DAT321/DIT843: Quality Assurance and Testing

Questions about the exam contact:
Examiner: Francisco Gomes, tel. 073 856 4439, francisco.gomes@cse.gu.se
For questions about the exam, call the number above.

Allowed Aid:

- English dictionary
- **NOT ALLOWED:** Anything else not explicitly mentioned above (including additional books, other notes, previous exams, or any form of electronic device: dictionaries, agendas, computers, mobile phones, etc.)

The questions in this exam refer to the **ISO 25010:2011** that categorises internal and external software quality attributes into eight characteristics, as well as the quality in use characteristics.

Important advice:

- Each question has points assigned shown in the square brackets. When the question is broken down into smaller sub-questions the part of the points for those specific sub-questions are also shown. Example: 1. [10 pts]. 1a. [2 pts] 1b. [8 pts].
- You must write clear, readable, understandable, and unambiguous answers. If we cannot easily read what you are trying to write, then we cannot positively judge your knowledge and you will most likely fail.
- Motivate your answers (a simple statement of facts not answering the question is invalid). Whenever possible refer to modules, features and information about the software product or software development process described in the context.
- Start each question on a new paper. Sort your answers in order (by question and sub-task) before handing them in.
- Write your student code on each page and put the number of the question on every paper.

The points and the corresponding grade are presented below (100 points in total):

- 00 – 49: U (Fail)
- 50 - 69: 3 (Pass)
- 70 - 89: 4 (Pass with credit)
- 90 - 100: 5 (Pass with distinction)

The exam review will be done via Zoom scheduled for:
Date: 2022-02-03 between 14:00 – 15:30.
Zoom Link: https://chalmers.zoom.us/j/6651614233
Password: DIT043

Description of Context:

The questions in this exam are related to the context below. Note that you **must** justify your exam answers with i) the theory and terminology from the subject studied during the course and ii) their connections to the elements in this context, such as the teams, product, processes, architecture, customers, system under test (SUT), etc.

You are hired by the company Vaccine-Co responsible for handling the booking of COVID-19 vaccines during the current pandemic. This system will be deployed in a country where none of the population has started being vaccinated yet. Vaccination of people in the risk groups is ongoing already by another agency. There are still circa 20 million individuals outside the risk groups (age between 18 – 50 years old) that are eligible for the vaccine and should receive them as soon as possible. Healthcare in this country is public and accessible to all, meaning that everyone will be vaccinated for free. The goal is to release our software product within two months. This is a high priority of the company as the government and media create pressure to book vaccines.

Features and architecture of the product:

The software product uses a publish/subscribe architectural style (Figure 1), where each component is independent of each other. Their communication is handled by a **Broker Component** that sends and receives messages between components.

