Card

A card is a medium that describes a user story (e.g., it can be a physical card in the form of a piece of paper placed on a scrum board[5] or an entry in an electronic board). The card identifies the requirement, its criticality or priority, constraints, expected development and testing time, and—very importantly—acceptance criteria for the story. The description must be accurate, as it will be used in the product backlog. Agile teams can document user stories in a variety of ways. One popular format is:

As a (*intended user*)
I want (*intended action*),
so that (*purpose/result of the action, profit achieved*),

followed by acceptance criteria. These criteria can also be documented in different ways (see Sect. 4.5.2). Regardless of the approach taken to documenting user stories, the documentation should be both concise and sufficient for the team that will implement and test it.

Cards *represent* customer requirements rather than *document* them. While the card may contain the text of the story, the details are worked out in conversation and recorded in the confirmation [61].

Conversation

The conversation explains how the software will be used. The conversation can be documented or verbal. Testers, having a different point of view than developers and business representatives, make valuable contributions to the exchange of thoughts, opinions, and experiences. The conversation begins during the release planning phase and continues when the story is planned. It is conducted among stakeholders with three primary perspectives on the product: the customer/user, the developer, and the tester.

Confirmation

Confirmation comes in the form of acceptance criteria, which represent coverage items that convey and document the details of a user's story and which can be used to determine when a story is complete. Acceptance criteria are usually the result of a conversation. Acceptance criteria can be viewed as test conditions that the tester must check to verify story completeness.

User stories should address both functional and nonfunctional features. Typically, the tester's unique perspective will improve the user story by identifying missing details or nonfunctional requirements. The tester can provide input by asking business representatives open-ended questions about the user story, suggesting ways to test it, and confirming acceptance criteria.

Shared authorship of user stories can use techniques such as brainstorming or *mind mapping*.[6] Good user stories meet the so-called INVEST properties, that is, they are [62]:

[5] In Scrum, a scrum board is an optionally used physical board that shows the current status of an iteration. It provides a visual representation of the schedule of work to be done in a given iteration.
[6] For more on mind maps, see, for example, https://www.mindmapping.com/.

- **I**ndependent—they do not overlap and can be developed in any order.
- **N**egotiable—they are not explicit feature contracts; rather, the details will be co-created by the client and developer during development.
- **V**aluable—once implemented, they should bring added value to the customer.
- **E**stimable—the client should be able to prioritize them, and the team should be able to estimate their completion time to more easily manage the project and optimize the team's efforts.
- **S**mall—this makes their scope easy for the team to understand and the creation of the stories themselves more manageable.
- **T**estable—this is a general characteristic of a good requirement; the correctness of the story implementation should be easily verifiable.

If a customer doesn't know how to test something, it may indicate that the story isn't clear enough or that it doesn't reflect something of value to the customer or that the customer simply needs help with testing [62].

Example The following example shows a user story written from the perspective of a customer of an e-banking web application. Note that the acceptance criteria are not necessarily directly derived from the content of the story itself but can be the result of a conversation between the team and the customers. These criteria are used by the tester to design acceptance tests that will verify that the story has been fully and correctly implemented.

User Story US-001-03
As a customer of the bank
I want to be able to log into the system
So that I can use bank products
Acceptance criteria

- Login must be a valid email address.
- System rejects login attempt with incorrect password.
- System rejects login attempt for non-existent user login.
- System allows the user to enter in the fields "login" and "password" only alphanumeric characters, as well as a period and the symbol "@".
- Login and password fields can take up to 32 characters.
- After clicking on the link "remind password" and entering the user's email address, a link to the password reminder system is sent to this address.
- System starts the login process when the "login" button is clicked or when the Enter key is pressed; in the latter case, the login process starts when three conditions are met simultaneously: (1) the "login" field is non-empty, (2) the active window is the "password" window, (3) the "password" field is non-empty.

4.5.2 Acceptance Criteria

Acceptance criteria 🕮 are conditions that a product (to the extent described by the user story or Product Backlog Item of which these acceptance criteria are a part) must meet in order to be accepted by the customer. From this perspective, acceptance criteria can be viewed as test conditions or coverage items that should be checked by acceptance tests.

Acceptance criteria are used:

- To define the boundaries of the user's story
- To reach a consensus between the developer team and the client
- To describe both positive and negative test scenarios
- As the basis for user story acceptance testing (see Sect. 4.5.3)
- As a tool for accurate planning and estimating

Acceptance criteria can be considered as a great help to determine and evaluate the Definition of Ready (DoR) or Definition of Done (DoD). A team can decide not to start the implementation when the Acceptance Criteria of a User Story are not exhaustively elicited. Similarly, a team can decide a User Story is not considered a candidate for release (or for demo) when not all Acceptance Criteria are met (Acceptance Criteria coverage below 100%).

Acceptance criteria are discussed during the conversation (see Sect. 4.5.1) and defined in collaboration between business representatives, developers, and testers. Acceptance criteria—if met—are used to confirm that the user story has been implemented fully and in accordance with the shared vision of all stakeholders. They provide developers and testers with an expanded vision of the function that business representatives (or their proxies) will validate. Both positive and negative tests should be used to cover the criteria. During confirmation, different stakeholders play the role of the tester. These can range from developers to specialists focused on performance, security, interoperability, and other quality attributes. The agile team considers a task complete when a set of acceptance criteria is deemed to have been met.

There is no single, established way to write acceptance criteria for a user story. The two most common formats are:

- *Scenario-oriented* acceptance criteria
- *Rule-oriented* acceptance criteria

Scenario-Oriented Acceptance Criteria

This format for writing acceptance criteria usually uses the Given/When/Then format known from the BDD technique. However, in some cases, it is difficult to fit acceptance criteria into such a format, for example, when the target audience does not need the exact details of the test cases. In that case, a rule-oriented format can be used.

Rule-Oriented Acceptance Criteria

Typically in this format, acceptance criteria take the form of a bulleted verification list or a tabular form of mapping inputs to outputs. Some frameworks and languages for writing Given/When/Then style criteria (e.g., Gherkin) provide mechanisms that allow you to create a set of rules within a scenario, each of which will be tested separately (this is a form of *data-driven* testing).

Most acceptance criteria can be documented in one of the two formats mentioned above. However, the team can use any other nonstandard format, as long as the acceptance criteria are well-defined and unambiguous.

Next to the scenario-oriented and rule-oriented acceptance criteria mentioned above, less mature teams typically elicit and document acceptance criteria in a free format, as additions or sub-sections to a User Story. Whatever format a team decides to use, the bidirectional traceability between the user story and its acceptance criteria is vital.

Example The user story described in the previous section (logging into the e-banking system) has acceptance criteria described in the form of rules. Some of them can be detailed and represented in the form of mapping inputs to outputs by defining specific examples of test data. For example, in Gherkin, the first three points of the acceptance criteria list for this story:

- Login must be a valid email address.
- System rejects login attempt with incorrect password.
- System rejects login attempt for non-existent user login.

can be specified as follows:

```
Scenario outline: correct and incorrect logins and passwords

Given User enters <login> as login
 And User enters <password> as password
When User clicks "login" button
Then Login result is <result>.

Examples:

| login                      | password    | result   |
| edina.monsoon@gmail.com    | abFab       | OK       |
| patsy.stone@gmail          | martini     | NOT OK   |
| saffron@mail.abc.com       |             | NOT OK   |
| mods→london@gmail.com      | swinging    | NOT OK   |
|                            | somePasswd  | NOT OK   |
```

In the above example, there are references to variables (in brackets "<" and ">") in the description of the acceptance criteria presented in Given/When/Then format, and the specific test data is in the Examples section below. This test will be executed five times, each time with a different set of test data. The name of the section where the test data is placed suggests that the tests are in the form of *examples*, which show on specific data how the system is supposed to behave. For example, a login is supposed to be correct if the login and password meet the set conditions (login is a valid email address, login and password are non-empty and contain only alphanumeric characters). On the other hand, if the email address is invalid (line 2), the password is blank (line 3), non-alphanumeric characters are used (line 4), or the login is blank (line 5), the login shall not succeed.

Example Let us consider another example of a user story, this time for a CRM system for a certain bank. The story concerns the implementation of certain business rules related to offering credit cards to customers who meet certain requirements.

