OULUN YLIOPISTO
UNIVERSITY of OULU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Petri Niemelä

DYNAMIC FUNCTIONAL END-TO-END TESTING IN THE CASE OF SAP E-COMMERCE

Master's Thesis
Degree Programme in Computer Science and Engineering
October 2014

Niemelä P. (2014) Dynamic functional end-to-end testing in the case of SAP e-commerce. University of Oulu, Department of Computer Science and Engineering. Master's Thesis, 74 p.

ABSTRACT

Software testing is an important part of software development projects. As the role of information technology (IT) becomes bigger and bigger in our everyday activities, it is clear that business operations and human well-being are dependent on information systems. To efficiently operate and run a business, companies reflect their processes to IT systems. A business process can cover many different organizational units, both in real life and in the IT system. Organizational units can have their own separate IT system modules implemented, and data flows from module to module via interfaces. To ensure the correct functionality of the business process, end-to-end testing of the complete process across the IT systems is required.

With the advancement of technology, it has been a trend to replace human work with machines. Same applies to software testing, where repetitive testing tasks and otherwise manually unfeasible test activities are automated to be run by a machine. To achieve this, a test automation tool needs to be able to simulate real usage in the system under test. As systems consist of multiple modules and technologies, it is a challenge for the test tool to support such a technical variety. In many companies, such a heterogeneous system landscape includes software implemented by SAP AG, one of the world's largest software manufacturers.

This work presents an end-to-end business process test automation library for an SAP e-commerce environment. The test library enables to extend the normal test automation of a web shop to cover the back-end processing of the SAP system as well. This is achieved by building a test library on top of SAP's communication methods. The test library is driven from a common keyword-driven test automation framework, Robot Framework. In this work, the related research and technologies for the implementation are discussed and presented. The design is demonstrated, and the implementation process is described in detail. Other known approaches to SAP test automation are introduced, and when compared, no other similar test tools were found available with such ease of operational deployment. Test results and live project usage of the test library show that the library works as expected. The performance is also promising, not having a noticeable impact on the total test execution duration. There are a lot of future development possibilities to further extend the usage of the test library in SAP test automation.

Keywords: SAP, test automation, Robot Framework

Niemelä P. (2014) Dynaaminen toiminnallisuuden E2E-testaus SAP-verkkokaupankäynnissä. Oulun yliopisto, Tietotekniikan osasto. Diplomityö, 74 s.

TIIVISTELMÄ

Ohjelmistotestaus on tärkeä osa ohjelmistokehitysprojekteja. Tietotekniikan roolin kasvaessa päivittäisessä asioinnissa on selvää, että liiketoiminta sekä ihmisten hyvinvointi ovat riippuvaisia informaatiojärjestelmistä. Yritykset heijastavat liiketoimintaprosessinsa tietojärjestelmiin tehostaakseen liiketoiminnan harjoittamista. Yksi liiketoimintaprosessi voi kulkea usean eri organisaatioyksikön läpi, sekä tosielämässä että IT-järjestelmässä. Organisaatioyksiköillä voi olla erilliset IT-järjestelmämoduulit toteutettuina, ja tieto välittyy moduulien välillä rajapintojen kautta. Liiketoimintaprosessi on testattava päästä päähän koko informaatiojärjestelmässä oikean toiminnallisuuden varmistamiseksi.

Tekniikan kehittyessä suuntauksena on ollut ihmistyön korvaaminen koneilla. Sama pätee myös ohjelmistotestaukseen, jossa toistuvat testaustehtävät sekä muutoin manuaalisesti toteuttamiskelvottomat testausaktiviteetit automatisoidaan koneella suoritettavaksi. Tämän saavuttamiseksi testiautomaatiotyökalun on pystyttävä simuloimaan oikeaa käyttöä testattavassa järjestelmässä. Järjestelmät koostuvat useista moduuleista sekä teknologioista, joten on haaste saada testiautomaatiotyökalut tukemaan järjestelmien teknistä vaihtelevuutta. Monissa yrityksissä teknisesti heterogeeninen järjestelmäympäristö sisältää ohjelmistoa, jonka toimittaja on SAP AG, yksi maailman suurimmista ohjelmistovalmistajista.

Tämä työ esittelee liiketoimintaprosessien päästä päähän testaukseen suunnatun testiautomaatiokirjaston SAP:n verkkokauppaympäristöille. Testiautomaatiokirjasto mahdollistaa normaalin verkkokaupan testiautomaation kattavuuden ylettymään myös SAP:n taustajärjestelmään. Tämä saavutetaan kehittämällä testikirjasto SAP:n kommunikaatiomenetelmiin perustuen. Testiautomaatiokirjastoa suoritetaan avainsanaohjautuvalla Robot Framework –testiautomaatiokehyksellä. Tässä työssä esitellään aiheeseen liittyvää tutkimusta sekä teknologiaa. Testiautomaatiokirjaston suunnittelu esitellään, sekä toteutus kuvataan yksityiskohtaisesti. Muita tunnettuja lähestymistapoja SAP:n testiautomaatioon esitellään. Kirjallisuuskatsauksessa ei löydetty vastaavaa ratkaisua, joka olisi yhtä vähäisellä vaivalla saatu operatiiviseen käyttöön. Testitulokset ja tuotannollinen projektikäyttö osoittavat, että testiautomaatiokirjasto toimii kuten odotettu. Suorituskyky on myös lupaava, eikä automaatiotestien kokonaiskestoon nähty merkittävää hidastusta. Testiautomaatiokirjastolle on paljon tulevaisuuden kehitysmahdollisuuksia, joilla kirjaston käyttöä SAP:n testiautomaatiossa voidaan laajentaa.

Avainsanat: SAP, testiautomaatio, Robot Framework

TABLE OF CONTENTS

PREFACE

This Master's Thesis was implemented in Bilot Oy, which is an SAP partner company. First and foremost, I would like to thank the whole staff of Bilot for their excellent support and understanding throughout the project in the years of 2013 and 2014. It has been very rewarding to work with them.

A special thanks belong to my instructor Doctor Jakub Rudzki, who provided the topic, gave me feedback, had patience and provided guidance when needed.

From the Department of Computer Science and Engineering, I want to express my gratitude to Professor Jukka Riekki for the supervision he provided for this Master's Thesis. I would also like to thank the second supervisor Doctor Mika Rautiainen for his efforts.

Finally, I would like to thank my family and my friends for the constant encouragements. Thank you.

Helsinki, October, 2014

Petri Niemelä

LIST OF ABBREVIATIONS AND SYMBOLS

ABAP	Advanced Business Application Programming
AG	Aktiengesellschaft
API	Application Programming Interface
B2B	Business-to-Business
B2C	Business-to-Consumer
BPCA	Business Process Change Analyzer
CBTA	Component Based Test Automation
CPU	Central Processing Unit
CRM	Customer Relationship Management
eCATT	Extended Computer Aided Test Tool
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
HTML	HyperText Markup Language
IEC	International Electrotechnical Commission
ISA	Internet Sales
ISO	International Organization for Standardization
IT	Information Technology
I/O	Input/Output
JAR	Java Archive
MDM	Master Data Management
RFC	Remote Function Call
RPC	Remote Procedure Call
SAP	Systems, Applications, and Products in Data Processing
SAP JCo	SAP Java Connector
SLOC	Source Lines Of Code
SUT	System Under Test
TAO	Test Acceleration and Optimization
TREX	Text Retrieval and information EXtraction
UI	User Interface
WCEM	Web Channel Experience Management
XML	EXtensible Markup Language

1. INTRODUCTION

In today's world, a majority of our normal everyday activities are controlled by information systems. Cars, medical equipment, household appliances; an increasing amount of products and commodities contain software. Business-to-consumer and business-to-business services are more and more providing an electrical alternative via internet for running errands, and some companies have completely moved their customer interaction online. It is clear that information technology has an important role in running businesses.

Information systems are not perfect. They contain errors. These systems are built by humans, and humans are fallible. As software evolves into complex information systems containing millions of lines of programming code, it is also natural that the amount of errors increase. Errors manifest themselves as malfunctions or unexpected behavior in the system. Sometimes these errors are minor cosmetic glitches, but many bugs have costly effects, even endangering human lives.

Companies' businesses are dependent on their information systems, and so is human well-being. Thus, it is crucial that the software functions correctly and as expected, and that critical errors in the system are found and corrected before the software is moved from testing environment into productive usage. Quality assurance and software testing are the main activities in trying to find those errors.

No matter what software development methodologies are used in developing a software, they all share a common characteristic; at some point of the software life cycle, be it development, maintenance or some other phase, programmatic changes are introduced into the system. The number of functions, components and interfaces will grow in the software, requirements are updated or existing errors need to be fixed; modifications to the software cannot be avoided [1]. While introducing these changes into a partially completed or ready-made system, there is a risk that the change breaks a previously working functionality. This phenomenon is called software regression. Identifying all the effects a change can have to a system can be very difficult, and therefore the previously working functionalities have to be re-tested. Regression testing is performed to ensure that the updated software still has the functionality it had before it was updated [2, p.7].

From business perspective, it is crucial that the business processes work as intended in the information system. To ensure this, the testing should focus in validating the business processes of the company. In a multi-dimensional organization, also the business processes cover multiple departments and organizational units. This is reflected to the IT system with different modules, each serving the department's specific tasks. Data flows from module to module via interfaces. To ensure the correct functionality of a business process across the various modules, the business process is tested from one end to the other. This is known as end-to-end testing [3].

With the advancement of technology, it has been a trend to replace human work with machines. Same applies to software testing. Testing performed by another software is called test automation. Implementing test automation takes time and effort, thus it is not always feasible. Good tests to consider for automation are the ones that are run multiple times, such as regression tests [4, p.248]. It has to be analyzed if the benefits of developing the automated tests outweigh the situation where the testing would be performed manually. Test automation is also utilized in situations where it would not be practical to perform the tests manually. An example of this would be testing the performance of a system with hundreds or even thousands of simultaneous users.

When creating test automation, the machine has to be explicitly instructed via scripts to perform actions. This becomes an issue with test data, as the test automation engineer has to define what will be used as data to perform the tests. Often the test data is part of the test scenario, hard coded into the test scripts. This makes the test data inflexible and difficult to reuse. An example scenario would be testing user registration to a web shop, which only accepts unique email addresses for its users. Testing the scenario with an automated test script twice would result in test data conflict, because the system would not accept the second registration with the same email address. Thus, the test case would fail. One way to avoid such a situation is to add a test tear down to the automated test script. The purpose of the tear down is to restore the system to a similar state that it was before the test was executed, and the automated test script could be run again without a conflict in the test data.

Using pre-defined, hard coded test data is not an optimal way of testing functionalities because of the invariability of the test data. This could result in not caching some errors due to always performing the test with the same data sets. Additionally, it is not easy to provide valid test data that would cover many scenarios and environments. In SAP (Systems, Applications, and Products in Data Processing) systems development, there are usually multiple environments used: one for development, one for testing, optionally one for customer's own quality assurance, and lastly the production environment. To ensure the system works well in each environment, the test data has to be updated. In these cases when testing the functionalities, it would be much better to use the system's own database as a data source to fetch the data to be used in the automated test scripts.

The purpose of this Master's Thesis is to implement a test automation library for SAP software systems. From SAP systems, the scope of the test library is in SAP e-commerce. The test library would fetch valid test data from the SAP back-end to be used in automated test scripts. In this case, SAP back-end can mean SAP Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) or Master Data Management (MDM) systems, but the scope of the thesis is SAP ERP. The data would be fetched automatically, without human intervention, and dynamically during test execution. Based on the test data, the test library would also make assertions about the outputs of the SAP solution under test. Moreover, the test library would be easy to deploy into use in a changing environment by not requiring any installations to the SAP system landscape.

The structure of this thesis is the following: Chapter 2 gives an overview of software testing and the key topics related to the implementation. In Chapter 3, the target environment is presented in more detail, as well as tools and topics related closely to the implemented test automation library. Moreover, an overview of known existing approaches is given. Chapter 4 defines the specific requirements to be fulfilled, and a design of the implementation is presented. The implementation process is described in detail and the development results are presented. Chapter 5 summarizes the testing of the implementation, using the requirements as basis. The results of the testing are analyzed in Chapter 6, with a comparison to the initial objectives. Personal experiences and future development ideas are also discussed. Finally, Chapter 7 presents the summary of the Master's Thesis.

2. SOFTWARE TESTING

This chapter gives an introduction to software testing and the basic principles that revolve around testing. The fundamentals are presented but the main focus is to introduce the topics related to the thesis. After the basic topics, test automation is covered with a description of different test automation approaches.

2.1. Software testing in general

Software testing is the primary method for evaluating a software under development [2, p.3]. Testing is a part of software development life cycle, and its main focus is to find defects in the test object. A defect is the result of an error in the software code or documentation, and test object is the object which is being tested [5, p.553]. As software grows more complex every day, it is a fact that everything regarding the software's correctness cannot be tested. This is simply because the possible testable combinations grow so large that it is practically impossible to cover all of them in testing, let alone having the resources to do that. Decisions regarding test coverage have to be made based on business processes to identify the areas in need of testing, as well as test thoroughness. [5] [6]

Testing is not just checking that the software functions correctly. Testing is more than that, and it can be defined as follows [5, p.36]:

> "Testing is a process that provides insight into, and advice on, quality and the related risks."

The test object undergoing testing can be an information system or a part of it, such as software, hardware, documentation, procedure or organization. Risk can be described as a harmful event, which has a probability of realization and causes damage to business when realized. Risks can be determined via risk analysis, and a "no risk, no test" principle can be applied. Product risk can be formulated as follows [5, p.472]:

> Product risk = Chance of failure * Damage, where
> Chance of failure = Chance of defects * Frequency of use.

Quality, on the other hand, is not as straightforward to define as test object or risk. Even if an information system functions correctly and is capable of performing all the actions that were expected, it does not mean unambiguously that the system is of good quality. If the information system is sluggish, does not look pleasing to the eye or is extremely complicated to learn, then it is hard to say the system is of high quality. Also, quality can be subjective, as users value different things in a software. Because of the diversity of quality, ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) have defined standards for software quality. In the ISO/IEC 9126 –standard, quality has been described with different quality characteristics. Koomen et al. [5, p.495-501] have refined quality characteristics to fit more into software testing perspective. These quality characteristics are presented in Table 1.

Table 1. Quality characteristics [5, p.495-501]

Quality characteristic	**Description**
Connectivity	The ease with which an interface can be created with another information system or within the information system, and can be changed.
Continuity	The certainty that the information system will continue without disruption, i.e. that it can be resumed within a reasonable time, even after a serious breakdown.
Data controllability	The ease with which the accuracy and completeness of the information can be verified (over time).
Effectivity	The degree to which the information system is tailored to the organization and the profile of the end users for whom it is intended, as well as the degree to which the information system contributes to the achievement of the company goals.
Efficiency	The relationship between the performance level of the system (expressed in the transaction volume and the total speed) and the volume of resources (CPU cycles, I/O time, memory and network usage, etc.) used for these.
Flexibility	The degree to which the user is able to introduce enhancements or variations on the information system without amending the software.
Functionality	The degree of certainty that the system processes the information accurately and completely.
(Suitability of) infrastructure	The appropriateness of the hardware, the network, the system software, the database management system and the (technical) architecture in a general sense to the relevant application and the degree to which these infrastructure elements interconnect.
Maintainability	The ease with which the information system can be adapted to new requirements of the user, to the changing external environment, or in order to correct faults.
Manageability	The ease with which the information system can be placed and maintained in an operational condition.
Performance	The speed with which the information system handles interactive and batch transactions.
Portability	The diversity of the hardware and software platform on which the information system can run, and the ease with which the system can be transferred from one environment to another.
Reusability	The degree to which parts of the information system, or of the design, can be used again for the development of other applications.
Security	The certainty that consultation or mutation of the data can only be performed by those persons who are authorized to do so.

Quality characteristic	Description
Suitability	The degree to which the manual procedures and the automated information system interconnect, and the workability of these manual procedures for the organization.
Testability	The ease and speed with which the functionality and performance level of the system (after each adjustment) can be tested.
User-friendliness	The ease of operation of the system by the end users.

Based on the quality characteristics, it is more convenient to measure quality and to define what needs to be tested. The scope of the test library implemented in this thesis is in testing the functionality characteristic of the system under test (SUT). Furthermore, functionality can be described as the degree to which the system processes the supplied input and mutations correctly, according to the specifications, into consistent data collections and output [5, p.498].

2.2. Software testing fundamentals

In every software project, testing is involved. A common process is to structure software testing into test levels, and different types of testing is carried out. Next, an introduction to test levels is given, and a couple of important test types from the aspect of business importance and test automation are introduced.

2.2.1. Test levels

When considering system development process from the perspective of testing, there are two groups involved; the accepting party and the supplying party [5, p.46]. The accepting party is the client who made the request for a software system. The supplying party is the party developing the software system. This makes two general aims for software testing [5, p.47]:

1. The supplying party demonstrates that the supplied implementation fulfills the requested requirements.
2. The accepting party verifies whether what has been requested has actually been received.

A much used concept in software development and testing is the V-model. There are multiple different versions and interpretations of the V-model [5, p.47] [7, p.127] [8, p.5] [2, p.30], but the common idea is to list development phases to the left side of the V-model, and test levels to the right side. Test levels group together a group of testing activities they are created for each software development phase, structuring the test cycle to model the development cycle [5, p. 47] [7, p.127]. Testing is done at the right side of the V-model, using the development phases as a basis for testing. Test levels also divide the testing responsibilities between the supplying party and the accepting party [5, p. 47].

Figure 1 shows Koomen et al. [5, p.48] interpretation of the V-model, which has three test levels defined; development tests, system tests and acceptance tests. Development tests and system tests are placed under the responsibility of the supplying party and the acceptance tests are performed by the accepting party. Inputs from the development phases functioning as the test basis for the test levels are also shown in the figure.

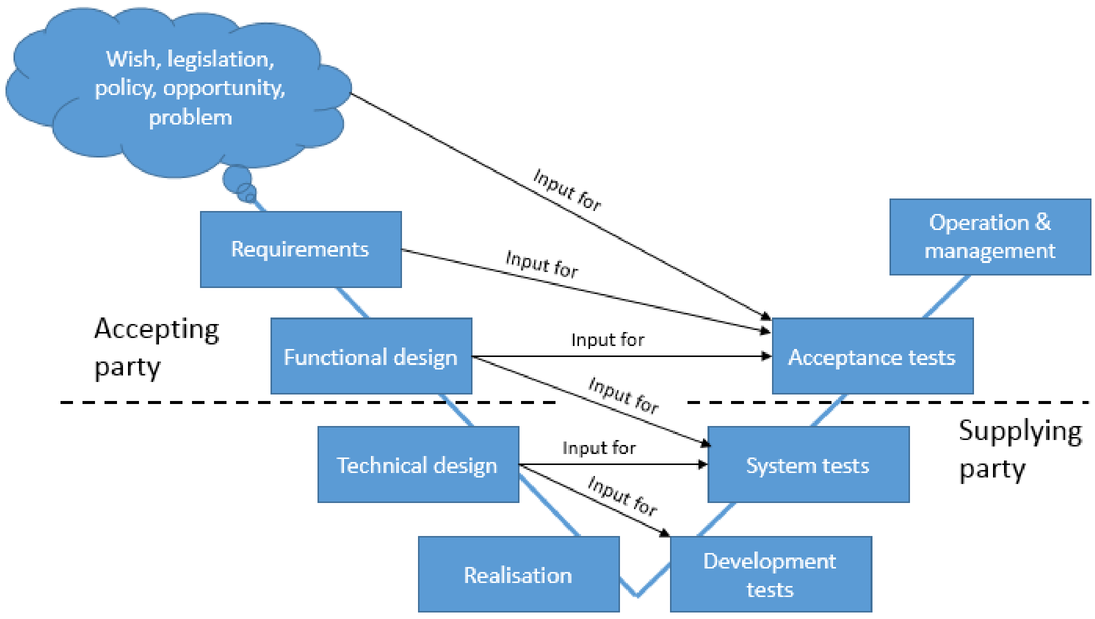


Figure 1. V-model.

Development tests consist of unit testing (also known as component or module testing). Typically in unit testing, the programmer who wrote the component's code is involved with access to the program code. Defects are usually fixed as soon as they are found, without going through the formal defect management procedures. The test basis for unit testing is derived from the technical design. [9, p.24]

The aim in system tests is for the supplying party to demonstrate that the product meets the specifications made by the accepting party. System testing tests the behavior of a whole system/product, and it should test the quality characteristics of the system against the functional and non-functional requirements specified in the product's functional design. [5, p.48] [9, p.26]

In acceptance tests, the accepting party verifies that the product meets their expectations [5, p.48]. The goal is to establish confidence in the product and assess the system's readiness for deployment and business usage [9, p.26]. Acceptance testing must involve business users and subject matter experts who have strong domain knowledge of the business processes [2, p.30]. Business users also participate in evaluating the test results. This ensures that the system is assessed in real-world situations and the test coverage is for the full range of business usage [8, p.10].