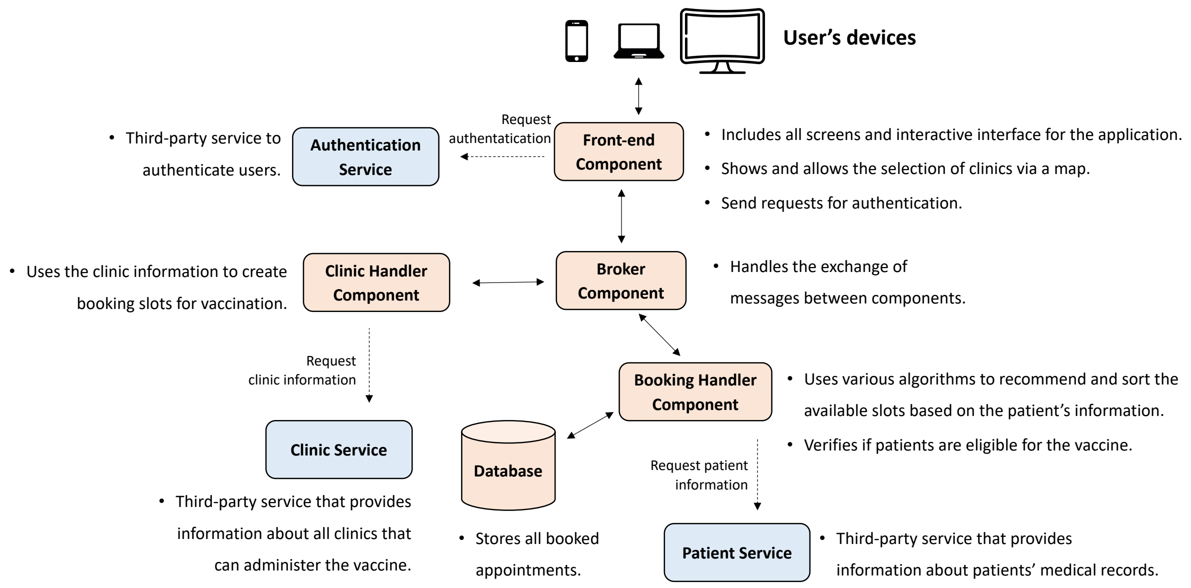
User's devices
Front-end Component
• Includes all screens and interactive interface for the application.
• Shows and allows the selection of clinics via a map.
• Send requests for authentication.
Request authentatication
Authentication Service
• Third-party service to authenticate users.
Broker Component
• Handles the exchange of messages between components.
Clinic Handler Component
• Uses the clinic information to create booking slots for vaccination.
Request clinic information
Clinic Service
• Third-party service that provides information about all clinics that can administer the vaccine.
Database
• Stores all booked appointments.
Booking Handler Component
• Uses various algorithms to recommend and sort the available slots based on the patient's information.
• Verifies if patients are eligible for the vaccine.
Request patient information
Patient Service
• Third-party service that provides information about patients' medical records.

Figure 1 – Architecture of the Software Product

The graphical user interface (GUI) of your system can be accessed by end-users in different devices such as mobile phones, personal computers, and tablets. The variations of interface and messages/requests sent from the device are handled by the **Front-end Component** which is installed in the user's device. To access the software, users must first login using their social security number. The authentication is done by a third-party application via the **Authentication Service** and is not developed by your organization.

Your system retrieves the information of clinics that administer the vaccine (opening hour, staff, available doses, etc.) from a **Clinic Service**. That service is hosted and handled by the public healthcare agency, such that you cannot alter the registry. The **Clinic Handler Component** is responsible for fetching that information as well as creating the vaccination slots and storing them in our own database.

Our system has a **Booking Handler Component** that suggests specific vaccination slots to users. Those slots are sorted based on the patient's medical records (e.g., age, past diseases, and treatments) retrieved from the healthcare **Patient Service**. The goal is to prioritise available slots to patients more vulnerable to COVID-19. The booked appointments are then stored in a **database**.

Your team has access to the source code and documentation of all components. However, the owners of the services (Authentication, Clinic and Patient) only provide documentation explaining the services' features such that you do not have access to their source code.

Development process and team:

There is only one development team named Team X composed of 8 people: three developers, one architect, one product owner, one process manager, one user-experience (UX) designer, and one quality assurance engineer (you). Your entire team has worked with agile development and Continuous Integration (CI) in the past. However, the toolchain to create a CI pipeline in your development process is not done yet and your team will have to create one.

1. **[30 pts]** Using the context above, answer the following questions:

   a. [15 pts] Choose **three** software product quality's characteristics, **and, for each,** provide examples of why they are relevant to the software product at Vaccine-Co. When justifying your choices, you **must** use the features and components/services listed in Figure 1 as sources for your examples.

Any three examples of the 8 below would work. The key aspect is **description** of the software product quality characteristic **in connection** to the context above, especially the components/service. The number of points depends on the clarity of the justification behind the relevance.