As a financial institution

I want to make sure that only customers with sufficient annual income get a credit card

So that credit cards are not offered to customers who will not be able to repay the debit on the card

Scenario: There are two types of cards: one with a debit limit of $2500, the other with a debit limit of $5000. The maximum credit card limit depends on the customer's earnings (rounded to the nearest $). The customer must have a salary of more than $10,000 per month to get the lower debit limit. If the salary exceeds $15,000 per month, the customer gets a higher debit limit.

Given a customer with monthly earnings <earnings amount>.

When a customer applies for a credit card

Then the application should be <accepted> or <rejected> and if it is accepted, the maximum credit card limit should be <maximum limit>.

Examples of test cases written based on this story are shown in Table 4.14. Note that in this example, the tester, while creating test cases to check the rule described in the story, simultaneously tried to cover the boundary values of the "monthly salary" domain.

4.5.3 Acceptance Test-Driven Development (ATDD)

Acceptance Test-Driven Development (**ATDD**) 🕮 is a test-first approach (see Sect. 2.1.3). Test cases are created *before* the user story is implemented. Test cases are created by team members with different perspectives on the product, such as customers, developers, and testers [63]. Test cases can be manual or automated.

The first step is the so-called specification workshop, during which team members analyze, discuss, and write the user story and its acceptance criteria. During this process, all kinds of problems in the story, such as incompleteness, ambiguities,

Table 4.14 Business outcomes

TC	Earnings	Expected result	Maximum limit	Comments
1	$10,000	Rejected	$0	Income ≤ $1000
2	$10,001	Accepted	$2500	$10,000 < income ≤ $15,000
3	$15,000	Accepted	$2500	$10,000 < income ≤ $15,000
4	$15,001	Accepted	$5000	$15,000 < income

contradictions, or other kinds of defects, are fixed. The next step is to create tests. This can be done collectively by a team or individually by a tester. In either case, an independent person, such as a business representative, performs validation of the tests. Tests are *examples*, based on acceptance criteria, that describe specific characteristics of the user story (see Sect. 4.5.2). These examples help the team implement the user story correctly. Because examples and tests are the same thing, the terms are often used interchangeably.

Once the tests are designed, the test techniques described in Sects. 4.2, 4.3, and 4.4 can be applied. Typically, the first tests are positive tests, confirming correct behavior with no exceptions or error occurrences, involving a sequence of actions performed if everything goes as expected. These types of scenarios are said to implement so-called *happy paths*, i.e., execution paths where everything goes as planned without any failures occurring. After positive test executions, the team should perform negative tests and tests regarding the nonfunctional attributes (e.g., performance, usability). Tests should be expressed in terms that stakeholders can understand. Typically, tests include natural language sentences containing the necessary preconditions (if any), inputs, and associated outputs.

The examples (i.e., tests) must cover all features of the user story and should not go beyond it. However, acceptance criteria may detail some of the issues described in the user story. In addition, no two examples should describe the same features of the user story (i.e., the tests should not be redundant).

When tests are written in a format supported by the acceptance test automation framework, developers can automate these tests by writing support code during the implementation of the function described by the user story. The acceptance tests then become *executable requirements*. An example of such support code is shown in Sect. 2.1.3 when discussing the BDD approach.

Example Suppose a team intends to test the following user story.

User story US-002-02
As a logged-in bank customer
I want to be able to transfer money to another account
So that be able to transfer funds between accounts
Acceptance criteria

- (AC1) the amount of the transfer cannot exceed the account balance
- (AC2) target account must be correct
- (AC3) for valid data (amount, account numbers) the source account balance decreases and the destination account balance increases by the amount of the transfer
- (AC4) the transfer amount must be positive and represent the correct amount of money (i.e., be accurate to two decimal places max)

Keeping in mind that tests should verify that acceptance criteria are met, the examples of positive functional tests here could be the following tests relating to (AC3) and verifying also (AC4):

- TC1: make a "typical" transfer, e.g., a transfer for \$1500 from an account with a balance of \$2735.45 to another correct account; expected result: account balance = \$1235.45, the balance of the target account increases by \$1500
- TC2: make a correct transfer of the entire account balance to another correct account (e.g., a transfer for \$21.37 from an account with the balance of \$21.37 to another correct account); expected result: account balance = \$0, target account balance increases by \$21.37

The second test case uses boundary value analysis for the difference between the account balance and the transfer amount.

The next step is to create negative tests, keeping in mind that the tests are to verify that acceptance criteria are met. For example, a tester might consider the following situations:

- TC3: attempt to make a transfer to another, correct account, when the transfer amount is greater than the balance of the source account (verification of AC1)
- TC4: attempt to make a transfer with correct transfer amount and balance, but to a non-existent account (verification of AC2)
- TC5: attempt to make a transfer with the correct transfer amount and balance to the same account (verification of AC2)
- TC6: attempt to make a transfer for an incorrect amount (verification of AC4)

Of course, for each of these tests, the tester can define a series of test data to verify specific program behavior. In creating these test cases, the tester can use black-box techniques (e.g., equivalence partitioning or boundary value analysis). For example, for TC1, it is worth considering transfers for the following amounts: the minimum possible (e.g., \$0.01), "typical" (e.g., \$1500), amounts with different levels of accuracy (e.g., \$900.2, \$8321.06), or a very large amount (e.g., \$8,438,483,784).

For TC3, on the other hand, it is worth checking the following situations (here we use the equivalence partitioning and boundary value analysis):

- When the transfer amount is much larger than the account balance
- When the transfer amount is greater by 1 cent (the minimum possible increment) than the account balance

For TC4, an invalid account can be represented as:

- An empty character string
- A number with the correct structure (26-digit), but not corresponding to any physical account
- A number with incorrect structure—too short (e.g., 25-digit)
- A number with incorrect structure—too long (e.g., 27-digit)
- A number with invalid structure—containing forbidden characters, such as letters

Finally, for TC6, it is worth considering in particular:

- A negative number representing the correct amount (e.g., \$−150.25)
- Number 0
- A character string containing letters (e.g., \$15B.20)

- A number represented as the result of a mathematical operation (e.g., \$15+\$20)
- A number with more than two decimal places (e.g., \$0.009)

Note that each of the test data sets given above checks a significantly different potential risk existing in the system than the other cases. Thus, the tests are nonredundant; they do not check “the same thing.”

Sample Questions

Question 4.1

(FL-4.1.1, K2)

Test design, test implementation, and test execution based on software input domain analysis are an example of:

A. White-box test technique.
B. Black-box test technique.
C. Experience-based test technique.
D. Static testing technique.

Choose one answer.

Question 4.2

(FL-4.1.1, K2)

What is a common feature of techniques such as equivalence partitioning, state transition testing, or decision tables?

A. Using these techniques, test conditions are derived based on information about the internal structure of the software under test.
B. To provide test data for test cases designed based on these techniques, the tester should analyze the source code.
C. Coverage in these techniques is measured as the ratio of tested source code elements (e.g., statements) to all such elements identified in the code.
D. Test cases designed with these techniques can detect discrepancies between requirements and their actual implementation.

Choose one answer.

Question 4.3

(FL-4.2.1, K3)

The system assigns a discount for purchases depending on the amount of purchases expressed in \$, which is a positive number with two decimal places. Purchases up to the amount of \$99.99 do not entitle to a discount. Purchases from \$100 to \$299.99 entitle to a 5% discount. Purchases above \$299.99 entitle to 10% discount.

Indicate the minimum set of values (given in \$) to achieve 100% equivalence partitioning technique.

A. 0.01; 100.99; 500.
B. 99.99; 100; 299.99; 300.
C. 1; 99; 299.
D. 0; 5; 10.

Choose one answer.

Question 4.4
(FL-4.2.1, K3)

The coffee vending machine accepts 25c, 50c, and $1 coins. The coffee costs 75c. After inserting the first coin, the machine waits until the amount of coins inserted by the user is equal to or greater than 75c. When this happens, the coin slot is blocked, coffee is dispensed, and (if necessary) change is given. Assume that machine has always sufficient numbers and denominations of coins to give change. Consider the following test scenarios:

Scenario 1:
Insert coins in order: 25c, 25c, 25c.
Expected behavior: vending machine dispenses coffee and no change.

Scenario 2:
Insert coins in order: 25c, 50c.
Expected behavior: vending machine dispenses coffee and no change.

Scenario 3:
Insert $1 coin.
Expected behavior: vending machine dispenses coffee and 25c change.

