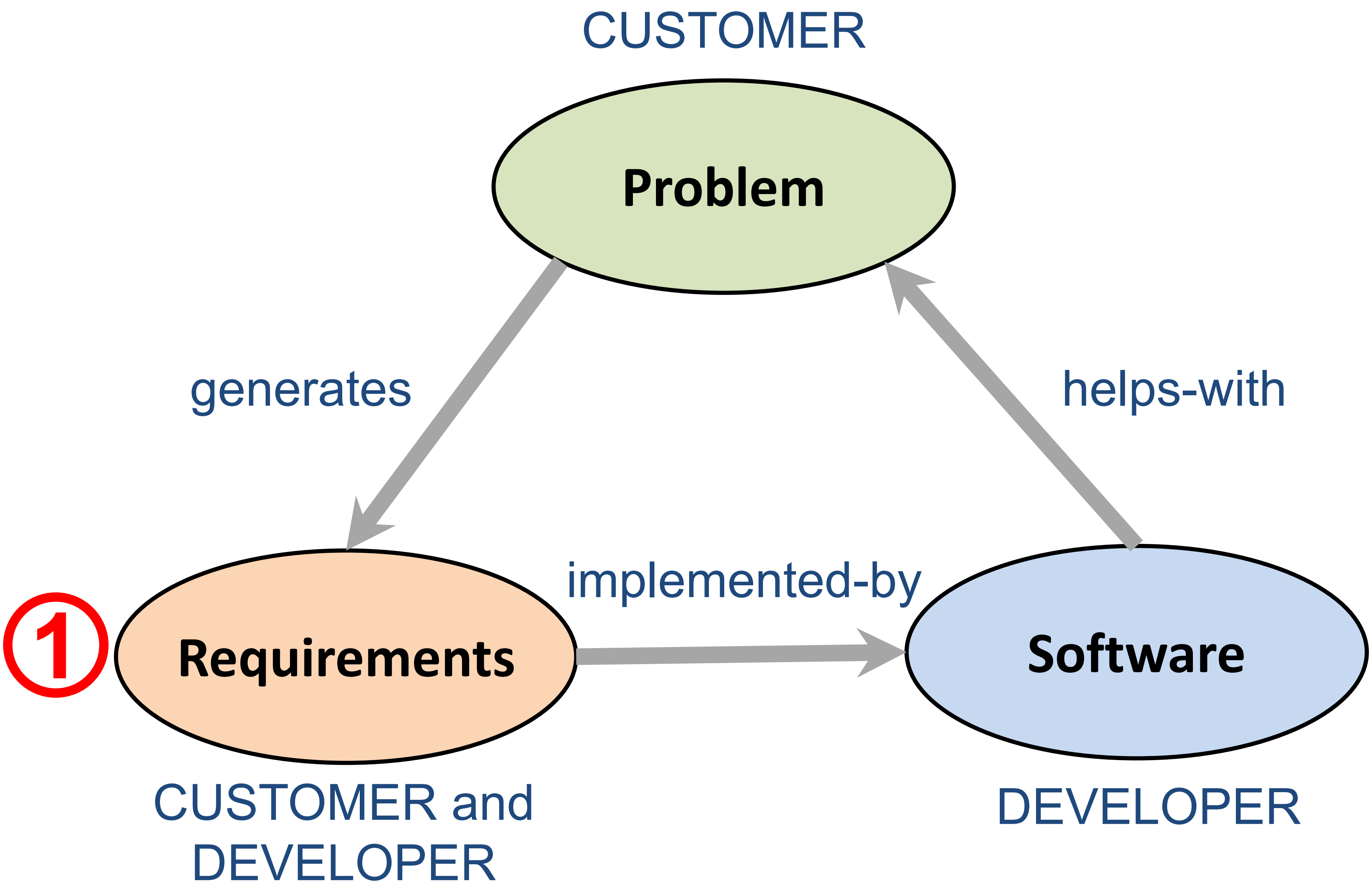
Software Engineering
and
Project Management

Analyze → **Design** → **Build** → **Test** → **Deliver**

- **Analyze**: Requirements definition
- **Design**: System and Software design
- **Build**: Implementation and unit testing
- **Test**: Integration and system testing
- **Deliver**: Operation and maintenance

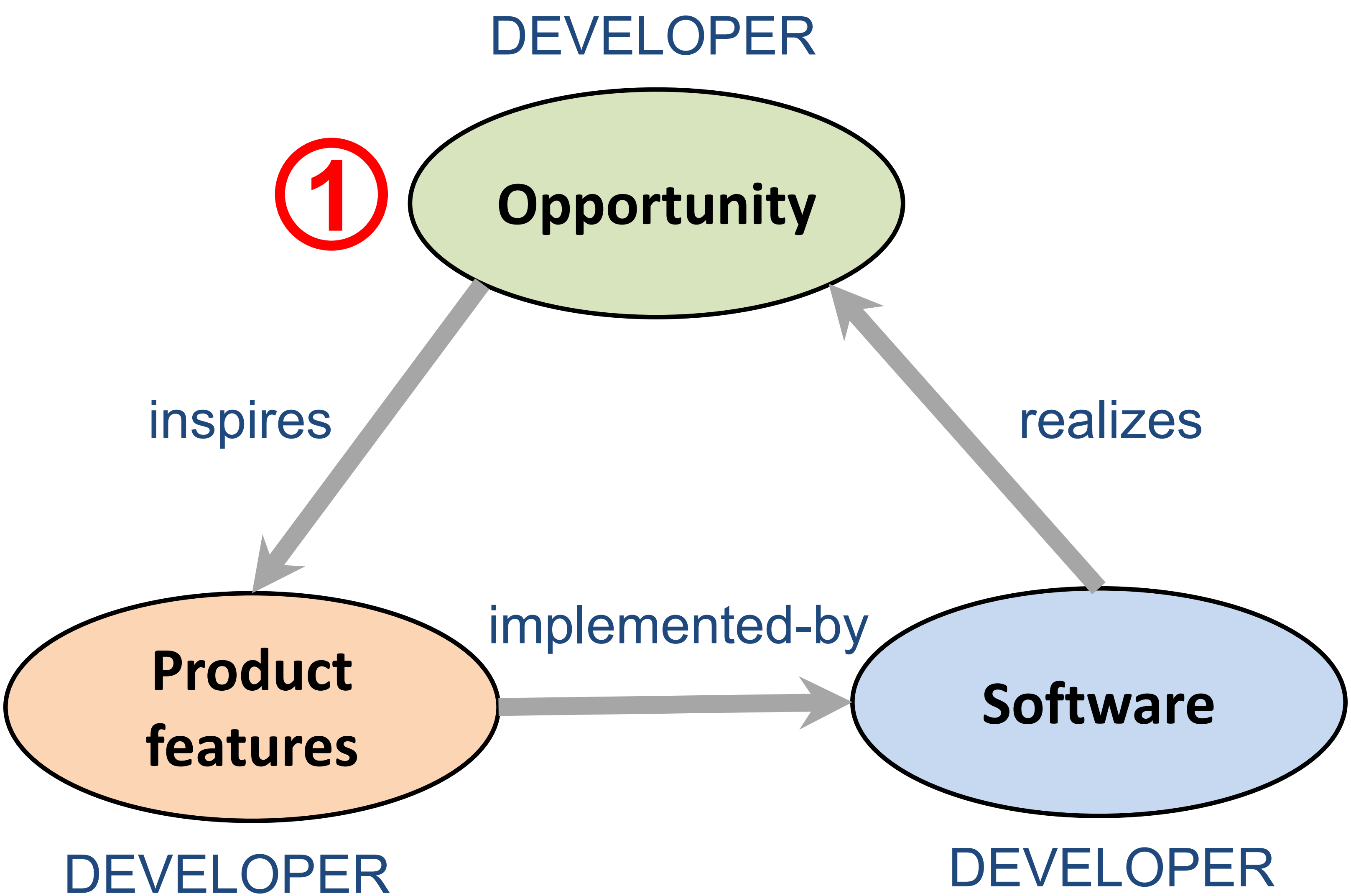
Project Management

Project-based software engineering



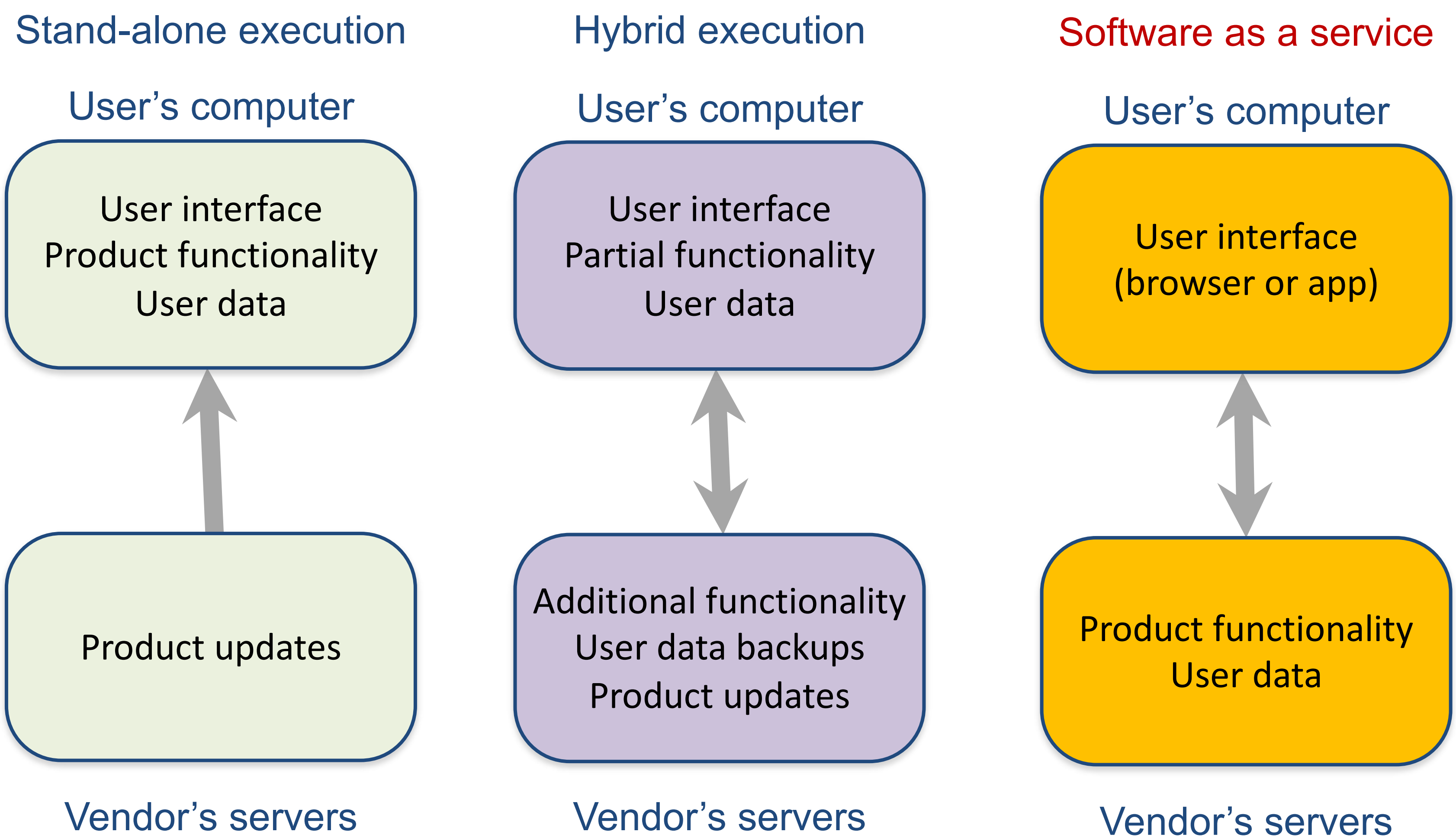
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Product software engineering



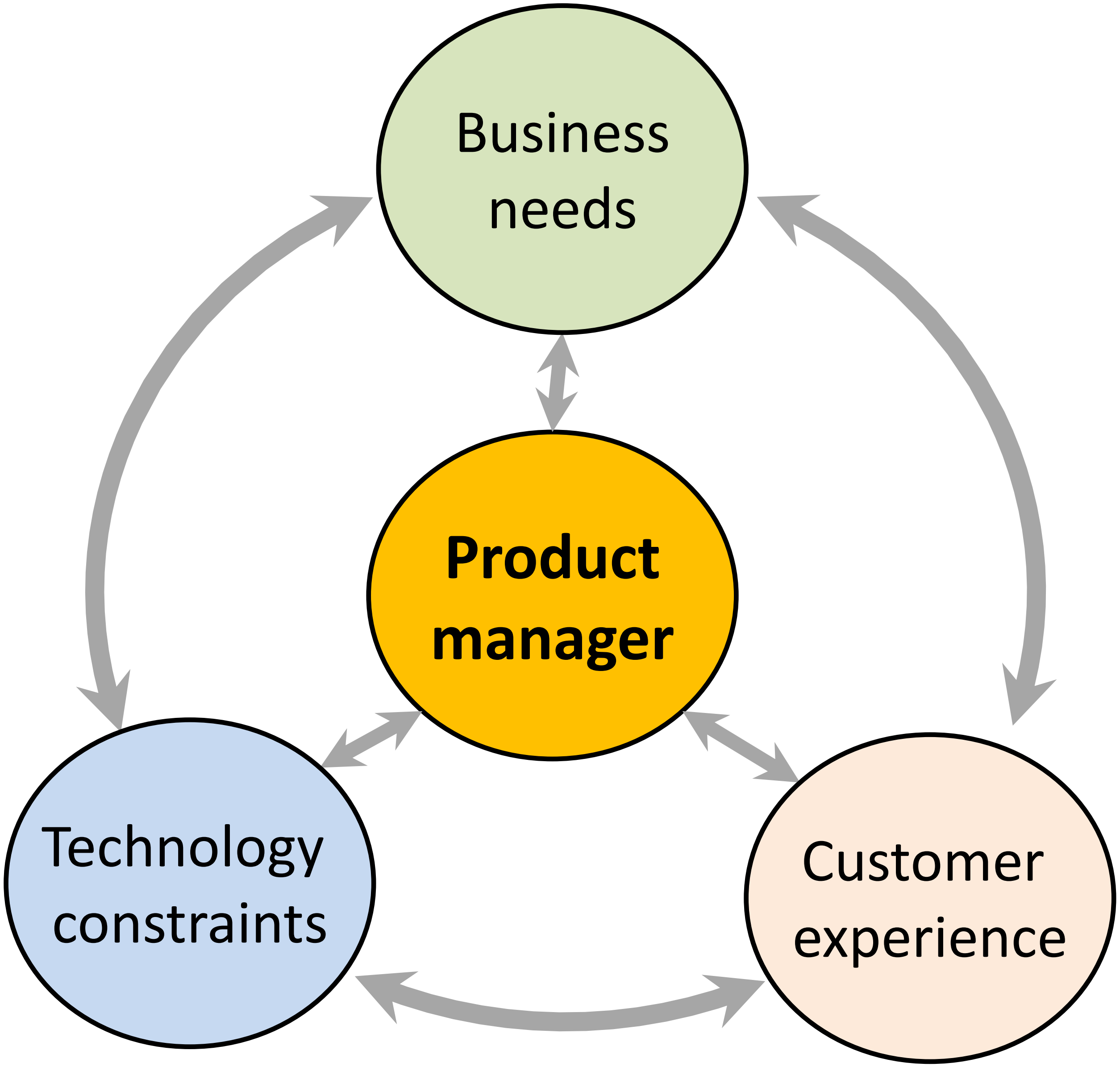
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software execution models



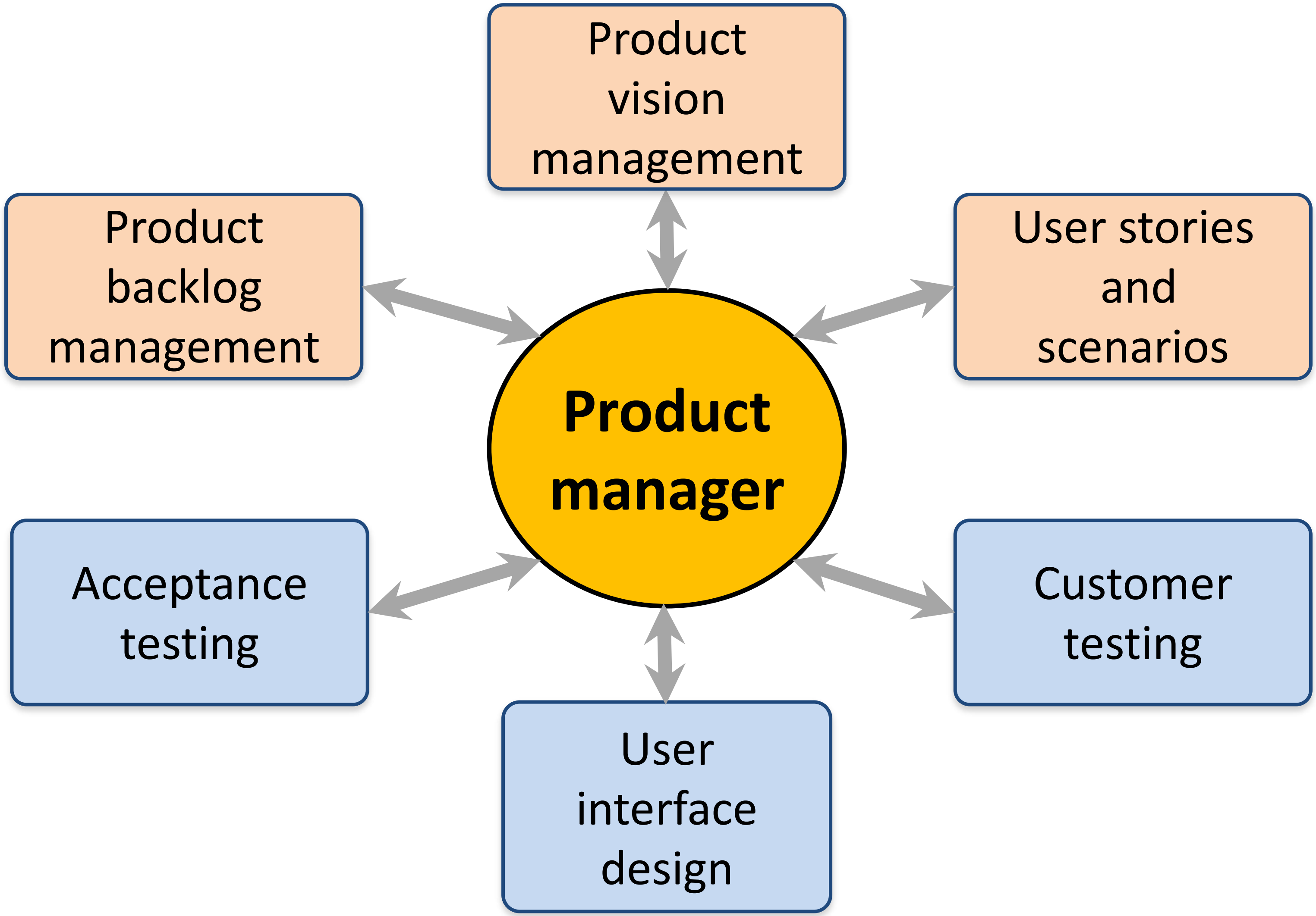
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Product management concerns



