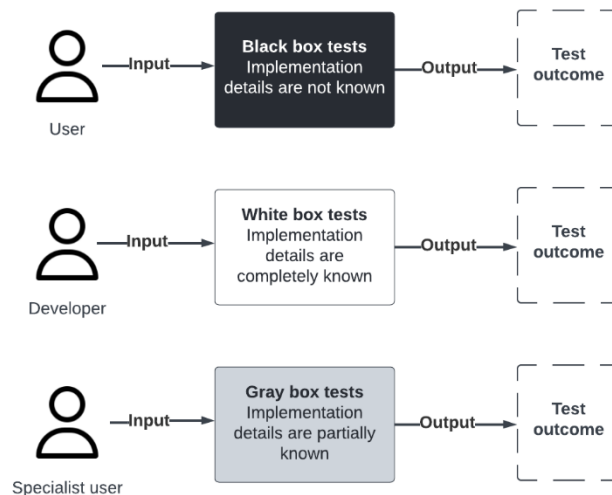
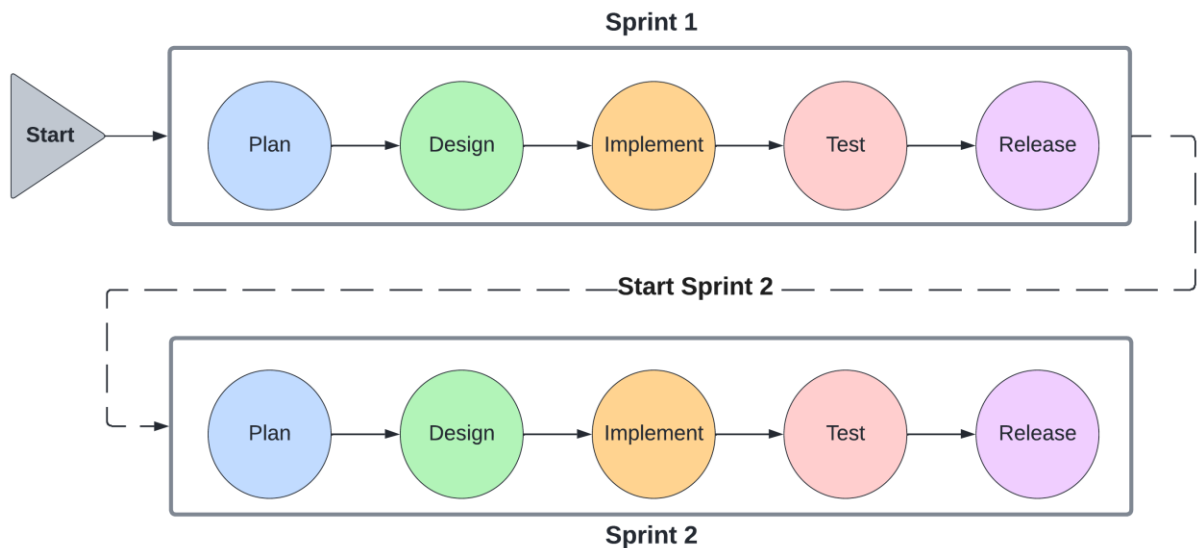
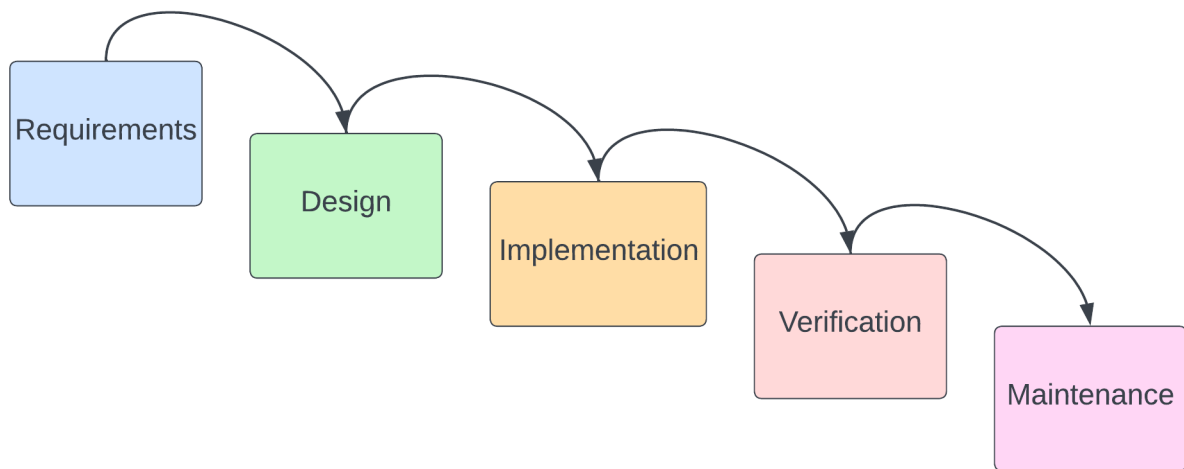
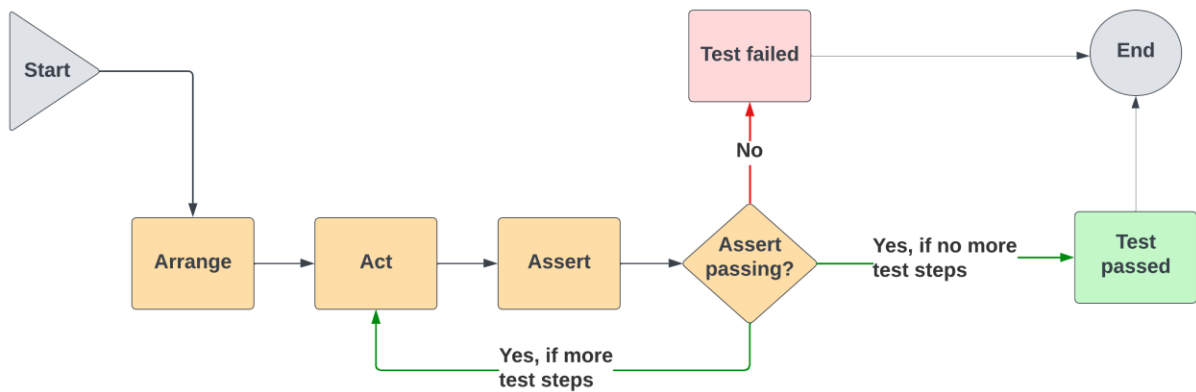
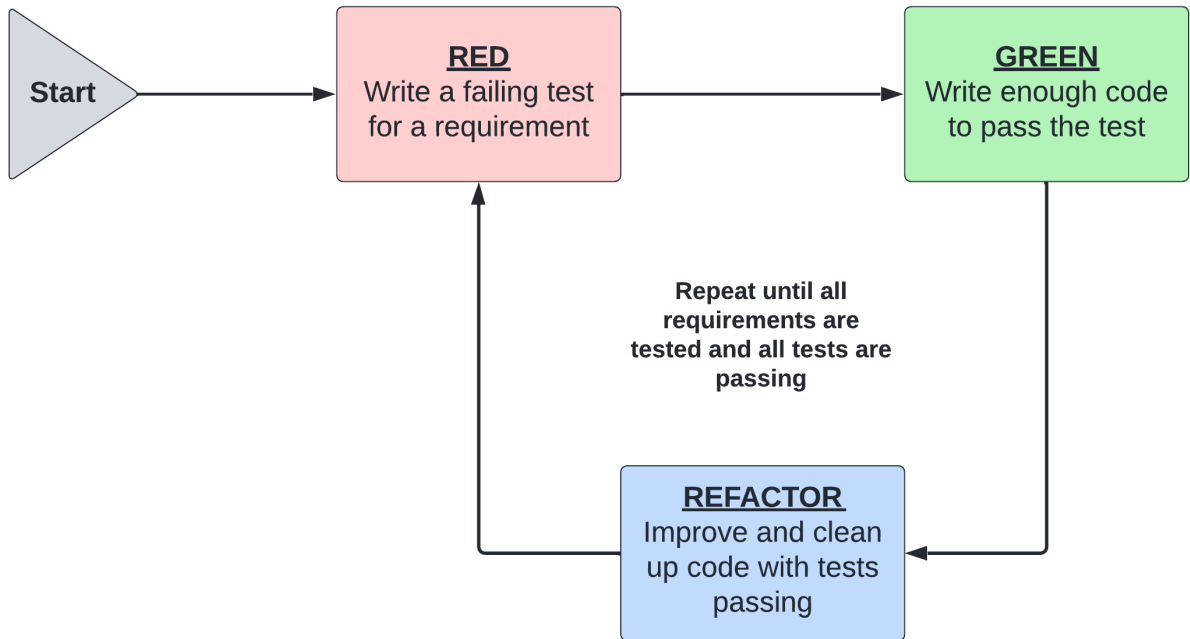
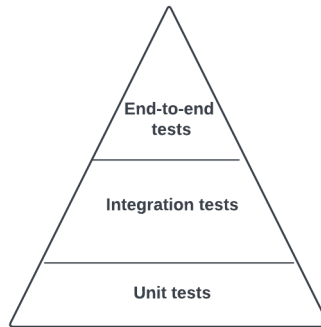
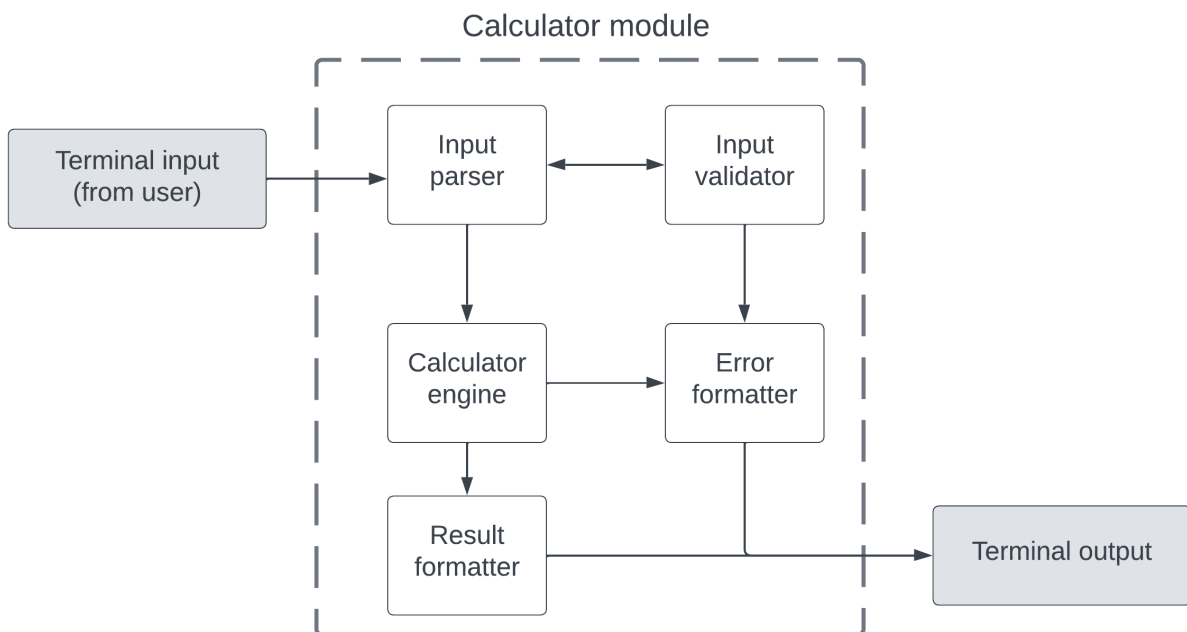
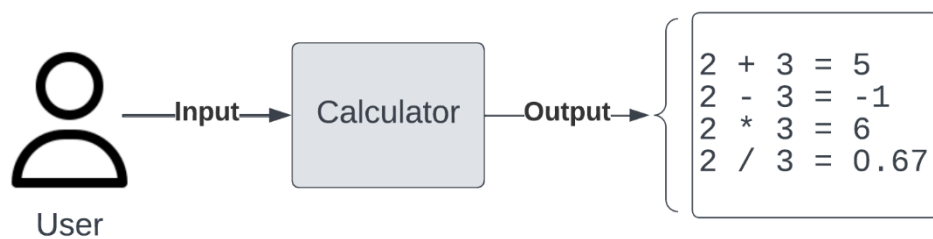