Scenario 4:
Insert coins in order: 25c, 25c, 50c.
Expected behavior: vending machine dispenses coffee and 25c change.

You want to check whether the machine actually gives change when the user has inserted coins for more than 75c and whether it gives no change at all if the user has inserted exactly 75c.

Which of these scenarios represent the minimum set of test cases that achieves this goal?

A. Scenario 1, Scenario 2.
B. Scenario 3, Scenario 4.
C. Scenario 1, Scenario 3.
D. Scenario 1, Scenario 2, Scenario 3.

Choose one answer.

Question 4.5
(FL-4.2.1, K3)

You are testing a card management system that entitles you to discounts on purchases. There are four types of cards, regular, silver, gold, and diamond, and three possible discounts: 5%, 10%, and 15%. You use the equivalence partitioning technique to test whether the system works correctly for all types of cards and all possible discounts. You already have the following test cases prepared:

TC1: regular card, discount: 10%
TC2: silver card, discount: 15%
TC3: gold card, discount: 15%
TC4: silver card, discount: 10%

What is the LEAST number of test cases you have to ADDITIONALLY prepare to achieve 100% "each choice" coverage of both card types and discount types?

A. 1
B. 3
C. 2
D. 8

Choose one answer.

Question 4.6
(FL-4.2.2, K3)
The user defines the password by entering it in the text field and clicking the "Confirm" button. By default, the text field is blank at the beginning. The system considers the password format correct if the password has at least 6 characters and no more than 11 characters. You identified three equivalence partitions: password too short, password length OK, and password too long. The existing test cases represent a minimum set of test cases achieving 100% 2-value BVA coverage. The test manager decided that due to the criticality of the component under test, your tests should achieve 100% 3-value BVA coverage.

Passwords of what lengths should be ADDITIONALLY tested to achieve the required coverage?

A. 5, 12
B. 0, 5, 12
C. 4, 7, 10
D. 1, 4, 7, 10, 13

Choose one answer.

Question 4.7
(FL-4.2.2, K3)
Users of the car wash have electronic cards that record how many times they have used the car wash so far. The car wash offers a promotion: every tenth wash is free. You are testing the correctness of offering the promotion using a boundary value analysis technique.

What is the MINIMAL set of test cases that achieves 100% 2-value BVA coverage? The values in the answers indicate the number of the given (current) wash.

Table 4.15 Decision table for free fare allocation rules

	R1	R2	R3
Conditions			
Member of parliament?	YES	–	–
Disabled?	–	YES	–
Student?	–	–	YES
Action			
Free ride?	YES	YES	NO

A. 9, 10
B. 1, 9, 10
C. 1, 9, 10, 11
D. Required coverage cannot be achieved

Choose one answer.

Question 4.8
(FL-4.2.3, K3)

The business analyst prepared a minimized decision table (Table 4.15) to describe the business rules for granting free bus rides. However, the decision table is flawed.

Which of the following test cases, representing combinations of conditions, shows that the business rules in this decision table are CONTRADICTORY?

A. Member of parliament = YES, disabled = NO, student = YES.
B. Member of parliament = YES, disabled = YES, student = NO.
C. Member of parliament = NO, disabled = NO, student = YES.
D. Member of parliament = NO, disabled = NO, student = NO.

Choose one answer.

Question 4.9
(FL-4.2.3, K3)

You create a full decision table that has the following conditions and actions:

- Condition “age”—possible values: (1) up to 18; (2) 19–40; (3) 41 or more.
- Condition “place of residence”—possible values: (1) city; (2) village.
- Condition “monthly salary”—possible values: (1) up to $4000; (2) $4001 or more.
- Action “grant credit”—possible values: (1) YES; (2) NO.
- Action “offer credit insurance”—possible values: (1) YES; (2) NO.

How many columns will the full decision table have for this problem?

A. 6
B. 11
C. 12
D. 48

Choose one answer.

Table 4.16 Valid transitions of the system for the login process

Transition	State	Event	Next state
1	Initial	Login	Logging
2	Logging	LoginOK	Logged
3	Logging	LoginError	Initial
4	Logged	Logout	Initial

Question 4.10

(FL-4.2.4, K3)

Table 4.16 shows in its rows ALL valid transitions between states in the system for handling the login process. The system contains three states (Initial, Logging, Logged) and four possible events (Login, LoginOK, LoginError, Logout).

How many INVALID transitions are in this system?

A. 8
B. 0
C. 4
D. 12

Choose one answer.

Question 4.11

(FL-4.2.4, K3)

Figure 4.17 shows a state transition diagram for a certain system.

What is the MINIMAL number of test cases that will achieve 100% valid transitions coverage?

A. 2
B. 3
C. 6
D. 7

Choose one answer.

Question 4.12

(FL-4.3.1, K2)

After test execution, your test cases achieved 100% statement coverage. Which of the following statements describes the correct consequence of this fact?

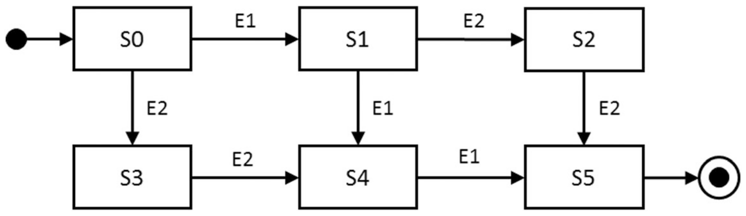
$$\begin{tikzcd}
\bullet \arrow[r] & S0 \arrow[r, "E1"] \arrow[d, "E2"] & S1 \arrow[r, "E2"] \arrow[d, "E1"] & S2 \arrow[d, "E2"] & \\
& S3 \arrow[r, "E2"] & S4 \arrow[r, "E1"] & S5 \arrow[r] & \odot
\end{tikzcd}$$

Fig. 4.17 Transition diagram of a state machine

A. 100% branch coverage is achieved.
B. Every logical value of every decision in the code was exercised.
C. Every possible output that the program under test can return was enforced.
D. Each statement containing a defect was exercised.

Choose one answer.

Question 4.13
(FL-4.3.2, K2)

Consider a simple program of three statements (assume that statements 1 and 2 take nonnegative integers (0, 1, 2, etc.) and that these statements will always execute correctly):

```
1. Get the input value x
2. Get the input value y
3. Return x + y
```

How many test cases are needed to achieve 100% branch coverage in this code and why?

A. Two test cases are needed, because one should test the situation where the returned value is be 0, and the other should test the situation where the returned value is a positive number.
B. There is no need to run any tests, because branch coverage for this code is satisfied by definition, since the program consists of a sequential passage of three statements and there are no conditional branches to test.
C. One test case is needed with arbitrary input values x and y, because each test will result in the execution of the same path covering all branches.
D. It is impossible to achieve branch coverage with a finite number of test cases, because to do so, we would have to force all possible values of the sum of x+y in statement 3, which is infeasible.

Choose one answer.

Question 4.14
FL-4.3.3 (K2)

Which of the following BEST describes the benefit of using white-box test techniques?

A. The ability to detect defects when specifications are incomplete.
B. The ability of developers to perform tests, as these techniques require programming skills.
C. Making sure tests achieve 100% coverage of any black-box technique, as this is implied by achieving 100% code coverage.
D. Better residual risk level control in code, as it is directly related to measures such as statement coverage and branch coverage.

Choose one answer.

Question 4.15
(FL-4.4.1, K2)
The tester uses the following document to design the test cases:

1. Occurrence of an arithmetic error caused by dividing by zero.
2. Occurrence of a rounding error.
3. Forcing a negative value result.

What technique does the tester use?

A. Boundary value analysis.
B. Checklist-based testing.
C. Use case based testing.
D. Error guessing

Choose one answer.

Question 4.16
(FL-4.4.2, K2)
Which of the following is the correct sentence regarding the use of formal (i.e., black-box and white-box) test techniques as part of a session-based exploratory testing?

A. All formal test techniques are allowed, because exploratory testing does not impose any specific method of operation.
B. All formal test techniques are forbidden, because exploratory testing is based on tester's knowledge, skills, intuition, and experience.
C. All formal test techniques are forbidden, because the steps performed in exploratory testing are not planned in advance.
D. All formal test techniques are allowed, because an exploratory tester needs a test basis from which to derive test cases.

Choose one answer.

Question 4.17
(FL-4.4.3, K2)
What benefit can be achieved by using checklist-based testing?