2.2.2. Test types

Test types are a group of test activities which aim to check the system under test in respect of one or multiple quality characteristics, which were presented in Table 1. Test type can also aim to test a subset of a quality characteristic. For example, in performance testing the tester could perform load testing, where the system is put under heavy load and expected to still perform the tasks that were specified, or stress testing to check how the system behaves under extreme conditions. [5, p.50] [7, p.133-134]

The scope of the thesis is to implement a testing tool which specifically concentrates on testing the functionality of the system under test with a combination of end-to-end regression testing. There is plenty of literature in testing books regarding test types. Therefore, only end-to-end testing and regression testing are introduced in this section.

End-to-end testing

End-to-end testing is used to test the flow of an application by performing complete processes from the start to finish. The purpose is to ensure that the overall process flows as expected. This is achieved by checking that the system components integrate correctly and verifying that the right information is passed between the components. [3]

Companies' functional units can grow big, covering complex business processes, and disperse to a wide are geographically. To efficiently run such a business, the IT systems needs to support the business processes. Performing end-to-end business process testing is needed to verify a correct flow of information in complex IT systems. Organizing and coordinating such a task in multiple different business units can prove challenging.

Regression testing

Regression testing is retesting of software [10, p.176]. Regression tests are run because when programmers make changes to the system, they may break parts of the program that used to work [11]. The goal is to ensure that the new version of the software still possesses the capabilities of the old version and that no new errors have been introduced due to the changes [10, p.176].

With the increasingly popular agile software development methods, where software deliveries are done rapidly in an iterative manner, the role of regression testing is emphasized [12]. Because regression tests need to be run multiple times, it is a good idea to automate them [11, p.143]. In many cases, the cost of automating regression tests return the investment when compared to time consuming manual testing.

2.3. Automated software testing

Kenneth White [13] defines automated software testing as follows: "Automated Software Testing is using one program to 'drive' another. It does this either by mimicking a human user through the User Interface (UI) or by interacting directly with the source code via an Application Programming Interface (API)." Such a driver program is called automation tool, or test library. Software test automation runs test cases, performs actions with the system under test, validates the outputs of the system and logs the results [14]. This is all done without human intervention. Before a testing tool can do the testing, it needs to be specifically instructed what exactly needs to be done. This is done manually with test scripts [13]. The implementation techniques for automated test scripts can be roughly divided into two methods; using capture-playback tools or manual programming [14].

This section aims to shortly introduce the different approaches for automated software testing. Manual test script programming opens up more powerful options than capture-playback tools, and a special introduction is given on data-driven test automation and keyword-driven test automation approaches. The concept of test automation framework is also introduced.

2.3.1. Capture-playback tools

Capture-playback tools do basically what the name implies. The user starts to record a session with the tool, and the tool captures every keystroke, mouse movement and click that the user performs during the session. Once finished, the tool stores all the actions performed into a test script. The generated test script can be played back to automatically test the application later when necessary. [4] [13]

Creating test scripts with capture-playback tools can be fast and requires only little, if any, technical skills from the tester, but they often result in test script maintenance issues. To perform a small change into the test script, or if the application's user interface changes even slightly, it is often required to fully re-record the test script from the beginning. Test automation scripts in general always require maintenance, but with capture-playback tools, the needed maintenance usually overwhelms the gained benefits in the long run. Test data is also hard coded into the test script, and provides no means for test data variability. [13] [15, p.50-57]

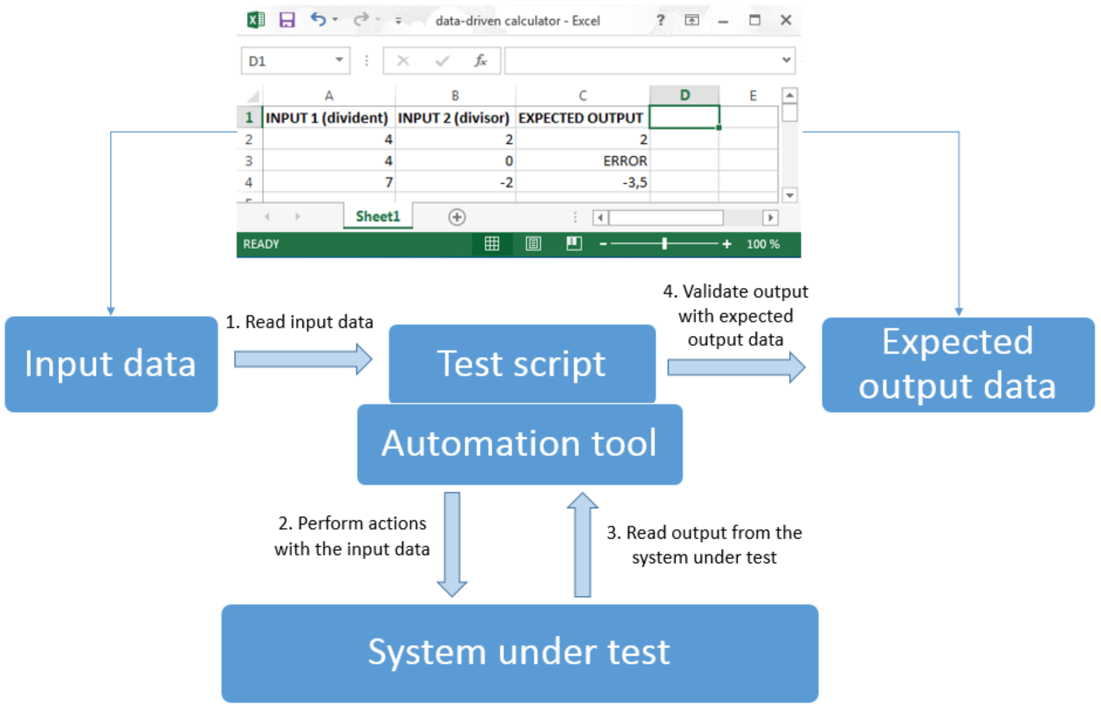
2.3.2. Manual programming

A more flexible technique for implementing test automation is manual programming of the test scripts for the automation tool. At a low level, it is comparable to software development where programming or scripting language is used [16, p.65]. With good software development practices utilized, the maintainability of the automated tests increase, reducing the work effort needed after initial implementation. Maintainability of automated tests is one key factor for achieving a positive return on investment for repetitive testing tasks when compared to traditional manual testing. Utilizing manual test script programming and levels of abstraction, a more sophisticated approach to test automation can be achieved.

2.3.3. Data-driven test automation

Data-driven test automation can be described as taking a step further from basic manual test script programming. The principle in data-driven testing is to differentiate the actual test data from the test scripts [13] [15] [17]. This is done by creating the test script manually, then replacing the used inputs and expected outputs with variables. The values for the variables are stored in external data files. The test data is iterated for the test script until the test script has been run for each test data provided. The test data can contain a large number of different data combinations, increasing the test coverage in a way that would not be feasible with manual testing. [15] [17]

Example of data-driven test automation can be seen in Figure 2. Figure illustrates how, for example, a division operation of a calculator could be tested. The test data can be provided in a tabular format using a spreadsheet program. The test script contains a user function which reads the input data and expected output data for the test execution. After executing the actions and performing validations, test data from the next row is read and the same sequential test steps are repeated with the new test data.


data-driven calculator - Excel
D1
Sheet1
READY
100 %
Input data
1. Read input data
Test script
4. Validate output with expected output data
Expected output data
Automation tool
2. Perform actions with the input data
3. Read output from the system under test
System under test

	A	B	C	D	E
1	INPUT 1 (divident)	INPUT 2 (divisor)	EXPECTED OUTPUT		
2	4	2	2		
3	4	0	ERROR		
4	7	-2	-3,5		

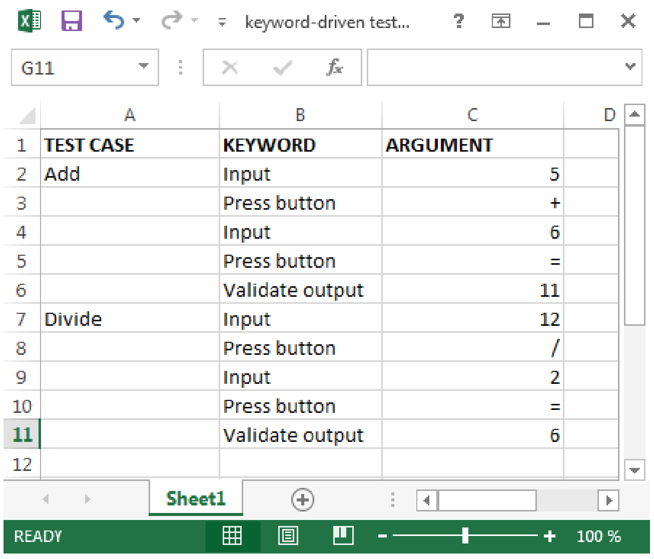
Figure 2. Principle of data-driven test automation.

Data-driven testing enables flexible test case creation. Because the test data is differentiated from the test script and automation tool, creating test cases does not necessarily need any technical knowledge from the tester if the scripts are already created. Maintenance work for the automated tests decrease, because if the functionality changes in the system under test, the maintenance is needed for the test script and not the numerous test cases using the test script. [15]

While data-driven testing has testing scenarios where it thrives, for example performing testing on the syntax of data fields, it does have its limitations. If the testing scenarios provided by the test scripts prove insufficient, it always requires a technical person with programming skills or automation tool specific scripting skills to create new test scripts [15]. Additionally, as John Kent points out, most regression and system tests are not about repetitively inputting data into the user interface, but rather trying to fully execute business functionality in a realistic way [17]. Data-driven test automation is not an optimal platform for such a purpose.

2.3.4. Keyword-driven test automation

Keyword-driven testing is taking a step further from data-driven testing, where the test data was stored in an external file. In keyword-driven test automation, also the test actions are stored in an external file in addition to the test data. These test actions are called keywords. Test data becomes the actual test script describing the sequence of actions to be followed [17]. Test data is passed as arguments with the keywords. Figure 3 [18] illustrates a test data file containing keywords and test data. This provides an additional level of abstraction between the test script and the actual scripts that drive the system under test. Figure 4 [16] [18] illustrates the usage of this test data file in a keyword-driven test automation. The driver script interprets the keywords from the test script and executes the specified actions. These actions are implemented outside of the driver script as supporting scripts, which call a specific function in the system under test. [15] [16]



	A	B	C	D
1	TEST CASE	KEYWORD	ARGUMENT	
2	Add	Input	5	
3		Press button	+	
4		Input	6	
5		Press button	=	
6		Validate output	11	
7	Divide	Input	12	
8		Press button	/	
9		Input	2	
10		Press button	=	
11		Validate output	6	
12				

Figure 3. Keyword-driven test data file.

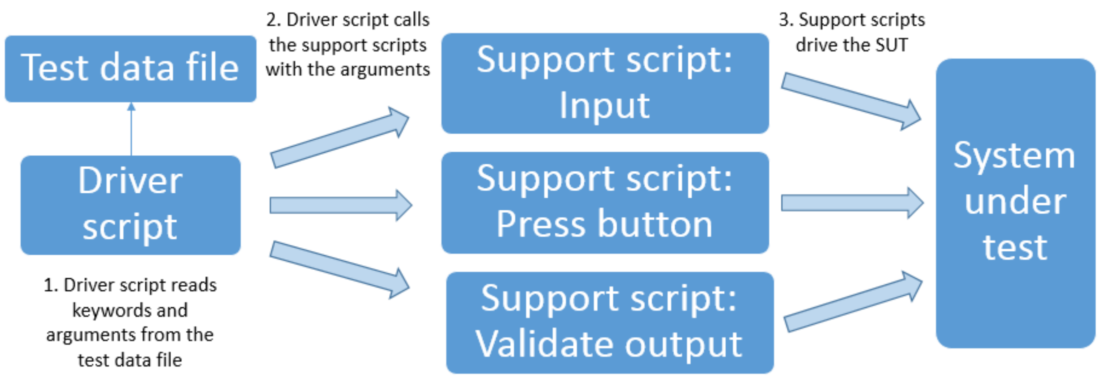

Test data file
Driver script
1. Driver script reads keywords and arguments from the test data file
2. Driver script calls the support scripts with the arguments
Support script: Input
Support script: Press button
Support script: Validate output
3. Support scripts drive the SUT
System under test

Figure 4. Principle of keyword-driven test automation.

One of the main benefits of the keyword-driven approach is that creating test cases does not require technical skills from the tester because test cases are created using common language or easy to understand keywords. This can increase the number of people involved in software test automation and making better use of the automation investment. People with business and testing knowledge could concentrate on creating the test cases, while technical engineers create the driver scripts and supporting scripts that interpret the keywords. [13] [16]

As stated earlier, writing automated tests can be comparable to writing computer programs. Programming tests to work correctly can be as challenging as programming software to work correctly [16]. This brings us to a situation, what John Kent calls as "The software test automation paradox", where the automated tests need to be tested as well. Resolving this fundamental problem is one key factor for a successful, large-scale test automation. Keyword-driven test automation aims to tackle the test automation paradox. [17]

2.3.5. Test automation framework and test libraries

A test automation framework provides the basic set of software tools and services to aid software testers in developing and executing automated test cases [19]. An important aim has been to move the creation of automated tests away from the test tools scripting or programming language to higher levels of abstraction, reducing the need for technical skills from people using them [17]. By using a test automation framework, the testers can focus on to the actual testing of the software instead of developing the infrastructure needed to support their test environment [19]. The system under test can consist of several different components. A good test automation framework should be generic enough to provide helpful functions for testers to create automated tests for all the different components [19]. Some of the important requirements for a large scale test automation framework presented by Laukkanen are listed below [18]:

- The framework must execute test cases automatically.
- The framework must be easy to use without programming skills.
- The framework must verify test results.
- Test execution must be logged.
- Test report must be created automatically.
- The framework must be modular.

Test automation framework's capabilities can be extended with test libraries. A test library is a software library with the purpose of aiding in software testing. Software library can be defined as a controlled collection of software and related documentation designed to aid in software development, use, or maintenance [20]. In the keyword-driven automation example presented in the previous section, the supporting scripts could be bundled into a test library and driven using the automation framework. Figure 5 illustrates a setup of driving system under test using a test automation framework and test libraries. The different components of the software system can consist of different technologies, and a test library is specifically targeted for a specific technology component. Test libraries provide the generic functions to drive the system under test, and test cases are created in the test automation framework.

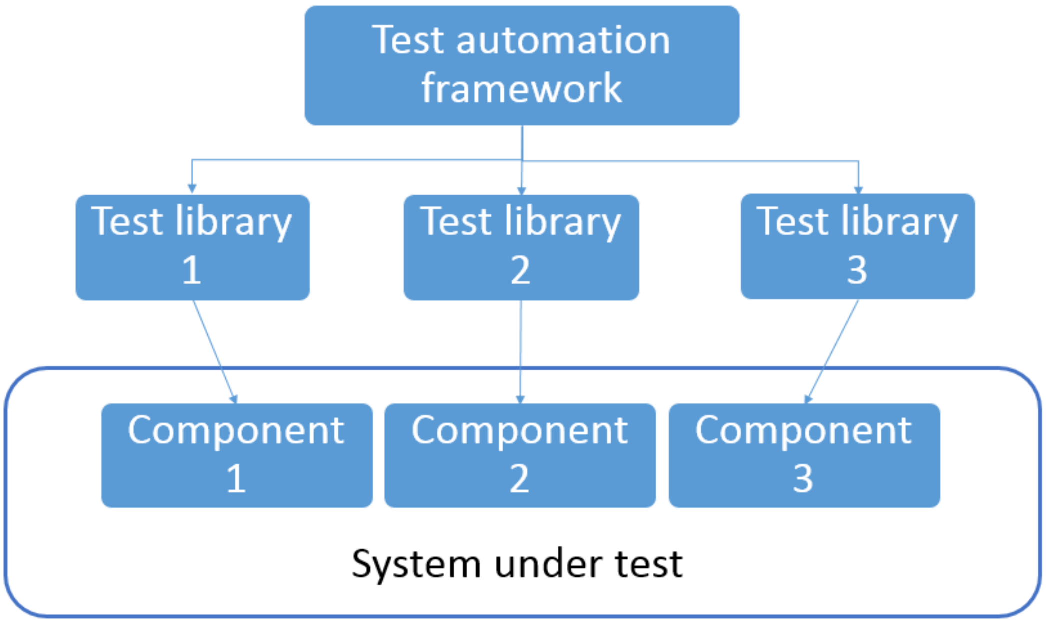


Figure 5. Test automation framework driving different components using test libraries.

3. BACKGROUND FOR THE TEST LIBRARY

The previous chapter introduced the basic principles of software testing as well as test automation. Topics of test automation framework and test libraries were also covered. This chapter aims to cover the theory regarding the test library implementation, such as the target environment, key concepts and existing tools utilized. Known existing test automation approaches for the target systems are also introduced, and where the test library implementation stands in comparison.

3.1. Test library target environment

3.1.1. SAP

SAP AG[1] (Aktiengesellschaft) is one of the world's largest software manufacturers and the leader in enterprise applications in terms of software and software-related service revenue [21]. SAP provides software to manage business operations, and their main product is SAP ERP. The main principles in SAP products are; flexible integrated platforms that are designed to change and adapt for customer's business needs, real-time data extraction, scalable infrastructure and adopting of new technology innovations [22].

SAP systems are highly modular. This enables customers to pick the modules they need for acquiring the IT functionality for their business processes. High level of integration amongst SAP applications enables data consistency across the company. The modularity and integrations makes often the testing of full end-to-end processes difficult, as the SAP modules can be in use in different functional departments of the customer organization. Organizing a managed and controlled testing that spans across multiple departments can prove challenging and time consuming. The modularity and integrations also create gaps when considering test automation, as the systems, technologies and data changes dynamically. Normally, in a full end-to-end process, these gaps have to be filled manually when interacting with different modules.

The test library implemented in this thesis aims to fill the manual gaps caused by interfaces in SAP related test automation. The purpose of the library is to dynamically fetch data from SAP back-end to be used as inputs in test automation scripts, and also fetch data from the back-end with the purpose of checking that the provided inputs were handled correctly in the system. The scope for the thesis was set to cover selected scenarios for automated end-to-end test process of an SAP e-commerce solution connected to SAP ERP back-end.

3.1.2. SAP e-commerce

In the thesis, SAP e-commerce can be stated to generally mean SAP applications for e-commerce solutions. These e-commerce solutions are either business-to-business (B2B) or business-to-consumer (B2C) web shops, where customers place orders. SAP provides different implementation options and technologies for SAP e-commerce applications. These different options will not be introduced in this thesis,

[1] http://global.sap.com/corporate-en/index.epx. Accessed 24.8.2014

as it is not seen relevant regarding the test library development. The current SAP e-commerce implementations all share three specific technology characteristics. These characteristics are:

1. A front-end web shop, consisting of HyperText Markup Language (HTML) elements for the interaction with the customer.
2. A back-end, to which the e-commerce application is connected to. All orders placed in the front-end are replicated to the back-end.
3. Product data replication from the back-end to the e-commerce web shop.

The aforementioned characteristics are relevant regarding the test library development. A back-end is an SAP system running on Advanced Business Application Programming (ABAP) application server [23]. ABAP is the proprietary programming language of SAP [24]. A back-end can be an SAP ERP, CRM or MDM system, but in the scope of the thesis only ERP is included.

Figure 6 illustrates a generic SAP e-commerce scenario. Customer uses a web browser to access, navigate and place orders in the web shop. The web shop is connected to an SAP back-end, from which the web shop data is retrieved. This data includes the product data, for which customers can place orders. When a sales order is submitted, it is replicated to SAP back-end, where a sales document object is created from the order. Sales document is a database document in SAP ERP, representing a business transaction in the sales department [25]. The sales document then goes to further processing to eventually fulfill the order and deliver the products, but that process is excluded altogether from this thesis.

