WHITE PAPER

Development Approach for Customized Testing Tools - Selecting Testing Tools Efficiently

- **Vaibhav Suneja**

Abstract

At times, the testing process requires the use of customized tools along with standard automation tools. Customized testing tools are required mainly for the conversion of data from one format to another. This paper presents a brief comparison of open source, closed source (proprietary) and custom build approaches to building a testing tool. The comparison is demonstrated through a case study and through the challenges faced while selecting one alternative for developing the tool over another. The paper also highlights the importance of the tools team in an organization.

Infosys®

Introduction

When a testing project needs to be carried out across several organizations, typically an onsite-offshore model is used. However, in such a scenario, consistently communicating the client's needs to different teams working onsite and offshore can be a challenge. Moreover, testing teams often operate in crisis mode when tools need to be implemented to perform the testing. Usually, teams can choose from many open source tools that can be implemented. On the other hand, they may have access to several platforms that can be used to develop the tool from the ground up.

Before deciding the approach, testing teams need to answer a few key questions regarding the tool and where it needs to be implemented. The client's concerns must be factored in. These may include the client's inclination toward a certain technology, their past experience with some solutions and, more importantly, the ease of use and accessibility.

Open Source Solutions

Open source applications are free-to-distribute solutions which may be modified and implemented in a testing project. It is safe to assume a high probability of the client choosing an open source solution as opposed to a closed source one.

The most common problem faced while using an open source solution is the lack of adequate support. In case of commercial software, the vendor is responsible for timely assistance, especially when resolving security bugs. But if the testing team finds a critical bug in an open source application and needs assistance in fixing it, they may be required to pay an expert to fix it [1]. Support for open source is only in the form of forums and there may not be a proper helpdesk, or a support center. This underscores the need for a well-experienced technical team that can understand the code and implement the fixes, adding to the cost of using the open source solution.

Open source software concentrates on addressing the needs of developers and the end-users' demands are not always high on priority. Many open source projects do not focus on user interface, and do not provide adequate documentation. [2]

The other factor that can lead to the rejection of an open source solution is the company's (either the client company or the service provider) dislike for open source solutions. This can be due to their previous experience and the concealed terms and conditions, along with copyright terms associated with open source.

Open source may be a good choice for startups, but large organizations may find it difficult to gain the confidence of their stakeholders. Proving the advantages of open source over closed source may become a challenge.

```
<?php
$con=mysqli_connect("db","user","******","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$result = mysqli_query($con,"SELECT * FROM Tab");

while($row = mysqli_fetch_array($result)) {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br>";
}
```

Figure 1: In case of open source solution code is visible

In sum, while considering an open source solution, the requirement and funding should be cumulative. This means, the technical support required to maintain the solution must be added to the existing requirement of the client.

Closed Source Solutions

If the open source route presents problems, can closed source be looked at as a more effective solution? While this question can be answered in the affirmative, mainly due to the full technical support available, everything in the closed source approach comes with a cost.

This cost may be low till the time the team is working with a standard software tool. If a slight modification is required, the cost increases and this must be taken into consideration. For instance, if the tool offers to generate output in text file format and the team needs the output from the tool in a PDF format, this slight custom modification can send the total cost of the solution soaring.

The biggest disadvantage while implementing closed source tools could be the fact that the team deals with a set of binaries and cannot see or modify the code. To obtain even a slight modification, they need to approach the vendor and ensure that the requirement is fulfilled.