- 5pts per characteristic. (Each description of relevance: 3-5 points, Using features of Components/Services: 1-3 points).
- (There should be 3 characteristics at least … if there are more, we will look at the totality of what you wrote. For instance, 4 examples in which only 3 of them are correct will not award you full points since there are still wrong descriptions provided.)
- Be careful and **avoid circular definition / justifications** by using the chosen characteristic in the text of the definition. For instance, avoid answers such as: maintainability is important for the system to be maintainable; reliability is important to make a reliable system, etc.). Focus on the unique aspect of each characteristic.

Maintainability: Should be connected to changes in any of the components/services. In theory, any component or service can be used as example. Some examples are requests to update the features of the services (e.g., owner removes or adds news features, such as changing the forms of authentication, or added / removed fields in the medical records of patients), or requests to change the features in one of the components such as different algorithms for the Booking Handler component). The relevance must be justified by the need to reduce the impact or minimize effort of modifications to the software product.

Reliability: Should be connected to the ability to perform the functionalities under specific conditions and periods of time. In theory, any component or service can be used as example. Some examples are that the system should not crash due to high workload (condition) of access to book vaccines during the same day or hour (time), or that the system should not cause inconsistences (failures) in the booked slots due to concurrency booking (time and condition), or that the services should not crash when accessed by our system when trying to book a slot (time) and information from patient / clinic is needed from the service (condition).

Portability: Should be connected to the ability to run the application on different devices (hardware). This is a relevant characteristic for the Front-end Component as the context describes that the application GUI can be accessed by different devices. Some examples of features are being able to book the vaccina by using the GUI on a mobile phone, laptop, tablet or other specified device.

Compatibility: Should be connected to the application's ability to share information with other applications. Therefore, this is a relevant characteristic for the components accessing the services (Front-end and Authentication, Clinic Handler and Clinic, or Booking Handler and Patient). This is a relevant characteristic because sharing the information between those components enables our application to retrieve and request information from other software systems (the services) that are required for our system to work. Some examples of features is the exchange of patient information to verify whether they are eligible for the vaccine, or retrieving information from the clinics so that our application can create and register a booking slot for the vaccine, etc. If any part of the system can be referenced it would be the Broker to enable all components to share and exchange information and function together.

Usability: Should be connected to the ability of our system to be used by its target users, hence it is relevant for the Front-end Component. Usability is relevant because the user needs an accurate and functioning GUI to successfully booking their vaccine (that is what they use to interact with the system). Some examples are creating a GUI that is straightforward to avoid confusion or mistakes during the booking; creating an inclusive interface for people that have disabilities using, e.g., graphical and interactive elements (layout, color, etc.) that brings calmness to the user, etc.

Security: Should be connected to the ability of providing the correct level of access to the application. Security is relevant because it enables protection of the information and data of the user and their medical records from other individuals or organizations. The part of the system connected to security is the Authentication Service and the Front-end. The examples should be related to the usage of login/password, or other authentication method to verify the identity of the user trying to book a vaccine.

Functional Suitability: Should be connected to the ability of the system to meet the needs of the user. Functional suitability is important because it allows the user to book the vaccines according to their choice of clinic or available time slot. Any of the components and services are connected to Functional Suitability, so the example should be cohesive to the choice of component and feature. For instance, retrieving the clinics' information (Clinic Handler + Clinic Service) is needed to obtain the availability of slots, whereas the patient information (Booking Handler + Patient Service) is needed to verify whether the patient can book the vaccine. Lastly, the Booking Handler provides the main functionality of registering the booking slot in the database.

Performance Efficiency: Should be connected to the number of resources used by the system to complete tasks under specified conditions. Performance is important to avoid unnecessary delays or waste of resources when the user is trying to book vaccines. Some examples are having enough space (resource condition) to store and handle all booking requests (task), avoiding high latency in the recommendations (task) due to high workload (resource condition), having enough instances or resources to handle many bookings (task) at the same time (resource condition). Any component can be used as an example for this one, granted that the justification is based on the resources used.

b. [10 pts] Choose **two** metrics to assess the external quality of our system. For each chosen metric you should: (i) indicate one or more components to collect that metric, (ii) the quality characteristic/attribute associated with your chosen metric (it can be the same characteristic for both metrics), and (iii) the purpose of that metric for the chosen component(s).