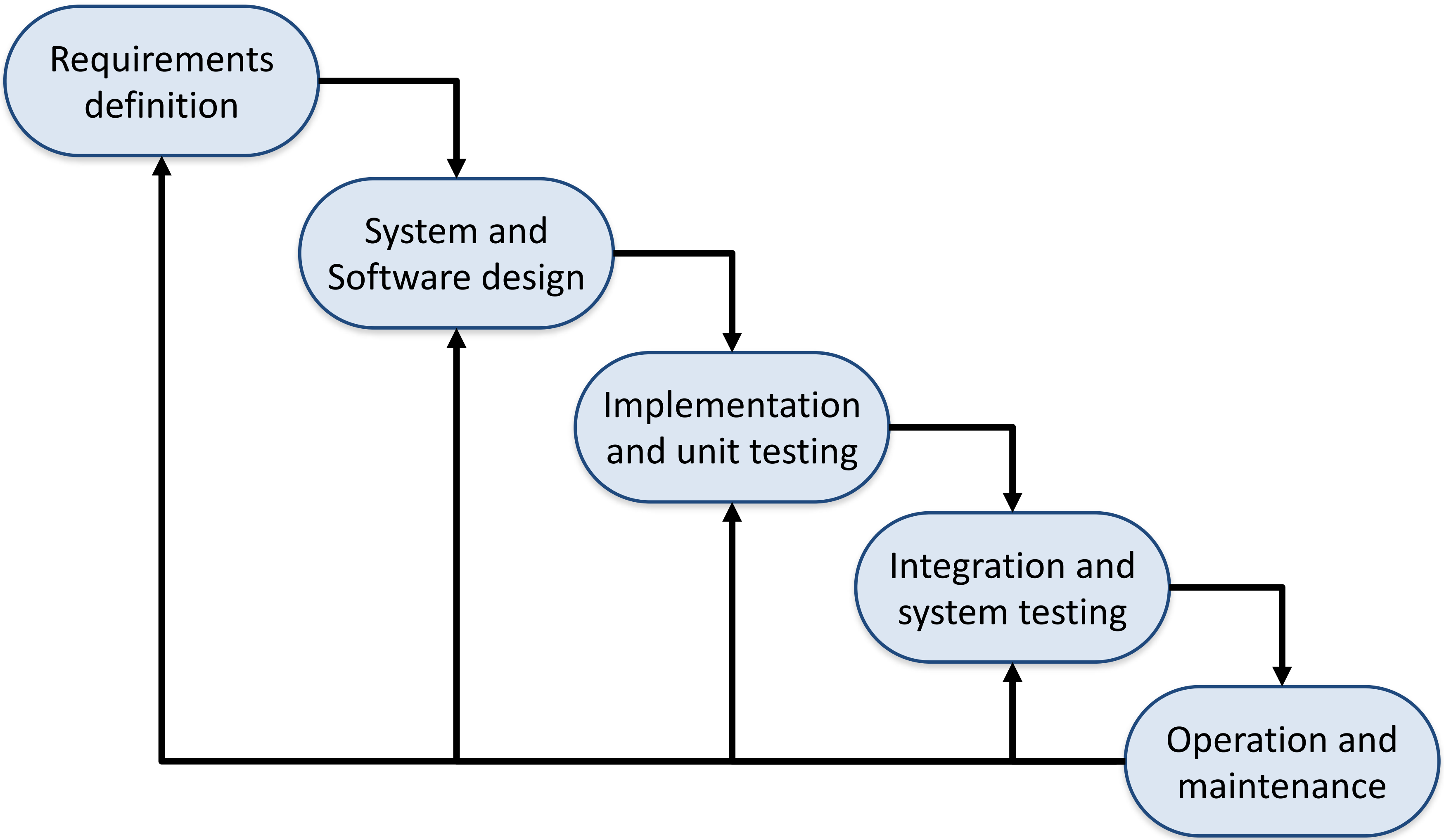
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Technical interactions of product managers



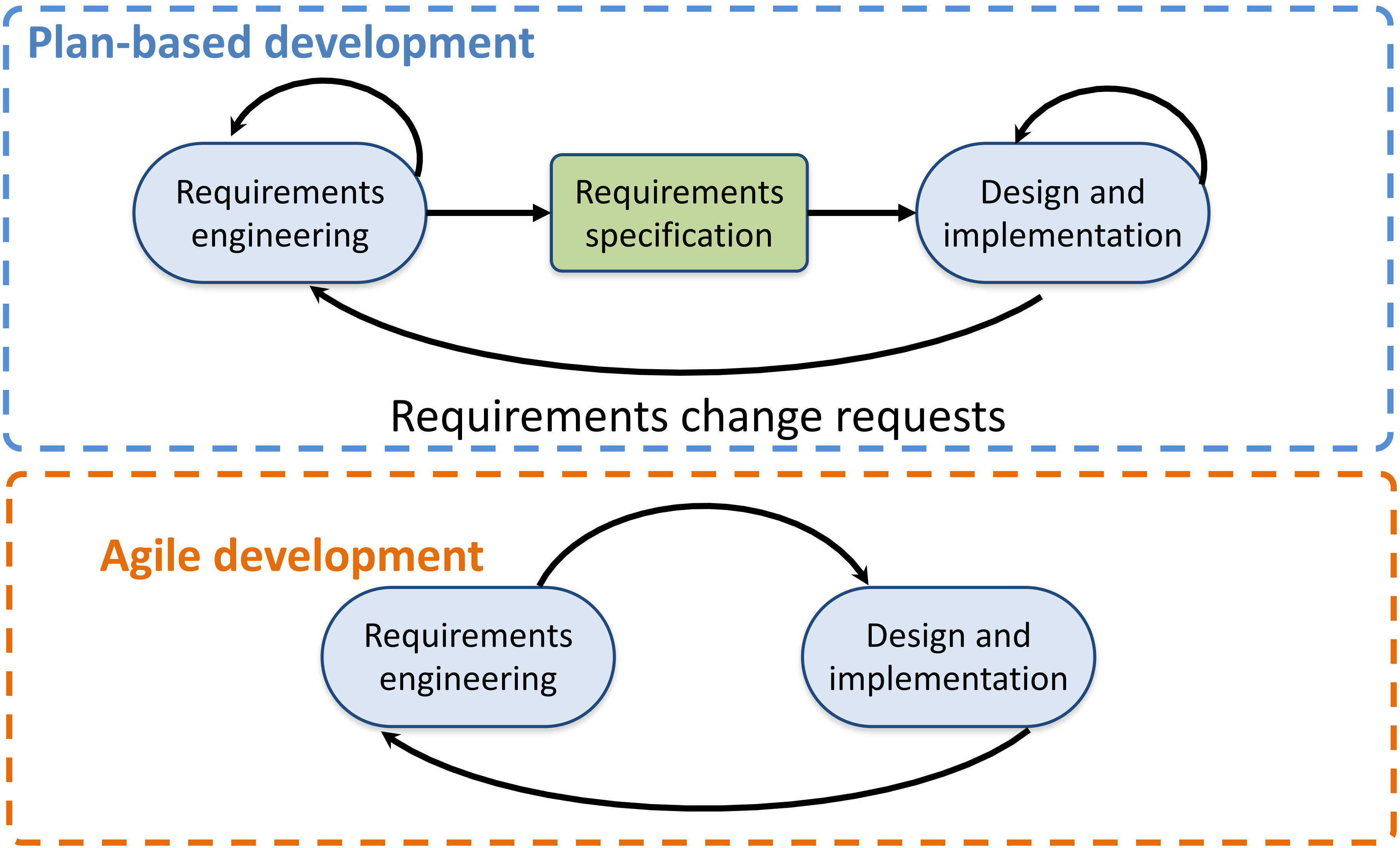
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software Development Life Cycle (SDLC)
The waterfall model



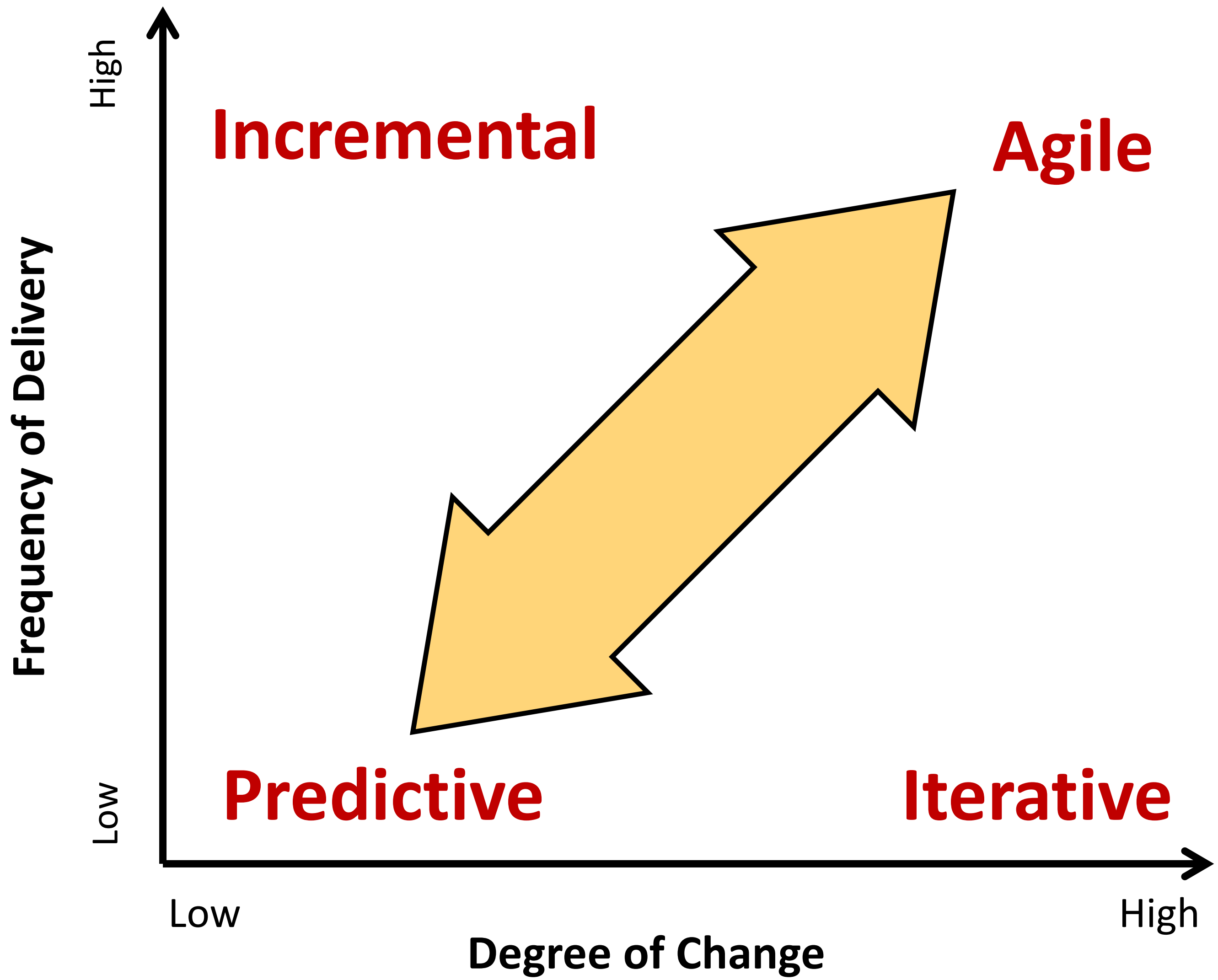
Source: Ian Sommerville (2015), Software Engineering, 10th Edition, Pearson.

Plan-based and Agile development



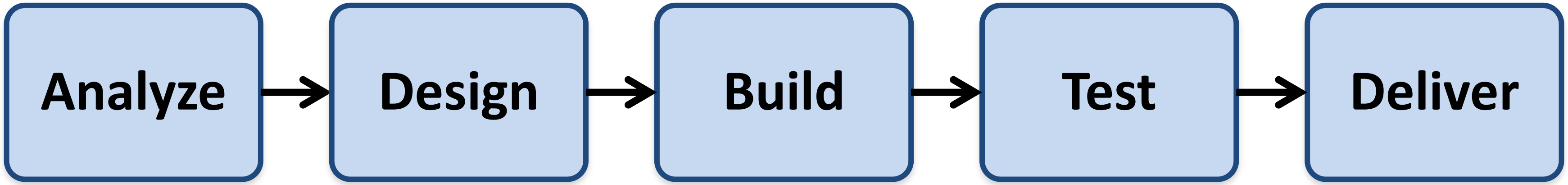
Source: Ian Sommerville (2015), Software Engineering, 10th Edition, Pearson.

The Continuum of Life Cycles



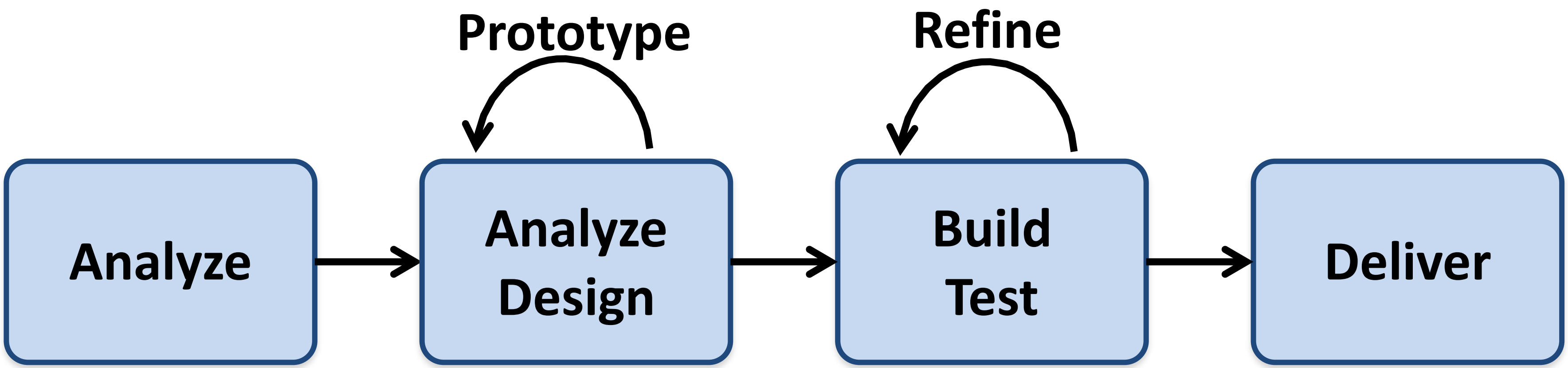
Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

Predictive Life Cycle

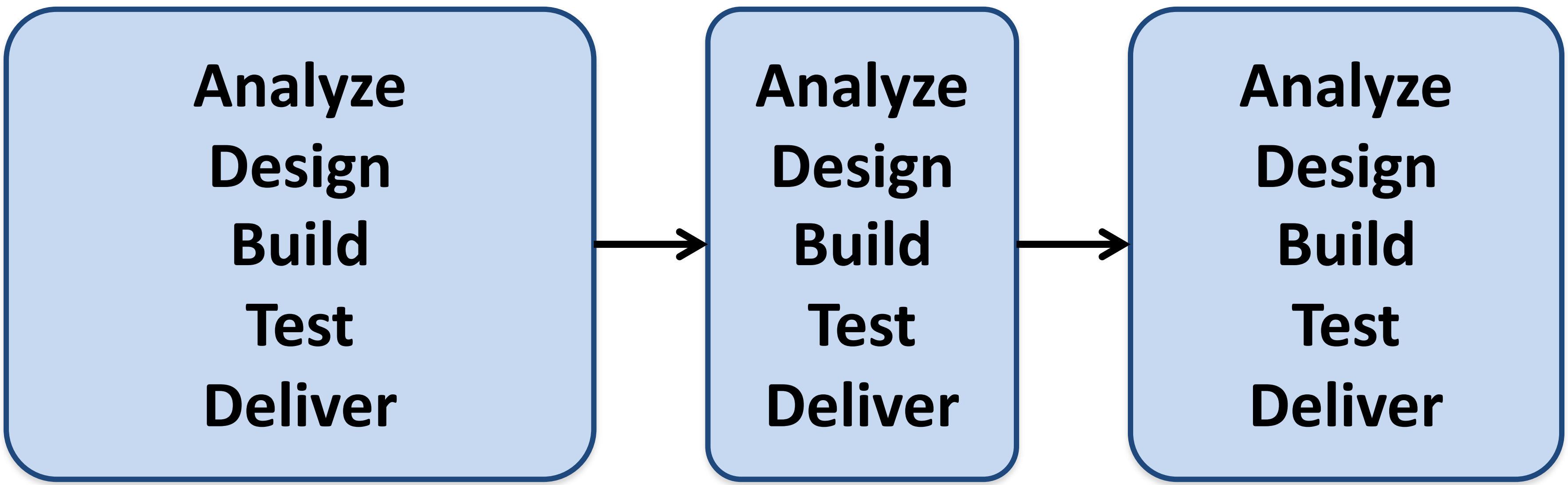


Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

Iterative Life Cycle



A Life Cycle of Varying-Sized Increments



Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

Iteration-Based and Flow-Based Agile Life Cycles

Iteration-Based Agile

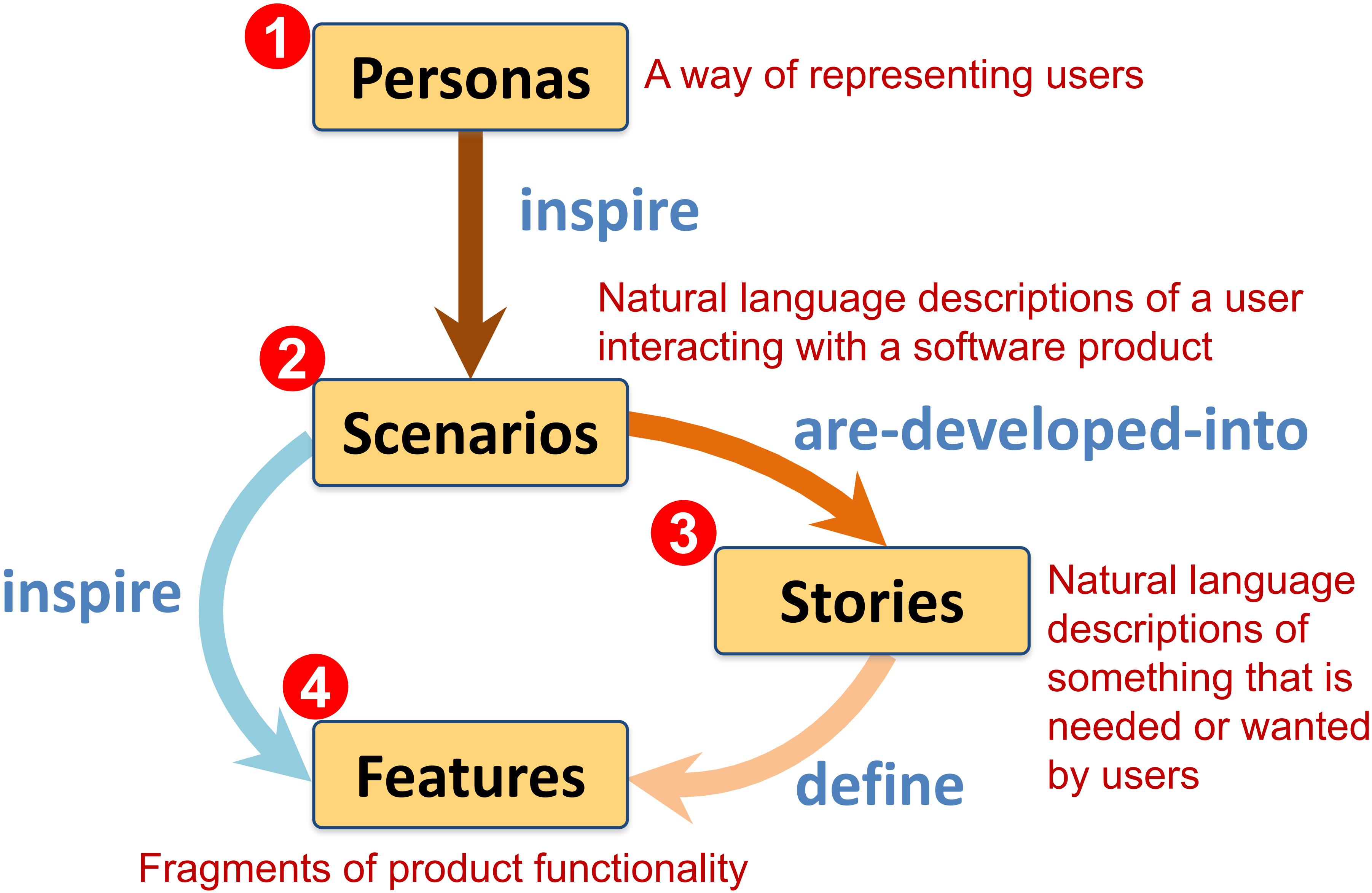
Requirements Analysis Design Build Test	Requirements Analysis Design Build Test	Requirements Analysis Design Build Test	Requirements Analysis Design Build Test	Repeat as needed …	Requirements Analysis Design Build Test	Requirements Analysis Design Build Test