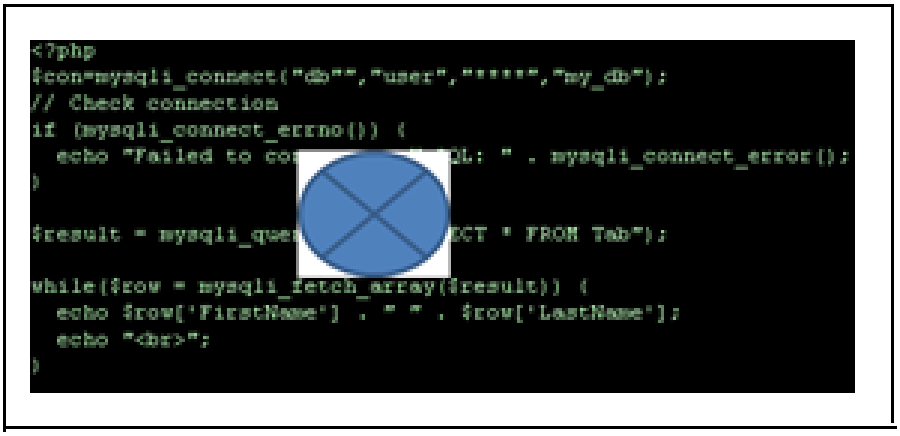


Figure 2:In case of closed source solution code is not visible

Building a Tool Ground-up or Reusing Existing Tools

Building a tool from the ground up can be a very effective option if the workforce possesses the required expertise. The following factors must be considered before proceeding thus:

The nature of the requirement

It is important to clarify that the reference here is not to large software implementations in the testing process such as QuickTest Professional (QTP), but to small-size tools such as parsers implemented in the process.

The process must start with a requirement analysis. Weighing the advantages and the disadvantages of the available open and closed source solutions is vital. Most importantly, the team needs to consider the value the solution can add to the project and its ability to increase client satisfaction.

Many a time, the client has had better experience with closed or open source solutions that can perform the same function the testing team proposes to achieve through a custom-build testing tool. In such a case, the team needs to explore the solution the client refers to, understand what it offers and determine if the proposed solution can function better.

Skills requirement

If the required tool is related to project test execution and the project team does not have the capability to develop the tool, the best option is to consult the tools group in the testing unit to deliver the solution.

Platform requirement

The platform can be a challenge for most startup organizations. For instance, if the testing team needs .NET to develop the tool then the license for the .Net framework needs to be bought. However, if the .Net framework is not required for any other application, then ordering the framework merely for one custom-build tool may not be a feasible option.

Skills requirement as opposed to platform requirement

There may be a discrepancy between the platform chosen for the development of the tool and the skill sets available within the tools team.

The tools team needs to be flexible enough to convert the code written in one platform to the other and vice-versa. It is preferable that some members of the tools team are trained in open source languages because these are the cheapest solutions that can be offered to the client.

Reusing the existing solution

Depending upon the requirements, it may be possible to reuse an existing tool from the tools repository. The existing solution or tool should be analyzed properly to explore the reusability. Experts must decide whether the reuse can be more productive than developing the tool from the ground up and encourage reuse wherever possible.

Application type	Requirement for development	Requirement for running at client organization
.NET Web Application	Visual Studio	.NET Framework
.NET Console/ Windows Application	Visual Studio	.NET Framework
Java	JDK/Editor	JRE
PHP	WAMP Server or any other server which can render php	WAMP Server or any other server which can render php

Table1: Sample requirements for developing different applications

Advantages of Developing over Reusing a Tool

- **Ability to build exactly what is required**

It is possible that a closed or open source tool offers the same functionality that the testing team is trying to achieve. If the functionality is insignificant for the majority of developers or users, then to present the package better, tool vendors might combine it with some features which are not part of the team's requirements.

This makes the tool more complicated to use. On the other hand, when the team is building the solution from the ground up, they can concentrate on their end-users who need to work on it and build the tool based on the users' skill sets and abilities.

- **Adequate support**

The tool team developing the tool knows it completely as they go through each line of code. This means adequate support is always available. The tool can be modified or extended as and when required.

- **Making the tool reusable**

The new tool can be developed by the tools team in such a way that it can be easily reused for other projects with slight modifications.

- **Development of project asset**

The tool developed by the tools team can become a project asset. The organization can refine it later and use it in other projects. The organization can also patent it and explore market opportunities for it.