There are many acceptable metrics. 5 pts per metric and must yield a number or a category, both quantitative and qualitative measures are acceptable. The key aspect is that: [1-2pt] (i) the choice of component is cohesive to the metric and quality attribute chosen, [2-3pts] (ii) the metric must be obtained from the behavior of the software (execution) and be connected to the correct quality characteristic/attribute, (iii) [2-3pt] the purpose must be correct in relation to the definition of the metric. Some examples below:

Chosen Metric: Probability of Failure on Demand (POFOD). (i) This can be measured on any of the services since our application sends requests to them, or the Broker, Booking Handler and Clinic Handler components can be used as examples. (ii) POFOD is associated with reliability. (iii) The purpose is to measure the proportion of requests that fail based on the total amount of requests made; both failures and crashes are counted as failed requests.

Chosen Metric: Rate of Occurrence of Failures (ROCOF). (i) Can be measured in the same components or services as the POFOD. (ii) ROCOF is related to reliability. (iii) ROCOF conveys the number of failed requests in relation to a time interval in which those requests were sent (hours, days, weeks, etc.)

Chosen Metric: Availability. (i) Can be measure on the same services or components as the ROCOF and POFOD. (ii) Availability is also related to reliability. (iii) Availability conveys the probability to access and complete tasks using the application / service. Availability is measured based on the mean uptime and downtime of the application.

Chosen Metric: Number of failures (unexpected outcomes). (i) Can be measure on components connected to the expected behavior of the software such as Booking Handler or Clinic Handler. (ii) Failures are connected to reliability or functional suitability. (iii) The purpose of failures is to convey the number of test cases where the actual output did not match an expected output when executing the system. Should indicate that a fault exists.

Chosen Metric: Latency and jittering. (i) Can be measure on components focused on exchange messages, so good candidates are the services and the Broker Component. (ii) Latency and jittering are connected to performance efficiency. (iii) The purpose of latency is to convey the time needed to process a request in the system (i.e., send the request and receive a response), whereas jittering is the variation around that time after performing multiple requests.

Other examples of measures are: Mean time between failures, throughout (if mentioned in terms of tasks, and not in terms of requests, otherwise that is latency), uptime, downtime, etc.

c. [5 pts] Vaccine-Co wants to promote sustainability in the product that we are creating. Using the Sustainability Awareness Framework, you must (i) **choose or suggest one functionality for our vaccine booking system** and explain the (ii) immediate, (iii) enabling and (iv) structural effects of those features. You must also indicate the **corresponding type/dimension of sustainability** in which those effects occur.

The key components are: [1-2pt] (i) choosing or suggesting a feature for the vaccine system and describing a [2-3pt] cohesive connection between the immediate, enabling and structural effects of the mentioned feature, and [2-3pts] the correct dimensions (environmental, technical, economical, social or individual) for each of those effects.

Suggested feature: Suggest slots from clinics at walking distance from the patient's residence. Immediate effect: Convenience to get to the clinic avoiding delays or missed vaccine slots (individual); individuals do not need to take public transport or vehicles to arrive at the clinics (environmental); structural effect: areas with more dense populations (urban areas) can have scarce availability of vaccines (social).

Suggested feature: Verifying if a patient is eligible for the vaccine. Immediate effect: Prioritise the patients that are in risk groups or that require quick access to the vaccine (individual), enabling effect: Faster immunisation of individuals that are more vulnerable to the virus (social), structural: Reducing the number of hospitalisation and costs in the healthcare (economical)

Suggested feature: Introducing accessibility features in the booking GUI (text-to-speech, high contrast interfaces, etc.). Immediate effect: Creation of different options of interactions between the user and the software system (technical), enabling effect: Reduces accessibility obstacles for users with disabilities to book vaccinations (individual); structural effect: Reduces discrimination in offering vaccines to all eligible individuals (social).