A. Appreciation of nonfunctional testing, which is often undervalued
B. Enabling accurate code coverage measurement.
C. Leveraging the tester's expertise.
D. Improved test consistency.

Choose one answer.

Question 4.18
(FL-4.5.1, K2)
During an iteration planning meeting, the team shares thoughts with each other about the user story. The product owner wants the customer to have a single-screen form for entering information. The developer explains that this feature has some

technical limitations, related to the amount of information that can be captured on the screen.

Which of the following BEST represents the continuation of writing this user story?

A. The tester decides that the form must fit on the screen and describes this as one of the acceptance criteria for the story, since the tester will be the one to conduct acceptance tests later.
B. The tester listens to the points of view of the product owner and the developer and creates two acceptance tests for each of the two proposed solutions.
C. The tester advises the developer that the performance acceptance criteria should be based on a standard: a maximum of 1 second per data record. The developer describes this as one of the acceptance criteria, since the developer is responsible for the application's performance.
D. The tester negotiates with the developer and the product owner the amount of necessary input information, and together they decide to reduce this information to the most important to fit on the screen.

Choose one answer.

Question 4.19
(FL-4.5.2, K2)

Consider the following user story:

As a potential customer of the e-shop,
I want to be able to register by filling out the registration form,
so that I am able to use the full functionality of the e-shop.

Which TWO of the following are examples of testable acceptance criteria for this story?

A. The process of carrying out registration must take place quickly enough.
B. After registration, the user has access to the "home ordering" function.
C. The system refuses registration if the user enters as a login an e-mail already existing in the database.
D. The system operator can send the order placed by the user for processing.
E. Acceptance criteria for this user story should be in Given/When/Then format.

Select TWO answers.

Question 4.20
(FL-4.5.3, K3)

The business rule for borrowing books in the university library system states that a reader can borrow new books if the following two conditions are met together:

- The time of the longest held book does not exceed 30 days.
- After borrowing, the total number of borrowed books will not exceed five books for a student and ten books for a professor.

The team is to implement a user story for this requirement. The story is created using the ATDD framework, and the following acceptance criteria are written as examples in the Given/When/Then format:

```
Given <UserType> has already borrowed <Number> of books
And the time of the longest held book is <Days> days
When user wants to borrow <Request> new books
Then the system <Decision> to loan the books

Examples:

No | UserType    | Number  | Days  | Request  | Decision
1  | Student     |  3      | 30    | 2        | does not allow
2  | Student     | 4       | 1     | 1        | allows
3  | Professor   | 6       | 32    | 3        | does not allow
4  | Professor   | 0       | 0     | 6        | allows
```

How many of the above four test cases are INCORRECTLY defined, i.e., they violate the business rules of book lending?

A. None—they all follow the business rule.
B. One.
C. Two.
D. Three.

Exercises

Exercise 4.1

(FL-4.2.1, K3)

The user fills out a web form to purchase concert tickets. Tickets are available for three concerts: Iron Maiden, Judas Priest, and Black Sabbath. For each of the concerts, one of two types of tickets can be purchased: a sector in front of the stage and a sector away from the stage. The user confirms the choice by checking the boxes on the corresponding drop-down lists (one of which contains the names of the bands, the other—the type of ticket). Only one ticket for a single band can be purchased per session.

You want to check the correctness of the system for each band and, independently, for the type of ticket.

A) Identify the domains and their equivalence partitions.
B) Are there any invalid partitions? Justify your answer.
C) Design the smallest possible set of test cases that will achieve 100% equivalence partitioning coverage.

Exercise 4.2

(FL-4.2.1, K3)

Table 4.17 Discount rules

Amount	Discount granted
Up to $300	No
Over $300, up to $800	5%
Over $800	10%

A natural number greater than 1 is called prime if it is divisible by only two numbers: by 1 and by itself. The system takes a natural number (entered by the user in the form field) as input and returns whether it is prime or not. The form field for the input data has a validation mechanism and will not allow any string to be entered that does not represent a valid input (i.e., a natural number greater than one).

A) Identify the domain and its equivalence partitioning.
B) Are there any invalid partitions in the problem? Justify your answer.
C) Design the smallest possible set of test cases that will achieve 100% equivalence partitioning coverage.

Exercise 4.3
(FL-4.2.2, K3)

You are testing the payment functionality of the e-shop. The system receives a positive amount of purchases (in $ with an accuracy of 1 cent). This amount is then *rounded up* to the nearest integer, and based on this *rounded* value, a discount is calculated according to the rules described in Table 4.17:

You want to apply a 2-value BVA to check the correctness of the discount calculation. The input data in the test case is the amount *before rounding*. Conduct equivalence partitioning, determine the boundary values, and design the test cases.

Exercise 4.4
(FL-4.2.2, K3)

The system calculates the price of picture framing based on the given parameters: width and height of the picture (in centimeters). The correct width of the picture is between 30 and 100 cm inclusive. The correct height of the picture is between 30 and 60 cm inclusive.

The system takes both input values through an interface that accepts only valid values from the ranges specified above.

The system calculates the area of the image as the product of width and height. If the surface area exceeds 1600 cm^2, the framing price is $500. Otherwise, the framing price is $450.

Apply 2-value BVA to the above problem:

A) Identify the partitions and their boundary values.
B) Design the smallest possible number of test cases covering the boundary values for all relevant parameters. Is it possible to achieve full 2-value BVA coverage? Justify your answer.

Exercise 4.5
(FL-4.2.3, K3)

The operator of the driver's license test support system enters the following information into the system, for a candidate who is taking the exams for the first time:

- The number of points from the theoretical exam (integer number from 0 to 100).
- The number of errors made by the candidate during the practical exam (integer number 0 or greater).

The candidate must take both exams. A candidate is granted a driver's license if they meet the following two conditions: they scored at least 85 points on the theoretical test and made no more than two errors on the practical test. If a candidate fails one of the exams, they must repeat this exam. In addition, if the candidate fails both exams, they are required to take additional hours of driving lessons.

Use decision table testing to conduct the process of designing test cases for the above problem. Follow the five-step procedure described in Sect. 4.2.3, that is:

A) Identify all the possible conditions, and list them at the top part of the table.
B) Identify all the possible actions that can occur in the system, and list them at the bottom part of the table.
C) Generate all combinations of conditions. Eliminate infeasible combinations (if there are any), and list all remaining, feasible combinations in individual columns at the top part of the array.
D) For each combination of conditions identified in this way, determine, based on the specification, what actions should take place in the system, and enter them in the appropriate columns at the bottom part of the table.
E) For each column, design a test case that includes a name (describing what the test case tests), pre-conditions, input data, expected output, and post-conditions.

Exercise 4.6
(FL-4.2.3, K3)
Figure 4.18 (after Graham et al., *Foundations of Software Testing*) shows the process of allocating seats to passengers based on whether they have a gold card and whether there are seats available in business and economy classes, respectively.

Describe this process with a decision table. Did you notice any problem with the specification while creating the table? If so, suggest a solution to the problem.

Exercise 4.7
(FL-4.2.4, K3)
The ATM is initially in a waiting state (welcome screen). After the user inserts the card, card validation takes place. If the card is invalid, the system returns it and terminates with the message "Card error." Otherwise, the system asks the user to enter a PIN. If the user provides a valid PIN, the system switches to the "Logged" state and terminates the operation. If the user enters a wrong PIN, the system asks to enter it again. If the user enters the wrong PIN three times, the card is blocked, the user receives the message "Card blocked," and the system goes to the final state representing the card being blocked.

