

LoadSplunker

Integration between HP Performance Center and Splunk



LUND UNIVERSITY
Campus Helsingborg

LTH School of Engineering at Campus Helsingborg
Department of Computer Science

Bachelor thesis:
Markus Jönsson
Simon Karlsson

© Copyright Simon Karlsson, Markus Jönsson

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2015

Abstract

IKEA is today using HP's application lifecycle testing environment which consists of several tools. One of the tools is Performance Center which is used to schedule and run performance tests. When a test run is finished, the tester uses LoadRunner Analysis which is another tool in HP's testing environment. This tool is used to visualize a test run in graphs. These graphs are used by the tester to manually write a static report which is sent to the stakeholders. The creation of graphs and test reports can be very time consuming and easing this process would most likely result in a higher quality of testing and therefore also result in better deliverables. This was the objective of this thesis work and was accomplished by creating a system that could replace the analysis tool with Splunk, a log analyzing tool that can manage several large amounts and different types of data. The program that was developed during this thesis work is called LoadSplunker and integrates Performance Center and Splunk. LoadSplunker is a real time java system that is informed when a test run in Performance Center is finished and automatically retrieves results from the test run needed for an analysis. The test results are then uploaded into Splunk where different views of the test results can be created and modified.

Keywords: IKEA, dashboard, Performance Center, Splunk, test, analysis

Sammanfattning

IKEA använder idag HPs applikations livscykel testmiljö som består av flera verktyg. Ett av verktygen är Performance Center vilken är använd för att schemalägga och köra prestanda test. När en test körning är färdig använder testaren LoadRunner Analysis, ett annat verktyg i HPs test miljö. Detta verktyget används för att visualisera en test körning i grafer. Graferna används av testaren för att manuellt skriva en statisk rapport som skickas till intressenterna. Att skapa dessa grafer of test rapporter kan vara väldigt tidskrävande och underlättandet av denna processen skulle förmodligen resultera i en testning av högre kvalité och därför också bättre leverabler. Detta var avsikten med examensarbetet och blev uppnått genom att skapa ett system som kan ersätta analysverktyget med Splunk, ett log analyseringverktyg som kan ta emot stora mängder och flera olika typer av data. Programmet som utvecklades under detta examensarbetet kallas för LoadSplunker och är en integration mellan Performance Center och Splunk. LoadSplunker är ett java Realtidsystem som vet när test körningar är färdiga i Performance Center och hämtar automatiskt ner det resultat från en test körning som behövs för att utföra en analys. Test resultaten laddas sedan upp i Splunk där olika vyer av test resultaten kan skapas och modifieras.

Nyckelord: IKEA, dashboard, Performance Center, Splunk, test, analys

Foreword

This thesis is written for the bachelor in Computer Science and Engineering at Lund University. The thesis work has been done at IKEA and during that time we have come in contact with many helpful people, they made our stay at IKEA pleasant and for that we are very grateful. We would like to thank our supervisor Jacek Goralski for the time spent to guide and support us. We would also like to thank Ola Berggren for the fast response on questions and for helping us in general. At Campus Helsingborg we would like to thank our academic supervisor Christian Nyberg at the department of information technology for his guidance throughout this thesis work.

List of contents

1 Introduction	1
1.1 Background and purpose	2
1.2 Goal	4
1.3 Problem specification	5
1.4 Delimitations	5
2 Technical background	7
2.1 Application Lifecycle Management (ALM)	7
2.1.1 Virtual User Generator	7
2.1.2 Controller	7
2.1.3 Performance Center	7
2.1.4 Analysis	8
2.2 Splunk	8
2.2.1 Apps	9
2.2.2 Dashboards	9
2.2.3 Monitors	10
2.2.4 Searches	10
2.3 Application Programming Interfaces (APIs)	11
2.3.1 HP ALM Performance Center	11
2.3.2 Splunk	11
2.3.3 HP ALM Site Admin	11
2.4 Used libraries	11
3 Method	13
3.1 Information gathering	14
3.1.1 LoadRunner and Performance Center	14
3.1.2 Splunk features	14
3.1.3 IKEA's requirements of the product	14
3.1.4 Development solutions	15
3.2 Implementation	15
3.3 Testing	15
3.3.1 Performance Center test server	16
3.3.2 Splunk test server	16
3.3.3 cURL	16
3.4 Validation	16
3.4.1 Communication with IKEA	16
3.5 Analysis	17
3.6 Source criticism	17
4 Result	19
4.1 Download result from Performance Center	20
4.2 Translate data	20

4.3 XML format	20
4.4 Splunk settings	23
4.4.1 Splunk forwarder	23
4.4.2 Splunk server	25
4.5 Searches	26
4.6 Dashboards	27
4.6.1 Alternative one	27
4.6.2 Alternative two	29
4.6.3 Alternative three	29
4.6.4 Export PDF	32
4.7 User management	33
4.7.1 Real time system.....	34
4.7.2 ScheduleThread.....	34
4.7.3 ResultThread	35
4.7.4 SplunkThread.....	36
4.7.5 Logging	36
4.7.6 Settings.....	37
5 Conclusion	39
5.1 Future possibilities	42
5.1.1 Alerts.....	42
5.1.2 Chart overlay.....	42
5.1.3 Customized dashboards	42
5.1.4 Dashboard legend.....	43
5.1.5 Extended logging	43
5.1.6 SideView utils.....	44
5.1.7 SLA rules	44
5.1.8 System test	44
6 Terminology	45
7 Sources	47
7.1 Trusted sources	47
7.2 Untrusted sources	49
8 Appendix	51

1 Introduction

With over 345 stores in 45 countries and around 775 million visits each year, IKEA is one of the largest companies in the world. IKEA has over 9000 servers and each year they handle about 250 million cash transactions and 1.2 billion visits on their website. With this in mind one can understand that IKEA is very dependent on a working IT environment. To secure that every new web release works as expected, IKEA executes a large amount of performance tests. These tests are executed to determine how the software affects the stability and responsiveness of a system under a specified workload. For an effective product development, communication between developers and testers is essential. In some companies developers and testers work in different countries which can complicate communication. IKEA is one of these companies who outsource most of their development overseas.

After every test result, a report is written and sent to the developers and/or other stakeholders. Writing these reports can be very time consuming. It would be of great value to IKEA if these reports could be generated automatically and be made dynamic so that information easily can be added and modified. This thesis is going to be about creating a system that will reduce the time IKEA IT has to spend on creating test reports. This thesis work is for the bachelor's degree in Computer Science at Lund University.

1.1 Background and purpose

This thesis work will be done on the behalf of IKEA IT Test & configuration center. They are currently performing performance tests with the help of Application Lifecycle Management (ALM), a suite owned by Hewlett Packard, see 2.1. ALM includes three components for automated testing and non-automated reporting, LoadRunner Virtual User Generator, Performance Center and LoadRunner Analysis.

The workflow for testing with ALM has the following steps:

1. *Script creation*

In this step scripts are developed in LoadRunner Virtual User Generator. The scripts will replicate a real user's behavior and is called a virtual user. They are then uploaded and utilized in Performance Center.

2. *Creation of a test run*

The creation of test runs is done in Performance Center. A simulation of the system in production can be done by loading virtual users on load generators. These are chosen by the tester as well as which and the amount of virtual users. A test run can in this way run virtual users with different behavior at the same time.

3. *Analysis of the test result*

The test result is analyzed with LoadRunner analysis tool. Here different graphs are created and modified. These graphs are then copied and pasted into a report.

4. *Analysis of the test report*

The developers and other stakeholders will read the report and use the information to fix flaws in a system. If information is missing, the stakeholder will ask for a new report.

The test department of IKEA IT is not pleased with the LoadRunner Analysis tool as it is very ineffective and unnecessarily complex. For example, when the developers receive a test report from the test department they usually ask for missing information about specific parts of the tests. To include this specific part a new report has to be created by the test department. This workflow is not efficient because more time is spent on writing different reports than testing.

The purposes of this thesis work was to develop a system that could replace the LoadRunner Analysis with Splunk a log analysis tool created by the American multinational corporation Splunk INC, see 2.2. The Splunk product has functionality to capture data and use it to generate graphs, reports, alerts, dashboards and visualizations. Splunk can handle large amounts and different types of data.

The system developed in this thesis work is called LoadSplunker. LoadSplunker is a further development of a LoadSplunker prototype, developed in the course 'project year 3' included in the bachelor for Computer Science and Engineering at Lund University. The prototype had the functionality to connect itself to both a database server and Splunk. It also created XML files of all the databases in a database server to a folder and then uploaded the XML data into Splunk.

1.2 Goal

The main goal of this thesis work is to replace the LoadRunner Analysis tool. For this it's important that Splunk is able to show the same information as LoadRunner Analysis, in other words, any data loss would not suffice. The graphs in Splunk will include options for the stakeholders so they can easily view specific information about a test. Some of that information might be sensitive and therefore a role-based system is needed in Splunk. It is important that the role-based system only grants access to the correct users. Another goal was to find a possible way to ease the report management by making it more flexible and reducing the amount of manual work needed.

The ideal test reporting at IKEA is shown in figure 2. A test run will first be prepared by the tester and when a test run has finished, LoadSplunker uploads its result data into Splunk. In Splunk, default dashboards can be created and then modified by the tester as a replacement of the PDF report.

Figure 2 can be compared with how the test reporting is done today which is shown in figure 1. A test run will first be prepared by the tester and when the test run is finished, the tester will use LoadRunner Analysis to create one or more reports which are sent as a PDF to stakeholders.

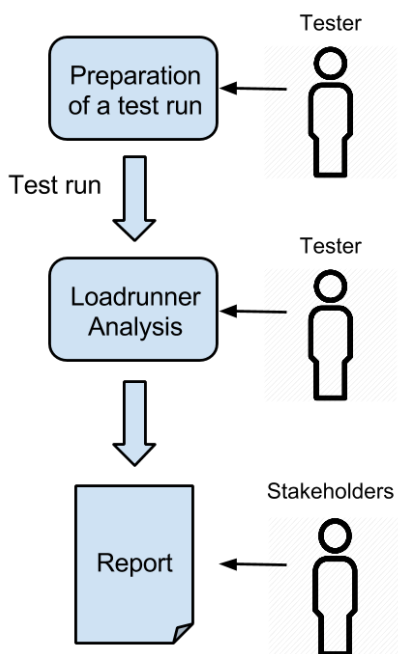


Figure 1: This figure shows how test result is compiled today.

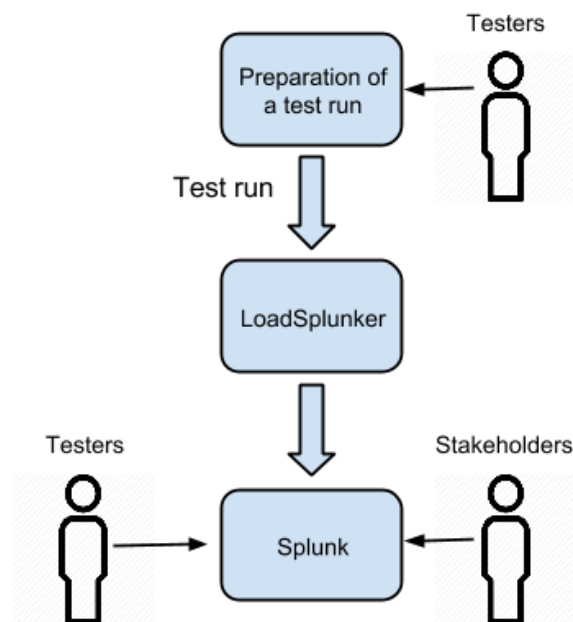


Figure 2: This figure shows how test results ideally would be viewed when using LoadSplunker.

1.3 Problem specification

For the goal of this thesis work to be met there are four main problems that have to be solved, these are an automatic data flow between Performance Center and LoadSplunker, an automatic report creation within Splunk, access rights to these reports for users at IKEA and how to install this system in IKEA's environment.

