UNIVERSITY OF MANNHEIM

Using Test Sheets for asynchronous testing of Real Time Software

Master Thesis

submitted: July 2016

by: Denys Zalisky
dzaliskyi@mail.uni-mannheim.de
born November 29th 1991
in Svetlovodsk

Student ID Number: 1440397

University of Mannheim
Chair of Software Engineering
D – 68159 Mannheim
Phone: +49 621-181-3912, Fax +49 621-181-3909
Internet: http://swt.informatik.uni-mannheim.de

Abstract

- 2-3 sentences - current state of art
- 1-2 sentences - contribution to improvement
- 1-2 sentences - specific result of the paper and main idea behind it
- 1 sentences - how result is demonstrated and defend

While providing of simple way for test description is a hot topic in software development. There is no software developed for a realization of Test Sheet concept, pragmatic way of defining tests which lays between two extreme paradigms FIT and hard coded test definitions.

This paper describes processes of design and implementation of the Test Sheets' concept together with integration of the product to business processes of figo GmbH for a real-time testing/validation of internet banking web pages.

Result of this research is following: developed conventions for Test Sheets definitions in particular use case, implemented interpreter from Test Sheets to executable JavaScript code.

Conventions and the code listing of main module together with example of executable JavaScript file are provided as well as statistics regarding improvement of user experience and overall system fault prevention improvements.

Some feedback from Bianca and Sebastian + statistics regarding user experience improvement

Contents

List of Figures

List of Tables

1. Introduction

- What precisely did I answer

  what question did I answer

  why should the reader care

  what larger question does this address

- What is my result

  What new knowledge have I contributed that reader can use else where

  What previous work do I build on

  What precisely and in detail my new result

- Why should the reader believe in my result

  What standard was used to evaluate the claim

  What concrete evidence shows that me result satisfies my claim

Relevance of the topic and the necessity for scientific investigation: No researches found regarding semi automated tests generation for web page verification.
Practical and theoretical value of the topic: Implementation of enginee for Test Sheets with application of software design and development practices.
Motives for choosing a particular topic: Necessity of tests defined by non-developers for figo for a real-time testing (will be provided later)
Research problem and why it is worthwhile studying - definition of convention for test sheets definition, usage of test sheets for testing of asynchromous systems in a real-time.
Research objectives - design and development of software for translation of test sheets in to executable java script for testing asyncronous calls to external system.

Structure of the thesis : A paragraph indicating the main Contribution of each chapter and how do they relate to the main body of the study Limitations of the

study

2. figo GmbH

2.1. General Information

According to Banks Germany[4] "The German banking system is divided by three large sectors: private, public, and cooperative. The cooperative sector is represented by 1,144 credit unions and 2 cooperative central banks. The public sector employs 431 savings bank, 10 land banks and other institutions. Private banks represented 4 transnational banks, 42 investment banks, and 176 regional and other banks. There is also operating are 167 registered branches of foreign banks, including 60 investment banks." While introduction of Payment Services Directive (PSD) and PSD2 by European Commission in European Union together with initiatives of UK Government regarding API provision and standardization have obligated Banks with the implementation of on-line access points to their services[22][25][3]. Within the Single European Payment Area acceptance of directive by European Bank Authority scheduled within 2017 year[8].

figo GmbH is a financial technology company with headquarter in Hamburg. It was founded in 2012 with the mission to "build the backbone of next generation financial services"[16]. Currently, the API is fully functional in Germany, partly in Austria and England.[9][10]

Functions available through API:[11]

- Create, Read, Update, Delete Bank account(s);
- Read, Update, Delete Bank account(s) transactions;
- Create, Read, Update, Delete Bank account(s) standing orders;
- Create, Read, Update, Delete Bank account(s) direct debits;
- Read, Update Bank account(s) securities;
- Create, Read, Update, Delete Bank payment(s);

To see the list of figo's partners and customers with their use-cases please visit `http://figo.io/use_cases.html`.

figo Connect API was created to accelerate innovations in the FinTech area and to allow figo's partners to offer products with real added value[30]. It enables developers, startups and even banks to connect to every financial service provider. These partners can access every bank account (current, savings, loan, securities, ...), credit card, eWallet and other financial services like PayPal through one single REST-API. [30][16][33]

2.2. IT infrastructure. Banking Server

The high level IT infrastructure of figo GmbH consists two parts (Fig. 2.2). The **API Server** - implements interfaces to figo's customers and partners for accessing banking information and services (lays outside of this paper's scope). The **Banking Server** - implements connection to banks via three possible communication channels, they are description provided below together with basic motivation for each of them.

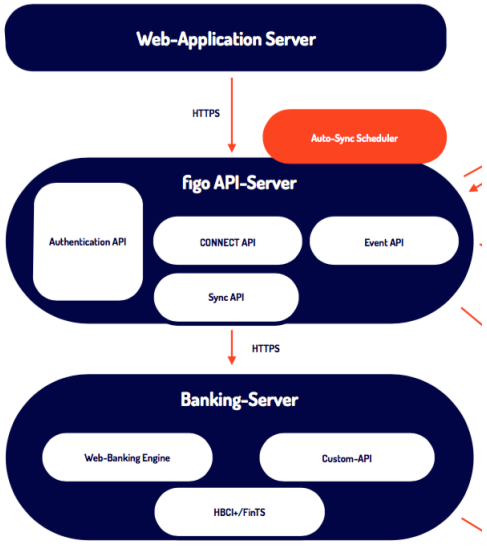


Figure 2.1.: figo GmbH high level architecture

2.2.1. Banking Server Architecture

Banking server has three parts for communication with banks via three separate channels. Each of them is a realization of different technology: ***Custom-API*** is responsible for connection to custom APIs provided by banks, ***HBCI/FinTS*** is responsible for connection to banks' interfaces via FinTS/HBCI, ***Web-banking Engine*** is responsible for communication with banks which does not provide API or HBCI.

Custom-API implements the client for custom banks APIs. Some of them provide full functionality while some only partial. All this APIs vary in their structure and functionality but most of them implements REST API specification.

HBCI+/FinTS is an implementation of an adapter OOP pattern for jsHBCI library. Home Banking Computer Interface (HBCI) is an open publicly available protocol. Its specification was originally designed by the two German banking groups *Sparkasse and Volksbanken und Raiffeisenbanken* and *German higher-level associations as the Bundesverband deutscher Banken e.V.*. [12]

Web-Banking Engine is an implementation of a factory OOP pattern for scraping libraries which responsible for communication with banks which does not provide API or HBCI. Here figo GmbH uses web scraping technology to perform interaction with internet banking web page. From the banks perspective an interaction looks completely like direct communication with an user, while an user does not feel the difference between interaction via Custom-API or HBCI or Web-Banking Engine. This is the most sensitive part from the developer's perspective since every change to the bank's web page can leads to failure of the specific scripts.

The aim of this paper is an application of Test Sheet for early (before any user's interaction will take place) recognition of page changes and notification of developers regarding failed part of the script. Testing scripts generated from Test Sheets are used as a (demon task/crown job) in timely fashion with notification

of the developer in case of unexpected behavior of the scripts via (email/slack) communication channel.