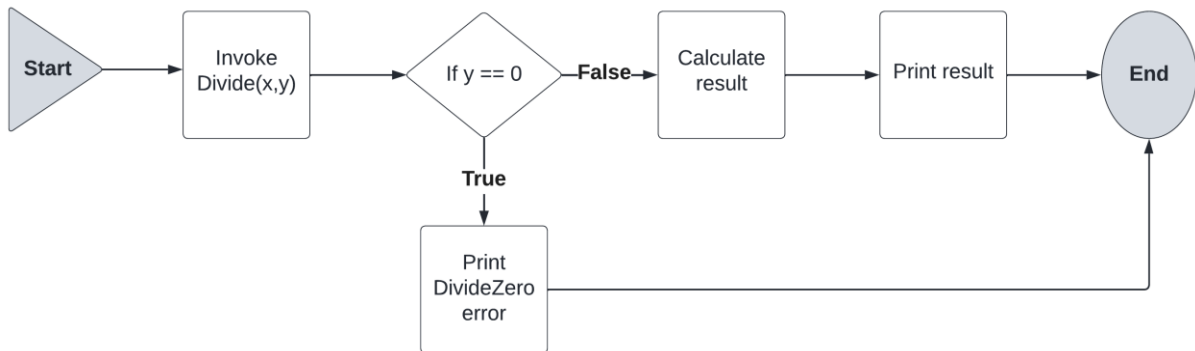
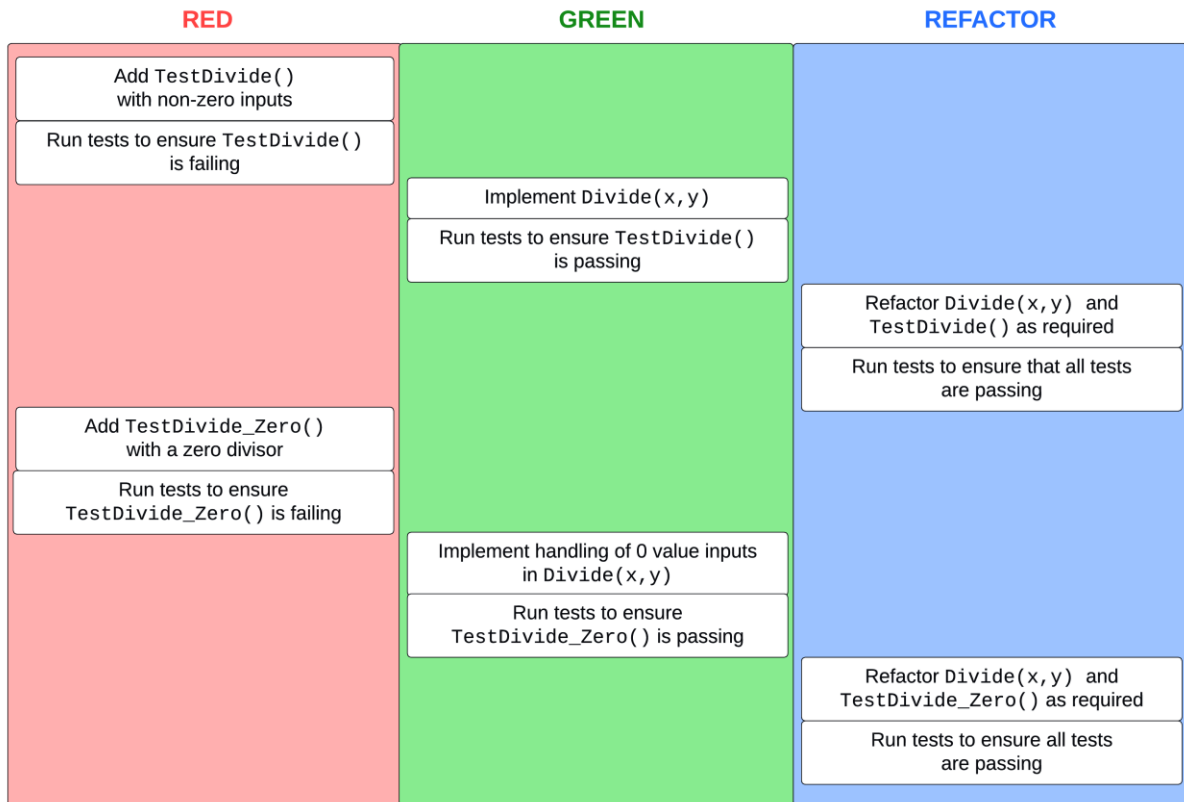
Chapter 1: Getting to Grips with Test-Driven Development





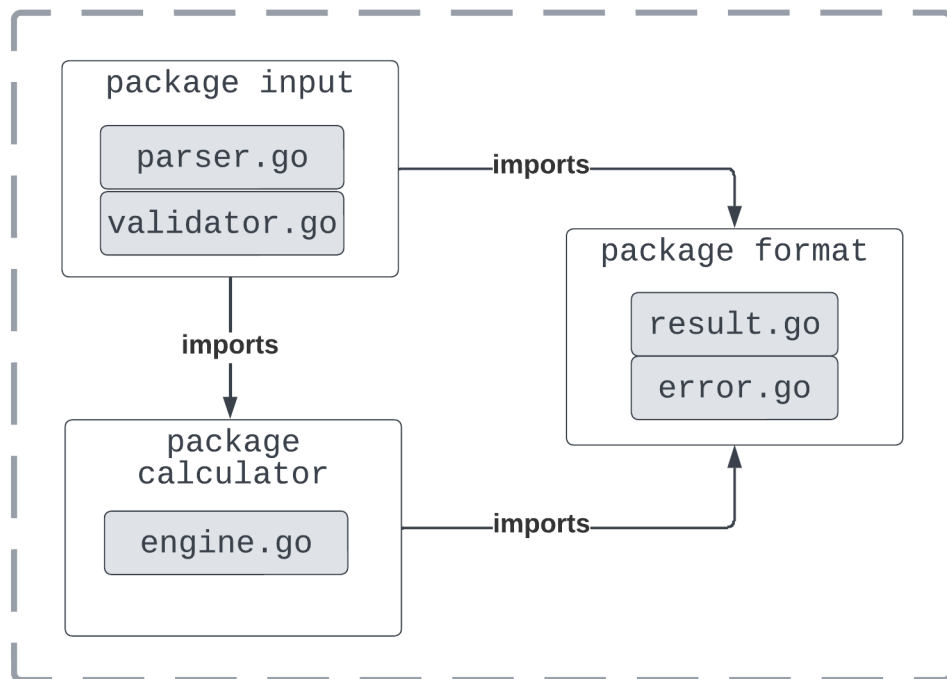
Pros	Cons
+ Allows the development and testing processes to be done at once + Developers analyse requirements in the beginning of each sprint + Increased confidence of code changes + Developers have ownership of their code quality	- Requires writing more code up-front - Requirements need to be elaborated in more detail - Test maintenance can be time consuming





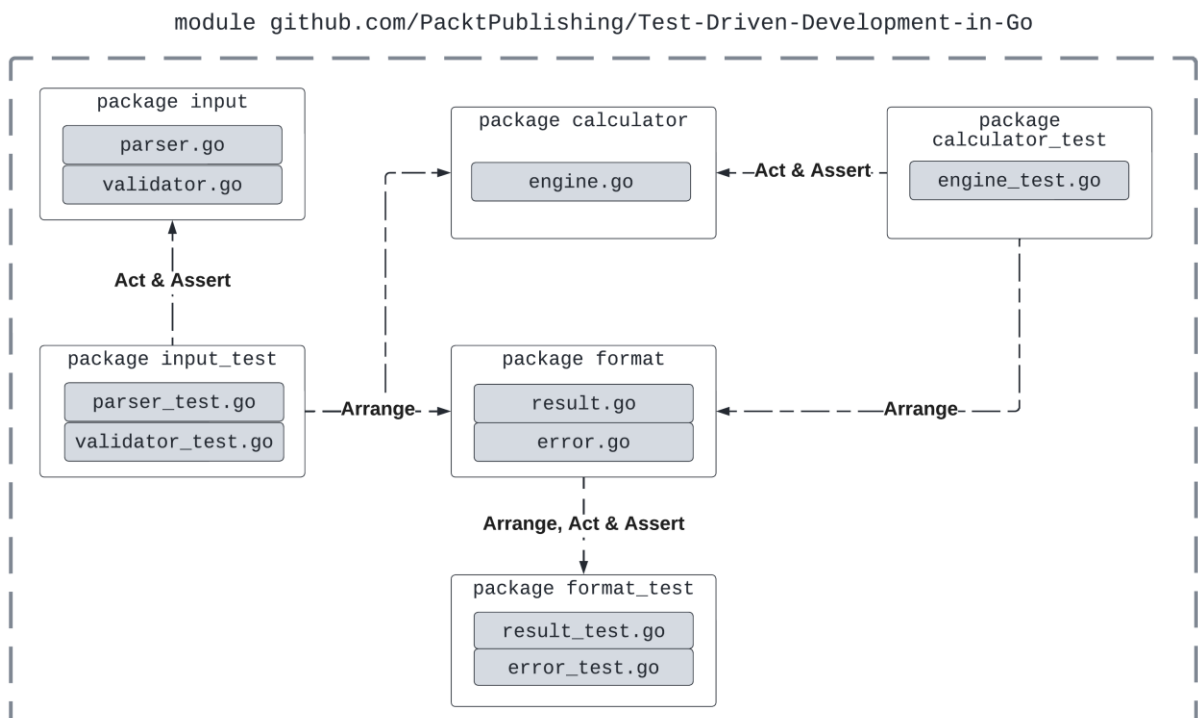
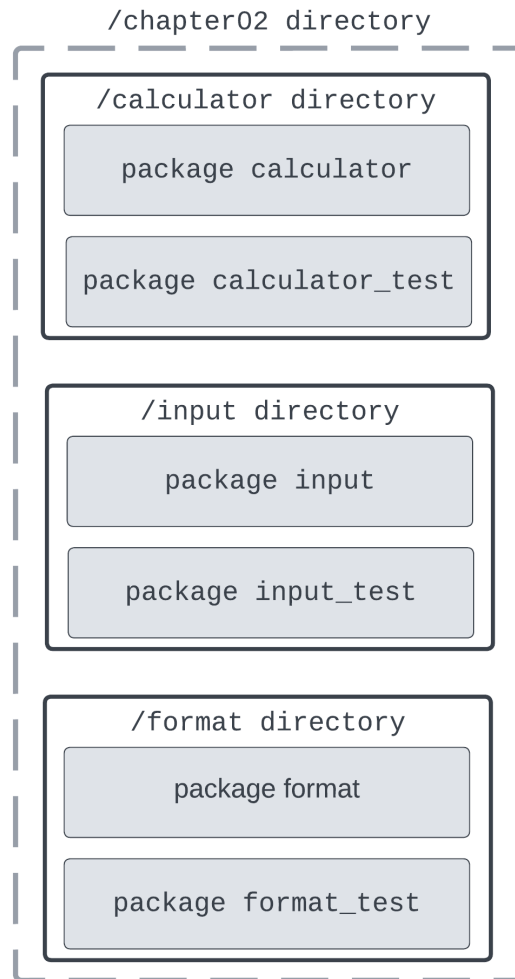
Chapter 2: Unit Testing Essentials

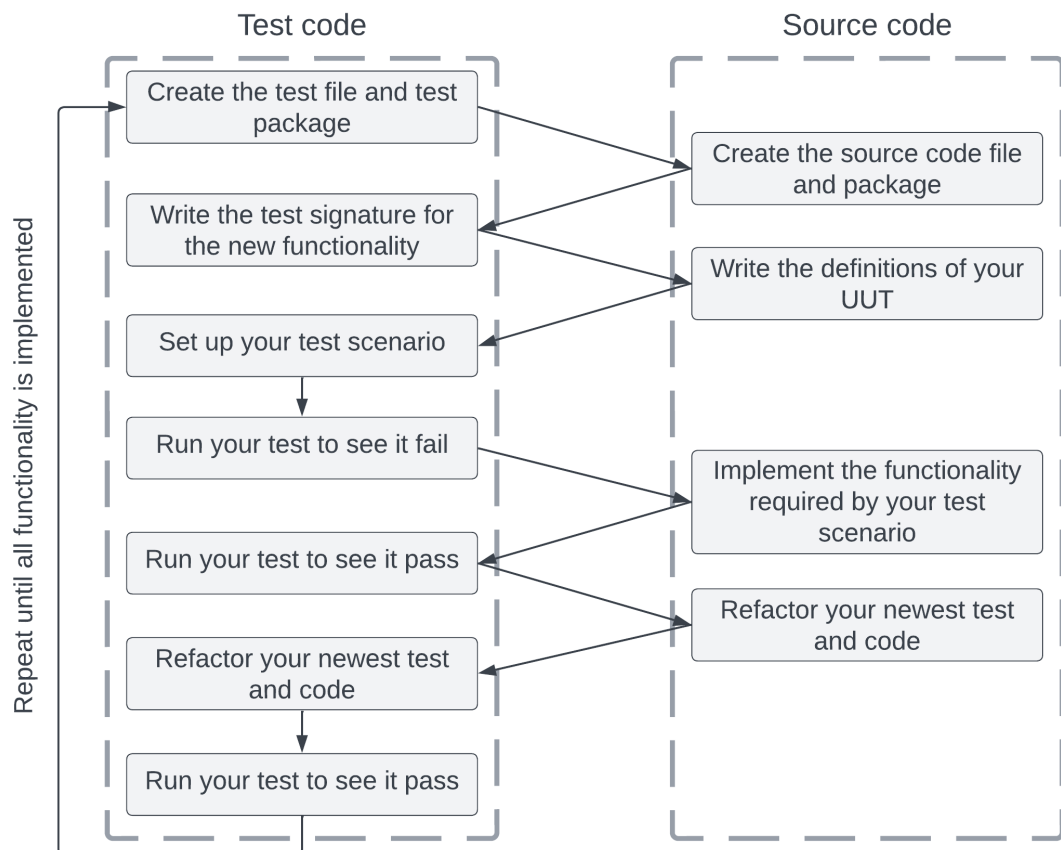
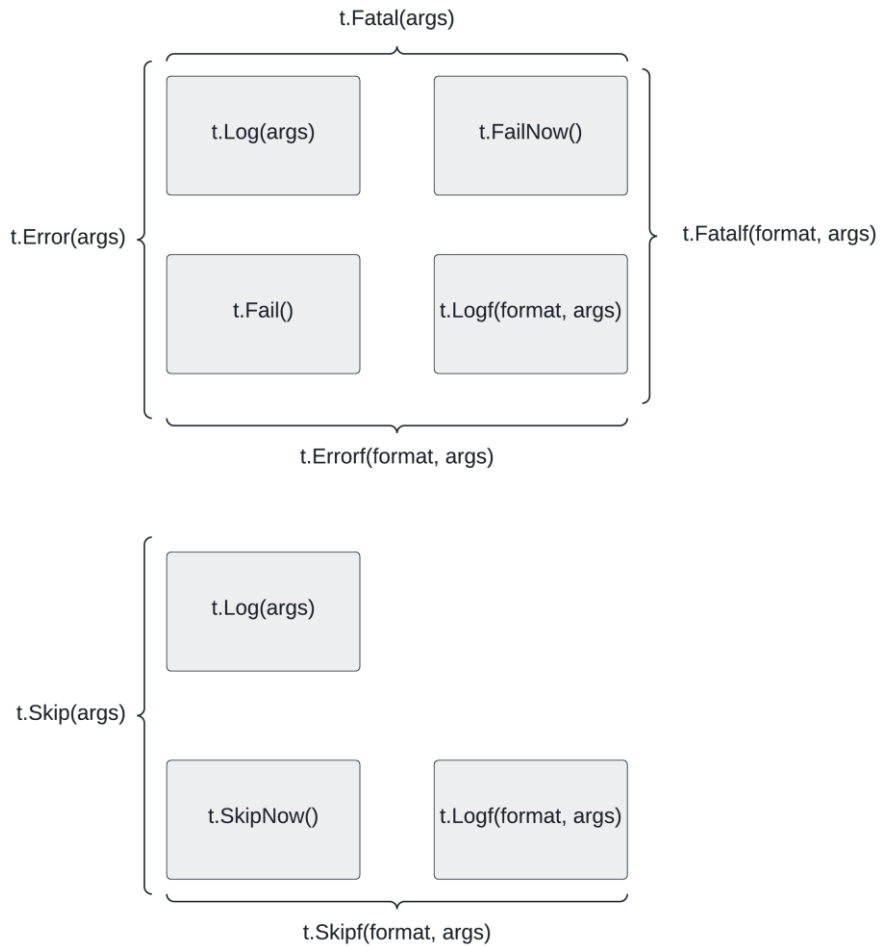
```
module github.com/PacktPublishing/  
Test-Driven-Development-in-Go
```