Flow-Based Agile

Requirements **Analysis** **Design** **Build** **Test** the number of features in the WIP limit	**Requirements** **Analysis** **Design** **Build** **Test** the number of features in the WIP limit	**Requirements** **Analysis** **Design** **Build** **Test** the number of features in the WIP limit	**Repeat** **as needed** **…**	**Requirements** **Analysis** **Design** **Build** **Test** the number of features in the WIP limit	**Requirements** **Analysis** **Design** **Build** **Test** the number of features in the WIP limit

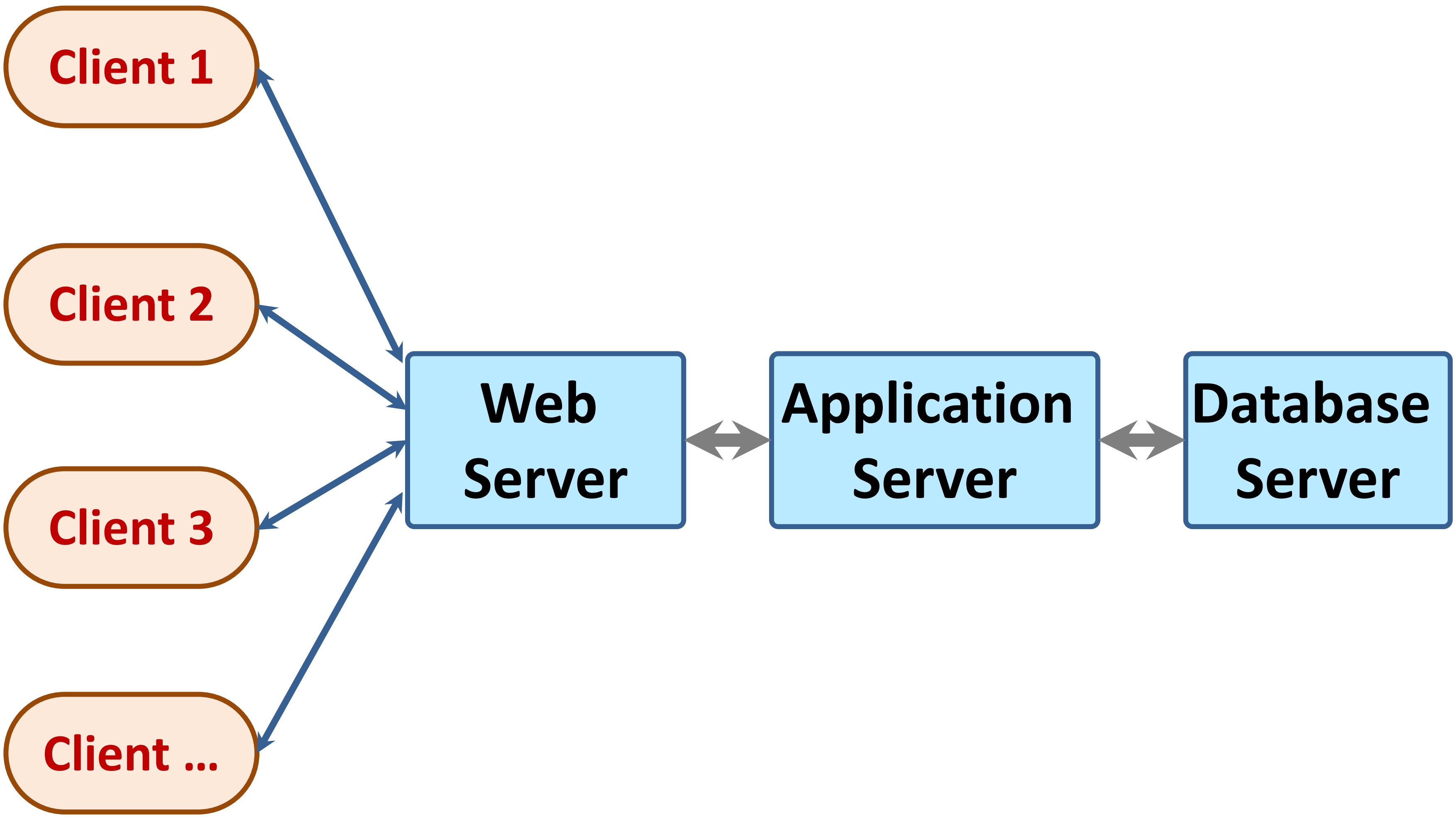
Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

From personas to features



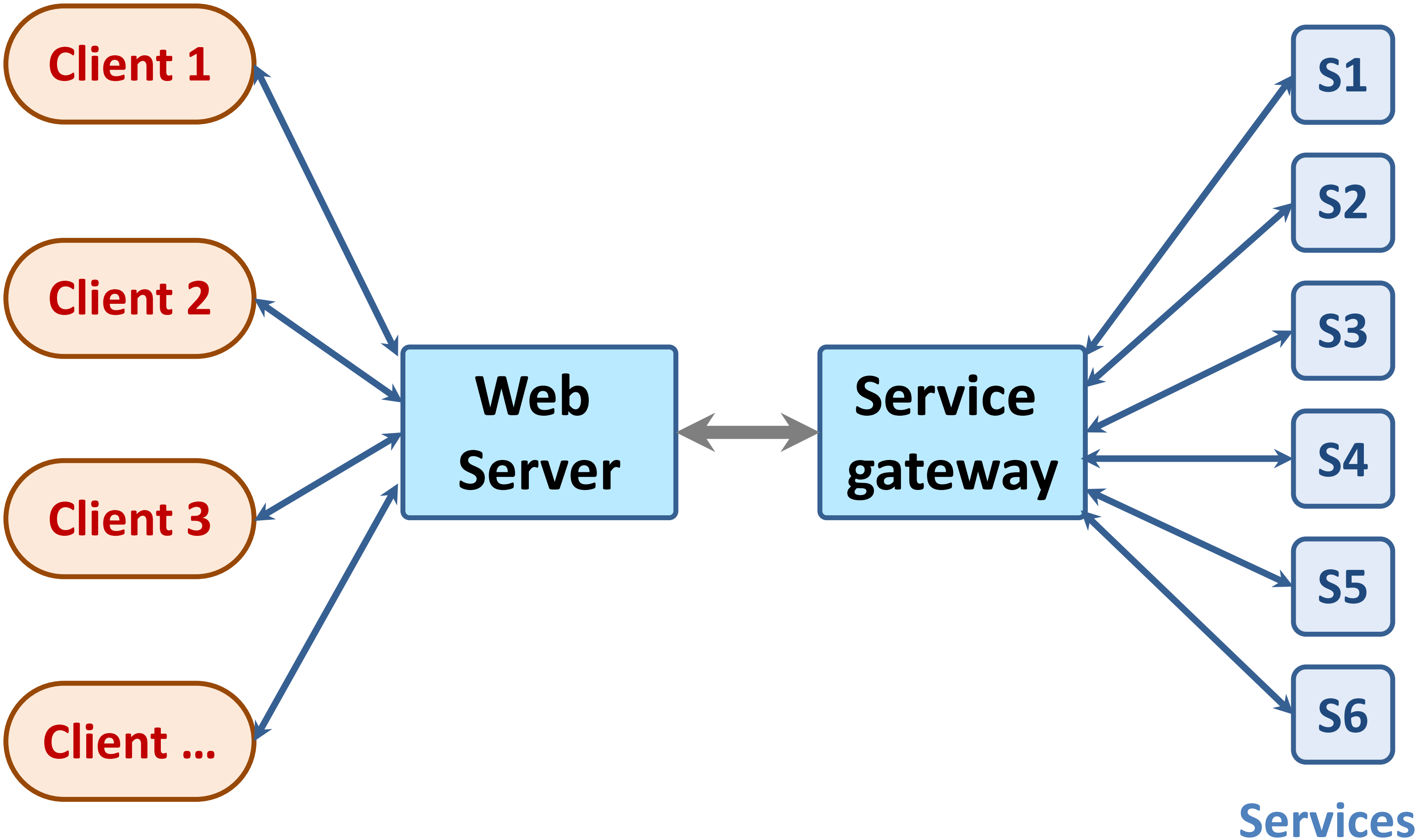
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Multi-tier client-server architecture

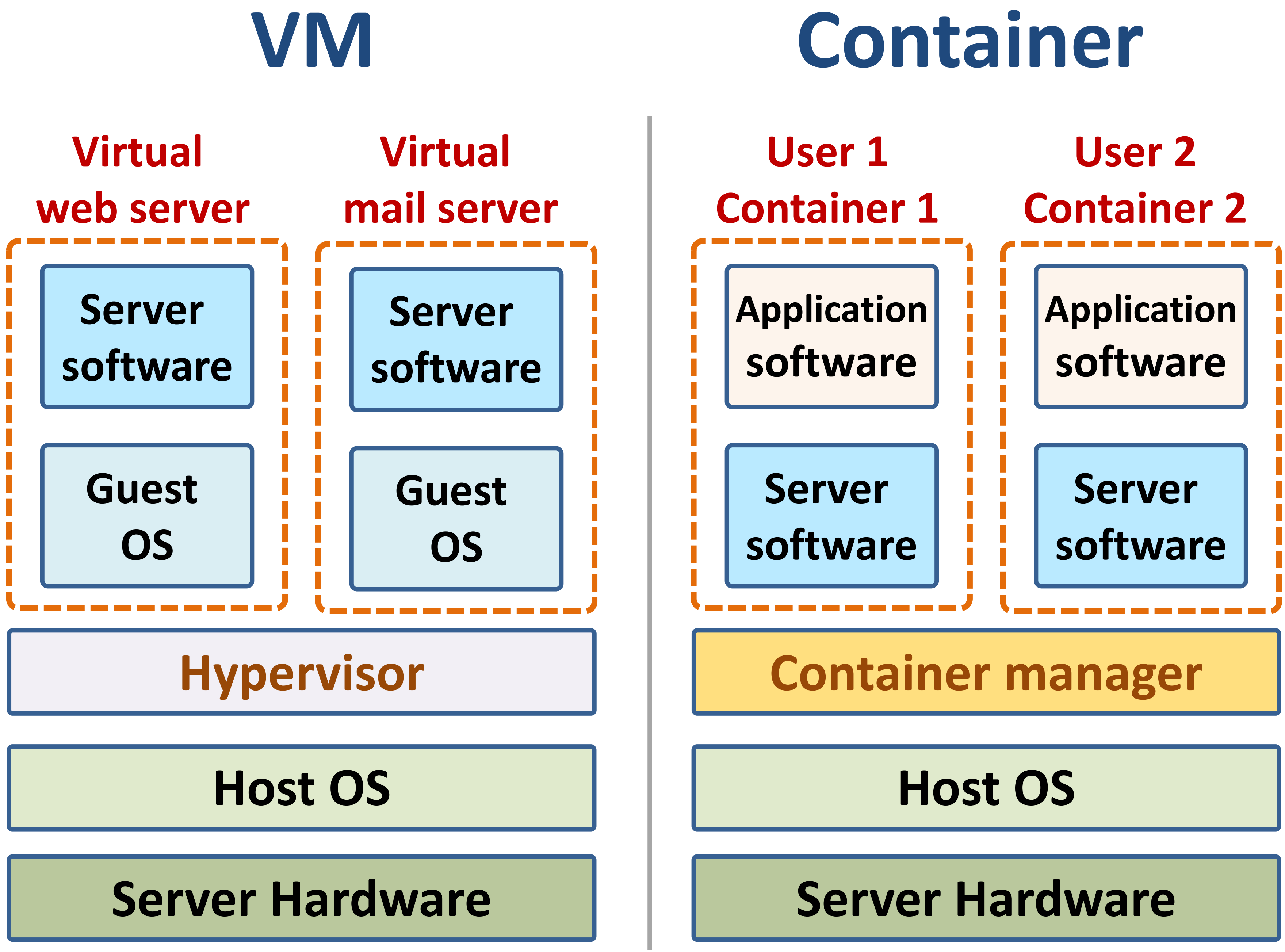


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Service-oriented Architecture

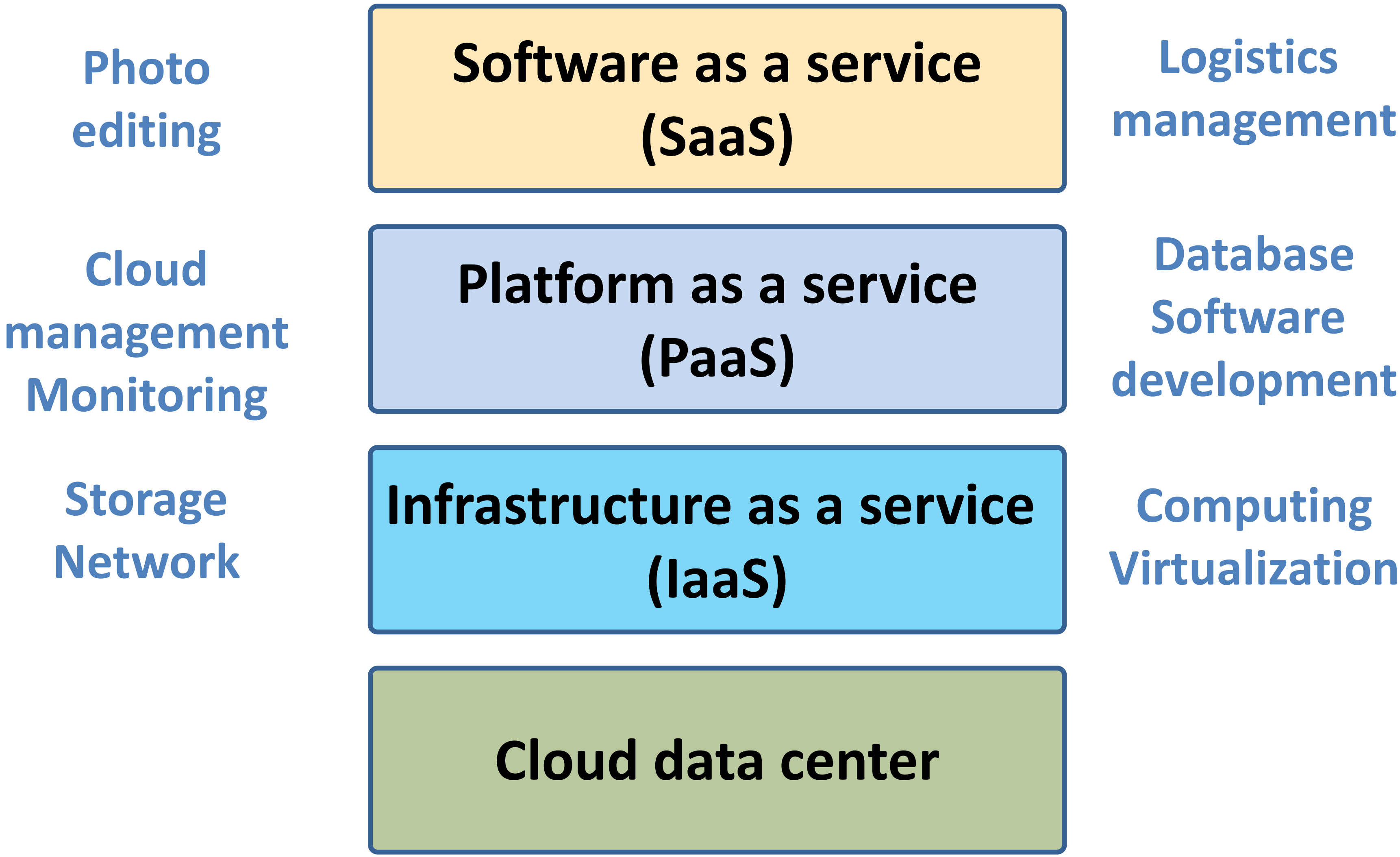


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.