2. **[25 pts]** Considering the decision to introduce Continuous Integration to the development team, answer the following questions:

   a. [8 pts] Write **two tools** required to implement a Continuous Integration pipeline. For each chosen tool, you must explain the benefit that they bring to the development cycle in a CI pipeline.

CI relies on a few core tools to be implemented correctly. Some of those tools are [4pt per correct tool and benefit explanation]. Note that unit tests are not required to implement CI pipelines, but are very important, hence it is covered mainly in the practices question. Also, **careful** when distinguishing git (control version) versus GitHub (platform) and GitHub Workflow (CI automation framework). Also, the explanation of the benefit that the tool brings adds more value than naming specific tools.

Version control: The benefit of version control is being able to track and restore each version of the artefacts produced in the project (code, tests, requirements, design, etc.). This is one of the pillars of configuration management and, consequently, a pillar for continuous integration. Therefore, tools like git (e.g., GitHub, Gitlab, etc.) or SVN are required for CI.

Automation server: The benefit of an automation server is to run the integration in a server or machine that is independent and physically different than the environment used to develop the software. Therefore, we solve the "but it runs on my machine" problem. Some tools offer servers configured already with version control or other tools mentioned here, but they should still be separate mentions. Some examples of those tools are Jenkins, Gitlab, Travis CI.

Build tool/framework: The benefit of build tools is to configure and automatically execute the core tasks required to integrate different components of the software such as compiling the code, downloading or installing dependencies, etc. This reduces the risks with integrating components that have missing or broken dependencies and introducing faults in the built version of the system. Some examples of tools are Maven, Graddle, Ant, etc.

b. [7 pts] List **two development practices/activities** necessary to successfully use Continuous Integration. Justify why those practices are relevant for CI.

[3-4 pts] per activity and correct justification. Some examples of important activities are:

Commit changes often: It relevant to CI because it enables developers to quickly identify faults introduced by their changes and revert them. With less frequent changes, faults are often identified at a later stage in development which increases the costs for maintenance and bug fixing.

Automated and continuous testing: It is relevant to CI because if provides feedback on whether the different integrated parts work correctly, CI requires an automated test suite (e.g., unit or integration level) such that the introduced changes can be verified to avoid introducing faults that break the current version of the system.

Configuration management: Configuration management is relevant to CI because it brings control awareness and reproducibility for setup and deployment of the software system. This is beneficial because it allows the team to work simultaneously on different components while enabling easy integration of those different components, hence facilitating the release of the software.

c. [10 pts] Regression testing enables the identification of regression faults. However, one of the risks with regression testing in modern software development is the lack of time or resources to run all regression tests. Choose and explain **one regression testing technique** that can be used to reduce the costs with regression tests. Your explanation must include **how** the chosen technique makes regression testing more cost-effective.

There are three regression testing techniques (referred to as test optimisation techniques). The key aspect is to [4pt] provide a correct description of the technique and [6pt] explain the unique aspect in which the technique reduces costs with test execution.

Test suite minimisation: The goal is to identify subsets of test cases by removing unnecessaries redundancies between tests. The technique makes regression tests more cost-effective by discarding redundant test cases which, in turn, are tests cases that fulfil the same test requirement (i.e., a redundant test does not bring any new value to the test suite such as covering new parts of the code or covering a unique set of system requirements). If the test suite has no redundancies, the costs for executing the tests will be the same for the original and minimised test suite.

Test case selection: The goal is to identify a subset of test cases that meet a specific criterion. There are different types of criteria that can be used for selection such as coverage of modifications, test diversity, requirements or simply a plain cut off in the number of test cases. The choice of criteria is context dependent and can vary depending on the software product or the domain. The technique makes regression testing more cost-effective because tests that do not meet the criteria can be skipped.