3. Testing. Existing approaches. Test Sheets

3.1. Software testing

Oxford dictionary defines test as a procedure intended to establish the quality, performance, or reliability of something, especially before it is taken into widespread use. The price of software project for at least 30% consists of costs for tests[40]. Robert Cecil Martin in his book 'Clean Code: A Handbook of Agile Software Craftsmanship'[48] says following: "Tests are as important to the health of a project as the production code is. Perhaps they are even more important, because tests preserve and enhance the flexibility, maintainability, and reusability of the production code. "

Software testing is an important technique to evaluate and assess software's quality and reliability[42], it provides possibility to determine whether the development of product conform the requirements [17]. The knowledge on software testing have become crucial for all software developers and engineers[38].

Tsai, Fang and Bi described/suggested/introduced/just showed the general testing and debugging strategy (Fig. 3.1).

The economical costs of unexpected software behavior (bug, failure or error) can go up to several millions or even billions of USD.
Web application failures only in USA lead to looses of $6.5 million per hour in financial services and $2.4 million per hour in credit card sales applications[39].
Alarm-management fault was one of the reasons of the black-out occurred in the northeastern US on August 2003. Estimated costs were between US$7 and $10 billion.[26]
In general, according to report made by US National Institute of Standards and Technology (NIST) in 2009 the estimated economy loses were $60 billion anually as associated with developing and distributing software patches and reinstalling

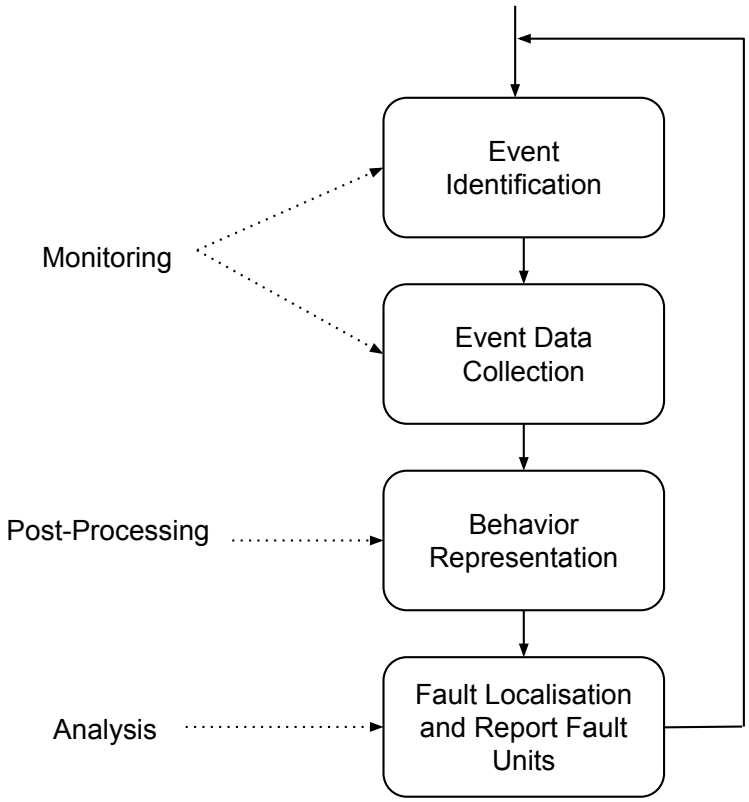


Figure 3.1.:

Figure 3.2.: General Testing and Debugging Strategy[52]

systems that have been involved, together with losses in productivity due to errors and malware infections[26].

The human causalities caused by software misbehavior can rich dramatic scales.

The case of Toyota unintended acceleration caused by software defect in 2009 - 2011 led to 89 deaths and 57 injures. Toyota was fined $1.2 Billion for concealing safety defects[5].

The Patriot, surface-to-air missile, system failed to track and intercept an incoming Scud missile on 25 of February 1991. As the result 28 soldiers were killed and around 100 were injured[26].

A Therac-25, linear accelerator, which failure caused to 6 know cases where can-

cer patients received deadly radiation overdoses during their treatment between June 1985 and January 1987. The dosages were 100 times exceeding typically used for treatment[26][47].

Software testing is a significant part of software development[27]. The variety of tests coincides variety of requirements for the software under test. Test processes determine whether the software conform to the requirements and/or satisfies its intended use and user needs[17]. As well as an important technique to evaluate and help to improve software quality, reliability and saftiness[42]. General requirements to test definitions are following: fast, independent, repeatable, self-validating, timely. More detailed information regarding requirements can be found in a book of Robert Cecil Martin 'Clean Code: A Handbook of Agile Software Craftsmanship' [48] as well as via following URL: `http://www.extremeprogramming.org/rules/testfirst.html`

3.2. Existing approaches for test definitions

In most of the cases developers are using testing frameworks or internal DSLs which requires usage of formal programming languages. This in a result makes writing of tests not different from the programming. It requires knowledge of languages and understanding of basic programming concepts from everyone who is involved in the creation of a test, reading its result or updating/deleting the test.

Since it is common practice to base tests on top of business requirements tests should be used by different shareholders who define this requirements.

Below provided brief overview of different testing approaches and their analysis with respect to modern business requirements.

3.2.1. xUnit

xUnit is a family of unit testing frameworks with shared architecture and functionality which is derived from Smalltalk's SUnit, designed for tests automation[50][35]. The general simplicity and lightweight made them popular tool for Test Driven

Development[35].

xUnit basic features implemented by all members of the family provide functionality to perform following tasks[50]:

1. Specify a test as a *Test Method*;
2. Specify the expected results within the test method in the form of calls to *Assertion Methods*;
3. Aggregate the tests into test suites that can be run as a single operation;
4. Run one or more tests to get a report on the results of the test run

Test ***cases*** in xUnit are defined as a methods united in to ***suites*** with shared preconditions called ***fixtures*** all this is executed by a ***runner*** which compares actual and expected results using ***assertion*** function.

The family includes a wide variety of implementations for multiple programming languages, and diverse enhancement of functionality (e. g. code coverage statistics, assertion add-ons etc.)[36].

Definition of tests with formal general programming language from one side makes tests definition and execution fast but in a same time makes impossible the creation of tests for people without developer background.

3.2.2. Fit

Framework for Integrated Test (Fit) - is a way of defining test cases with HTML pages. It enhances the communication and collaboration connecting customers and programmers. Moreover, it creates feedback loop between them. Fit automatically checks HTML pages against actual program[13].

Fit reads tables in HTML files, each table is interpreted by a *fixture* written by programmers. This fixture checks the examples in the table by running the actual program[14].

Programmers use a *ColumnFixture* to map the columns in the table to variables and methods in the fixture. The first columns provide correspond to variables in the fixture. The last column has the expected result and corresponds to the method in the fixture[14].

Different implementation provide different user experience from Wiki pages (i.e. FitNess) to complete standalone application with internal functionality for tables definitions and tests execution (i.e. GuiRunner).

With Fit, customers can provide more guidance in a development process by lending their subject matter expertise and imagination to the effort[13] requiring developer to write only *fixture*, the middleware between tests and code. In the same time it provides media between create, read, update, delete operations over tests and the results of their execution for people without development experience.

3.2.3. Cucumber

Cucumber is a Behavior Driven Development tool which allows users to define tests specifications with *Gherkin*, the language which is structured plain text with support of internationalization for 60+ different languages[7].