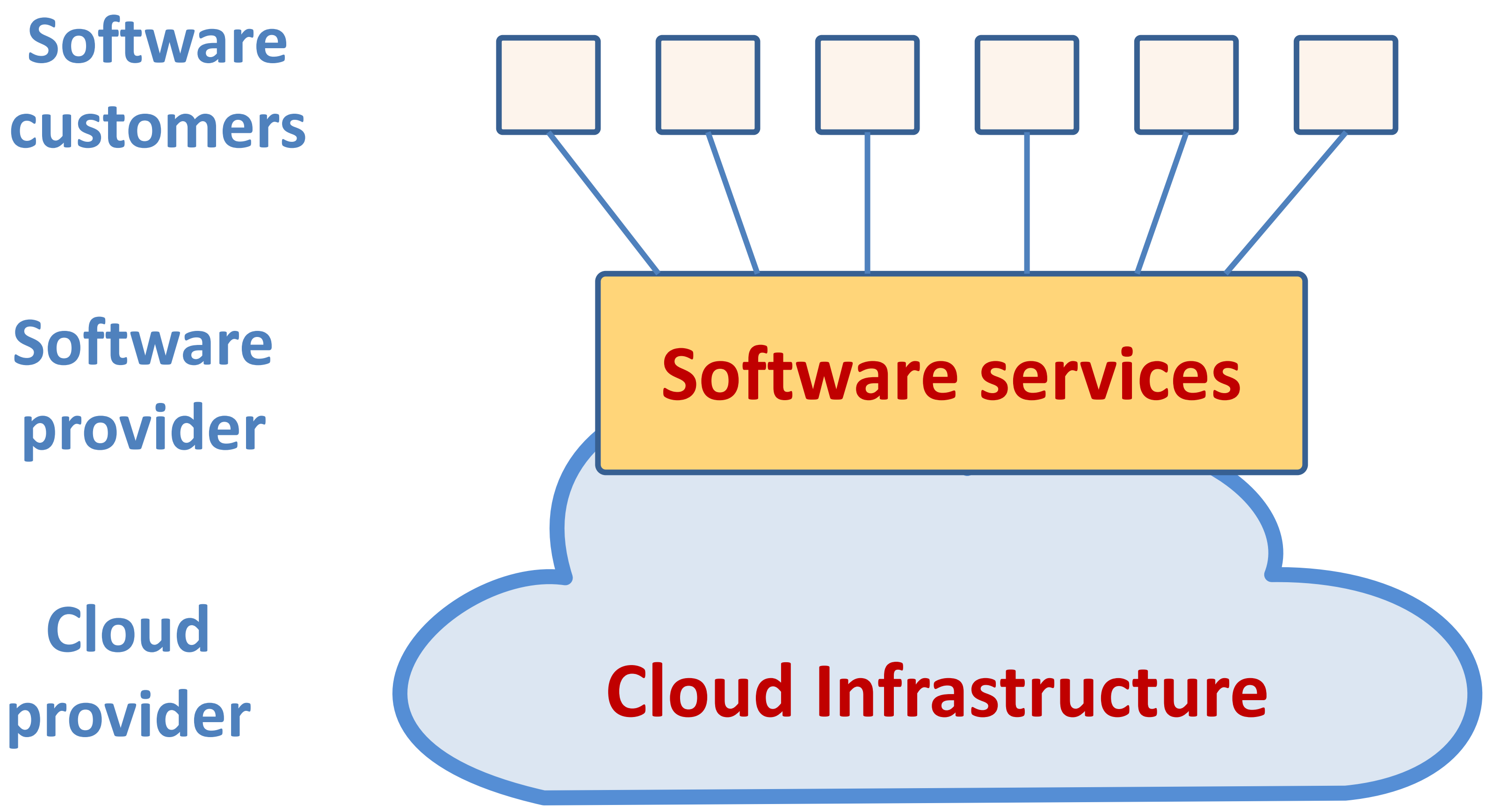
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Everything as a service



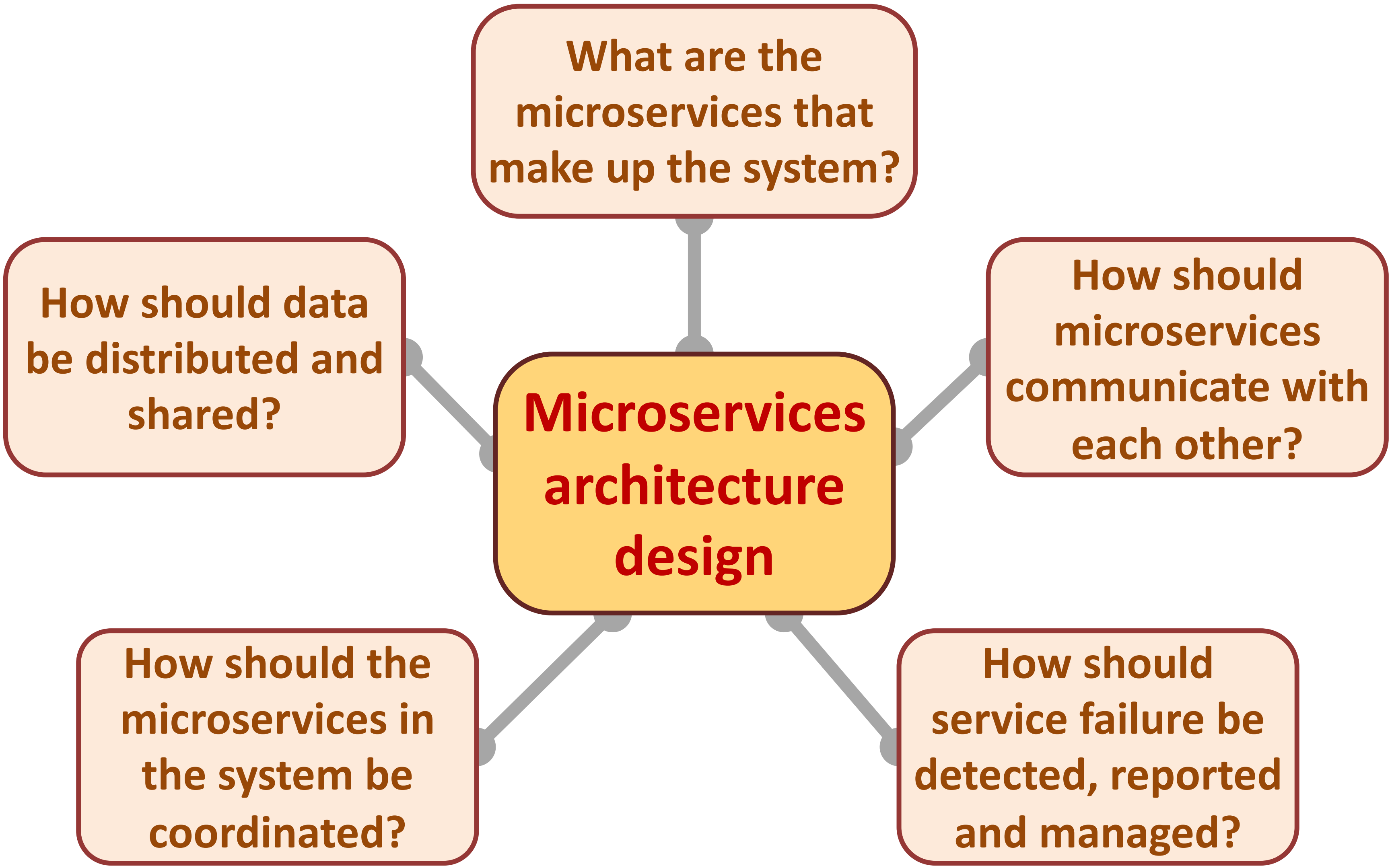
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software as a service



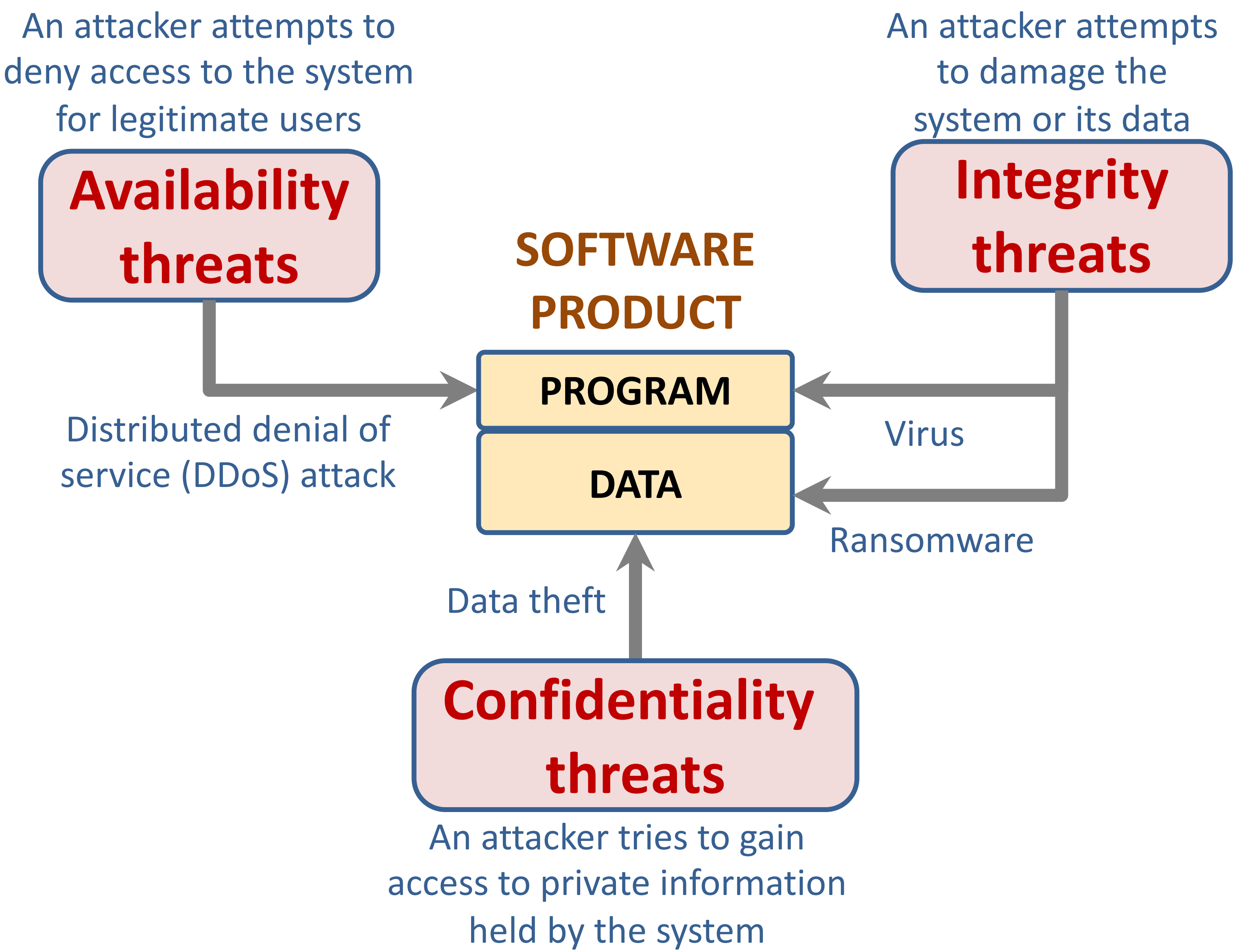
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Microservices architecture – key design questions



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Types of security threat

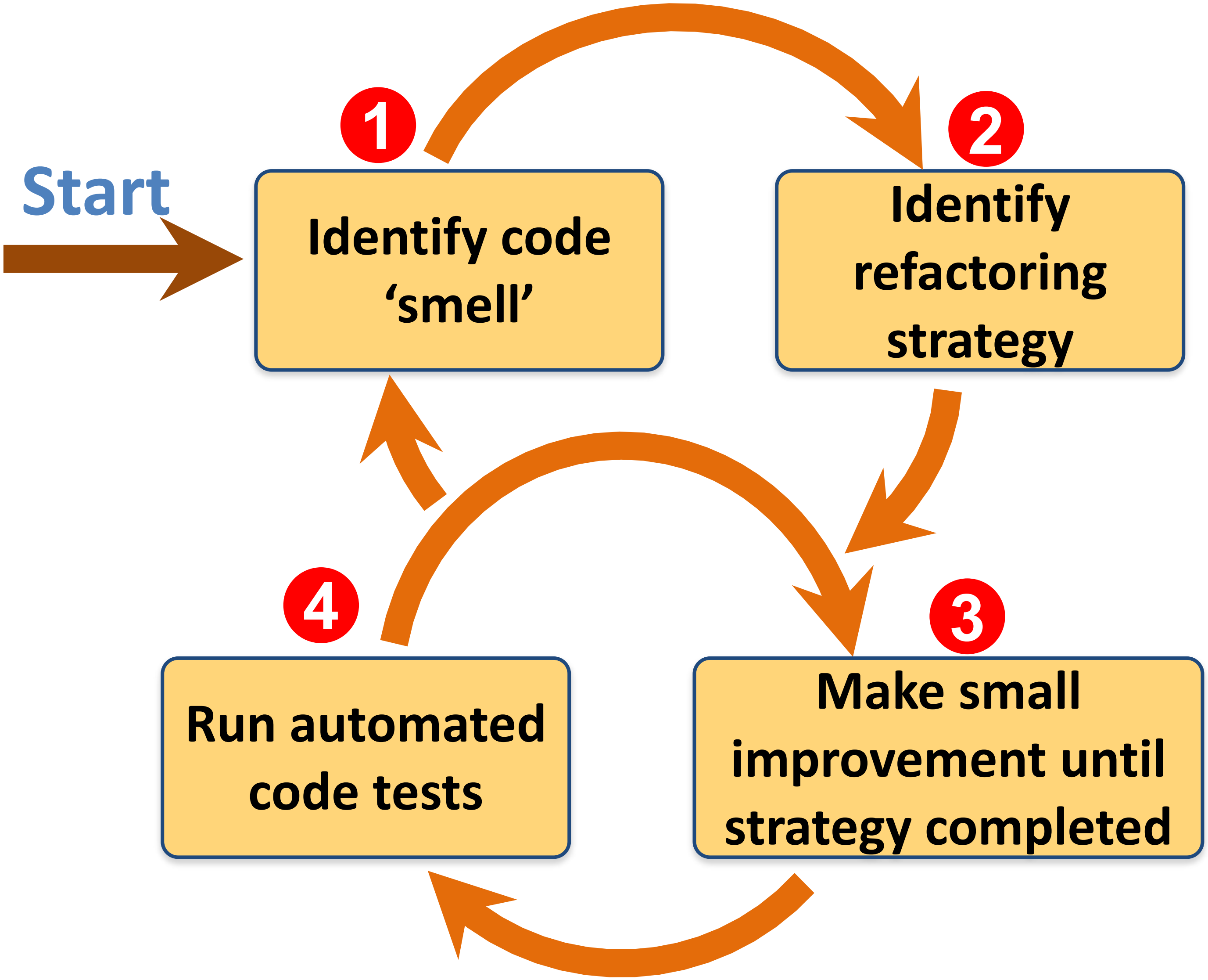


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software product quality attributes

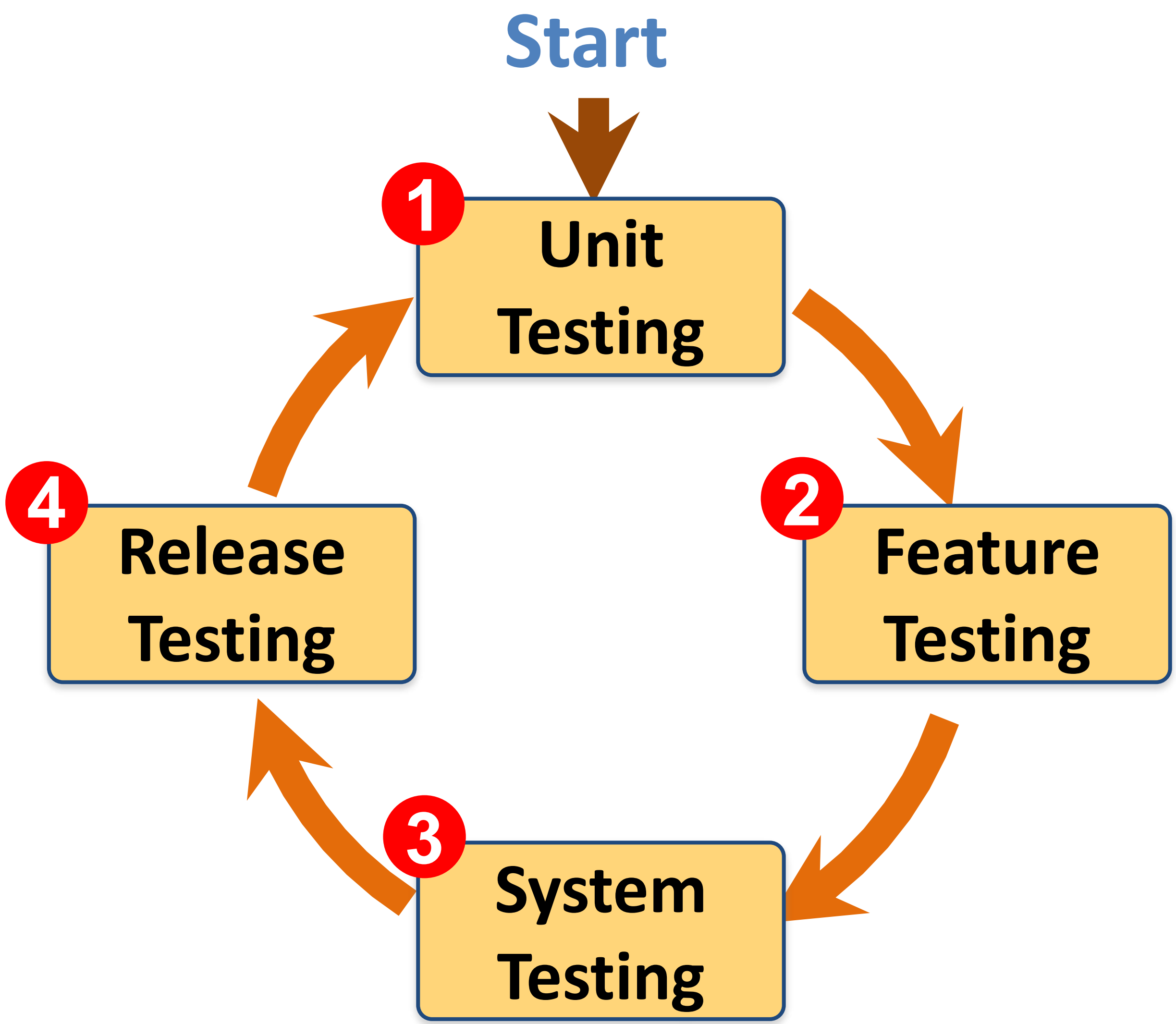
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