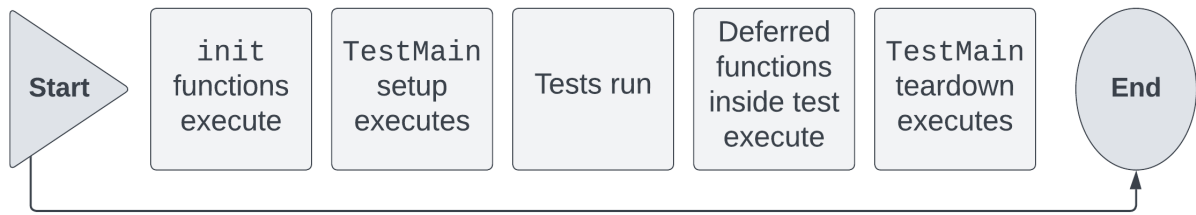


```
.  
├── LICENSE  
├── README.md  
├── chapter02  
│   ├── calculator  
│   │   ├── engine.go  
│   │   └── engine_test.go  
│   ├── format  
│   │   ├── error.go  
│   │   ├── error_test.go  
│   │   ├── result.go  
│   │   └── result_test.go  
│   ├── input  
│   │   ├── parser.go  
│   │   ├── parser_test.go  
│   │   ├── validator.go  
│   │   └── validator_test.go  
│   └── main.go  
└── go.mod
```

4 directories, 14 files



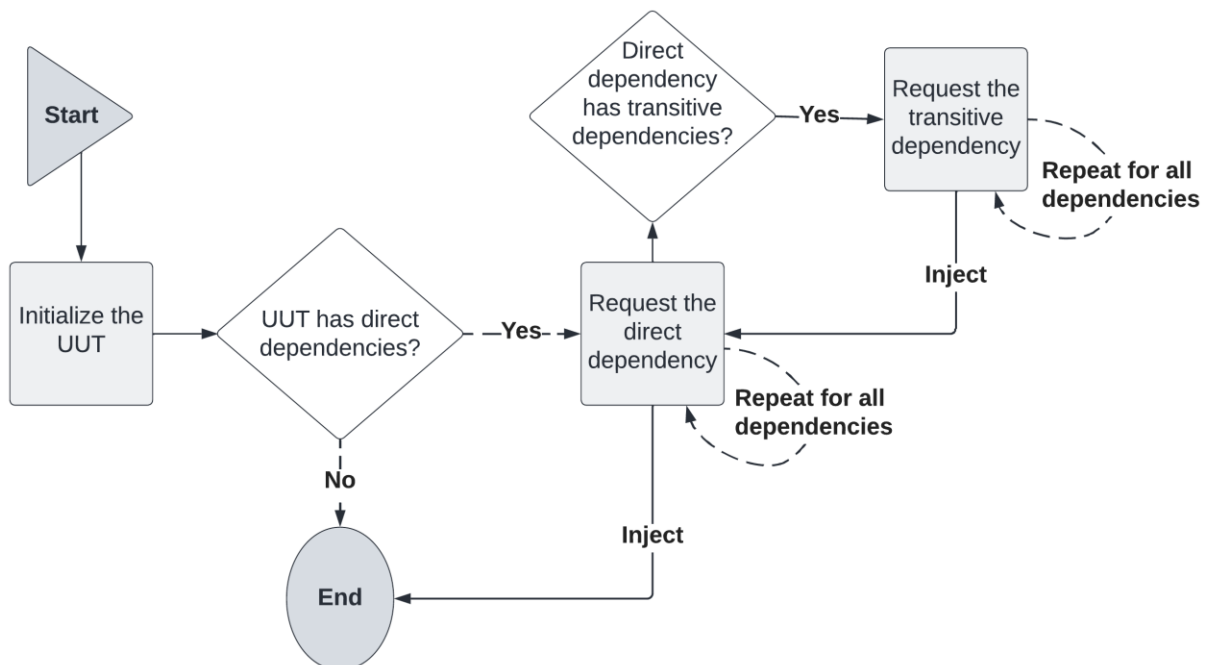
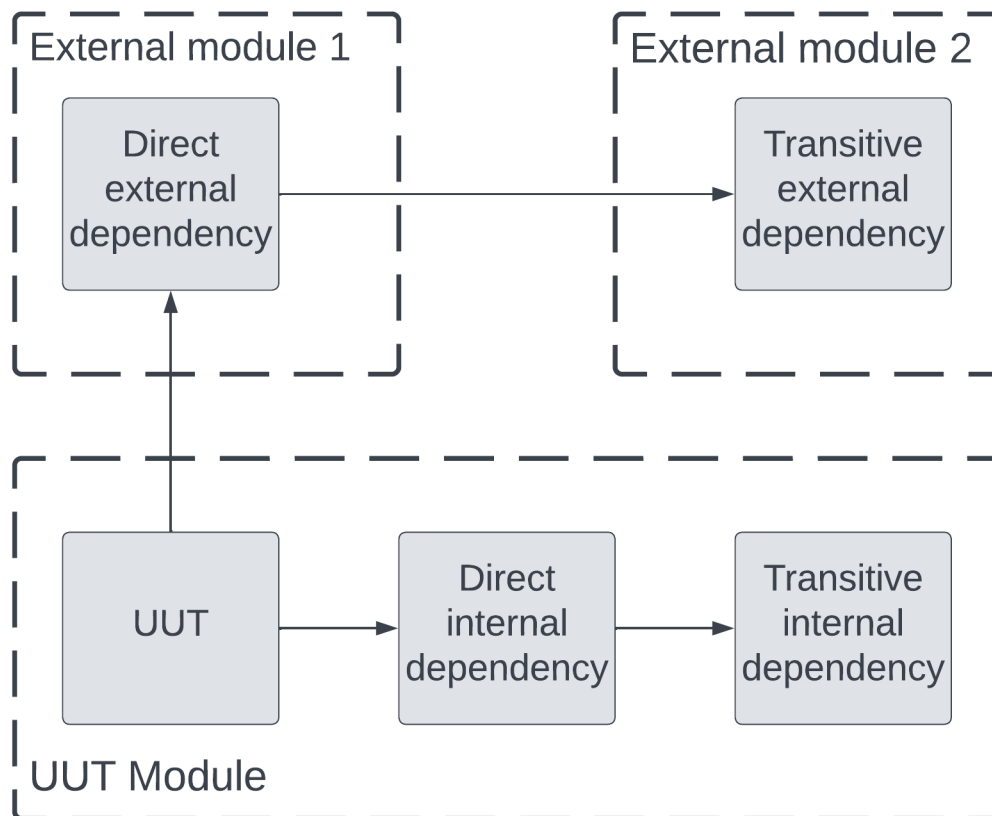




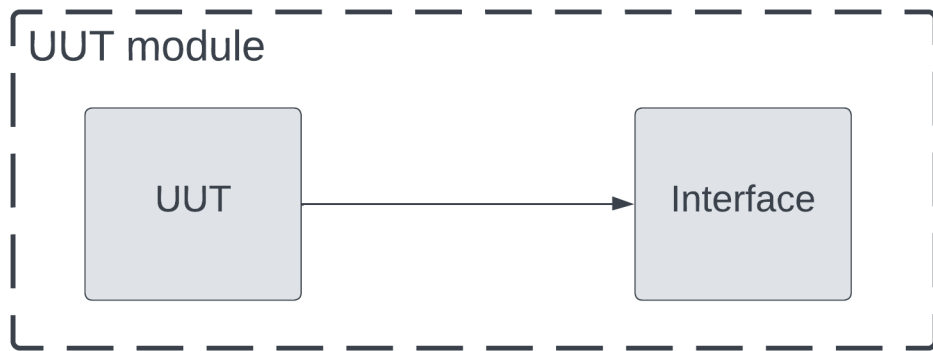
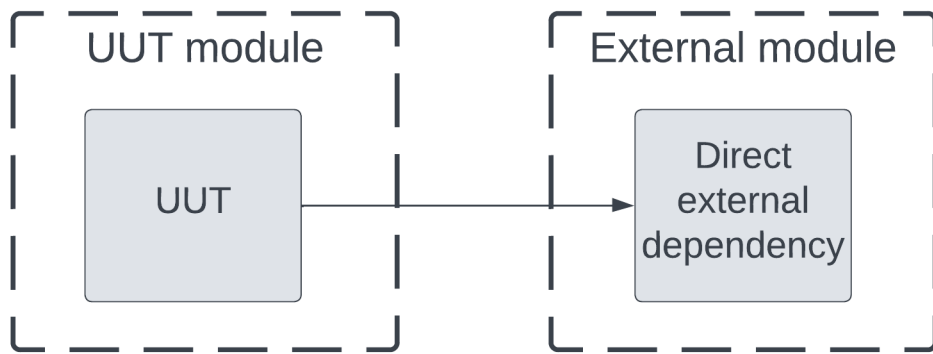
github.com/PacktPublishing/Test-Driven-Development-in-Go/chapter02/calculator/engine.go (100.0%) ▾ not tracked not covered covered

```
package calculator
type Engine struct{}
func (e *Engine) Add(x, y float64) float64 {
    return x + y
}
// ... other methods
```

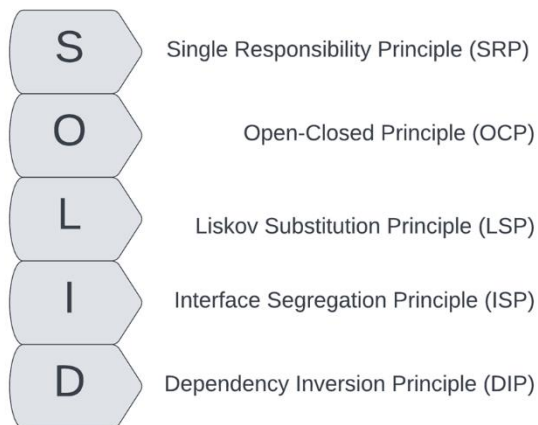
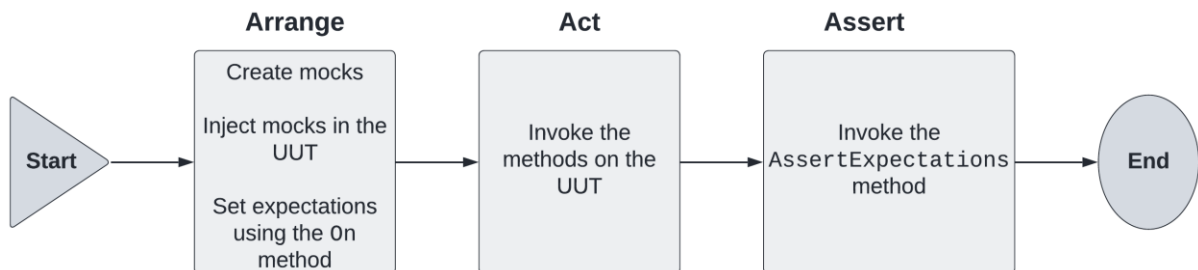
Chapter 3: Mocking and Assertion Frameworks