Test case prioritisation: The goal is to assign an order/priority to each test case so that they can execute in a specific order. The priority is assigned according to a criterion so that fault-revealing test cases execute first. The prioritised set of tests has the same size as the original set of tests; therefore, all test cases are, in theory, executed (but that execution can be interrupted in case we do not have execution resources). Test prioritisation makes regression testing more cost-effective by enabling failures to be revealed earlier during test execution such that debugging of those failures can begin earlier while the prioritised test suite executes.

Another acceptable answer would be combinatorial interaction testing (CIT) used to test systems that have many parameters that need to be tested (e.g., different choice of browsers, operating systems, options of programming languages, etc.). In those cases, testing all possible combinations is too costly, hence CIT reduces the costs of testing by providing a list of all t-way interactions between those parameters. Therefore, CIT reduces the number of test executions while guaranteeing that a subset of important interactions will be exercised by the tests. Other techniques will be evaluated based on the answer.

3. **[20 pts]** Considering the context for Vaccine-Co and your knowledge of levels of testing and types of testing techniques, answer the questions below:

   a. [7 pts] Explain **one advantage and one disadvantage** of white-box testing techniques. Choose **one component or service** from Figure 1 where a white-box technique can be used to create test cases and justify your choice.

Each advantage or disadvantage scores 3-4 pts, split between stating a correct advantage/disadvantage [2-3 pts] and corresponding component [1-2pts]. Note that stating "access to code" is an advantage is incomplete, as the goal is explaining why that is beneficial. The benefit should be correct and specific to white-box techniques (not just general benefits with testing)

As far as choosing components, it would be incorrect to choose any service or the database as the context states that there is no access to the code of the services, only the specification. Other components are suitable choices, particularly the Booking Handler and the Clinic Handler components.

Some examples of advantages and disadvantages below:
Advantages:

- Having access to code can reveal specific lines of code or code constructs (control-flow, methods) to cover with tests.
- Knowledge of code coverage helps designing specific tests (or enables selection of tests) covering specific parts of the code such as parts impacted by changes.
- There are a lot of available and white-box testing tools used in practice and in research, which offers a variety of tools to improve the test process.
- White-box techniques are easier to automate (execution or generation) due to the access and knowledge of the source code of the system under test.
- The access to code can facilitate identifying the faulty line of code based on failed tests.
- The access to code makes white-box testing technique suitable for designing tests for isolated units in the code (unit testing).

Disadvantages:

- White-box techniques are often dependent on the programming language of the system under test.
- White-box techniques rely on code instrumentation which is hard to achieve and can lead to slow compilation and testing cycles.
- White-box techniques can be affected by low-quality code. For instance, complex code can hinder the design of the test cases.
- White-box techniques are sensitive to changes in the code such as refactoring that, in turn, should not affect the behavior of the software.

b. [8 pts] Explain **one advantage and one disadvantage** of black-box testing techniques. Choose **one component or service** from Figure 1 where a black-box technique can be used to create test cases and justify your choice.

Each advantage or disadvantage scores 3-4 pts, split between stating a correct advantage/disadvantage [2-3 pts] and corresponding component [1-2pts]. Note that stating "closer to the specification" as an advantage is incomplete, as the goal is explaining why that is beneficial. The benefit should be correct and specific to black-box techniques (not just general benefits with testing).

As far as choosing components/services, any choice would be applicable as all components and services have specification. The services are the most suitable choice as white-box techniques are not applicable to them, hence, black-box techniques are the only option to test them in our context.

Some examples of advantages and disadvantages below:
Advantages:

- Designing black-box tests do not require knowledge of a specific programming language, therefore, specific technical expertise is not a requirement
- Black-box techniques are applicable across projects that are implemented in different programming languages.
- The quality of the source code (e.g., code smells or high complexity) should not affect the design of the test cases.
- Since there are no dependencies to the implementation, black-box testing can be done in all levels of testing (unit, integration, and system).