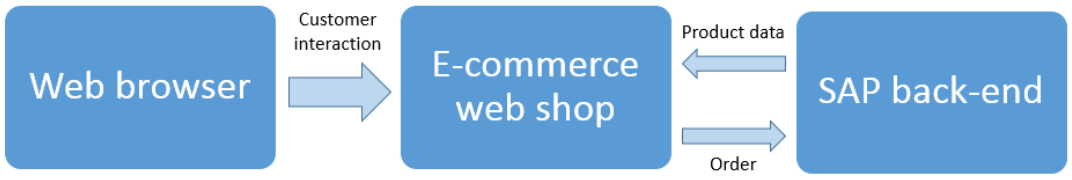


Figure 6. A generic SAP e-commerce scenario.

Regarding terminology, sales order is used in the thesis to describe the order placed in the web shop. A sales document is created from the sales order in SAP back-end. Sales document is inspected in more detail in Section 3.3 “ERP sales document structure”. Next, the current approaches of SAP test automation are introduced and where the test library aims to take place.

3.2. SAP test automation

This section gives a brief introduction to existing test automation approaches for SAP systems. The thesis implementation scope is demonstrated in an SAP e-commerce functional test automation scenario.

3.2.1. Known existing approaches

There are different options for implementing SAP test automation. Many of the available test automation software revolve around capture-playback approach, which also utilize data-driven and keyword-driven automation. Such tools communicate with SAP GUI (Graphical User Interface) or SAP WebClient UI, which are the presentation layer of the three layer architecture of SAP systems. The other two layers are application layer and database layer [26]. Presentation layer is where the users interact with SAP systems. SAP provides an API for SAP GUI scripting, which can be utilized in creating custom test automation. The presentation layer encapsulates the logic of the application layer, and therefore following business processes in the presentation layer is the natural way of interacting with the system.

Another approach to SAP test automation is to utilize the testing tools and infrastructure provided by SAP. The usage of SAP provided test automation infrastructure requires SAP Solution Manager, a software which functions as the central management hub for all SAP systems. It provides tools and methodologies for efficient implementation, operation, monitoring and support of SAP products. SAP provides Test Automation Framework, which allows the creation and execution of automated test cases. Test Automation Framework is integrated in SAP Solution Manager. SAP Solution Manager and Test Automation Framework enable the usage of the following SAP test automation offerings:

- Component Based Test Automation (CBTA)
- Extended Computer Aided Test Tool (eCATT)
- SAP Test Acceleration and Optimization (TAO)
- SAP Quality Center by Hewlett-Packard
- Business Process Change Analyzer (BPCA)

Another key topic from test automation perspective is the Business Blueprint. [27] [28]

In SAP Solution Manager, the Business Blueprint is used to document the business processes of a company. Business processes are organized in a hierarchical structure, the Business Process Hierarchy. When the business processes are configured to SAP systems, the Business Blueprint is used as a reference. All test plans utilizing SAP testing infrastructure are based on the Business Blueprint. Business Process Change Analyzer keeps track of the parts of the system which are affected during new development or patch installations. This is achieved by keeping track of the technical objects in each described process in the Business Blueprint. When new installation arrives, the technical objects of the installation are checked and compared to the existing ones to determine which system areas are affected. This information is used by the Business Process Change Analyzer to determine the regression test scope. [29] [30] [31]

CBTA, eCATT and SAP TAO are all SAP's proprietary tools for creating test automation scripts for SAP systems. They all record the scripts from traditional SAP user interfaces, which are SAP GUI and WebClient UI. CBTA differs from eCATT in its modular approach, where test components can be reused and repaired fast. CBTA and SAP TAO aim in enabling the business users to create test scripts, without much technical knowledge, where eCATT requires more developers' expertise. CBTA and eCATT are free of charge and delivered with SAP systems.

SAP TAO is a separately licensed product from SAP with tight integration to Hewlett-Packard's products, such as Quality Center and QuickTest Professional. If customers use Quality Center and QuickTest Professional, then SAP TAO is a good consideration for SAP systems test automation. [28]

A common issue with SAP test automation is the heterogeneous system landscape; customers' IT systems compose of SAP and non-SAP systems, where end-to-end processes go through multiple different technologies in various different areas, such as mobile and web platforms. Because of the variety in technology and modularity, systems are integrated with multiple interfaces. Commercial test automation tools have the challenge of keeping up with the heterogeneous system landscape, and many fall short in their technology coverage. Utilizing SAP provided test automation tools are limited to just interacting with SAP's user interfaces [28]. Currently, only one commercial testing platform claims to fully cover automated end-to-end testing processes in the major technology areas in an SAP and non-SAP system landscape [32]. For SAP system testing, the aforementioned test platform utilizes the SAP eCATT interface. Prerequisites are also the maintenance of Business Blueprint in SAP Solution Manager.

Business Blueprint is the key for using some of the advanced features of SAP Solution Manager, such as the test infrastructure. Many customers do not utilize SAP Solution Manager to its full potential. In a survey, conducted by Panaya Inc. in 2010, a total of 347 SAP customers and partners were interviewed regarding the use of SAP Solution Manager. The respondent profile consisted of 83% of SAP customers, who run their own business on SAP systems, and 17% represented SAP partner system integrators. The results of the survey indicated that 42% of the respondents did not maintain any business processes in the SAP Solution Manager, therefore lacking the Business Blueprint. Only 3% had fully documented their business processes. In addition, 60% responded that they do not use SAP Solution Manager for test management purposes. [33]

To implement test automation with any test tool that utilizes SAP test automation infrastructure, the majority of the customers need to spend time and resources for setting up the Business Blueprint and SAP Solution Manager before it's even possible to implement the test automation. Such testing tools do not serve the purpose for SAP partners implementing SAP solutions for multiple customers. To ensure quality deliveries, the implementations need to be thoroughly tested. The test library implemented in the thesis aims to be a generic, customer independent testing tool. The goal is to enhance the test automation coverage to reach SAP systems without commercial testing tools and the need for complex installations and maintenance in the customer landscape.

3.2.2. SAP e-commerce functional test automation

Figure 7 shows how automated functional testing for e-commerce implementations is currently performed in Bilot, which is the SAP partner company the author is implementing the thesis in. This is a common approach, which currently applies to such implementations in general. First, the needed setup needs to be done, which includes manually determining the product data to be used in different test scenarios. Test scripts are developed and automated functional testing is done by driving a web browser, which simulates customer interaction with the web shop. Outputs from the web shop are validated against expected outcomes. If a sales order is sent to the back-end, the back-end sends a response to the web shop with a sales document number, indicating a successfully submitted sales order. If further validation for the sales document is needed, the validation needs to be performed manually in the back-end.

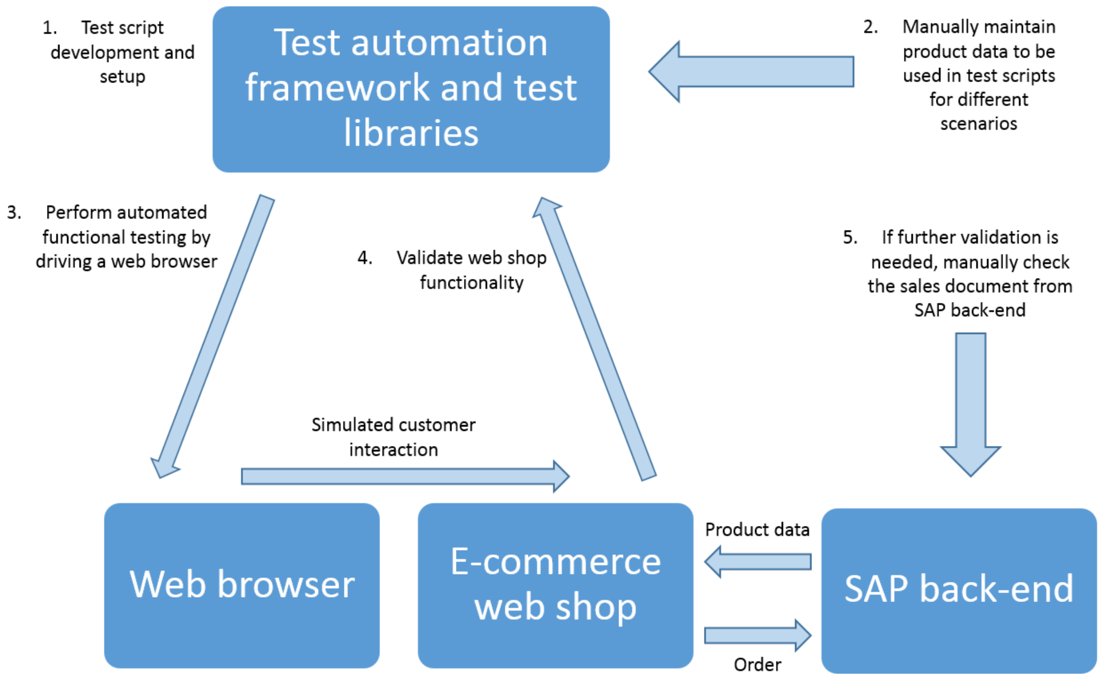

1. Test script development and setup
Test automation framework and test libraries
2. Manually maintain product data to be used in test scripts for different scenarios
3. Perform automated functional testing by driving a web browser
4. Validate web shop functionality
5. If further validation is needed, manually check the sales document from SAP back-end
Simulated customer interaction
Web browser
E-commerce web shop
Product data
Order
SAP back-end

Figure 7. Current functional test automation process for SAP e-commerce solutions.

Figure 8 shows where the test library aims to fill the manual gaps performed in the test process, reducing manual maintenance of product data and increasing validation coverage to SAP back-end. By reducing the manual work, the chain of test automation can be extended. This makes it possible to provide more automated test processes, increasing the quality of the deliveries with fully automated end-to-end test cases. In addition, SAP landscape includes multiple different environments for development and quality assurance. By automating the product selection, the test data maintenance in different environments for products is reduced, and ideally removed alltogether. This reduces the maintenance needed for the test automation scripts, thus increasing the return of investment of test automation.

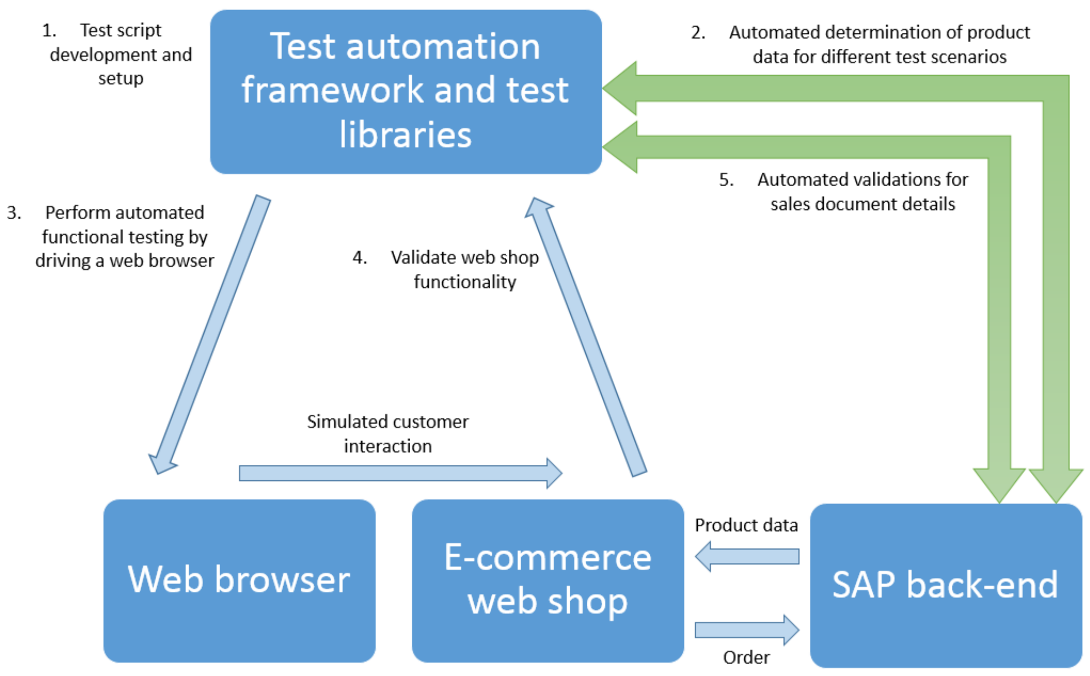

1. Test script development and setup
Test automation framework and test libraries
2. Automated determination of product data for different test scenarios
5. Automated validations for sales document details
3. Perform automated functional testing by driving a web browser
4. Validate web shop functionality
Simulated customer interaction
Product data
Order
Web browser
E-commerce web shop
SAP back-end

Figure 8. Functional test automation process extended with the test library.

3.3. ERP sales document structure

The sales document structure is utilized in the development of the test library and sales order validation. All sales-related business transactions, such as inquiries, quotations, sales orders and deliveries, are recorded as sales documents in SAP ERP. Sales documents consist of three levels; header, item and schedule line. Each sales document consists of a document header and any number of items. Furthermore, items can be divided into any number of schedule lines. Figure 9 [34] demonstrates the structure of a sales document. [35]

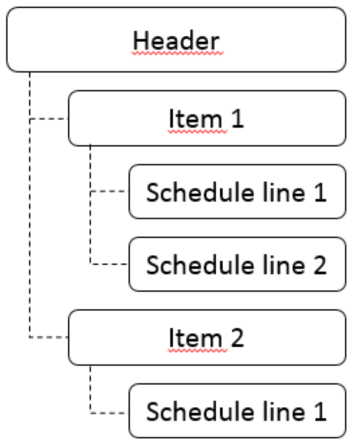


Figure 9. Sales document structure.

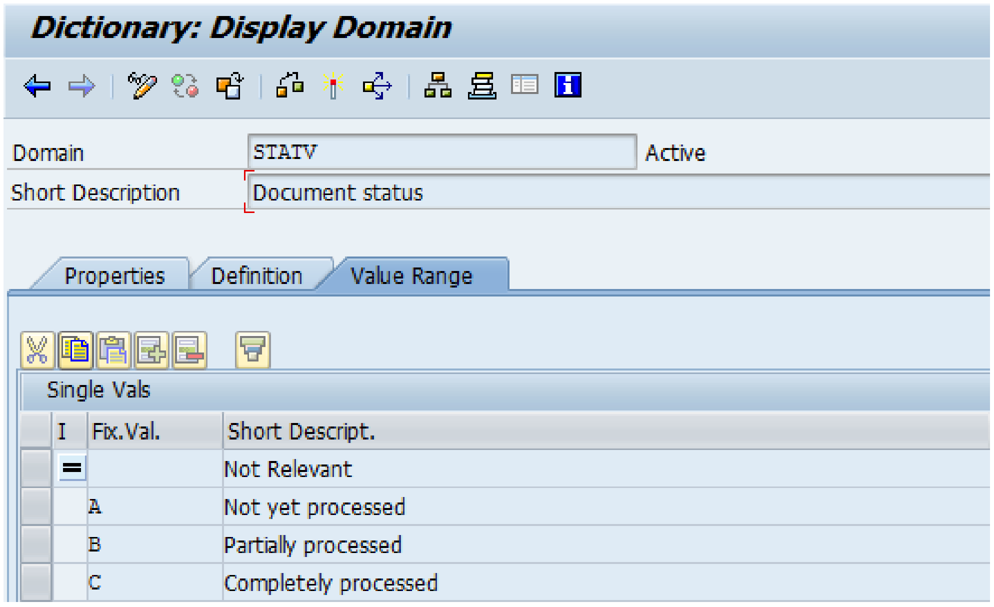
Document header holds the data that is general for the entire sales document. Examples of such a data would be sold-to party (to whom the sale was made), ship-to party (to whom the delivery will be shipped), document currency and delivery date.

The data in document header applies to all items, but some data applies only to the item itself. This data is stored at item level, such as product number, target quantity, item specific ship-to party and pricing elements. Schedule lines hold all the data that is needed for the delivery. Using Figure 9 as an example of a sales document that a customer has placed, Item 1 could be an order of 10 units of a particular product. The supplier can only deliver 5 pieces immediately and the remaining 5 pieces next month, so two scheduled deliveries are needed. The deliveries are stored in two separate schedule lines, including data about the delivery date and confirmed quantity for the schedule line. [35]

In the test library, validations for the sales document details are done based on the queried data in different levels. The scope of the thesis was set to check header level data of a sales document. Automated product determination is done utilizing the schedule line information for products by simulating sales order creation in the back-end. Based on the simulation results, the usage of the product in the test script is determined.

3.4. Domain

The test library receives the sales document header level data in a tabular format. The tables contain data elements, and data elements have values. For each value, a domain has been assigned in the SAP back-end. Domains define the value range that a table element can have [36]. Domain also provides the description for the values. This means that value "A" can mean a different thing in different fields of the table. Figure 10 shows an example domain STATV from SAP ERP. The domain is used with fields associated with document statuses, and its possible values and their descriptions can be seen from the screenshot.



Domain	STATV Active
Short Description	Document status

I	Fix.Val.	Short Descript.
=		Not Relevant
	A	Not yet processed
	B	Partially processed
	C	Completely processed

Figure 10. A screenshot from SAP ERP describing domain STATV.

Test execution logs need to be human readable. If the validated values do not mean anything to the log inspector, then there is no benefit in the log itself. The information regarding data elements' domains are needed for this purpose, to make the test execution log meaningful.

3.5. Framework and selected libraries

The test library is run by a test automation framework. The library implementation itself requires a set of other libraries. The chosen framework is introduced in this section, as well as the libraries needed in the implementation.

3.5.1. Robot Framework

Robot Framework[23] was chosen to be used as a test automation framework. Robot Framework is an open source, Python-based generic test automation framework. It utilizes the keyword-driven testing approach, and its capabilities can be extended by using test libraries. The selection of the tool was based on the following factors:

1. Robot Framework is widely in use in the company the author is implementing the thesis in.
2. Consulting co-workers, who work in the profession of software testing and test automation, on different test automation tools.
3. Previous author's personal experience and basic knowledge on working with Robot Framework.
4. Previous author's Java programming knowledge, and the possibility to create customized test libraries for Robot Framework in Java.
5. Good user guide and documentation for the framework.
6. Robot Framework has an active user base and on-going development.

Some of the useful features Robot Framework has are listed below:

- Easy-to-use tabular syntax for creating test cases in a uniform way.
- Users can create reusable higher-level keywords from the existing keywords.
- Easy-to-read reports and logs in HTML format.
- Platform and application independent.
- Simple library API for creating customized test libraries implemented with Python or Java.
- Command line interface and EXtensible Markup Language (XML) formatted output files enable integration into existing build infrastructure.
- Data-driven test case support.
- Built-in support for variables.
- Easy integration with source control.
- Provides test-case and test-suite –level setup and teardown.

[2] http://eliga.fi/writings.html. Accessed 24.8.2014
[3] http://robotframework.org. Accessed 24.8.2014

Robot Framework comes also bundled with a set of standard test libraries that are automatically installed with the framework. These features makes the framework a very useful tool in test automation. [37]

The architecture of Robot Framework is highly modular. A high level architecture diagram can be seen in Figure 11 [37].

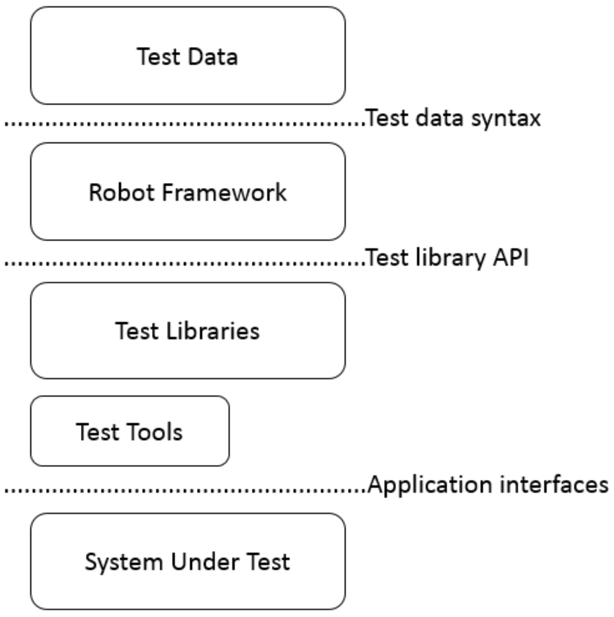


Figure 11. Robot Framework architecture.

The test data is presented in a tabular format. Robot Framework processes the test data, executes the test cases and generates reports and logs of the execution. The core framework itself does not know anything about the system under test. Instead, the interaction with the system is handled by test libraries. Test libraries can use application interfaces directly, or alternatively use lower level test tools as drivers to drive the system. [37]

3.5.2. Libraries used in the implementation

Robot Framework's capabilities were extended in this thesis with chosen libraries. The following libraries were required in the implementation project of the test library; Selenium2Library[4], SAP Java Connector (SAP JCo) [38], Robot Framework's Remote library[5] as well as AnnotationLibrary[6].

