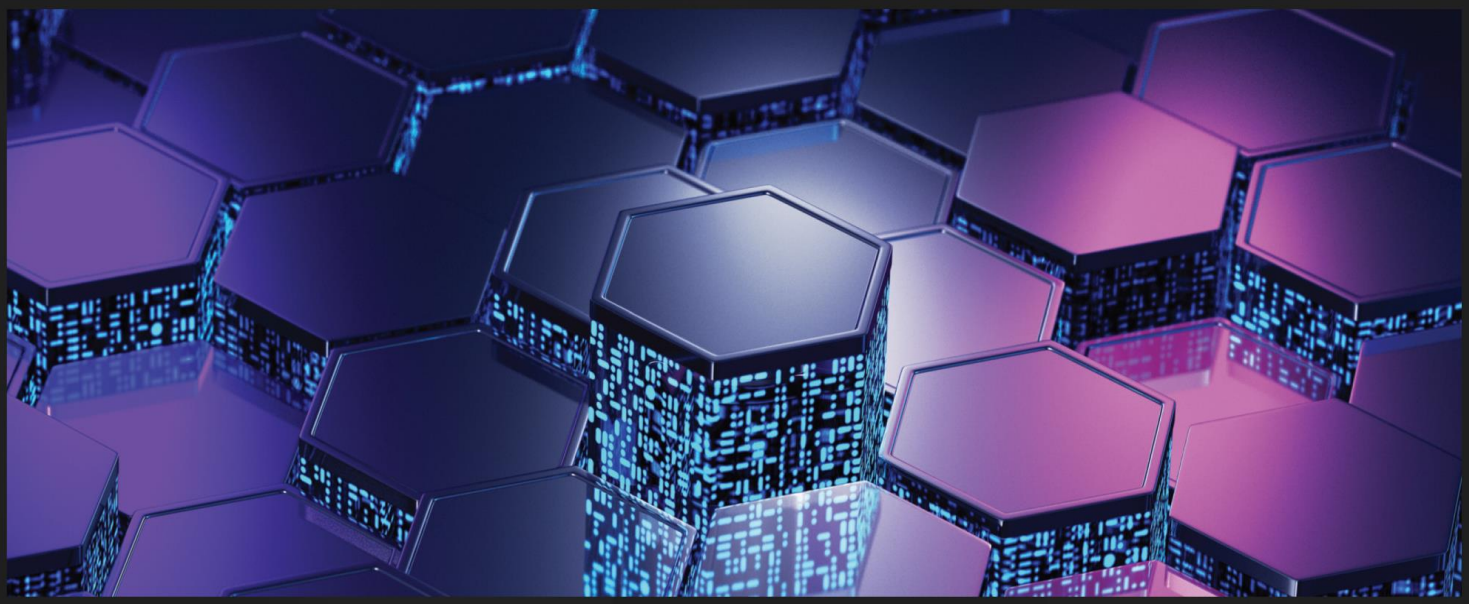
packt



1ST EDITION

Software Test Design

Write comprehensive test plans to uncover critical bugs in web, desktop, and mobile apps

SIMON AMEY

Preface

This audiobook will show you how to comprehensively test new software features, whether implemented on web pages, a desktop, or mobile applications. It covers everything from generating in—depth specifications to detailed black— and white-box testing, handling error cases and user experience considerations, to areas sometimes missed off test plans such as maintainability and operating under failure conditions.

The same bugs and defects crop up repeatedly in software development, so this audiobook has a huge range of tests to run and scenarios to check to help you achieve great test coverage. They're illustrated by dozens of example issues I've encountered during my decades in the software industry. While it's never possible to guarantee a product is bug—free, the recommendations here give you the best chance of catching defects.

Who this audiobook is for

Many different people can perform testing during a development cycle. In addition to dedicated testers, developers should always try their code, and product owners, support engineers, and technical writers can also provide more or less formal feedback.

This audiobook is for anyone involved in software testing, regardless of their official job title. Everyone can use the suggestions here to increase test coverage and improve the quality of the features you release.

Testing has a lovely learning curve. You can start with manual testing on easy—to—use public interfaces that have been well polished and documented. From there, you can go deeper into the system, checking logs and metrics and using internal interfaces. Testers with the right skill set can improve code directly by performing code reviews and writing unit tests. This audiobook describes all these approaches to improve your testing, whether you're a novice or an expert tester.

What this audiobook covers

Part 1, Preparing to Test, covers the necessary steps before you can start writing the test plan.

Chapter 1, *Making the Most of Exploratory Testing*, describes how best to perform exploratory testing and when you should do it. The main goal of exploratory testing is not to find bugs but to understand a feature better in order to improve the feature specification and test plans. It also finds any blocking issues that may delay future testing.

Chapter 2, *Writing Great Feature Specifications*, shows how to write comprehensive and useful feature specifications. This document will be the basis for all subsequent testing, so it needs to cover all the questions that could arise. That requires input from multiple groups, so this document always needs a thorough review.

That specification review is covered in *Chapter 3*, *How to Run Successful Specification Reviews*, in which you check the specification with the developers and product owners. The developers should highlight any special cases or conditions they had to add code for, while the product owner makes decisions that could affect the user experience and the scope of the changes. Successfully running that review to get what testers need from it is your responsibility.

Armed with the specification and the experience of exploratory testing, you can start designing the test plan. *Part 2, Functional Testing,* of this audiobook explains the many different types of functional testing, in which your product responds to different inputs.

Chapter 4, *Test Types, Cases, and Environments*, examines how best to write test cases and where you should run them. It shows how unit, integration, and system tests can work together and where they fit in the release cycle.

Chapter 5, *Black-Box Functional Testing*, is the most familiar area of testing, and the first one people think of. When you use the feature, does it work correctly and achieve its stated goals? Even within this type of testing, there are many ideas and suggestions for how to find common weaknesses and issues.

Chapter 6, *White-Box Functional Testing*, is informed by a knowledge of the underlying code and its architecture. Understanding how the code works lets you add another important set of tests.

Chapter 7, *Testing of Error Cases*, is devoted to invalid inputs or situations and how the application handles them. This is a large area of testing because the number of invalid inputs often massively outweighs the possible valid inputs. In addition, error cases are more likely to have defects because less time is spent thinking through their consequences than happy path scenarios.

Chapter 8, *User Experience Testing*, covers the unique set of considerations to ensure your application is easy to use. This is far more subjective than other areas of testing and may require judgments on, for instance, which command name is clearest, or which function will be used most and should be accessed most easily. It's vital to get those questions right to give your users the best experience.

Chapter 9, *Security Testing*, lists common security vulnerabilities that should routinely be checked by your test plans, as well as different attacks you can craft to ensure the appropriate protections are in place.

Chapter 10, *Maintainability*, considers your application's logging, event generation, and monitoring. If there is an issue, how easy will it be to diagnose and fix? This can be of lower importance to businesses, since it is not directly customer-facing, but is of high importance to the test team and other internal users who spend large amounts of time chasing down issues.

Those chapters conclude the functional areas of testing, where you perform a certain action and check a certain outcome. If the application under test has passed all these tests, then it will work for users under normal circumstances. *Part 3, Non—Functional Testing,* considers abnormal circumstances such as high load and system failures.

Chapter 11, *Destructive Testsing*, considers scenarios in which different parts of the system are deliberately disabled or degraded to ensure that the rest of the system behaves gracefully and can recover from issues.

Chapter 12, *Load Testing*, checks the behavior when your application runs at its maximum performance. While it may be able to perform individual actions correctly, is it reliable when they're repeated many times? These tests also check that your application continues to perform well, even when some subsystems are placed under load.

Finally, *Chapter 13*, *Stress Testing*, describes tests that deliberately load the system beyond its capabilities – for example, if too many entities are configured or if the system is taken beyond its loading limits. As with destructive testing, the correct behavior is to fail gracefully and recover once the stress condition is lifted.

The *Appendix* contains an example test plan that puts these ideas into practice for an example feature – users signing up to a web page.

Applying this guide gives you the best chance of finding defects in your software. However, despite all the suggestions here, in my experience, you find the most

interesting bugs when you go off the test plan. Trying out ideas for yourself or following up on something odd that you spot is a great way to think up test cases no one else has considered. Observation and curiosity are vital throughout testing, so always keep your eyes open.

The suggestions here aren't a recipe to be followed but instead a guide from which you can create your own test plans. Testing is like fishing. You can't guarantee what you'll find, and there'll be plenty of smaller catches among the big issues that you're really seeking. However, by looking in the most promising places and using the best techniques, you give yourself the greatest chance of success. Happy bug hunting!

Chapter 1

Tables

Advantages	Disadvantages
Quick to perform	Difficult to reproduce issues
Easy to see bugs	Should only be carried out by experienced testers
No prerequisites other than the code itself	Difficult to measure coverage
	Tests will be repeated later
	Poor coverage of non-functional tests