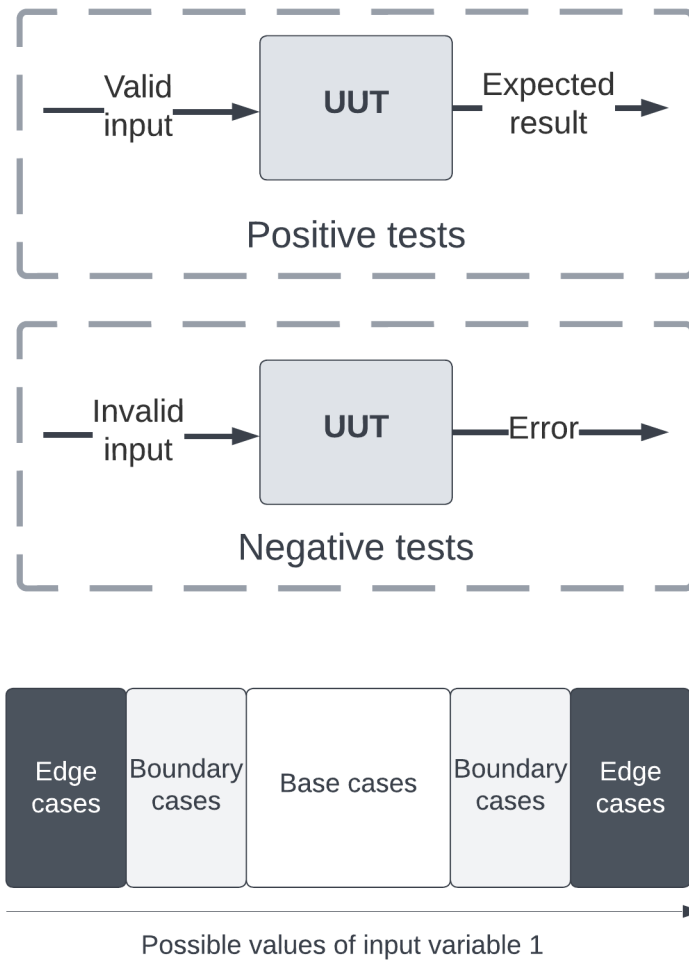
Tightly coupled struct dependencies

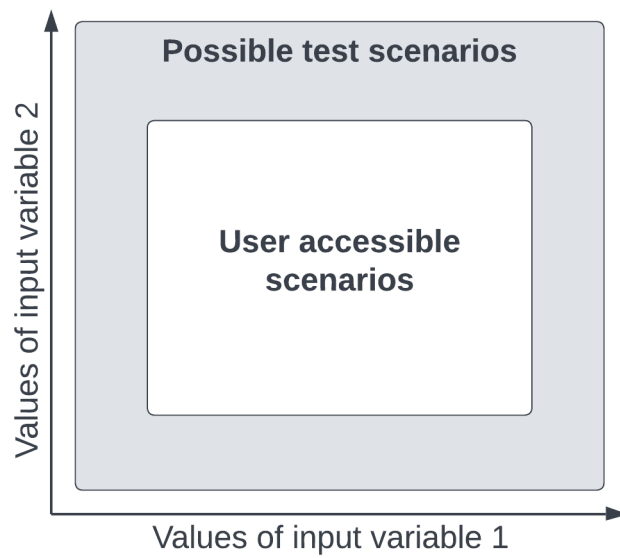
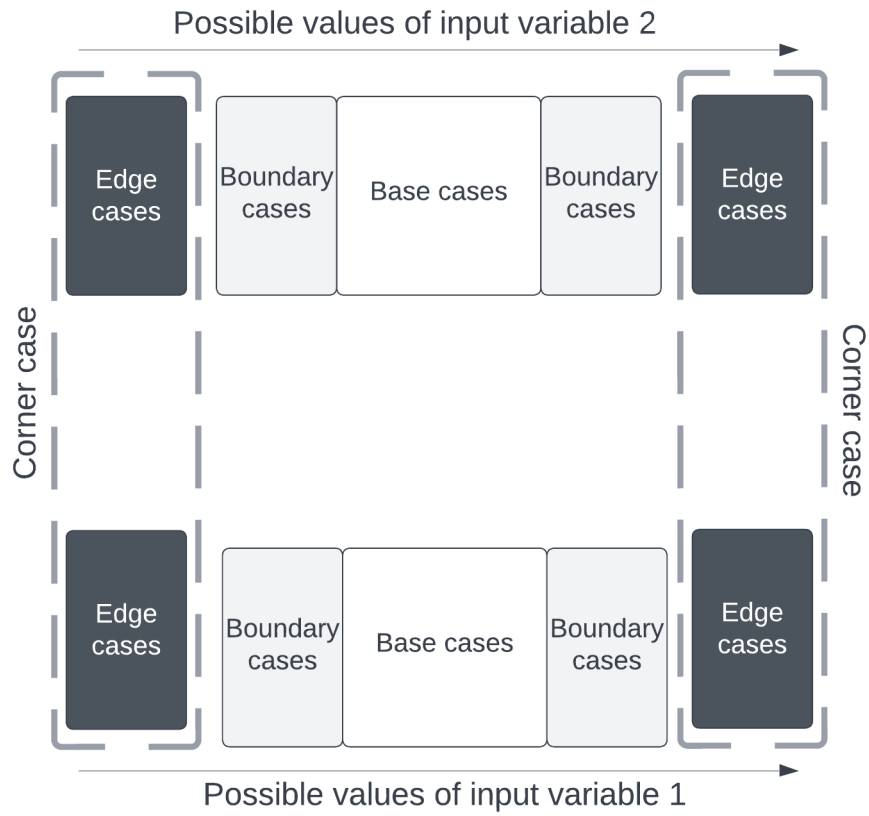


Loosely coupled interface dependencies

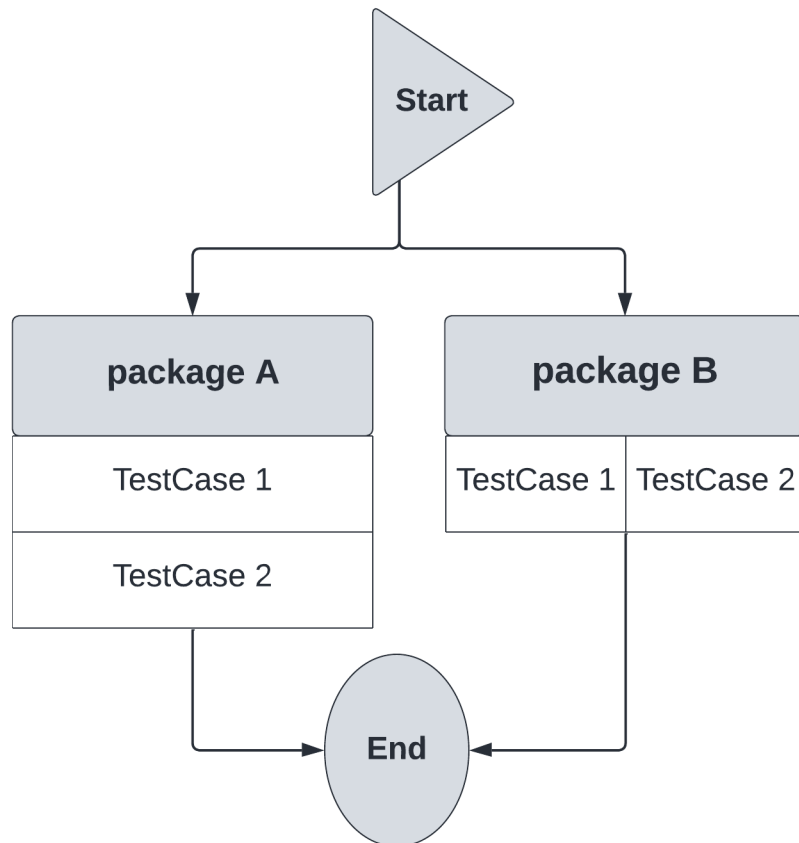
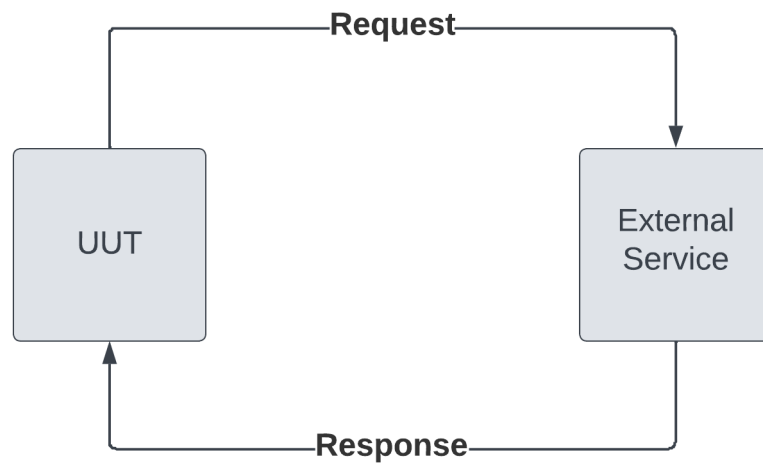


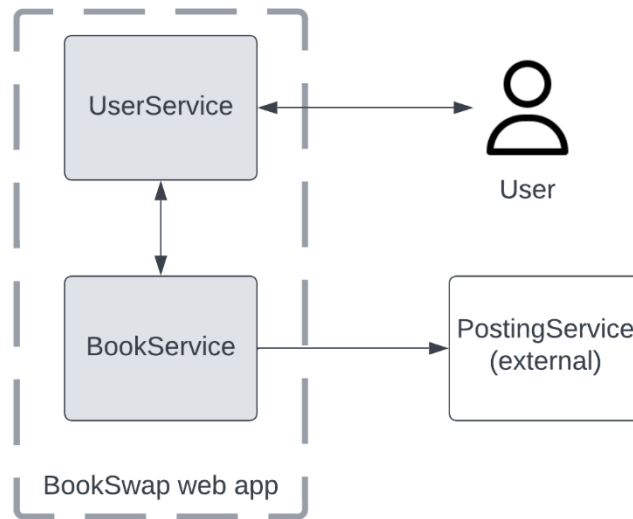
Chapter 4: Building Efficient Test Suites





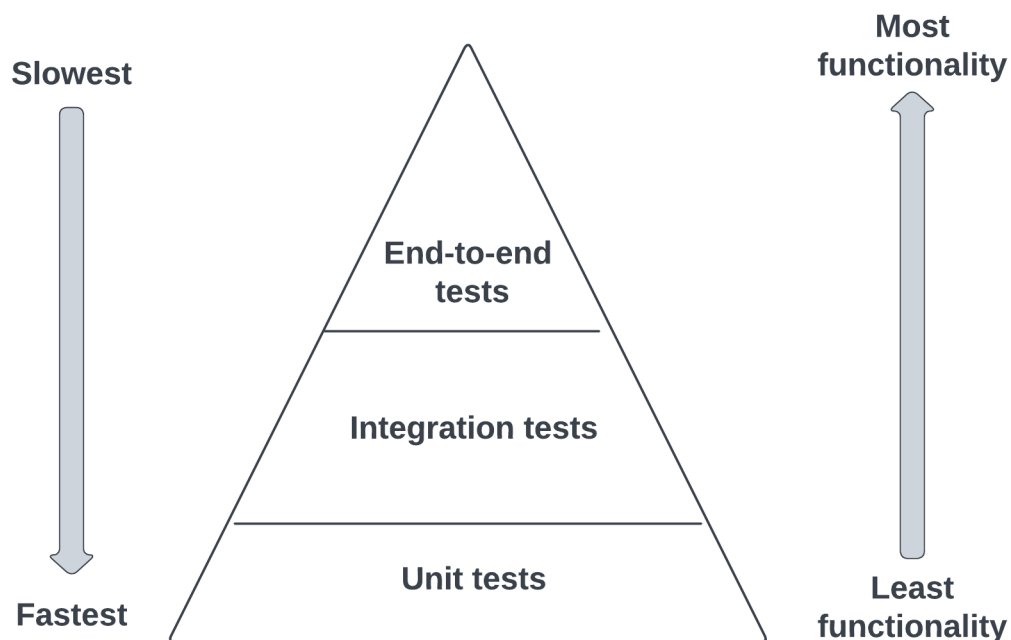
String type	Numeric type	Custom struct type	Collection types
• Empty string • Long string • Special characters • Multi-line string	• Zero value • Min/Max value according to numeric type • Positive and negative values	• Zero value • Nil value (if referenced by pointer) • Combinations of set fields	• Zero elements • One element • Nil value • Duplicate elements • Large amount of elements



