HTWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Fakultät Ingenieurwissenschaften

6010 - Praxisprojekt

Implementierung einer OCPP-Backend-Testumgebung für automatische Intergrationstests einer DC High Power Ladestationen der Plattform Sicharge D

Author Ivan Agibalov
Betreuer Prof. Dr.-Ing. Andreas Pretschner

28. Juni 2022

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Introduction

SomeIntroducation

1.1 OOP Design Patterns

some Introduction

1.1.1 Observer

Das OOP Design Pattern **Observer** ermöglicht dynamische Verbindungen zwischen den einzelnen Objecten im Programm, um über die geschehenen Ereignisse im Programm alle Interessenten zu informieren.

Das Pattern besteht aus 2 Teilen: **Publisher** und **Observers** oder **Subscribers**

Subscribers können bestimmte Events des **Publishers** abonnieren und deabonnieren. Der **Publisher** informiert alle auf das geschehene Event abonnierten **Subscribers**, bzw. wenn es auftritt. Den **Subscribers** kann im Falle des Eintretens des Events ein gewisses Verhalten definiert werden.

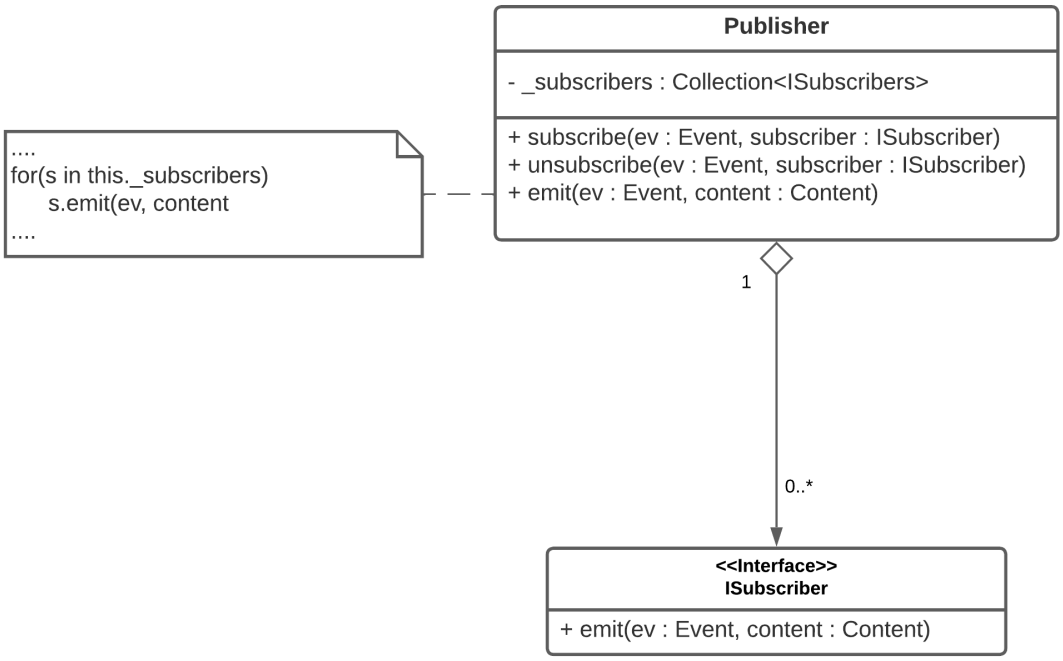


Abbildung 1: Klassendiagrammm Observer

Source: Eigene Quelle

1.1.2 Proxy

Das OOP Design Pattern **Proxy** ermöglicht die Aufrufe von bestimmten Objekten zu empfangen und ein gewisses Verhalten vor sowie nach dem eigentlichten Aufruf zu definieren.

1.1.3 TemplateMethod

Das OOP Design Pattern **Template Method** ermöglicht den allgemeinen Ablauf im Form von einzelnen Schritten zu definieren. Einzelne Schritte können dabei bei der Implementierung neu definiert werden, um das gewünschte Verhalten festzulegen.

1.1.4 Builder

Das OOP Design Pattern **Builder** ermöglicht das Erstellen von komplexen, zusammengesetzten Objekten in einzelne einfache Schritte zu zerlegen

1.1.5 Facade

Das OOP Design Pattern **Facade** ermöglicht für eine komplexe Klasse, die aus vielen Methoden besteht, eine einfachere Klasse zu erstellen, die nur die notwendigen Methoden von der komplexeren Klasse besitzt, ohne das Verhalten zu verändern.