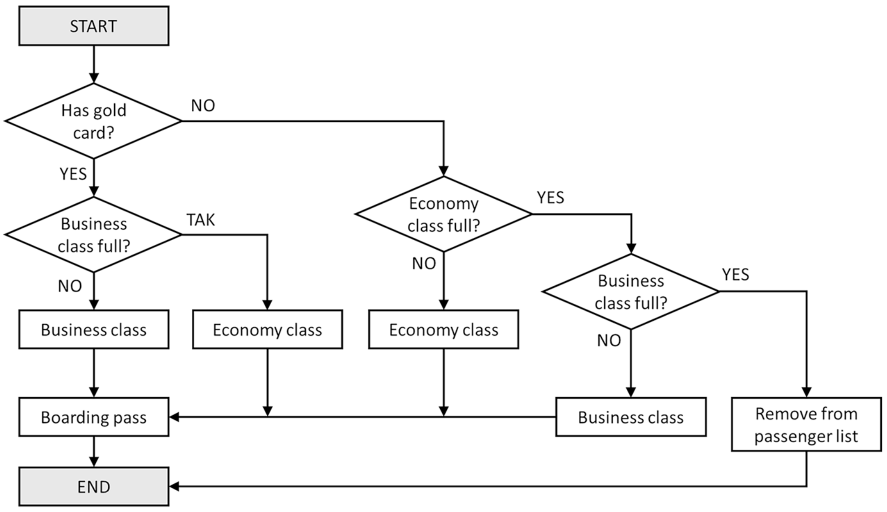

Fig. 4.18 The process of allocating seats on an airplane

A) Identify possible states, events, and actions, and design a state transition diagram for the above scenario *without* using guard conditions.
B) Design a state transition diagram for the same scenario but using guard conditions. You should get a model with fewer states.
C) For the state transition diagram from (A), design the smallest possible number of test cases that achieve:

   a) 100% all states coverage.
   b) 100% valid transitions coverage.

Exercise 4.8
(FL-4.2.4, K3)
The operation of the mechanical robot dog is described by the state transition diagram shown in Fig. 4.19. The initial state is "S."

A) How many *invalid* transitions are in this diagram?
B) Design test cases that achieve full all transitions coverage. Adopt the rule that when testing invalid transitions, one test case tests only one invalid transition.

Exercise 4.9
(FL-4.5.3, K3)
As a tester, you start working on acceptance test design for the following user story:

US-01-002 New user registration **Effort estimation: 3**
As: any user—potential customer
I want to: be able to fill out the web registration form
So that: I am able to use the service provider's offerings
Acceptance criteria:

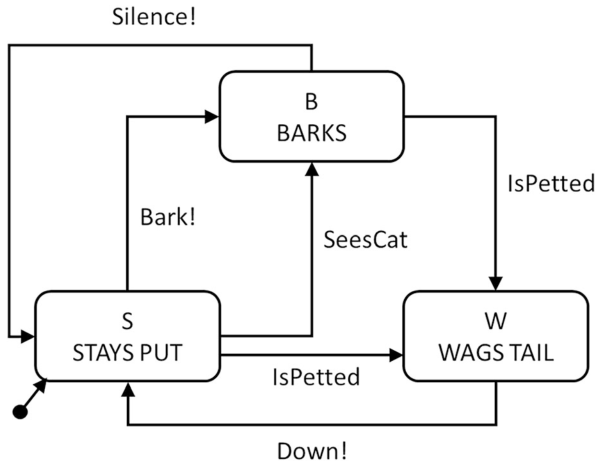

Fig. 4.19 Robot dog state transition diagram

AC1 The user's login is a valid e-mail address

AC2 User cannot register using an existing login

AC3 For security reasons, the user must enter the password in two independent fields, and these passwords must be identical; the system does not allow the user to paste a string into these fields; it must be entered manually

AC4 Password must contain at least 6 and at most 12 characters, at least 1 number and at least 1 capital letter

AC5 Successful registration results in sending an email containing a link, the clicking of which confirms registration and activates the user's account

Propose some sample acceptance tests with specific test data and expected system behavior.

Chapter 5 Managing the Test Activities

Keywords

Defect management	The process of recognizing, recording, classifying, investigating, resolving, and disposing of defects
Defect report	Documentation of the occurrence, nature, and status of a defect. *Synonyms:* bug report
Entry criteria	The set of conditions for officially starting a defined task. *References:* Gilb and Graham
Exit criteria	The set of conditions for officially completing a defined task. *After* Gilb and Graham. *Synonyms:* test completion criteria, completion criteria
Product risk	A risk impacting the quality of a product
Project risk	A risk that impacts project success
Risk	A factor that could result in future negative consequences
Risk analysis	The overall process of risk identification and risk assessment
Risk assessment	The process to examine identified risks and determine the risk level
Risk control	The overall process of risk mitigation and risk monitoring
Risk identification	The process of finding, recognizing, and describing risks. *References:* ISO 31000
Risk level	The measure of a risk defined by impact and likelihood. *Synonyms:* risk exposure
Risk management	The process for handling risks. *After* ISO 24765
Risk mitigation	The process through which decisions are reached and protective measures are implemented for reducing or maintaining risks to specified levels

L. Stapp et al., *ISTQB® Certified Tester Foundation Level*,
https://doi.org/10.1007/978-3-031-42767-1_8

Risk monitoring	The activity that checks and reports the status of known risks to stakeholders
Risk-based testing	Testing in which the management, selection, prioritization, and use of testing activities and resources are based on corresponding risk types and risk levels. *After* ISO 29119-1
Test approach	The manner of implementing testing tasks
Test completion report	A type of test report produced at completion milestones that provides an evaluation of the corresponding test items against exit criteria. *Synonyms:* test summary report
Test control	The activity that develops and applies corrective actions to get a test project on track when it deviates from what was planned
Test monitoring	The activity that checks the status of testing activities, identifies any variances from planned or expected, and reports status to stakeholders
Test plan	Documentation describing the test objectives to be achieved and the means and the schedule for achieving them, organized to coordinate testing activities. *References:* ISO 29119-1
Test planning	The activity of establishing or updating a test plan
Test progress report	A type of periodic test report that includes the progress of test activities against a baseline, risks, and alternatives requiring a decision. *Synonyms:* test status report
Test pyramid	A graphical model representing the relationship of the amount of testing per level, with more at the bottom than at the top
Testing quadrants	A classification model of test types/test levels in four quadrants, relating them to two dimensions of test objectives: supporting the product team versus critiquing the product and technology facing versus business facing

5.1 Test Planning

FL-5.1.1 (K2)	Exemplify the purpose and content of a test plan
FL-5.1.2 (K1)	Recognize how a tester adds value to iteration and release planning
FL-5.1.3 (K2)	Compare and contrast entry criteria and exit criteria
FL-5.1.4 (K3)	Use estimation techniques to calculate the required test effort

FL-5.1.5 (K3)	Apply test case prioritization
FL-5.1.6 (K1)	Recall the concepts of the test pyramid
FL-5.1.7 (K2)	Summarize the testing quadrants and their relationships with test levels and test types

5.1.1 Purpose and Content of a Test Plan

Test plan is a document that provides a detailed description of the test project's objectives, the means necessary to achieve those objectives, and a schedule of test activities. In typical projects, a single test plan, sometimes called a master test plan or project test plan, is usually created. However, in larger projects, you may encounter several plans, such as a master test plan and plans corresponding to the test levels defined in the project (level test plan or phase test plan). In such a situation, there may be, for example, a component integration test plan, a system test plan, an acceptance test plan, etc. Detailed test plans may also relate to the types of tests that are planned to be carried out in the project (e.g., a performance test plan).

The test plan describes the **test approach** and helps the team make sure that test activities can be started and, when completed, that they have been conducted properly. This is accomplished by defining specific entry and exit criteria (see Sect. 5.1.3) for each test activity or activity in the test plan. The test plan also confirms that, if followed, testing will be conducted in accordance with the project's testing strategy and the organization's testing policy.

Very rarely will the project turn out 100% as we planned it. In most situations, minor or major modifications will be necessary. The natural question then arises: why waste time planning in such a case? Dwight Eisenhower, US Army General and later President of the United States, once famously said, "In preparing for battle, I have always realized that plans are useless, but planning is indispensable." [1] Indeed, the most valuable part of creating a test plan is the **test planning** process itself. Planning in a sense "forces" testers to focus their thinking on future challenges and risks related to schedule, resources, people, tools, cost, effort, etc.

During planning, testers can identify risks and think through the test approach that will be most effective. Without planning, testers would be unprepared for the many different undesirable events that will happen during the project. This, in turn, would create a very high risk of project failure, i.e., not completing the project or completing it late, over budget, or with a reduced scope of work completed.