A refactoring process



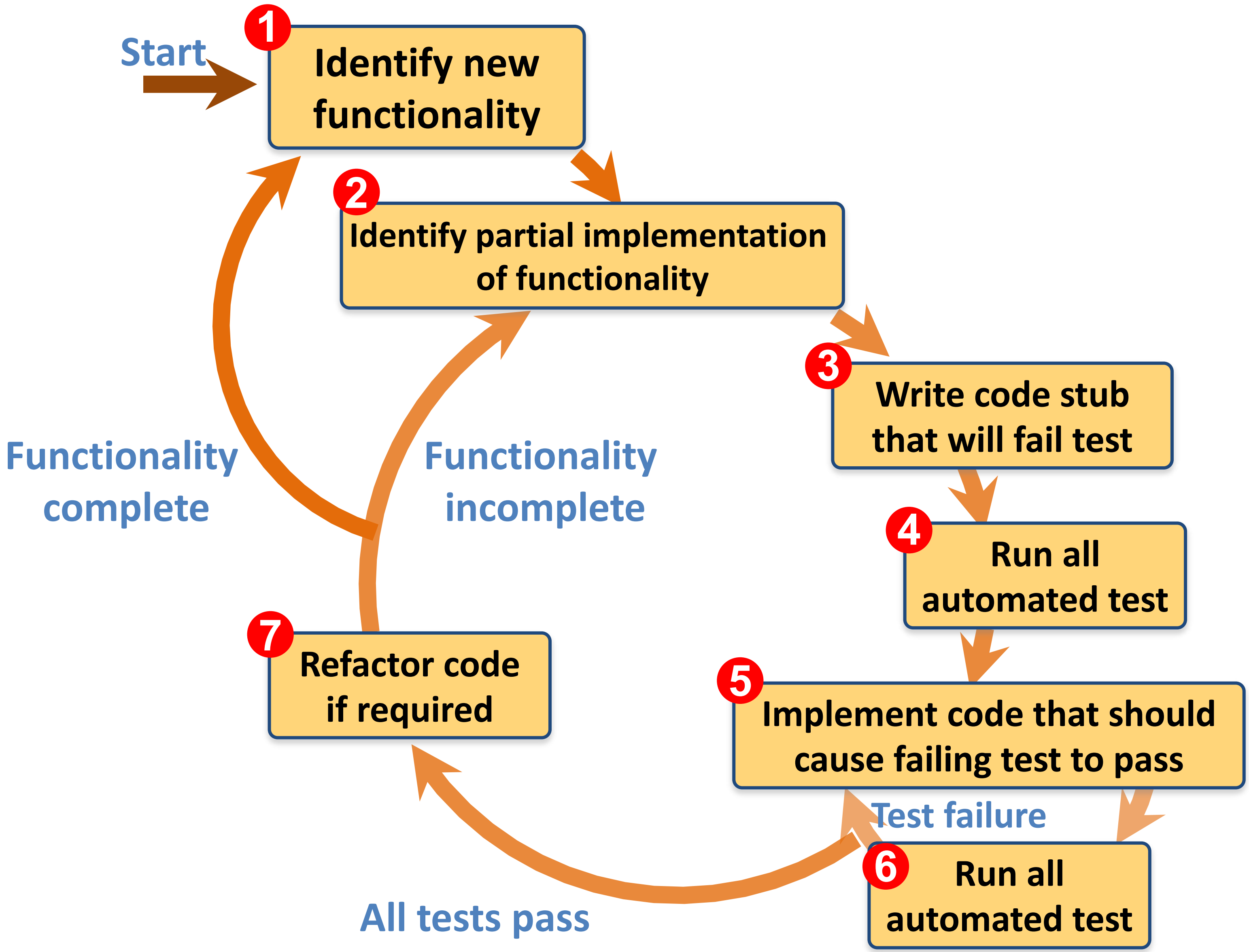
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Functional testing



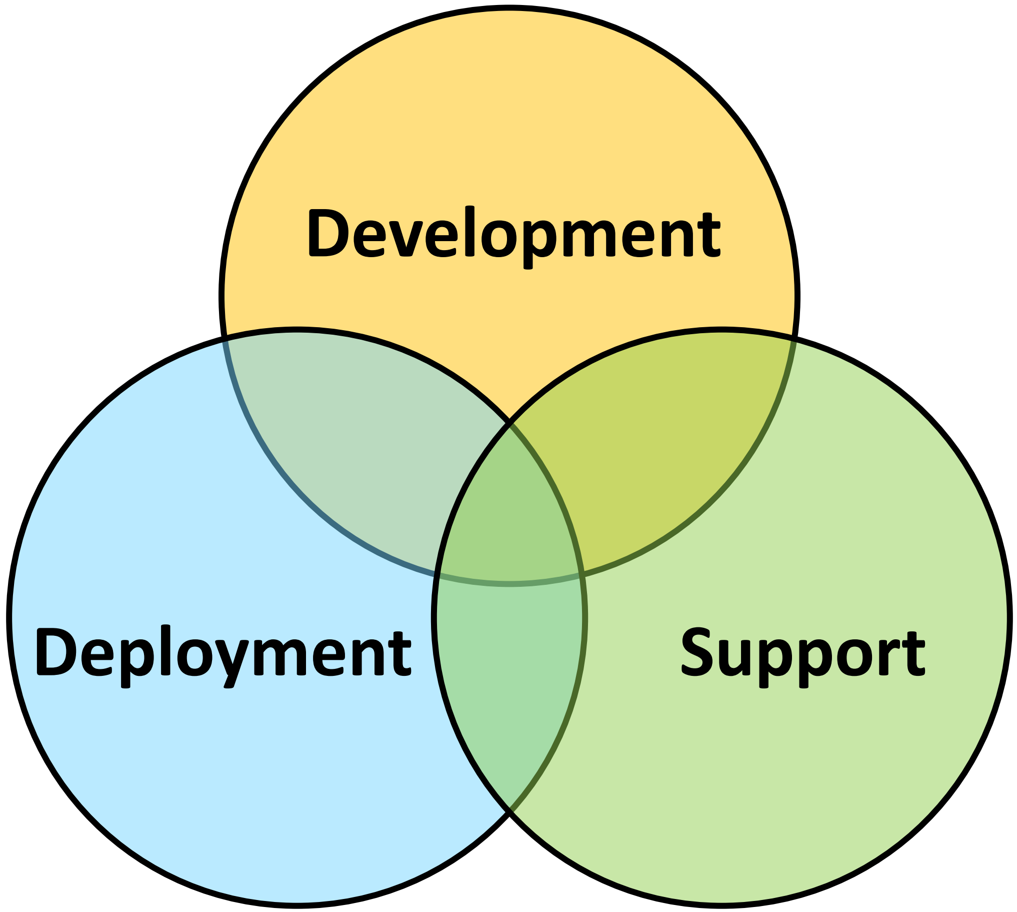
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Test-driven development (TDD)



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

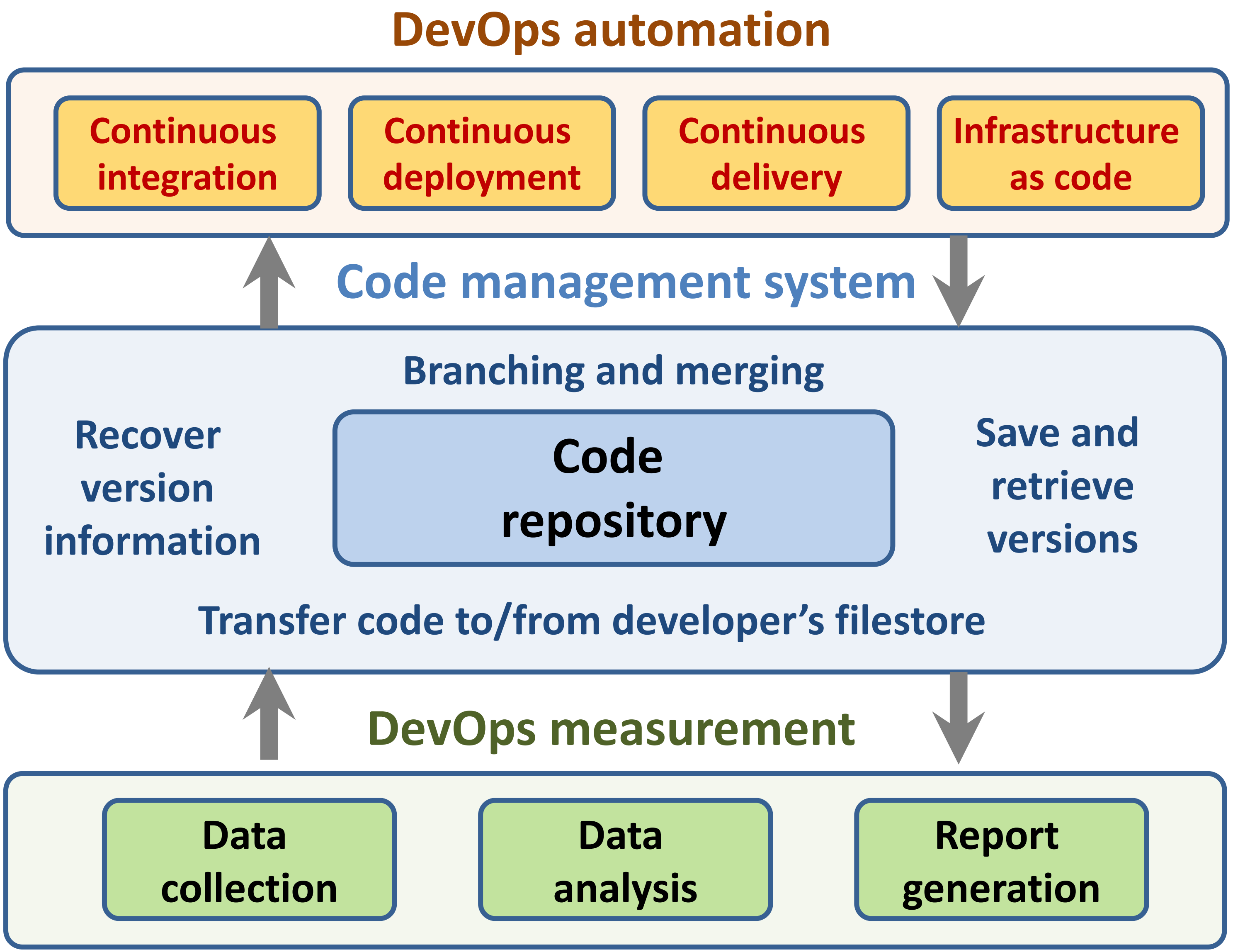
DevOps



Multi-skilled DevOps team

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Code management and DevOps



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Testing:
Functional testing, Test automation, Test-driven development, and Code reviews

Outline

- Software testing
- Functional testing
- Test automation
- Test-driven development
- Code reviews

Software testing

- Software testing is a process in which you **execute** your **program** using data that **simulates** user **inputs**.
- You observe its behaviour to see whether or not your program is doing what it is supposed to do.
  - Tests pass if the behaviour is what you expect. Tests fail if the behaviour differs from that expected.
  - If your program does what you expect, this shows that for the inputs used, the program behaves correctly.
- If these inputs are representative of a larger set of inputs, you can infer that the program will behave correctly for all members of this larger input set.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Program bugs

- If the behaviour of the program does not match the behaviour that you expect, then this means that there are bugs in your program that need to be fixed.
- There are two causes of program bugs:
  - Programming errors
    - You have accidentally included faults in your program code. For example: ‘off-by-1’ error
  - Understanding errors
    - You have misunderstood or have been unaware of some of the details of what the program is supposed to do.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Types of testing

Functional testing	Test the functionality of the overall system.
User testing	Test that the software product is useful to and usable by end-users.
Performance and load testing	Test that the software works quickly and can handle the expected load placed on the system by its users.
Security testing	Test that the software maintains its integrity and can protect user information from theft and damage.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

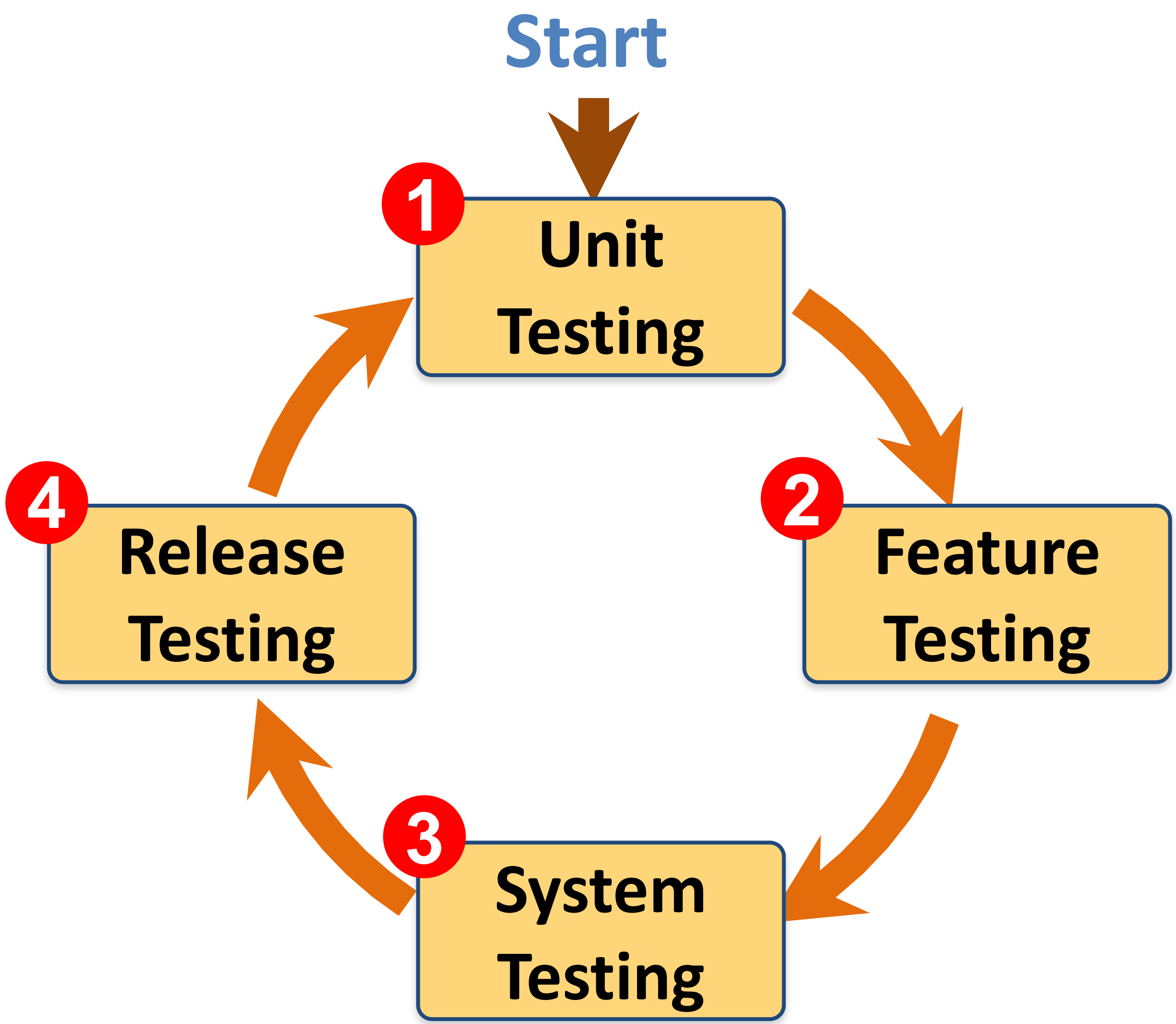
Functional testing

- Functional testing involves developing a large set of program tests so that, ideally, all of a program's code is executed at least once.
- The number of tests needed obviously depends on the size and the functionality of the application.
- For a business-focused web application, you may have to develop thousands of tests to convince yourself that your product is ready for release to customers.

Functional testing

- Functional testing is a staged activity in which you initially test individual units of code.
  You integrate code units with other units to create larger units then do more testing.
- The process continues until you have created a complete system ready for release.

Functional testing



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

A name checking function

```
def namecheck(s):

    # Checks that a name only includes alphabetic characters, - or
    # a single quote. Names must be between 2 and 40 characters long
    # quoted strings and -- are disallowed

    namex = r"^[a-zA-Z][a-zA-Z-']{1,39}$"
    if re.match(namex, s):
        if re.search("'.*'", s) or re.search("--", s):
            return False
        else:
            return True
    else:
        return False
```

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Equivalence partitions for the name checking function

- Correct names 1
  The inputs only includes alphabetic characters and are between 2 and 40 characters long.
- Correct names 2
  The inputs only includes alphabetic characters, hyphens or apostrophes and are between 2 and 40 characters long.
- Incorrect names 1
  The inputs are between 2 and 40 characters long but include disallowed characters.
- Incorrect names 2
  The inputs include allowed characters but are either a single character or are more than 40 characters long.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Unit testing guidelines (1)

- Test edge cases
  If your partition has upper and lower bounds (e.g. length of strings, numbers, etc.) choose inputs at the edges of the range.
- Force errors
  Choose test inputs that force the system to generate all error messages. Choose test inputs that should generate invalid outputs.
- Fill buffers
  Choose test inputs that cause all input buffers to overflow.
- Repeat yourself
  Repeat the same test input or series of inputs several times.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Unit testing guidelines (2)

- Overflow and underflow
  If your program does numeric calculations, choose test inputs that cause it to calculate very large or very small numbers.
- Don’t forget null and zero
  If your program uses pointers or strings, always test with null pointers and strings.
- Keep count
  When dealing with lists and list transformation, keep count of the number of elements in each list and check that these are consistent after each transformation.
- One is different
  If your program deals with sequences, always test with sequences that have a single value.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Feature testing

- Features have to be tested to show that the functionality is implemented as expected and that the functionality meets the real needs of users.
  - For example, if your product has a feature that allows users to login using their Google account, then you have to check that this registers the user correctly and informs them of what information will be shared with Google.
  - You may want to check that it gives users the option to sign up for email information about your product.

Feature testing

- Normally, a feature that does several things is implemented by multiple, interacting, program units.
- These units may be implemented by different developers and all of these developers should be involved in the feature testing process.

Types of feature test

- Interaction tests
  - These test the interactions between the units that implement the feature. The developers of the units that are combined to make up the feature may have different understandings of what is required of that feature.
  - These misunderstandings will not show up in unit tests but may only come to light when the units are integrated.
  - The integration may also reveal bugs in program units, which were not exposed by unit testing.
- Usefulness tests
  - These test that the feature implements what users are likely to want.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

User stories for the sign-in with Google feature

- User registration
  As a user, I want to be able to login without creating a new account so that I don't have to remember another login id and password.
- Information sharing
  As a user, I want to know what information you will share with other companies. I want to be able to cancel my registration if I don't want to share this information.
- Email choice
  As a user, I want to be able to choose the types of email that I'll get from you when I register for an account.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Feature tests for sign-in with Google

- Initial login screen
  Test that the screen displaying a request for Google account credentials is correctly displayed when a user clicks on the 'Sign-in with Google' link. Test that the login is completed if the user is already logged in to Google.
- Incorrect credentials
  Test that the error message and retry screen is displayed if the user inputs incorrect Google credentials.

Feature tests for sign-in with Google

- Shared information
  Test that the information shared with Google is displayed, along with a cancel or confirm option. Test that the registration is cancelled if the cancel option is chosen.
- Email opt-in
  Test that the user is offered a menu of options for email information and can choose multiple items to opt-in to emails. Test that the user is not registered for any emails if no options are selected.

System and release testing

- System testing involves testing the system as a whole, rather than the individual system features.