The problem of finding an automatic data flow between Performance Center and LoadSplunker was divided into three questions:

- *Is it possible to gather all data points from performance tests without any data loss?*
- *How can LoadSplunker be informed when a test is finished?*
- *How can LoadSplunker access result files from tests in Performance Center?*

The problem of an automatic report creation in Splunk was divided into two questions:

- *What information is the most interesting for testers, developers and other stakeholders?*
- *Is it possible to make use of a template to create a default view of the information?*

The problem of implementing access rights for IKEA users in Splunk was divided into three questions:

- *How to gain access information about users at IKEA?*
- *How to create restriction in Splunk using that information?*
- *How to keep Splunk updated with changes on IKEA's users?*

The last problem of installing this system at IKEA will be solved by answering the following question:

- *How can this system be installed in IKEA's environment?*

1.4 Delimitations

For this thesis work, IKEA will provide laptops and a test environment with a Performance Center server and a Splunk server. The Performance Center server will be in version 12.01 and the Splunk server in version 6.2.0. LoadSplunker will be developed to be compatible with these versions of the tools.

The thesis work will not include interviews with stakeholders for IKEA's test reporting because it is out of the scope of this thesis work. Instead some examples of graphs that are required for basic test reporting will be recommended by the thesis workers supervisor at IKEA. Large test results will also be created by the supervisor at IKEA for the same reason.

2 Technical background

This section provides the technical background needed to understand this thesis work. Some prior knowledge of the reader is required. The knowledge required is a basic understanding of API, Java, ALM and XML.

2.1 Application Lifecycle Management (ALM)

ALM [26] is owned by Hewlett Packard and is used by IKEA for testing software. It is one of the most used test environments today. The part of ALM used for performance testing by IKEA consists of three components: Virtual User Generator, Performance Center and Analysis.

2.1.1 Virtual User Generator

In Virtual User Generator [27] the tester develops scripts to simulate the actions of real users. The action of users varies greatly between projects. The goal of these scripts is to simulate different scenarios, checking that the software fits the requirements for it to be put into production.

2.1.2 Controller

The scripts created in Virtual User Generator can be uploaded into Controller [28]. Controller simulates users by executing scripts on load generators. The amount of users and load generators varies and are chosen by the tester. Controller measures how the system's performance is affected. CPU and memory utilization are examples of what is measured. After the test run is finished it can be opened in LoadRunner Analysis for an analysis of the test result.

2.1.3 Performance Center

Performance Center [29] is an extension of Controller and is used by IKEA. Its functionality is mainly to schedule test runs which is not possible with Controller. The extension of Controller is implemented in Performance Center by letting the tester choose a Controller unit at the scheduling of a test run. Test runs can contain several different types of test results such as an output log, raw results, analyzed results, a rich report and a HTML report, see figure 3. The analyzed results contain data from the test run, this is the data used for an analysis in LoadRunner Analysis. The same data will be utilized by LoadSplunker for the analysis in Splunk.

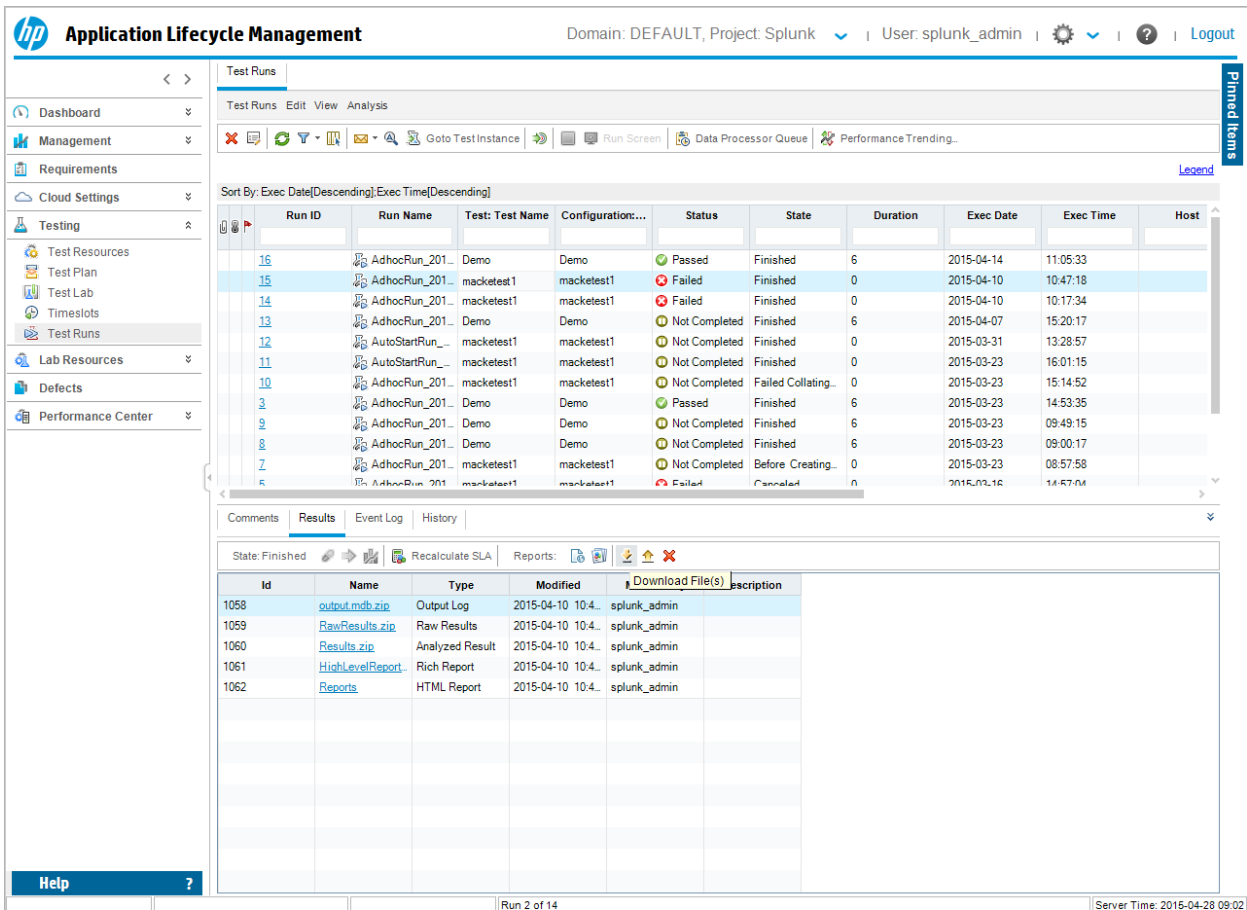


Figure 3: This figure is a screenshot in Performance Center showing a list of runs and different types of results for the selected run.

2.1.4 Analysis

Analysis [30] is the program where testers analyze the test results visually. It allows testers to create specific visualizations where problems are presented and described for stakeholders. These visualizations can be many different types of graphs, for example summarizing bar charts or line charts displaying average response times.

2.2 Splunk

Splunk [31] will be used for analysis of test data from Performance Center. Splunk has a web interface which means that it can be used from all devices with a web browser. Splunk is a log analyzing tool focusing on handling large amounts and different types of data. Handling large amount of data is important since there can be several of measurements in a test run. Handling different types of data is important since it allows the tester to correlate data in test results with any other data. Splunk can collect data from virtually any source in real time. This data can then be searched, monitored, analyzed and visualized in different forms such as graphs and diagrams.

2.2.1 Apps

An app [1], see figure 4, in Splunk is typically used to address several use cases and can contain one or more views. Examples of supported views [24] in Splunk are dashboards, forms and searches. A view can show visualizations [25] such as chart, event listing, map, table or single value. Different types of charts are area, bar, bubble, column, filler gauge, line, marker gauge, pie, radial gauge and scatter.

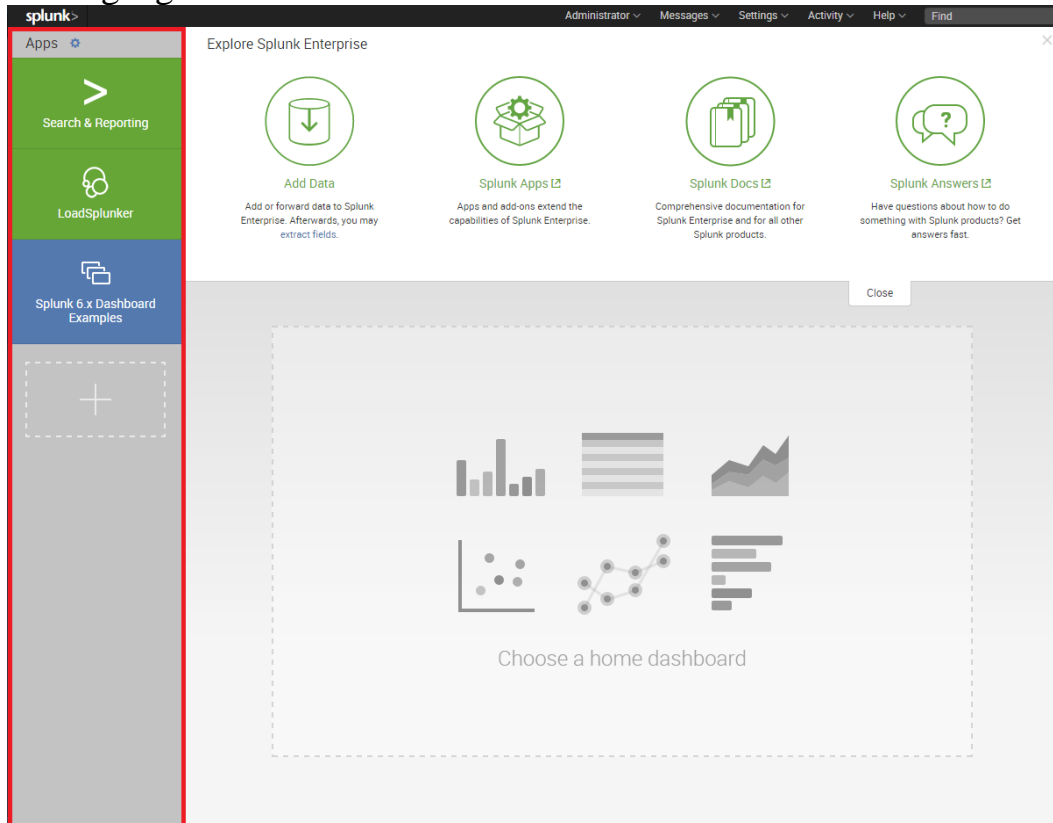


Figure 4: This figure shows the home page in Splunk with apps to the left.

2.2.2 Dashboards

A dashboard [7] is a type of view that is associated with an app in Splunk. A dashboard can contain one or more visualizations. The visualization objects within a dashboard are often generated by searches. All dashboard views are defined in XML. JavaScript and CSS can be added to develop more advanced dashboards. Input fields can be added in dashboards so that users can modify the visualizations. Dashboards with inputs are called forms in the XML source but in this report the word dashboard will be used for forms as a simplification. Drilldowns [9] can be used inside dashboards to provide a detailed look at specific data. Drilldowns are triggered by clicking different components inside a dashboard. Dashboards can also make use of tokens. Tokens [8] are used to pass values inside dashboards and can be predefined or set by input fields. The value of a token can be used inside the searches of the dashboard.

2.2.3 Monitors

A monitor [32] is a type of data input in Splunk. Monitors are designed to import data from files, directories, script outputs and network ports. If a directory is chosen to be monitored all the data inside is imported, even data in subdirectories. A batch [38] is the same as a monitor with the difference that it will delete files after Splunk has finished indexing them. All data imported into Splunk is divided into events [22]. It can be specified how the imported files are divided into events, default is on row break.

2.2.4 Searches

Searches [23] can be used in Splunk to gather and manipulate data for visualization in event, table or chart form, see figure 5. The searches are done with Splunk's own search language which resembles SQL but has more functionality. The visualization from a search can be added to a dashboard where visualizations can be viewed side by side.