Feature files written in Gherkin consists of plain text and general key-words are used for identification of following concepts:

- Feature under the test: *Feature*;
- Test scenario: *Scenario*;
- Scenario Outline: *Scenario Outline*
- Test Steps: *Given, When, Then, And, But*;
- Test Background: *Background*;
- Test Examples: *Examples*

Specific characters are used for identification of:

- Step Inputs: String - *"""*, Table - |;
- Step Tags: @;
- Comments: #

Step definitions are written by developers which parses feature file with attached pattern to link all the step matches and code executed by cucumber when match is met.

Human readable input and output of the tests defined in gherkin makes cucumber a good media for communication between people who create requirements and

those who are trying to meet them. But in a same time software development experience required for writing code with step definitions.

3.3. Test Sheets

Test Sheets is a representation for tests a developed with the goal to combine the power and completeness of formal programming language with a representation that is easy to understand and work with even for people with little IT knowledge[27].

Test Sheets approach uses usual spreadsheet for definition of test and representation of their result.

- Rows - represent operations being executed;
- Columns - variables for input or output parameters;

The actual content of a cell can be made dependent on other cells by addressing them via their location. Just like in Fit result of tests execution is provided in a same table with coloring and actual return delimetered by \symbol in case of failing test. [28]

There are three types of test sheets which can be used in combination:

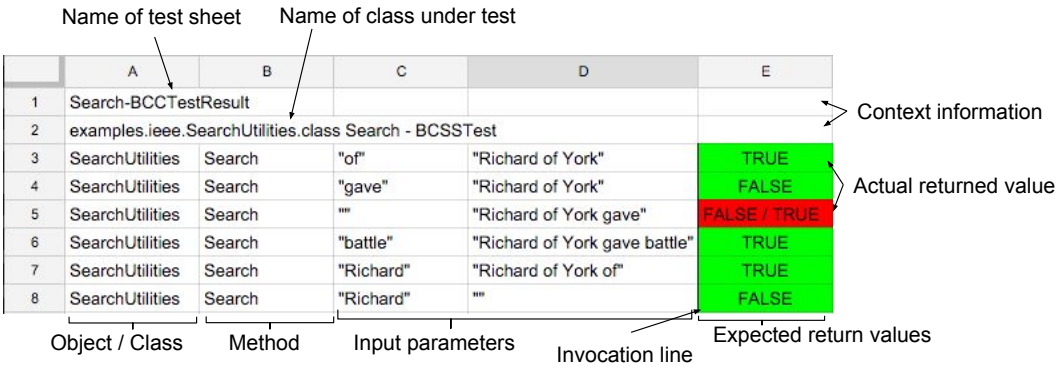
- Basic - order of test step execution is defined by order of rows in a table;
- Non-Linear - order of test steps execution is defined by finite state machine with states represented by test steps and transition function by step execution results or number of test step execution;
- High-Order/Parametrized - The actual value used for Parameterized Test Sheets is specified by a Higher-Order Test Sheet in a column with letter Parametrized Test Sheet refers to.

The scope of this paper covers only Basic Test Sheets. For more details about Test Sheets and Research Topics of the Chair of Software Engineering please visit `http://swt.informatik.uni-mannheim.de/de/home/`.

3.3.1. Basic Test Sheets

A Basic Test Sheet consists of a name and a class being tested provided in the first and the second rows representatively. Each row after represents one method call

(test step). The first column identifies the object being tested. The second column indicates method under test with input parameters provided in a next columns but before invocation line. Right after the invocation line expected return values are specified. The description provided above is shown in a following summarized example from the web site of Chair of Software Engineering of University of Mannheim (Pic 3.3.1).



	A	B	C	D	E
1	Search-BCCTestResult				
2	examples.ieee.SearchUtilities.class Search - BCSSTest				
3	SearchUtilities	Search	"of"	"Richard of York"	TRUE
4	SearchUtilities	Search	"gave"	"Richard of York"	FALSE
5	SearchUtilities	Search	""	"Richard of York gave"	FALSE / TRUE
6	SearchUtilities	Search	"battle"	"Richard of York gave battle"	TRUE
7	SearchUtilities	Search	"Richard"	"Richard of York of"	TRUE
8	SearchUtilities	Search	"Richard"	""	FALSE

Figure 3.3.: Basic Test Sheet

The main benefit over *xUnit*, *Cucumber*, *Fit* is an exclusion of software developers from test create, read, update, delete operations which can be performed with usage of any spreadsheet editor.

4. Real-Time Software. Web scrapping

Donald Gillies defined a real-time software as a system : *"[...] in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred."*

While Robert L. Glass[45] defines this term as: *"[...] a software that drives a computer which interacts with functioning external devices or objects. It is called real-time because the software actions control activities that are occurring in an ongoing process".*

Within this research we will define *Real-Time software* as a combination of this to definitions which covers both logical and time correctness as well as an interaction with external systems controlled by the ongoing process.

The Web-Banking engine of figo GmbH matches this definition due to the following facts. First it performs communication with external systems (Web Banking HTML pages). Next its result correctness depends time restrictions (if child process responsible for script execution was not finished within 1200 seconds, and with each task performed within 0.5 seconds it is treated as failed) as well as logical correctness depended upon ability of the script to perform necessary actions for a fulfillment of a requested task.

Tsai, Fang and Bi[52] state that testing and debugging of real-time software are very difficult because of timing constraints and non-deterministic execution behavior. In a real-time system, the processes receive inputs from real world processes as a result of asynchronous interrupts and it is almost impossible to precisely predict the exact program execution points at which the inputs will be supplied to the system. Consequently, the system may not exhibit the same behavior upon repeated execution of the program. In addition, in a real-time system, the pace of execution of processes is determined not only by internal criteria, but also by real world processes and their timing constraints.

4.1. Web scraping

!!!give some motivation

"Web scraping (web harvesting or web data extraction) is a computer software technique of extracting information from websites. Usually, such software programs simulate human exploration of the World Wide Web by either implementing low-level Hypertext Transfer Protocol (HTTP), or embedding a fully-fledged web browser, such as Mozilla Firefox.

Web scraping is the process of automatically collecting information from the World Wide Web. It is a field with active developments sharing a common goal with the semantic web vision, an ambitious initiative that still requires breakthroughs in text processing, semantic understanding, artificial intelligence and human-computer interactions. Current web scraping solutions range from the ad-hoc, requiring human effort, to fully automated systems that are able to convert entire web sites into structured information, with limitations." [31]

"The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents." [29]

?!?!?! add alternative technologies and compare them from the companies perspective!

4.1.1. CasperJS

[6] CasperJS is an open source navigation scripting & testing utility written in JavaScript for the PhantomJS WebKit headless browser and SlimerJS (Gecko). It eases the process of defining a full navigation scenario and provides useful high-level functions, methods & syntactic sugar for doing common tasks such as:

- defining & ordering browsing navigation steps
- filling & submitting forms
- clicking & following links
- capturing screenshots of a page (or part of it)

- testing remote DOM
- logging events
- downloading resources, including binary ones
- writing functional test suites, saving results as JUnit XML
- scraping Web contents

!!! Add a conclusion

5. Asynchronous programming. Node.js. Strategies and performance

5.1. Asynchronous programming

Asynchronous programming is the programming model in which operations should be done are interleaved with one another within the single control thread. Analogy to it can be package multiplexing in a Computer Networks.