Case Study

Development of Parser for Converting SWIFT Message into XML for SOA Testing on iTKO Lisa

Introduction to SWIFT message

The Society for Worldwide Interbank Financial Telecommunication (SWIFT) provides a network to allow financial and non-financial institutions (such as corporate establishments) to transfer financial transactions through a 'financial message'.[7]

Some examples of message standards supported by SWIFT are:

- SWIFT MT
- ISO 15022 MT
- ISO 20022 MX

Project requirements

iTKO LISA does not recognize a SWIFT MT message. It can only recognize XML. The requirement was to develop a solution that can parse SWIFT message to XML. XML needed be processed by LISA later post which it needed to be converted back to SWIFT and sent back to the application (which recognizes SWIFT).

To achieve this, two tools were required:

- SWIFT to XML Parser
- XML to SWIFT Parser

Introduction to iTKO LISA

iTKO LISA is a middleware automation tool for Service Oriented Architecture(SOA) testing. iTKO LISA enables testing of individual components, process, and workflows during design and development, integration, and in completed applications in deployment. Individual functional tests and system-wide business processes are load tested using the same environment and test suites, with performance reporting and error checking within each test instance. [1]

iTKO LISA solutions offer a unified solution for Testing, Validation and Virtualization. It provides three key capabilities to help firms mitigate risk and get better results from enterprise IT.

Open source solutions available

consultations with subject matter experts (SMEs), we found an open source solution termed WIFE.

According to WIFE's documentation, this community is an open source Java library for SWIFT messages parsing, writing and processing.

The main features of the solution are:

- Parsing of SWIFT MT messages into Swift Message Java objects
- Writing SWIFT MT messages from Swift Message Java objects
- De/Serialization of Swift Message objects into XML
- Hibernating the mappings for Swift Message objects
- Simplifying the persistence in applications

Problems faced using WIFE

Most features were not useful: The WIFE package comes with more than 100 classes, out of which we required only three or four classes. The rest of the classes were never required in our project.

Lack of knowledge or documentation: There was no documentation and no one in our team was aware of how the WIFE actually functions and how it can be integrated with iTKO LISA.

Approvals for open source: WIFE being an open source solution, the stakeholders perceived it as unreliable. This meant that taking approval from the client was a challenge.

Missing functionality: After thoroughly examining the requirement, we were not able to find out whether WIFE had the functionality for performing the required task or not. Going ahead with it was a risk.

What we did

After considering all the pros and cons, we decided to develop the parser from the ground up. We decided to build an application to identify the tags in SWIFT message and convert them into XML.

Initial requirements

- The application should be able to read the SWIFT message from .txt file.
- After reading the message, the application must be able to process the SWIFT message.
- There could be a different number of blocks in each SWIFT message. Some messages could consist of 4 blocks while others could have 5 or more blocks.
- The application should be able to convert an XML message back to SWIFT message.
- The application should integrate with iTKO LISA.

Requirements at later stage

- The application should be able to process multiple SWIFT messages in one .txt file, identify the start-and-stop sequence and mark the start and end.
- The multiple XMLs resulting from multiple SWIFT messages must be stored in different .XML files.
- The application should not be heavy.

The road we took

After understanding all the requirements we decided to code the parser. We had not finalized a specific technology for developing the parser. Therefore, we chose ASP.NET, the technology we already had with us.

Development of parser application using ASP.NET:

ASP.NET is a Microsoft proprietary technology which is used for developing active server pages or, in simpler terms, web pages.

We chose this technology for the development of the parser because we already had this application installed on our system.

Requirements:

Requirements	Software required
Developing ASP.NET application	Visual Studio
Running ASP.NET application	Windows IIS Server

We developed the ASP.NET application on Visual Studio 2008. We had IIS installed on our computers and we could easily see its working and integration with iTKO LISA. However, the problem occurred when we tried the same using the client's network.

Challenges we faced:

There was no IIS installed on the client's computer and therefore, we could not run the application.

Even if we installed IIS on a computer in the client's network, we needed to have the administration rights on the machine, which was a challenge.