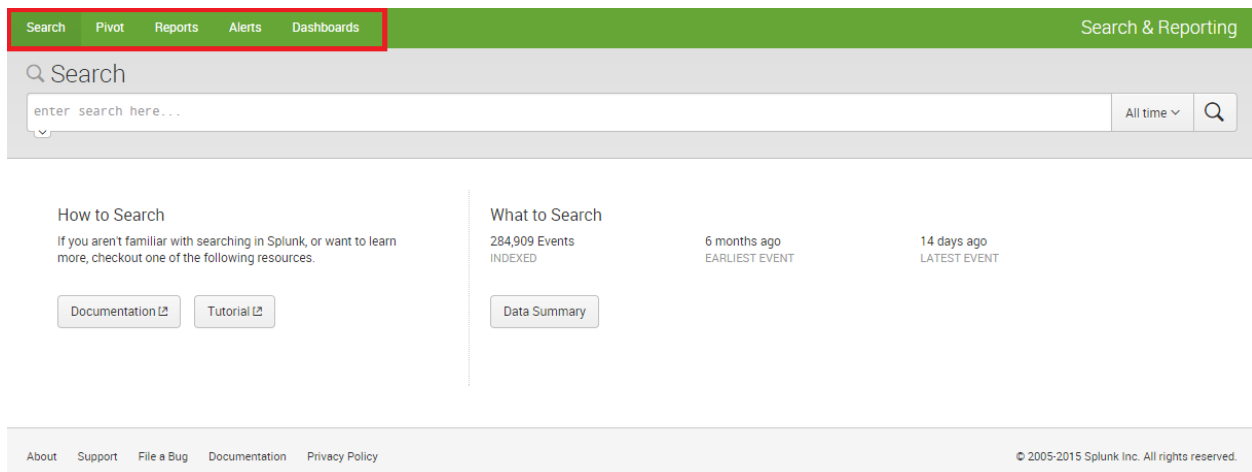


Figure 5: This figure shows search inside the default Splunk application Search & Reporting. During this thesis work, mostly search and dashboard were used in the menu to the left at the top. Here in search one can search for data inside the Splunk system and then save the result in a dashboard.

2.3 Application Programming Interfaces (APIs)

2.3.1 HP ALM Performance Center

The Performance Center API [2] is a REST API. Representational State Transfer (REST) [4] is an architecture style for network applications using only HTTP requests. The REST architecture works by separating resources through giving them their own URIs, also called endpoints. This API provides functionality for creating, managing and running tests in Performance Center without using the user interface. The API also provides functionality to download test results.

2.3.2 Splunk

The Splunk API [5] is a REST API. REST is explained in section 2.3.1. This API provides methods for accessing every feature in Splunk.

2.3.3 HP ALM Site Admin

The Site Admin API [2] is a COM-based API. Component Object Model (COM) APIs [19] are used to enable software components to communicate within Microsoft Windows operating systems. This API allows integration with third-party tools by providing functionality to organize, manage and maintain all ALM users, projects, domains and connections. It also provides functionality to query the content in the SQL database used by Site Admin.

2.4 Used libraries

For easier implementation of functionality, LoadSplunker makes use of a few Java libraries. A short explanation is given on each of these libraries in this section.

Com4j

This library is a bridge between Java and COM-based APIs. Com4j [11] does not belong to any organization and was started in December 2010 now consisting of 214 members. It is used by LoadSplunker for communication with the Site Admin COM-based API.

Jackcess

This library is for reading and writing to Microsoft Access databases developed by Health Market Science. Jackcess [12] is used by LoadSplunker to read the Microsoft database files included in test results from Performance Center. For the jackcess library to function, the following libraries had to be included:

- *Commons Lang*
- *Commons Logging*
- *Log4j*
- *JUnit*
- *POI*

They have all been created by the Apache Software Foundation.

Jersey

This library is used for communication with RESTful APIs. For the jersey library [13] to function, the following library has to be included in the project it is used:

- *Jax-RS*

Both the javax.ws.rs and the jersey library were created by the Oracle Corporation.

Joda time

This is a library for handling date and time in Java, it was created by Joda.org. [14]

Ljrt

This is a real time system library for java, created by the faculty of engineering in Computer Science at Lund University [33]. It is used to create the periodic threads in LoadSplunker.

Log4j

This is a library for logging in Java, created by the Apache Software Foundation. It is used for the logging of LoadSplunker for easier error management. [15]

3 Method

This section will cover methods, tools and rules that were used in the process of producing the result of this thesis work. This section will also cover an analysis of the workflow and how the thesis workers worked with source criticism.

At the start of this thesis work Kanban [36] was set as the project methodology. It was mainly chosen because of its ability to respond to change and that there are no prescribed roles. A very simple Kanban board was used with only two columns, to-do and done. The WIP limit used was implicitly set to two. This was because both thesis workers had to be able to work at the same time. The WIP limit was not set higher than two because of the belief that a person works better when focusing on only one thing at a time.

Google Drive was used for documentation because of its easy access from any client and for the thesis workers to be able to work in the same documents at the same time. For the ability to trace back in the thesis work, weekly protocols were created where everyday thoughts, decisions and tasks in progress was noted.

A meeting with the supervisor at IKEA was held discussing the resources needed to carry out this thesis work. Needed resources were IKEA laptops, IKEA pass card and a workplace at IKEA. The laptops and workplace were essential because some servers could only be reached from the laptops using IKEAs own network. After this, the rest of the thesis work was divided into two main problems. One problem was the development of LoadSplunker and the other was the search for reporting possibilities in Splunk. Implementation and development in Splunk was less of a technical challenge than implementing the integration. Considering this, the integration was prioritized giving IKEA the possibility to improve the reporting in Splunk.

To solve the problems specified in 1.3, each one was divided into tasks and listed in the to-do column. Each task went through a workflow of four steps, see the list below:

- Information gathering
- Implementation
- Testing
- Validation

3.1 Information gathering

During this thesis work, different methods for gathering information have been used depending on which of these four main domains the information was gathered from:

- Performance Center and LoadRunner features
- Splunk features
- IKEA's requirements of the product
- Development solutions.

3.1.1 LoadRunner and Performance Center

To gather basic knowledge about the ALM test environment, time was spent experimenting in the graphical interface by creating simple test runs. To gain knowledge on information specific for IKEA, employees in the test department were consulted. This information was used to create a basic understanding of how LoadSplunker and Performance Center could be integrated. This information was then used to write the code to download test results from Performance Center.

3.1.2 Splunk features

To explore the possibilities with Splunk, an example test result with a large amount of data was created by the supervisor. If the result was opened in LoadRunner Analysis, default graphs that were created could be observed. The thesis workers then recreated these graphs using the same data in Splunk. In this way it could be checked that everything that can be done in LoadRunner Analysis also can be done in Splunk.

3.1.3 IKEA's requirements of the product

The requirement for LoadSplunker to work at IKEA was found in two ways. One was by consulting the supervisor at IKEA because this person had good knowledge of using ALM and also what information that is valuable to see in a report. The other way was to explore the test environment at IKEA, see 3.3.1 and 3.3.2.

3.1.4 Development solutions

The design of LoadSplunker was modified several times during the thesis work. This was mainly because one piece of the system was developed at a time. Adding new parts to LoadSplunker often resulted in code having to be changed. If all the parts had been thoroughly specified before implementation it could have resulted in less rework and a more structured system. An implementation specification could for example have been made. However it is not until the implementation is tested that one can know if a solution is possible. To design the LoadSplunker system before implementation would therefore risk large amount of rework as well. In worst case the entire design would have to be changed.

3.2 Implementation

The development environment used for Java development was IntelliJ for its well-built features like code navigation, debugging, auto-completion and refactoring. For parallel coding, GIT was used with a repository at a Linux server at IKEA. For the most part of the development, documentation on Splunk and ALM was used. When some needed functionality wasn't documented a trial and error approach was instead taken to solve the problem. When an error occurred during development, a solution was looked for on the internet. A solution could sometimes be found in forums where other users have had the same errors. However, the answers in forums are not always as comprehensive as needed but they will often point in the right direction.

3.3 Testing

During the development of LoadSplunker a test environment at IKEA was used. This test environment consisted of two servers, a Performance Center server and a Splunk server. Testing LoadSplunker was done by creating dummy tests in the Performance Center server and verifying that the system handles the test results correctly all the way into the Splunk server. The testing part of the workflow was done very frequently by running LoadSplunker using the main method to test different parts of the code. The focus of this thesis work lied on finding and informing IKEA on the possibilities of this system. Therefore almost all energy was put into finding out if a solution was reliable and less energy was put into testing if that solution was implemented problem free.

3.3.1 Performance Center test server

To test the integration between Performance Center and LoadSplunker, a test environment with Performance Center was created. The environment was used to test the LoadSplunker integration without disturbing any systems in production at IKEA. The environment was also used to decrease the chance of failure at installation and to avoid wasting time setting up the environment locally on each laptop used by the thesis workers. In the Performance Center environment test results were created. In Performance Center the thesis workers ran dummy test to get data into Splunk. However these dummy tests did not generate enough data for experimenting with different views in Splunk. IKEAs test results can contain a large amount of data. Therefore LoadSplunker must be able to process the same size of data to work in production. For this an example result with large amount of data was provided by the IKEA supervisor.

3.3.2 Splunk test server

To test the integration between LoadSplunker and Splunk, a Splunk test environment was created at IKEA. This was also used to allow both thesis workers to work with the others dashboard implementations. MobaXterm [35] was used for remote access to the server for configuration.

3.3.3 cURL

When developing the communication between LoadSplunker and Splunk REST API cURL was used to test the API endpoints since the Splunk REST API documentation uses it in its examples. It was also used to find syntax errors easier. The cURL commands were later translated into Java code.

3.4 Validation

Validation was chosen as the last step of solving a task. This was because many solutions found during development have shown themselves not to work and therefore time has been saved by not validating them before. The downside with this method is if a solution would show itself to not meet IKEAs requirements at all, the implementation and testing of the solution would have been wasteful. Fortunately this never happened during this thesis work but there were situations where the thesis workers did not really know which solution that would fit IKEA. In these cases, all different alternatives were researched and presented to the supervisor at IKEA.

3.4.1 Communication with IKEA

To validate significant solutions with IKEA, good communication was needed. Throughout the thesis work, meetings with the supervisor at IKEA were held about once a week. Because the supervisor at IKEA works in Älmhult and the thesis workers in Helsingborg, meetings could not be held

that often in person. Instead video and WebEx played a key role in the communication. A WebEx meeting is a telephone conference with the possibility to share computer screens. For less important communication either email or an internal chat at IKEA was used. A couple of times meetings in person were held at IKEA either in Älmhult or Helsingborg.

3.5 Analysis

An agile and lean project methodology like Kanban worked well for this thesis work. Kanban made it easy for the thesis workers to change their priorities when needed and when working side by side no more prescribed rules were wanted.

The communication between the thesis workers worked well. When not working in proximity, Skype was used for distant communication. The thesis workers often worked with different solutions while keeping each other updated. This results in actually working with two solutions at the same time, trying to provide thoughts on both. Working this iteratively makes it hard to get a perspective of what has changed since the change is continuous. This was experienced as both effective and challenging.

The thesis workers tried to document this thesis work in a document during the development process. A disadvantage was that almost everything has to be rewritten when more knowledge is gained. The considered improvement on this was to write this thesis after the thesis work is finished and simply documenting everything in the daily protocol. Keeping in mind that one should not delete anything used during the thesis work so that it can be found for the documentation later.

3.6 Source criticism

The information searched for during this thesis work has been either a solution to a practical problem or a specification on how some tool works. Solutions to a practical problem were always tested if they worked and therefore the credibility of the source wasn't important. Untrusted sources encountered were mostly forums where someone can ask a question and anyone can answer. When reading an answer on a forum, consideration was taken on the reputation of the user and the amount of votes. This was just a secondary precaution to not waste time testing impossible solutions. When considering a solution given by an untrusted source, the solution was tested and validated more often than from a trusted source. This was because of the skepticism that the solution could work.