Selenium2Library is a test library for Robot Framework that is made specifically for web testing. It uses Selenium WebDriver libraries from the Selenium project. Selenium is a suite of tools for automating web browsers across many platforms, and the Selenium WebDriver is a collection of language specific bindings for driving a browser [39]. Selenium2Library was used to interact with the e-commerce web shop,

[4] https://github.com/rtomac/robotframework-selenium2library. Accessed 24.8.2014
[5] https://code.google.com/p/robotframework/wiki/RemoteLibrary. Accessed 24.8.2014
[6] http://code.google.com/p/robotframework-javatools/wiki/AnnotationLibrary. Accessed 24.8.2014

simulating a real usage by a customer placing an order. The library was not used in the test library implementation directly, but it was required in creating a complete automated regression test scenario, which begun in the web shop by placing a sales order with a chosen product.

SAP JCo is an SAP proprietary toolkit for connecting external applications to SAP systems. SAP JCo is licensed without additional fees as part of other SAP solution or component licenses, therefore being practically free for SAP customers. It enables the development of Java components and applications that can communicate with SAP systems. SAP systems are written in ABAP, and SAP JCo supports communication in both directions; Java calls to ABAP and ABAP calls to Java. SAP provides different implementation versions of SAP JCo. The standalone version of SAP JCo was used in the thesis, as it can be installed independently of an SAP system. The standalone version enables communication between external non-SAP Java application and ABAP application server. SAP JCo communicates with ABAP function modules in ABAP application servers. Function module can be described as a subroutine written in ABAP, which serves as a general-purpose function [40]. Only standard function modules provided by SAP were used in the thesis. [38]

Remote library is one of the standard libraries of Robot Framework. It works as a proxy between Robot Framework and an actual test library implementation. The test library implementation is served to Robot Framework as a remote server. Remote library and the remote library interface provide two very useful features for implementing and using test libraries. First, the test library does not have to be located on the same machine that Robot Framework is running on. Secondly, the test library does not have to be implemented in the natively supported programming language of Robot Framework. AnnotationLibrary is a part of Robot Framework's Java tools. It allows to use Java annotations to tag Java methods, which are then registered to Robot Framework as keywords. [37]

The test library implemented was decided to be written in Java. One of the main reasons for the selection of Java as an implementation language was the use of SAP JCo to communicate with SAP systems. Robot Framework comes also as a Jython[7] implementation running on Java Virtual Machine and would therefore offer a native support for Java libraries. Nevertheless, Python version is the most mature and fastest version of Robot Framework [37]. The Python version was used in the thesis, and as the custom library was to be implemented in Java, Remote library offered the perfect solution for using the Java-based library with the Python implementation of Robot Framework.

[7] http://www.jython.org/. Accessed 24.8.2014

4. DESIGN AND IMPLEMENTATION

Chapter 2 introduced the basic concepts of software testing and test automation. In Chapter 3, the target environment was presented, as well as where the test library aims to fill the manual gaps of test automation in the scope of the thesis implementation. The key concepts and tools needed in the implementation were described, and where the test library stands in comparison to existing approaches.

This chapter goes more into detail in defining the requirements for the test library. The design of the library is illustrated and the implementation process is described in detail.

4.1. Requirements

The requirements are distributed under functional requirements, as well as a subset of quality characteristics presented in Table 1 in Chapter 2.

Functional requirements

The test library must fetch dynamically, at test execution run-time, data from SAP back-end and use it in the end-to-end test scripts. The test library must enable assertions for the fetched data. The following three scenarios are in the scope of the thesis.

1. Sales document validation from SAP back-end.
   a. Thesis scope: SAP back-end covers only ERP.
   b. Thesis scope: Sales document header status data is fetched for validation purposes. Assertions are made to the fetched data.

2. In stock product determination from SAP back-end.
   a. Thesis scope: SAP back-end covers only ERP.
   b. Product is in stock on the given date.
   c. ID of the product is returned for test script usage.

3. Out of stock product determination from SAP back-end
   a. Thesis scope: SAP back-end covers only ERP.
   b. Product is out of stock on the given date.
   c. ID of the product is returned for test script usage.

Connectivity

The test library must be usable from a test automation framework. Test cases utilizing the test library must be able to be run from a continuous integration service.

Flexibility

Tester must be able to adapt the assertions to his/her needs by defining the pass/fail criteria for a test case.

Maintainability

The test library must use a modular architecture in utilizing external libraries. The test library must be able to further develop and extend to different SAP back-ends.

Manageability

The test library must be relatively easy to deploy into operational condition.

Performance

Using the test library should not hinder overall test performance by a significant amount.

Reusability

The test library must be a generic solution which can be utilized in multiple SAP environments with similar requirements.

User-friendliness

The test library must be easy to use and to learn for end-users. In this case, end-users possess expert knowledge on test automation and SAP systems.

4.2. Design

In this section, the design of the software is presented. The design faced some adjustments from the initial planning. This was due to a constant learning process regarding the technical side of SAP and implementation of a custom Robot Framework test library. The section presents the final design of the software.

4.2.1. Architecture

Figure 12 presents the full architecture of the implemented test library and the related components in the overall scenario. Test library implementation consists of different components; *fi.bilot.robot* Java package, AnnotationLibrary, RemoteServer and SAP JCo. The Java package *fi.bilot.robot* is the developed component, which is dependent on the other components. RemoteServer is a component of the Robot Framework's Remote library, which together with SAP JCo and AnnotationLibrary were introduced in Chapter 3.

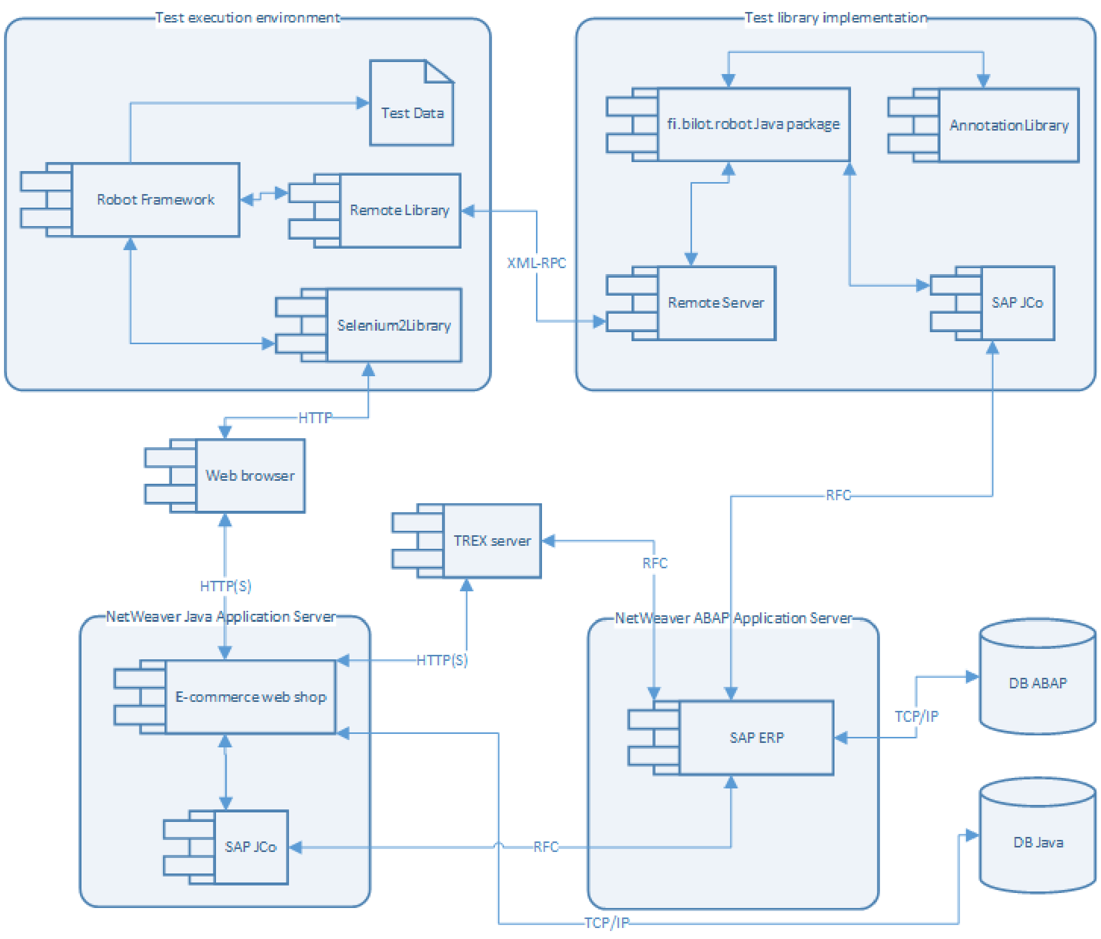


Figure 12. Architecture diagram.

The test library implementation is driven from Robot Framework, and the communication is done with XML-RPC protocol via Remote library interface. XML-RPC protocol is the EXtensible Markup Language (XML) implementation of the Remote Procedure Call (RPC) protocol. The test library implementation communicates with SAP systems via SAP JCo. The communication is done with Remote Function Calls (RFC), which in turn is SAP proprietary implementation of the RPC protocol [26, p.275]. RFC is the standard SAP interface for communication between SAP systems [41]. SAP JCo performs the interface functions and maps the ABAP data types to Java data types [42].

The e-commerce web shop resides in NetWeaver Java Application Server. A Text Retrieval and information EXtraction (TREX) server can be used to enhance the search functionality of the web shop, improving system performance for the customers [43]. As the web shop resides in a Java server, the communication with NetWeaver ABAP Application Server is handled by SAP JCo using RFC calls to SAP ERP system.

In a full scenario, Robot Framework imports the test library implementation. Test scripts are read from a test data file and executed. Product fetching is first initiated from Robot Framework and implemented in *fi.bilot.robot* Java package. Using SAP JCo, the products for the selected scenarios are fetched from SAP ERP and returned to Robot Framework. Web browser is driven by Selenium2Library, and the actions defined in the test data are performed in the e-commerce web shop. Using the fetched products, order is placed in the web shop and the ID of the order is collected. The order is replicated via RFC call to SAP ERP. The ID of the order is passed from Robot Framework to the test library implementation, and the corresponding sales document is fetched from SAP ERP. Validations defined in the test data file are performed in *fi.bilot.robot* Java package, and the validation results are passed to Robot Framework.

The RFC communication between SAP JCo and SAP ERP contains tables and structures. The test library sends in the function module specific parameters which are required for the execution of the function module. Function module has a specific purpose, a set of actions it performs with the parameters. Once finished, the function module returns the results as tables to SAP JCo. The returned tables are then used for the test library's purposes. This chain of events is described in Figure 13.

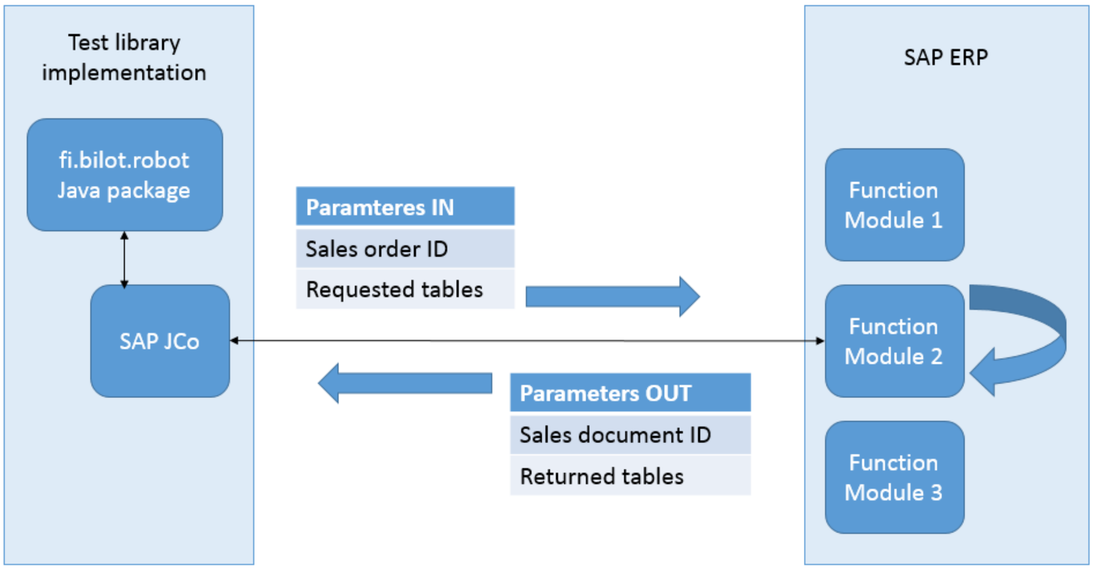


Figure 13. Example of a data exchange between the test library and SAP ERP.

4.2.2. Class diagrams

The class diagrams presented in this section are modeled using The ObjectAid UML Explorer for Eclipse[9]. The purpose is to model the implemented custom library. Classes from components not implemented in the thesis are only shown to the first level of association. The custom library Java classes reside under *fi.bilot.robot* Java package and these are implemented in the thesis. For the classes in other components and packages, only the name of the class is presented in many cases for the sake of simplicity.

The class diagrams are grouped using functional grouping as follows;

- server implementation,
- keyword implementations which fall under the following topics; back-end connection, sales document validation and product determination.

[9] http://www.objectaid.com. Accessed 10.7.2014

Server implementation

Figure 14 shows the class diagram of the server. *Server* class is the main starting point of the program. It has a dependency on *RemoteServer* class, which implements the Robot Framework's remote server architecture. Server class inherits from *AnnotationLibrary* class. *AnnotationLibrary* searches all class files in package *fi.bilot.robot.keywords* for the custom Robot Framework keywords.

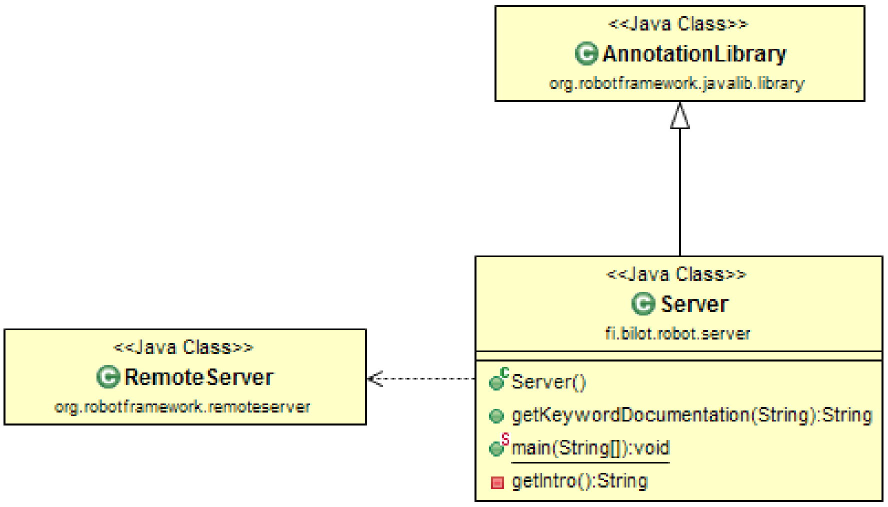
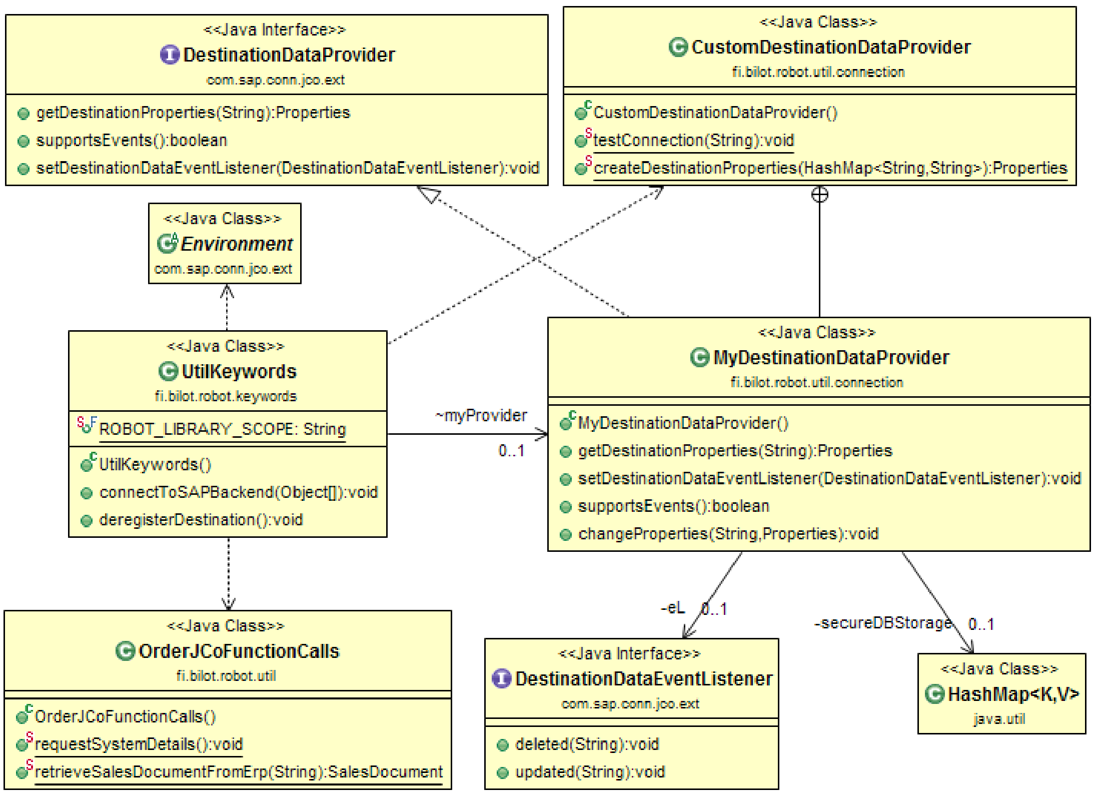

<<Java Class>>
AnnotationLibrary
org.robotframework.javalib.library
<<Java Class>>
Server
fi.bilot.robot.server
Server()
getKeywordDocumentation(String):String
main(String[]):void
getIntro():String
<<Java Class>>
RemoteServer
org.robotframework.remoteserver

Figure 14. The class diagram of the server.

When the server is started, all the implemented keywords are registered to Robot Framework. Next in this section, the implemented keyword class diagrams are presented. The keywords are distributed under three classes; *UtilKeywords*, *OrderKeywords* and *ProductKeywords*.

Back-end connection

UtilKeywords contain the keywords for utilities, which include SAP back-end connection establishment and de-establishment. All the methods in the class represent a Robot Framework keyword. The class diagram can be seen in Figure 15.



<<Java Interface>>
DestinationDataProvider
com.sap.conn.jco.ext
getDestinationProperties(String):Properties
supportsEvents():boolean
setDestinationDataEventListener(DestinationDataEventListener):void
<<Java Class>>
CustomDestinationDataProvider
fi.bilot.robot.util.connection
CustomDestinationDataProvider()
testConnection(String):void
createDestinationProperties(HashMap<String,String>):Properties
<<Java Class>>
Environment
com.sap.conn.jco.ext
<<Java Class>>
UtilKeywords
fi.bilot.robot.keywords
ROBOT_LIBRARY_SCOPE: String
UtilKeywords()
connectToSAPBackend(Object[]):void
deregisterDestination():void
~myProvider
0..1
<<Java Class>>
MyDestinationDataProvider
fi.bilot.robot.util.connection
MyDestinationDataProvider()
getDestinationProperties(String):Properties
setDestinationDataEventListener(DestinationDataEventListener):void
supportsEvents():boolean
changeProperties(String,Properties):void
-eL
0..1
-secureDBStorage
0..1
<<Java Class>>
OrderJCoFunctionCalls
fi.bilot.robot.util
OrderJCoFunctionCalls()
requestSystemDetails():void
retrieveSalesDocumentFromErp(String):SalesDocument
<<Java Interface>>
DestinationDataEventListener
com.sap.conn.jco.ext
deleted(String):void
updated(String):void
<<Java Class>>
HashMap<K,V>
java.util

Figure 15. Class diagram showing *UtilKeyword* class associations with other classes.