Table 1.1 – Advantages and disadvantages of exploratory testing

Figures

Figure 1.1 – An idealized waterfall model

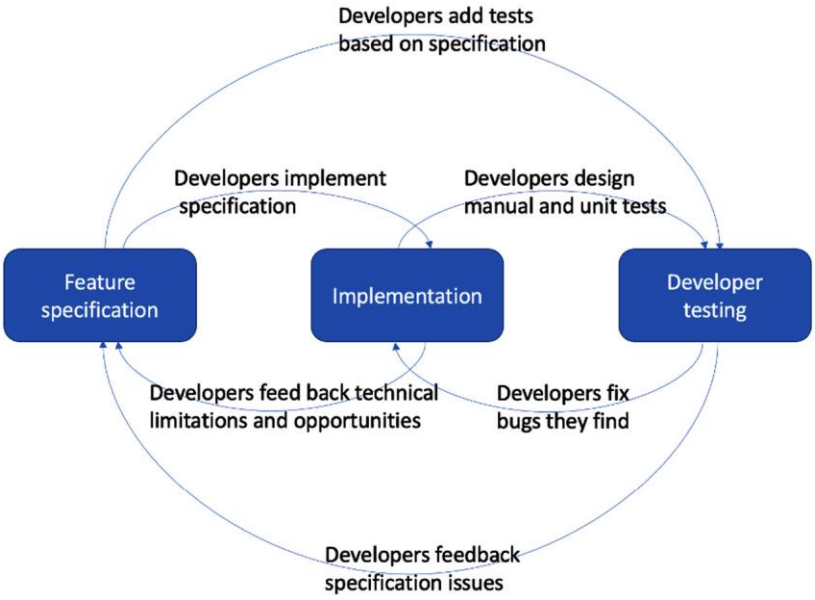


Figure 1.2 – Interactions and feedback in an Agile development model

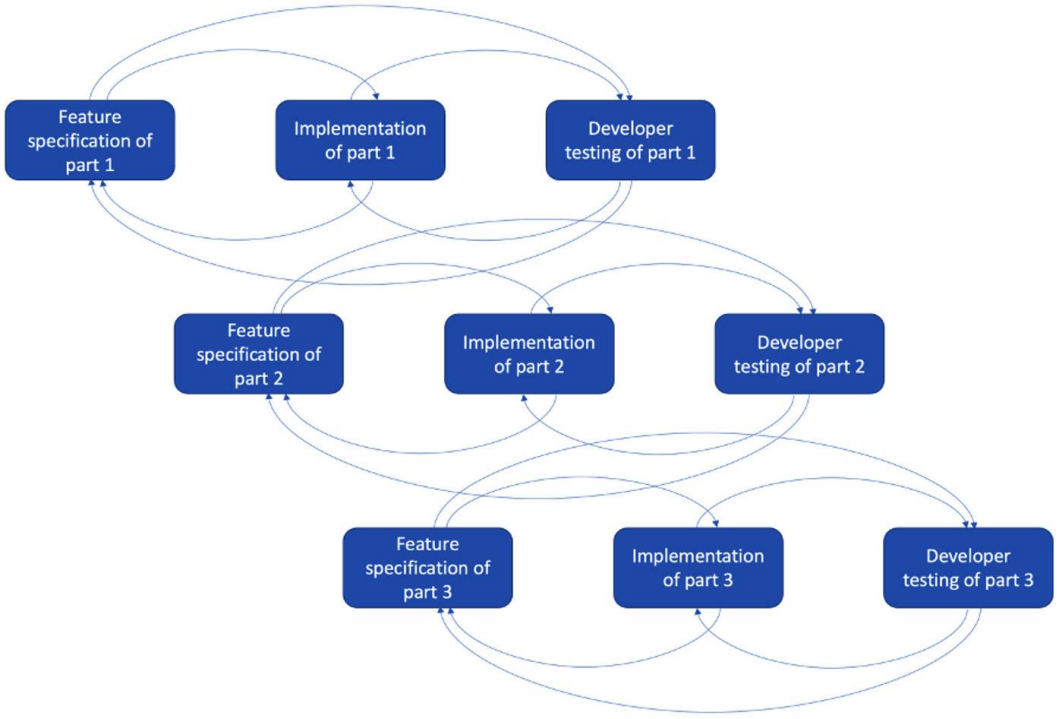


Figure 1.3 – Different parts of a feature developed in parallel within an Agile model

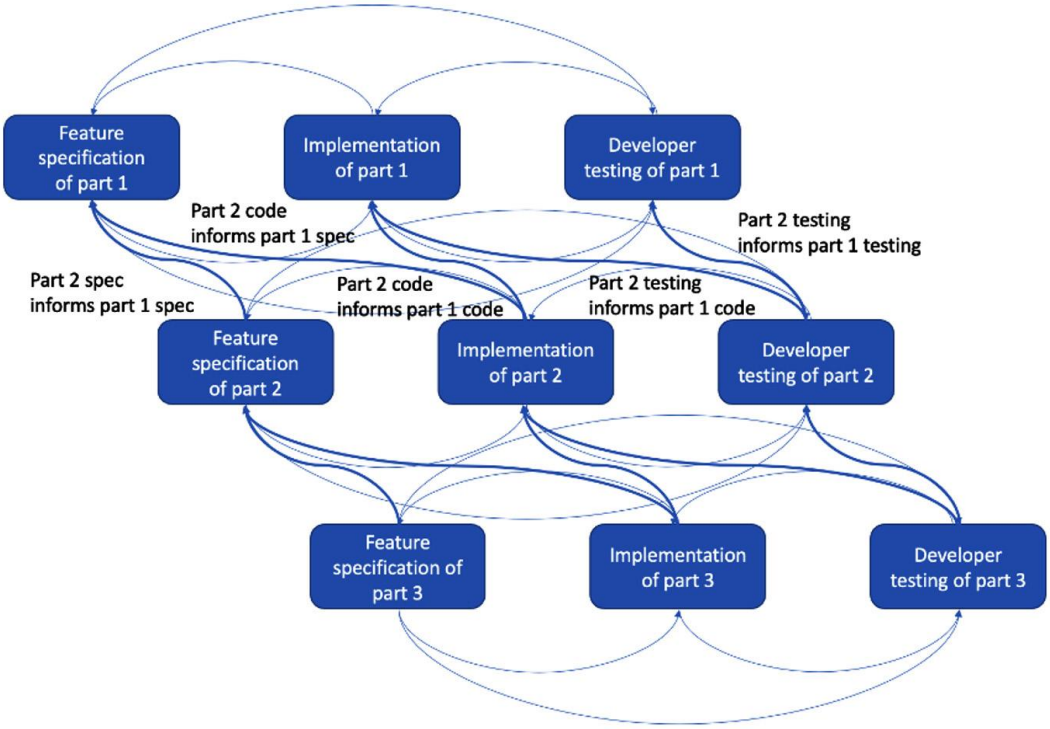


Figure 1.4 – Interaction between different parts of an Agile feature development

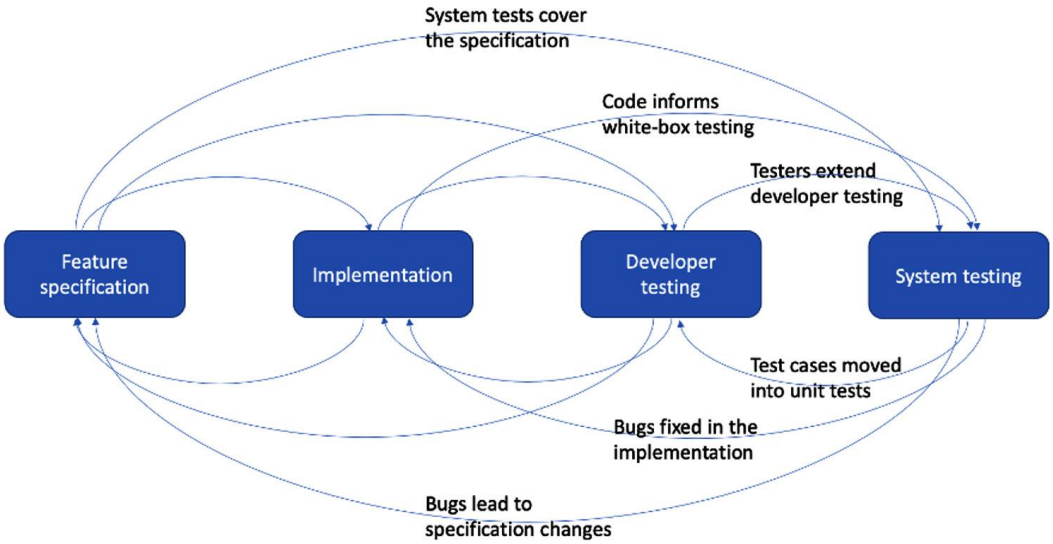


Figure 1.5 – System testing as part of an Agile development model

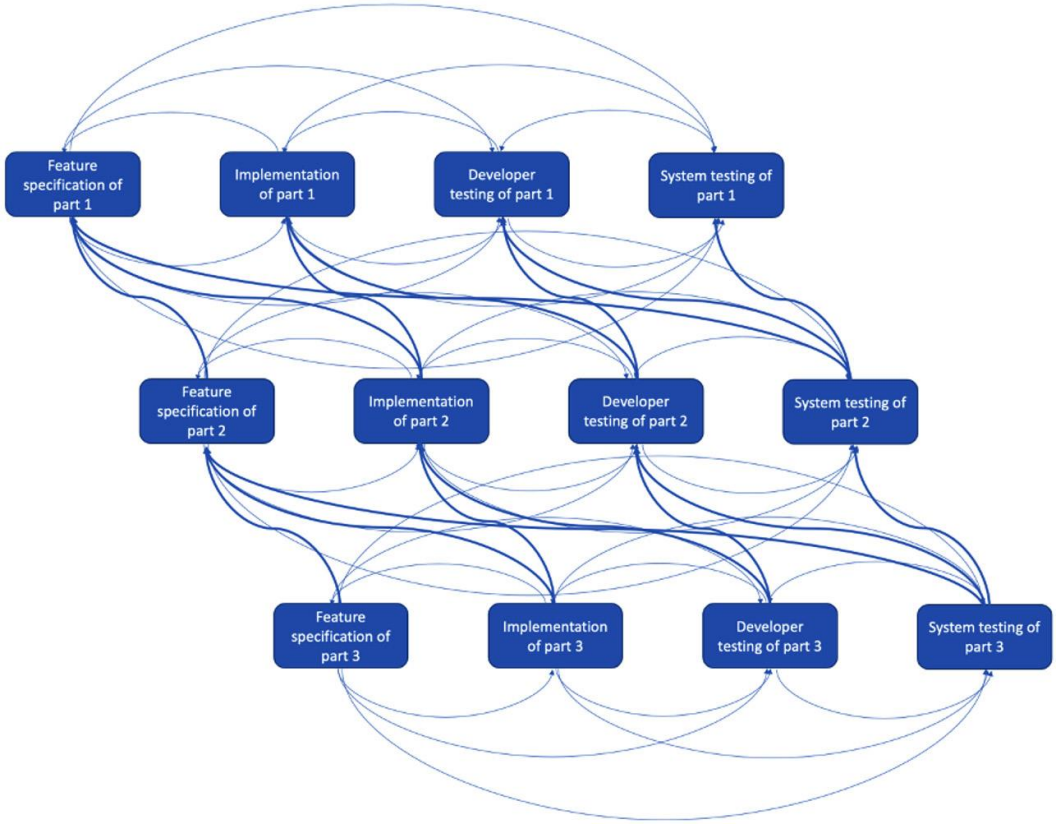


Figure 1.6 – Interaction between different parts of a feature under development, including system testing