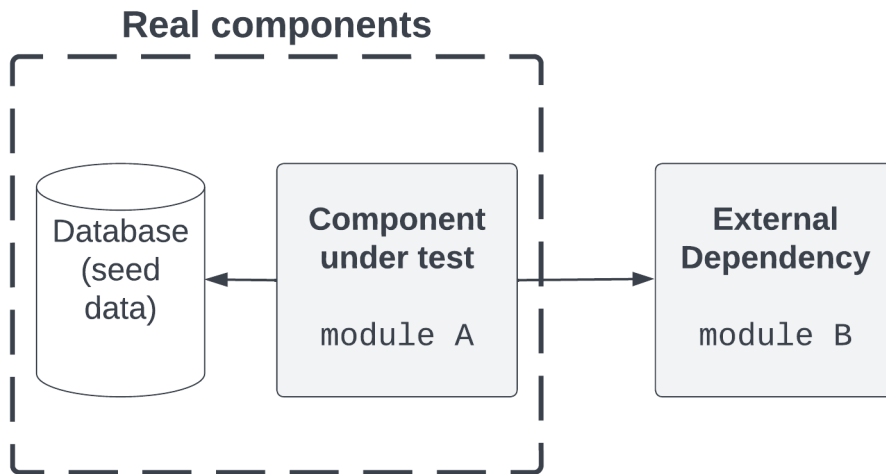


Chapter 5: Performing End-to-End Testing

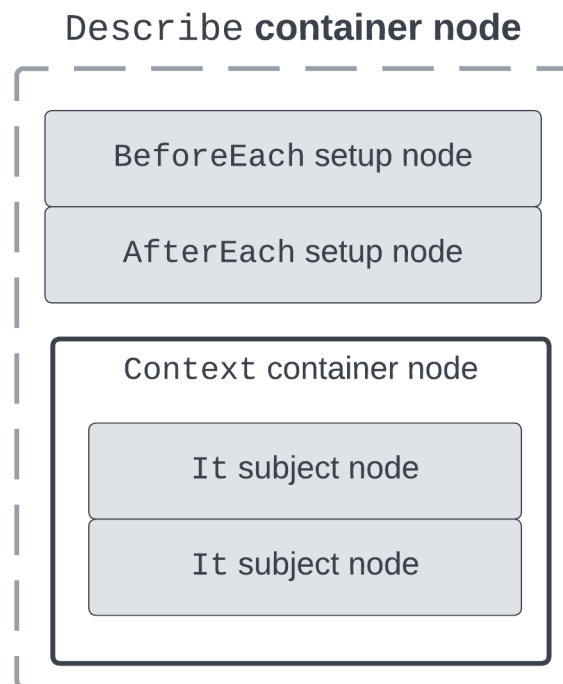
Advantages	Disadvantages
+ Support refactoring + Early bug detection + Easier debugging + Better code design + Documentation alongside implementation	- Increase the amount of code - Increased refactor effort - Difficult to identify realistic scenarios - Difficulties testing user interfaces (UI)

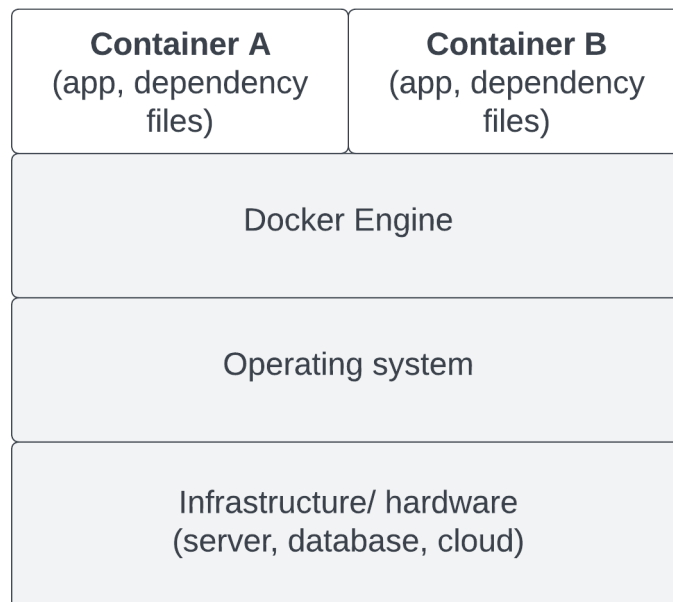
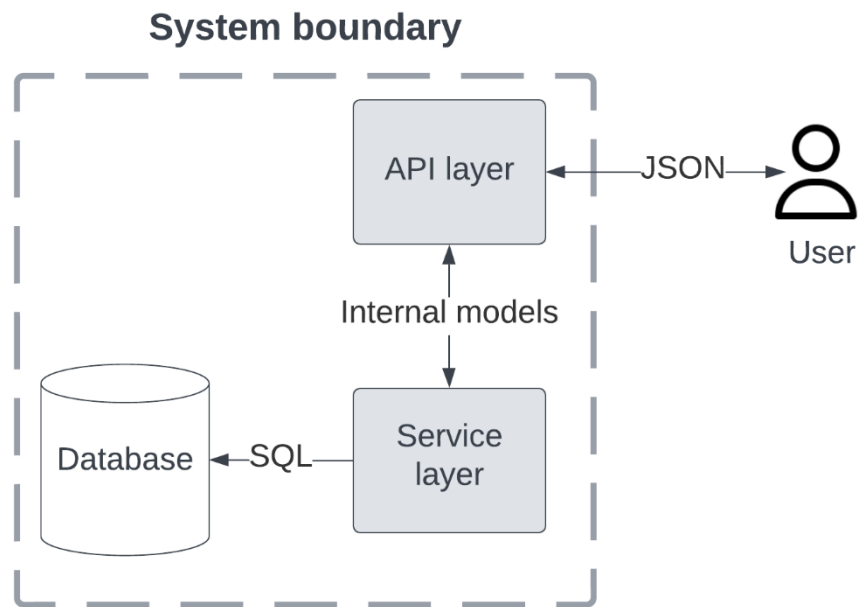


Integration tests	End-to-end tests
+ Performed early in the development process, shortening feedback cycle + Faster and cheap to run + Test integration with external and internal modules	- Performed at the end of the development process, when the product is complete - Slow and possibly expensive - Test user flow and experience



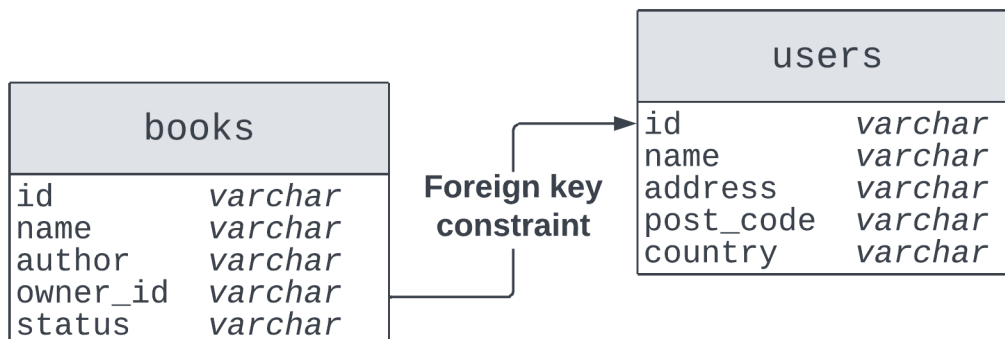
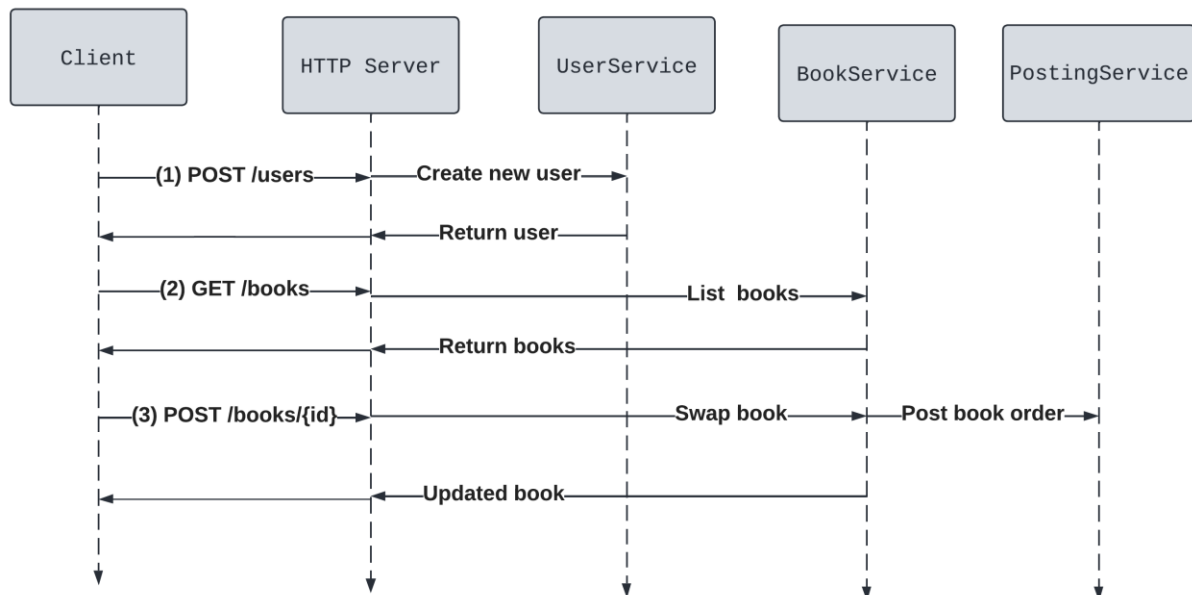
Advantages	Disadvantages
+ Single source of truth + Tests as documentation + Specific behaviors + Wider involvement	- Time consuming - Requires commitment - Dependent on good BDD practices





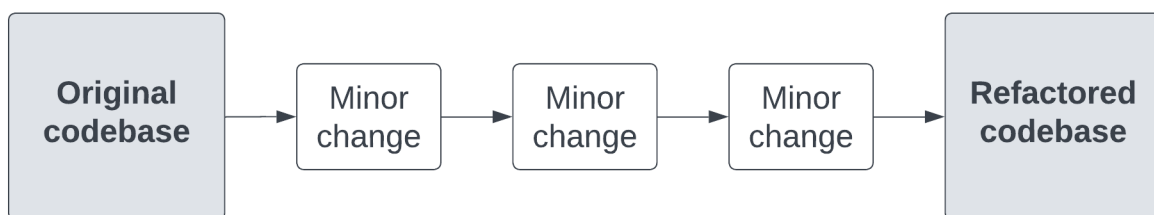
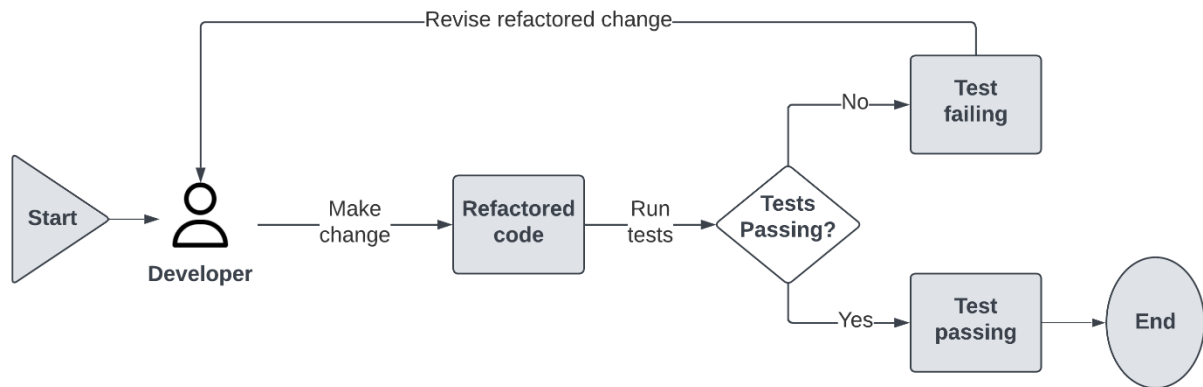
Chapter 6: End-to-End Testing the BookSwap Web Application

BookService	UserService	PostingService
• Creating books • Updating books • Listing books • Filtering by user ID	• Creating users • Updating users • Fetching user by ID and their owned books	• External service • Handling all the posting functionality

