UNIVERSITY OF MANNHEIM

Using Test Sheets for Real Time Testing in the case of figo GmbH

Master Thesis

submitted: June 2011

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

The figo GmbH's mission is to "build the backbone of next generation financial services"[9], company was founded in 2012 and has its headquarter in Hamburg. Total Equity Funding: $778.660 in 3 Rounds from 7 Investors (November 3, 2015). Currently, the API is fully functional in Germany and partly in Austria.[5][6]

figo Connect API was created to accelerate innovation in the FinTech area and to allow figo's partners to offer products with real added value. 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. [20][9][23]

Functions available through API:[7]

- 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);

List of partners of figo API with their use-cases: `http://figo.io/use_cases.html`

2.2. IT infrastructure

Information technology infrastructure of figo GmbH has two parts related to the external connections. **API Server** - implements interfaces to figo's customers

and partners for accessing banking inforamtion and services. **Banking Server** - implements connection to banks via three possible communication channels. This particular part lays in scope of this research as a part where implemented system is running. Below provided description introduces basic concepts of Banking Service Architecture. For more information regarding IT infrastructure please use (pics with arch scheme attached)

2.2.1. Banking Server Architecture

Custom-API. Introduction of Directive on Payment Services (PSD) and PSD2 by European Commission in European Union and initiatives of Government in United Kingdom regarding provision of API by Banks and its standartization obligated with providing of online access points to their services.[13][15][2] Some of the banks already implemented Application Program Interfaces to access their services. Some of which provide full functionality while some only partial. Within the Single European Payment Area acceptance of directive by European Bank Authority scheduled within 2017 year. [4] All this APIs very in their structure and functionality, and connection process lays out of the scope of this research.

Moreover some of the banks implements standardized programmable entry point.

HBCI+/FinTS - bank-independent protocol for online banking, developed and used by German banks. Home Banking Computer Interface (HBCI) 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.. The result of this effort was an open protocol specification, which is publicly available. The standardisation effort was necessary to replace the huge number of deprecated homemade software clients and servers (some of them still using BTX emulation). While IFX (Interactive Financial Exchange), OFX (Open Financial Exchange) and SET are tailored for the North American market, HBCI is designed to meet the requirements of the European market.[8]
Features[8]:

- Support for online-banking using PIN/TAN one time passwords.

- Support for online-banking with SWIFT.
- DES and RSA encryption and signatures.
- Making use of XML and SOAP for data-exchange, encryption and signatures.
- Implemented on top of HTTP, HTTPS and SMTP as communication layer.
- Multibanking: The software clients are designed to support accounts on multiple banking companies.
- Platform Independency: The specification allows software development for various types of clients.
- Storage of the encryption keys on an external physical device (smart card) for improved security.
- Possibility to use so called "Secoder" smart card readers to allow the user to cross check the transaction data on a secure device before signing it to uncover manipulations caused by malware. To use Secoder the bank as well as the home banking software have to support the Secoder protocol extension of FinTS

Low level vocabulary for message communication is defined by ISO20022 for more information please have a look to `https://www.iso20022.org/`

Web-Banking Engine. Since not all of the banks provide API nor HBCI figo uses web scraping technology (implemented within Web-Banking Engine) to access account's information and perform interaction with internet banking web page. From the banks perspective interaction completely looks like directly with user, while user does not feel the difference between interaction via Custom-API or HBCI or Web-Banking Engine.
In a same time 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 critical part to recognize such significant for scraping changes before any user's interaction and notify a developer regarding part of the script which failed. Exactly for this purpose testing scripts generated from Test Sheets are used as a (demon task/crown job) in real time fashion with recording unexpected behavior to the general log and notification of the developer (email/slack).

Next chapter provides general information regarding web scraping and high-level description of scraping tool used by figo Gmbh.