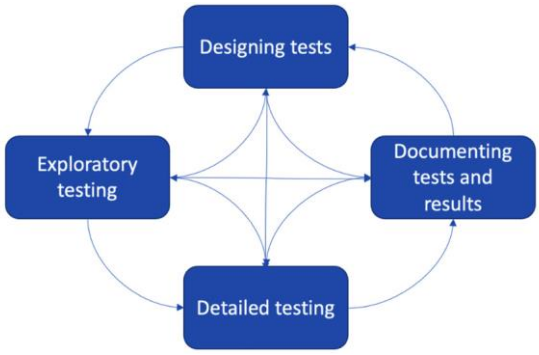


Figure 1.7 – The main test activities and their interactions

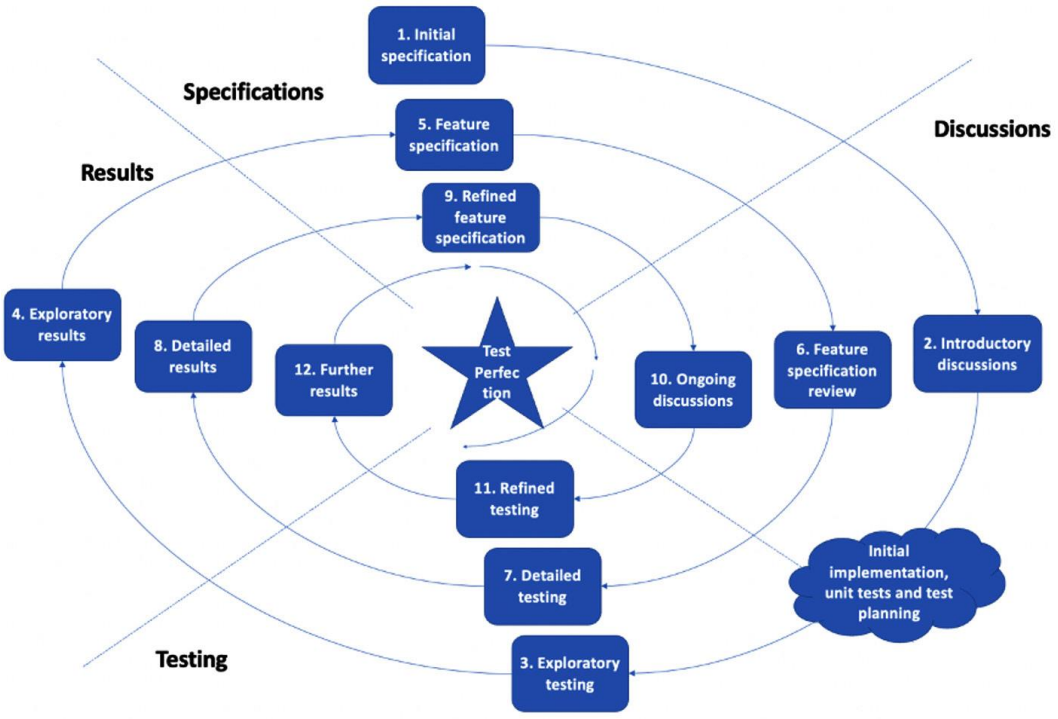


Figure 1.8 – The spiral model of test development

Chapter 3

Figures

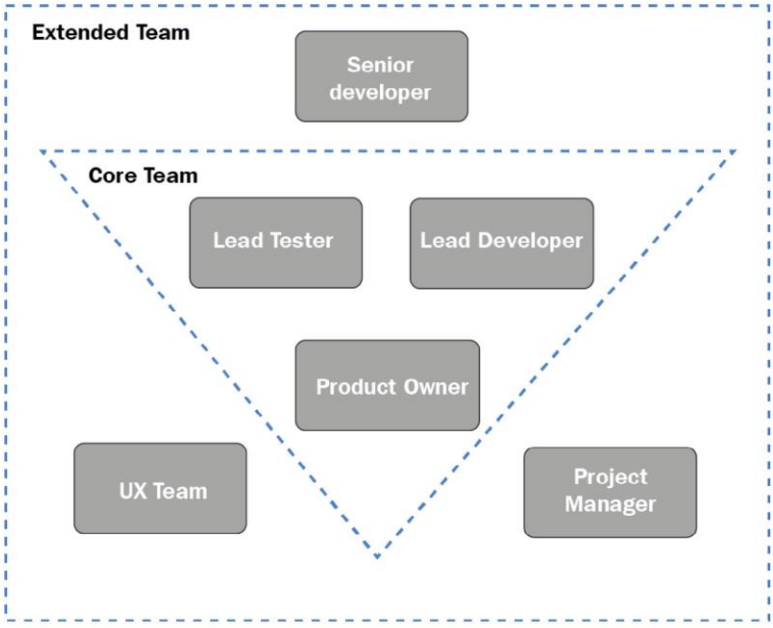


Figure 3.1 – Attendees for specification reviews

Chapter 4

Figures

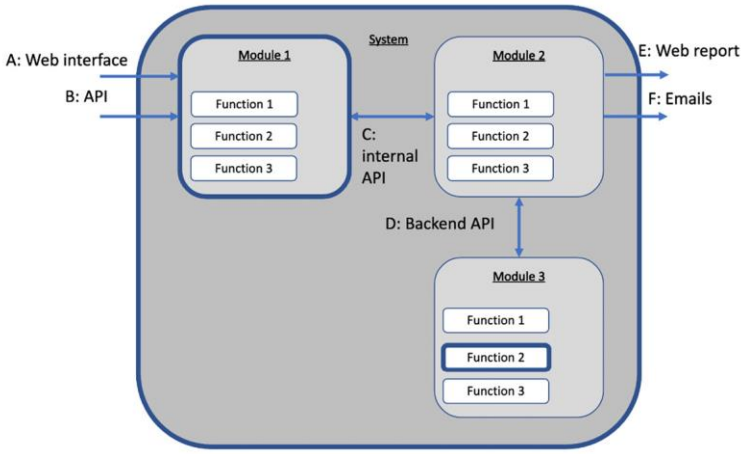


Figure 4.1 – Diagram of an example system, showing unit, integration, and system tests

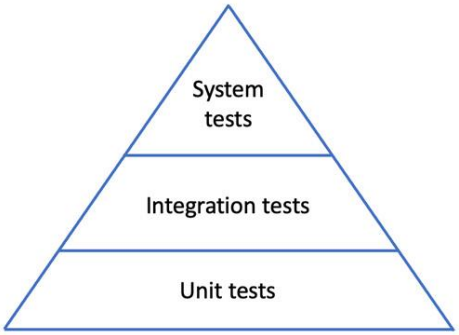


Figure 4.2 – The testing pyramid in terms of unit, integration, and system tests

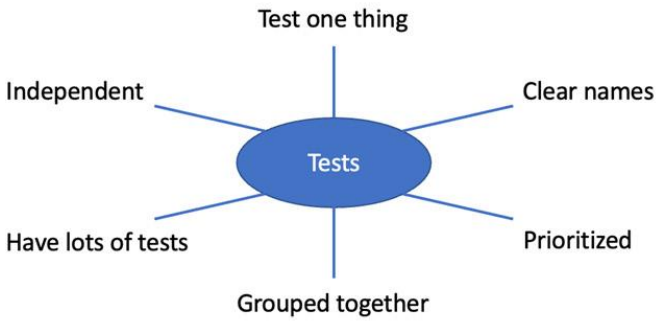


Figure 4.3 - Principles when writing test cases

Figure 4.4 - Subsets of test cases

Tables

Level	When it is run	Example test
Comprehensive tests	Once, when a new feature is added	Test that users can be created with every possible country setting
Regression tests	Any time that feature is changed	Test that users can be created with all localized languages and a selection of others
Manual/nightly tests	Nightly or on-demand	Test that all pages are localized for each language
CI-CD tests	After every code change	Test that one screen is localized for each language

Table 4.1 – Examples of different test plan levels

Chapter 5

Tables

Advantages	Disadvantages
Realistic	May not cover all the code
Covers the main functionality	Does not cover error cases
Takes a user's point of view	Difficult to debug
No need to learn implementation details	Repeats previous tests

Table 5.1 – Advantages and disadvantages of black-box testing

Module A	Module B	Feature flag	State
Downgraded	Downgraded	Disabled	1. Initial state
Upgraded	Downgraded	Disabled	2. First dependency ready
Upgraded	Upgraded	Disabled	3. Upgrades complete
Upgraded	Upgraded	Enabled	4. Feature fully enabled

Table 5.2 – The state of a gradual upgrade process

Field Name	Type	Required?	Parameters
UserName	string	Yes	Minimum length: 1 Maximum length: 256
Age	integer	No	
Country	string	No	ISO 3166 3 letter code

Table 5.3 – An example API command to create a user

Field Name	Type	Notes
UserID	integer	Used to reference a user in future API commands

Table 5.4 – An example API response to create a user

Error Name	Code	Notes
InvalidInput	**400**	There was an error in the request message
UserLimitReached	**405**	The system already has the maximum number of users configured
InternalError	**500**	A problem on the server meant the request could not be fulfilled

Table 5.5 – An example of API error responses to creating a user

Partition	Example test
Strings including spaces	"`hello world`"
Strings including accented characters	"`Élan`"
Strings including capital letters	"`Hello`"
Strings with special characters	"`";:!@£$%îèà{[('")}]`"

Table 5.6 – Examples of equivalence partitions for strings

	Email capitalization	Email domain
Test case 1	All lowercase	Company domain
Test case 2	Including capital (uppercase) letters	Shared domain

Table 5.7 – A test plan with independent variables

	Email capitalization	Email domain
Test case 1	All lowercase	Company domain
Test case 2	Including capital letters	Company domain
Test case 3	All lowercase	Shared domain
Test case 4	Including capital letters	Shared domain

Table 5.8 – A test plan with dependent variables

OS	Browser	App mode	Is it supported?

Windows	Chrome	Basic	Yes
Windows	Chrome	Advanced	Yes
Windows	Edge	Basic	Yes
Windows	Edge	Advanced	Yes
Windows	Safari	Basic	No
Windows	Safari	Advanced	No
Mac	Chrome	Basic	Yes
Mac	Chrome	Advanced	Yes
Mac	Edge	Basic	Yes
Mac	Edge	Advanced	No
Mac	Safari	Basic	Yes
Mac	Safari	Advanced	No

Table 5.9 – A decision table for application support on different operating systems and web browsers

OS	Browser	App mode	Is it supported?
Windows	Chrome	Basic	Yes
Windows	Chrome	Advanced	Yes

Table 5.10 – An excerpt of an expanded decision table

OS	Browser	App mode	Supported?
Windows	Chrome	–	Supported

Table 5.11 – An excerpt of a collapsed decision table

OS	Browser	App mode	Supported?
–	Chrome	–	Supported
–	Edge	Basic	Supported
Windows	Safari	–	Unsupported
Windows	Edge	Advanced	Supported
Mac	Edge	Advanced	Unsupported
Mac	Safari	Basic	Supported

Mac	Safari	Advanced	Unsupported

Table 5.12 – A collapsed decision table for application support

A	B	C	D	E1	E2	Effect 1	Effect 2
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	1
0	0	1	0	0	1	0	0
0	0	1	1	0	0	0	1
0	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1
0	1	1	0	1	1	1	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	1
1	0	1	0	1	1	1	0
1	0	1	1	1	0	0	1
1	1	0	0	1	0	0	1
1	1	0	1	1	0	0	1
1	1	1	0	1	1	1	0
1	1	1	1	1	0	0	1

Table 5.13 – The truth table for the cause-effect graph shown in Figure 5.4

Links

There are more complex rules governing the precise format of allowed email addresses. For more details, see https://datatracker.ietf.org/doc/html/rfc5322.