To compare to multithreaded systems asynchorous allow execution of one task per unit of time. Further, tread suspension/invocation lays outside programmer's control while in asynchonous system task will run untill programmer will perform explicit control delegation to other task.[18]

In contrast to synchronous systems where process can be blocked by one another (i. e. waiting response from I/O) asynchronous system instead of waiting will switch to execution of the next task in a thread.

Generally asynchronous systems are easier to control and to develop rather then multithreaded, and they perfrom better then synchronous in following cases[18]:

- Large number of tasks and at least one task likely to make progress;
- A lot of I/O operations;
- Tasks are independent from one another;

More detailed asynchronous system will be described in the end of the next section on an example of Node.js.

5.2. Node.js

Node.js is an asynchronous event driven framework designed to build scalable network applications. Node.js is similar in design to and influenced by systems

like Ruby's Event Machine or Python's Twisted with difference that it presents the event loop as a language construct instead of as a library[1].

Node.js applications are written in JavaScript "a lightweight, interpreted, programming language with first-class functions. Most well-known as the scripting language for Web pages, many non-browser environments use it such as node.js and Apache CouchDB. JS is a prototype-based, multi-paradigm, dynamic scripting language, supporting object-oriented, imperative, and functional programming styles" as defined by Mozilla[20] which makes lowers entry border for the developers due to the language age and commonality.

npm ecosystem includes *npm* - package manager for Node.js, *npm Registry* - public collection of packages of open-source code, *npm comand line clinet* which allows developers to install and publish those packages. On a diagram (5.2) stated the comparison of npm with other package managers made by module counts, for more up to date information please visit `http://www.modulecounts.com/`

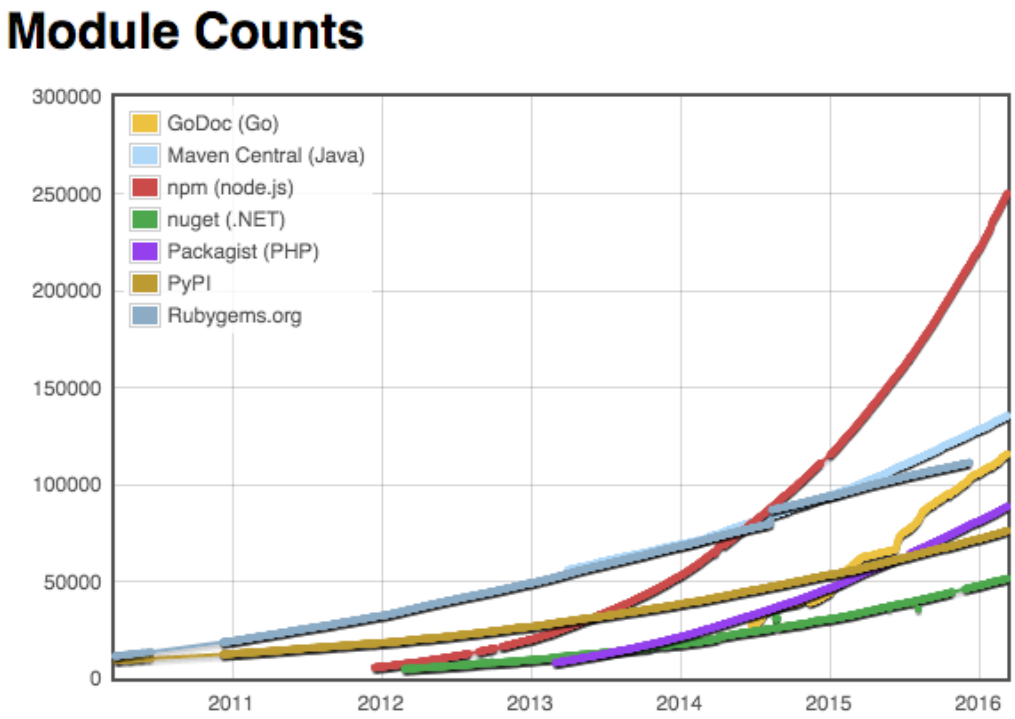


Figure 5.1.: npm comparison with other package managers[23]

The highlevel architecture of Node.js looks following:(5.2).

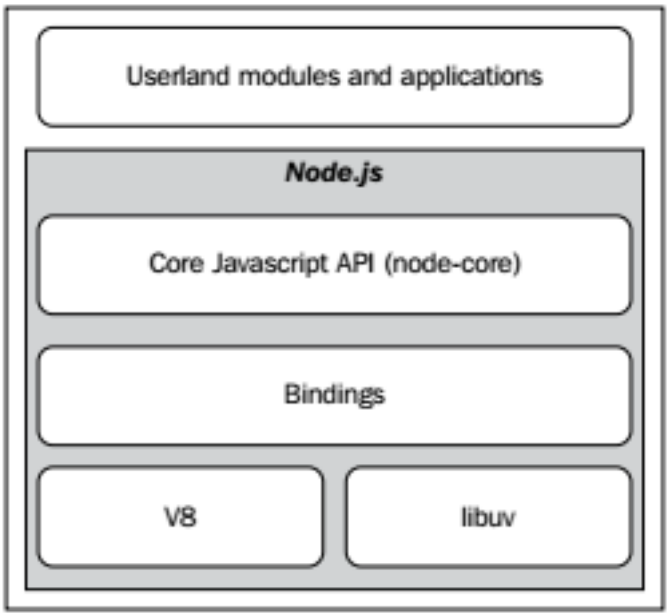


Figure 5.2.: Node.js architecture [41]

Node core is a JavaScript library (called node-core) that implements the high-level Node.js API.

Bindings responsible for wrapping and exposing libuv and other low-level functionality to JavaScript.[41]

Non blocking I/O Non blocking I/O in NodeJS is provided by libuv[1][41]. Which is "a multi-platform support library with a focus on asynchronous I/O. "[32] with following properties[19]:

- Abstract operations, not events
- Support different nonblocking I/O models
- Focus on embeddability and perfomace

V8/Chakra the JavaScript engine originally developed by Google for the Chrome browser/ Microsoft for IE 9 browser"[41]

5.2.1. Event

"Much of the Node.js core API is built around an idiomatic asynchronous event-driven architecture in which certain kinds of objects (called "emitters") periodically emit named events that cause Function objects ("listeners") to be called.

All objects that emit events are instances of the EventEmitter class. These objects expose an eventEmitter.on() function that allows one or more functions to be attached to named events emitted by the object.