1.2 CleanArchitecture

some info about clean architecture

1.3 Software Testing

Jede Software erlebt im Laufe der Zeit sehr viele Änderungen. Die erste Version einer Software kann beispielsweise mit einer Version der selben Software nach 10 Jahren, wenig bis keine Gemeinsamkeiten besitzen Jede Änderung des Quellcodes ist ein Risiko für Softwareentwickler, da immer die Gefahr besteht versehentlich funktionierenden Code zu beschädigen.

Um das Risiko auf defekte zu minimieren, können automasierte Tests verwendet werden. Je nach Ausführung stellen diese fest, ob das bestehende Verhalten noch dem Sollverhalten entspricht. Hierbei wird nur das Verhalten geprüft, welches mit den entsprechenden Tests abgedeckt wurde.

Es gibt mehrere Typen von automasierten Tests, die hier kurz beschrieben werden.

1.3.1 Testing Pyramide

Some text to testing pyramid

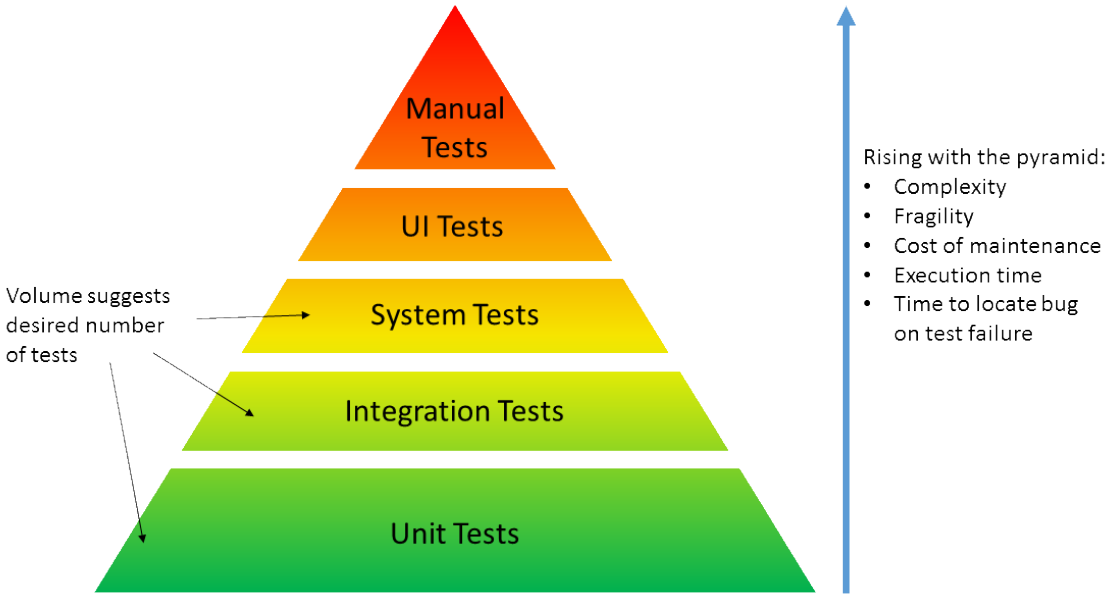


Abbildung 2: Caption written below figure.

Source: https://www.cqse.eu/de/news/blog/junit3-migration/

1.3.2 Unit Tests

Die Unit Tests überprüfen, ob die kleinsten Module (meistens einzelne Funktionen und Objekte), wie gewünscht funktionieren.

Alle Module, welches das zu testende Objekt verwendet, werden während der Ausführung von Unit Tests manipuliert. Auf diese Weise können diverse Situationen simuliert werden.

Die Mehrzahl an Tests in einem Projekt repräsentieren die Unit Tests. Schlägt ein Unit Test fehl, kann der Fehler im Programm sofort lokalisiert und dadurch die Dauer der Fehlersuche minimiert werden.

1.3.3 Integration Tests

Die Integration Tests stellen fest, ob die zusammengesetzte Module, Komponenten oder Klassen, welche die Unit Tests bestanden haben, wie gewünscht funktionieren.

Alle anderen verwendeten Module werden analog zu den Unit Tests manipuliert.