Figures

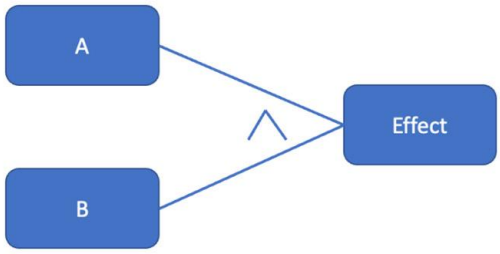


Figure 5.1 – AND relationship between two variables and an effect

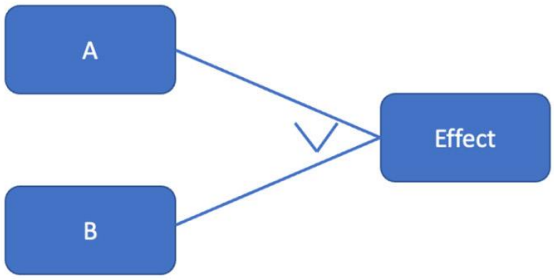


Figure 5.2 – OR relationship between two variables and an effect

Figure 5.3 – NOT relationship between variable A and an effect

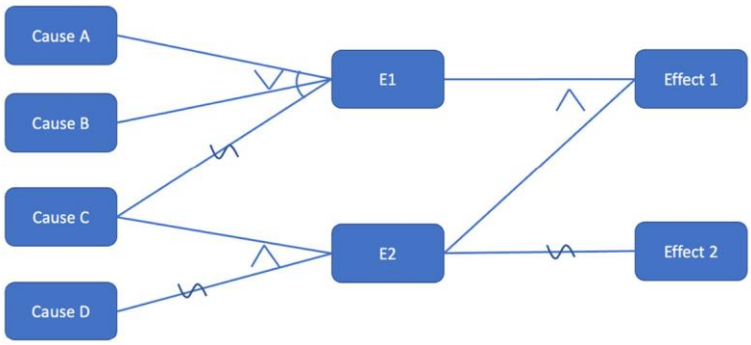


Figure 5.4 – A complete example cause-effect graph

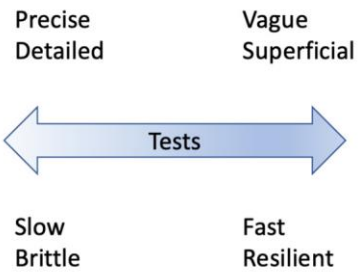


Figure 5.5 – Trading off precision versus resilience in tests

Strings

String 5.1

```
Hello ;:!@£$%îèà{[('")}]
```

Chapter 6

Code

Code 6.1

```
function_a() {
      print("This is function A")
}
FunctionB()
{
  print("Function B behaves the same but looks different")
}
```

Code 6.2

```
create_user(permission)
{
     if(permission == ADMIN)
     {
           user = create_user()
           make_admin(user)
     }
     else
     {
           user = create_user()
     }
      return user
}
```

Code 6.3

```
create_user(permission)
{
     user = create_user()
     if(permission == ADMIN)
     {
           make_admin(user)
     }
```

```
      return user
}
```

Code 6.4

```
create_user(str name, int age)
{
      if(name == "")
            throw(error, "Username is blank")
      if(age < 18)
            throw(error, "Users must be 18 or over")
      if(age > 130)
            throw(error, "Age value is too large")
      …
}
```

Tables

Advantages	Disadvantages
Can find issues without running the code	Requires testers with coding skills
Easy to automate	Hard to catch unimplemented requirements
Easy to measure coverage	Fragile to code refactoring
Can check all code paths	
Easy to investigate failures	

Table 6.1 – Advantages and disadvantages of white-box testing

	Coverage type	Detail
1	Function	Function coverage ensures that every function has been called at least once.
2	Statement	Statement coverage checks that each line of the program has been run at least once.
3	Decision	Decision (or branch) coverage ensures that each logical branch of a program has been run at least once.

4	Condition	Condition coverage ensures that each variable in conditional statements takes all possible values at least once.
5	Modified condition/decision	Modified condition/decision coverage ensures that each variable in a conditional statement has affected the statement outcome at least once.
6	Multiple condition	Multiple condition coverage ensures that every possible combination of variables is checked for each conditional statement.
7	Parameter value	Parameter value testing uses equivalence partitioning to ensure that all main types of parameter input have been tested at least once.
8	Loop	Loop coverage ensures that loops have been run multiple times.
9	State	State coverage ensures that each state in a finite state machine has been tested.

Table 6.2 – Types of code coverage

Code coverage type	**Test steps required to test "If (A || B || C) then…"**
Statement coverage	1
Decision coverage	2
Condition coverage	2
Condition/decision coverage	2
Modified condition/decision coverage	4
Multiple condition coverage	8

Table 6.3 – Number of tests required by different coverage types

Advantages	**Disadvantages**
Quicker to write	Incomplete test coverage
Quicker to run	May give a false sense of security
Easier to measure	

Easier to understand	

Table 6.4 – Advantages and disadvantages of lighter forms of unit test coverage

Test	Login	First action	Second action
1	Fail, then Pass	Action A. Continue? No	Quit
2	Pass	Action B	Quit
3	Pass	Action C	Quit

Table 6.5 – Tests achieving state coverage for the example state machine

Test	Login	First action	Second action	Third action
1	Fail, then Pass	Action A. Continue? No	Quit	–
2	Pass	Action A. Continue? Yes	Action B	Quit
3	Pass	Action B	Quit	–
4	Pass	Action C	Quit	–

Table 6.6 – Tests achieving transition coverage for the example state machine

Test	Login	First action	Second action	Third action
1	Fail, then Fail, then Pass	Action A. Continue? No	Quit	
2	Fail, then Pass	Action B	Quit	
3	Fail, then Pass	Action C	Quit	
4	Pass	Action A. Continue? No	Action A	Quit
5	Pass	Action A. Continue? Yes	Action B	Quit
6	Pass	Action A. Continue? No	Action C	Quit
7	Pass	Action B	Action A	Quit
8	Pass	Action B	Action B	Quit
9	Pass	Action B	Action C	Quit

10	Pass	Action C	Action A	Quit
11	Pass	Action C	Action B	Quit
12	Pass	Action C	Action C	Quit

Table 6.7 – Tests achieving 1-switch coverage for the example state machine

Figures

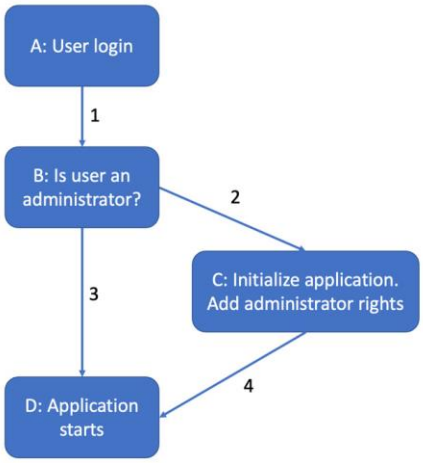


Figure 6.1 – Program initialization with a single if statement

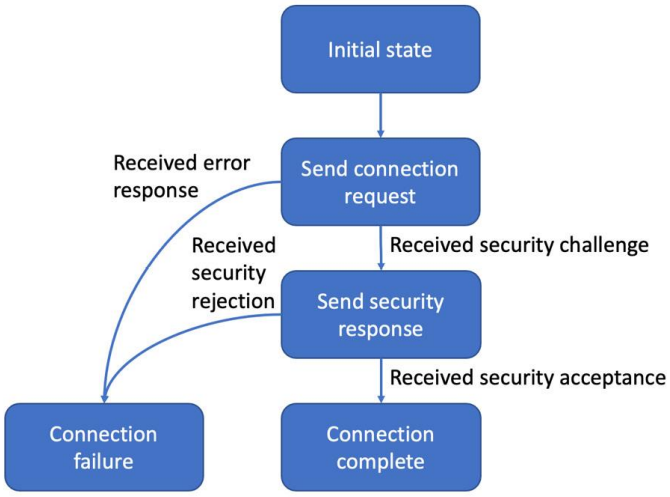


Figure 6.2 – A simple state machine

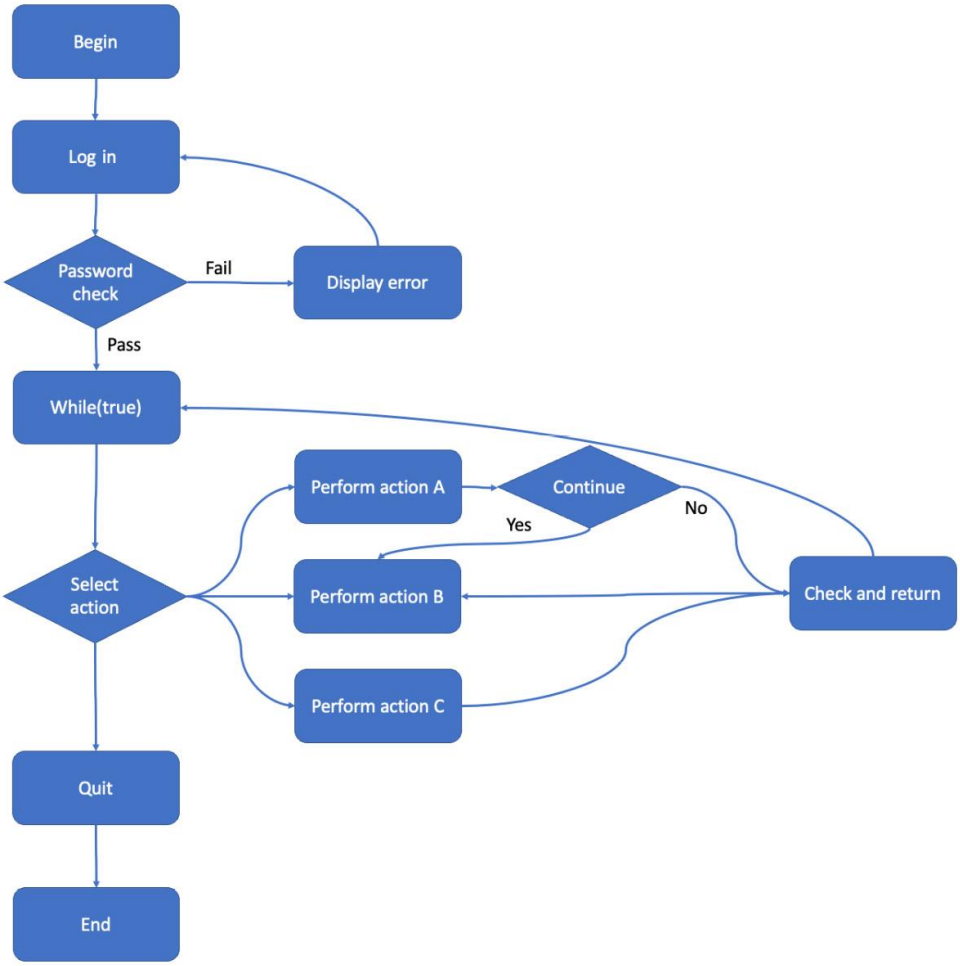


Figure 6.3 – An example state machine

Chapter 7

Tables

Advantages	Disadvantages
May not have been considered by the product owners	These are rarer conditions, so these tests are a lower priority
May not have been considered by developers, so failures may be more serious	Can be difficult to generate specific error cases
Important to give user feedback if there is a problem	Difficult to tell the ideal behavior in a failure mode
Good error handling makes debugging easier	
Can help to find linked issues	

Table 7.1 – Advantages and disadvantages of testing error cases

Packet	Time sent	Transit time	Time received
1	0ms	20ms	20ms
2	10ms	200ms	210ms
3	20ms	20ms	40ms

Table 7.2 – Varying transit times causing packet re-ordering

Packet	Time sent	Transit time	Time received
1	0ms	20ms	20ms
2	10ms	200ms	210ms
3	20ms	20ms	40ms
2, retry	110ms	20ms	130ms