For the connection handling, *UtilKeywords* class creates a data provider *myProvider* for the back-end connection details. The back-end connection details are provided from Robot Framework. The data provider is implemented in *MyDestinationDataProvider* class, which in turn implements the SAP JCo *DestinationDataProvider* interface. At runtime, JCo will use the provided *DestinationDataProvider* interface implementation to get the destination configuration for the connection. *Environment* class is the central anchor for embedding JCo into the custom library. Registering and deregistering the provided destination configuration are needed, and this is done using the static methods of the *Environment* class. After registering the destination, it can then be accessed by JCo to communicate with the back-end in later scenarios. [44]

When connection establishment has been requested from Robot Framework and the connection has been successfully registered in JCo, *UtilKeywords* class calls the method *requestSystemDetails* from *OrderJCoFunctionCalls* class to get a response from the back-end system. The response from the back-end contains the system's details, and these are logged into Robot Framework's execution log if manual verification is needed for a correct back-end connection.

Sales document validation

OrderKeywords class contains the keywords for fetching the sales document from the back-end and performing validations to the sales document. All the methods in the class represent a Robot Framework keyword. The class diagram for *OrderKeywords* can be seen in Figure 16.

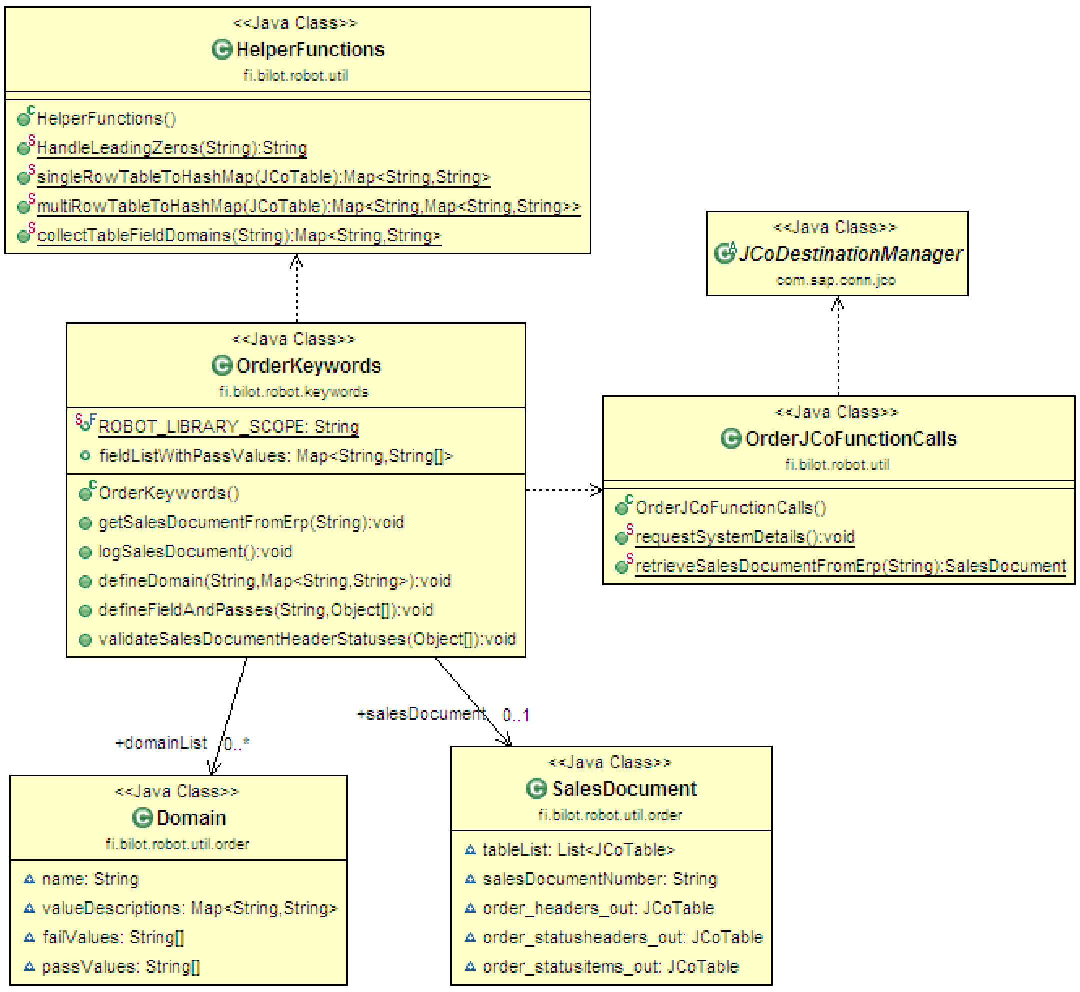

<<Java Class>>
HelperFunctions
fi.bilot.robot.util
HelperFunctions()
HandleLeadingZeros(String):String
singleRowTableToHashMap(JCoTable):Map<String,String>
multiRowTableToHashMap(JCoTable):Map<String,Map<String,String>>
collectTableFieldDomains(String):Map<String,String>
<<Java Class>>
JCoDestinationManager
com.sap.conn.jco
<<Java Class>>
OrderKeywords
fi.bilot.robot.keywords
ROBOT_LIBRARY_SCOPE: String
fieldListWithPassValues: Map<String,String[]>
OrderKeywords()
getSalesDocumentFromErp(String):void
logSalesDocument():void
defineDomain(String,Map<String,String>):void
defineFieldAndPasses(String,Object[]):void
validateSalesDocumentHeaderStatuses(Object[]):void
<<Java Class>>
OrderJCoFunctionCalls
fi.bilot.robot.util
OrderJCoFunctionCalls()
requestSystemDetails():void
retrieveSalesDocumentFromErp(String):SalesDocument
+salesDocument 0..1
+domainList 0..*
<<Java Class>>
Domain
fi.bilot.robot.util.order
name: String
valueDescriptions: Map<String,String>
failValues: String[]
passValues: String[]
<<Java Class>>
SalesDocument
fi.bilot.robot.util.order
tableList: List<JCoTable>
salesDocumentNumber: String
order_headers_out: JCoTable
order_statusheaders_out: JCoTable
order_statusitems_out: JCoTable

Figure 16. Class diagram showing *OrderKeyword* class associations with other classes.

When a sales document is fetched using *getSalesDocumentFromErp*, the static method *retrieveSalesDocumentFromErp* in *OrderJCoFunctionCalls* class is invoked. The destination for the back-end JCo function call is fetched from *JCoDestinationManager*, a JCo class holding information about the registered destination for the runtime environment. The fetched sales document is stored in an object, which is implemented in *SalesDocument* class. The *Domain* class represents the SAP ERP domains for data elements. The details for value – description pairs of a domain are passed from Robot Framework. The actual data element – domain pairs are hard coded in the custom library. The reasoning for this is that there is no data returned from the back-end regarding which domain is assigned to a data field, meaning that there is no information regarding what the data field's character code actually represents. To make the Robot Framework execution log readable and meaningful for human reading, the data fields' values are interpreted using the defined domains. The purpose of *HelperFunctions* class is to hold generic functions for modularity.

Product determination

ProductKeywords class contains the keywords for determining the products from the back-end for the different e-commerce scenarios. All the methods in the class represent a Robot Framework keyword. The class diagram for *ProductKeywords* can be seen in Figure 17.

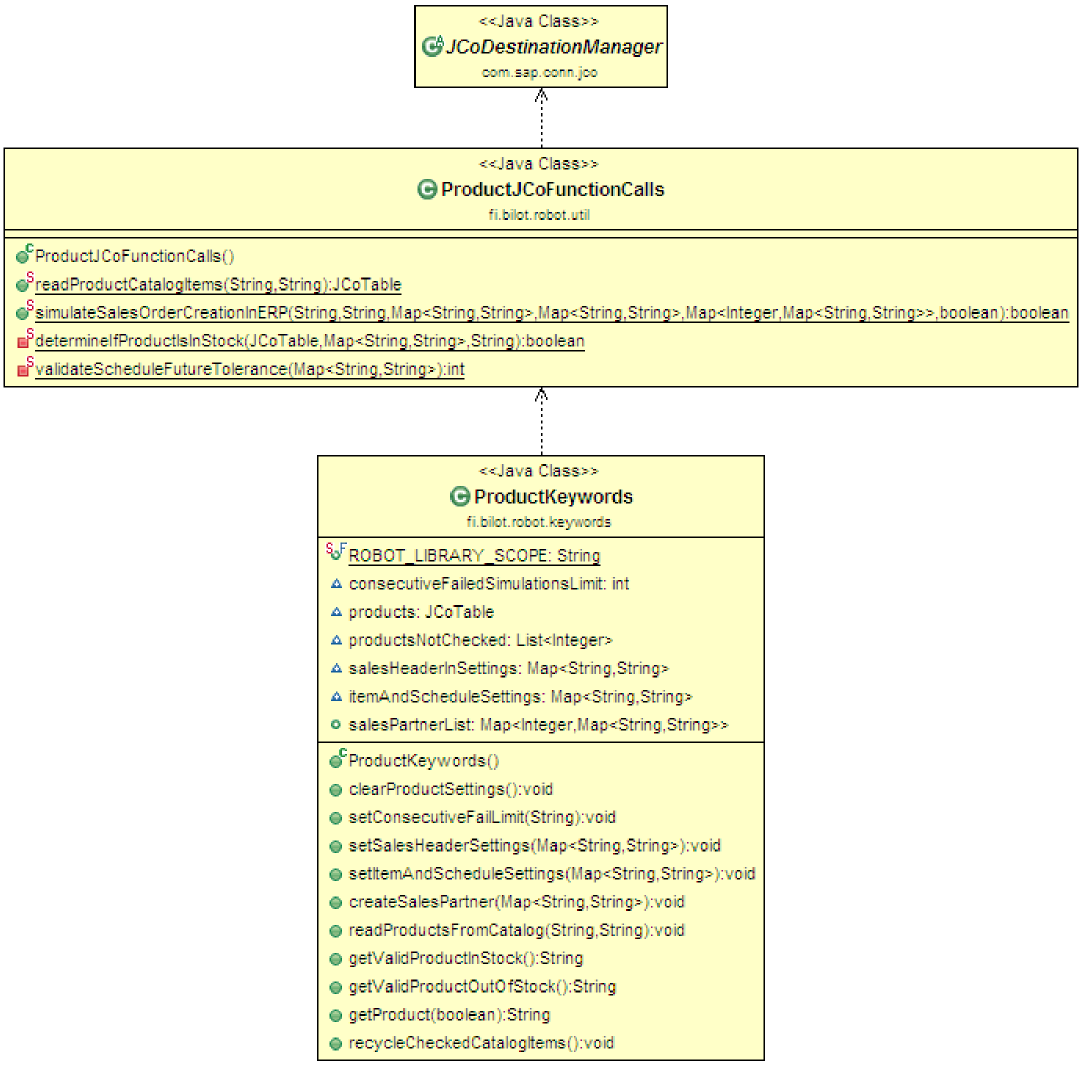

<<Java Class>>
JCoDestinationManager
com.sap.conn.jco
<<Java Class>>
ProductJCoFunctionCalls
fi.bilot.robot.util
ProductJCoFunctionCalls()
readProductCatalogItems(String,String):JCoTable
simulateSalesOrderCreationInERP(String,String,Map<String,String>,Map<String,String>,Map<Integer,Map<String,String>>,boolean):boolean
determineIfProductIsInStock(JCoTable,Map<String,String>,String):boolean
validateScheduleFutureTolerance(Map<String,String>):int
<<Java Class>>
ProductKeywords
fi.bilot.robot.keywords
ROBOT_LIBRARY_SCOPE: String
consecutiveFailedSimulationsLimit: int
products: JCoTable
productsNotChecked: List<Integer>
salesHeaderInSettings: Map<String,String>
itemAndScheduleSettings: Map<String,String>
salesPartnerList: Map<Integer,Map<String,String>>
ProductKeywords()
clearProductSettings():void
setConsecutiveFailLimit(String):void
setSalesHeaderSettings(Map<String,String>):void
setItemAndScheduleSettings(Map<String,String>):void
createSalesPartner(Map<String,String>):void
readProductsFromCatalog(String,String):void
getValidProductInStock():String
getValidProductOutOfStock():String
getProduct(boolean):String
recycleCheckedCatalogItems():void

Figure 17. Class diagram showing *ProductKeyword* class associations with other classes.

The sales order simulation settings for product determination are defined in Robot Framework, and passed as keyword arguments to the library. This is done by calling the dedicated methods, such as *setSalesHeaderSettings* and *setItemAndScheduleSettings*. Product catalog is fetched from the back-end using *readProductsFromCatalog* method. A product catalog is a structured hierarchy of products which are presented in the web shop [45]. When all the required simulation settings are defined, methods *getValidProductInStock* or *getValidProductOutOfStock* are called, depending on the chosen scenario. This picks a randomly selected product from the catalog, and calls method *simulateSalesOrderCreationInERP* from class

ProductJCoFunctionCalls. *ProductJCoFunctionCalls* then reads the registered destination from *JCoDestinationManager*, and executes the JCo call to the back-end with the passed parameters. SAP system responds with a return table, containing the simulation results. The results are checked with method *determineIfProductIsInStock*, and a boolean true or false is returned based on the table's values. The product is then collected and the ID is returned to Robot Framework, if the simulation results fit the purpose of the chosen scenario.

4.3. Implementation

4.3.1. Development environment

The thesis was implemented in Bilot, which provides SAP solutions for client companies. The test library was developed in real development environments of SAP e-commerce implementation projects. The projects were either internal to Bilot, or web shop implementations for client companies. The development environments provided good, real e-commerce web shop architecture and were thus ideal for the test library development. The development of the test library did not interfere in any way with the development of the web shops. The existing infrastructure was used for testing the library, as well as testing the web shop using the library. Using the pre-existing environments allowed the effort to be concentrated on the test library development instead of building a feasible development environment.

Development tool used was SAP NetWeaver Developer Studio, which is based on an open-source development platform, Eclipse.

4.3.2. Software development process

This section is divided into subsections. Each subsection describes the progress during each month when the test library development took place. The development was a constant learning process for the author, and new findings during the development are also described.

October 2013

The planning and design of the thesis started in September 2013, but it was October 2013 when the plan started to solidify. Feasibility study kicked off in the beginning of October. People involved in the feasibility study were; the solution owner of e-services business area of Bilot, team leader of e-services, quality manager of Bilot, SAP consultant from e-services, and e-commerce software architect from a partner company. The technology for communication between SAP systems and third-party application was narrowed down to two options: JCo and SAP NetWeaver Gateway. Performance tests performed by the SAP consultant resulted in superior performance by JCo. Also, the work load for the generic usage of the technologies favored JCo; JCo can be used as a stand-alone library from the third-party application, where SAP NetWeaver Gateway needs to be installed to a customer's SAP landscape. As the test library was meant to be used in multiple different projects and SAP environments for a variety of customers, the installation of SAP NetWeaver Gateway to their SAP landscape could prove very difficult. Based on these observations, a decision was made to proceed with JCo as chosen technology for technical communication.

It was also decided to start the development process from the automated sales document validations, because it was seen as the more straightforward starting point when compared to fetching product data from the back-end. In the sales document validation phase, the following sequence of actions was to be automated:

1. Create sales order in the web shop.
2. Collect the order number from the web page.
3. Query the back-end with the order number to get the sales document details as a response.
4. Perform validations for the sales document details.

For navigating and performing actions in the web shop, the existing automated test cases that had been developed by the project team could be utilized.

The development environment was set up during October. The development was done in an existing virtual machine that had been done for the purpose of developing the e-commerce solution for a customer. The test library development was done as a side project in the virtual machine, utilizing the e-commerce web shop and the ERP back-end.

November 2013

As the author had no prior technical experience or knowledge regarding SAP before starting the thesis, a lot of research and study on the topic of ABAP function modules was in place during November. Connection with the test library and ABAP application server was successfully established. This was tested by performing a simple function call to function module STFC_CONNECTION, which returns the SAP system details one is connected to. The connection creation and function call to STFC_CONNECTION was wrapped as a Robot Framework keyword using the Robot Framework's remote library interface. Figure 18 shows Robot Framework log for the executed keyword "Connect To SAP Backend", which takes SAP connection details and SAP authorization credentials as arguments.

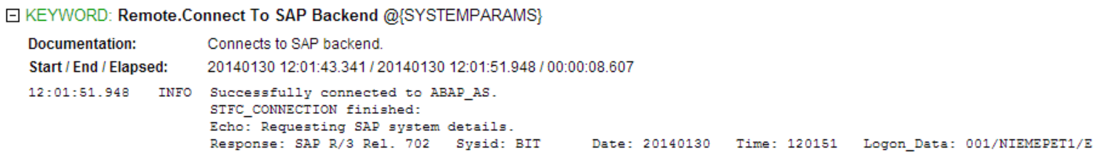

```
⊟ KEYWORD: Remote.Connect To SAP Backend @{SYSTEMPARAMS}
   Documentation:      Connects to SAP backend.
   Start / End / Elapsed:  20140130 12:01:43.341 / 20140130 12:01:51.948 / 00:00:08.607
   12:01:51.948   INFO   Successfully connected to ABAP_AS.
                         STFC_CONNECTION finished:
                         Echo: Requesting SAP system details.
                         Response: SAP R/3 Rel. 702   Sysid: BIT      Date: 20140130   Time: 120151   Logon_Data: 001/NIEMEPET1/E
```

Figure 18. Robot Framework log for establishing connection with SAP back-end.

The principle with calling function modules using JCo is that one first defines the function module specific import parameters from the Java application. Next, the function module is executed using JCo, and the function module returns data as export tables to JCo. A fully usable Robot Framework keyword utilizing JCo was created, as described in the previous paragraph, so the principle was clear for other keywords as well. At this point, the effort was in identifying the correct function modules that would be needed for the thesis, and how to use them, as some function modules can require dozens of import parameters.

Function module BAPI_ISAORDER_GETDETAILEDLIST was used to retrieve the sales document from ERP. The design was that all sales document details would be passed on to Robot Framework for customizable validations, allowing the test library to be as generic as possible. This design proved to be not the most feasible solution. This was due to the limitations of Robot Framework, as well as the remote library interface; there was no suitable way to handle the returned tables conveniently. The design was altered so that the sales document details were stored in the Java application. The validation criteria would be passed from Robot Framework to the application. This design also fitted the purpose of customizing the validation criteria. Figure 19 shows Robot Framework log for executed keyword "Get Sales Document From Erp". The keyword takes the sales document number as an argument. In a complete automated test scenario, the sales document number would be collected from the web page when the sales order is submitted.

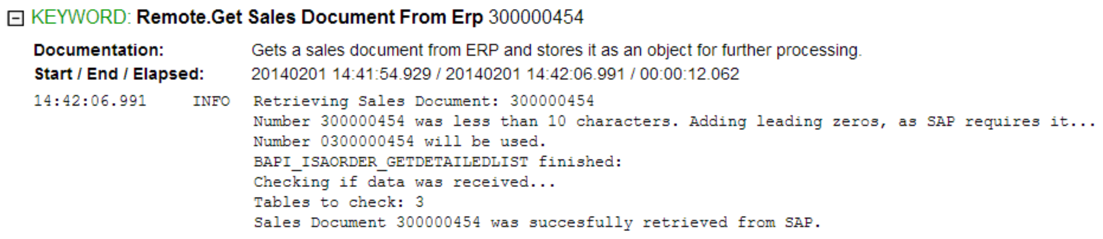
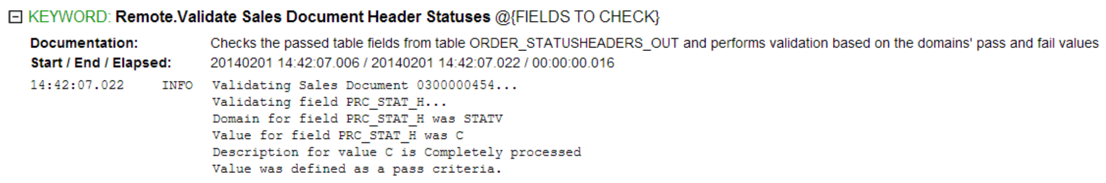

```
⊟ KEYWORD: Remote.Get Sales Document From Erp 300000454
    Documentation:        Gets a sales document from ERP and stores it as an object for further processing.
    Start / End / Elapsed: 20140201 14:41:54.929 / 20140201 14:42:06.991 / 00:00:12.062
    14:42:06.991   INFO   Retrieving Sales Document: 300000454
                          Number 300000454 was less than 10 characters. Adding leading zeros, as SAP requires it...
                          Number 0300000454 will be used.
                          BAPI_ISAORDER_GETDETAILEDLIST finished:
                          Checking if data was received...
                          Tables to check: 3
                          Sales Document 300000454 was succesfully retrieved from SAP.
```

Figure 19. Robot Framework log for retrieving sales document details from SAP ERP.