Die Anzahl an Integration Tests in einem Projekt ist geringer als Anzahl an Unit Tests. Auf der einen Seite ist ein Integration Tests größer als ein Unit Test. Andererseits ist die Fehlersuche im Programm bei negativem Testergebnis deutlich komplexer und zeitintensiver, als bei Unit Tests.

Der Vorteil von Integrationstests liegt darin, dass mit ihrer Hilfe ein großer Teil des Codes mit Tests abgedeckt werden kann.

1.3.4 SystemTests

Bei Systemtests wird diese wie ein Produktivsystem gestartet und entsprechend getestet.

1.3.5 UI Tests

Während der Ausführung von UI Tests (User Interface Tests) wird die Eingabe des Benutzers an der Oberfläche simuliert. Jeder UI Test braucht eine laufende Runtime, um den User zu simulieren, ein UI Test wird nur mittels der Ausgabe an der Oberfläsche validiert.

1.3.6 Manual Tests

some Info about manual tests

1.4 SOLID

some info about solid

1.4.1 single-responsibility principle

SRP

1.4.2 open–closed principle

OCP

1.4.3 Liskov substitution principle

LSP

1.4.4 interface segregation principle

ISP

1.4.5 dependency inversion principle

DIP

1.5 GRASP

grasp

1.6 OOP Principles

some info aboout oop principles

1.6.1 Abstraction

some about abstractions

1.6.2 Encapsulation

some abput Encapsulation

1.6.3 Inheritance

some about Inheritance

1.6.4 Polymorphism

some about Polymorphism

2 Aufgabenstellung

Hier steht iwas zur Aufgabestellung

Bedarf:

- Testframework für die Integrationstest von der Ladesäule
- Eigenständiges OCPP1.6 und später OCPP2.0 Server für die manuellen Tests
- ERK (Eichrechtkonformität) automatisiert überprüfen

Lösung: Beide Anforderungen sind sehr ähnlich zueinander = soweit es geht gleiches Code benutzen, nur die unterschiedlichen Teile jeweils für den Bedarf schreiben.

2.1 Anforderungen an Standalone Server

Der Server wird großtenteils für die manuellen Tests benutzt.

- Der Zustand des Servers soll für den Nutzer erreichbar sein (z.B. eine REST Schnittstelle)
- Der Nutzer soll in der Lage sein den Server zu parametrieren

2.2 Anforderungen an Testframework

- Der OCPP Server soll den Port selber auswählen können und dann ihn zurückgeben =¿ um mehrere Tests parallel starten zu können.
- Das Verhalten von dem Testserver soll geändert werden können (auch während der Tests)
- Das Defaultverhalten von dem Testserver soll parametrierbar sein (z.b. einen Benutzer hinzufügen)
- Alle Events, die den Zustand der getesteten Ladesäule aufdecken, sollen beobachtbar sein

2.3 Anforderungen an ERK Teil

BESCHREIBUNG DER LADESAULE

Die Zertifizierung nach dem deutschen Eichrecht entsprechend, muss jede Ladesäule auf Eichrechtskonformität überprüft werden. Dieser Prozess wird immer wieder auf die gleiche Art und Weise im gleichen Umfang durchgeführt. Er beinhaltet somit ein entsprechendes Automatisierungspotential.

Dafür muss für jede Ladesäule ein Ladevorgang gestartet werden (Transaction), währendderen Strom fließt und gemessen wird. Um die Datenintegrität der Messwerte und deren Transport sicherzustellen wird der OCPP Server verwendet. Mit dessen Hilfe kann nachgewiesen werden, dass die Daten nirgendwo in der Software geändert und ebenso unverändert an den Server übertragen und dort abgerechnet wurden. Nach Beenden der Transaction werden mittels einer Drittsoftware die gemessenen Daten mit den Transactiondaten verglichen.

3 Lösung der Aufgabe

hier wird die Lösung der Aufgabe beschrieben mit Bildern usw.

4 Gewünschtes Interfaces

Jedes Tool, Bibliothek, Framework, das von den anderen Menschen benutzt wird, soll soweit wie möglich selbsterklärend sein und intuitiv klar sein.

Damit diese Eigenschaft umgesetzt wird, könnte man die zukunftigen Anwendungen festlegen und daraus eine gute selbsterklärende Schnittstelle erstellen.