When the EventEmitter object emits an event, all of the Functions attached to that specific event are called synchronously."[**?**]

5.2.2. Event handling

Node.js event handler is an implementation of reactor pattern (5.2.2). The description of process lifecycle is shown on Fig. 5.2.2:

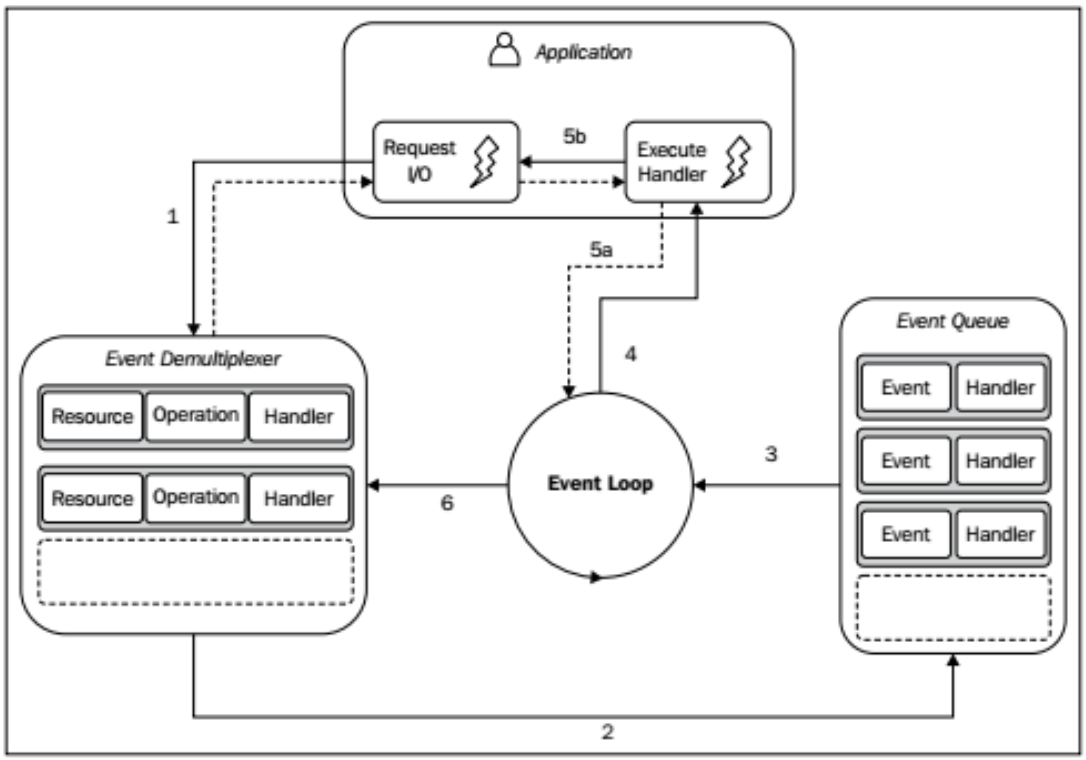


Figure 5.3.: Node.js event handling system [41]

1. The application generates a new I/O operation by submitting a request to the Event Demultiplexer. The application also specifies a handler, which will be invoked when the operation completes. Submitting a new request to the Event Demultiplexer is a non-blocking call and it immediately returns the control back to the application.
2. When a set of I/O operations completes, the Event Demultiplexer pushes the new events into the Event Queue.
3. At this point, the Event Loop iterates over the items of the Event Queue.
4. For each event, the associated handler is invoked.
5. The handler, which is part of the application code, will give back the control to the Event Loop when its execution completes. However, new asynchronous operations might be requested during the execution of the handler, causing new operations to be inserted in the Event Demultiplexer, before the control is given back to the Event Loop.
6. When all the items in the Event Queue are processed, the loop will block again on the Event Demultiplexer which will then trigger another cycle.

5.3. Asynchronous handling strategies

5.3.1. Callbacks

Presence of Functions as a first class citizens in javascript allows the usage of functions for handling asynchronous program behavior. Callbacks are handlers for the reactor pattern described before. They are similar to Visitor Pattern in OOP, they also represent an operation to be preformed on the elements of an object structure and it lets you define a new operation without introducing any changes to the definition of the object.

Callbacks implement continuation-passing-style from FP. By convention callbacks must be passed as the last argument and accept two parameters, the first is an error and the second is a data to be processed further.

There are negative sides of using callbacks. One of them is so called *callback hell* occurs due to the abundance of closures and in-place callback definitions. This

makes the code hard to be read because of high level nesting, as well as written due to a scope of nesting and difficult to manage because of possible memory leaks created by closures. Another negative side is called *releasing Zalgo*[15], it occurs only in case of inconsistent function behavior, when under some hidden conditions a function performs synchronous action but some under other - asynchronous.

5.3.2. Data focused strategies / Imperative

The explanation of asynchronous patterns in this chapter will be performed via mapping of *synchronous* patterns to the *asynchronous*. Another dimension of mapping is *singular* vs *plural*.

Before starting imperative strategies for handling of asynchronous data flow we would like to show the concept of duality showed by Erik Meijer `https://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2014/Keynote-Duality` and hardly used by Kris Koval in his General Theory of Reactivity `https://github.com/kriskowal/gtor/`. This chapter is based on works of Kris Koval, Erik Meijer, Conal Eliot and Mark S. Miller in a field of Reactive Programming.

During communication can occurred the problem caused by the fact that different parts for the dialog can have different load and different performance, this case is mostly related to asynchronous case since this parts are distributed. The problem can be divided in to two types of situations.

The first situation is **Fast producer - slow consumer**. It means that get of one entity works faster then set of next entity in the chain. This situation occurs when values are ***pushed*** by producer. The second situation is **Slow producer - fast consumer**. It means that get of one entity works slower then set of next entity in the chain. This situation occurs when values are ***pulled*** by consumer.

Solution of this problems lays on scope of the system design which should allocate push and pull entities in a appropriate sides of the communication channels.

Value is a singular unit of *synchronous* data. It has two duals, they are *getter (pull)* and *setter (push)*. Setter accepts value to be assigned and return nothing and getter accepts nothing and returns a value. The chaining process can be

performed here by applying setter to getter and getter to setter and by applying same logic to their analogs for further entities.

Collection is *synchronous* and it is a plural form of the value. The duals of a collection are *iterator (pull)* and *generator (push)*. Iterator as a plural analog of getter, it accepts nothing and returns the element from collection. Generator is a plural analog of setter it accepts element to be added to collection and returns nothing.

?! Which of the duals will do blocking ?!

Deferred is *asynchronous* analog of the value. The duals for promise are *resolver (push)* and *promise (pull)*. The resolver is an asynchronous analog of setter. It accepts value which will be assigned as soon as it will be resolved. The promise is an asynchronous analog of the getter . It allows to obtain the value of the promise as soon as it will be resolved. The deferred concept guaranties unidirectional data flow which means that data can go in one and the only one direction from resolver to promise. Further deferred entities guaranties asynchronicit execution of am operation which means they are 'Zalgo safe'[2].

Stream is *asynchronous* analog of the collection. It can be treated as a collection of deferred elements. The duals for stream are *write (pull)* and *read (push)*. The read is an analog of iterator it accepts nothing and pulls values from the stream.The write is an analog of generator it accepts values from the stream and returns nothing. As a plural analog of deferred it guaranties unidirectional data flow which means that data can go in one and the only one direction from read to write. The special case for streams in Node.js are transform and duplex streams which are combination of read and write.

For more information about reactive programming with javascript please visit `https://github.com/kriskowal/gtor/`.

The matrix of presented on (Fig. 5.3.2).