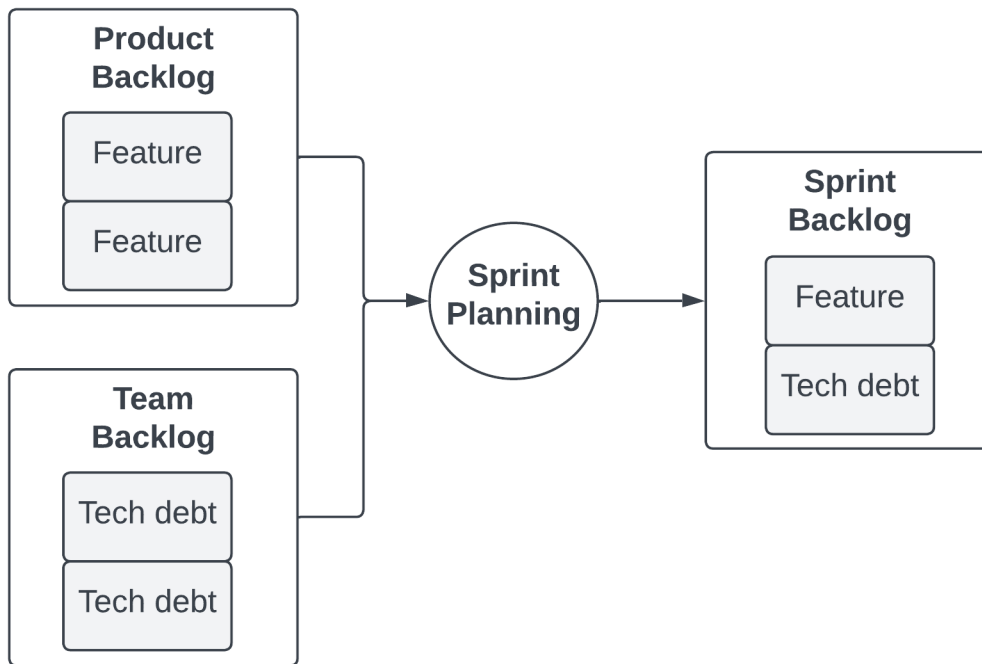


	Name	Request
1	aNewUserProfileIsCreated	GET /users/{id}
2	sentToTheUsersEndpoint	POST /users
3	theBookSwapAppIsUp	GET /
4	userDetails	

Chapter 7: Refactoring in Go

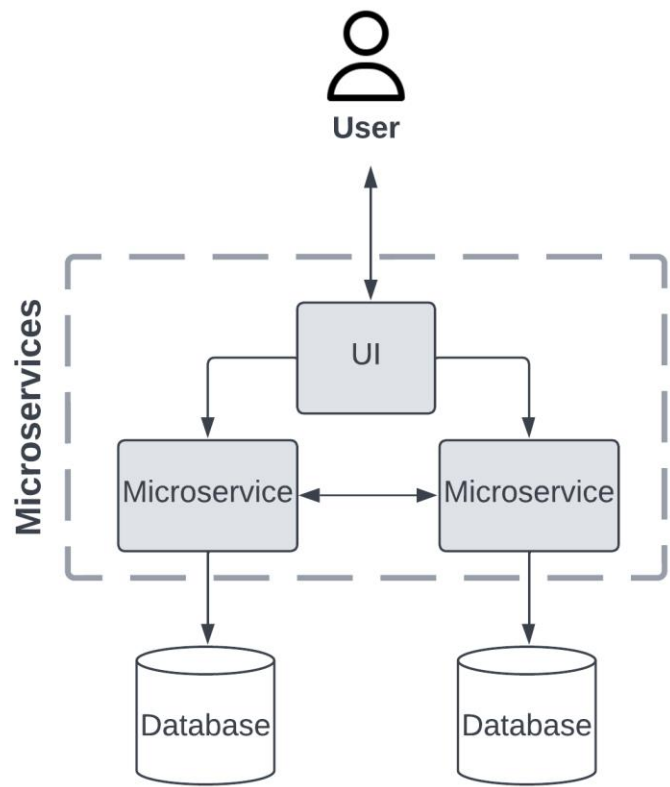
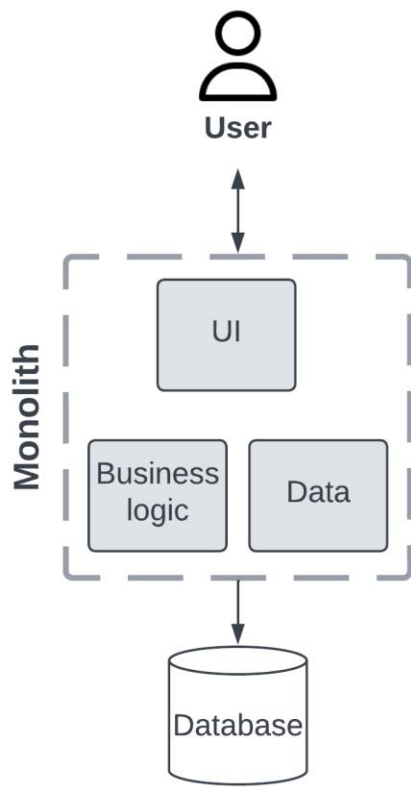


Technique Name	Main approach	Test changes?
Red-green-refactor	Three systematic steps that involves writing a Red test, writing the functionality required to make the test Green and then Refactoring the initial implementation.	Most likely
Extract	Extract an existing code fragment from its original function to a newly established function, which describes its purpose.	Not likely
Simplify	Simplify the logic of an existing function by consolidating its conditional expressions and adjusting the function parameters	Most likely
Inline	Remove redundant functions and their dependencies by removing their code and inlining them to its callers.	Not likely
Abstraction	Create new levels of abstraction such as interfaces to remove repetition and complex method signatures.	Most likely



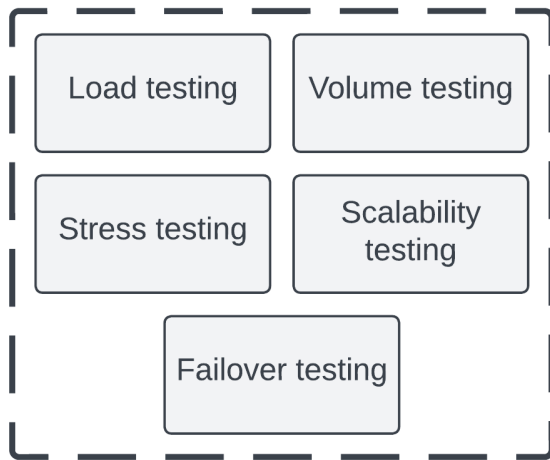
Pros	Cons
+ Easy to deploy + Lower cognitive burden + Simple to test and debug	- Harder to scale - Slower development speed - Errors can cause full outage

Pros	Cons
+ Flexible scaling + Easier to maintain + Smaller scope of tests	- Higher infrastructure costs - Higher organizational overhead - More difficult to debug

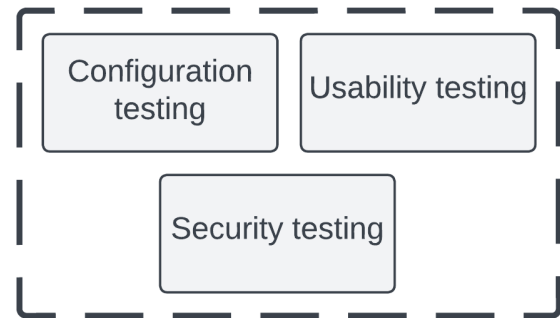


Chapter 8: Testing Microservice Architectures

Performance testing



Usability testing

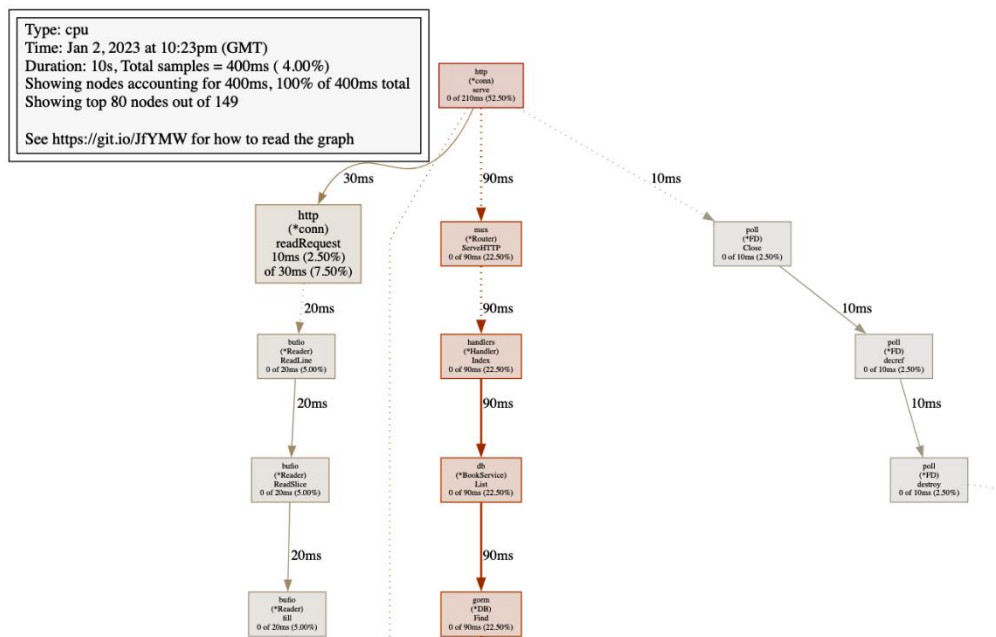


Is the system usable?

- Detect **stability** and **speed** issues
- Verify **error handling**
- Ensure it is capable of handling **expected user load**

Is the system scalable?

- Identify **bottlenecks** or potential improvements
- Verify the performance of the **individual parts** of the system
- Assess the **growth runway** that your system can support



Development complexity

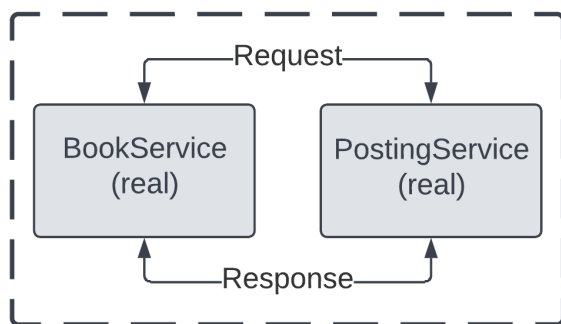
- Service design
- Data separation and structure
- Tests for each service

Deployment complexity

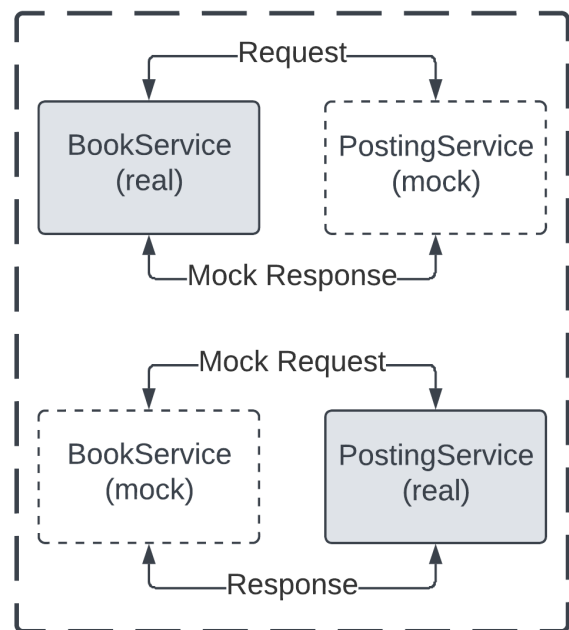
- Infrastructure maintenance
- Release strategies
- Deployment automation

- Service ownership
- Design and implementation standardisation
- Communication between teams