Table 7.3 – Varying transit times causing duplicate packets

Figures

Figure 7.1 – Invalid inputs blocked by the frontend

Figure 7.2 – Invalid inputs blocked by the backend

Figure 7.3 – Invalid inputs blocked by the storage system

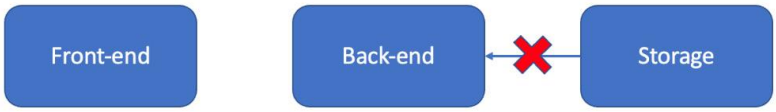


Figure 7.4 – Invalid data detected when being read

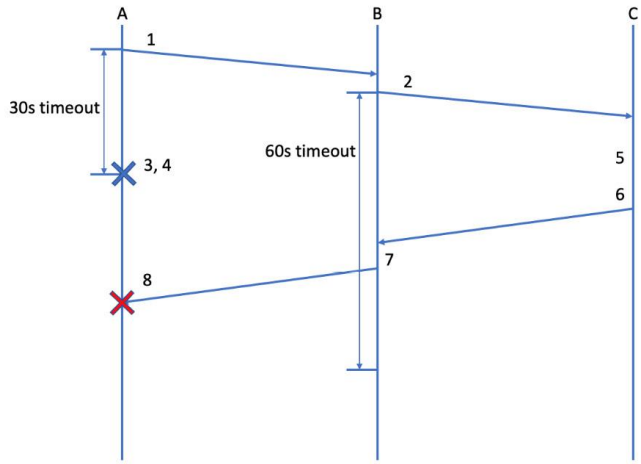


Figure 7.5 – Message failures with incorrectly figured timeouts

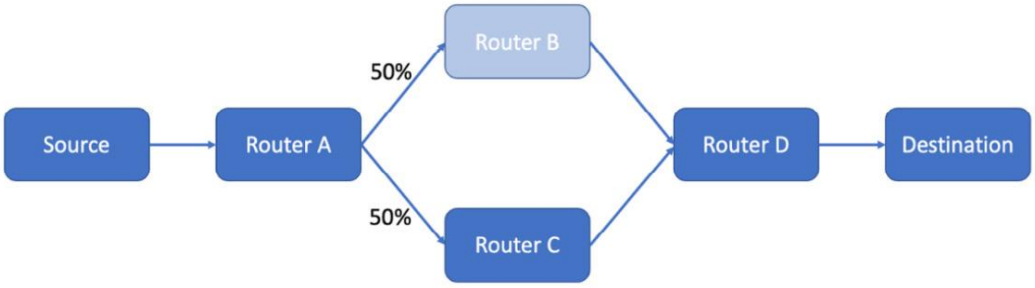


Figure 7.6 – An example network suffering degradation

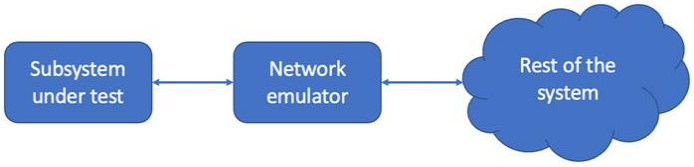


Figure 7.7 – A network emulator in line with a subsystem under test

Chapter 8

Tables

Strengths	Weaknesses
Best way to detect this class of issue	Mainly manual
Can find blocking issues	Subjective
Directly affects users	Difficult to predict what will cause problems for customers
	Costly to run usability studies

Table 8.1 – Strengths and weaknesses of UX testing

User	Connection	Status
bob.bobson@example.com	UDP Rate up: 10 Mbps Rate down: 20 Mbps	Online 5 minutes
john.smith@example.com	TCP Rate up: 5 Mbps Rate down: 8 Mbps	Offline 5 days
jane.doe@example.com	UDP Rate up: 10 Mbps Rate down: 100 Mbps	On call 5 minutes

Table 8.2 – A table with too much information in each column

User	Connection Type	Rate up (Mbps)	Rate down (Mbps)	Status	Duration
bob.bobson@example.com	UDP	10	20	Online	5 minutes
john.smith@example.com	TCP	5	8	Offline	5 days

jane.doe@example.com	UDP	10	100	On call	10 minutes

Table 8.3 – A table with the same data spread across columns

Inviter	Invitee	Date
adam.blake@example.com	**bob.bobson@example.com**	2/1/22
bob.bobson@example.com	**john.smith@example.com**	4/3/22
bob.bobson@example.com	**jane.doe@example.com**	5/6/22

Table 8.4 – A table with the same data in two different columns

Figures

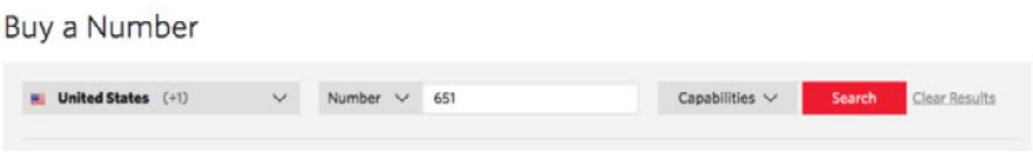


Figure 8.1 – A Twilio search box

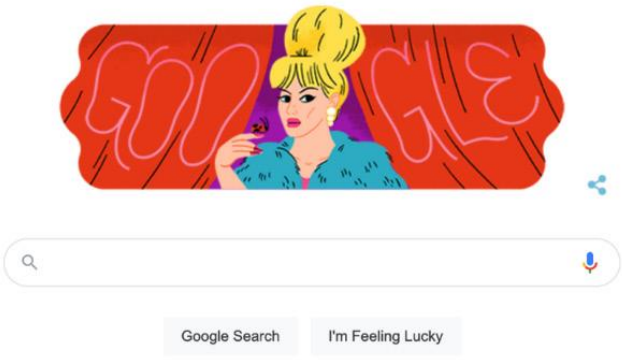


Figure 8.2 – Google's famously sparse home page

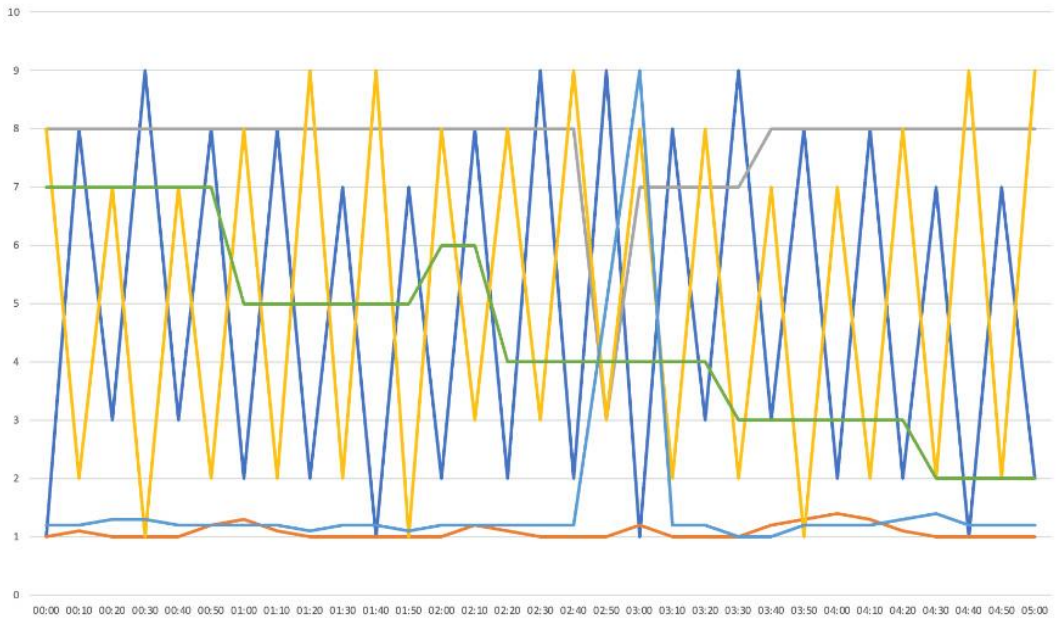


Figure 8.3 – An example of a graph containing too much information

Big important customer 1	
Small new customer 2	
Long time customer 3	

Figure 8.4 – Information displayed with the edit icon separate from the name

Chapter 9

Tables

Advantages	Disadvantages
The best way to discover security vulnerabilities	Requires a wide breadth of testing
Many tools available	Requires dedicated knowledge and skills
Easy to automate	Constantly evolving security vulnerabilities and threats
	Only part of a broader security story

Table 9.1 – Advantages and disadvantages of security testing

Impact Likelihood	Negligible	Minor	Moderate	Major	Severe
Highly likely	Low	Medium	High	High	High
Likely	Low	Medium	Medium	High	High
Possible	Low	Low	Medium	Medium	High
Unlikely	Low	Low	Medium	Medium	Medium
Highly unlikely	Low	Low	Low	Medium	Medium

Table 9.2 – Determining the risk of a vulnerability given its impact and likelihood

Links

To help you choose a security scanner to use for your testing, OWASP maintains an extensive list of different security scanners here: owasp.org/www-community/Vulnerability_Scanning_Tools.

Code

SQL injection

Consider this snippet of Python that uses a string without validating it first:

```
SQLCommand = 'INSERT INTO users VALUES (username);'
```

This works fine if `username` is **"Simon Amey"**:

```
SQLCommand = 'INSERT INTO users VALUES ("Simon Amey");'
```

However, it leaves it open to attack if the `username` string contains control characters:

```
username = 'Simon Amey"); DROP TABLE users;':
```

This now makes the string look like this:

```
SQLCommand = 'INSERT INTO users VALUES ("Simon Amey"); DROP
TABLE users;");'
```

HTML injection

At a simple level, you can test for this by adding HTML tags to your inputs, for example:

```
username = "<b>Simon Amey</b>"
```

Cross-site scripting attacks

```
Username = "Simon Amey<script>alert('Test')</script>"
```

Figures

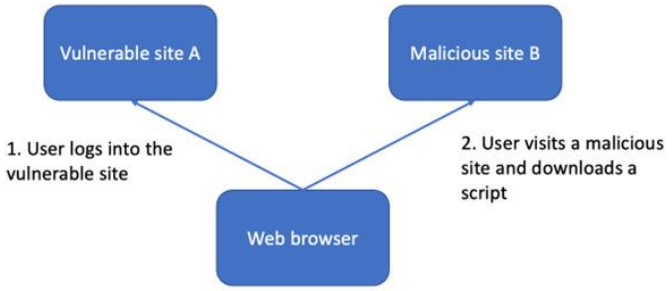


Figure 9.1 – CORS attack stage 1

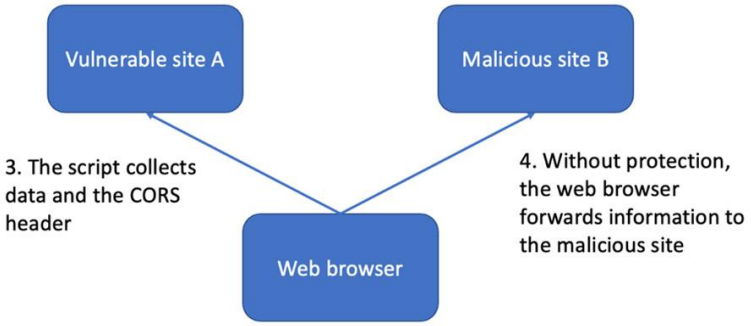


Figure 9.2 – CORS attack stage 2

Chapter 10

Tables

Advantages	Disadvantages
The best way to discover maintainability issues	Not a priority for product development
It makes your life easier	Requires knowledge of user experience design
Speeds up the development process	Hard to predict what information you will need
	Benefits from feedback late in the development cycle

Table 10.1 – Advantages and disadvantages of maintainability testing

Alert level	Action	Examples
Critical	Requires immediate attention	System unavailable Users unable to sign up Data loss in progress A minor function is unavailable An error preventing internal users from doing their jobs Consistent rejections from third-party systems Security issues
Error	Requires attention	High disk usage A transient error not currently occurring Intermittent rejections from third-party systems An internal problem with a workaround Unhandled exceptions
Warning	Does not require engineering attention An expected failure	An outgoing API request fails Customer mistakes (incorrect passwords, misconfiguration, invalid API queries, and so on)