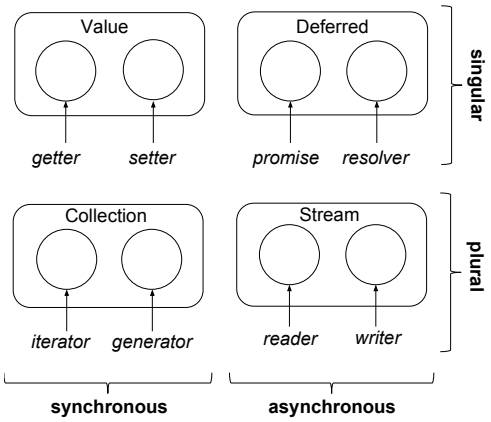


Figure 5.4.: Duality Matrix [21]

5.4. Performance

Following measures are taken from the article 'Analysis of generators and other async patterns in node' by Gorgi Kosev (`https://spion.github.io/posts/analysis-generators-and-other-async-patterns-node.html`).

There are no hardware characteristics provided for the experiment execution environment except following: "On my machine redis can be queried about 40 000 times per second; node's "hello world" http server can serve up to 10 000 requests per second; postgresql's pgbench can do 300 mixed or 15 000 select transactions per second." [34]

The performance metrics were taken from the experiment under the following conditions: "All external methods are mocked using setTimeout 10ms to simulate waiting for I/O with 1000 parallel requests (i.e. 100K IO / s) [34]

pattern	**time(ms)**	**memory (MB)**
promises-bluebird	512	57,45
promises-bluebird-generator	364	41,89
callbacks	316	34.97

Table 5.1.: Performance comparison of patterns for asynchronous information flow [34][2]

"The original and flattened solutions are the fastest, as they use vanilla callbacks"[2]

Note that this table has only fastest among promise objects and since there were

no measurements performed for streams but according to their nature the most optimistic performance for them should be equal to the promise generator.

The mismatch which can occur between results of this experiment and real life lays withing fast producer - slow consumer / slow producer - fast consumer occasion described above.

6. OOP and FP

6.1. Object Oriented Programming

"Programming languages with objects and classes typically provide dynamic lookup, abstraction, subtyping, and inheritance. These are the four main language concepts for object-oriented programming. They may be summarized in the following manner: *Dynamic lookup* means that when a message is sent to an object, the function code (or method) to be executed is determined by the way that the object is implemented, not some static property of the pointer or variable used to name the object. In other words, the object "chooses" how to respond to a message, and different objects may respond to the same message in different ways. *Abstraction* means that implementation details are hidden inside a program unit with a specific interface. For objects, the interface usually consists of a set of public functions (or public methods) that manipulate hidden data. *Subtyping* means that if some object a has all of the functionality of another object b, then we may use a in any context expecting b. *Inheritance* is the ability to reuse the definition of one kind of object to define another kind of object. "[51]

6.1.1. Design Principles

Agile design is a process, not an event. It's the continous application of principles, patterns, and practices to improve the structure and readability of the software. It is the dedication to keep the design of the system as simple, clean, and expressive as possible at all times.[49]

Symptoms of not agile design [49]:

- Rigidity - The design is difficult to change;
- Fragility - The design is easy to break;
- Immobility - The design is difficult to reuse;

- Viscosity - It is difficult to do the right thing;
- Needless complexity - Overdesign;
- Needless repetition - Mouse abuse;
- Opacity - Disorganized expression;

Foundamental Object Oriented design principles[43][49] :

- Closing - Encapsulate things in your design that are likely to change.
- Code to an Interface - rather then to an implementation.
- Do not repeat yourself (DRY) - Avoid duplicate code.
- The Single-Responsibility Principle (SRP) - A class should have only one reason to change
- The Open/Closed Principle (OCP) - Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification
- The Liskov Substitution Principle - Subtypes must be substitutable for their base types.
- The Dependency-Inversion Principle - A) High-level modules should not depend on low-level modules. Both should depend on abstractions. B) Abstractions should not depend upon details. Details should depend upon abstractions.
- The Interface Segregation Principle (ISP) - Clients should not be forced to depend on methods they do not use.
- Principles of Least Knowledge (PLK) - Talk to your immediate friends.
- Principle of Loose Coupling - object that interact should be loosely coupled with well-defined intefaces.

6.1.2. Design Patterns

"Design patterns make it easier to resuse successful designs and architectures. Expressing proven techniques as design patterns makes them more accessible to developers of new sysntems. Desugb patterns help to choose design alternatives that make a system reusable and avoid alternatives that compromise resuability. "[44]

Creational Design Patterns abstract the instantiation process and provide indepedecne for object creation, composition and representation. Factory - "The Factory Design Pattern is probably the most used design pattern in modern programming languages like Java and C#. It comes in different variants and implementations. If you are searching for it, most likely, you'll find references about the GoF patterns: Factory Method and Abstract Factory.

- creates objects without exposing the instantiation logic to the client.
- refers to the newly created object through a common interface

"[24]

Abstract Factroy(for HO test sheets) - "Using this pattern a framework is defined, which produces objects that follow a general pattern and at runtime this factory is paired with any concrete factory to produce objects that follow the pattern of a certain country. In other words, the Abstract Factory is a super-factory which creates other factories (Factory of factories). Abstract Factory offers the interface for creating a family of related objects, without explicitly specifying their classes."[24]

ObjectPool (passes sheet object through streams) - "pattern offer a mechanism to reuse objects that are expensive to create. "[24]

Behavior Patterns - bahaviour patters are concerned with algorithms and the assignmens of resposnibilities between objects. Beahvioral pattersndescribe not just patterns of objects or classes but also the patterns of communication between them, These patterns characterize complex control flow thats difficult to tfollow at run time. They shift your focus away from flow of control to let you concentrate just on the way objects are interconnected"[44] Interpreter (translation from ts to js) - "The Interpreter is one of the Design Patterns published in the GoF which is not really used. Ussualy the Interpreter Pattern is described in terms of formal grammars, like it was described in the original form in the GoF but the area where this design pattern can be applied can be extended.

- Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the lan-

guage.

- Map a domain to a language, the language to a grammar, and the grammar to a hierarchical object-oriented design

"[24]
Strategy (multipiping/piping of streams) - "lets the algorithm vary independently from clients that use it."[24]
Observer (event emitor) - "The Observer Design Pattern can be used whenever a subject has to be observed by one or more observers."[24] Visitor - "

- Represents an operation to be performed on the elements of an object structure.
- Visitor lets you define a new operation without changing the classes of the elements on which it operates.

"[24]

6.2. Functional Programming

Functional Programming languages are languages which supports functions as a first-class citizens. Which mean that languge provides an opotunity to store them in data strucutures, pass and return them from functions they are higher-order functions[46].

Declarative languages in contrast to imperative ones are characterized as having no implicit state. Functional languages are declarative languages whose underlaying computational model is the function.[46]

The most fundamental influence developing of functional languages was the work of Alnso Church on lambda calculus. "Church's lambda calculus was the first suitable treatment of the computational aspects of functions."[46]

Introduction of lambda functions to Java SE 8 as a new and important feature[**?**] indicates the growing need of imperative programming benefits in an Enterprise Software development.

"Modeling with objects is powerful and intuitive, largely because this matches the perception of interacting with a world of which we are part. However, as we've seen repeatedly throughout this chapter, these models raise thorny problems of constraining the order of events and of synchronizing multiple processes. The possibility of avoiding these problems has stimulated the development of functional programming languages, which do not include any provision for assignment or mutable data. In such a language, all procedures implement well-defined mathematical functions of their arguments, whose behavior does not change. The functional approach is extremely attractive for dealing with concurrent systems." [37]

"John Backus, the inventor of Fortran, gave high visibility to functional programming when he was awarded the ACM Turing award in 1978. His acceptance speech (Backus 1978) strongly advocated the functional approach. A good overview of functional programming is given in Henderson 1980 and in Darlington, Henderson, and Turner 1982."[37]

7. Contributions

For particular implementation of Test Sheets paradigm were developed conventions conventions described below.