System testing

- System testing should focus on four things:
  - Testing to discover if there are unexpected and unwanted interactions between the features in a system.
  - Testing to discover if the system features work together effectively to support what users really want to do with the system.
  - Testing the system to make sure it operates in the expected way in the different environments where it will be used.
  - Testing the responsiveness, throughput, security and other quality attributes of the system.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Scenario-based testing

- The best way to systematically test a system is to start with a set of scenarios that describe possible uses of the system and then work through these scenarios each time a new version of the system is created.
- Using the scenario, you identify a set of end-to-end pathways that users might follow when using the system.
- An end-to-end pathway is a sequence of actions from starting to use the system for the task, through to completion of the task.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Choosing a holiday destination
End-to-end pathways

1. User inputs departure airport and chooses to see only direct flights. User quits.
2. User inputs departure airport and chooses to see all flights. User quits.
3. User chooses destination country and chooses to see all flights. User quits.
4. User inputs departure airport and chooses to see direct flights. User sets filter specifying departure times and prices. User quits.
5. User inputs departure airport and chooses to see direct flights. User sets filter specifying departure times and prices. User selects a displayed flight and clicks through to airline website. User returns to holiday planner after booking flight.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Release testing

- Release testing is a type of system testing where a system that's intended for release to customers is tested.
- Preparing a system for release involves packaging that system for deployment (e.g. in a container if it is a cloud service) and installing software and libraries that are used by your product.
- You must define configuration parameters such as the name of a root directory, the database size limit per user and so on.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Release testing and System testing

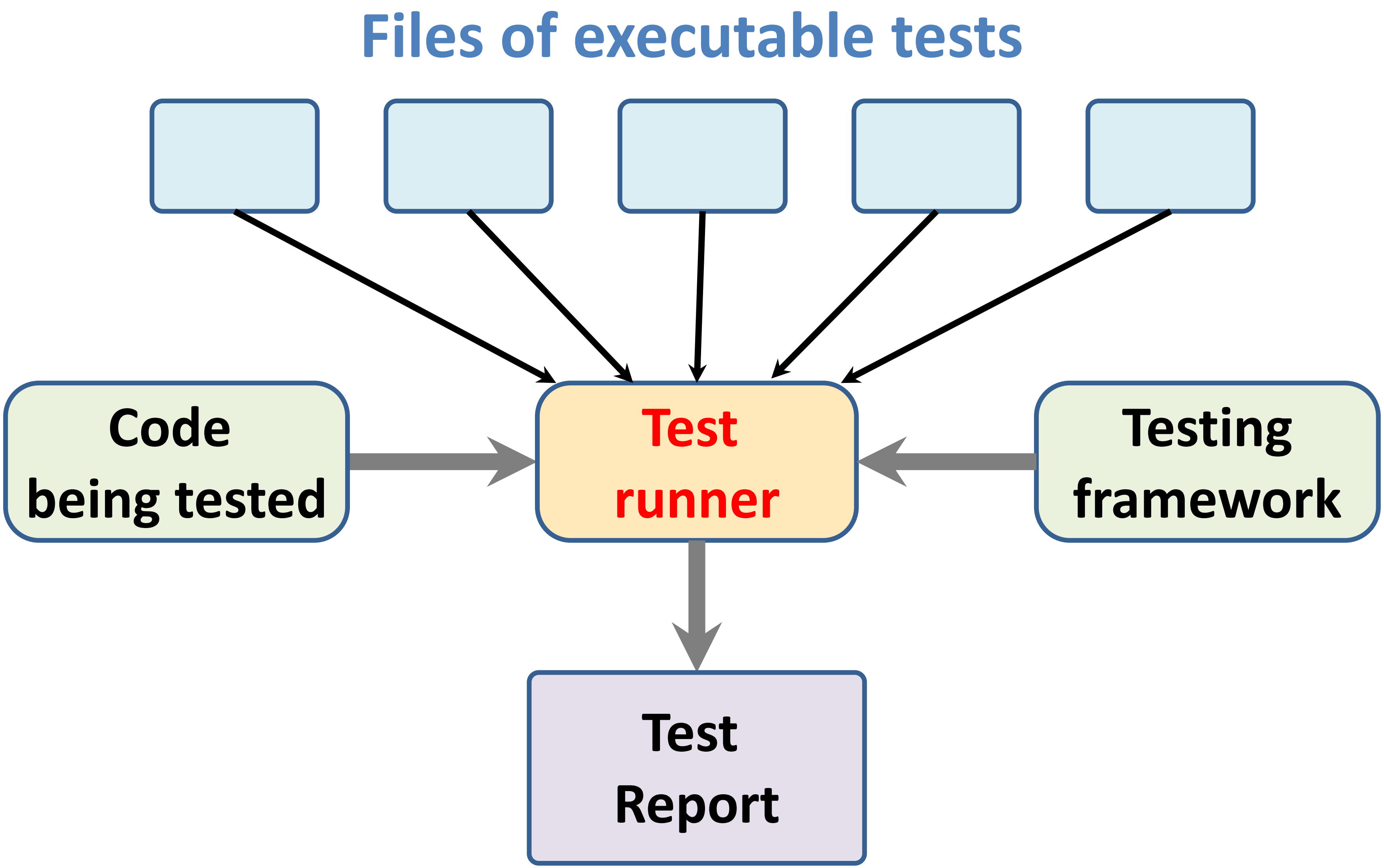
- The fundamental differences between release testing and system testing are:
  - Release testing tests the system in its real operational environment rather than in a test environment. Problems commonly arise with real user data, which is sometimes more complex and less reliable than test data.
  - The aim of release testing is to decide if the system is good enough to release, not to detect bugs in the system. Therefore, some tests that 'fail' may be ignored if these have minimal consequences for most users.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Test automation

- Automated testing is based on the idea that tests should be executable.
- An executable test includes the input data to the unit that is being tested, the expected result and a check that the unit returns the expected result.
- You run the test and the test passes if the unit returns the expected result.
- Normally, you should develop hundreds or thousands of executable tests for a software product.

Automated testing



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Test methods for an interest calculator

```
# TestInterestCalculator inherits attributes and methods from the class
# TestCase in the testing framework unittest

class TestInterestCalculator(unittest.TestCase):
    # Define a set of unit tests where each test tests one thing only
    # Tests should start with test_ and the name should explain what is being tested
    def test_zeroprincipal(self):
        #Arrange - set up the test parameters
        p = 0; r = 3; n = 31
        result_should_be = 0
        #Action - Call the method to be tested
        interest = interest_calculator (p, r, n)
        #Assert - test what should be true
        self.assertEqual(result_should_be, interest)

    def test_yearly_interest(self):
        #Arrange - set up the test parameters
        p = 17000; r = 3; n = 365
        #Action - Call the method to be tested
        result_should_be = 270.36
        interest = interest_calculator(p, r, n)
        #Assert - test what should be true
        self.assertEqual(result_should_be, interest)
```

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Automated tests

- It is good practice to structure automated tests into three parts:
  1. Arrange
     - You set up the system to run the test. This involves defining the test parameters and, if necessary, mock objects that emulate the functionality of code that has not yet been developed.
  2. Action
     - You call the unit that is being tested with the test parameters.
  3. Assert
     - You make an assertion about what should hold if the unit being tested has executed successfully. AssertEquals: checks if its parameters are equal.

Executable tests for the namecheck function (1)

```
import unittest
from RE_checker import namecheck

class TestNameCheck (unittest.TestCase):

    def test_alphaname (self):
        self.assertTrue (namecheck ('Sommerville'))

    def test_doublequote (self):
        self.assertFalse (namecheck ("Thisis'maliciouscode'"))

    def test_namestartswithhyphen (self):
        self.assertFalse (namecheck ('-Sommerville'))

    def test_namestartswithquote (self):
        self.assertFalse (namecheck ("'Reilly"))

    def test_nametoolong (self):
        self.assertFalse (namecheck ('Thisisalongstringwithmorethen40charactersfrombeginningtoend'))

    def test_nametooshort (self):
        self.assertFalse (namecheck ('S'))
```

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Executable tests for the namecheck function (2)

```
def test_namewithdigit (self):
    self.assertFalse (namecheck('C-3PO'))

def test_namewithdoublehyphen (self):
    self.assertFalse (namecheck ('--badcode’))

def test_namewithhyphen (self):
    self.assertTrue (namecheck ('Washington-Wilson'))

def test_namewithinvalidchar (self):
    self.assertFalse (namecheck('Sommer_ville'))

def test_namewithquote (self):
    self.assertTrue (namecheck ("O'Reilly"))

def test_namewithspaces (self):
    self.assertFalse (namecheck ('Washington Wilson'))

def test_shortname (self):
    self.assertTrue ('Sx')

def test_thiswillfail (self)
    self.assertTrue (namecheck ("O Reilly"))
```

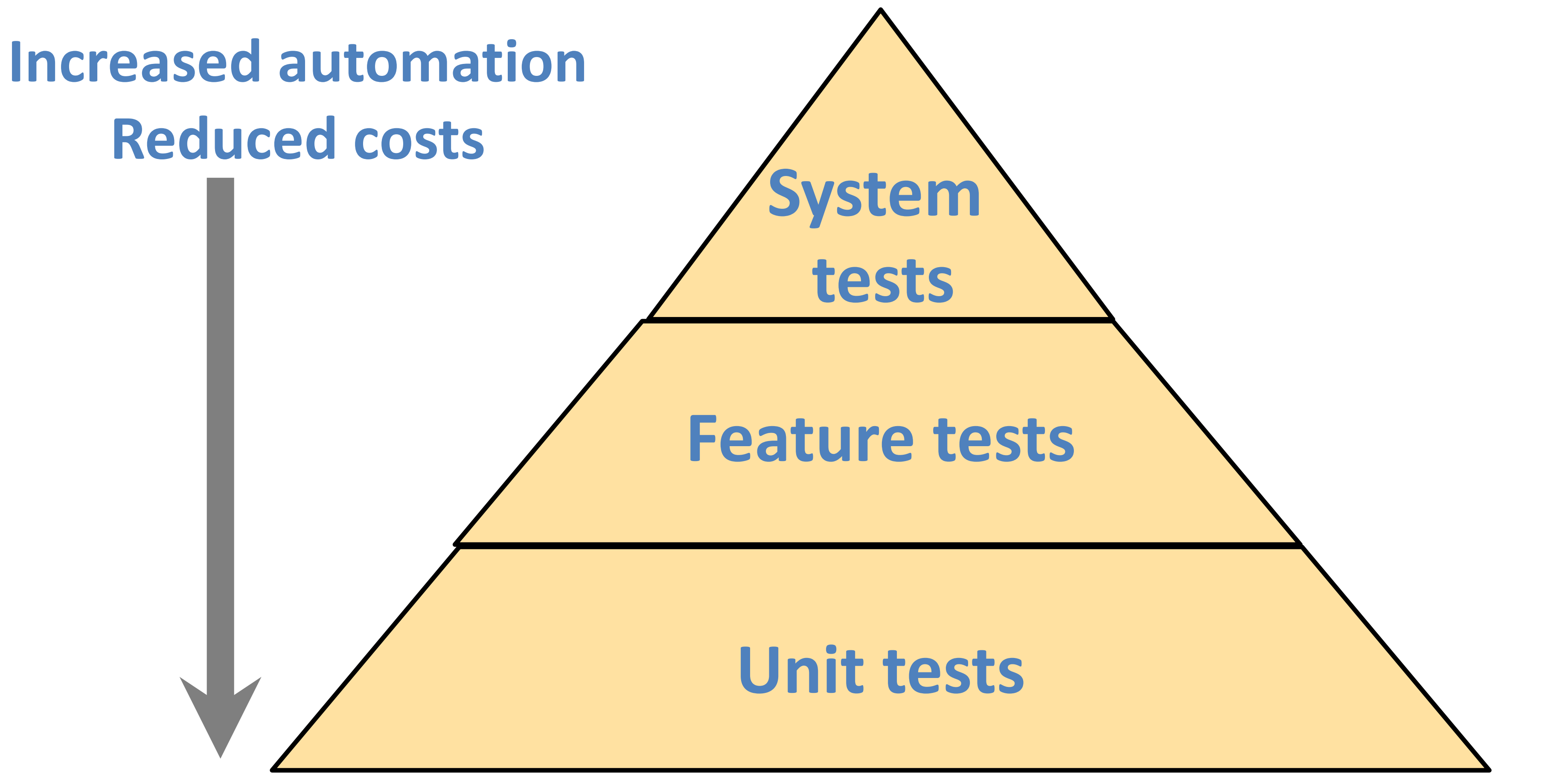
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Code to run unit tests from files

```
import unittest

loader = unittest.TestLoader()

#Find the test files in the current directory

tests = loader.discover('.')

#Specify the level of information provided by the test runner

testRunner = unittest.runner.TextTestRunner(verbosity=2)
testRunner.run(tests)
```

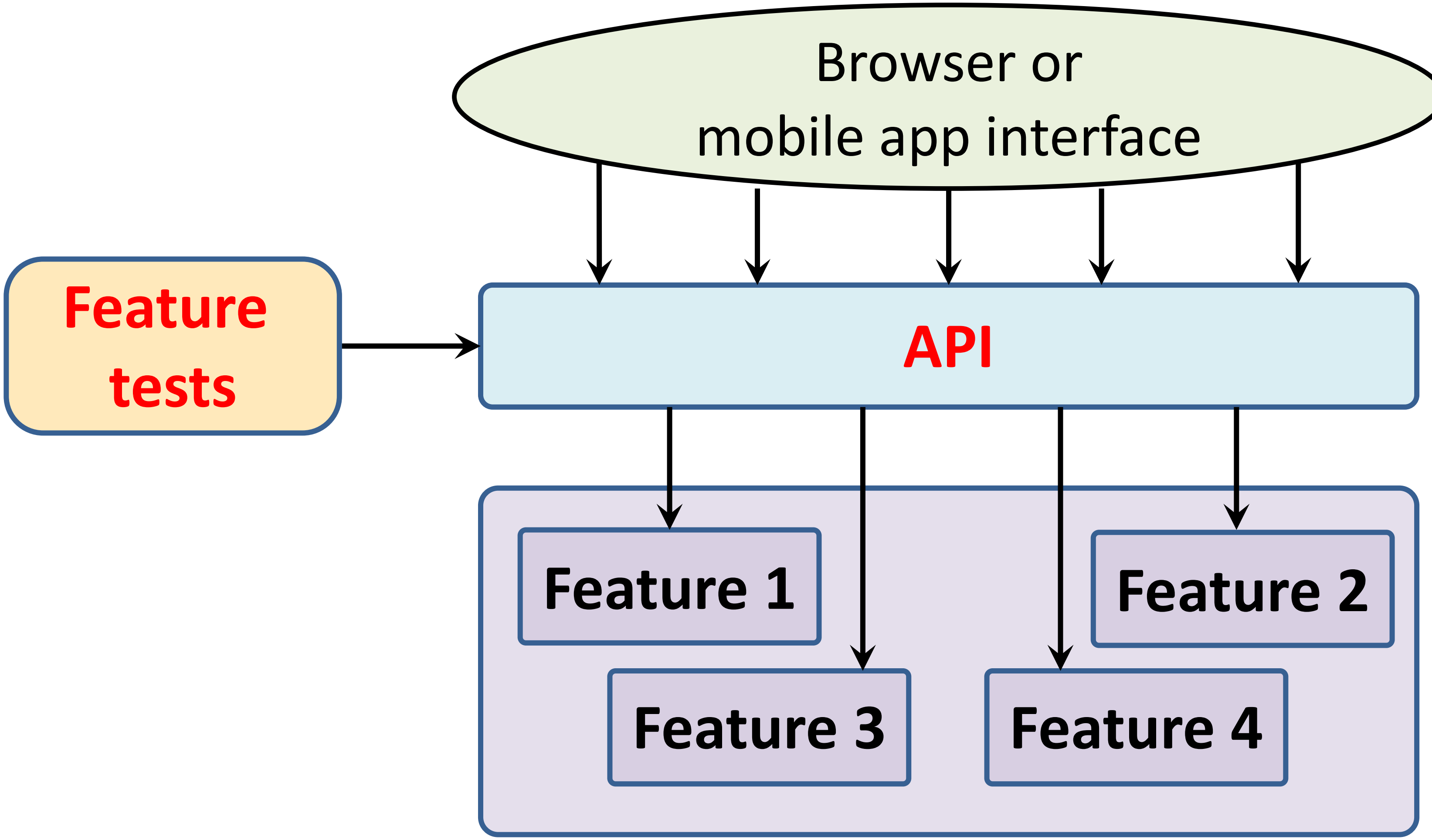
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

The test pyramid



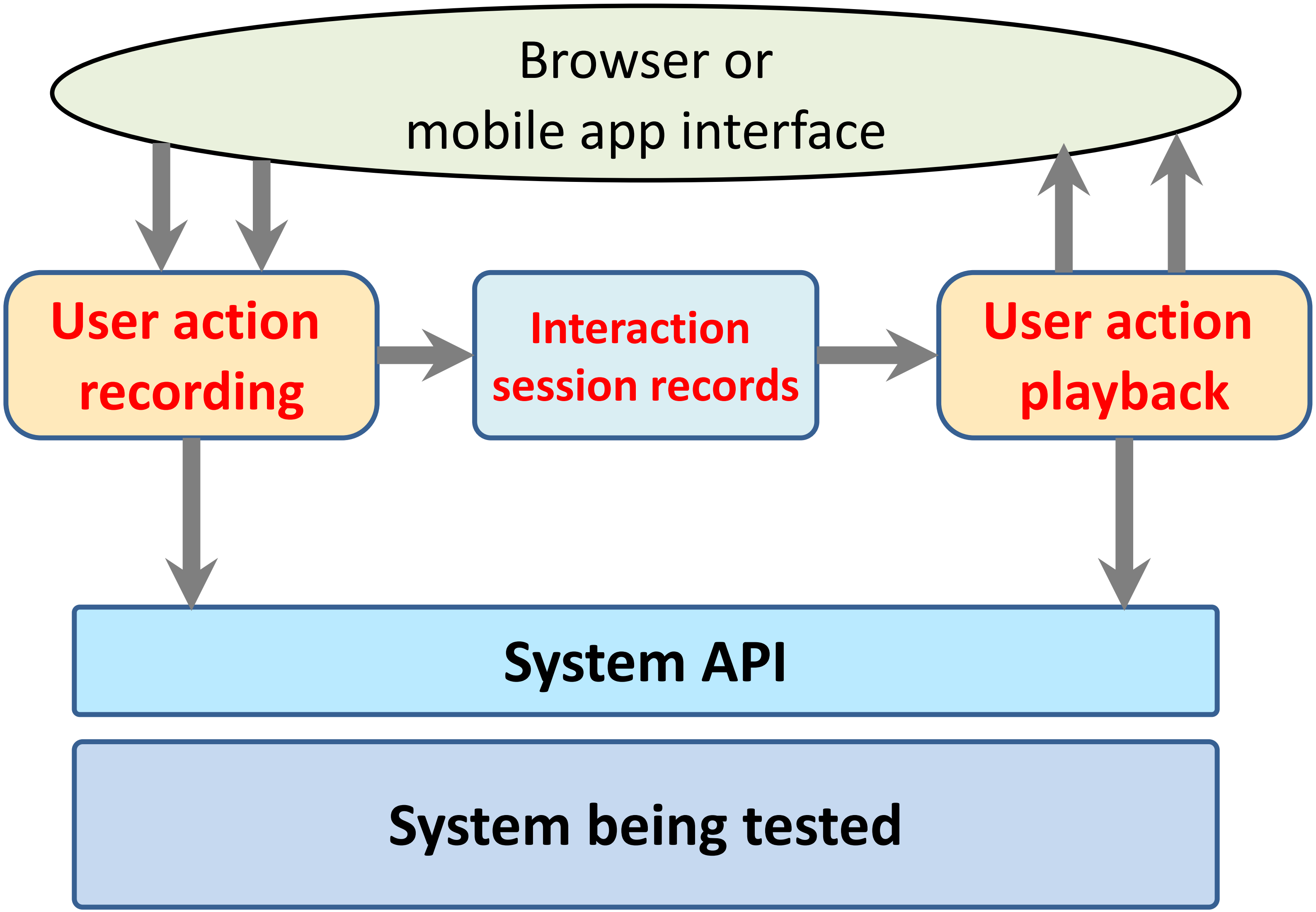
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Feature editing through an API



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Interaction recording and playback