Running an IIS server on a machine could create security issues in the client network. This meant finding an alternative.

The next steps

As we had Visual Studio installed on our machines, we decided to convert the application created in ASP.NET into a Windows console application.

Rewriting the code:

As ASP.NET is a .NET technology, converting its code into Windows console application was not a tedious task.

Requirements:

Requirements	Software required
Developing Windows console application	Visual Studio
Running Windows console application	.NET Framework

The Windows console application was the best and quickest way to eliminate the need for IIS on client machines. It also helped side-step administration related issues.

Challenges we faced:

We could not run the application because .NET framework was not installed on the client machine.

The final destination

To avoid further delays, additional effort and rework, we analyzed the client machine and checked the framework installed. Since all the modern-day operating systems (OS) come with Java Runtime Environment (JRE) package, we decided to create the parser in Java.

Place Swift message in _inputSWIFT\\readSWIFTmsg.txt	Converts all the swift message into xml and all the xml are stored in _tempfiles \\newfile.txt all xmls are seprated by ^ delimiter	Divides all the xmls in .txt file that were delimited by^, into separate xml files and store it in _outputxml\\Convertedxml1.xml
Place Swift message in _inputSWIFT\\readSWIFTNorkomMsg.txt	Converts all the swift message into xml and all the xml are stored in _tempfiles \\newfilenorkom.txt all xmls are seprated by ^ delimiter	Divides all the xmls in .txt file that were delimited by^, into separate xml files and store it in _outputxml\\Convertedxml1.xml
Place XML File in _inputSWIFT\\swift.xml	Converts the xml and store in _ tempfiles\\ReconvertedSWIFTmsg.txt	Processes the swift message and generate .txt and .doc file for proper formatting.

Figure 3: A representation of the information flow

Requirements:

Requirements	Software required
Developing the parser in Java	Java Development Kit (JDK) and editor
Running the application developed in Java	Java 2 Runtime Environment (JRE)

Basic architecture:

The parser developed in Java consists of only six classes which can be packed and run on any machine with JRE.

Integration with LISA:

We found out that the parser can be fully integrated with iTKO LISA as it comprises only .Class files that are to be invoked. These can be easily invoked through the batch schedule.

Currently we are working with the following:

- Conversion of SWIFT MT message into XML
- Conversion of Norkom SWIFT message into XML
- Conversion of XML back into SWIFT MT message