		Temporary loss of connection Attempting to access an invalid address on your server

Table 10.2 – Alert levels and their definitions

Figures

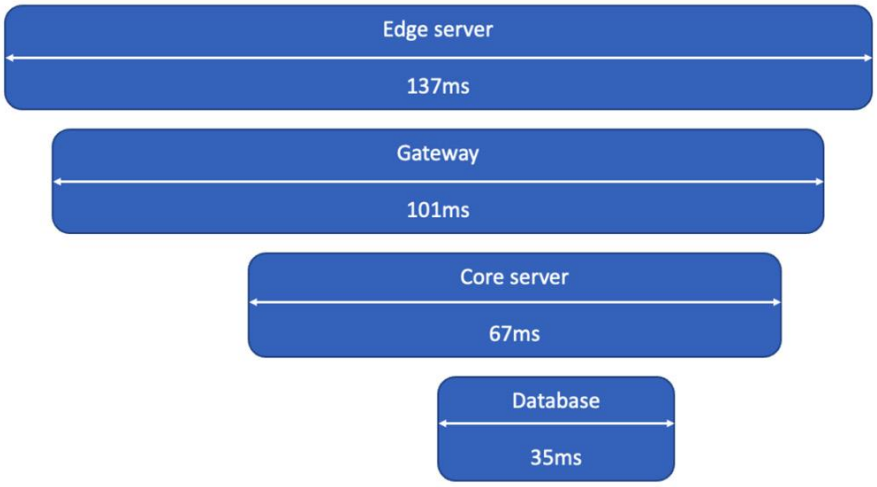


Figure 10.1 – Tracing a command through layers of a system

Chapter 11

Tables

Advantages	Disadvantages
The only way to detect this class of issue	Tests rare cases
Verifies your resilience and recovery plans	Requires specialist environments, tools, and tests
	Need to filter valid failures from invalid ones

Table 11.1 – Advantages and disadvantages of destructive testing

Figures

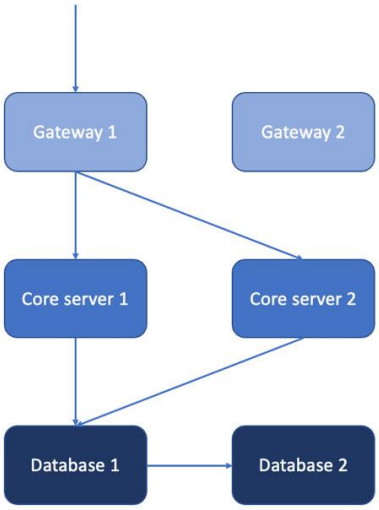


Figure 11.1 – Different failover strategies at different application layers

Users

ID	First name	Second name
123	Alice	Smith
124	Bob	Jones
125	Charles	Brown

Statistics

ID	User ID	Stat 1
32	123	4
32	124	7
34	125	2

Figure 11.2 – Database tables before adding entries

Users

ID	First name	Second name
123	Alice	Smith
124	Bob	Jones
125	Charles	Brown
126	**Daniela**	**Wilson**

Statistics

ID	User ID	Stat 1
32	123	4
32	124	7
34	125	2
35	**126**	**0**

Figure 11.3 – Database tables after successfully adding an entry

Users

ID	First name	Second name
123	Alice	Smith
124	Bob	Jones
125	Charles	Brown
126	Daniela	Wilson
127	**Eliza**	**Taylor**

Statistics

ID	User ID	Stat 1
32	123	4
32	124	7
34	125	2
35	126	0

Figure 11.4 – Database tables interrupted during writing

Figure 11.5 – Blocking the service under test from sending messages

Figure 11.6 – Blocking the remote service from receiving messages

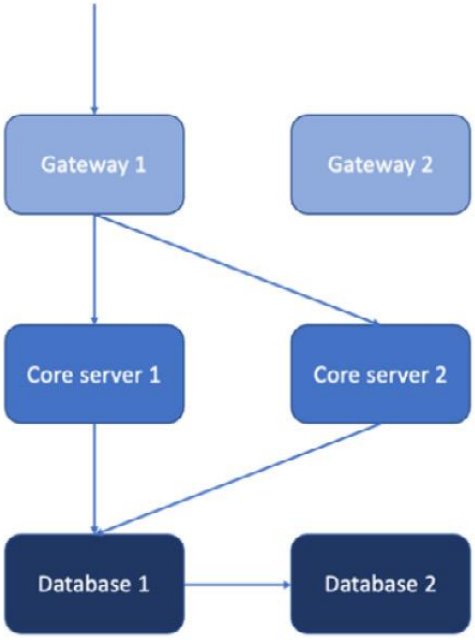


Figure 11.7 – Different failover strategies at different application layers

Chapter 12

Tables

Advantages	Disadvantages
The only way to find this class of issue	Requires dedicated tools
The only way to measure system performance	Requires high-quality tools
Uncovers hard-to-find issues	Harder to debug
Finds software inefficiencies	Less realistic scenarios
	Expensive

Table 12.1 – Advantages and disadvantages of load testing

Layer	Purpose
Test control	The logic and intelligence for choosing which test sequences to run
Test sequences	Sequences of actions, such as create-check-delete-check loops
Individual commands	Individual actions such as creating, checking, and deleting entities

Table 12.2 – Layers of load testing logic

Test ID	1
Time	20.43 minutes
Requests	103,472
Successes	103,244
Failures	228
Success rate	99.78%
Minimum latency	10.3 ms
Average latency	15.98 ms
Maximum latency	302.44 ms
Inbound data total	10.3 GB
Outbound data total	30.4 GB

Table 12.3 – Example load test result output

Figures

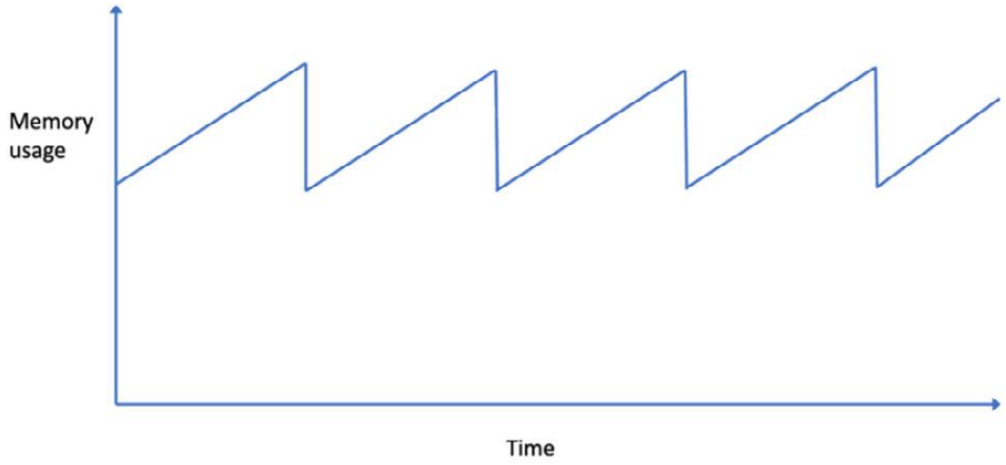


Figure 12.1 – A sawtooth pattern of memory usage indicating a memory leak

Figure 12.2 – Example load testing output with a period of high latency

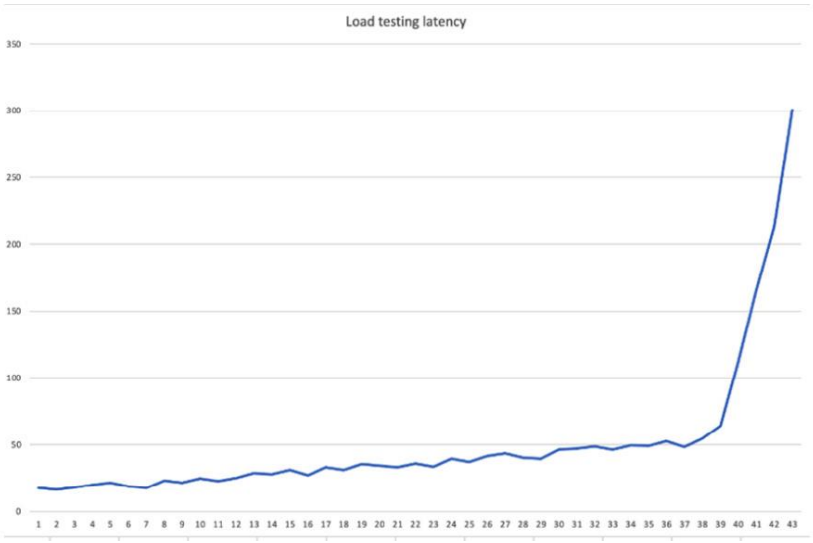


Figure 12.3 – Example load testing output with increasing latency

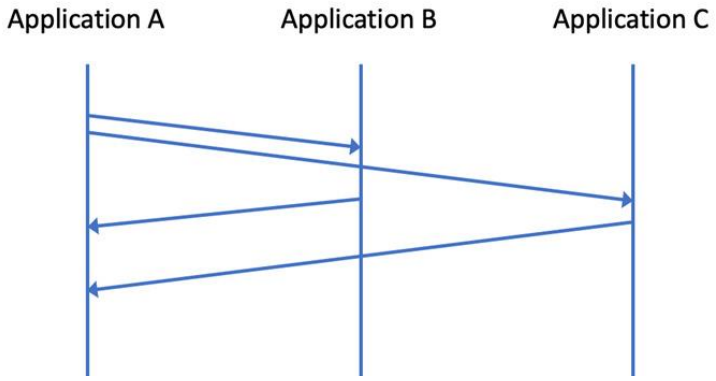


Figure 12.4 – Sending messages to external systems and receiving replies in order B then C

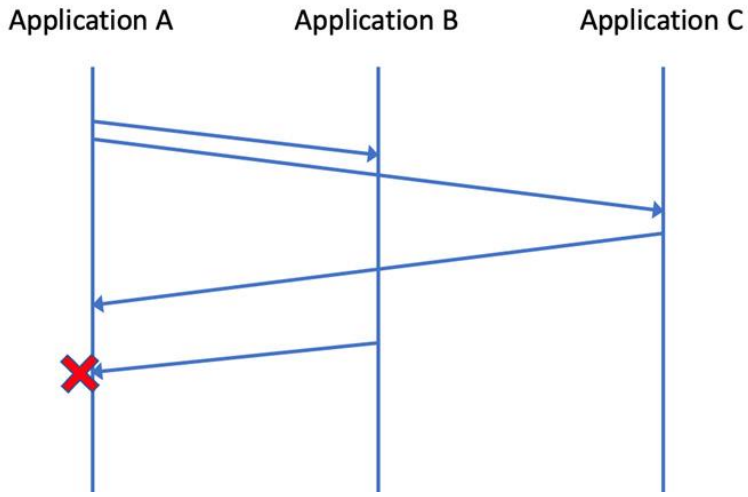


Figure 12.5 – Sending messages to external systems and receiving replies in order C then B

Chapter 13

Tables

Advantages	Disadvantages
The only way to find this class of issue	Requires dedicated tools
Discovers the limits of your system	Requires high-quality tools
Uncovers hard-to-find issues	Harder to debug
Finds software inefficiencies	Less realistic scenarios
Prepares you for overload scenarios	Expensive
	Difficult to tell the ideal behavior in a failure mode