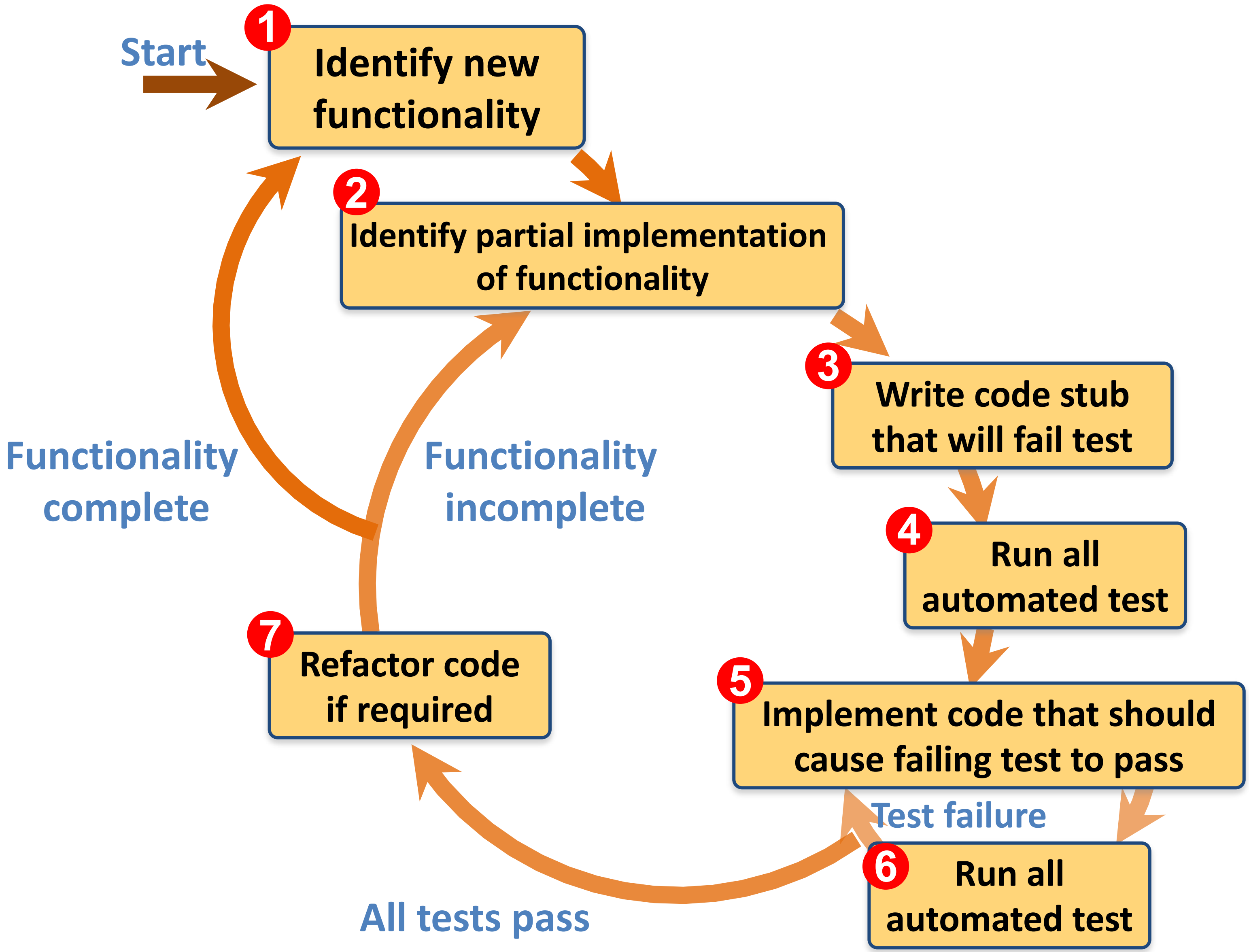
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Test-driven development (TDD)

- Test-driven development (TDD) is an approach to program development that is based around the general idea that you should write an executable test or tests for code that you are writing before you write the code.
- It was introduced by early users of the Extreme Programming agile method, but it can be used with any incremental development approach.
- Test-driven development works best for the development of individual program units and it is more difficult to apply to system testing.
- Even the strongest advocates of TDD accept that it is challenging to use this approach when you are developing and testing systems with graphical user interfaces.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Test-driven development (TDD)



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Stages of test-driven development

1. **Identify partial implementation**
   Break down the implementation of the functionality required into smaller mini-units. Choose one of these mini-units for implementation.
2. **Write mini-unit tests**
   Write one or more automated tests for the mini-unit that you have chosen for implementation. The mini-unit should pass these tests if it is properly implemented.
3. **Write a code stub that will fail test**
   Write incomplete code that will be called to implement the mini-unit. You know this will fail.
4. **Run all existing automated tests**
   All previous tests should pass. The test for the incomplete code should fail.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Stages of test-driven development

5. **Implement code that should cause the failing test to pass**
   Write code to implement the mini-unit, which should cause it to operate correctly
6. **Rerun all automated tests**
   If any tests fail, your code is probably incorrect. Keep working on it until all tests pass.
7. **Refactor code if necessary**
   If all tests pass, you can move on to implementing the next mini-unit. If you see ways of improving your code, you should do this before the next stage of implementation.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Benefits of test-driven development

- It is a systematic approach to testing in which tests are clearly linked to sections of the program code.
  - This means you can be confident that your tests cover all of the code that has been developed and that there are no untested code sections in the delivered code.
- The tests act as a written specification for the program code. In principle at least, it should be possible to understand what the program does by reading the tests.
- Debugging is simplified because, when a program failure is observed, you can immediately link this to the last increment of code that you added to the system.
- TDD leads to simpler code as programmers only write code that's necessary to pass tests. They don't over-engineer their code with complex features that aren't needed.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Reasons for not using TDD

- TDD discourages radical program change
- I focused on the tests rather than the problem I was trying to solve
- I spent too much time thinking about implementation details rather than the programming problem
- It is hard to write ‘bad data’ tests

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Security testing

- Security testing aims to find vulnerabilities that may be exploited by an attacker and to provide convincing evidence that the system is sufficiently secure.
- The tests should demonstrate that the system can resist attacks on its availability, attacks that try to inject malware and attacks that try to corrupt or steal users' data and identity.
- Comprehensive security testing requires specialist knowledge of software vulnerabilities and approaches to testing that can find these vulnerabilities.

Risk-based security testing

- A risk-based approach to security testing involves identifying common risks and developing tests to demonstrate that the system protects itself from these risks.
- You may also use automated tools that scan your system to check for known vulnerabilities, such as unused HTTP ports being left open.
- Based on the risks that have been identified, you then design tests and checks to see if the system is vulnerable.
- It may be possible to construct automated tests for some of these checks, but others inevitably involve manual checking of the system's behaviour and its files.

Risk analysis

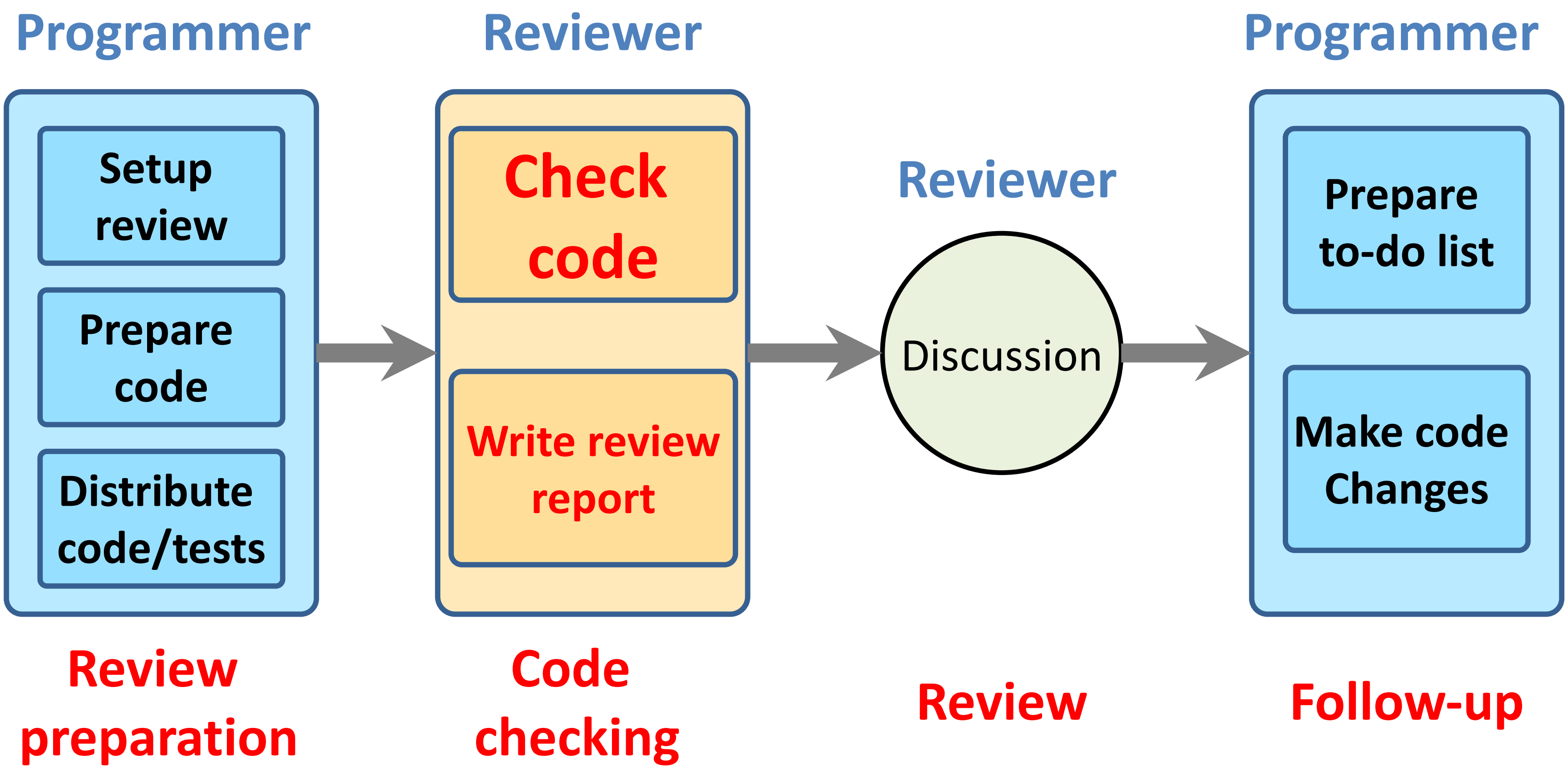
- Once you have identified security risks, you then analyze them to assess how they might arise.
  - The user has set weak passwords that can be guessed by an attacker.
  - The system's password file has been stolen and passwords discovered by attacker.
- Develop tests to check some of these possibilities.
  - For example, you might run a test to check that the code that allows users to set their passwords always checks the strength of passwords.

Code reviews

- Code reviews involve one or more people examining the code to check for errors and anomalies and discussing issues with the developer.
- If problems are identified, it is the developer's responsibility to change the code to fix the problems.
- Code reviews complement testing. They are effective in finding bugs that arise through misunderstandings and bugs that may only arise when unusual sequences of code are executed.
- Many software companies insist that all code has to go through a process of code review before it is integrated into the product codebase.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Code reviews



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Summary

- The aim of program testing is to find bugs and to show that a program does what its developers expect it to do.
- Four types of testing that are relevant to software products are
functional testing, user testing,
load and performance testing and security testing.
- Unit testing involves testing program units such as functions or class methods that have a single responsibility.
- Feature testing focuses on testing individual system features.

Summary

- System testing tests the system as a whole to check for unwanted interactions between features and between the system and its environment.
- Identifying equivalence partitions, in which all inputs have the same characteristics, and choosing test inputs at the boundaries of these partitions, is an effective way of finding bugs in a program.
- User stories may be used as a basis for deriving feature tests.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Summary

- Test automation is based on the idea that tests should be executable. You develop a set of executable tests and run these each time you make a change to a system.
- The structure of an automated unit test should be arrange-action-assert. You set up the test parameters, call the function or method being tested, and make an assertion of what should be true after the action has been completed.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Summary

- Test-driven development is an approach to development where executable tests are written before the code. Code is then developed to pass the tests.
- A disadvantage of test-driven development is that programmers focus on the detail of passing tests rather than considering the broader structure of their code and algorithms used.

Summary

- Security testing may be risk driven where a list of security risks is used to identify tests that may identify system vulnerabilities.
- Code reviews are an effective supplement to testing. They involve people checking the code to comment on the code quality and to look for bugs.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

DevOps and Code Management: Code management and DevOps automation

Outline

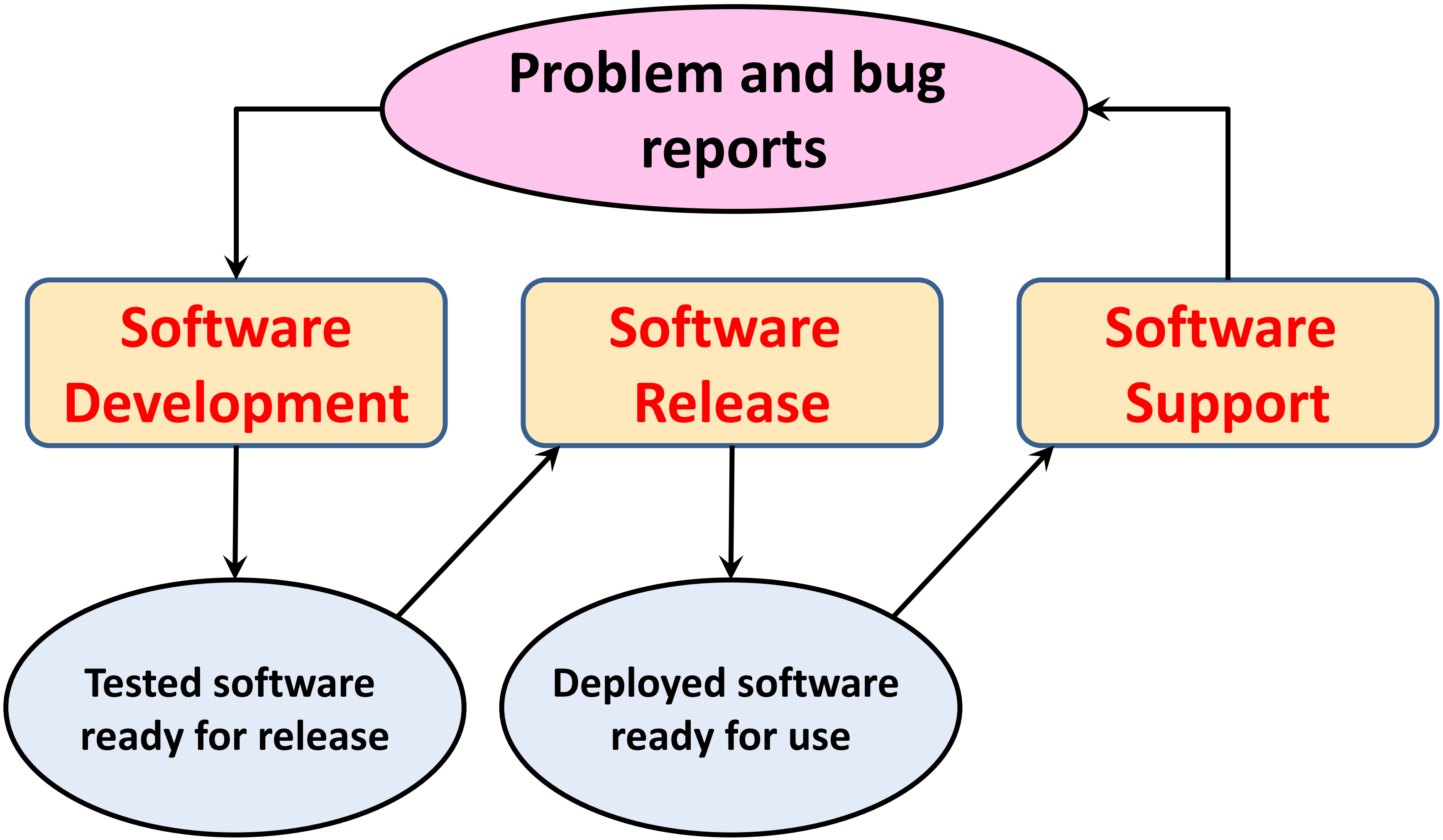
- **Source code management**
- **DevOps automation**
- **DevOps measurement**

Software support

- Traditionally, separate teams were responsible software development, software release and software support.
- The development team passed over a ‘final’ version of the software to a release team.
  - Built a release version, tested this and prepared release documentation before releasing the software to customers.
- A third team was responsible for providing customer support.
  - The original development team were sometimes also responsible for implementing software changes.
  - Alternatively, the software may have been maintained by a separate ‘maintenance team’.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software Development, release and support



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

DevOps

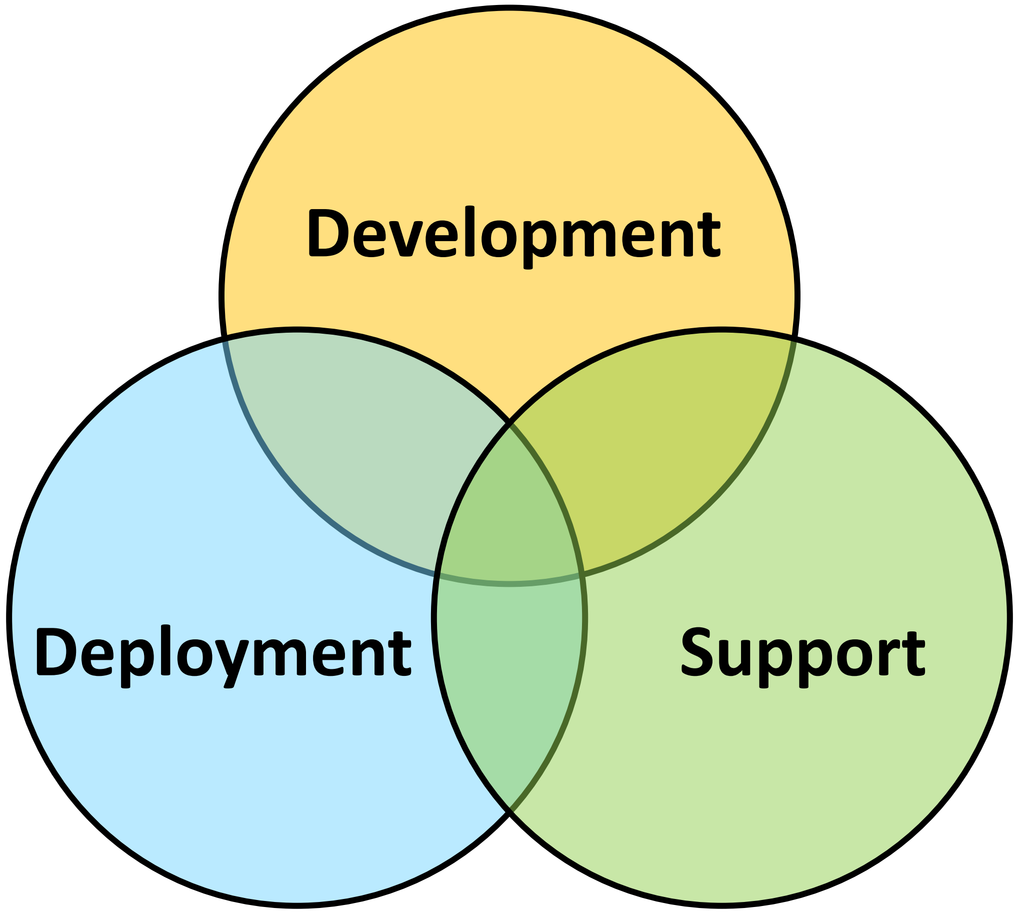
- There are inevitable delays and overheads in the traditional support model.
- To speed up the release and support processes, an alternative approach called DevOps (Development + Operations) has been developed.