Disadvantages:

- Black-box tests alone do not offer insight on how many statements or branching statements we are covering in the code.
- Designing black-box tests is harder if the specification is unclear, not written in a document, or the type of input is complex (e.g., the input are XML files for a parser).
- Automatic generation of tests for black-box test is not as mature as generation for white-box testing, so tool support is limited.
- Black-box tests can be very redundant when a set of input values cover the same code statements. By redundant tests we mean different test cases that exercise the same software "behaviour".

c. [5 pts] Choose **one component or service** from Figure 1 that would be a good candidate for doing integration-level testing. Justify your choice of component.

The two most suitable choices are:

- The Broker Component because its only purpose is to handle the exchange of messages between the different components, hence it enables the integration of those different functionalities. In other words, failures in the Broker can lead to failures in the interaction between all other components.
- The Booking handler component and its connection to the database. The booking handler only functions properly if the booked slots are properly saved and retrieved from the database. In other words, failures between the booking handler and the database can cause vaccine slots to be lost or become inconsistent.

Careful not to mention the interaction between the services and the corresponding components (i.e., Authentication and Front-end; Clinic Handler and Clinic Service; Patient service and Booking Handler), but they could also be seen as system-level tests as the services are a black-box and there is no control over its interaction. If you chose the services, only 2 pts would apply, **unless** you provided a justification quite similar to the ones mentioned above for the broker and booking handler where a faulty interaction can lead to failures or crashes of the system.

4. **[25 pts]** The code below is a method used to concatenate words in an array. The method has two input values: (i) an array of words to be concatenated and (ii) a string to be used as separator between those words. The output is a single string where all elements of the array are concatenated and separated by the separator string. For invalid input values (null values or an empty array) the method returns an empty string as result. Using your knowledge of different testing techniques, answer the questions below:

```java
// Method to paste together words from an array
//  using a specific separator.
public static String concatenateWords(String[] words, String separator){

     //If the array is empty, or the method received
     //   null values, return empty string.
     if(words == null || separator == null || words.length == 0){
         return "";
     }

     String result = "";
     //Concatenate each word with the separator, except for the last.
     for (int i = 0; i < words.length - 1; i = i + 1) {
       // '+' concatenates strings in Java.
          result = result + words[i] + separator;
      }

      //Note that the separator is not added when
      //  concatenating the last word.
      return result + words[words.length - 1];

}
```

a. [7 pts] Using Property-based testing (PBT) write down **at least three properties** for verifying the method `concatenateWords`. You should write the properties using natural language.

Be mindful that many of the typical property patterns (commutative, reverse, etc.) are not applicable to this function because the output (string) is different than the input (string array and a string). Your property must cover both inputs as both would be generated by the PBT tool.

Some examples of properties are listed below [2-3pts each]:

- For any array of words and separator equals to null, the result must be an empty string.
- For any separator and an empty or null array of words, the results must be an empty string.
- For all arrays of size one, the result will be equal to the content element of the array.
    - `For w.length == 1; concatenate(w, sep) == w[0]`

- For any non-empty and non-null array of words and separator (w,sep), the length of the resulting string after using the method twice (i.e., concatenating the results) is equal to two times the length of the resulting string after using method once
  - `concatenate(w, sep).length + concatenate(w, sep).length`
    `==`
    `2 x concatenate(w, sep).length`

- For any pairs of non-empty and non-null arrays of words (w1, w2) and separators (sep1, sep2), the length of the resulting string from using the method on those two inputs is equal to the length of the resulting string of using the method in the opposite order
  - `concatenate(w1, sep1).length + concatenate(w2, sep2).length`
    `==`
    `concatenate(w2, sep2).length  + concatenate(w1, sep1).length`

- For any non-empty and non-null array of words and separator, the length of the resulting string must be equal to the sum of the length of each word and (N-1) times the length of the separator (where N is the number of words).

b. [8 pts] Write **at least three equivalence partition** classes that can be used to test the method below. Use the method under test (specification or code) to justify your choice of partition classes.