Table 13.1 – The advantages and disadvantages of stress testing

		Policed operation	Unpoliced operation
Valid operations	Ongoing state	Game length Call length Processing time	Web session length
	Separate operations	Transactions per day Operations available on a free plan	Signing up Logging in
Invalid operations	Ongoing state	Detecting heartbeat timeouts Killing long-running transactions/processes	System degradation
	Separate operations	Incorrect password attempts	Signing up too many users Creating too many entities Accessing invalid URLs

Table 13.2 – Different classes of soak testing tests

Figures

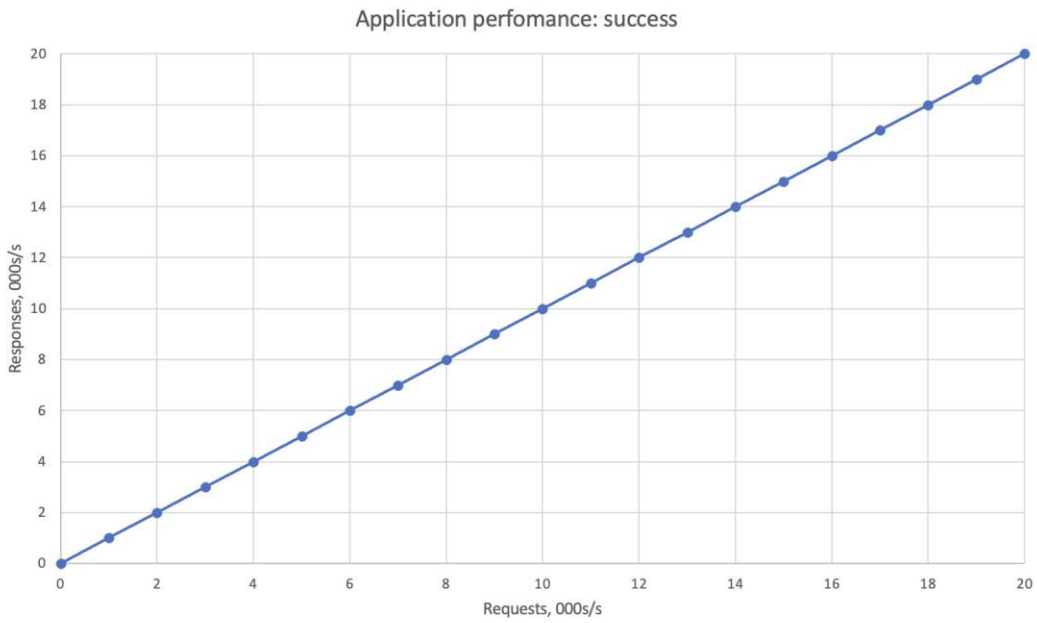


Figure 13.1 – Handling additional load successfully

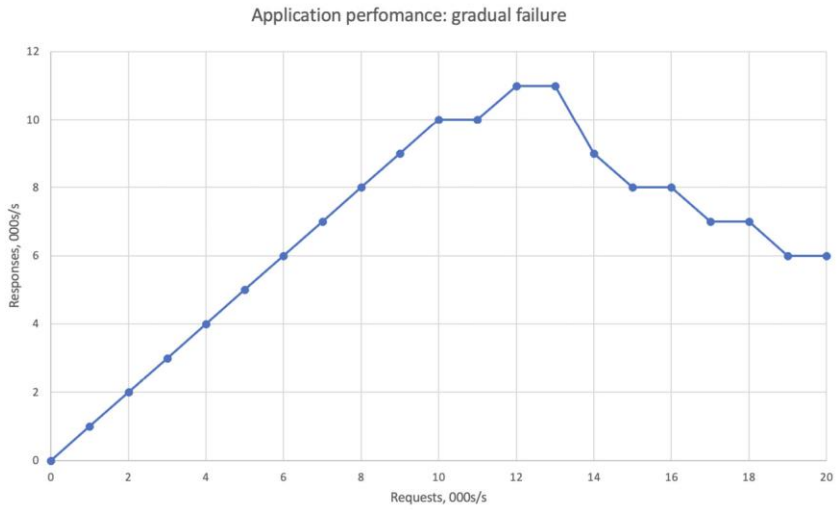


Figure 13.2 – Handling additional load with gradual degradation

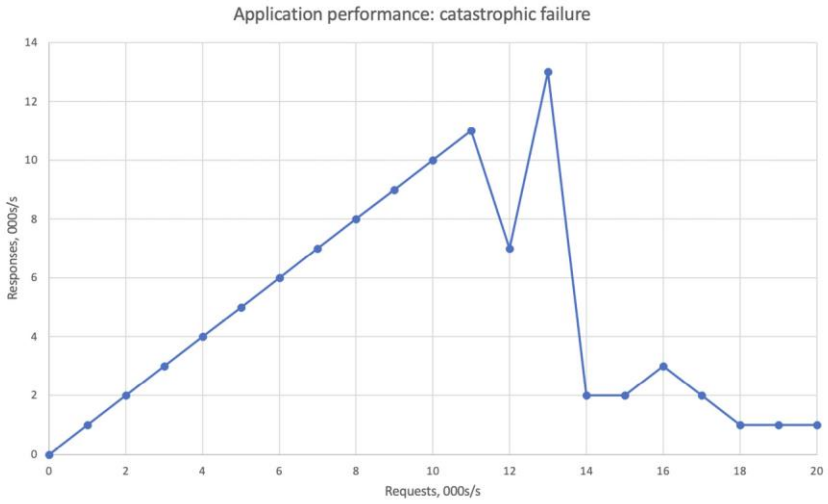


Figure 13.3 – Handling additional load with catastrophic failure

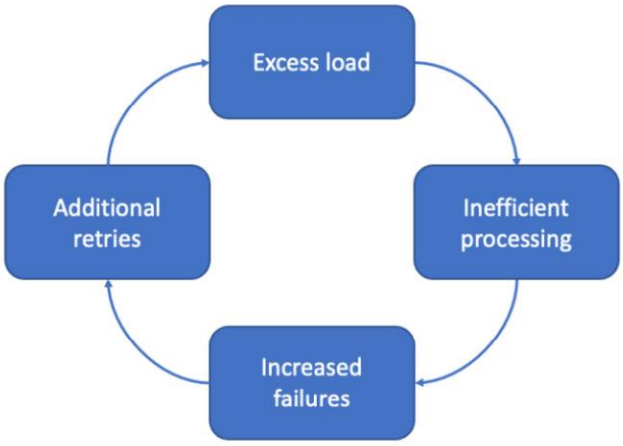


Figure 13.4 – A positive feedback loop of loading failures producing even more load