The information on how a tool works were mostly taken from the documentation which is considered a reliable source. Both Splunk and HP

Performance Center had documentation on their programming interfaces. Some information was given to the thesis workers by IKEAs employees and since they are experts in their testing tools and environment this was also considered credible information. Another credible source that was used is IKEAs own internal wiki where information between colleagues can be shared.

4 Result

The result of this thesis work consists of the LoadSplunker system and the development of dashboards for visualization of test results in Splunk.

LoadSplunker is a real time system and an integration of Performance Center and Splunk. The integration is done by downloading analyzed test results from Performance Center which contain raw data from events during the test run. Then LoadSplunker translates the analyzed result to XML. The translated XML data is then imported and parsed in a local Splunk instance before it is forwarded to another Splunk server. The second Splunk server is the final destination, i.e. where the data will be analyzed by testers and other stakeholders. For LoadSplunker to accomplish all this it has to communicate with several components, see figure 6.

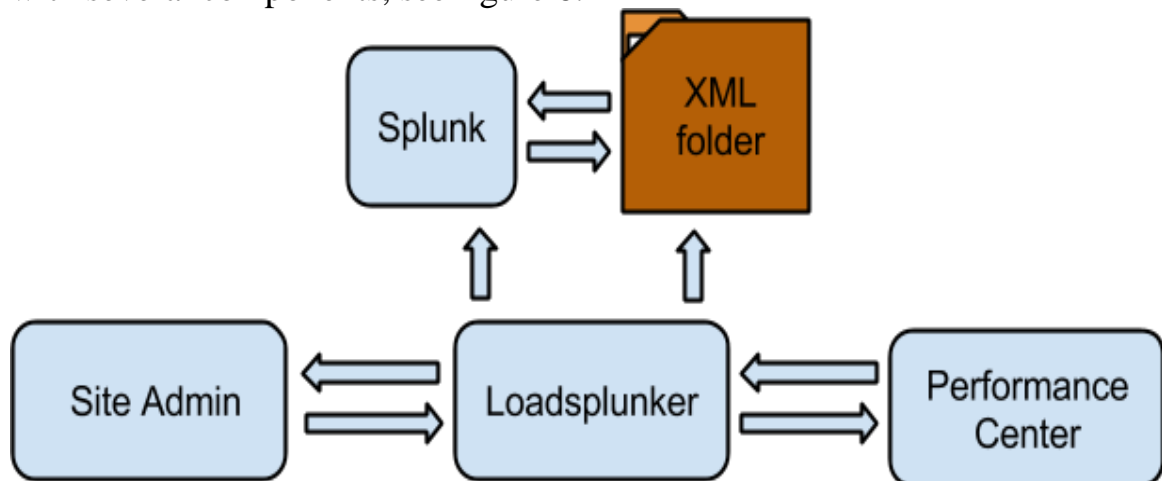


Figure 6: This figure shows the LoadSplunker's communication.

Communication between LoadSplunker and Performance Center

LoadSplunker uses a REST-based API to communicate with Performance Center. Analyzed data is downloaded through this communication.

Communication between LoadSplunker and Site Admin

LoadSplunker uses a COM-based API to communicate with Site Admin. Through this API, information is gathered about scheduled test runs, users and their roles.

Communication between LoadSplunker and Splunk

LoadSplunker will make use of two Splunk instances. A local instance used as a forwarder and an external Splunk server used as the receiver. LoadSplunker uses the Splunk REST API to create roles and users.

Communication between LoadSplunker and XML folder

LoadSplunker uses a folder for storage of XML data.

4.1 Download result from Performance Center

To integrate Performance Center and Splunk, test results from Performance Center have to be downloaded and then imported into Splunk. HP ALM Performance Center REST API was used to download these test results. A test result in Performance Center is uniquely identified with domain, project, run and result id. There are different types of test results in Performance Center. Analyzed results are the ones used in this system because they are the ones containing the raw data collected during the test run. This means that if a test run does not have an analyzed result, that run will not be handled by this system. HP ALM Site Admin API is used for this system to know when a test run should be downloaded and from which domain, project and run. This API is 32-bit COM-based and therefore it will only work on a Microsoft Windows machine with a 32-bit Java Runtime Environment in version 1.8. Downloaded test result is inside a zip archive so it has to be extracted. This is done by LoadSplunker using the standard java library `java.util.zip.ZipEntry` and `java.util.zip.ZipInputStream`.

4.2 Translate data

In test results downloaded from Performance Center, there will be a Microsoft database file containing all the measured data points during the test run. The test result will also have a folder SLA Config which contains `SLAConfiguration.xml` and in this file Service Level Agreement (SLA) rules are defined. SLA is an agreement between the customer and a service provider. The rules in the SLA are requirements on the software, how it should behave and respond in different situations. To read the Microsoft database file the `jackcess` library was used. The LoadSplunker system iterates through the database and writes it to XML files. Tables generated in Analysis are ignored in the translation since these only contain already manipulated data. Before translating each table the system reads the table result which contains the start time of the test run. This is used to convert the time field which is included with every data point. The time field is split into two attributes, absolute time and relative time. Absolute time can be useful if you want to relate to other events or logs during that time. Relative time might be used to compare test runs within a project.

4.3 XML format

Test results handled by LoadSplunker contain large amount of data, all this data had to be imported into Splunk. One standard format had to be chosen for all this data, the chosen format was XML. The XML format is represented in the following way: each row from a database table is represented as a row element, see figure 7. The attribute name in the table element is the name of the table. For every row there exist one element corresponding to columns in

the database table. XML was used since Splunk knows how to read elements in an XML file, XML is also easily read by humans. Even though XML is a very large format the advantages of a transparent and portable format overweighs the loss of efficiency in memory and computing resources. The XML files will be deleted when Splunk is done indexing them, so even though several spaces may be required, it is only for a short period of time.

```
<table name="Event_meter">
...
<row>
  <EventInstanceID>187643</EventInstanceID>
  <EventID>49</EventID>
  <EndTime>6784.555</EndTime>
  <Value>0.046802</Value>
  <Status1>1</Status1>
  <GroupID>20</GroupID>
  <VuserID>0</VuserID>
  <HostID>0</HostID>
  <ScriptID>20</ScriptID>
  <ResultID>0</ResultID>
  <ThinkTime>0.0</ThinkTime>
  <WastedTime>0.0</WastedTime>
  <LocationID>0</LocationID>
</row>
...
</table>
```

Figure 7: This figure shows an example of data in XML format.

In the analyzed test results, SLA rules are provided in XML format, see figure 8.

```
<Rules>
...
  <ComplexLoadRule>
    <Measurement>transaction_response_time</Measurement>
    <SubMeasurement>vuser_init_Transaction</SubMeasurement>
    <CriteriaMeasurement>running_vusers</CriteriaMeasurement>
    <CalcDirection>less_than_equal</CalcDirection>
    <StartLoadValue>NegativeInfinity</StartLoadValue>
    <MiddleValue>1</MiddleValue>
    <EndLoadValue>PositiveInfinity</EndLoadValue>
    <ThresholdValueFirst>0.000</ThresholdValueFirst>
    <ThresholdValueSecond>3.000</ThresholdValueSecond>
    <RuleType>ComplexLoadRule</RuleType>
  </ComplexLoadRule>
...
  <SimpleRule>
    <Measurement>total_throughput</Measurement>
    <AggFunction>Total</AggFunction>
    <CalcDirection>greater_than_equal</CalcDirection>
    <Value>100000.000</Value>
    <RuleType>SimpleRule</RuleType>
  </SimpleRule>
...
</Rules>
```

Figure 8: This figure shows an example on SLARules.xml.

The thresholds for the SLA rules are modified by LoadSplunker see figure 9. This modification was done for easier handling of the SLA rules in Splunk and to get rid of the redundancy of repeating the middle value twice.

<p><u>OLD:</u></p> <pre><ThresholdsCollection> <Threshold> <StartLoadValue>NegativeInfinity</StartLoadValue> <EndLoadValue>400</EndLoadValue> <ThresholdValue>3.000</ThresholdValue> </Threshold> <Threshold> <StartLoadValue>400</StartLoadValue> <EndLoadValue>PositiveInfinity</EndLoadValue> <ThresholdValue>5.000</ThresholdValue> </Threshold> </ThresholdsCollection></pre>
<p><u>NEW:</u></p> <pre><StartLoadValue>NegativeInfinity</StartLoadValue> <MiddleValue>1</MiddleValue> <EndLoadValue>PositiveInfinity</EndLoadValue> <ThresholdValueFirst>0.000</ThresholdValueFirst> <ThresholdValueSecond>3.000</ThresholdValueSecond></pre>

Figure 9: This figure compares the threshold structures that are modified in LoadSplunker.

4.4 Splunk settings

Splunk is easily configurable through editing configuration files. Integrating LoadSplunker and Splunk was done by using a Splunk forwarder. The Splunk forwarder was used to parse, index and then forward data to the targeted Splunk server. This instance used to forward is called Splunk forwarder and the other one is called Splunk server. The configuration files that are configured exists in %Splunk%/etc/system/local/. The forwarding will be done because LoadSplunker will not be able to reside in the same server as the Splunk server used by IKEA.

4.4.1 Splunk forwarder

The settings in figure 10 specify where and how Splunk should import data. Batch in figure 8 is used as monitor input with the difference that the files are deleted after they are indexed in Splunk. Move policy is the exact distinction between monitor and batch input, this setting tells the batch input to destroy

files after they are indexed. Deleting the files after indexing is done so that the memory used is cleared and can be reused by the next test result. Before Splunk starts indexing files it will by default check the beginning of the file, this is to check if the file has been indexed before or not. Therefore `crcSalt` is set to `<SOURCE>` which means that the file path will be added as a salt to the file. This allows Splunk to separate similar files if they are in different folders or have different names. `Path/xml` is the path to the XML folder which is set in `connection.props`, see figure 25.

```
[batch://path/xml]
move_policy      =
sinkhole
disabled = false
crcSalt          =
<SOURCE>
queue           =
parsingQueue
```

Figure 10: This figure shows settings in `inputs.conf`.

The settings in figure 11 specify how Splunk will handle incoming data and will be applied to files that have a source path that matches “`...(\\\)xml(\\\)*(\\\)run_*(\\\)*.xml`”. The incoming data will be parsed and a timestamp will be added to each new event. This expression is used since it will match all files that originate from the folder used to store all the XML data.

```
[source::...(\\\)xml(\\\)*(\\\)run_*(\\\)*.xml]
LINE_BREAKER      =      (\<|?.+?\>) /
(\<V?(table/row).*\>) /
(\<V?(ComplexLoadRule/SimpleRule/Rules).*\>)
TIME_PREFIX = \<AbsoluteTime\>
TIME_FORMAT = %s.%3Q
```

Figure 11: This figure shows settings in `props.conf`.

...	matches any number of characters
*	matches anything except directory separator
	used as OR
parentheses	used to limit the scope of OR

Figure 12: This figure explains the syntax used to match source path.

The syntax of the settings in figure 11 and 15 is explained in figure 12. The settings in figure 11 tell Splunk how the data will be parsed before it is sent to the Splunk server. Line breaker will tell Splunk how to divide a file into events and this is set by defining a regular expression. Each time the regular expression finds a match, a new event is started and the old one is ended. The match will not be included in any of the events. The line breaker is configured to divide files into events. One event will correspond to one row in the database table in a test result. Time prefix tells Splunk where in each event it will find its time attribute. Time format only tells Splunk which format that time is in. Adding this configuration for time is important since events in test results are worthless if it is unknown when they happened. In figure 13 settings for how the local Splunk instance will forward its data is shown.

```
[tcpout]
defaultGroup = default-autolb-
group

[tcpout:default-autolb-group]
server = ip-address:port

[tcpout-server://ip-address:port]
disabled = false
```

Figure 13: This figure shows settings in outputs.conf.

4.4.2 Splunk server

The settings in figure 14 specifies from where the Splunk server will import data.

```
[splunktcp://9997]
disabled = false
queue =
parsingQueue
```

Figure 14: This figure shows settings in inputs.conf.