[1] https://pl.wikiquote.org/wiki/Dwight_Eisenhower

The planning process is influenced by many factors that testers should consider in order to best plan all the activities of the testing process. Among these factors, the following should be considered in particular:

- Test policy and test strategy
- SDLC
- Scope of testing
- Objectives
- Risks
- Limitations
- Criticality
- Testability
- Resource availability

A typical test plan includes the following information:

Context of testing. Scope, objectives, limitations, and test basis information

Assumptions and limitations of the test project

Stakeholders. Roles, responsibilities, influence on testing process (e.g., power vs. interest), and hiring and training needs of people

Communication. Types and frequency of communication and templates of documents used

List of risks. Product risks and project risks (see Sect. 5.2)

Approaches to testing. Test levels (see Sect. 2.2.1), test types (see Sect. 2.2.2), test techniques (see Chap. 4), test work products, entry and exit criteria (see Sect. 5.1.3), level of test independence (see Sect. 1.5.3), definition of test metrics used (see Sect. 5.3.1), test data requirements, test environment requirements, and deviations from organizational best practices (with justification)

Schedule (see Sect. 5.1.5)

As the project is being realized and test plan is being implemented, additional information appears, detailing the plan. Thus, test planning is an *ongoing* activity and is performed throughout the product lifecycle; sometimes, it includes a maintenance phase. It is important to realize that the initial test plan will be updated, as feedback from test activities will be used to identify changing risks and adjust plans accordingly.

The results of the planning process can be documented in a master test plan and in separate plans for specific test levels and test types.

The following steps are performed during test planning (see Fig. 5.1):

- Defining the scope and objectives of testing and the associated risks
- Determining the overall approach to testing
- Integrating and coordinating the test activities within the SDLC activities
- Deciding what to test, what personnel and other resources will be needed to perform the various test activities, and how the activities should be performed

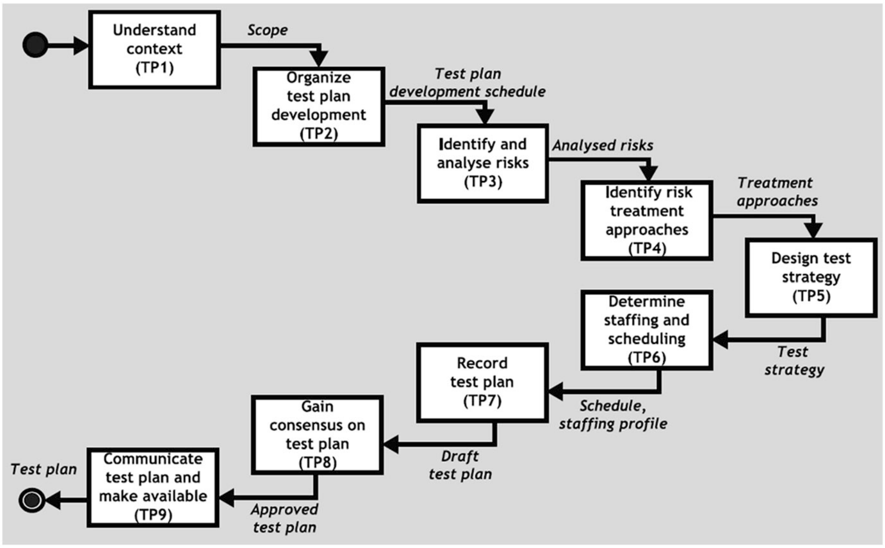
Understand context (TP1)
Scope
Organize test plan development (TP2)
Test plan development schedule
Identify and analyse risks (TP3)
Analysed risks
Identify risk treatment approaches (TP4)
Treatment approaches
Design test strategy (TP5)
Test strategy
Determine staffing and scheduling (TP6)
Schedule, staffing profile
Record test plan (TP7)
Draft test plan
Gain consensus on test plan (TP8)
Approved test plan
Communicate test plan and make available (TP9)
Test plan

Fig. 5.1 Test planning process (according to ISO/IEC/IEEE 29119-2 standard)

- Planning the activities of test analysis, design, implementation, execution, and evaluation by setting specific deadlines (sequential approach) and placing these activities in the context of individual iterations (iterative approach)
- Making a selection of measures for test monitoring and test control
- Determining the budget for the collection and evaluation of metrics
- Determining the level of detail and structure of documentation (templates or sample documents)

One part of the test plan—the test approach—will in practice be the implementation of a specific test strategy in effect in the organization or project. The test strategy is a general description of the test process being implemented, usually at the product or organization level.

Testing strategies
Typical testing strategies are:

- **Analytical strategy**. This strategy is based on the analysis of a specific factor (e.g., requirements or risk). An example of an analytical approach is risk-based testing, in which the starting point for test design and prioritization is the risk level.
- **Model-based strategy**. With this strategy, tests are designed on the basis of a model of a specific required aspect of the product, for example, function, business process, internal structure, or non-functional characteristics (such

(continued)

as reliability). Models can be created based on business process models, state models, reliability growth models, etc.

- **Methodical strategy**. The basis of this strategy is the systematic application of a predetermined set of tests or test conditions, for example, a standard set of tests or a checklist containing typical or likely failure types. According to this approach, checklist-based testing, fault attacks, and testing based on quality characteristics, among other things, are performed.
- **Process-compliant** (or standard-compliant) **strategy**. This strategy involves creating test cases based on external rules and standards (derived, e.g., from industry standards), process documentation or rigorous identification and use of the test basis, as well as any process or standard imposed by the organization.
- **Directed** (or consultative) **strategy**. This strategy is primarily based on advice and guidance from stakeholders, subject matter experts, or technical experts—including those outside the test team or organization.
- **Regression-averse strategy**. This strategy—motivated by the desire to avoid regression of already existing functionality—envisions the reuse of legacy testware (especially test cases), the extensive automation of regression testing, and the use of standard test suites.
- **Reactive strategy**. With this strategy, testing is geared more toward reacting to events than following a predetermined plan (as with the strategies described above), and tests are designed and can be immediately executed based on knowledge gained from the results of previous tests. An example of this approach is exploratory testing, in which tests are executed and evaluated in response to the behavior of the software under test.

In practice, different strategies can and even should be combined. The basics of test strategy selection include:

- Risk of project failure:
  - Risks to the product
  - Danger to people, the environment, or the company caused by product failure
  - Lack of skills and experience of the people in the project
- Regulations (external and internal) on the development process
- Purpose of the testing venture
- Mission of the test team
- Type and specifics of the product

5.1.2 Tester's Contribution to Iteration and Release Planning

In iterative approaches to software development, there are two types of planning related to products: release planning and iteration planning. There exist other types of planning, on a higher, organizational, or strategical level (such as "big room" planning or "product increment" planning). These types of planning are not further discussed.

Release planning
Release planning looks ahead to the release of a product. Release planning defines and redefines the product backlog and may include refining larger user stories into a collection of smaller stories. Release planning provides the basis for a test approach and test plan covering all iterations of the project. During release planning, business representatives (in collaboration with the team) determine and prioritize user stories for a release. Based on these user stories, project and product risks are identified (see Sect. 5.2), and high-level effort estimation is performed (see Sect. 5.1.4).

Testers are involved in release planning and add value especially in the following activities:

- Defining testable user stories, including acceptance criteria
- Participating in project and product (quality) risk analysis
- Estimating testing effort related to user stories
- Defining the necessary test levels
- Planning the testing for the release

Iteration planning
Once release planning is complete, iteration planning begins for the first iteration. Iteration planning looks ahead to the end of a single iteration and addresses the iteration backlog. During iteration planning, the team selects user stories from the prioritized product backlog, refines (clarifies) them and slices them (when needed), develops them, performs risk analysis for the user stories, and estimates the work needed to implement each selected story (see Sect. 5.1.4). If a user story is unclear and the attempts to clarify it have failed, the team can refuse it and use the next user story based on priority. Business representatives must answer the team's questions about each story so that the team understands what it should implement and how to test each story.

The number of selected stories is based on the so-called team velocity[2] and the estimated size of the selected user stories, as well as technical constraints. Once the

[2]Team velocity is the empirically determined amount of work a team is able to perform during a single iteration. It is usually expressed in terms of so-called user story points. The size of each story is also estimated in these units, so the team knows how many user stories it can take into the iteration backlog—the sum of their complexity cannot exceed the team's speed. This reduces the risk that the team will not have time to complete all the work to be done in an iteration and that the team will not finish the work before the end of the iteration, causing so-called empty runs.

content of the iteration is finalized, the user stories are divided into tasks to be executed by the corresponding team members.