Jeder Ablauf eines Systemtests kann mit folgender Schema beschrieben werden:

- 1. Erstellen des Servers mit gewünschten Netzwerkeinstellungen
- 2. Parametrieren/Festlegen des gewünschten Verhaltens des Servers
- 3. Server starten
- 4. Festlegen die Bedingungen für den erfolgreichen Test
- 5. Warten bis der Test abgeschlossen wird
- 6. Ergebnisse validieren
- 7. Alle Instanzen löschen

5 Implementierung

Bei der vorgestellten Implementierung werden Systemtests verwendet. Bei dieser Art von Test wird die Software komplett und analog zum Produktivumfeld getestet. Deshalb sind Systemtests sehr zeitintensiv. Mit folgenden Möglichkeiten kann der Einsatz von Systemtest optimiert werden:

- mehrere Tests pro Setup definieren und ausführen.
- die Tests die in unterschiedlichen Setups ablaufen (d.h. nicht miteinader verbunden sind) parallel durchfuhren

Um die Lesbarkeit des Tests zu verbessern wäre es vom Vorteil, wenn die erstellte OCPPServer Testinstanz bereits ein vordefiniertes Verhalten besitzt, das man ändern kann.

Es soll auch möglich sein das vordefinierte Verhalten zu parametrieren. Dies erfordert Interfaces, die bestimmte Parameter der Instanz ändern können.

Da nur das Verhalten von dem Charging Point getestet werden soll, sollen nur die Ereignisse, die den Zustand des Charging Points abbilden, abrufbar sein. Zum Beispiel: geschickte Nachrichten von dem Charging Point zu dem Server, Reihenfolge der Nachrichten.

5.1 Achitecture des Frameworks

Es wurde entschieden das Framework in 7 Abstraktionsschichten aufzuteilen.

5.1.1 Ports

Ports haben die Aufgabe die Schnittstelle nach Außen aufzubauen und die Verbindungen zu .... (z.B. WebSocket Server, Datenbank)

In dem Framework wird nur ein Port gebraucht - WebSocket Server

5.1.2 Adapters

Adapters sollen die ankommenden Events/Messages vom Port an den zugehörigen Controller zu übersetzen.

In dem Framework wird nur einen Adapter gebraucht - OCPP16 Adapter

5.1.3 Controllers

Controllers besitzen alle Informationen die den Zustand des jeweiligen Components (Controller + Adapter + Port) abbilden.

In dem Framework werden mehrere Controllers gebraucht:

- OCPP Controller(übernimmt die Verantwortung über die Verbindungen zu den Charging Points)
- User Controller(übernimmt die Verantwortung über die Nutzer der Charging Points und ihrer Berechtigungen)
- Transaction Controller(übernimmt die Verantwortung über die Kontroller über die Ladevorgängen)

- Charger Controller(übernimmt die Verantwortung über die Charging Points, die von dem Server bekannt sind und ihrer Zuständen)
- Payment Controller(übernimmt die Verantwortung über die Bezahlvorgang nach dem Ladevorgang)

Die Contoller können von dem Nutzer des Frameworks parametriert werden, um das Verhalten des Servers zu ändern.

5.1.4 Dispatcher

Der Dispatcher informiert alle abonnierten UseCases über das eingetretene Event.

In dem Framework sind nur OCPP Events wichtig (Nachrichten und Verbindungsevent).

5.1.5 UseCases

UseCases beschreiben den Vorgang beim Auslösen eines Events, welches sie abonniert haben. Die vordefinierten UseCases dürfen nur Interactors benutzen um das Verhalten zu definieren.

Dieses Framework beinhaltet UseCases, welche ein vordefiniertes Verhalten besitzen. Diese kann auch entsprechend umgeschrieben werden.

5.1.6 Interactors

Eine atomare Operation im Programm (die Operation lässt sich nicht mehr sinnvoll im Rahmen der Anwendung aufgeteilen). Interactors benutzen Contoller "Dependency Injection"

Benutzt mittels "Dependency Injection"die Controller.

In dem Framework werden sie nur als "Wrapper"für alle Funktionen von Controllern implementiert.

5.1.7 Domain

Eine Domain definiert alle Types und Interfaces der Applications. Sie beschreibt die Verbindungen zwischen den Interfaces und Types, die dann in den anderen 6 Layers umgesetz werden.