2.2.2. Web scraping

"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." [21]

"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." [19]

2.2.3. CasperJS

[3] 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

3. Testing

"Software testing has grown as an important technique to evaluate and help to improve software quality. Numerous techniques and tools have appeared in the last decade, ranging from static analysis to automatic test generation and application. One can argue that software is the dominant part of an embedded system, either as a final product (executable code) or during its development lifecycle (system modeling in specific languages and computation models). In both cases, software must be thoroughly verified to ensure product quality and reliability. One can observe a growing number of academic and industrial works on the topic of embedded SW testing in the last four or five years, and this seems to be a good time for reflection: how exactly is embedded software testing different from traditional software testing? Is it an engineering or computer science problem? Does it need extra support from platform developers? What is the role of the SW engineer and of the designer in developing a high-quality software-based embedded application? Many authors suggest that, on top of the ordinary software testing challenges, software usage in an embedded application brings additional issues that must be dealt with: the variety of possible target platforms, the different computational models involved during the design, faster time-to-market and even more instable and complex specifications, platform-dependent constraints (power, memory, and resources availability), etc. On the other hand, current platforms are more and more powerful, and the specificities of the embedded application can help to reduce the search space during test generation: application domains, strong code reuse paradigm, use of less advanced programming language resources, and common availability of system models subject to or already verified with respect to specific properties, for instance. Furthermore, a major part of the so-called embedded software does not depend directly on hardware, and one can argue that only a small percentage really needs to be tested together with the target platform, and this test is part of the platform design, not the system design. "[30]

Introduction of Agile and Extreme development paradigms with increased requirements for control over the existing code, its reuse and maintenance lifted requirements for tests and their quality to the new level and shifted responsibility regarding testing process from software engineers to people without development background. This led to new requirements for testing definition and representation tools since old tools like JUnit have high entry level for test engineers and requires its users to have an ability to write and read an executable code in a corresponding programming language.

"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. "[37]

"The ideas and techniques of software testing have become essential knowledge for all software delopers"[25]

Testing is responsible for at least 30% of the cost in a software project.[28]

NIST 2002 report, "The Economic Impacts of Inadequate Infrastructure for Software Testing" claimed that - inadequate software testing costs the US alone between $22 and $59 billion annually. [27]

Huge losses due to web application failures Financial services : $6.5 million per hour (just in USA!) Credit card sales applications : $2.4 million per hour (in USA) [27]

3.1. Test Driven Development

Test Driven Development is a software development paradigm which combines test-first development and refracting. This means that programmers first define test and only after write code to fulfill test requirements, both code and test maintained by developers. This reduces amount of redundant code and improves developers' confidence during refractoring process. Refactroing in TDD requires developer before introducing new feature first ask question the existing design fits best for implementation of new functionality.[10] [26]

"Consider the following three laws: 1)First Law You may not write production code until you have written a failing unit test. 2)Second Law You may not

write more of a unit test than is sufficient to fail, and not compiling is failing. 3)Third Law You may not write more production code than is sufficient to pass the currently failing test.

These three laws lock you into a cycle that is perhaps thirty seconds long. The tests and the production code are written together, with the tests just a few seconds ahead of the production code. If we work this way, we will write dozens of tests every day, hundreds of tests every month, and thousands of tests every year. If we work this way, those tests will cover virtually all of our production code. The sheer bulk of those tests, which can rival the size of the production code itself, can present a daunting management problem.[37]

Pros. As a result you will always be improving the quality of your design, thereby making it easier to work with in the future.[10] Excelent fault isolation. Support by big variety of test frameworks [35] Tests can replace code documentation at certain level of abstraction [37]

Cons. Very hard to perform TDD without testing framework. [35] Also [35] states among the limitations of TDD is its bottom-up nature which provides little oportunity for elegant design. In contrast [10] citates Robert C Martin with following words: "The act of writing a unit test is more an act of design than of verification. It is also more an act of documentation than of verification. The act of writing a unit test closes a remarkable number of feedback loops, the least of which is the one pertaining to verification of function" For the same bottom-up approach [35] claims that some of the faults can be tracked only by data flow testing.