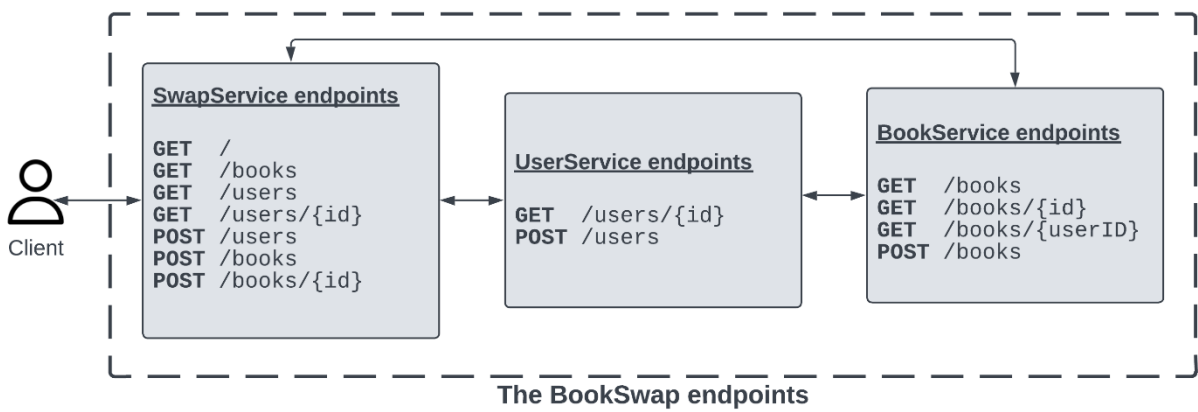
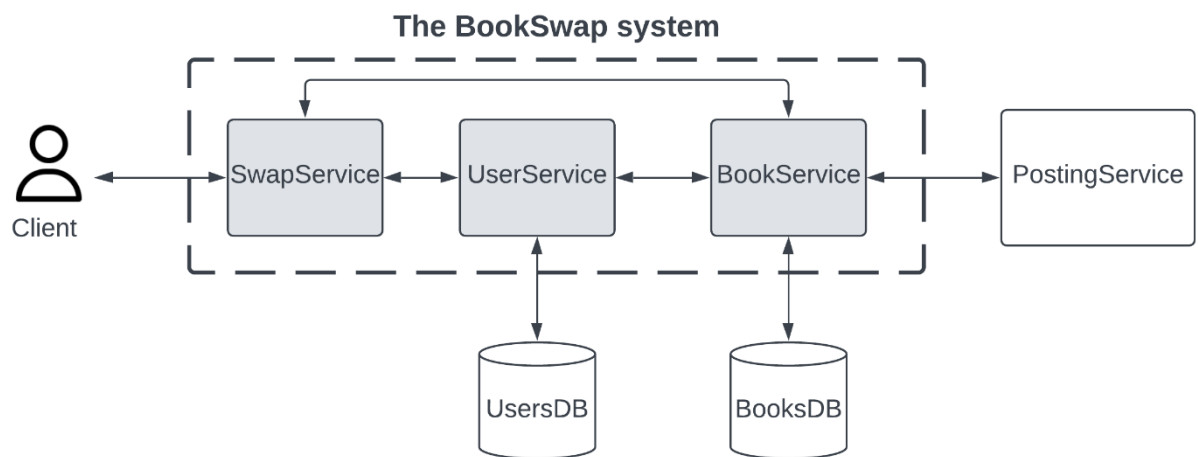
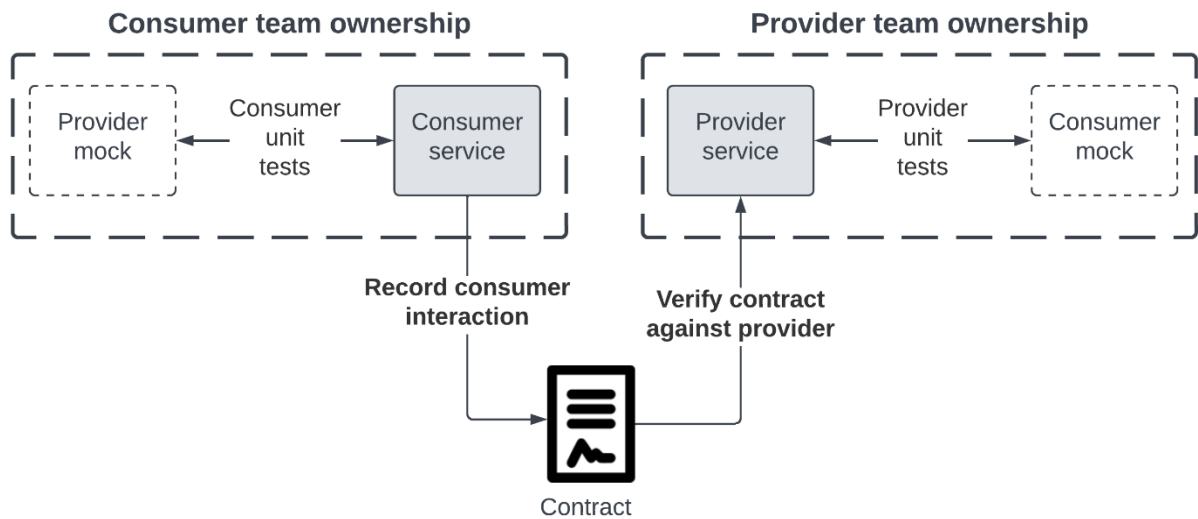
Organisational complexity



Option A: Integration test with real services

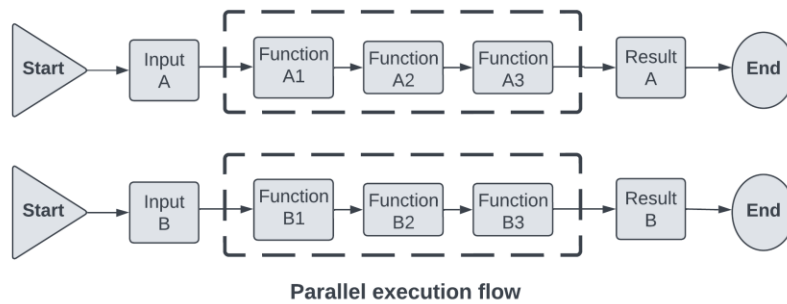
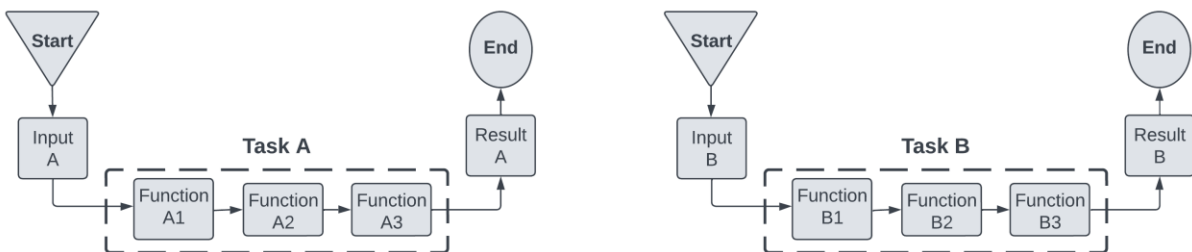
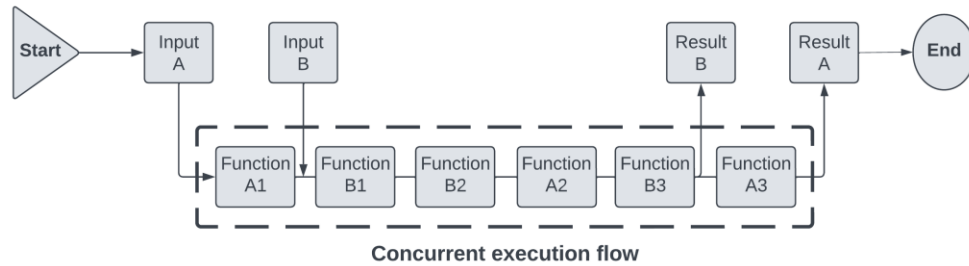
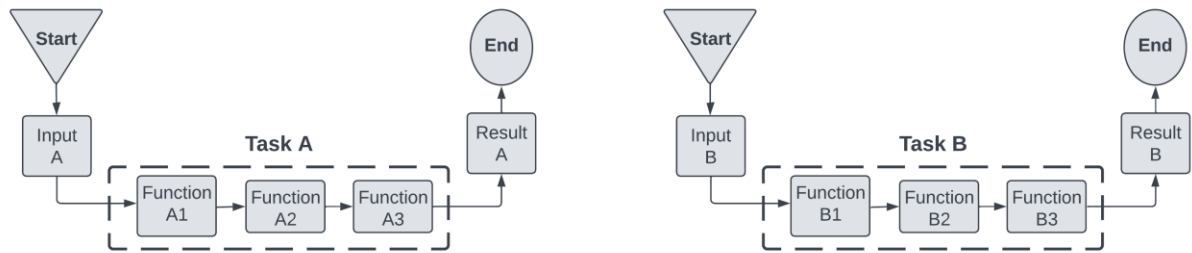


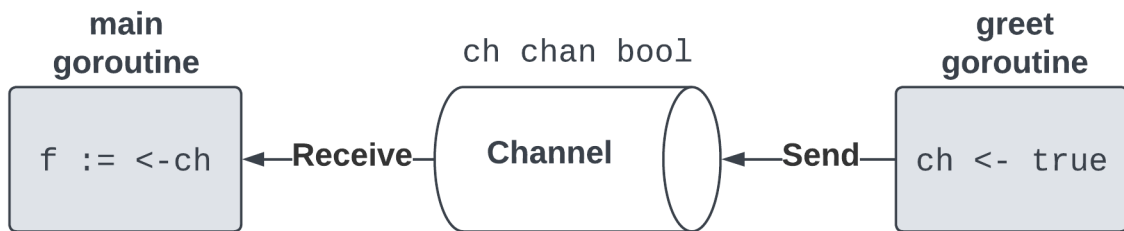
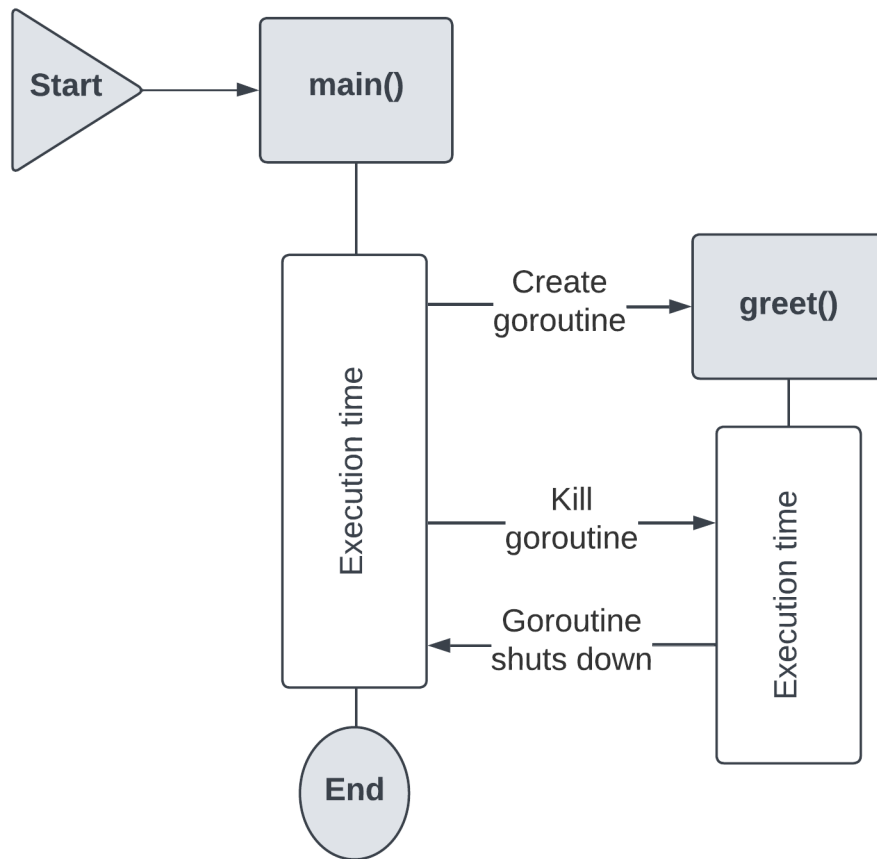
Option B: Integration test with mocks



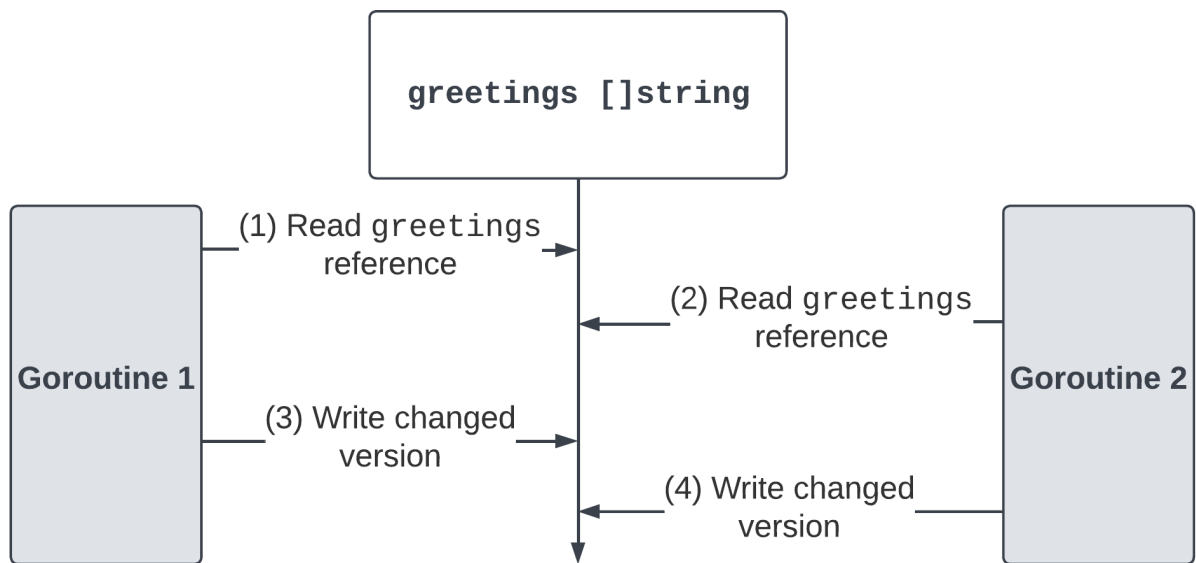
Consumer	Provider
Client code	SwapService
SwapService	BookService
SwapService	UserService
UserService	BookService
BookService	PostingService