The returned tables from function modules contain data elements. For each data element, a domain has been assigned. Domains define the value range that the data element uses. Information regarding the field's domain is not carried over in the returned tables to the test library, but this information is needed to make the execution log and validation results human readable. As the number of different domains in SAP systems is tens of thousands, it was not feasible to define and hard code all of the domains into the test library. The scope of the thesis was to fetch the sales document details, and validating the statuses of the sales document was sufficient. For this purpose, only one of the returned tables was to be checked, and not many different domains were needed.

The values for passed and failed test case for each data element needed to be defined. Those values needed to be adjustable for the purpose of the library to serve as a generic library for different testing needs. The design evolved so that the library included a Robot Framework settings file. The needed domains and fields to be validated, including their pass and fail values, were defined in the settings file and passed to the test library. The test library contained the return table descriptions as hard coded field name – domain –pairs for identifying the correct domains for each field. Figure 20 shows Robot Framework log for keyword "Validate Sales Document Header Statuses". The keyword takes a list of fields as an argument, for which validations are performed. This enables the tester to customize the fields that are to be validated. Prior to calling the keyword, another keyword "Define Domains" was called in the test setup, which set the domain descriptions.



```
KEYWORD: Remote.Validate Sales Document Header Statuses @{FIELDS TO CHECK}
  Documentation:        Checks the passed table fields from table ORDER_STATUSHEADERS_OUT and performs validation based on the domains' pass and fail values
  Start / End / Elapsed: 20140201 14:42:07.006 / 20140201 14:42:07.022 / 00:00:00.016
  14:42:07.022    INFO   Validating Sales Document 0300000454...
                         Validating field PRC_STAT_H...
                         Domain for field PRC_STAT_H was STATV
                         Value for field PRC_STAT_H was C
                         Description for value C is Completely processed
                         Value was defined as a pass criteria.
```

Figure 20. Robot Framework log for sales document validation.

December 2013

The implementation for sales document validations started to come together, so the goal in December 2013 was to create the first distribution of the test library to be used in a live project. The existing automated test cases for the web shop were utilized in creating a test case for placing a sales order. The test case was extended by collecting the sales order number from the web page using Selenium2Library. Next, the according sales document was retrieved from the back-end and validated using the test library.

So far, the library had only been used from SAP NetWeaver Developer Studio. The library was bundled to an executable Java Archive (JAR) file. JAR bundles together the needed class files, including JCo. As the library was implemented using Robot Framework's remote library interface, a server is started when executing the JAR file. The server is served locally using an arbitrary free port, and imported to Robot Framework.

January 2014

In January 2014, the products part of the test library kicked off. Research and study was made on the function modules that would serve the purpose of the thesis. The scope was clarified and narrowed to fetching two kinds of products from the back-end for different test scenarios:

1. Valid product, in stock.
2. Valid product, out of stock.

The term "valid product" can be defined as follows: A product, for which an assumption can be made that placing a sales order with the product results in successfully processed sales document. The term "in stock" also needs explanation; product is in stock if the returned schedule line is within a specified time frame. The time frame can vary from customer to customer, because the logic for calculating schedule lines involves multiple factors, like working hours, working shifts, delivery routes and weekends, just to name a few. If a product's confirmed schedule line does not match the time frame, it is determined as out of stock. As the time frame varies for each customer, it needs to be taken in as a parameter from Robot Framework.

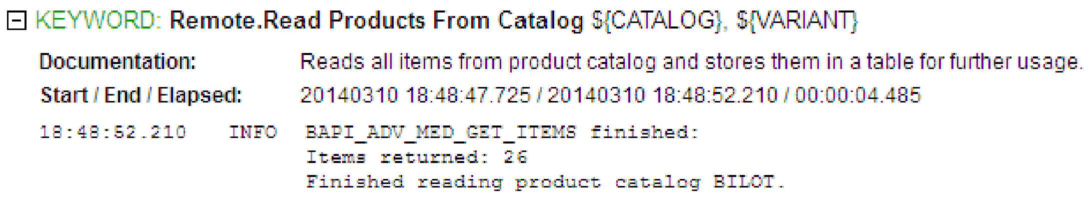
Consulting experienced SAP consultants on the topic of fetching such products programmatically resulted in the following outcome; there was no feasible way to create generic logic for fetching valid products from the back-end. This was due to customers having different logic and criteria for determining the valid products. While discussing of possible solutions, the following was suggested; simulate a sales order creation in the back-end with the product. The simulation uses the customer-specific logic for products. Based on the simulation outcome, observations can be made to determine if the product is valid.

The starting point for the implementation was to get the products that are available in the web shop. Consulting colleagues and researching on the matter brought up the concept of product catalogs. By standard, one product catalog is assigned to a web shop, and the catalog contains the products which are shown in the web shop. By reading the catalog from the back-end, the web shop's product ID's could be collected.

Continuing with the products part of the thesis was not as straightforward as the sales document validations. First issue came up with the web shop environmental setup; even though the web shop itself was connected to an SAP ERP back-end, the product catalog of the web shop was in fact stored in an SAP MDM system. The role of MDM in the web shop environment was not known in the design phase when the development environment was selected. This, together with the fact that the web shop development environment was soon to be torn down by the customer, left the project in search of a new development environment. Finding a new environment proved not to be a trivial task, as many of the ongoing web shop projects did not match the thesis scope, or getting the proper authorizations for the customer's SAP ERP back-end was not possible.

February 2014

In February, a new development environment was acquired matching the thesis scope and the development could continue. Function module BAPI_ADV_MED_GET_ITEMS was used to read the products from a product catalog stored in ERP. A screenshot of the execution log can be seen in Figure 21. The function module takes the catalog name and variant as a parameter, which are given in the Robot Framework setup file. Product catalogs can have different variants for specific language and currency needs, for example. [45]



```
KEYWORD: Remote.Read Products From Catalog ${CATALOG}, ${VARIANT}
Documentation:       Reads all items from product catalog and stores them in a table for further usage.
Start / End / Elapsed:   20140310 18:48:47.725 / 20140310 18:48:52.210 / 00:00:04.485
18:48:52.210   INFO   BAPI_ADV_MED_GET_ITEMS finished:
                      Items returned: 26
                      Finished reading product catalog BILOT.
```

Figure 21. Robot Framework log for reading product catalog items.

So far, the setup file was a single file containing the needed setup for connection details and sales document validations. The setup file was split for easier maintenance and end-user experience. Each area of connection setup, sales document validation setup and products setup was stored in its own Robot Framework setup file.

Proceeding with the development to a point of testing a full end-to-end scenario, more previously unknown details revealed. The product catalog in the ERP back-end does not unambiguously match the available products in the web shop. Three different scenarios for the mismatches were identified:

1. Product catalog is maintained manually in the back-end, and replicated to a TREX indexing server from which the web shop reads the product information. Between the replications, the updated catalog in ERP does not match the web shop catalog in TREX server. This can result in a situation where products in the back-end catalog do not yet exist in the web shop.
2. Catalog views are used in the web shop. Catalog views are used to create customer-specific views of the product catalog, enabling different customers to see different products in the web shop [45].
3. Customers use a custom replication program to determine products for the web shop from a product catalog in a way that products are left out based on custom programmatic logic.

The identified scenarios were scoped out of the thesis implementation, as handling such scenarios were determined to introduce too much complexity for the thesis. The handling of the scenarios were parked for future development. The discovery left the final development and testing without a proper developing environment for the duration of March 2014 to May 2014.

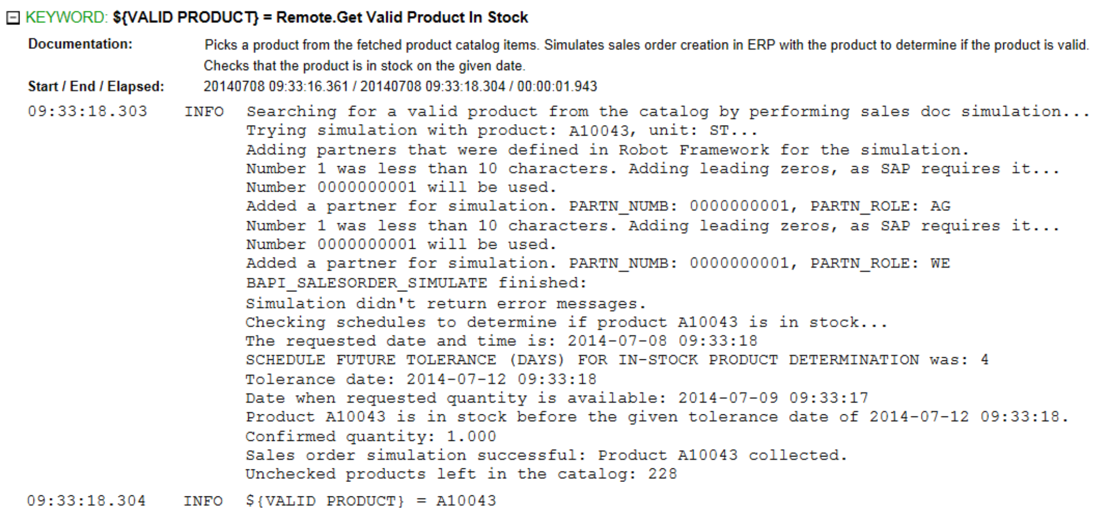
June and July 2014

Bilot had started an internal project implementing a web shop using the latest SAP e-commerce technology, hybris.[10] The web shop was integrated to the company's own demo ERP system. This provided the perfect development environment to finish the thesis development.

The work with product determination continued. Mandatory settings for sales order simulation, like sold-to party and ship-to party, were defined in Robot Framework and passed to the library. Also, a parameter to determine the time frame for in stock product determination was configured in Robot Framework. The time frame was used to define the tolerance for how much in the future can the confirmed schedule line date be to be still accepted as an in stock product.

The library calls function module BAPI_SALESORDER_SIMULATE with the defined parameters, and receives the outcome of the simulation as return tables. The return tables are checked in case error messages are received. If no error messages exist, then the returned schedule line date is compared against the specified time frame tolerance. If the confirmed schedule line date is within the time frame, product is determined and collected as an in stock product. If no error messages exist, and the confirmed date is not within the time frame, product is out of stock. Figure 22 shows the Robot Framework execution log for keyword "Get Valid Product In Stock."

[10] http://www.hybris.com. Accessed 11.7.2014



```
KEYWORD: ${VALID PRODUCT} = Remote.Get Valid Product In Stock
Documentation:        Picks a product from the fetched product catalog items. Simulates sales order creation in ERP with the product to determine if the product is valid.
                      Checks that the product is in stock on the given date.
Start / End / Elapsed: 20140708 09:33:16.361 / 20140708 09:33:18.304 / 00:00:01.943
09:33:18.303    INFO   Searching for a valid product from the catalog by performing sales doc simulation...
                       Trying simulation with product: A10043, unit: ST...
                       Adding partners that were defined in Robot Framework for the simulation.
                       Number 1 was less than 10 characters. Adding leading zeros, as SAP requires it...
                       Number 0000000001 will be used.
                       Added a partner for simulation. PARTN_NUMB: 0000000001, PARTN_ROLE: AG
                       Number 1 was less than 10 characters. Adding leading zeros, as SAP requires it...
                       Number 0000000001 will be used.
                       Added a partner for simulation. PARTN_NUMB: 0000000001, PARTN_ROLE: WE
                       BAPI_SALESORDER_SIMULATE finished:
                       Simulation didn't return error messages.
                       Checking schedules to determine if product A10043 is in stock...
                       The requested date and time is: 2014-07-08 09:33:18
                       SCHEDULE FUTURE TOLERANCE (DAYS) FOR IN-STOCK PRODUCT DETERMINATION was: 4
                       Tolerance date: 2014-07-12 09:33:18
                       Date when requested quantity is available: 2014-07-09 09:33:17
                       Product A10043 is in stock before the given tolerance date of 2014-07-12 09:33:18.
                       Confirmed quantity: 1.000
                       Sales order simulation successful: Product A10043 collected.
                       Unchecked products left in the catalog: 228
09:33:18.304    INFO   ${VALID PRODUCT} = A10043
```

Figure 22. Robot Framework log for fetching a valid product which is in stock.

In the early days of July, the thesis implementation was finished. The defined requirements for the custom library were implemented.

4.4. Developed application

The development resulted in a total of 10 Java source code files. The same functional grouping as in the class diagrams are used to demonstrate the source lines of code (SLOC) in each functional group. Some overlapping between the functional areas and the source code files exist, but the proportion is quite accurate. These are presented in Table 2.

A total of 22 keywords for Robot Framework were created. High-level keywords serve the purpose of creating a simplified level of abstraction to combine logic available in Robot Framework's standard libraries, and the custom library. The keywords fall under the following categories:

- Library setup and back-end connection
  - 5 keywords
    - 3 high-level keywords
    - 2 custom library keywords
- Sales document validation
  - 7 keywords
    - 1 high-level keyword
    - 6 custom library keywords
- Product determination
  - 10 keywords
    - 1 high-level keyword
    - 9 custom library keywords

The complete keyword documentation can be seen in Appendix 1.

Table 2. Development results demonstrated in source lines of code

Functional group	Source file	SLOC	Proportion
Back-end connection			
	UtilKeywords.java	82	
	Server.java	40	
	CustomDestinationDataProvider.java	113	
	Total	235	17%
Sales document validation			
	OrderKeywords.java	173	
	OrderJCoFunctionCalls.java	116	
	Domain.java	86	
	SalesDocument.java	114	
	HelperFunctions.java	162	
	Total	651	47%
Product determination			
	ProductKeywords.java	206	
	ProductJCoFunctionCalls.java	288	
	Total	494	36%
Overall total		1380	100%

5. TESTING

This chapter presents the testing results of the implemented test library. The defined requirements in Section 4.1 are used as a test basis. Unit tests have been conducted throughout the development phase. It was agreed that the functional requirements are tested with full end-to-end test scripts describing the real usage, with additional manual verifications conducted in the SAP back-end. The non-functional requirements presented in Section 4.1 are addressed and described how they were overcome. The final test environment was the same as the final development environment.

Functional requirements

The full end-to-end test scripts are documented in Appendix 2. It is web shop specific if orders are allowed to be made for out of stock products. In this case, orders are not allowed, and this is validated in test case 3. Table 3 presents the results of the end-to-end test cases.

Table 3. Functional end-to-end test case results

Test case	Functionality under test	Test case description	Test result
Test case 1.	Sales document validation from SAP back-end.	Test case 1: Submit a sales order and validate order status from back-end.	Pass
Test case 2.	In stock product determination from SAP back-end.	Test case 2: Fetch an in stock product from SAP back-end for sales order creation and validate the outcome.	Pass
Test case 3.	Out of stock product determination from SAP back-end.	Test case 3: Fetch an out of stock product from SAP back-end for sales order creation and validate the outcome.	Pass

Connectivity

Requirement: The test library must be usable from a test automation framework. Test cases utilizing the test library must be able to be run from a continuous integration service.

Throughout the development and functional testing, the test library was used from Robot Framework. The functional end-to-end tests were taken into use in the internal project and added to the continuous integration functional test cases. Each time developers committed their working copies of the web shop via the continuous integration service, the tests were run automatically. The continuous integration service used was a setup of Jenkins.[11]

[11] http://jenkins-ci.org. Accessed 15.7.2014

The requirement was stated to be fulfilled.

Flexibility

Requirement: Tester must be able to adapt the assertions to his/her needs by defining the pass/fail criteria for a test case.

The scope of the thesis was to perform validations on sales document header statuses. The assertions for the values are done on table's field level. The testers are able to define the pass and fail criteria for each field that they want to validate from the fetched sales document header status tables.

The requirement was stated to be fulfilled.

Maintainability

Requirement: The test library must be relatively easy to deploy into operational condition.

The test library, along with all the required third-party libraries and files, was bundled into a distributable archive file. The archive file included the Robot Framework setting files for the library, with default configurations and example end-to-end test cases demonstrating the usage of the library. The keyword documentation, installation instructions and step-by-step guide were created and uploaded to Bilot's internal wiki pages.

The requirement was stated to be fulfilled.

Performance

Requirement: Using the test library should not hinder the overall test performance by a significant amount.

The test library is communicating with back-end function modules using JCo. Therefore, a lot of the library's performance is dependent on the back-end processing. The product determination proved not to be a simple process. Due to the complexity, the product determination is identified as a possible bottleneck in the performance. The factors affecting the product determination performance are the size of the product catalog, along with how many valid products it contains for the chosen scenario. The more simulations are needed in the back-end, the more time it takes to determine the product for the selected scenario. Test cases utilizing the test library are also using other keywords, not just the test library specific. Performance tests were collected by inspecting the continuous integration system's build logs, where the automated test cases were automatically executed.

The performance of the test library was inspected in 5 different test runs. Each run consisted of a total of 21 test cases, of which 3 were test cases utilizing the test library. The three test cases were the following:

1. Basic end-to-end test case with validation of order in ERP.

2. Product availability checking against the back-end and order placing with a valid in stock product.
3. Product availability checking against the back-end and out of stock product behavior validation in the web shop.

The product catalog in the web shop setup contained 182 products, from which the in stock and out of stock products were determined.

The complete test duration listing per keyword can be seen in Appendix 3. Only the test library implementation specific keywords are shown. In addition, the total duration of the test case is shown, which includes keywords outside of the test library implementation, like navigating the web shop. Table 4 summarizes the performance test results.

Table 4. Test execution duration comparison between the test library and overall execution

	Case 1	Case 2	Case 3	Case 4	Case 5
Total duration of test library specific test cases	206 sec	203 sec	198 sec	179 sec	202 sec
Total duration of all test cases	575 sec	569 sec	568 sec	544 sec	566 sec
Proportion of the duration of test library specific test cases to all test cases	35,8 %	35,7 %	34,9 %	32,9 %	35,7 %
Total duration of the test library keywords	11,1 sec	10,3 sec	8,1 sec	5,7 sec	9,5 sec
Proportion of the duration of test library keywords to total execution	1,9 %	1,8 %	1,4 %	1,0 %	1,7 %

The performance test results are analyzed more in detail in Section 6.1.

Reusability

Requirement: The test library must be a generic solution which can be utilized in multiple SAP environments with similar requirements.

Building an environment just for the development is quite a big task, because there needs to be a complete setup of an SAP back-end, a fully configured web shop connected to the back-end and all the product configurations in place. Due to the difficult nature of setting up such a development environment, a total of two different customers' and one internal project's development environment was utilized during the implementation of the thesis. Conveniently, these three distinct web shops were all implemented with different technologies, covering the web shop technology choices of SAP. The different technologies were Internet Sales (ISA), Web Channel Experience Management (WCEM) and hybris. Some design issues were faced when switching the environment. This was due to the fact that there are multiple different configuration possibilities to be made in the back-end and in the web shop, and these can vary from customer to customer. Therefore, new design issues may be expected when utilizing the test library in a new, different setup of a web shop. Customer specific custom coding is difficult to overcome with a generic solution, and this was a known fact, but the general design issues can be solved with further development. Overall, the test library was proved to be reusable in changing environments and technologies.

User-friendliness

Requirement: The test library must be easy to use and to learn for end-users. In this case, end-users possess expert knowledge on test automation and SAP systems.

The test library is implemented for Robot Framework, so the usage is similar to test case creation with Robot Framework in general. The settings for the library require, to some extent, subject matter expertise of SAP ERP and knowledge on the web shop configuration.

The requirement was stated to be fulfilled.

6. DISCUSSION

The goal was to implement a test library for SAP systems. The purpose was also for the test solution to not require installations to the SAP system landscape. The reasoning behind it was that the test library was to be used by an SAP partner company in testing its deliveries, and installing extra content to clients' landscape can be a challenging task. Objectives were to remove the need for pre-defined, hard coded test data in the test scripts for increased test coverage and reduced maintenance in environment switches. SAP e-commerce was set to be the environment scope with selected scenarios. The test library needed to fetch data for the selected scenarios from the SAP back-end, to which the web shop was connected to. The data fetching needed to be automatic, during test execution. The library needed to enable adjustable validations for the fetched data. In this chapter, the test results are analyzed. Also, the high level objectives are revisited to check if the implementation fulfilled its expectations.

6.1. Analyzing the results

The requirements defined in Section 4.1 were fulfilled within the scope of the thesis. Regarding the reusability of the library in other SAP environments, some limitations were discovered in the implementation phase, and the findings were scoped out of the implementation. The test library was implemented by the author himself, but the idea and high level design came from Bilot. The lack of author's domain knowledge in the beginning played a role in not knowing to ask the right questions regarding the design. If SAP e-commerce scenarios would have been familiar in the beginning, the findings in the implementation could have been known already in the design phase.