5.2 Zugriff auf das Testframework

IRGENDWAS EINLEITENDES

- OCPPPort soll nur bei der Initialisierung der Instanz parametrierbar sein (Netzwerkeinstellung)
- Adapters sollen nicht von der Seite des Frameworks aufrufbar sein
- Contollers sollen nicht von der Seite des Frameworks aufrufbar sein
- Dispatcher darf nur zum Abonnieren/Deabonnieren benutzt werden um das Verhalten des Charging Points beobachten zu können
- UseCases sollen überschreibar und erweitbar sein, falls man bestimmtes Verhalten hinzufügen möchte.
- Interactors sollen von der Seite des Frameworks aufrufbar sein, um die Serverinstanz parametrieren zu können.

- Domain beinhaltet alle Typen die in den anderen Layers verwendet werden. Aus diesem Grund sollen die Typen von der Seite des Frameworks benutzbar sein.

5.3 Testbeispiel

```
describe("example test", () => {
    let testOcppInstanz: MyOCPPTestFramework;

    before(async () => {
        // 1. Create test server instanz
        testOcppInstanz = new MyOCPPTestFramework({ host: "https://127.0.0.1", port: 8080 });

        //2.1. rewrite behavior of server
        testOcppInstanz.rewriteUseCase("nameOfUseCase", {});
        //2.2. add behavior to server
        testOcppInstanz.addUseCase({});

        //2.3. parametrize the behavior
        testOcppInstanz.interactors.interactor_1();
        testOcppInstanz.interactors.interactor_2();

        // 3. start server
        await testOcppInstanz.start();
        return;
    });
    it("test 1", async () => {
        // 5. wait until test is done
        const res = await testOcppInstanz.waitOfEvent("BootNotification");

        // 4. and 6. set the requirement test is valid, validate the result
        expect(res.name).equal("BootNotification");
    });
    it("test 2", async () => {
        // 5. wait until test is done
        const res = await testOcppInstanz.waitOfNextEvent();

        // 4. and 6. set the requirement test is valid, validate the result
        expect(res.name).equal("Heartbeat");
    });
    after(async () => {
        // 7. stop server
        testOcppInstanz.stop();
        return;
    });
});
```

6 Conclusion

But the fact that some geniuses were laughed at does not imply that all who are laughed at are geniuses. They laughed at Columbus, they laughed at Fulton, they laughed at the Wright Brothers. But they also laughed at Bozo the Clown - Sagan (1993).

Bibliography

Ghia, U., K. N. Ghia und C. T. Shin (1982). „High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method“. In: *Journal of Computational Physics* 48, S. 387–411.

NTNU, Department of Marine Technology (2020). *IMT Software Wiki - LaTeX*. URL: https://www.ntnu.no/wiki/display/imtsoftware/LaTeX (besucht am 15. Sep. 2020).

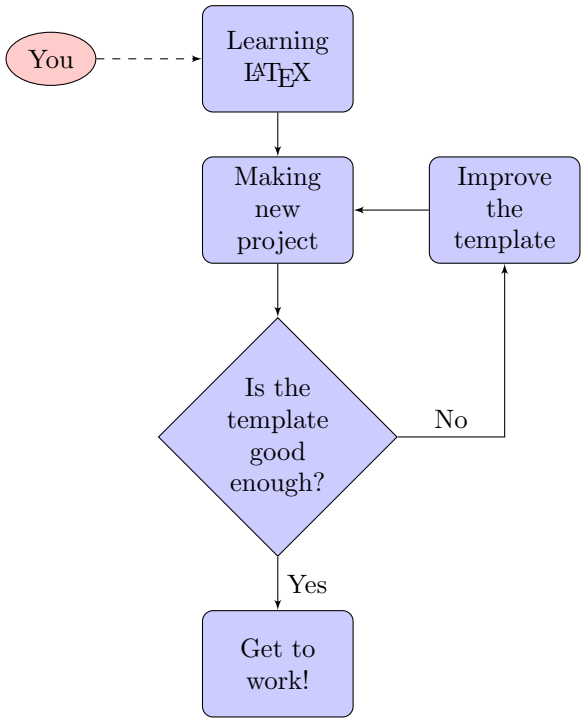
Sagan, Carl (1993). *Brocas brain: reflections on the romance of science*. Presidio Press.