There are different partitions that can be written, and each partition should score [2-3 pts]. The partitions must be written in terms of the input and the inputs from the same partition should all result in the same behaviour of the software. Some examples:

- Partition 1 (all are different ways to write the indicate the partition). If you only specify this one, you are just specifying one partition, so repetitions do not count as new partitions.
  - (Null arrays; Non-null separator)
  - (Empty arrays; Non-null separator)
  - (Arrays with at least one element; Null separator)
  - (Null arrays; Null separator)
  - (Empty arrays; Null separator)
  - All of the above produce the same outcome which is an empty string.
- Partition 2: (Arrays with only one element; non-empty separator)
  - Note that, in the result, the loop is not exercised, and the separator is not used, so the result is equal to the input
- Partition 3: (Arrays with more than one element; non-empty separator)
  - The outcome of this partition is the behaviour of concatenating the words with the separator.

c. [5 pts] Using Boundary Value Analysis (BVA), write **at least four test cases** (input and expected output) that exercise the boundaries of the partition classes you identified above.

There are many cases depending on the correct partitions. Each boundary must have two test inputs (counting two test cases, so only two boundaries are really required for the question). The test must contain the input values (for both input parameters) and the expected output. Some examples:

- Boundary 1:
  - `Input: words = [], sep = ""; Exp. Out: ""`
  - `Input: words = ["hello"], sep = ""; Exp. Out: "hello"`
- Boundary 2:
  - `Input: words = [], sep = ""; Expected output: ""`
  - `Input: words = ["hello", "there"], sep = " "; Expected output = "hello there"`
- Boundary 3:
  - `Input: words = ["hello"], sep = null; Exp. Out: ""`
  - `Input: words = ["hello"], sep = ""; Exp. Out = "hello"`
- Boundary 4:
  - `Input: words = ["hello"], sep = "-"; Exp. Out: "hello"`
  - `Input: words = ["hello", "there"], sep = "-"; Exp. Out = "hello-there"`
- Boundary 5:
  - `Input: words = null, sep = ""; Exp. Out: ""`
  - `Input: words = ["hello"], sep = ""; Exp. Out: "hello"`
- Boundary 6:
  - `Input: words = null, sep = "-"; Exp. Out: ""`
  - `Input: words = ["hello", "there"], sep = "-"; Exp. Out: "hello-there"`

d. [5 pts] Choose **one mutation operator** and create one mutant of the `concatenateWords` method. You must indicate the line of code(s) where the mutation is done.

There are many valid mutants for the code above. The main thing to be careful is not creating equivalent mutants or mutants that cause compilation problems (some listed below). Below, see some examples for different mutation operators:

AOR: Arithmetic Operator Replacement. Only valid for line 9 or line 16

- `Line 9. i = i +1 → i = i - 1; (or other operator)`
- `Line 9. words.length - 1→ words.length + 1`
- `Line 16. words.length - 1→ words.length + 1`

- **Important:** Changing Line 10 is NOT a mutant because the '+' operator in Java concatenates strings and changing it to any other operator would cause a compilation error.

ROR - Relational Operator Replacement. Only valid for lines 3 or 9

- `Line 3: words == null --> words != null (or other relational)`
- `Line 3: separator == null --> separator != null`
- `Line 3: words.length == 0 --> words.length != 0`
- `Line 9: i < words.length – 1 --> i` **`<=`** `words.length – 1`

COR - Conditional Operator Replacement. Only valid for line 3

- `Line 3: Change any of the conditions || to &&.`

AOR - Assignment Operator Replacement. Lines 9 or 11.

- `Line 11: result = result + words[i] + separator; -> result += ...`
- `Line 9: i = i +1 --> i` **`+=`** `i – 1; (or other operator)`

SVR - Scalar Variable Replacement.

- There are no mutants for this operator because the only two variables that can be swapped (same type) are results and separator and swapping them would cause compilation errors. The other variables are of type array (words) or integer (i)