The new findings came up when the development environment was switched from one SAP landscape to another. This was because there can be multiple different variations in an SAP e-commerce setup, depending on customers' requirements and configuration. For this reason, it is possible that more currently unidentified technical issues come up with further landscape switches. The new findings, as well as the identified existing ones, should be overcome with a patch upgrade for the library to fulfill the evolving requirements.

The test library was in productive usage in two different projects as part of a continuous integration testing routine, where the test execution was automatically triggered when developers committed changes to the web shop. Performance tests were collected by inspecting the test execution logs of those projects, and 5 execution logs were documented to demonstrate the performance. Each test execution consisted of 21 test cases, in which 3 test cases utilized the test library. That is 14% of the total test cases. Those 3 test cases were considered as part of the critical tests of the test suite with the purpose of testing the key functionalities of the web shop. The test execution duration for those 3 test library specific test cases took on average 35% of the total execution duration. But that percentage does not characterize the performance of the test library, as the test cases contained other keywords as well. In the 5 documented cases, the test library specific keywords took on average 2% of the total execution duration. The other keywords were about navigating the web shop and placing sales orders, thus performing full end-to-end scenarios in the web shop with login, multiple screen switches and replication of sales order to a back-end. Figure 23 shows the test library specific keyword execution duration compared to the total test execution duration.

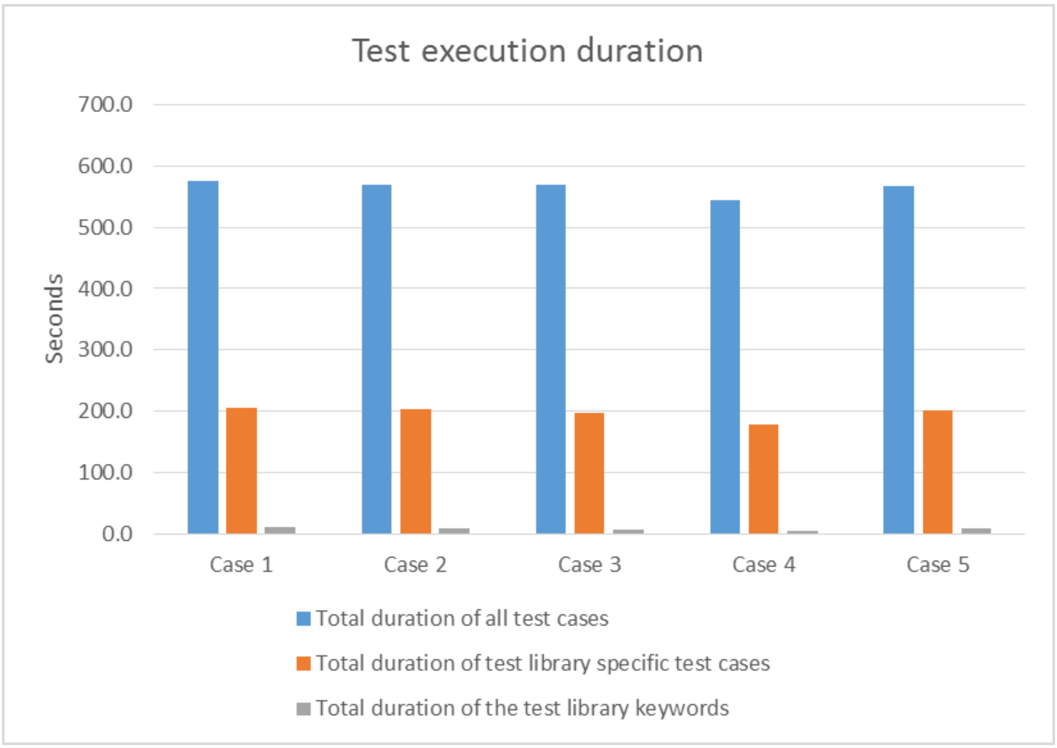


Figure 23. Test execution duration comparison.

Based on the observations, it can be stated that the test library does not hinder the overall test performance by a significant amount. Instead, the proportion is quite minimal in comparison to the total test execution duration. This can, however, change radically if the product catalog of the web shop is not proportionate regarding the in stock and out of stock products; if 99% of the products are in stock, finding the 1% out of stock products can be time consuming.

6.2. Revisiting the objectives

The test library enabled the automatic fetching of product data from the back-end. Initial settings needed to be defined in Robot Framework, reflecting the configuration of the sales document in the back-end. Such configurations were, for example, the mandatory business partners of sold-to party and ship-to party of the transaction. After defining the initial settings, products could be fetched with the calling of just one keyword in Robot Framework. The initial settings defined would be valid from system to system within a customer's landscape, therefore reducing the manual maintenance needed for product determination for the test scripts. The automatic fetching was randomized, thus providing more variety in the test data when compared to hard coded test data.

The fetching of sales document from the back-end was also implemented. Normal e-commerce test automation just covers the actions and validations in the web shop. By fetching the sales document from the back-end, the test coverage could be extended for a full end-to-end business process testing. The validation criteria was defined in Robot Framework, enabling testers to adjust and define what data is the criteria for a passed test case.

To take the test library into use in SAP test automation setup, there are no requirements to install anything to the SAP system itself. This is an important feature, as many test automation tools for SAP systems require the Business Blueprint to be maintained in SAP Solution Manager. By being a lightweight, easy-to-deploy testing tool, it can be easily utilized in a changing SAP landscape. This is useful especially to an SAP partner company, implementing SAP solutions for client companies.

The implemented test automation library was a contribution to Bilot's quality practices. The library was made available in Bilot's intranet library for distribution. The usage of the library and keyword documentation were documented in the company's knowledge sharing platform. The test library is recommended to be used in e-commerce solutions by the company's quality manager.

6.3. Personal experiences

The whole thesis implementation process from the selection of the topic to finalizing the test library was a constant learning process. I had no prior technical knowledge from SAP systems, and very limited functional experience. Robot Framework was a rather new software to me as well, with only very basic usage experience. I knew this was going to be a challenge during the implementation, and that was also the major reason for choosing the topic. From learning perspective, the topic presented a great opportunity to support my day-to-day work with SAP systems in Bilot.

The implementation took more time than expected. A lot of time were spent on studying the related topics and tackling newly discovered issues and limitations during the implementation. A major slowdown turned out to be acquiring a suitable development environment, as building one for just the development of the test library was too big a task. A development environment switch occurred a total of three times during the project. Another difficulty was the full testing of the test library. This was due to utilizing the existing automated test cases for the developed web shops, and developers made constantly changes to the system. The automated test scripts broke down from time to time, and occasionally I found myself fixing the web shop test scripts so that I could proceed with the testing of the test library.

Eventually, the project was extremely rewarding. I gained excellent know-how on Robot Framework and test automation, Java development, e-commerce scenarios and SAP systems. SAP systems are complex, and even though I had just scratched the surface of the technical side of SAP, the project made it possible to get views, aspects and understanding I wouldn't have had the possibility of through normal day-to-day work assignments. And what is best, all the gained knowledge supports closely my chosen career path.

6.4. Future development

Limitations for the selected e-commerce scenarios were discovered during the development of the test library. Some of the limitations are hard to handle programmatically. One of such limitations is the custom replication program for the catalog. Another one is a TREX server scenario where the web shop catalog is in

TREX server and in the meantime the catalog is manually modified in ERP back-end without replicating the updates to TREX. Limitation regarding the catalog view, on the other hand, was decided to be further development for the library. Catalog view is a customer specific view of available products in the web shop. The scenario is relatively common, and the test library should be able to overcome the limitation eventually.

Further area for development is the supported SAP back-ends. In addition to SAP ERP, e-commerce web shop can be connected to SAP CRM and SAP MDM systems as well. These systems have different function modules and tables than SAP ERP, and therefore require Java implementation for supporting the JCo calls and test case validations.

The selected automated e-commerce scenarios were only the first scenarios in need of implementation. In theory, the library could be utilized in any test scenario where the required interaction with an SAP system can be done with the system's function modules. The number of function modules in SAP systems is hundreds of thousands, and programmers can also create new function modules. Therefore, the extension possibilities for the test library are diverse and show promising potential.

7. CONCLUSION

In this Master’s Thesis, a test automation library for SAP systems was developed. The test library is available for Robot Framework, an open source keyword-driven test automation framework. The library was implemented in Java, utilizing open source and free tools, such as Robot Framework’s standard libraries. The only licensed product used in the implemented test library was SAP JCo. SAP JCo is used for communication with SAP systems. Its usage is governed by the SAP system license and doesn’t require additional license for SAP system license holders.

First, the key topics regarding software testing were presented. This included a big picture of the related software testing topics, and went further on to describe different test automation approaches. The concept of test automation framework and test libraries were introduced.

To further cover the background and justifications for the thesis, the target environment of SAP systems were introduced. SAP e-commerce scenario was chosen to be the scope of the thesis. It was also demonstrated how the implemented test library fills the current manual gaps in automated SAP e-commerce testing. The chosen scenarios were automatic product fetching and sales order processing validation. Products were determined and fetched from SAP ERP to be used in different scenarios in the web shop. The sales document created from the web shop order was fetched from the back-end for validation and verification of a correctly processed sales document. The chosen framework and test libraries utilized in the thesis were also described.

Then, the detailed requirements were defined. The presented design of the implemented test library consisted of an architecture diagram, as well as class diagrams representing the implemented Java classes. The development environment was introduced, consisting of real development environments for SAP solution deliveries for clients. The test library was implemented as a side project in the development environments. Moreover, the whole development process was described in detail. Eventually, the developed application results were demonstrated with the amount of source lines of code produced, as well as the number of Robot Framework keywords created.

Testing of the developed product was conducted, and the test basis used for testing were the defined requirements. Overall, the tests passed and the requirements were fulfilled. However, some findings were identified in the development phase, limiting the usage of the implementation in some of the e-commerce scenarios. The identified limitations should be overcome in future development of the library, as those were scoped out of the thesis. The reason for scoping the findings out of the thesis was due to the notable increase in complexity, if such scenarios were to be implemented within the scope.

The automated product determination from the web shop back-end reduced the amount of manual test data maintenance for the automated test scripts. The benefits of such automation is emphasized in SAP solutions development, as there can be multiple different environments for development and testing purposes, each containing different product data. In addition, the test library allowed testers to define customizable pass and fail criteria for the sales document validation. This was done in Robot Framework by defining what data fields are checked, and what values are expected. By fetching the sales document from the back-end for validation, the test coverage could be extended for a full end-to-end business process testing.

When conducting research on the matter, no other similar test tools were found. SAP provides tools for automating tests, but they are limited to SAP's traditional user interfaces. To fully cover business process test automation, the coverage has to be for non-SAP systems as well. Commercial test tools provide options for utilizing the SAP's provided test tools, enabling the testing of business processes covering SAP and non-SAP technologies. Utilizing SAP's test tools require maintenance of SAP Solution Manager and the Business Blueprint to customers' landscape, which many don't have. None of these test automation solutions fit the purpose for a lightweight deployment for a changing SAP landscape, and commercial tools also require expensive licensing.

The developed test library, along with Robot Framework, is free of charge to SAP customers and partners. The test library is ideal for SAP partner companies, implementing SAP solutions for multiple clients. As the test library does not require any installations to the client's SAP landscape, it is a lightweight, easy-to-deploy testing tool which helps in end-to-end testing across interfaces for SAP systems.

As the scope was to cover selected SAP e-commerce scenarios, there is plenty of development areas left for future work. The future work can include the handling of other SAP back-ends, more business processes and enhancing the existing e-commerce scenarios. During its development, the test library implementation was used successfully in two live projects, performing automated end-to-end SAP e-commerce testing in a continuous integration testing routine. As an overall outcome, the implementation project was deemed successful.

8. REFERENCES

[1] Leung H K N & White L (1990) A Study of Integration Testing and Software Regression at the Integration Level. Proc. 1990 IEEE Conference on Software Maintenance. San Diego, California, USA, 290-301. DOI: http://dx.doi.org/10.1109/ICSM.1990.131377.

[2] Ammann P & Offutt J (2008) Introduction to Software Testing. New York, USA, Cambridge University Press.

[3] End to end testing. URL: http://acutest.co.uk/acutest/end-to-end-testing. Accessed 18.8.2014

[4] Dustin E, Rashka J & Paul J (1999) Automated Software Testing: Introduction, Management, and Performance. New York, USA, Addison-Wesley Professional.

[5] Koomen T, Aalst L, Broekman B & Vroon M (2006) TMap Next for result-driven testing. ’s-Hertogenbosch, Netherlands, UTN Publishers.

[6] Li E (1990) Software Testing In A System Development Process: A Life Cycle Perspective. Journal of Systems Management, 41(8): 23-31

[7] Myers G, Badgett T, Thomas T & Sandler C (2004) The Art of Software Testing, Second Edition. Hoboken, New Jersey, USA, John Wiley & Sons, Inc.

[8] Bentley J (2004) Software Testing Fundamentals – Concepts, Roles, and Terminology. Proceedings of 12th Annual SouthEast SAS User Group (SESUG) Conference, Nashville, Tennessee, USA.

[9] Müller T & Friedenberg D (2011) Certified Tester Foundation Level Syllabus. International Software Testing Qualifications Board.

[10] Burnstein I (2003) Practical software testing: a process-oriented approach. New York, USA, Springer-Verlag.

[11] Kaner C, Falk J & Nquen H (1999) Testing Computer Software, Second Edition. John Wiley & Sons, Inc.

[12] Moniruzzaman A & Hossain S (2013) Comparative Study on Agile software development methodologies. E-print arXiv: 1307.3356.

[13] White K (2007) Software Test Automation 101. Proceedings of International Conference on Software Quality, Lakewood, Colorado, USA.

[14] Patton R (2005) Software Testing, Second Edition. Indianapolis, Indiana, USA, Sams Publishing.

[15] Hayes L (2004) The Automated Testing Handbook. Software Testing Institute.

[16] Fewster M & Graham D (1999) Software Test Automation. Great Britain, ACM Press.

[17] Kent J (2007) Test Automation: From Record/Playback to Frameworks. Proceedings of EuroSTAR 2007, Stockholm, Sweden.

[18] Laukkanen P (2006) Data-Driven and Keyword-Driven Test Automation Frameworks. Master's Thesis. Helsinki University of Technology, Department of Computer Science and Engineering.

[19] Cervantes A (2009) Exploring the Use of a Test Automation Framework. Proceedings of 2009 IEEE Aerospace Conference, Big Sky, Montana, USA.

[20] IEEE Std 610.12-1990 (1990) IEEE Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Engineers, New York, USA.

[21] Company information. URL: http://global.sap.com/corporate-en/our-company/index.epx. Accessed 24.8.2014

[22] Why SAP for Technology. URL: http://www.sap.com/pc/tech/strategy.html. Accessed 24.8.2014

[23] ABAP Application Server. URL: http://help.sap.com/saphelp_nw2004s/ helpdata/en/fc/eb2e8a358411d1829f0000e829fbfe/content.htm. Accessed 24.8.2014

[24] ABAP Programming. URL: http://help.sap.com/saphelp_nw73ehp1/ helpdata/en/43/41341147041806e10000000a1553f6/frameset.htm. Accessed 24.8.2014

[25] Sales Document. URL: http://help.sap.com/saphelp_erp60_sp/helpdata/en/ 0b/013acf967011d2ac740000e829fbfe/content.htm. Accessed 24.8.2014

[26] Curran T & Ladd A (2000) SAP R/3 Business Blueprint: Understanding Enterprise Supply Chain Management, Second Edition. Pennsylvania State University, USA, Prentice Hall PTR.

[27] SAP Solution Manager. URL: https://support.sap.com/solutionmanager. Accessed 24.8.2014

[28] SAP Business Suite powered by SAP HANA Cookbook (2014), pages 130-135. SAP AG. Available from URL: https://proddps.hana.ondemand.com/ dps/d/preview/6932132d32b54f58a961a755e043e21e/1.0/en-US/index. html#url=a335bd4fdd634e48aaa711abab223fa4.html. Accessed 24.8.2014

[29] Business Blueprint in Implementation Projects. URL: https://help.sap.com/saphelp_smehp1/helpdata/en/f1/30e9e8252111d5b3760050dade3beb/content.htm. Accessed 24.8.2014

[30] Business Blueprint. URL: http://help.sap.com/saphelp_smehp1/helpdata/en/45/f6da633a292312e10000000a11466f/content.htm. Accessed 24.8.2014

[31] Business Process Change Analyzer. URL: http://help.sap.com/saphelp_smehp1/helpdata/en/e7/ae57e0291149adaca93a9dcce33abb/content.htm. Accessed 24.8.2014

[32] Worksoft Automated Business Process Validation. URL: http://www.worksoft.com/files/resources/Worksoft-Automated-Business-Process-Validation.pdf. Accessed 24.8.2014

[33] SAP Solution Manager Survey (2010). Panaya Inc. Available from URL: http://go.panayainc.com/SAPSolutionManagerSurveyReport.html. Accessed 24.8.2014

[34] How Sales Documents are Structured. URL: http://help.sap.com/saphelp_470/helpdata/en/dd/55faab545a11d1a7020000e829fd11/content.htm. Accessed 24.8.2014

[35] About Sales Documents. URL: http://help.sap.com/saphelp_470/helpdata/en/dd/55fa8c545a11d1a7020000e829fd11/content.htm. Accessed 24.8.2014

[36] Domains. URL: https://help.sap.com/saphelp_nw04s/helpdata/en/cf/21ede5446011d189700000e8322d00/content.htm. Accessed 24.8.2014

[37] Robot Framework User Guide (2014). URL: http://robotframework.googlecode.com/hg/doc/userguide/RobotFrameworkUserGuide.html?r=2.8.5. Accessed on 24.8.2014

[38] SAP Java Connector. URL: http://help.sap.com/saphelp_nwpi711/helpdata/en/48/70792c872c1b5ae10000000a42189c/content.htm. Accessed 24.8.2014

[39] SeleniumHQ Browser Automation. URL: http://docs.seleniumhq.org/. Accessed 24.8.2014

[40] Function Modules. URL: http://help.sap.com/saphelp_nw70/helpdata/EN/9f/db988735c111d1829f0000e829fbfe/content.htm. Accessed 24.8.2014

[41] RFC. URL: http://help.sap.com/saphelp_nw73ehp1/helpdata/en/48/88068ad9134076e10000000a42189d/content.htm. Accessed 24.8.2014

[42] RFC Communication Scenarios. URL: http://help.sap.com/saphelp_nw73ehp1/helpdata/en/48/6ec38a20623ff6e10000000a42189c/content.htm. Accessed 24.8.2014

[43] Indexing Catalogs from ERP to a TREX Server. URL: http://help.sap.com/saphelp_crm700_ehp03/helpdata/en/46/2ddd9716d300c0e10000000a11466f/content.htm. Accessed 24.8.2014

[44] SAP JCo API 3.0. Available from http://service.sap.com/connectors. Accessed 30.10.2014

[45] Product Catalog. URL: http://help.sap.com/saphelp_crm700_ehp01/helpdata/en/06/7908be561a46928cce52ce1fb95fe7/content.htm. Accessed 24.8.2014

9. APPENDICES

Appendix 1. Keyword documentation

Library setup and back-end connection

Keyword	Arguments	Keyword location	Description
Custom Library And Suite Set Up		setup_connection.robot	A suite setup keyword. Starts a Java process for the remote library. Imports the remote library using port 8270. Calls other test suite setup keywords.
Stop Remote Server And Clean Up Suite		setup_connection.robot	A suite teardown keyword. Stops the remote server. Calls other test suite teardown keywords.
Connect To SAP Backend	@{SYSTEMPARAMS}	Library / Util	Establishes connection to SAP backend. @{SYSTEMPARAMS} has the syntax below. Parameters are separated by tabs and passed as a List to the library. @{SYSTEMPARAMS} | ASHOST | SYSNR | CLIENT | USER | PASSWD | LANG
Deregister Destination And Clean Up Test with Logoff		setup_connection.robot	A test case teardown keyword. Deregisters the destination and ends the connection to SAP back-end. This needs to be called when ending the test case. Calls keyword Deregister Destination and other teardown keywords.
Deregister Destination		Library / Util	Deregisters the destination and ends the connection to SAP back-end.