Testers are involved in iteration planning and add value especially in the following activities:

- Participating in detailed risk analysis of user stories
- Determining the testability of user stories
- Co-creating acceptance tests for user stories
- Splitting user stories into tasks (especially test tasks)
- Estimating the testing effort for all testing tasks
- Identifying functional and non-functional testing aspects of the system under test
- Supporting and participating in test automation at multiple test levels

5.1.3 Entry Criteria and Exit Criteria

Entry criteria

Entry criteria (more or less similar to the *Definition of Ready* in an agile approach) define the pre-conditions that must be met before a test activity can begin. If they are not met, testing can be more difficult and time-consuming, costly, and risky.

Entry criteria include the availability of resources or testware:

- Testable requirements, user stories, and/or models
- Test items that met the exit criteria applicable to earlier levels of testing, mainly in the waterfall approach
- Test environment
- Necessary test tools
- Test data and other necessary resources

as well as the initial quality level of a test object (e.g., all smoke tests pass). Entry criteria protect us from starting tasks for which we are not yet fully prepared.

Exit criteria

Exit criteria (more or less similar to the *Definition of Done* in an agile approach) define the conditions that must be met in order for the execution of a test level or set of tests to be considered completed. These criteria should be defined for each test level and test type. They may vary depending on the test objectives.

Typical exit criteria are:

- Completion of the execution of scheduled tests
- Achieving the right level of coverage (e.g., requirements, user stories, acceptance criteria, code)
- Not exceeding the agreed limit of unrepaired defects

- Obtaining a sufficiently low estimated defect density
- Achieving sufficiently high reliability rates

Note that sometimes test activities may be shortened due to:

- The use of the entire budget
- The passage of the scheduled time
- The pressure of bringing a product to the market

In such situations, project stakeholders and business owners should learn about and accept the risks of running the system without further testing. Since these situations often occur in practice, testers (especially the person in the role of test team leader) should provide information describing the current state of the system, highlighting the risks.

Exit criteria usually take the form of measures of thoroughness or completeness. They express the desired degree of completion of a given job. Exit criteria allow us to determine whether we have performed certain tasks as planned.

Example The team adopted the following exit criteria from the system testing:

- (EX1) achieved 100% coverage of requirements by test cases
- (EX2) achieved at least 75% statement coverage for each component tested
- (EX3) no open (unrepaired) defects with the highest level of severity
- (EX4) at most two open defects of medium or low severity

After analyzing the report at the end of the system testing phase, it became clear that:

- For each requirement, at least one test case was designed
- For two of the four modules, full statement coverage was achieved; for the third, 80%; and for the fourth, 70%
- One of the tests detected a medium-priority defect that is still not closed

This analysis shows that criteria (EX1), (EX3), and (EX4) are met, while criterion (EX2) is not met for one of the components. The team decided to analyze what parts of this component's code are not covered and added a test case covering an additional control flow path in this module, which increased the coverage for this module from 70 to 78%. At this point, the criterion (EX2) has been met. Since all exit criteria from the system testing phase are met at this point, the team formally considers this phase complete.

5.1.4 Estimation Techniques

Test effort is the amount of test-related work needed to achieve the test project objectives. Test effort is often a major or significant cost factor during software development, sometimes consuming more than 50% of the total development effort [64]. One of the most common questions team members hear from their managers is "how long will it take to do this task?" The problems described above make test estimation a very important activity from a test management perspective. Every tester should have the ability to estimate test effort, be able to use appropriate estimation techniques, understand the advantages and disadvantages of different approaches, and be aware of issues such as estimation accuracy (estimation error).

Test effort can be measured as labor intensity, a product measure (time * resources). For example, an effort of 12 person-days means that a given job to be done will be done in 12 days by one person, or in 6 days by two persons, or in 4 days by three persons, and so on. In general, it will be work done in x days by y persons, where x * y = 12.

Be careful, however, because in practice, product measures "do not scale." For example, if a team of three people does some work in 4 days, the labor intensity is 12 person-days. However, if a manager adds a new person to the team, hoping that four people will do the same work in 3 days, they may be disappointed—such an enlarged team can still do the job in 4 days or even longer. This is due to a number of factors, such as an increase in communication efforts (more people on the team) or the need for longer-serving team members to spend time apprenticing new member, which can cause delays. Project managers have a saying illustrating this phenomenon: "one woman will have a baby in nine months, but nine women will not have a baby in a month."

When making an effort estimate, often not all knowledge about the test object is available, so the accuracy of the estimate may be limited. It is important to explain to stakeholders that the estimate is based on many assumptions. The estimate can be refined and possibly adjusted later (e.g., when more data is available). A possible solution is to use an estimation range to provide an initial estimate (e.g., the effort will be 10 person-months with the standard deviation of 3 person-months, meaning that with high probability, the actual effort will be between 10 − 3 = 7 and 10 + 3 = 13 person-months).

Estimating for small tasks is usually more accurate than for large ones. Therefore, when estimating a complex task, you can use a decomposition technique called work breakdown structure (WBS). In this technique, the main (complex) task or activity to be estimated is hierarchically decomposed into smaller (simpler) subtasks. The goal is to break down the main task into easily estimable but executable components. The task should be broken down as precisely as possible based on current information, but only as much as necessary to understand and accurately estimate a single subtask. After estimating all subtasks at the lowest level, the top-level tasks are estimated (by summing up the estimates of the relevant subtasks) in a bottom-up manner. In the last step, the estimation of the main task is obtained.

An example use of the WBS method is shown in Fig. 5.2.

We want to estimate the effort of the entire test project. However, it is so large that it does not make sense to estimate the entire effort at once—as the result will be subject to a very large error. So we divide the test project into the main tasks: planning, defining the test environment, integration testing, and system testing. Let's assume that we are able to fairly accurately estimate the effort for the first two tasks (1 and 4 person-days, respectively). Integration testing, however, will require a lot of effort, so we hierarchically divide this task into smaller ones. We identify two

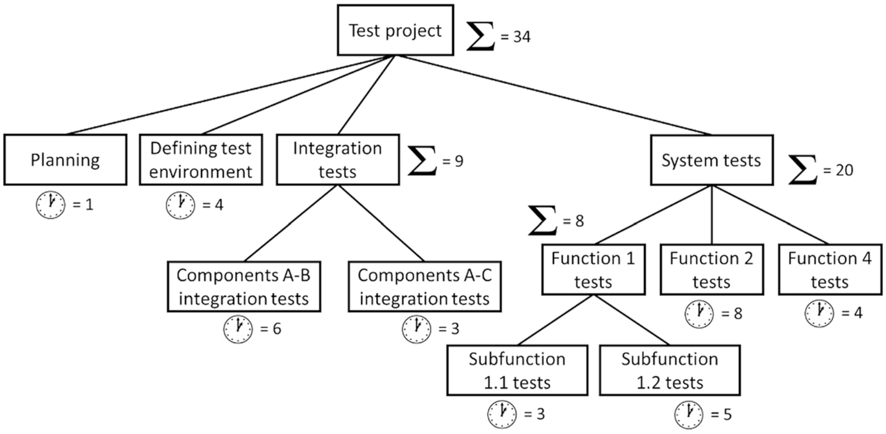

Fig. 5.2 Using the WBS method to estimate test project effort

subtasks within integration testing: integration of components A and B (6 person-days) and integration of components A and C (3 person-days). Under system testing, we identify three functions, F1, F2, and F3, as test objects. For F2 and F3, we estimate the testing effort to be 8 and 4 person-days, respectively. The F1 function is too large, so we split it into two subtasks and estimate the related effort as 3 and 5 person-days, respectively. Now we can collect the results from the lowest levels and estimate the tasks at higher levels by summing the corresponding subtasks. For example, the effort for integration testing (9 person-days) will be the sum of the effort of two subtasks with an effort of 6 and 3 person-days, respectively. Similarly, the effort for F1 is 3+5 = 8 person-days, and the effort for system testing is 8+8+4 = 20 person-days. At the very end, we calculate the effort for the entire test project as the sum of its subtasks: 1+4+9+20 = 34 person-days.

Estimating techniques can be divided into two main groups:

- Metrics-based techniques—where the test effort is based on metrics of previous similar projects, on historical data from the current project, or on typical values (so-called industry baselines)
- Expert-based techniques—where the test effort is based on the experience of test task owners or subject matter experts

The syllabus describes the following four frequently used estimation techniques in practice.