In figure 15, KV mode is used to tell Splunk that the events are in XML format. This is set to XML since all data from test results are saved in this format. When searching events in Splunk, attributes in the event can be searched. So if `<Value>1</Value>` exists inside an event then Value can be used to refer to the value 1. A field for the test run is added to each event which is done by extracting the information from its path. This run field is important for comparing different runs and therefore the extraction is done.

```
[source::...(\\|/)*xml(\\|/)*run_(\\|/)*.xml]
KV_MODE = xml
EXTRACT-run = ([\\|/]run_)(?<run>\\d+) in
source
```

Figure 15: This figure shows settings in props.conf.

4.5 Searches

When Splunk parses events it adds some fields to the event, one of those fields is source, which is from where the event originates. In the LoadSplunker system, XML files are imported from the directory hierarchy, “`../xml/<project_name>/run_<run_id>/<table name>.xml`”. This information is utilized in all searches to find data for specific projects and runs. When searching in Splunk, a base search is used to find specific data. A typical base search looks like the following:

```
source="*\\xml\\<project_name>\\run_<run_id>\\<table1_name>.xml" /
join type=inner <join-field> [search
source="*\\xml\\<project_name>\\run_<run_id>\\<table2_name>.xml"].
```

There are different types of tables used in the data from the test results, two important types of tables are meters and maps. Meters are tables containing raw data points with IDs and maps are tables linking the IDs to names. Table1 will usually be a meter table and table2 a map table. The reason the meter table is first is that Splunk will use the timestamps from the events in the first table and not the second when performing a join. When the specific data has been found it can be manipulated and viewed in many different ways. An example on a search generating a chart:

```
<base_search> | chart values(Value) over <over_field> by <by_field>
```

This search will generate a chart for viewing all values with the x axis as <over_field>, typically this is a time field, and this can also be done by using timechart which is a chart with values over time. Grouping the data will be done by specifying <by_field> which for example can be on runs, see figure 16.

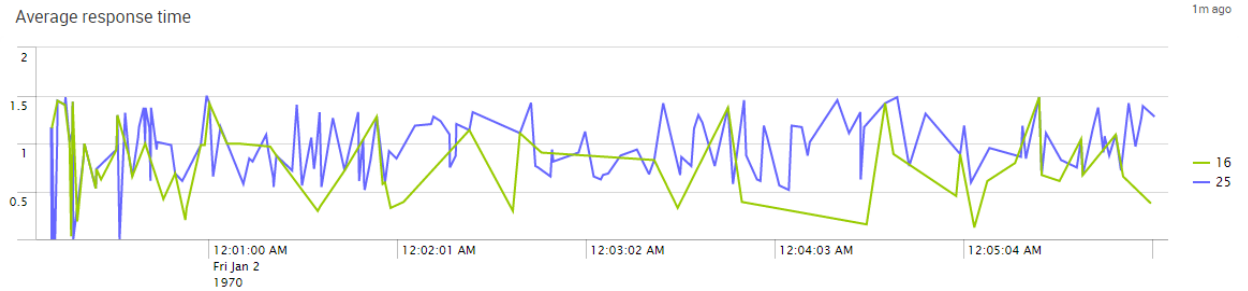


Figure 16: This figure shows a chart for average response time which is grouped by runs in Splunk with the default legend provided by Splunk shown to the right.

4.6 Dashboards

In this project three different solutions on how to divide projects and runs when displaying test results were investigated. In the paragraph below, these alternatives will be described and analyzed. How these solutions should be used for the best result was not found. The best option may be to use one or even a combination of them but this can only be answered with a more detailed survey. However, prototypes of the different solutions have been developed and tested. Input fields, visualizations, tokens, JavaScript and CSS were used to implement these prototypes.

4.6.1 Alternative one

Every dashboard has an XML source which describes the view of a dashboard. Changing the XML source will change the layout of the dashboard. The first solution is based on being able to create a dashboard for each run in a project from a template. The template should be retrieved and used by LoadSplunker every time a run finishes in Performance Center. The XML template that is used has marked those variables that will vary with each run with curly brackets. Figure 17 shows an example of how the XML would look. The LoadSplunker system would then replace all these marked variables to the correct value and store the result in a string which would be used to create the dashboard.

```
<dashboard>
  <label>{label}</label>
  <panel>
    <chart>
      <title>Summary Transaction</title>
      ...
      <query>source="*\XML\{run}\Event_meter.xml"</query>
      ...
    </chart>
  </panel>
</dashboard>
```

Figure 17: This figure shows an example of an XML template.

The Splunk REST API was used to create the dashboards using this string. To do this, a POST to /servicesNS/admin/search/data/ui/views/ was used with XML source and dashboard title as parameters. The title of the dashboard would be the project name followed by the run id. This was to name every dashboard uniquely. The benefit of this solution is that all information needed from a specific test run, can be viewed in one dashboard. A downside to this solution is that it would result in a large amount of dashboards. It would also be very performance demanding because a large amount of data would be handled by only one dashboard. To be able to compare runs within a project the administrator of the dashboard would be forced to add every new test run manually to the XML source. This would result in a large amount of manual work and is not a dynamic solution.

For each new test run, LoadSplunker will create a new dashboard in the following way:

1. Retrieve a dashboard XML source template.
2. Set the values that vary between each run:
 - Run id
 - Label
3. Create the dashboard using the modified XML source.

4.6.2 Alternative two

The second alternative would be to create one dashboard for each project. With this alternative, a comparison of runs will be available in each dashboard. It would also result in fewer dashboards if compared to alternative one. The downside of this alternative is the difficulty to correlate description fields dynamically to runs and comparison of runs. No solution to make this correlation was found during this thesis work. It would however be possible to add a text field that describes the project itself. This because the user would not be able to change which project is viewed in the dashboard.

For each new project, LoadSplunker will create a new dashboard in the following way:

1. Retrieve a dashboard XML source template.
2. Set the values that vary between each project:
 - Run id
 - Project
 - Label
3. Create the dashboard using the modified XML source.

4.6.3 Alternative three

The third solution is based on only creating one dashboard for each graph. The user can chose what project and which runs the dashboard will display data from. The advantage of this is that Splunk will not overflow with dashboards and no dashboards would have to be created or modified by LoadSplunker. Instead one dashboard would be created manually for each graph and when new runs and projects appear in Splunk the user will automatically be able to select them in an input field. It's also the easiest way for dashboard management because a dashboard only needs to be created one time for it to represent all runs and projects available.

To implement the third alternative, input fields were added so project and run could be specified. Input fields that were added to the dashboards were test run, project and time span. Tokens were used to save and use these values. Test run and project was added so that the dashboards could be used for all the projects and test runs. Time span was added so that the viewer can set granularity in the chart. This is important since duration and number of transactions differs between test runs, see figure 18.

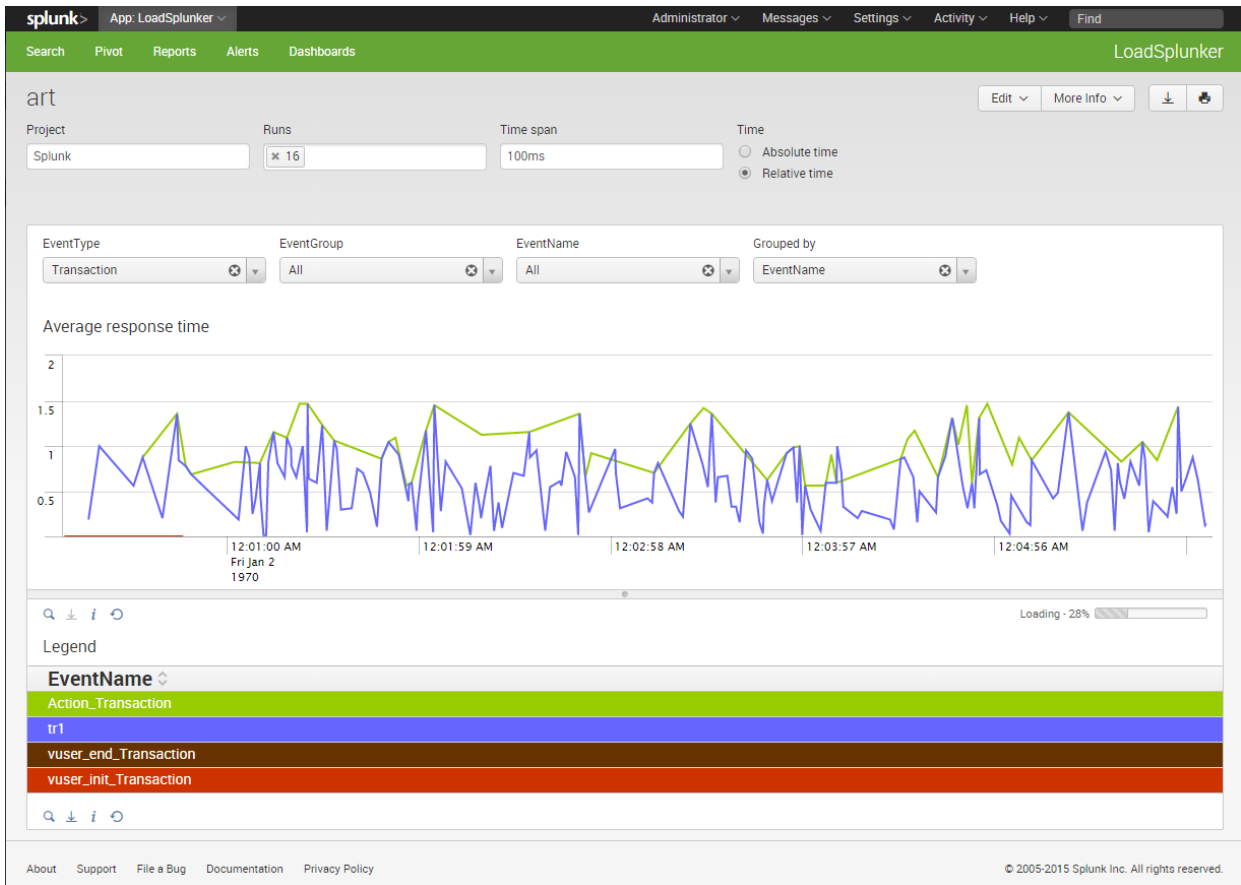


Figure 18: This figure shows an example of a dashboard that was developed during the thesis work.

IKEA wanted a similar legend in Splunk as in LoadRunner Analysis, see figure 19.

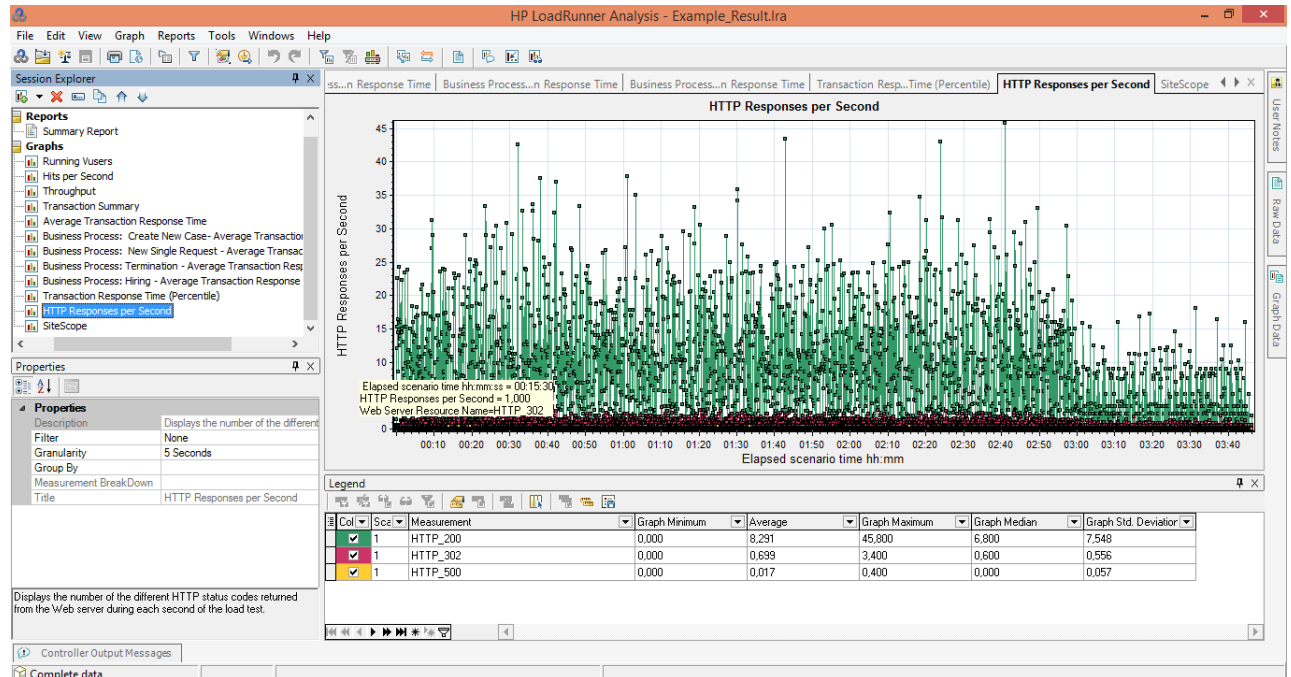


Figure 19: This figure shows a graph with its legend in LoadRunner Analysis.