Sales document validation

Keyword	Arguments	Keyword location	Description
Define Domains And Fields For Validation		setup_order.robot	A high-level setup keyword. Each table field in SAP has a domain assigned to it. Domains define the value range that the table field uses. For the custom library, each domain and its values and their descriptions need to be defined from Robot Framework. Defines the fields to the library which are validated as well as the test case pass values for each field. This keyword calls the following keywords: 1. Define Domain 2. Define Field And Passes
Define Domain	domain name, ${DOMAIN'S VALUES AND DESCRIPTIONS}	Library / Order	Please see SAP Library definition of domains. Defines the domain for the library. As of current design, mostly used for convenience with the purpose of printing field value descriptions into Robot Framework log. If multiple domains need to be defined, call the keyword multiple times. Example: ${STATV VALUES AND DESCRIPTIONS}= Create Dictionary | ${EMPTY} | Not Relevant | A | Not yet processed | B | Partially processed | C | Completely processed Define Domain | STATV | ${STATV VALUES AND DESCRIPTIONS}

Define Field And Passes	field name, @{FIELD PASSES}	Library / Order	Defines a field for validation and sets values that pass RF tests for the specific field. If multiple fields need to be defined, call the keyword multiple times. Field's pass values are defined as a list. Example: @{PRC_STAT_H PASSES} | A | B | C Define Field And Passes | PRC_STAT_H | @{PRC_STAT_H PASSES}
Get Sales Document From ERP	${SALES ORDER NUMBER}	Library / Order	Fetches the sales document from ERP using function module BAPI_ISAORDER_GETDETAILEDLIST. Fetches three tables; Order header (HEADER), Header status (STATUS_H), Item status (STATUS_I). The sales document is stored as an object for further usage. ${SALES ORDER NUMBER} needs to be collected from the web page.
Validate Sales Document Header Statuses	@{FIELDS TO CHECK}	Library / Order	Validates the defined table fields against the defined pass/fail criteria. @{fields to check} is the list of fields which are validated. Please note that the domains of the fields need to be defined before validation (check keyword Define Domains).
Log Sales Document		Library / Order	Logs the fetched Sales Document to Robot Framework execution log, mostly for debugging purposes.

Clear Order Settings		Library / Order	Needs to be called in test case teardown. Clears the order settings.

Product determination

Keyword	Arguments	Keyword location	Description
Set Product Settings	${REQ_DATE}=${EMPTY}	setup_products.robot	A high-level setup keyword. Can be given an optional argument to define the ATP check date of format yyyy-MM-dd. If argument is not defined, requested date will be set to current date. Calls the following keywords: 1. Set Consecutive Fail Limit 2. Set Sales Header Settings 3. Set Item And Schedule Settings 4. Create Sales Partner.
Set Consecutive Fail Limit	${FAILED CONSECUTIVE SIMULATION LIMIT}	Library / Product	Products are determined via simulated sales document creation. Defines a limit for consecutive failed simulations, which is counted as a failed test case. The limit is needed so that if something goes wrong, the simulation is not done for all possible products that are in the catalog, as the catalog can be very large.
Set Sales Header Settings	${SALES_HEADER_IN SETTINGS}	Library / Product	Defines the mandatory sales header settings for the sales document simulation.

Set Item And Schedule Settings	${SALES_ITEMS_IN AND SALES_SCHEDULES_IN SETTINGS}	Library / Product	Defines the item and schedule settings for sales document simulation. Also, passes a variable ${SCHEDULE FUTURE TOLERANCE (DAYS) FOR IN-STOCK PRODUCT DETERMINATION}, which determines how far in the future can the returned schedule date be to be counted as an in-stock product.
Create Sales Partner	${SALES PARTNER SETTINGS}	Library / Product	Defines the sales partner settings for the sales document simulation. You need to maintain sold-to party and ship-to party for the simulation.
Read Products From Catalog	${CATALOG}, ${VARIANT}	Library / Product	Reads the product ID's from the defined catalog and stores them for further usage. ${CATALOG} is the name of the catalog. ${VARIANT} is the variant of the catalog.
Get Valid Product In Stock		Library / Product	Picks a product from the fetched product catalog items. Simulates sales order creation (function module BAPI_SALESORDER_SIMULATE) in ERP with the product to determine if the product is valid. Checks that the product is in stock (schedule line, confirmed date) within the defined time frame (see keyword "Set Item And Schedule Settings"). Simulation is done until a product is found, consecutive fail limit is reached (see keyword "Set Consecutive Fail Limit") or all products in the catalog have been checked.

Get Valid Product Out Of Stock		Library / Product	Keyword for selecting an out-of-stock product. If a product does not pass keyword "Get Valid Product In Stock" simulation, it is determined as an out-of-stock product.
Recycle Checked Catalog Items		Library / Product	Products are stored as a shuffled list and the library keeps track on which products have been used in simulation. If you need to get products for different scenarios within a same test case, you can use this keyword to make the already simulated products available for further simulations. Example: Your fetched catalog has 300 items. You use keyword "Get Valid Product In Stock", which simulates through randomly picked 100 items from the catalog until it finds a product in stock. Next, in the same test case, you need to get product which is out of stock and you use keyword "Get Valid Product Out Of Stock" for the purpose. Calling this keyword would not use the already simulated 100 products in its simulations, unless you call keyword "Recycle Checked Catalog Items" in between the two different product determination scenarios.
Clear Product Settings		Library / Product	Needs to be called in test case teardown. Clears the product settings.

Appendix 2. Functional end-to-end test cases

Test case 1: Submit a sales order and validate order status from back-end.				
Step number.	Step description	Expected result	Actual result	Pass / Fail
1	In test library's settings file, define the SAP back-end connection and logon details.	Settings are saved.	As expected.	Pass
2	In test library's settings file, define the sales document fields that are to be validated.	Settings are saved.	As expected.	Pass
3	In test library's settings file, define the pass criteria for each field.	Settings are saved.	As expected.	Pass
4	Using Robot Framework, create a test case performing the following logical sequence: 1. Import test library into test case. 2. Establish connection to SAP back-end. 3. Login to web shop. 4. Create a sales order with a manually determined product which is in stock. 5. Submit the sales order and collect the returned sales order ID number. 6. Using the test library, fetch the corresponding sales document from the back-end using the sales order ID. 7. Perform the validations which were defined in the test library's settings file. 8. End connection to back-end.	Test case is created.	As expected.	Pass
5	Run the test case defined in step 3. Once finished, open the Robot Framework test execution log for inspection. Verify that the test case is executed successfully.	Test case is executed successfully, ending in either passed or a failed test case.	As expected.	Pass
6	From the log, verify that test case results match the	The fields that were defined in	As expected.	Pass

	settings defined in steps 1-3.	step 1 are read from the return tables. If the fields' values match the pass criteria defined in step 3, the step is a pass. If not, test case fails stating that the field value was not defined as a pass criteria.		
7	Login to SAP back-end and manually search for the sales document using the sales order ID.	The sales document is found from SAP back-end.	As expected.	Pass
8	Open the sales document. Verify that the sales document's field values match the ones in test case execution log.	The field values in the back-end match the values that were obtained using the test library.	As expected.	Pass
Test case result: Pass				
Tested on: 14.7.2014 **Tested by: Petri Niemelä**				

Test case 2: Fetch an in stock product from SAP back-end for sales order creation and validate the outcome.				
Step number.	Step description	Expected result	Actual result	Pass / Fail
1	In test library's settings file, define the SAP back-end connection and logon details.	Settings are saved.	As expected.	Pass.
2	In test library's settings file, define the needed setup for sales document simulation and product determination settings.	Settings are saved.	As expected.	Pass.
3	Using Robot Framework, create a test case performing the following logical sequence: 1. Import the test library into test case. 2. Establish connection to SAP back-end. 3. Read product catalog.	Test case is created.	As expected.	Pass.

	4. Fetch an in stock product from the back-end. 5. Login to web shop. 6. Use the fetched product in sales order creation. 7. Submit the sales order and write the returned sales order ID number into Robot Framework execution log. 8. End connection to back-end.			
4	Run the test case defined in step 3. Once finished, open the Robot Framework test execution log for inspection. Verify that the test case is passed.	Test case is executed successfully, ending in a passed test case.	As expected.	Pass.
5	Login to SAP back-end and manually search for the sales document using the sales order ID.	The sales document is found from SAP back-end.	As expected.	Pass.
6	Open the sales document. Verify that the sales document has the fetched product, and the schedule line is within the defined tolerance for in stock product determination.	The field values in the back-end match the values that were obtained using the test library. Confirmed schedule line date is within the defined tolerance.	As expected.	Pass.
Test case result: Pass.				
Tested on: 14.7.2014				
Tested by: Petri Niemelä				

Test case 3: Fetch an out of stock product from SAP back-end for sales order creation and validate the outcome.				
Step number.	Step description	Expected result	Actual result	Pass / Fail
1	In test library’s settings file, define the SAP back-end connection and logon details.	Settings are saved.	As expected.	Pass.
2	In test library’s settings file, define the needed setup for sales document simulation and product determination settings.	Settings are saved.	As expected.	Pass.

3	Using Robot Framework, create a test case performing the following logical sequence: 1. Import the test library into test case. 2. Establish connection to SAP back-end. 3. Read product catalog. 4. Fetch an out of stock product from the back-end. 5. Login to web shop. 6. Search for the fetched product and try to create a sales order with the product. 7. Verify that the web shop does not allow sales order creation for the product. 8. End connection to back-end.	Test case is created.	As expected.	Pass.
4	Run the test case defined in step 3. Once finished, open the Robot Framework test execution log for inspection. Verify that the test case is passed.	Test case is executed successfully, ending in a passed test case.	As expected.	Pass.
5	Login to SAP back-end and manually perform sales document simulation using function module BAPI_SALESORDER_SIMULATE with the same settings as defined in steps 1 and 2. Use the same product that was determined in Robot Framework test case.	Simulation can be executed.	As expected.	Pass.
6	Verify that the sales document simulation does not return a confirmed schedule line, or that the returned schedule line date is out of the defined tolerance range for in stock product determination.	The simulation does not return a confirmed schedule line date within the defined tolerance range for in stock product determination.	As expected.	Pass.
Test case result: Pass.				
Tested on: 14.7.2014				
Tested by: Petri Niemelä				

Appendix 3. Performance test measurements

Case 1: Internal project build #119 (04-Aug-2014 05:47:13)
Total test cases: 21
Passed test cases: 18
Failed test cases: 3
Test library specific test cases: 3
Passed test library specific test cases: 3
Failed test library specific test cases: 0
Total duration: 9 min 35 sec
Total test library specific test case duration: 3 min 26 sec

Test library specific keyword duration breakdown:

KEYWORD	DURATION (seconds)
Test suite setup: OperatingSystem.Start Process	0.001
Test suite setup: BuiltIn.Import Library	0.796
TOTAL test suite setup duration of the test library keywords	0.797
TOTAL test suite setup, including all suite setup keywords	0.802
Test suite teardown: Stop Remote Server	0.003
TOTAL test suite teardown duration of the test library keywords	0.003
TOTAL test suite teardown, including all suite teardown keywords	0.252
TEST CASE: Basic end-to-end test case with validation of order in ERP	
Define Domains And Fields For Validation	0.014
Connect To SAP Backend	0.381
Validate Sales Document Header Statuses	0.004
Test case teardown: Clear Order Settings	0.003
Test case teardown: Clear Product Settings	0.004
Test case teardown: Deregister Destination	0.025
TOTAL test case duration of the test library keywords	0.431
TOTAL test case duration, including all test case keywords	166.901
TEST CASE: Product availability is checked against the backend and order is placed for an available product	
Define Domains And Fields For Validation	0.009
Connect To SAP Backend	0.033
Set Product Settings	0.021
Read Products From Catalog	1.442
Get Valid Product In Stock	7.953
Validate Sales Document Header Statuses	0.002
Test case teardown: Clear Order Settings	0.003
Test case teardown: Clear Product Settings	0.002
Test case teardown: Deregister Destination	0.023
TOTAL test case duration of the test library keywords	9.488
TOTAL test case duration, including all test case keywords	26.908
TEST CASE: Product availability is checked against the backend for an out of stock product	
Define Domains And Fields For Validation	0.008
Connect To SAP Backend	0.016
Set Product Settings	0.019
Read Products From Catalog	0.032
Get Valid Product Out Of Stock	0.281
TOTAL test case duration of the test library keywords	0.356
TOTAL test case duration, including all test case keywords	11.629

Case 2: Internal project build #120 (05-Aug-2014 05:47:49)
Total test cases: 21
Passed test cases: 18
Failed test cases: 3
Test library specific test cases: 3
Passed test library specific test cases: 3
Failed test library specific test cases: 0
Total duration: 9 min 29 sec
Total test library specific test case duration: 3 min 23 sec

Test library specific keyword duration breakdown:

KEYWORD	DURATION (seconds)
Test suite setup: OperatingSystem.Start Process	0.001
Test suite setup: BuiltIn.Import Library	0.809
TOTAL test suite setup duration of the test library keywords	0.81
TOTAL test suite setup, including all suite setup keywords	0.812
Test suite teardown: Stop Remote Server	0.004
TOTAL test suite teardown duration of the test library keywords	0.004
TOTAL test suite teardown, including all suite teardown keywords	0.251
TEST CASE: Basic end-to-end test case with validation of order in ERP	
Define Domains And Fields For Validation	0.015
Connect To SAP Backend	0.656
Validate Sales Document Header Statuses	0.005
Test case teardown: Clear Order Settings	0.003
Test case teardown: Clear Product Settings	0.003
Test case teardown: Deregister Destination	0.025
TOTAL test case duration of the test library keywords	0.707
TOTAL test case duration, including all test case keywords	164.481
TEST CASE: Product availability is checked against the backend and order is placed for an available product	
Define Domains And Fields For Validation	0.009
Connect To SAP Backend	0.015
Set Product Settings	0.022
Read Products From Catalog	2.424
Get Valid Product In Stock	5.642
Validate Sales Document Header Statuses	0.003
Test case teardown: Clear Order Settings	0.002
Test case teardown: Clear Product Settings	0.002
Test case teardown: Deregister Destination	0.023
TOTAL test case duration of the test library keywords	8.142
TOTAL test case duration, including all test case keywords	26.222
TEST CASE: Product availability is checked against the backend for an out of stock product	
Define Domains And Fields For Validation	0.009
Connect To SAP Backend	0.017
Set Product Settings	0.02
Read Products From Catalog	0.035
Get Valid Product Out Of Stock	0.565
TOTAL test case duration of the test library keywords	0.646
TOTAL test case duration, including all test case keywords	11.892

Case 3: Internal project build #123 (06-Aug-2014 05:47:56)
Total test cases: 21
Passed test cases: 18
Failed test cases: 3
Test library specific test cases: 3
Passed test library specific test cases: 3
Failed test library specific test cases: 0
Total duration: 9 min 28 sec
Total test library specific test case duration: 3 min 18 sec

Test library specific keyword duration breakdown:

KEYWORD	DURATION (seconds)
Test suite setup: OperatingSystem.Start Process	0.001
Test suite setup: BuiltIn.Import Library	0.8
TOTAL test suite setup duration of the test library keywords	0.801
TOTAL test suite setup, including all suite setup keywords	0.807
Test suite teardown: Stop Remote Server	0.003
TOTAL test suite teardown duration of the test library keywords	0.003
TOTAL test suite teardown, including all suite teardown keywords	0.247
TEST CASE: Basic end-to-end test case with validation of order in ERP	
Define Domains And Fields For Validation	0.016
Connect To SAP Backend	0.86
Validate Sales Document Header Statuses	0.013
Test case teardown: Clear Order Settings	0.002
Test case teardown: Clear Product Settings	0.003
Test case teardown: Deregister Destination	0.026
TOTAL test case duration of the test library keywords	0.92
TOTAL test case duration, including all test case keywords	162.554
TEST CASE: Product availability is checked against the backend and order is placed for an available product	
Define Domains And Fields For Validation	0.008
Connect To SAP Backend	0.017
Set Product Settings	0.023
Read Products From Catalog	1.478
Get Valid Product In Stock	4.102
Validate Sales Document Header Statuses	0.003
Test case teardown: Clear Order Settings	0.002
Test case teardown: Clear Product Settings	0.003
Test case teardown: Deregister Destination	0.024
TOTAL test case duration of the test library keywords	5.66
TOTAL test case duration, including all test case keywords	23.056
TEST CASE: Product availability is checked against the backend for an out of stock product	
Define Domains And Fields For Validation	0.006
Connect To SAP Backend	0.018
Set Product Settings	0.022
Read Products From Catalog	0.029
Get Valid Product Out Of Stock	0.654
TOTAL test case duration of the test library keywords	0.729
TOTAL test case duration, including all test case keywords	11.767

Case 4: Internal project build #124 (07-Aug-2014 05:47:23)
Total test cases: 21
Passed test cases: 18
Failed test cases: 3
Test library specific test cases: 3
Passed test library specific test cases: 3
Failed test library specific test cases: 0
Total duration: 9 min 4 sec
Total test library specific test case duration: 2 min 59 sec

Test library specific keyword duration breakdown:

KEYWORD	DURATION (seconds)
Test suite setup: OperatingSystem.Start Process	0.001
Test suite setup: BuiltIn.Import Library	0.803
TOTAL test suite setup duration of the test library keywords	0.804
TOTAL test suite setup, including all suite setup keywords	0.809
Test suite teardown: Stop Remote Server	0.003
TOTAL test suite teardown duration of the test library keywords	0.003
TOTAL test suite teardown, including all suite teardown keywords	0.254
TEST CASE: Basic end-to-end test case with validation of order in ERP	
Define Domains And Fields For Validation	0.014
Connect To SAP Backend	0.612
Validate Sales Document Header Statuses	0.007
Test case teardown: Clear Order Settings	0.002
Test case teardown: Clear Product Settings	0.002
Test case teardown: Deregister Destination	0.025
TOTAL test case duration of the test library keywords	0.662
TOTAL test case duration, including all test case keywords	145.934
TEST CASE: Product availability is checked against the backend and order is placed for an available product	
Define Domains And Fields For Validation	0.009
Connect To SAP Backend	0.016
Set Product Settings	0.022
Read Products From Catalog	0.629
Get Valid Product In Stock	2.996
Validate Sales Document Header Statuses	0.003
Test case teardown: Clear Order Settings	0.002
Test case teardown: Clear Product Settings	0.002
Test case teardown: Deregister Destination	0.023
TOTAL test case duration of the test library keywords	3.702
TOTAL test case duration, including all test case keywords	22.037
TEST CASE: Product availability is checked against the backend for an out of stock product	
Define Domains And Fields For Validation	0.009
Connect To SAP Backend	0.016
Set Product Settings	0.02
Read Products From Catalog	0.028
Get Valid Product Out Of Stock	0.451
TOTAL test case duration of the test library keywords	0.524
TOTAL test case duration, including all test case keywords	10.286

Case 5: Internal project build #125 (08-Aug-2014 05:46:54)
Total test cases: 21
Passed test cases: 18
Failed test cases: 3
Test library specific test cases: 3
Passed test library specific test cases: 3
Failed test library specific test cases: 0
Total duration: 9 min 26 sec
Total test library specific test case duration: 3 min 22 sec

Test library specific keyword duration breakdown:

KEYWORD	DURATION (seconds)
Test suite setup: OperatingSystem.Start Process	0.001
Test suite setup: BuiltIn.Import Library	0.8
TOTAL test suite setup duration of the test library keywords	0.801
TOTAL test suite setup, including all suite setup keywords	0.807
Test suite teardown: Stop Remote Server	0.004
TOTAL test suite teardown duration of the test library keywords	0.004
TOTAL test suite teardown, including all suite teardown keywords	0.251
TEST CASE: Basic end-to-end test case with validation of order in ERP	
Define Domains And Fields For Validation	0.013
Connect To SAP Backend	1.343
Validate Sales Document Header Statuses	0.016
Test case teardown: Clear Order Settings	0.002
Test case teardown: Clear Product Settings	0.002
Test case teardown: Deregister Destination	0.025
TOTAL test case duration of the test library keywords	1.401
TOTAL test case duration, including all test case keywords	165.585
TEST CASE: Product availability is checked against the backend and order is placed for an available product	
Define Domains And Fields For Validation	0.009
Connect To SAP Backend	0.021
Set Product Settings	0.023
Read Products From Catalog	2.172
Get Valid Product In Stock	4.783
Validate Sales Document Header Statuses	0.003
Test case teardown: Clear Order Settings	0.003
Test case teardown: Clear Product Settings	0.001
Test case teardown: Deregister Destination	0.024
TOTAL test case duration of the test library keywords	7.039
TOTAL test case duration, including all test case keywords	24.55
TEST CASE: Product availability is checked against the backend for an out of stock product	
Define Domains And Fields For Validation	0.009
Connect To SAP Backend	0.02
Set Product Settings	0.018
Read Products From Catalog	0.031
Get Valid Product Out Of Stock	0.207
TOTAL test case duration of the test library keywords	0.285
TOTAL test case duration, including all test case keywords	11.3