3.2. Requirements for tests (F. I. R. S. T.)

Fast. Tests should be fast. They should run quickly. When tests run slow, you won't want to run them frequently. If you don't run them frequently, you won't find problems early enough to fix them easily. You won't feel as free to clean up the code. Eventually the code will begin to rot.[37]

Independent. Tests should not depend on each other. One test should not set up the conditions for the next test. You should be able to run each test independently and run the tests in any order you like. When tests depend on each other, then the first one to fail causes a cascade of downstream failures, making diagnosis difficult and hiding downstream defects.[37]

Repeatable. Tests should be repeatable in any environment. You should be able to run the tests in the production environment, in the QA environment, and on your laptop while riding home on the train without a network. If your tests aren't repeatable in any environment, then you'll always have an excuse for why they fail. You'll also find yourself unable to run the tests when the environment isn't available.[37]

Self-Validating. The tests should have a boolean output. Either they pass or fail. You should not have to read through a log file to tell whether the tests pass. You should not have to manually compare two different text files to see whether the tests pass. If the tests aren't self-validating, then failure can become subjective and running the tests can require a long manual evaluation.[37]

Timely. The tests need to be written in a timely fashion. Unit tests should be written just before the production code that makes them pass. If you write tests after the production code, then you may find the production code to be hard to test. You may decide that some production code is too hard to test. You may not design the production code to be testable.[37]

3.3. Dependability [36]

Dependability is defined as the trustwothiness of a computer system such that reliance can justifiably be place on the service it delivers. The service delivered by a system is its behaviour as it is perceptible by its users(s); a user is another system (human or physical) iwth the former.
Depening on applications inteded for the system, a different emphasis mau be put on the various facets fo dependability, that is dependablility can be veiwed

according to different, but complementary properties which enable the attributes of dependability to be defined:

- *The readiness for usage* leads to **availability**
- *The continuity of service* leads to **reliability**
- *The nonoccurence of catastrophic consequences on the environment* leads to **safety**
- *The nonoccirence of unauthorized disclosure of information* leads to **confidentiality**
- *The nonoccurrence of improper information* leads to **integrity**
- *The ability to undergo repeairs and evelutions* leads to **maintainability**

3.4. Real Time Testing

"Real-time software is a software thaht dries a computer which intercats with functioning external devices or objects. It is called real-time because the software actions control activities that are accuring in an ongoing process."[33] (MORE SHOULD BE ADDED)

3.5. Reliability testing

"Reliability is the probability that a system, or a system component, will deliver its intended functionality and quality for a specified period of "time", and under specified conditions, given that the system was functioning properly at the start of this "time" period. For example, this may be the probability that a real-time system will give specified functional and timing performance for the duration of a ten hour mission when used in the way and for the purpose intended. Since, software reliability will depend on how software is used, software usage information is an important part of reliability evaluation. This includes information on the environment in which software is used, as well as the information on the actual frequency of usage of different functions (or operations, or features) that the system offers. The usage information is quantified through operational profiles."[40]

4. Test Sheets

"Software testing is an important aspect of modern software development. Testing is performed to ensure that a product or component meets the requirements of all stakeholders. Just as the requirements themselves tests can vary widely in their nature. Some tests check whether executing certain code paths results in a correct state or answer while others may check whether a certain code path delivers its results in a specified amount of time.

However writing and evaluating tests is often not much different from programming itself. Tests are usually written in a formal programming language such as Java/JUnit. This necessitates that in order to create a test and understand its results knowledge of a formal programming language is required. This is a significant barrier of entry for stakeholders without a background in IT even though they may be interested in the tests themselves. Even without deep IT knowledge a stakeholder may still be interested in how well a product performs with regard to her requirements.

Visual test representations such as the UML Testing Profile try to lower the barrier of entry into testing. However most of the time these visualizations are only partial descriptions of the tests and so do not contain all the desired information for evaluation.