In Splunk every graph should have its own legend as viewed in figure 20.

Color in/link to graph	Business process name	Transaction name	Average response time	95th response time	Fail/Pass
Link/color will relate to the graphical presentation.	Taken from the "Transaction name".	Taken directly from the test result.		The percentile should be dynamic. Default 95%	Compare with the SLA rule of response time.

Figure 20: This figure shows Splunk's wanted fields for the legends in the dashboard.

During this thesis work an example test result was provided to work with. This test result did not have any SLA rules or transaction names in the format needed to extract the business process name. Test results with this included were planned to be delivered by an IKEA employee but the test environment used to create this test result did not work as planned. With this said, the legend was not created as shown above, the only thing included was the coloring and the transaction name.

To correlate a transaction name in the legend and a line in the chart, coloring was used. This coloring was done by using CSS. Since the coloring was done in order, the coloring will be wrong if the sorting inside the legend is changed. Coloring was very important and implementing it in this flawed way was the only solution found during the thesis work.

4.6.4 Export PDF

Every dashboard can be exported into a PDF format but the design done by CSS in the dashboard will not be shown. Compare figure 21 and 22.

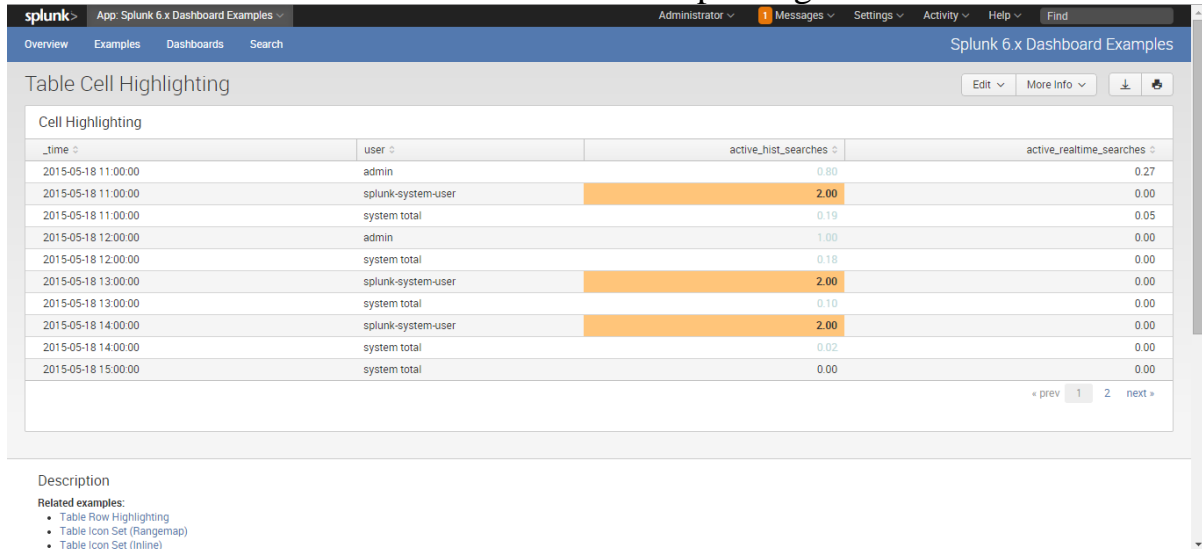


Figure 21: This figure shows a dashboard where cells are colored with CSS.

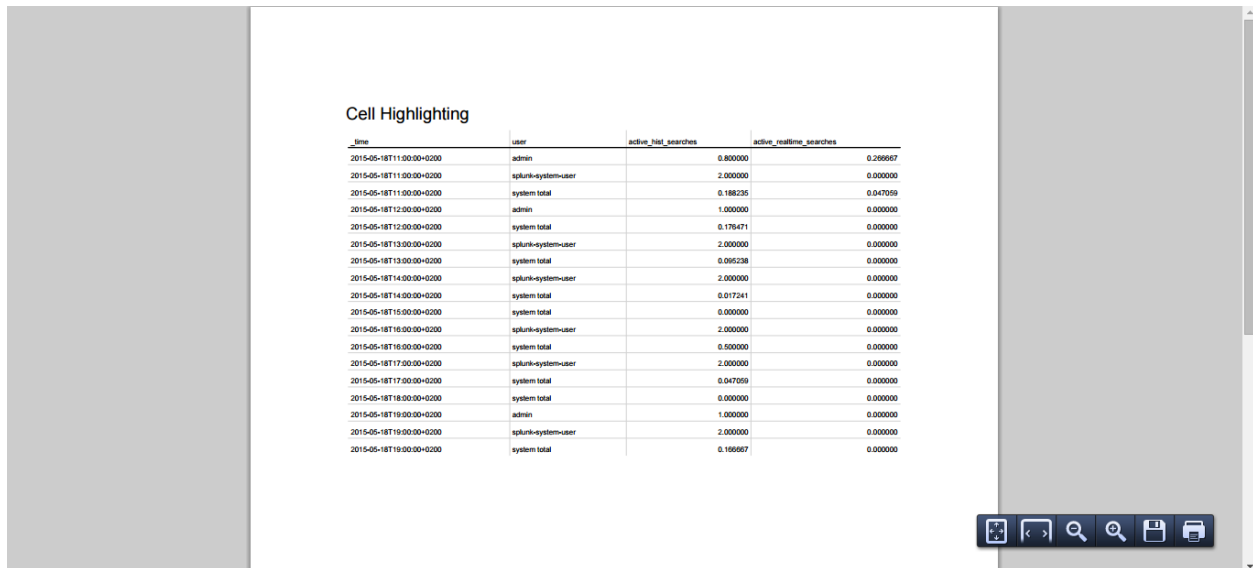


Figure 22: This figure shows figure 18 as exported PDF.

4.7 User management

There was a desire to grant all IKEA stakeholders read access to static dashboards in Splunk. A solution relying on using IKEAs Active Directory database was considered to automatically provide the correct access right to information in Splunk. The solution was based on using a LDAP filter to grant access to specific users in the database. This did not work since the access controls in Splunk are implemented by using roles and a role could only be provided to groups inside of the database. Without the possibility to provide default roles to users, read access had to be provided manually. This could be done either by adding users into Splunk or mapping users to groups in the database. None of these manual solutions were considered viable for IKEA since the number of employees and other stakeholders at IKEA is too large.

The testers for projects at IKEA exist as users in Performance Center. Using the Site Admin REST API information about an ALM project's users and roles can be retrieved and with the Splunk REST API it is possible to create users in Splunk. By using these two APIs, LoadSplunker can create a new user in Splunk for every user in Performance Center. With LoadSplunker informed on users and roles in Performance Center, users can be given the same roles in Splunk as in Performance Center. Some users in Performance Center have a specified email. When these users are created in Splunk a randomly generated password will be sent to them using IKEAs mail service. For users not having an email registered, a default password is provided. This password is provided within the installation manual of LoadSplunker. See figure 23 for LoadSplunker's user management flow chart. This implementation depends on how dashboards will be utilized later on which probably will result in desired changes of LoadSplunker in the future.

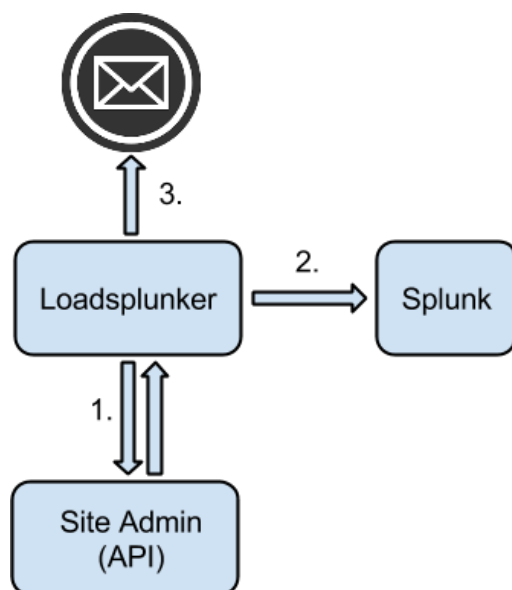


Figure 23: This figure shows how LoadSplunker manages users.

4.7.1 Real time system

LoadSplunker is a system consisting of three periodic threads, see figure 24. With periodic threads, each thread will be idle a given amount of time until it executes again. In this way LoadSplunker will not take computing resources unnecessarily. The different threads are named ScheduleThread, SplunkThread and ResultThread. ScheduleThread will get information on test runs such as when they are finished. This information will be sent to ResultThread which will download and handle the analyzed result from the test run. ResultThread will in turn inform SplunkThread of the new test run. SplunkThread will create a role and several users in Splunk. One role will be created for each project in Performance Center. All the users in a project will be created in Splunk with the information stored in Performance Center.

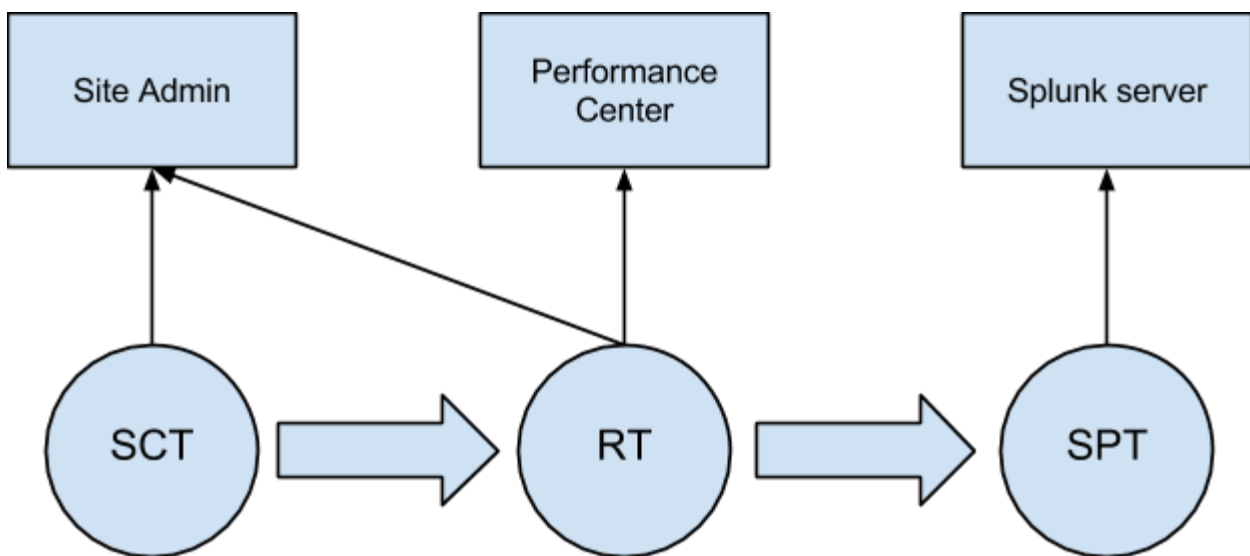


Figure 24: This figure shows the sequence of the threads and which monitors they communicate with, the large arrows representing a message being sent from one thread to another.