Appendix

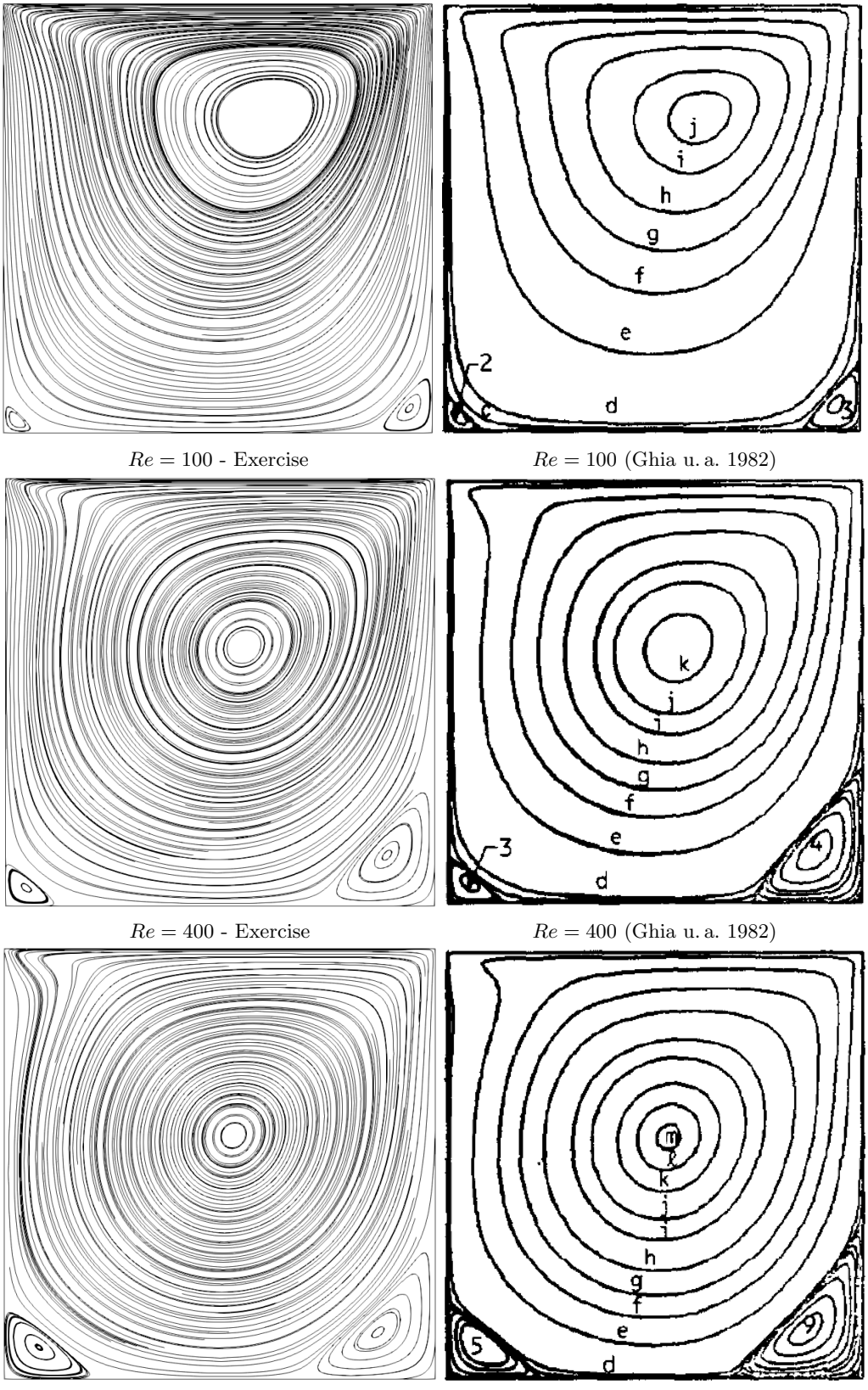
A Hello World Example

⋮

B Flow Chart Example



C Sub-figures Example



$Re = 100$ - Exercise

$Re = 100$ (Ghia u. a. 1982)

$Re = 400$ - Exercise

$Re = 400$ (Ghia u. a. 1982)

$Re = 1000$ - Exercise

$Re = 1000$ (Ghia u. a. 1982)

Abbildung 3: Streamlines for the problem of a lid-driven cavity.