The Software Engineering group has started to develop a new representation for tests called Test Sheets. The goal is to create a way to define tests which combines 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. This is achieved by representing a test in tabular form as a spreadsheet. Rows in a Test Sheet represent operations being executed while the columns represent input parameters and expected results. The actual content of a cell can be made dependent on other cells by addressing them via their location. This works in a way similar to existing spreadsheet software such as Microsoft Excel. After executing a test the cells for each expected result is colored according

to the result of the test. A successful test causes cells to become green while failed tests are indicated by red cells."[16]

4.1. Basic Test Sheets

"A Test Sheet consists of a name and a class being tested. Each row after that represents one method call. The first column identifies the object being tested while the second column indicates which method is being called on said object. [...] Input parameters are specified in the columns following the method name up to the invocation line. Right after the invocation line the expected return values can be specified."[17]

4.2. High Order, Parameterized Test Sheets

"The actual value used for Parameterized Test Sheets is specified by a Higher-Order Test Sheet as in the example below. The Higher-Order Test Sheet references the Parameterized Test Sheet as the 'class' being tested. On said pseudo-class it invokes the pseudo-method test followed the by the value to be used as parameter. ?C in the Parameterized Test Sheet is replaced by the value defined the third column (column C) for each execution. It is also possible to use more than one parameter. These are defined in the Higher-Order Test Sheet in subsequent columns (D, E, F, etc.) and referenced in the Parameterized Test Sheet via ?D, ?E, ?F, etc"[18]

5. NodeJS

"As an asynchronous event driven framework, Node.js is designed to build scalable network applications. Node is similar in design to and influenced by systems like Ruby's Event Machine or Python's Twisted. Node takes the event model a bit further, it presents the event loop as a language construct instead of as a library." [1]

"Node.js is considered by many as a game-changer—the biggest shift of the decade in web development.[...]
First, Node.js applications are written in JavaScript, the language of the web, the only programming language supported natively by a majority of web browsers. [...]
The second revolutionizing factor is its single-threaded, asynchronous architecture. Besides obvious advantages from a performance and scalability point of view, this characteristic changed the way developers approach concurrency and parallelism. [...]
The last and most important aspect of Node.js lies in its ecosystem: the npm package manager, its constantly growing database of modules, its enthusiastic and helpful community, and most importantly, its very own culture based on simplicity, pragmatism, and extreme modularity. "[29]

"JavaScript (JS) is 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."[12]

Non blocking I/O A set of bindings responsible for wrapping and exposing libuv and other low-level functionality to JavaScript.[29]

Non blocking I/O in NodeJS is provided by libuv[1][29]. Which is "libuv is a multi-platform support library with a focus on asynchronous I/O. It was primar-

ily developed for use by Node.js, but it's also used by Luvit, Julia, pyuv, and others."[22]
libuv properties[11]:

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

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

V8/Chakra the JavaScript engine originally developed by Google for the Chrome browser/ Microsoft for IE 9 browser"[29]

Event handling NodeJS asynchronous nature provided by event handler which is an implementation of reactor pattern. Here is the description of process lifecycle[29]:

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.

5.1. Approaches application flow

Callbacks

Async

Promises

Fibers

Generators

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. "[39]

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.[38]

Symptoms of not agile design [38]:

- 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[31][38] :

- 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. "[32]

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

"[14]

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."[14]

ObjectPool (passes sheet object through streams) - "pattern offer a mechanism to reuse objects that are expensive to create. . "[14]

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"[32] 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 language.
- Map a domain to a language, the language to a grammar, and the grammar to a hierarchical object-oriented design

"[14]
Strategy (multipiping/piping of streams) - "lets the algorithm vary independently from clients that use it."[14]
Observer (event emitor) - "The Observer Design Pattern can be used whenever a subject has to be observed by one or more observers."[14] Visitor (callbacks in JS) - "

- 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.

"[14]

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[34].

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.[34]

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."[34]

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." [24]

"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."[24]

6.3. Streams

"A stream is an abstract interface implemented by various objects in Node.js." [**?**] "Dominic Tarr (one of top contributors to the Node.js community [**?**]), defines streams as node's best and most misunderstood idea."[29]

Streams are the classic example of Pipe-and-filter architecture.