4.7.2 ScheduleThread

ScheduleThread will get a priority queue with information on scheduled test runs in Performance Center which has not already been handled by LoadSplunker. Separation on which tests that has been handled is done by using time as a reference point. The result is stored in a file so that LoadSplunker can be restarted without losing this value. The priority queue with scheduled tests is sent to ResultThread. ScheduleThread also checks if there are any empty folders in the XML folder because the Splunk batch only deletes the indexed files, not folders. Some simplified and clean code for this thread is provided in figure 25.

```

DateTime lastTime = updateTime();
PriorityQueue<Run> queue =
sa.getDownloadQueue(lastTime);
if(queue != null && !queue.isEmpty()){
    RunEvent event = new RunEvent(this, queue);
    rt.putEvent(event);
}
cleanFolders();

```

Figure 25: This figure shows simplified and cleaned code on how ScheduleThread works.

4.7.3 ResultThread

ResultThread will check if it has received a new message from ScheduleThread and concatenate the sent priority queue with its own internal priority queue. ResultThread then checks if the first run in the priority queue has passed its finish time. When a run has passed its finish time, a message is sent to SplunkThread. After this ResultThread downloads, extracts and translates the test result to XML files. This is only done if the result is downloadable. The result is downloadable if the test run has finished and the test result has not been deleted. Some simplified and clean code for this thread is provided in figure 26.

```

RunEvent msg = (RunEvent) mailbox.tryFetch();
if (msg != null && msg.queue != null) {
    queue.addAll(msg.queue);
}
DateTime now = new
DateTime(System.currentTimeMillis());
Run first = queue.peek();
if (first != null && first.compareTo(now) < 0) {
    Run run = queue.poll();
    if (sa.isDownloadable(run)) {
        sendSplunkEvent(run);
        downloadResult(run);
        Unzip.unzip(projectPath + ".zip");
        translateResult(run);
        delete(projectPath + ".zip");
        delete(projectPath);
    }
}
}

```

Figure 26: This figure shows simplified and cleaned code on how ResultThread works.

4.7.4 SplunkThread

SplunkThread will check if it has received a new message from ResultThread. Messages from ResultThread contain the following information: domain, project, run and users with their roles from the project in Performance Center. If a new message has been received, SplunkThread will create a new role if the project is being handled by LoadSplunker for the first time. All the users in the project will be added in Splunk with the role created. Some simplified and clean code for this thread is provided in figure 27.

```
SplunkEvent msg = (SplunkEvent)
mailbox.tryFetch();
if (msg != null && msg.users != null) {
    String role = "LoadSplunker-"+msg.project;
    splunk.createRole(role);
    for(User user: msg.users){
        if(!splunk.userExists(user.username, true)){
            String pass = "changeme";
            if(user.hasMail()){
                pass = passwordGeneration(8);
                sendMail(user.mail, pass);
            }
            splunk.createUser(user, pass, role, true);
        }
    }
}
```

Figure 27: This figure shows simplified and cleaned code on how SplunkThread works.

4.7.5 Logging

For easier management of the LoadSplunker system logging has been implemented with log4j. Log4j uses different logging levels, these are: debug, error, info, warn and trace. The levels are used in different situations. For example, trace is used as a finer grained tracing of a bug than debug. This logging was implemented for easier error management. [16]

4.7.6 Settings

LoadSplunker uses property files which are placed in %LoadSplunker%/props/, see figure 28 and 29. Figure 28 contains connection information for Site Admin, Performance Center and Splunk. This file also contains the paths to the XML folder, the results folder and the dashboard template. The property file format was chosen because of its simple management with the standard Java library `java.util.Properties`.

```
#Performance Center
pc.name=username
pc.pass=password
pc.host=ip-address/LoadTest/rest

#Site Admin
sa.name=username
sa.pass=password
sa.host=ip-address/qcbn

#Splunk
splunk.name=username
splunk.pass=password
splunk.host=ip-address

#Paths
# choose directory separator depending on operating
system
# / or \ for Windows
# / for Linux
# : for Mac
path.results=path/results
path.xml=path/xml
```

Figure 28: This figure shows settings in `connection.properties`.

```
# Path to the log file
log = path/log

# Define the root logger with appender file
log4j.rootLogger = DEBUG, APPEND

# Define the file appender
log4j.appender.APPEND=org.apache.log4j.FileAppender
log4j.appender.APPEND.File=${log}/log.out

# Define the layout for file appender
log4j.appender.APPEND.layout=org.apache.log4j.SimpleLayout
```

Figure 29: This figure shows LoadSplunker's settings in log4j.properties.

5 Conclusion

This section contains conclusions of the result, answers of the questions from the problem specification and some ideas for future development.

Today more and more processes are being automated with technology. This should not be any different for testers since they are extremely important in all kind of development. If the analysis process becomes more automated for testers, more time can be spent on actual testing and this should result in a higher quality of testing and therefore better deliverables. Default reporting is a viable way for testers to save time. This is unfortunately very difficult since testing can differ in so many ways. At IKEA a project can last for years, so building default reporting for one specific project might be a viable option. This would also improve the knowledge on implementing default reporting which may be an investment with great technical development outgrowth.

The assignment of this thesis work was also to develop software for IKEA that could replace their current test analysis tool with Splunk. To schedule performance test IKEA is using HP's software called Performance Center. In this thesis work, integration between HP Performance Center and Splunk was developed and named LoadSplunker. The LoadSplunker system downloads new test results from Performance Center and creates XML files of the result data. LoadSplunker then automatically uploads the XML data into Splunk where it's indexed and presented in dashboards. LoadSplunker also retrieves user information from Performance Center and recreates them in Splunk.

The purpose of the assignment was also to investigate how to automatically create default reports in Splunk. For automatic reporting in Splunk some different possibilities have been explored in this thesis work. The first one was creating a new dashboard every time a test run is finished. In this way, static description fields can be added by the tester to visualizations which is an important part of test reports. One flaw with this solution is the amount of dashboards considering the amount of projects at IKEA. Another flaw with this solution is the performance demand with the amount of data that would be shown when several projects are displayed inside the same dashboard. The second possibility was to create one dashboard for each project and in this way also make comparison of test runs available in the same dashboard. This solution has the same flaws as the first solution and it also restricts the possibility to add description fields in the dashboards. A description should correlate to one specific graph and how this correlation could be done in a dynamic dashboard is unknown. In the third possibility only one dashboard would be created for each graph. This solution will not overflow Splunk with dashboards. This also restricts the amount of data inside one dashboard to the

amount of data in one single graph. With the third possibility, analysis can be done in Splunk with the dashboards but the reporting will not be automatic. One idea for easing the workload of the tester is to make use of alerts in Splunk which is mentioned in future possibilities. The dashboards built during this thesis work were implemented to show IKEA how dynamic dashboards can be built but no final choice was made on how the reporting in Splunk should be done. Three alternative dashboard solutions were examined but how to utilize these was not found. A more detailed investigation will be needed to answer this and with more knowledge on how this reporting should be done, more functionality may have to be added to LoadSplunker.

The result of this thesis work did not lead to any automatic reporting at IKEA. However, replacing LoadRunner Analysis with LoadSplunker and Splunk would result in some important advantages. Developing dashboards is almost only restricted by the imagination with the option of using HTML, JavaScript and CSS. Therefore dashboards can be built for a more flexible analysis than in LoadRunner Analysis. Test results can be compared over test runs and can also be correlated with other data residing in Splunk. This other data can be logs or anything else from any part of the environments used for testing.

In the beginning of this thesis work, problems to be solved were specified, see 1.3. The following paragraphs contain the conclusions of these problems.

Is it possible to gather all data points from performance tests without any data loss?

In the analyzed results that are downloaded from Performance Center there is a database which contains tables with raw data. This data can be downloaded, translated and then imported into Splunk without any loss.

How can LoadSplunker get informed when a test is finished?

In the Site Admin database there is a table to keep track of when test runs are scheduled and will finish. This information is fetched by querying the database.

How can LoadSplunker access result files from tests in Performance Center?

Using the REST API provided for Performance Center all test results can be downloaded in form of a zip archive. The test result is then extracted from the downloaded zip archive.

What information is the most interesting for testers, developers and other stakeholders?

The supervisor of this thesis work specified some basic graphs which are created by Analysis. Unfortunately not much time was spent answering this since there are still some questions on how to best utilize Splunk for default reporting and dynamic analysis.

Is it possible to make use of a template to create a default view of the information?

Creating dashboards as default reports will result in an overflow of dashboards in Splunk. Instead dashboards can be created for an easier analysis of the test result than LoadRunner Analysis. Alerts may also be used to provide information about things out of the ordinary in the test.

How to gain access to information about users at IKEA?

In the database for Site Admin there are tables in each project containing information about the users and groups in that project. LoadSplunker queries this database on what groups and users exist.

How to create restriction in Splunk using that information?

User roles in Splunk can be used to provide access to apps, dashboards, alerts and more. This gives great possibilities on giving stakeholders access to very specific information. No final decision on how Splunk would be used for default reporting was made during this thesis work and therefore, no exact user management can be chosen.

How to keep Splunk updated with changes on IKEAs users?

An update of user information in a project can be done every time a new test run is handled by LoadSplunker. Another possibility is to add one more thread to the LoadSplunker system. This thread can periodically update all the user information in Splunk. This other possibility might be very demanding depending on the amount of projects and users that are handled but a very long period on the thread can be used to reduce this disadvantage.

How can this system be installed in IKEA's environment?

At the start of this thesis work, the goal was that LoadSplunker should be operating system independent. The Site Admin API is unfortunately COM-based and therefore LoadSplunker will only work in a Windows environment. This can be solved by running LoadSplunker either in a real or a virtual windows environment. Using a virtual windows environment will result in some performance loss.

Since the reporting in Splunk was not finished in this thesis work, any official installation at IKEA did not take place. Instead a general installation manual was written to ease the transition of LoadSplunker from the thesis workers to IKEA.

5.1 Future possibilities

This section includes some information that might be useful for future development of the LoadSplunker system and the reporting in Splunk.

5.1.1 Alerts

When a tester analyzes test results it is often things out of the ordinary that are interesting. Splunk enables user to specify saved searches and perform actions depending on the result of that search. Using this functionality, basic troubleshooting can be done which gives the tester time to focus on testing.

5.1.2 Chart overlay

Splunk has the possibility of adding a chart overlay, this overlay is done by using a field from the search. This field is often calculated by using stats or some similar function at the end of the search. Examples on chart overlays are shown in figure 30.



Figure 30: This figure shows lines running right above the bar charts, these are chart overlays.

5.1.3 Customized dashboards

Different people have different knowledge, with this in mind, not everyone wants to see test data shown in the same way. A tester would want to be able to view all the specific data in a test and a stakeholder on the business side would probably like to see more of a summary. With this in consideration, several customized dashboards viewing the same data could be created with each different dashboard being targeted to a specific group of stakeholders.

5.1.4 Dashboard legend

The dashboard legend was not created according to plan because of some unfortunate reasons as mentioned in 4.6, see figure 31. Only the coloring and the transaction name were included which is a very small part of the goal. The coloring was done by using an ordered color palette for the chart and coloring the legend table with the same colors in the same order. This means that the coloring was not dynamic as wanted, since the colors are wrong if the sorting inside the legend is changed. This flawed solution was used because no better solution was found within the time frame of this thesis work.