DevOps

- Three factors led to the development and widespread adoption of DevOps:
  - Agile software engineering reduced the development time for software, but the traditional release process introduced a bottleneck between development and deployment.
  - Amazon re-engineered their software around services and introduced an approach in which a service was developed and supported by the same team. Amazon's claim that this led to significant improvements in reliability was widely publicized.
  - It became possible to release software as a service, running on a public or private cloud. Software products did not have to be released to users on physical media or downloads.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

DevOps



Multi-skilled DevOps team

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

DevOps principles

- **Everyone is responsible for everything**
  All team members have joint responsibility for developing, delivering and supporting the software.
- **Everything that can be automated should be automated**
  All activities involved in testing, deployment and support should be automated if it is possible to do so. There should be minimal manual involvement in deploying software.
- **Measure first, change later**
  DevOps should be driven by a measurement program where you collect data about the system and its operation. You then use the collected data to inform decisions about changing DevOps processes and tools.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Benefits of DevOps

Faster deployment	Software can be deployed to production more quickly because communication delays between the people involved in the process are dramatically reduced.
Reduced risk	The increment of functionality in each release is small so there is less chance of feature interactions and other changes causing system failures and outages.
Faster repair	DevOps teams work together to get the software up and running again as soon as possible.
More productive teams	DevOps teams are happier and more productive than the teams involved in the separate activities.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Code management

- Code management is a set of software-supported practices that is used to manage an evolving codebase.
- During the development of a software product, the development team will probably create tens of thousands of lines of code and automated tests.
- These will be organized into hundreds of files. Dozens of libraries may be used, and several, different programs may be involved in creating and running the code.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

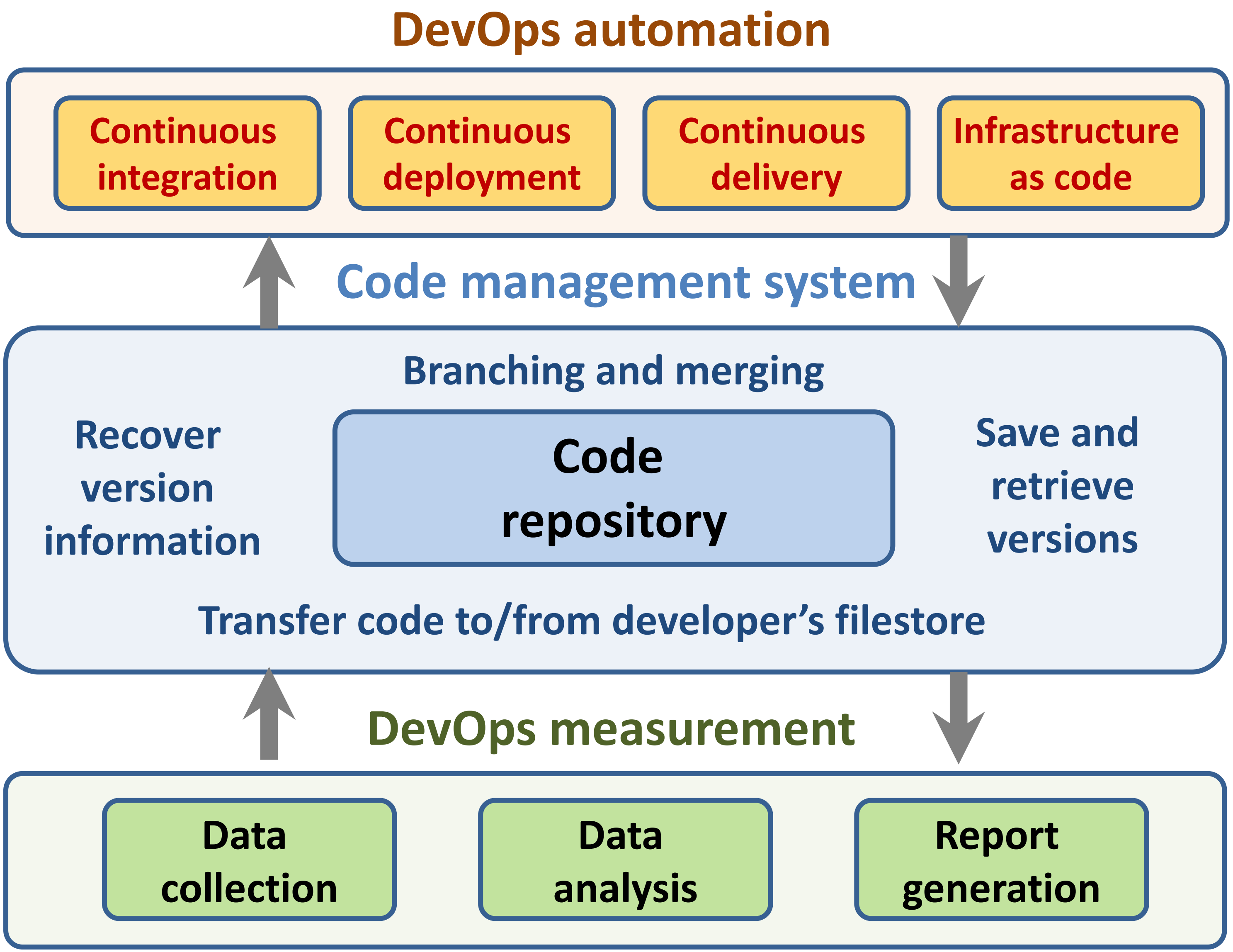
Code management

- You need code management to ensure that changes made by different developers do not interfere with each other, and to create different product versions.
- Code management tools make it easy to create an executable product from its source code files and to run automated tests on that product.

Code management and DevOps

- Source code management, combined with automated system building, is essential for professional software engineering.
- In companies that use DevOps, a modern code management system is a fundamental requirement for ‘automating everything’.
- Not only does it store the project code that is ultimately deployed, it also stores all other information that is used in DevOps processes.
- DevOps automation and measurement tools all interact with the code management system

Code management and Devops



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Code management fundamentals

- Code management systems provide a set of features that support four general areas:
- **Code transfer**
  - Developers take code into their personal file store to work on it then return it to the shared code management system.
- **Version storage and retrieval**
  - Files may be stored in several different versions and specific versions of these files can be retrieved.
- **Merging and branching**
  - Parallel development branches may be created for concurrent working. Changes made by developers in different branches may be merged.
- **Version information**
  - Information about the different versions maintained in the system may be stored and retrieved

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Code repository

- All source code management systems have the general form with a shared repository and a set of features to manage the files in that repository:
  - All source code files and file versions are stored in the repository, as are other artefacts such as configuration files, build scripts, shared libraries and versions of tools used.
  - The repository includes a database of information about the stored files such as version information, information about who has changed the files, what changes were made at what times, and so on.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Code repository

- Files can be transferred to and from the repository and information about the different versions of files and their relationships may be updated.
  - Specific versions of files and information about these versions can always be retrieved from the repository.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Features of code management systems

Version and release identification

Change history recording

Independent development

Project support

Storage management

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Git

- In 2005, Linus Torvalds, the developer of Linux, revolutionized source code management by developing a distributed version control system (DVCS) called Git to manage the code of the Linux kernel.
- This was geared to supporting large-scale open source development. It took advantage of the fact that storage costs had fallen to such an extent that most users did not have to be concerned with local storage management.
- Instead of only keeping the copies of the files that users are working on, Git maintains a clone of the repository on every user's computer

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

git

Source: https://git-scm.com/

Benefits of distributed code management

Resilience

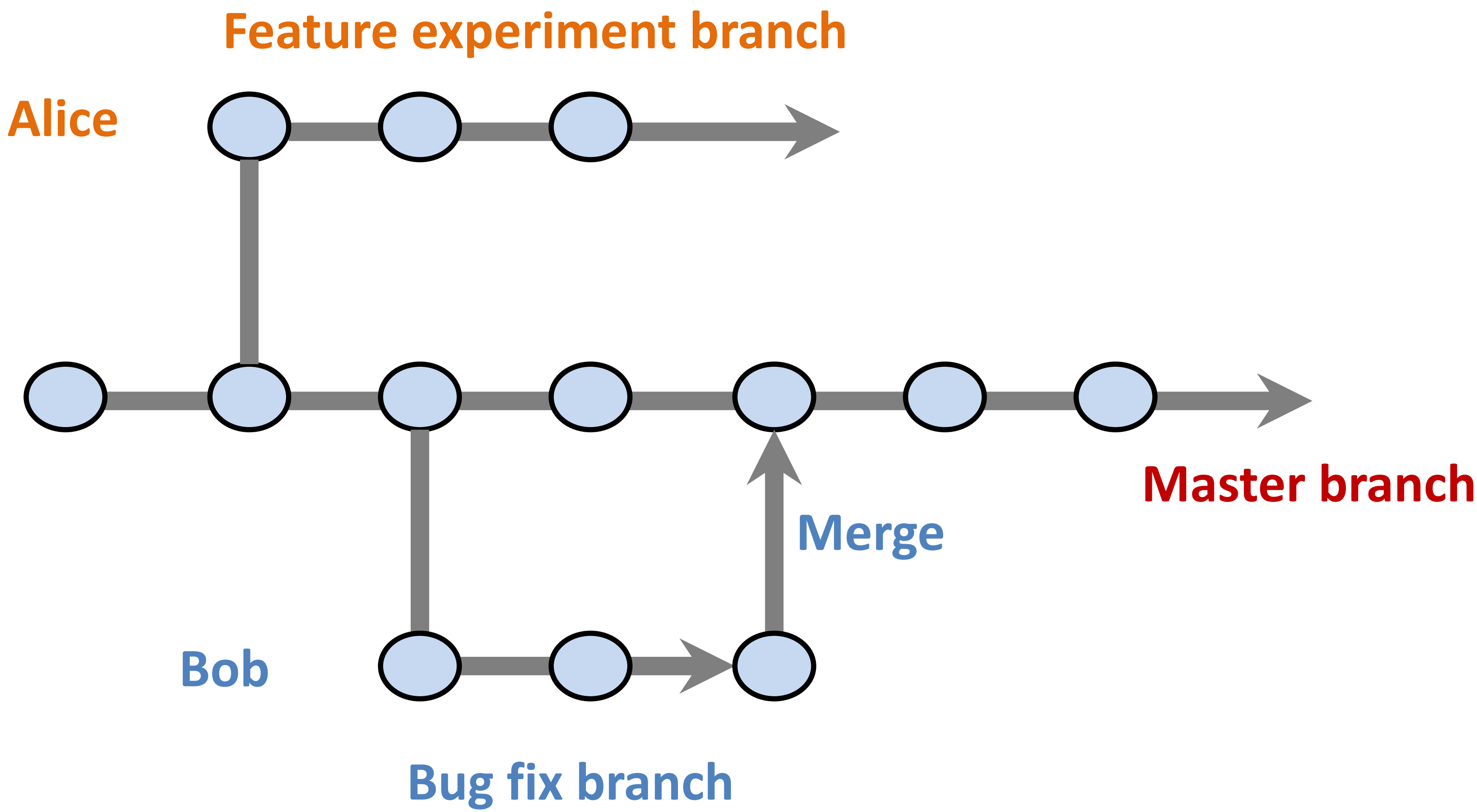
Speed

Flexibility

Branching and merging

- Branching and merging are fundamental ideas that are supported by all code management systems.
- A branch is an independent, stand-alone version that is created when a developer wishes to change a file.
- The changes made by developers in their own branches may be merged to create a new shared branch.
- The repository ensures that branch files that have been changed cannot overwrite repository files without a merge operation.

Branching and merging



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

DevOps automation

- By using DevOps with automated support, you can dramatically reduce the time and costs for integration, deployment and delivery.
- Everything that can be, should be automated is a fundamental principle of DevOps.
- As well as reducing the costs and time required for integration, deployment and delivery, process automation also makes these processes more reliable and reproducible.
- Automation information is encoded in scripts and system models that can be checked, reviewed, versioned and stored in the project repository.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Aspects of DevOps automation

Continuous integration	Each time a developer commits a change to the project's master branch, an executable version of the system is built and tested.
Continuous delivery	A simulation of the product's operating environment is created and the executable software version is tested.
Continuous deployment	A new release of the system is made available to users every time a change is made to the master branch of the software.
Infrastructure as code	Machine-readable models of the infrastructure (network, servers, routers, etc.) on which the product executes are used by configuration management tools to build the software's execution platform.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Characteristics of infrastructure as code

Visibility

Reproducibility

Reliability

Recovery

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

DevOps measurement

- After you have adopted DevOps, you should try to continuously improve your DevOps process to achieve faster deployment of better-quality software.
- There are four types of software development measurement:
  - Process measurement
  - Service measurement
  - Usage measurement
  - Business success measurement

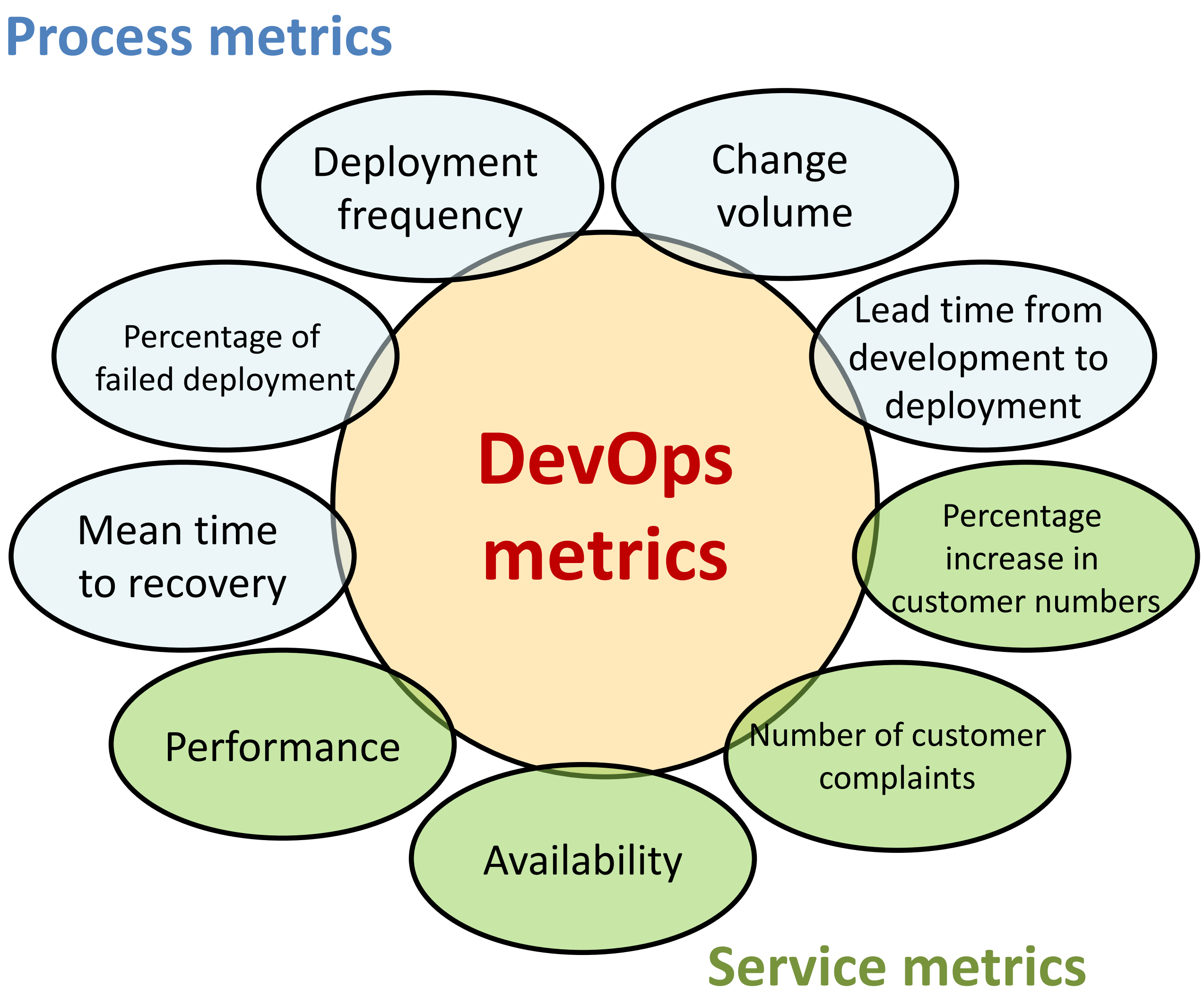
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Automating measurement

- As far as possible, the DevOps principle of automating everything should be applied to software measurement.
- You should instrument your software to collect data about itself and you should use a monitoring system to collect data about your software's performance and availability.
- Some process measurements can also be automated.
  - However, there are problems in process measurement because people are involved. They work in different ways, may record information differently and are affected by outside influences that affect the way they work.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Metrics used in the DevOps scorecard



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Summary

- DevOps is the integration of software development and the management of that software once it has been deployed for use. The same team is responsible for development, deployment and software support.
- The benefits of DevOps are faster deployment, reduced risk, faster repair of buggy code and more productive teams.
- Source code management is essential to avoid changes made by different developers interfering with each other.

Summary

- All code management systems are based around a shared code repository with a set of features that support code transfer, version storage and retrieval, branching and merging and maintaining version information.
- Git is a distributed code management system that is the most widely used system for software product development. Each developer works with their own copy of the repository which may be merged with the shared project repository.

Summary

- DevOps is the integration of software development and the management of that software once it has been deployed for use. The same team is responsible for development, deployment and software support.
- The benefits of DevOps are faster deployment, reduced risk, faster repair of buggy code and more productive teams.
- Source code management is essential to avoid changes made by different developers interfering with each other.

Summary

- Continuous integration means that as soon as a change is committed to a project repository, it is integrated with existing code and a new version of the system is created for testing.
- Automated system building tools reduce the time needed to compile and integrate the system by only recompiling those components and their dependents that have changed.
- Continuous deployment means that as soon as a change is made, the deployed version of the system is automatically updated. This is only possible when the software product is delivered as a cloud-based service.

Summary

- Infrastructure as code means that the infrastructure (network, installed software, etc.) on which software executes is defined as a machine-readable model. Automated tools, such as Chef and Puppet, can provision servers based on the infrastructure model.
- Measurement is a fundamental principle of DevOps. You may make both process and product measurements. Important process metrics are deployment frequency, percentage of failed deployments, and mean time to recovery from failure.

References

- Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.
- Ian Sommerville (2015), Software Engineering, 10th Edition, Pearson.
- Titus Winters, Tom Manshreck, and Hyrum Wright (2020), Software Engineering at Google: Lessons Learned from Programming Over Time, O'Reilly Media.
- Project Management Institute (2021), A Guide to the Project Management Body of Knowledge (PMBOK Guide) – Seventh Edition and The Standard for Project Management, PMI
- Project Management Institute (2017), A Guide to the Project Management Body of Knowledge (PMBOK Guide), Sixth Edition, Project Management Institute
- Project Management Institute (2017), Agile Practice Guide, Project Management Institute