"In an event-based platform such as Node.js, the most efficient way to handle

I/O is in real time, consuming the input as soon as it is available and sending the output as soon as it is produced by the application."[29]

- Spatial efficiency
- Time efficiency
- Composability

6.3.1. Anatomy of Streams

"Every stream in Node.js is an implementation of one of the four base abstract classes available in the stream core module:

- stream.Readable
- stream.Writable
- stream.Duplex
- stream.Transform

Each stream class is also an instance of EventEmitter. Streams, in fact, can produce several types of events, such as end, when a Readable stream has finished reading, or error, when something goes wrong.

One of the reasons why streams are so flexible is the fact that they can handle not only binary data, but practically, almost any JavaScript value; in fact they can support two operating modes:

- Binary mode: This mode is where data is streamed in the form of chunks, such as buffers or strings;
- Object mode: This mode is where the streaming data is treated as a sequence of discreet objects (allowing to use almost any JavaScript value).

Readable streams. A readable stream represents a source of data; in Node.js, it's implemented using the Readable abstract class that is available in the stream module.

Writable streams. A writ[e]able stream represents a data destination; in Node.js, it's implemented using the Writ[e]able abstract class, which is available in the stream module.

Duplex streams. A Duplex stream is a stream that is both Readable and Writ[e]able. It is useful when we want to describe an entity that is both a data source and a data destination, as for example, network sockets. Duplex streams inherit the methods of both stream.Readable and stream.Writable, so this is nothing new to us. This means that we can read() or write() data, or listen for both the readable and drain events.

Transform streams. The Transform streams are a special kind of Duplex stream that are specifically designed to handle data transformations. In a simple Duplex stream, there is no immediate relationship between the data read from the stream and the data written into it (at least, the stream is agnostic to such a relationship). On the other side, Transform streams apply some kind of transformation to each chunk of data that they receive from their Writable side and then make the transformed data available on their Readable side. From the outside, the interface of a Transform stream is exactly like that of a Duplex stream. However, when we want to build a new Duplex stream we have to provide the read() and write() methods while, for implementing a new Transform stream, we have to fill in another pair of methods: transform() and flush()."[29]

6.3.2. Piping patterns

"As in real-life plumbing, Node.js streams also can be piped together following different patterns; we can, in fact, merge the flow of two different streams into one, split the flow of one stream into two or more pipes, or redirect the flow based on a condition. In this section, we are going to explore the most important plumbing techniques that can be applied to Node.js streams."[29]

- Combining streams - encapsulation of sequentualy connected streams in to single looking stream with single I/O points and single error handling mechanism by pipeing readable stream in to writable stream.[29]
- Forking streams - piping single readable in to multiple writable streams.[29]
- Merging streams - piping multiple readable streams in to single writable stream.[29]
- Multiplexing and demultiplexing - forking and merging pattern which provides shared communication chanel for entities from different sreams.[29]

7. Contributions

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

7.1. 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)
- file extension .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.

7.2. 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

7.3. Architecture

This system implements Pipe-and-Filter Architecture. [31] [29]

"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[...]"[31].

7.4. 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>		||	{...}

7.4.1. 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;

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

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

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

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

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

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

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

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

[10] Iintroduction to test driven development (tdd). http://www.agiledata.org/essays/tdd.html.

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

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

[13] 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.

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

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

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

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

[18] Test sheets. http://swt.informatik.uni-mannheim.de/de/research/research-topics/test-sheets/parameterized-and-higher-order-test-sheets/.

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

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

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

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

[23] 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-.

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

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

[26] Dave Astels. *Test Driven Development: A Practical Guide*. Prentice Hall Professional Technical Reference, 2003.

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

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

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

[30] 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.

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

[32] 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.

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

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

[35] P.C. Jorgensen. *Software Testing: A Craftsman's Approach, Fourth Edition*. An Auerbach book. Taylor & Francis, 2013.

[36] Michael R. Lyu, editor. *Handbook of Software Reliability Engineering*. McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.

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

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

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

[40] Mladen A Vouk et al. Software reliability engineering. 2000.

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, February 1, 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, February 1, 2016 Unterschrift