7.1. Requirements analysis

This section describes and analyses requirements differences defined by Test Sheets concept and introduced by figo GmbH.

Test Sheets were originally designed for test definitions of OOP languages. And all available examples describes tests definitions for Java Classes. While figo GmbH case requires implementation of tests for nodeJS/casperJS, which are based on JavaScript - Object Oriented, imperative, Functional Oriented programming language with asynchronous information flow. Moreover figo GmbH requires input of test results to be recorded in to LogStash logs database. The execution requirements from figo GmbH are such that tests defined via Test Sheets should be automatically executed in a time manner (every 2 mins or so), while normal software testing is performed on demand. The figo's requirement for comparison is such that while defining a Test Sheet user should be able to select from two types of comparison: 1) Strict comparison - complete comparison of objects including both properties' structure and their values. 2) Not-Strict comparison - scheme comparison of objects.

(Probably should go to different section) Scraping scripts implement callback based approach for handling asynchronous data flow. This provides an opportunity to perform result comparison within the custom callback defined on a implementation stage. Standard convention for callback definitions limitates number of input parameters of a callback (error, data), while comparison requires compare data parameter with expected outputs from Test Sheet. The opportunity to resolve this issue lays in a JavaScript's support of functions as a class citizens, the function implementation of module for comparison and report: module

exports function which invoked with single parameters (expected_output) / (+ script name) and returns function which is used as a callback for scrapping script, this callback function performs comparison and writes its result to TestSheet or logstash depending on environment in which the program was executed

7.2. Conventions

Following conventions should be followed for Test Sheet passed verification.

General:

- Number of columns within one TS should not exceed 26 columns (from A to Z);
- Invocation delimiters must be allocated within single column the (aligned to the longest row);
- References to the columns with expected returns columns will take as value actual return value obtained from method execution;
- Files extensions should be .xlsx

Basic Test Sheets

- A1 cell(optional) - description of the test case;
- A2 cell - module under testing with an extension (.js);
- A3..n - name of the class/object under the test;
- B3..n - name of the method from representative class (same row) under the test;
- C2..n to Invocation Column - input parameters for representative method (same row) under the test;
- Invocation Column - the column for separation of input values from expected output value(s) filled with — (pipe)(for comparison by scheme and data types) —— (two pipes)(for deep comparison - by scheme, data types and values) as a cells values until the last line which includes objects under tests;

- Expected Return - column(s) after invocation line.

Parameterized and Higher-Order Test Sheets Lower order test sheets can belong to Basic of Non-Linear types of Test Sheets and respectively follow conventions, with next additional option:

- Input and/or output cells can contain parameters ?[B-Z]+ which represent the value of cells within the representative column of Higher-Order Test Sheet
- Rows 1 and 2 should follow conventions for Basic Test Sheet;
- Cells starting from second row inside of [B-Z] columns should contain values which will replace parameters inside of Parameterized Test Sheet.

7.3. Use Case

definition (possibly in tabular form)

- Test Sheets defined by users (clients or employees without development background).
- Tests themselves will be applied for identification of layout changes on a target page before any interaction will appear to avoid errors and minimize the time of scripts correction.

Execution stages:

- Automated transformation of Test Sheets into JavaScript tests;
- Scheduled task for running tests on web pages;
- Developer notification regarding failing test.

The program run by user

7.4. Architecture

This system implements Pipe-and-Filter Architecture. [43] [41]

"In a pipe-and-filter style architecture, the computations components are called filters and they act as transducers that take input, transform it according to one or mode algorithms, and then output the result to communications conduit. The input and outputs coduits are called pipes.
The filters must be intedependent components. [...] The clasic example of pipe-and-filter architectural style is the Unix shell[...]"[43].

7.5. Design and Implementation

Consists of two streams Reader and Writer both streams are in object mode.

The system's information workflow described with following explanatory Test Sheet:

	A	B	C	D	E	F
1	get Accounts					
2	BankAustria					
3	BankAustria	login	<credentials object>	<pin object>	|	{...}
4	BankAustria	getAccounts	<credentials object>		||	{...}

7.5.1. Reader

On a lower level Reader stream consists of two combined streams (streams combination pattern used);

First stream takes input as a directrory and returns list of absolute paths to all files within provided directory (including nested folders); Second stream accepts output of a first stream and for all .xlsx files obtains its schema invoking function from schema_maker library and as an output returns object with absolute path to the Test Sheet file with file content returned by reading with object returned by reading file (object pool pattern) with *xlsx* library (`https://www.npmjs.com/package/xlsx`) and scheme created by schema_maker library.
Schema structure for expample Test Sheet:

- pathToFile: 'absloute/path/to/test/sheet/file';
- testsheet: { < object returned by xlsx library >};
- schema:

description: 'get Accounts',

moduleUnderTest: 'BankAustria',

objectsUnderTest: ['A3', 'A4'],

methodsUnderTest: ['B3', 'B4'],

inputs: ['C3', 'C4', 'D3],

outputs: ['F3', 'F4'],

invocations: ['E3', 'E4'],

Correspondence to design principles:

- Closing - stream is closed over file extension, schema_maker - object structure returned by *xlsx* library;
- Code to Interface - stream obtains and returns values via standard stream interface, call to file system made via standard nodeJS File System stream interface;
- Do not Repeat Yourself - no code duplication;
- Single Responsibility Principle - can be changed only due to the change of input type;
- Open Close Principle - new pipes can be added in a single place;
- Liscov Substitution Principle - no inherited objects used;
- Interface Segregation Principle - no dependency on redundant methods;
- Dependency Inversion Principle - higher level module index.js does not depend on current library
- Least Knowledge Principle - communication to interfaces and invocation of used library;
- Loose Coupling Principle - standard interfaces;

7.5.2. Writer

8. Implementierung

Zusammen mit dem Betreuer werden use-cases entwickelt anhand deren die Software programmiert werden soll. Diese dienen auch als Bewertungsgrundlage.

Die allgemein empfohlene Verzeichnisstruktur eines Projektes sieht wie folgt aus:

- projektname
  - bin
  - doc
  - lib
  - src

Die zu erstellende Software soll im package de.uni_mannheim.informatik.swt.`projektname` unter `src` liegen.

Bei der Programmierung sollte durchgängig die englische Sprache verwendet werden. Hierzu zählen insbesondere Kommentare im Quellcode, Namen von Funktionen, Variable, Menüpunkte im Benutzerinterface, kurze Hilfestellungen und Ausgaben von Programmen.