Adding business process name to legend will not be difficult since LoadSplunker adds a field named BusinessProcess for each event. This is extracted from the transaction name. If the transaction name not is in the format expected then the BusinessProcess is set as unknown. Average response time can be calculated by using the avg function available in Splunk. Calculating the 95th percentile column might be the most difficult in this legend. The 95th percentile is often calculated by sorting the data set and then finding the value which is greater than the other 95 percent. How to calculate the 95th percentile response time has not been solved during this thesis work.

Fail/Pass can be implemented by comparing the threshold in SLARules.xml with the response time. A symbol for failure or success can be showed in this column by using JavaScript and CSS.

Color in/link to graph	Business process name	Transaction name	Average response time	95th response time	Fail/Pass
Link/color will relate to the graphical presentation.	Taken from the "Transaction name"	Taken directly from the test result		The percentile should be dynamic. Default 95%	Compare with the SLA rule of response time.

Figure 31: This figure shows wanted fields for the legend in the dashboard.

5.1.5 Extended logging

Logging in LoadSplunker can be modified so that it is possible to import the logs into Splunk. If the logs are imported into Splunk, alerts can also be set up to give notification. This is used so that if something goes wrong with LoadSplunker, the logging makes it easier to troubleshoot.

5.1.6 SideView utils

SideView utils is an application in Splunk that can be used to ease the development process of dashboards with advanced XML. Advanced XML has become less popular amongst Splunk dashboards developers for its high complexity. With HTML, JavaScript and CSS great dashboards can be built in a less complex way. The disadvantage with this is that CSS will not be shown if the dashboard is saved as a PDF which is a very important functionality. The SideView utils application has not been used during this thesis work because it is not free to use but could be one of the best options for building dynamic and default reporting.

5.1.7 SLA rules

LoadSplunker translates the SLA rules that are included in analyzed test results and they are imported in Splunk but not yet utilized. SLA rules often consist of thresholds, showing this threshold can be done in charts by adding a chart overlay, see 5.1.2. SLA rules will be used for implementing the desired legend as mentioned in 5.1.4. The SLA rules could be altered in Splunk if input fields are added. Then if the tokens are set those values are used, if not then the values from the SLA rules set in Performance Center are used. This is desired since SLA rules might change along with the goal of a project which means that it should be easily modifiable.

5.1.8 System test

The LoadSplunker system has not been formally tested in a system test. This is something that needs to be done before LoadSplunker is put into production. This could be done by using Performance Center to test LoadSplunker and then use LoadSplunker to get the test results into Splunk so that the analysis of this system can be done in Splunk.

6 Terminology

Active Directory(AD)	A special-purpose database designed to handle a large number of changes and updates. [20]
App	Typically used to addresses several use cases and can contain one or more views in Splunk. [1]
Batch	A type of monitor that deletes the files after they are indexed. [32]
cURL	A command line tool and library for transferring data with URL syntax. [21]
Dashboard	A view in Splunk which can show a collection of visualizations. [7]
Domain	A group of projects in Performance Center.
Git	A distributed version control system designed to handle everything from small to very large projects. [6]
Lightweight Directory Access Protocol(LDAP)	Provides a mechanism used to connect to, search, and modify existing internet directories. This is used in this project to connect to an Active Directory. [17]
Load generator	The hardware used during testing.
Monitor	Can point to directories. All the data in these directories are imported and indexed in Splunk. [32]
Project	A project in Performance Center with a group of users.
Project id	Each project in Performance Center has a unique Project id.
Prototype	The code that was produced in the course EDT655 (project year three).
Regular Expression	A sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. 'find and replace'-like operations. [3]

Run id	Each run in Performance Center has a unique run id.
Search	A query to modify and display data in Splunk. [23]
Service Level Agreement(SLA)	A contract between a service provider and the customer.
Test result	Result in Performance Center generated in different stages of testing.
Test run	A project is often tested several times, each time a new test run is created.
This system	The LoadSplunker system.
Token	A way to store values from input fields inside of dashboards in Splunk. [8]
View	Can show one or more visualizations in Splunk. [18]
Visualization	A visual object such as a chart or a table in Splunk. [25]
WebEx	Used for web and video conferencing online. [10]

7 Sources

7.1 Trusted sources

[1] <http://docs.splunk.com/Splexicon:App>

Electronic resource, available on the internet 2015-05-20

[2] HP ALM help documentations (see figure 32 in appendix)

Electronic resource, available in Performance Center 2015-05-20

[5] <http://dev.splunk.com/restapi>

Electronic resource, available on the internet 2015-05-20

[6] <http://git-scm.com/>

Electronic resource, available on the internet 2015-05-20

[7] <http://docs.splunk.com/Splexicon:Dashboard>

Electronic resource, available on the internet 2015-05-20

[8] <http://docs.splunk.com/Documentation/Splunk/6.2.2/Viz/tokens>

Electronic resource, available on the internet 2015-05-20

[9] <http://docs.splunk.com/Splexicon:Tablechartdrilldown>

Electronic resource, available on the internet 2015-05-20

[10] <http://www.webex.com/>

Electronic resource, available on the internet 2015-05-20

[11] <http://mvnrepository.com/artifact/org.jvnet.com4j/com4j>

Electronic resource, available on the internet 2015-05-20

[12] <http://jackcess.sourceforge.net/>

Electronic resource, available on the internet 2015-05-20

[13] <http://mvnrepository.com/artifact/com.sun.jersey/jersey-bundle>

Electronic resource, available on the internet 2015-05-20

[14] <http://www.joda.org/joda-time/>

Electronic resource, available on the internet 2015-05-20

[15] <http://mvnrepository.com/artifact/log4j/log4j>

Electronic resource, available on the internet 2015-05-20

[16] <https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/Level.html>

Electronic resource, available on the internet 2015-05-20

[17] <https://msdn.microsoft.com/en-us/library/aa367008%28v=vs.85%29.aspx>

Electronic resource, available on the internet 2015-05-20

[18] <http://docs.splunk.com/Splexicon:View>

Electronic resource, available on the internet 2015-05-20

[19] <https://www.microsoft.com/com/default.aspx>

Electronic resource, available on the internet 2015-05-20

[20] <https://msdn.microsoft.com/en-us/library/bb742424.aspx>

Electronic resource, available on the internet 2015-05-20

[21] <http://curl.haxx.se/>

Electronic resource, available on the internet 2015-05-20

[22] <http://docs.splunk.com/Splexicon:Event>

Electronic resource, available on the internet 2015-05-20

[23] <http://docs.splunk.com/Splexicon:Search>

Electronic resource, available on the internet 2015-05-20

[24] <http://docs.splunk.com/Splexicon:View>

Electronic resource, available on the internet 2015-05-20

[25] <http://docs.splunk.com/Splexicon:Visualization>

Electronic resource, available on the internet 2015-05-20

[26] <http://www8.hp.com/us/en/software-solutions/application-lifecycle-management.html>

Electronic resource, available on the internet 2015-05-20

[29] <http://www8.hp.com/us/en/software-solutions/performance-center-testing/index.html>

Electronic resource, available on the internet 2015-05-20

[31] http://www.splunk.com/en_us/homepage.html

Electronic resource, available on the internet 2015-05-20

[32] <http://docs.splunk.com/Splexicon:Monitor>
Electronic resource, available on the internet 2015-05-20

[33] <http://cs.lth.se/english#0>
Electronic resource, available on the internet 2015-05-20

[34] <http://sideviewapps.com/apps/sideview-utils/>
Electronic resource, available on the internet 2015-05-20

[35] <http://mobaxterm.mobatek.net/>
Electronic resource, available on the internet 2015-06-04

[37] Data Structures, Abstraction and Design Using Java, Koffman & Wolfgang, second edition, Course literature, published 2010 by John Wiley & Sons Inc.

[38] <http://docs.splunk.com/Documentation/Splunk/6.2.3/Data/Editinputs.conf>
Electronic resource, available on the internet 2015-06-18

7.2 Untrusted sources

[3] <http://www.princeton.edu/~mlovett/reference/Regular-Expressions.pdf>
Electronic resource, available on the internet 2015-05-20

[4] <http://rest.elkstein.org/>
Electronic resource, available on the internet 2015-05-20

[27] <http://www.guru99.com/understanding-vugen-in-LoadRunner.html>
Electronic resource, available on the internet 2015-05-20

[28] <http://www.guru99.com/how-to-use-controller-in-LoadRunner.html>
Electronic resource, available on the internet 2015-05-20

[30] <http://www.guru99.com/how-to-use-analyzer-in-LoadRunner-12-0.html>
Electronic resource, available on the internet 2015-05-20

[36] <https://www.crisp.se/file-uploads/Kanban-vs-Scrum.pdf>
Electronic resource, available on the internet 2015-06-04

8 Appendix

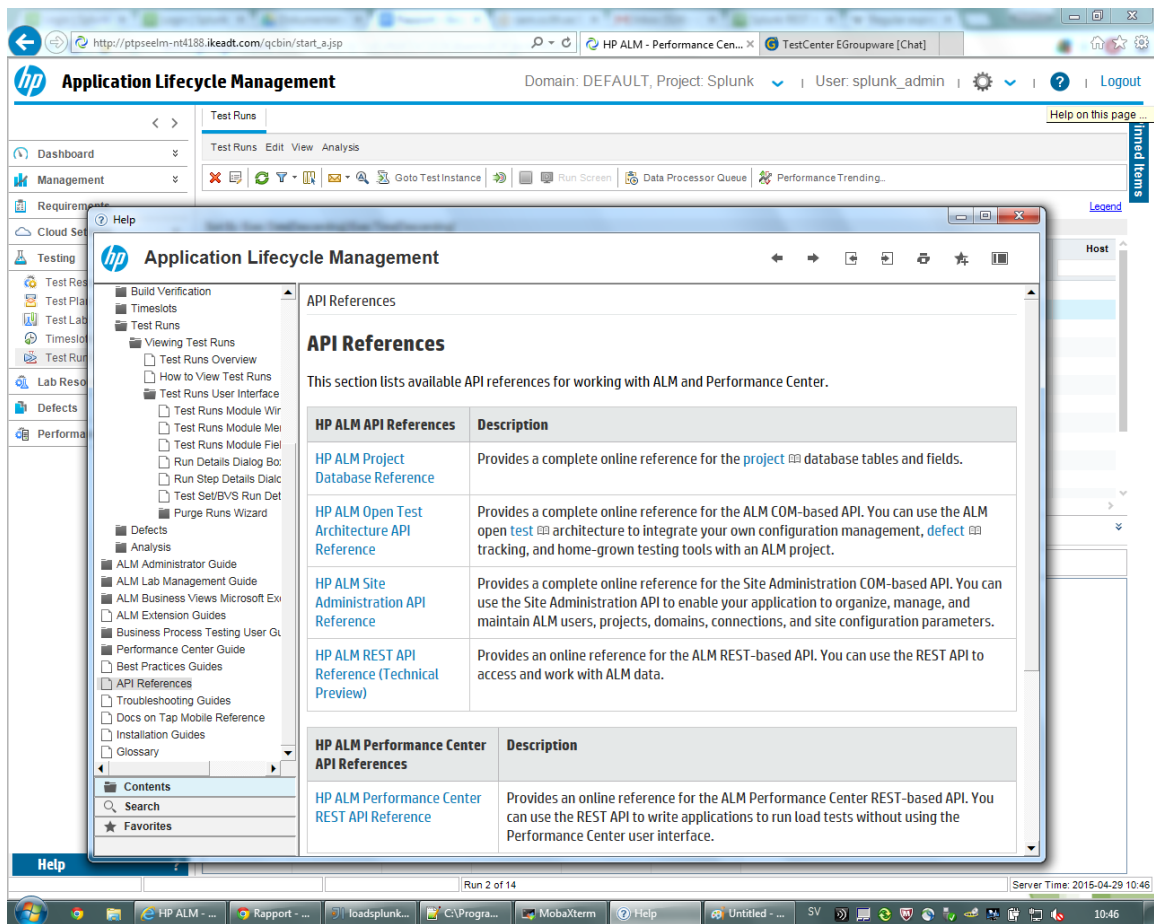


Figure 32: This figure shows ALM API reference.