Estimation based on ratios

In this metrics-based technique, as much historical data as possible is collected from previous projects, allowing the derivation of "standard" ratios of various indicators for similar projects. An organization's own ratios are usually the best source to use in the estimation process. These standard ratios can then be used to estimate the testing effort for a new project. For example, if in a previous similar project the ratio of implementation effort to test effort was 3:2, and in the current project the development effort is expected to be 600 person-days, the test effort can be estimated as 400 person-days, since the same or similar ratio of implementation to test will most likely occur as in the similar previous project.

Extrapolation

In this metrics-based technique, measurements are taken as early as possible to collect real, historical data from the current project. With enough such observations (data points), the effort required for the remaining work can be approximated by extrapolating this data. This method is very useful in iterative software development techniques. For example, a team can extrapolate the testing effort in the fourth iteration as an average of the effort in the last three iterations. If the effort in the last three iterations was 30, 32, and 25 person-days, respectively, the extrapolation of effort for the fourth iteration would be (30+32+25) / 3 = 29 person-days.

Note that proceeding in an analogous manner, we could estimate the effort for subsequent iterations using the calculated extrapolation for previous iterations. In our example, the extrapolation of effort for the fifth iteration will be the average of iterations 2, 3, and 4, and so it will be (32+25+29) / 3 = 14.66. In a similar way, we can extrapolate the effort for the sixth iteration: (25 + 29 + 14.66) / 3 and so on. Note, however, that the "farther" the data point we extrapolate, the greater the risk of making an increasing estimation error, since the first estimation already has some error. Applying such a result to the next estimation can cause the error to grow (reduce the accuracy of the estimated value), because the errors can cumulate.

Wideband Delphi

In the expert-based *Wideband Delphi* method, experts make estimates based on their own experience. Each expert, in isolation, estimates the workload. The results are collected, and the experts discuss their current estimates. Each expert is then asked to make new predictions based on this information. The discussion allows all experts to rethink their estimation process. It may be, for example, that some experts did not take certain factors into account when estimating. The process is repeated until consensus is reached or there is a sufficiently small range in the obtained estimates. In this situation, for example, the mean or median of the expert estimates can be considered the final result.

Figure 5.3 shows the idea behind the Delphi method. From iteration to iteration, the range of values estimated by experts narrows. Once it is sufficiently small, one can draw, for example, the mean or median of all the estimates and consider it the result of the estimation process. Very often, this result will not deviate too much from the true value. This phenomenon is known colloquially as the "wisdom of the crowd" and results from the simple fact that errors cancel each other out: one expert may overestimate a little, another may underestimate a little, another may underestimate a lot, and yet another may overestimate a lot. Thus, the error usually has a normal distribution *with the mean at zero*. The spread of results can be very large, but averaging them will often be a sufficient approximation of the true value.

A variation of the Delphi method is the so-called planning poker. This is an approach commonly used in agile methodologies. In planning poker, estimates are

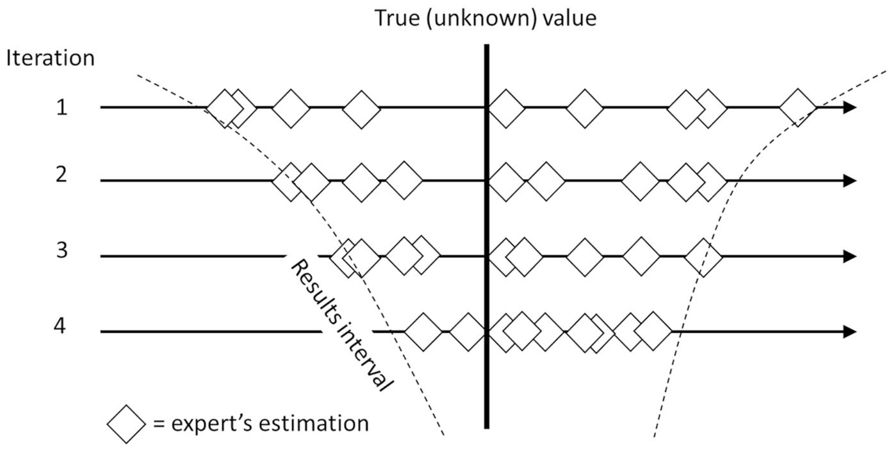

Fig. 5.3 The Wideband Delphi method

made using numbered cards. The values on the cards represent the amount of effort expressed in specific units that are well defined and understood by all experts.

Planning poker cards most often contain values derived from the so-called Fibonacci sequence, although for larger values, there may be some deviations. In the Fibonacci sequence, each successive term is the sum of the previous two. The following values are most often used:

1, 2, 3, 5, 8, 13, 20, 40, 100

As you can see, the last two values are no longer the sum of the previous two. This is because they are large enough that rounded values are assumed here, simply representing something large enough that it doesn't even make sense to estimate such a value accurately. If a team estimates, for example, the effort required to implement and test some user story and most experts throw a 40 or 100 card on the table, this means that the story is large enough that it is probably an *epic* and should be broken down into several smaller stories, each of which can already be reasonably estimated.

An example deck of planning poker cards is shown in Fig. 5.4. If a card with "?" is thrown, it means that the expert does not have enough information to do the estimation. A value of "0," on the other hand, means that the expert considers the story to be trivial to implement and that the time spent on it will be negligible. A card with coffee picture means "I am tired, let us do the break and grab some coffee!".

Other frequently used sets of values in planning poker are the set of successive powers of two:

1, 2, 4, 8, 16, 32, 64, 128

or so-called T-shirt sizes.

XS, S, M, L, XL.

In the latter case, the measurement (estimation) is not made on a numerical, *ratio* scale, and therefore does not directly represent the physical size of the estimated

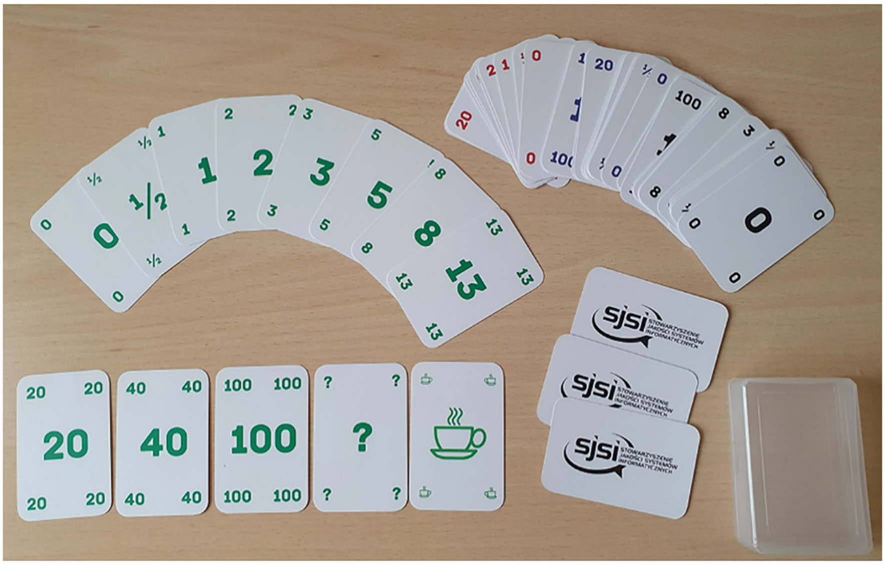

Fig. 5.4 Planning poker cards

effort. T-shirt sizes are defined only on an *ordinal* scale, which allows only the comparison of elements with each other. Thus, we can say that a story estimated as "L" is more labor-intensive than one estimated as "S," but we cannot say how many times more effort it will take to implement a story of size "L" relative to one of size "S." In this variant of the method, we are only able to prioritize stories by effort, grouping them into five groups of increasing effort size. However, this is not a recommended practice, because from the point of view of measurement theory, it is important that the differences between successive degrees of the scale are constant. In the case where the scale expresses story points, there is no problem: the difference between 4 and 5 points is the same as between 11 and 12 points and is 1 point. However, this is no longer so obvious in the case of a scale like "shirt sizes." So it should be assumed that, for example, the difference between S and XS is the same as between L and M, and so on.

Other scales are also acceptable. Which one specifically to use is a team or management decision. It is only important to understand how a unit of this scale translates into a unit of effort. Often, teams take as a unit the so-called user story point. Then one unit can mean the amount of effort required to implement/test one average user story. If the team knows from experience, for example, that implementing a 1 story point user story usually takes them 4 person-days, then a component estimated at 5 user story points should take about 5 * 4 = 20 person-days of effort.