Der Code sollte mit Hilfe von `lstinputlisting` formatiert und ausgegeben werden, wie in folgendem Beispiel:

```
package de.uni_mannheim.informatik.swt.projektname;

import javax.servlet.http.HttpServletRequest;

import de.unimannheim.wifo3.cobana.action.ActionForm;
import de.unimannheim.wifo3.cobana.action.ActionMapping;

public class GuestbookForm extends ActionForm {
    private String name = null;
```

```
10      private String message = null;

        public GuestbookForm() {
        }

15      public String getName() {
            return this.name;
        }
        public void setName(String name) {
            this.name = name;
20      }

        public String getMessage() {
            return this.message;
        }
25      public void setMessage(String message) {
            this.message = message;
        }

        public void reset(ActionMapping mapping, HttpServletRequest
            request) {
30          setName(null);
            setMessage(null);
        }

    }
```

Listing 8.1: GuestbookForm.java

Bibliography

[1] About node.js®. https://nodejs.org/en/about/.

[2] Analysis of generators and other async patterns in node.

[3] The bank business model for apis: Identity. http://tomorrowstransactions.com/2015/03/the-bank-business-model-for-apis-identity/.

[4] Banks in germany.

[5] A case study of toyota unintended acceleration and software safety.

[6] Casperjs. http://casperjs.org/.

[7] Cucumber: Reference.

[8] Eur-lex - 32015l2366 - en - eur-lex. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32015L2366s.

[9] figo — angel list. https://angel.co/figo-1.

[10] figo — crunchbase. https://www.crunchbase.com/organization/figo#/entity.

[11] figo api reference. http://docs.figo.io/.

[12] Fints - wikipedia, free encyclopedia. https://en.wikipedia.org/wiki/FinTS.

[13] Fit.

[14] Fit.

[15] glog.isz.me - designing apis for asynchrony.

[16] Ich möchte mehr zu eurer vision erfahren! https://figo.zendesk.com/hc/de/articles/200974801-Ich-m%C3%B6chte-mehr-zu-eurer-Vision-erfahren-.

[17] Ieee sa - 829-2008 - ieee standard for software and system test documentation.

[18] Introduction to asynchronous programming.

[19] An introduction to libuv: Basics of libuv. http://nikhilm.github.io/uvbook/basics.html.

[20] Javascript. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[21] kriskowal/gtor: A general theory of reactivity.

[22] Lars markull's answer to what are the benefits of an api for a consumer bank? - quora. https://www.quora.com/What-are-the-benefits-of-an-API-for-a-consumer-bank/answer/Lars-Markull?srid=hmj2&share=267222bd.

[23] Module counts.

[24] Object oriented design patterns. http://www.oodesign.com/.

[25] Open banking apis: Threat and opportunity — consult hyperion. http://www.chyp.com/open-banking-apis-threat-and-opportunity/.

[26] The real cost of software errors.

[27] Test sheets. http://swt.informatik.uni-mannheim.de/de/research/research-topics/test-sheets/.

[28] Test sheets. http://swt.informatik.uni-mannheim.de/de/research/research-topics/test-sheets/basic-test-sheets/.

[29] W3c document object notation. https://www.w3.org/DOM/#what.

[30] Was ist figo und was macht ihr? https://figo.zendesk.com/hc/de/articles/200862101-Was-ist-figo-und-was-macht-ihr-.

[31] Web scraping - wikipedia free encyclopedia. https://en.wikipedia.org/wiki/Web_scraping.

[32] Welcome to the libuv api documentation. http://docs.libuv.org/en/v1.x/#features.

[33] Wer sind eure partner? wie kann ich figo dort nutzen? https://figo.zendesk.com/hc/de/articles/200907292-Wer-sind-eure-Partner-Wie-kann-ich-figo-dort-nutzen-.

[34] Why i am switching to promises.

[35] Xunit.

[36] xunit - wikipedia, free encyclopedia.

[37] Harold Abelson and Gerald J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, USA, 2nd edition, 1996.

[38] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[39] Colin Atkinson. L1-introduction. 2015.

[40] Colin Atkinson. L2-testingintroduction. 2015.

[41] Mario Casciaro. *NodeJS Design Patterns*. Packt Publishing, 2014.

[42] Érika Cota. Embedded software testing: What kind of problem is this? In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1486–1486, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[43] John Dooley. *Software Development and Professional Practice*. Apress, Berkely, CA, USA, 1st edition, 2011.

[44] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[45] Robert L. Glass. Real-time: The “lost world” of software debugging and testing. *Commun. ACM*, 23(5):264–271, May 1980.

[46] Paul Hudak. Conception, evolution, and application of functional programming languages. *ACM Comput. Surv.*, 21(3):359–411, September 1989.

[47] N. G. Leveson and C. S. Turner. An investigation of the therac-25 accidents. *Computer*, 26(7):18–41, July 1993.

[48] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2008.

[49] Robert Cecil Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[50] Gerard Meszaros. *xUnit test patterns refactoring test code*. Safari Books Online. Addison-Wesley, Upper Saddle River, N.J., 2007.

[51] Robert W. Sebesta. *Concepts of Programming Languages*. Addison-Wesley Publishing Company, USA, 9th edition, 2009.

[52] J. J. P. Tsai, K. Y. Fang, and Y. D. Bi. On real-time software testing and debugging. In *Computer Software and Applications Conference, 1990. COMPSAC 90. Proceedings., Fourteenth Annual International*, pages 512–518, Oct 1990.

Appendix

A. First class of appendices

A.1. Some appendix

This is a sample appendix entry.

Eidesstattliche Erklärung

Hiermit versichere ich, dass diese Abschlussarbeit von mir persönlich verfasst ist und dass ich keinerlei fremde Hilfe in Anspruch genommen habe. Ebenso versichere ich, dass diese Arbeit oder Teile daraus weder von mir selbst noch von anderen als Leistungsnachweise andernorts eingereicht wurden. Wörtliche oder sinngemäße Übernahmen aus anderen Schriften und Veröffentlichungen in gedruckter oder elektronischer Form sind gekennzeichnet. Sämtliche Sekundärliteratur und sonstige Quellen sind nachgewiesen und in der Bibliographie aufgeführt. Das Gleiche gilt für graphische Darstellungen und Bilder sowie für alle Internet-Quellen.

Ich bin ferner damit einverstanden, dass meine Arbeit zum Zwecke eines Plagiatsabgleichs in elektronischer Form anonymisiert versendet und gespeichert werden kann. Mir ist bekannt, dass von der Korrektur der Arbeit abgesehen werden kann, wenn die Erklärung nicht erteilt wird.

Mannheim, March 17, 2016 Unterschrift

Abtretungserklärung

Hinsichtlich meiner Studienarbeit/Bachelor-Abschlussarbeit/Diplomarbeit räume ich der Universität Mannheim/Lehrstuhl für Softwaretechnik, Prof. Dr. Colin Atkinson, umfassende, ausschließliche unbefristete und unbeschränkte Nutzungsrechte an den entstandenen Arbeitsergebnissen ein.

Die Abtretung umfasst das Recht auf Nutzung der Arbeitsergebnisse in Forschung und Lehre, das Recht der Vervielfältigung, Verbreitung und Übersetzung sowie das Recht zur Bearbeitung und Änderung inklusive Nutzung der dabei entstehenden Ergebnisse, sowie das Recht zur Weiterübertragung auf Dritte.

Solange von mir erstellte Ergebnisse in der ursprünglichen oder in überarbeiteter Form verwendet werden, werde ich nach Maßgabe des Urheberrechts als Co-Autor namentlich genannt. Eine gewerbliche Nutzung ist von dieser Abtretung nicht mit umfasst.

Mannheim, March 17, 2016 Unterschrift