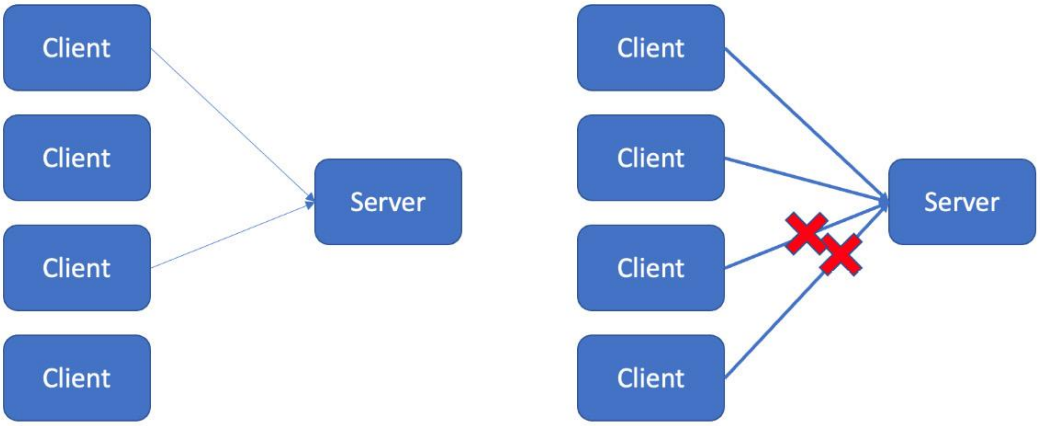


Figure 13.5 – A server fails under a heavy load

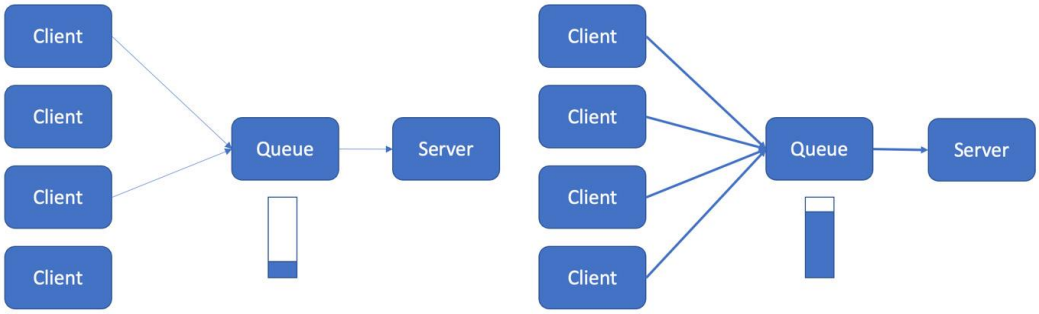


Figure 13.6 – A queue handles a spike in load

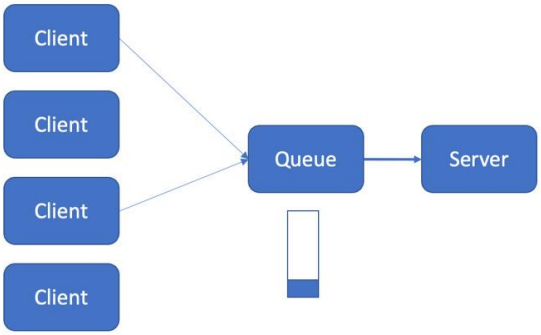


Figure 13.7 – A queue empties after a spike in load

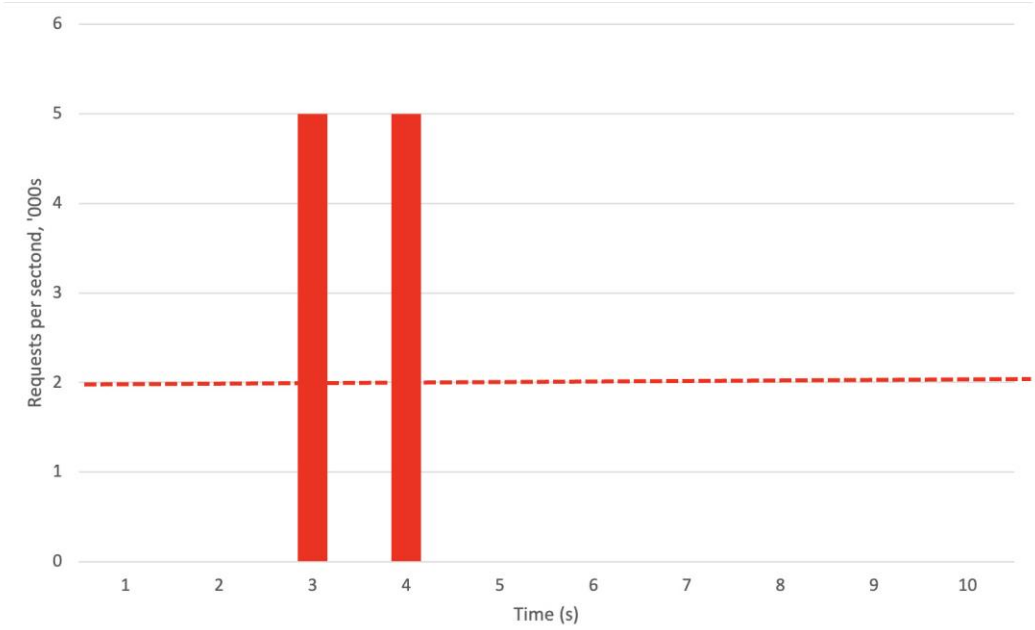


Figure 13.8 – Peaks of traffic overwhelm the server

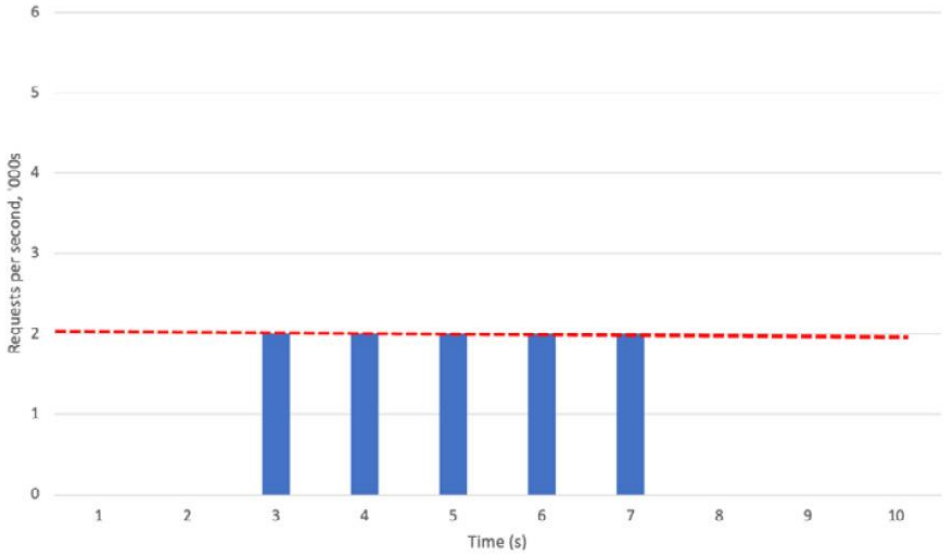


Figure 13.9 – A queue spreads out the load

Appendix – Example Feature Specification

This is an example of feature specification for a login feature. It is incomplete because it doesn't include the user interface descriptions, such as the wording of error messages and the appearance of the screens and buttons. All those gaps are noted, and to fully test the feature, you would need all that information from the user experience team.

However, it is complete as a feature specification of the login feature:

ID	Requirement	Notes
Section 1 – Enabling login		
1.1	In *version 5.7.1* and later, users can log in to the **example.com** website.	
1.2	By default, when upgrading to a version that supports logging in, logging in is disabled.	
1.3	Logging in can be enabled via a configuration option in *version 5.7.1* and later.	Set the **enable_login** option to **True** in the **login_configuration.conf** file.
1.4	Logging in can be disabled via a configuration option in *version 5.7.1* and later.	
1.5	When logging in is disabled, the **Login** button disappears from the web page.	
1.6	When logging in is disabled, all user details are saved.	

1.7	When logging in is re-enabled, users who had previously signed in can log in again.	
Section 2 – Login screen		
2.1	When a user is logged out, the **Login** button appears in the top right-hand corner of every page in the interface.	You will need to list all those pages.
2.2	The appearance of the **Login** button is shown in *graphic 1*.	(*Not included here*)
2.3	Clicking on the **Login** button takes users to the login screen.	
2.4	The login screen contains five elements: 1. An email address textbox 2. A password textbox 3. A **Submit** button 4. A sign-up page link 5. A forgotten password page link	
2.5	The appearance of the login screen is shown in *graphic 2*.	(*Not included here*)
2.6	Clicking on the sign-up page link on the login screen takes the user to the **Signup** page.	(*Remember to keep your requirements obvious!*)
2.7	Clicking on the forgotten password link on the login screen takes the user to the forgotten password page.	
2.8	The email address textbox on the login screen validates the email.	See the standard requirements for email validity.
2.9	The password box on the login screen does not show the text the user enters.	
2.10	The maximum length of the email address field on the login screen is 320 characters.	

2.11	The maximum length of the password field on the login screen is 50 characters.	
2.12	If the user enters an incorrect email and password combination, an error message appears letting them know.	(*Wording not included here*)
2.13	If a user enters a valid email address and password combination, they are taken to the home page.	
2.14	When a user is logged in, the account menu is shown at the top right of every page.	Instead of the login button
2.15	The appearance of the account menu is shown in *graphic 3*.	(*Not included here*)
2.16	The account menu has a single option to log out.	
2.17	Pressing the **Log out** button keeps the user on the same page but logs them out.	
2.18	Other than displaying the account menu, logging in has no effect on the web pages.	Actually, doing something with the login is covered in *version 2* and is not covered here.
2.19	If a user who has already logged out logs out again, it has no effect.	For instance, if they use a stale webpage having logged out elsewhere
2.20	Pressing the **Log out** button once logs out every login session for that user.	Even if they were logged in using 10 different browsers, logging 1 out logs them all out.
2.21	A single user can log in up to 20 times using different tabs and browsers.	There's no limit in the code to how many times a user can be logged in; we will test up to 20.
Section 3 – Security		
3.1	If the user enters an incorrect email and password combination, the same error message always appears.	There aren't separate error messages if the email address is correct for privacy.

3.2	If the user enters an incorrect email and password combination, the error message appears for the same length of time regardless of whether the email address is valid.	So, attackers can't use the length of time of response to detect valid email addresses.
3.3	Users are limited to three attempts to log in with the same password per minute (using a rolling window).	
3.4	If a user attempts to log in more than three times in a minute, then an error message appears, letting them know.	(*Wording not included here*)
3.5	The login, sign-up, and password reset screens are protected against CSRF attacks.	
3.6	Email addresses are stored securely and are only available to employees who require access.	(*That needs to be specified; support needs to be able to look people up, the development team shouldn't generally have access to live data, and so on*)
3.7	Passwords are hashed and stored securely.	The algorithm should be specified.
3.8	Users are logged out after 10 minutes of inactivity.	If they visit a page after that, they would need to log in again.
Section 4 – Sign-up		
4.1	On the **Signup** page, there are five elements: 1. An email address field 2. A password field 3. A password confirmation field 4. A **Submit** button 5. A link back to the login page	
4.2	The maximum length of the email address field on the sign-up screen is 320 characters.	

4.3	The maximum length of the password field on the login screen is 50 characters.	
4.4	Passwords entered on the sign-up screen are validated for complexity.	(*Complexity requirements not included here*)
4.5	The email address field on the **Signup** page is validated.	See the standard requirements for email validity.
4.6	The password field does not show the text that the user has entered.	
4.7	If the user enters an invalid email address, then they are shown a warning when they submit the form.	(*Wording not included here*)
4.8	If the user enters an invalid password, then they are shown a warning when they submit the form.	(*Wording not included here*)
4.9	If the user enters passwords that don't match, then they are shown a warning when they submit the form.	(*Wording not included here*)
4.10	If the user enters a new email address that has never been used before, then an email is sent to that address to verify the account.	
4.11	If the user enters an email address that has been entered previously but not verified, then another email is sent to the address to be verified.	
4.12	No feedback should be given to the requester as to whether the email already exists.	To avoid leaking information about email addresses already in use on the system
4.13	If the user enters an email address that has been verified but doesn't have a password set up, then another email is sent to the user	

	with a new link to the password page.	
4.14	If the user enters an email address with a verified email address with a password set, then they are sent another email with a link to the page to reset their password.	
4.15	Verification emails are sent within 1 minute of being requested.	
4.16	The appearance verification emails are shown in *graphic 4*.	(*Not included here*)
4.17	When the user clicks a link in the verification email, they are sent to the login screen.	
4.18	When the user clicks a link in the verification email, the system records that that email address has been verified.	(*State which database column records that state*)
4.19	If a user clicks the link to the **Login** page on the **Signup** page, they are taken to the **Login** page.	
Section 5 – Password reset		
5.1	The password reset page includes three elements: 1. An email address field 2. A **Submit** button 3. A link to the sign-in page	
5.2	The email address field on the password reset page is validated.	
5.3	If the user enters an invalid email address, then they are shown a warning when they submit the form.	(*Wording not included here*)
5.4	Whether the email address is on our system or not, the user is shown the same message.	To avoid leaking information about which email addresses have been signed up to the system

5.5	If a user requests to reset the password of an email address that is on the system, then the application sends a password reset email.	
5.6	The appearance of the password reset email is shown in *graphic 5*.	(*Not included here*)
5.7	If a user requests to reset the password of an email address that is not on the system, then the application does nothing.	
5.8	The password reset email includes a secure link to let that user reset their password.	
5.9	If a user clicks the link to the **Login** page on the password reset page, they are taken to the **Signup** page to enter a new password.	
5.10	If a user enters an existing email address and a new password on the **Signup** page after following a reset password email link, then the password on their existing account is updated.	
5.11	There are no requirements for new passwords to be different from previous passwords for a user.	
5.12	If an email delivery fails, then no indication is given to a user, but an internal error is raised.	
5.13	Password reset links are valid for 24 hours.	
5.14	If a user clicks an out-of-date password link, then they are redirected to the login page.	

5.15	Users can send a maximum of three password reset emails in a 10-minute rolling window.	
5.16	Password reset emails are sent within 1 minute of being requested.	
Section 6 – Performance		
6.1	The application can support 50 sign-ups per second.	
6.2	The application can support 100 logins per second.	
6.3	The application can support logging in and then out within 1 second.	
6.4	The application can support logging out and then back in within 1 second.	
6.5	All application web pages load within 3 seconds.	
6.6	The application supports 1,000,000 user accounts.	
6.7	The application supports 1,000 users logged in simultaneously.	
Section 7 – Interoperability		
7.1	The login function works on the following web browsers: Chrome Firefox Edge Safari Opera	
7.2	The login function works on the last three versions of each web browser.	
7.3	The login function works on the following operating systems: Windows	

	macOS iOS Android	
7.4	The minimum width of all screens is 512 pixels.	
7.5	The maximum width of all screens is 4,096 pixels.	(*Note that this is limited; it's just what we'll test up to*)
Section 8 – Documentation		
8.1	All textboxes include tooltips.	(*Wording not included here*)
8.2	The written documentation describes how to sign up, log in, and reset your password.	
8,3	All existing screenshots in the documentation include the account menu.	
8.4	All existing instructional videos include the account menu.	
Section 9 – Events		
9.1	An event is raised every time a user logs in.	
9.2	An event is raised every time a user signs up.	
9.3	An event is raised every time a user logs out or when their login times out.	
9.4	The system displays how many users have signed up.	On an internal metrics page, which also needs to be specified
9.5	The system displays how many users have signed in during the last 7 days.	

Table A.1 – A functional specification for a simple login page