Chapter 9: Challenges of Testing Concurrent Code

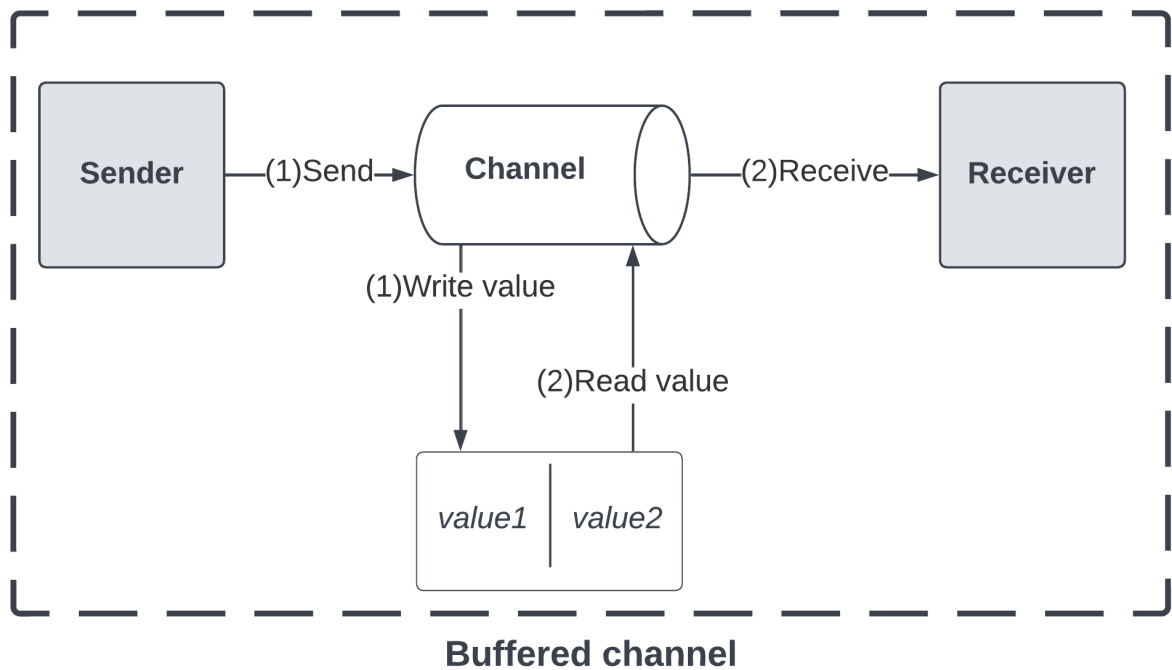
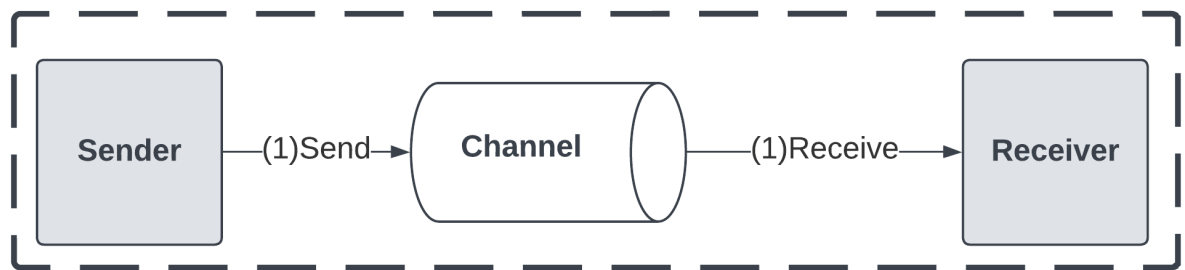


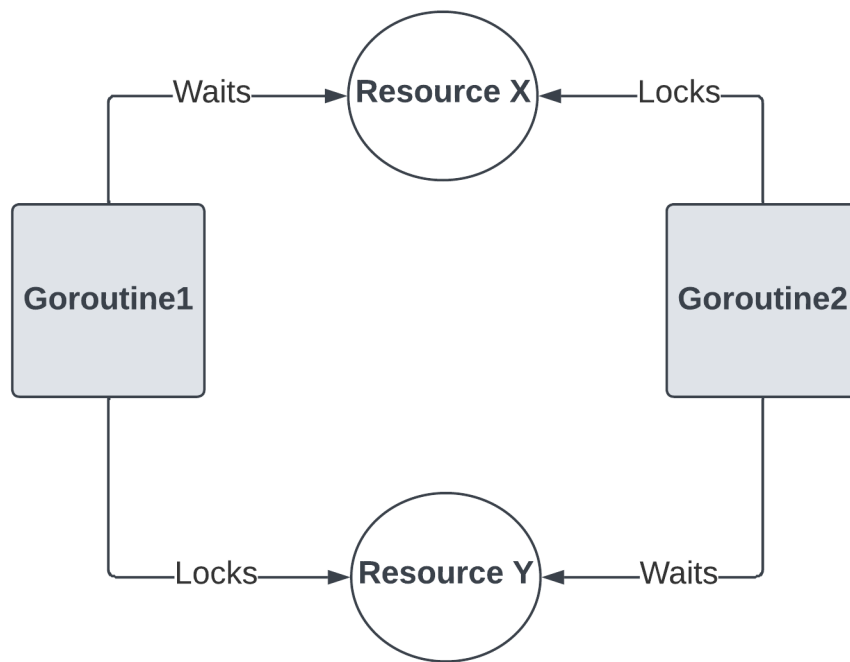


	Nil channels	Initialised channels	Closed channels
Send	Block	Block until a receiver arrives	Panic
Receive	Block	Block until a sender arrives	Complete immediately with zero value of data type
Close	Panic	Complete immediately	Panic



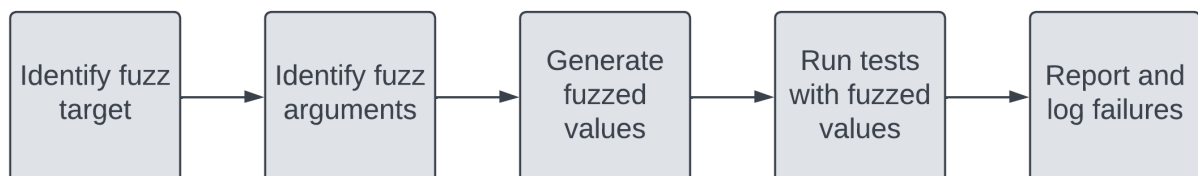
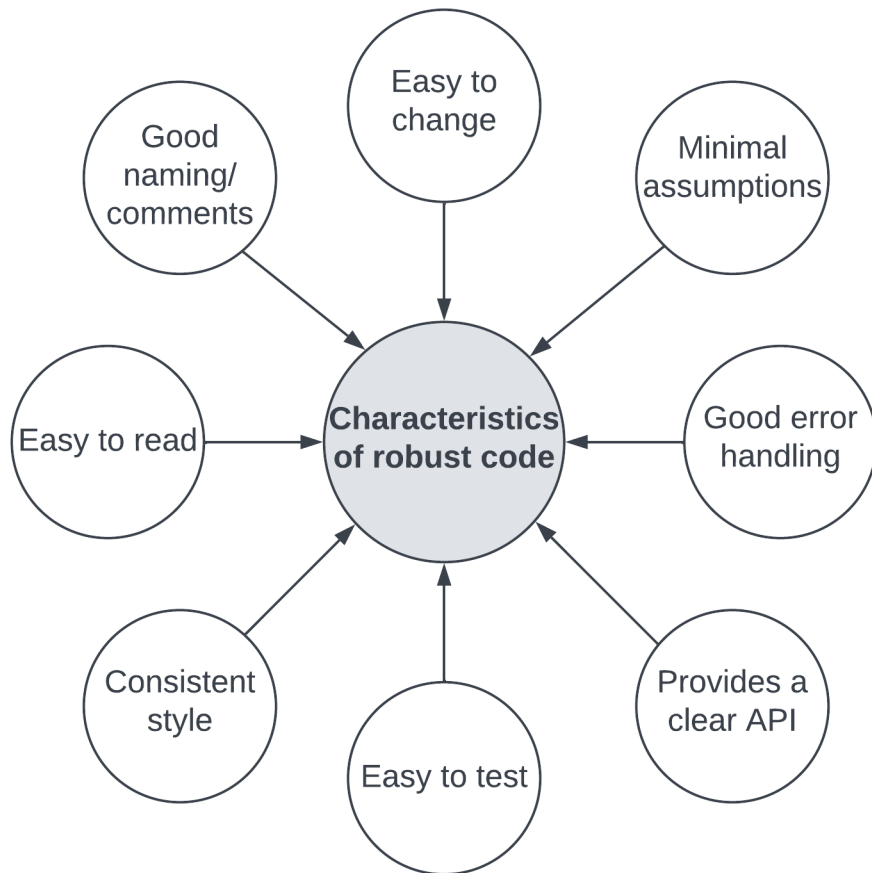
Unbuffered channel



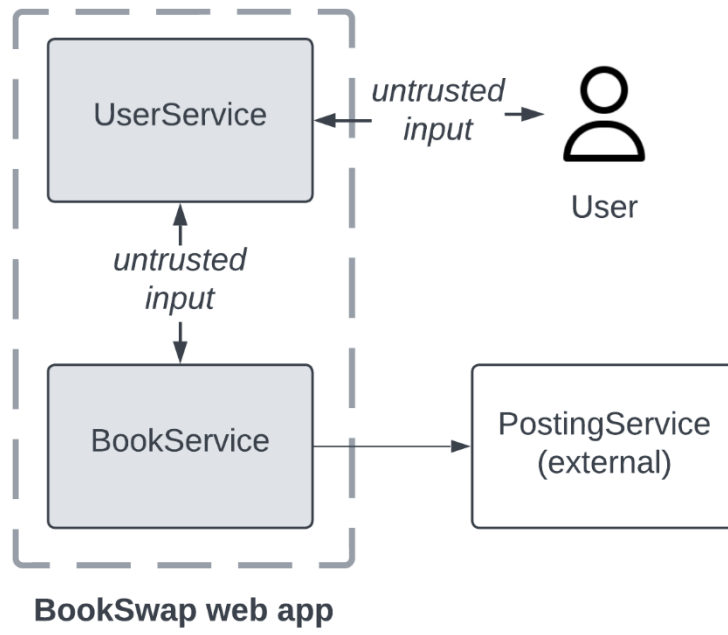


Rule	Description
Share values with channels.	Don't write results to shared variables, use channels instead.
Defer lock releases.	Immediately defer the call to the unlock method of any acquired lock.
Wait for all children to complete.	The goroutine which starts other goroutines should wait for them to complete before shutting down.

Chapter 10: Testing Edge Cases



Advantages	Disadvantages
+ Easy to use and implement	- Does not replace traditional testing
+ Can be used early in the development lifecycle	- Does not provide guarantees
+ Detects a wide variety of bugs	- Memory & CPU intensive



Chapter 11: Working with Generic

Generics	Interfaces
+ Specify type	+ Specify behavior
+ Built into the language	+ Defined by the application
+ Limited scope	+ Wide scope

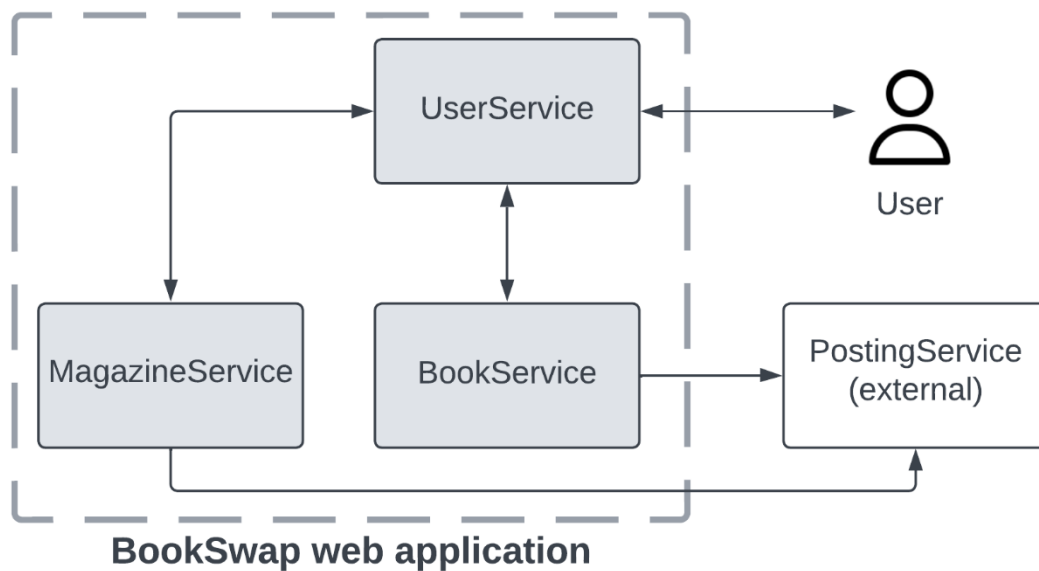
Type constraint

```
func sum[T int64 | float64](x, y T) T {  
    return x + y  
}
```

Type parameter

```
func main() {  
    fmt.Println(sum[int64](2, 3))  
}
```

Type argument



Development	Testing	Culture
+ Use TDD to write tests during development process	+ Write simple, isolated tests	+ Document customer requirements
+ Use interfaces to wrap dependencies	+ Use table-driven tests to cover a variety of scenarios	+ Mitigate for unexpected errors and outages
+ Refactor code to tackle technical debt	+ Write tests at every level of the application	+ Value and prioritize code quality
+ Write robust code which is able to handle a wide variety of inputs	+ Use Go's testing package	+ Document non-functional requirements
+ Adopt generics for easier code reuse	+ Use third-party testing libraries	