Advantages of building parser from the ground up

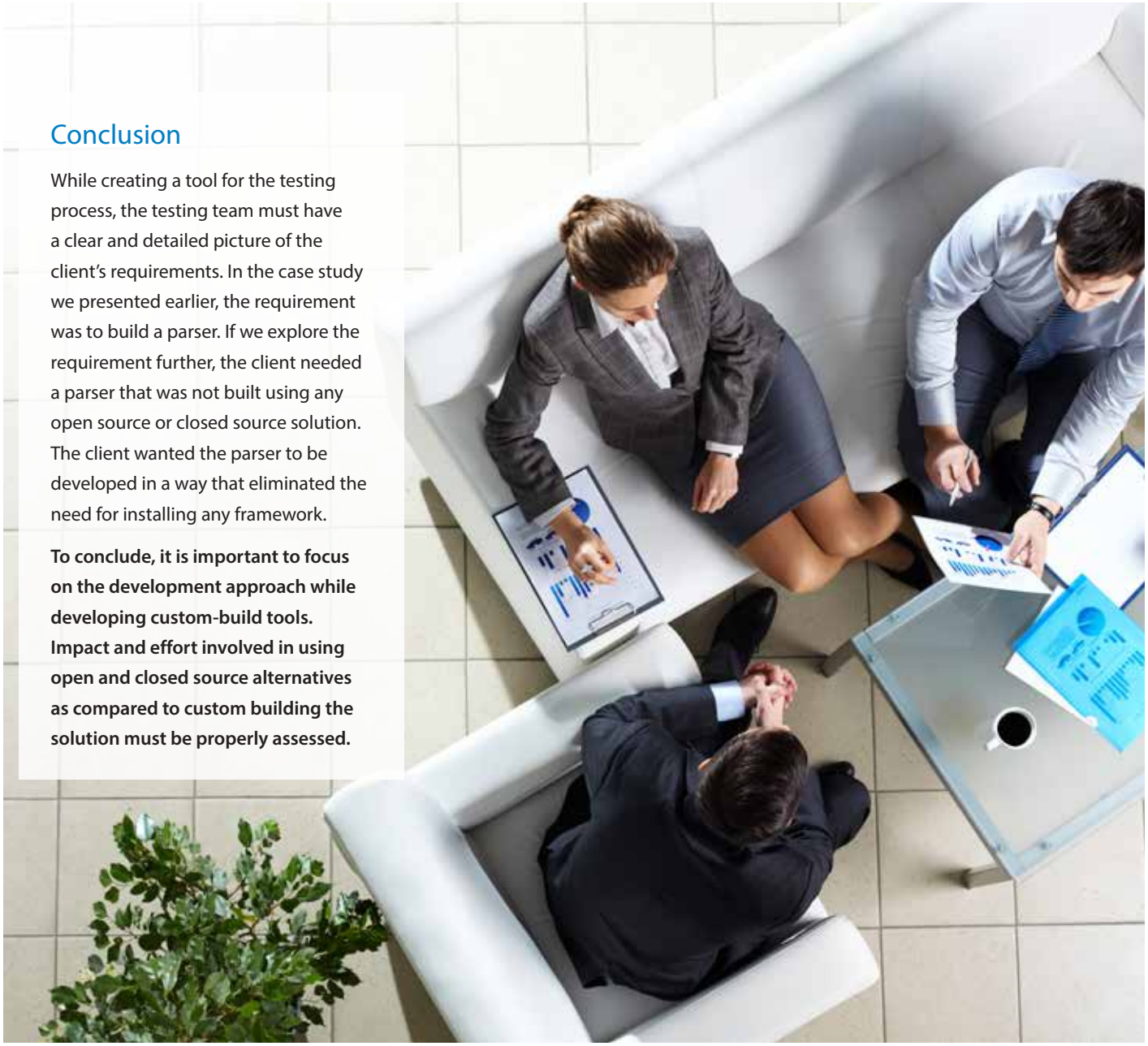
The parser we developed is light - there are only six classes for all the functioning required in our project.

- There are no integration issues.
- There are no licensing issues.

Conclusion

While creating a tool for the testing process, the testing team must have a clear and detailed picture of the client's requirements. In the case study we presented earlier, the requirement was to build a parser. If we explore the requirement further, the client needed a parser that was not built using any open source or closed source solution. The client wanted the parser to be developed in a way that eliminated the need for installing any framework.

To conclude, it is important to focus on the development approach while developing custom-build tools. Impact and effort involved in using open and closed source alternatives as compared to custom building the solution must be properly assessed.



About the author

Vaibhav Suneja

A Test Analyst with 4.5 years of work experience across various platforms and technologies. He has a diverse experience in Testing tools design, development and resuse

He can be reached at vaibhav_suneja@infosys.com

References

1. http://www.gnu.org/software/classpath/docs/hacking.html
2. Meffert, Klaus; Neil Rotstan (2007). "Brief summary of coding style and practice used in JGAP". Java Genetic Algorithms Package. http://jgap.sourceforge.net/doc/codestyle.html. Retrieved 2008-09-08.
3. http://www.tamingthebeast.net/articles5/open-source-software.htm
4. http://www.helium.com/items/514407-the-pros-and-cons-of-open-source-software
5. http://wapedia.mobi/en/SWIFT:Message_Types
6. "List of MT and MX Messages". SWIFT. http://www.swift.com/index.cfm?item_id=60538., pdf document from August 2008
7. http://www.technologyexecutivesclub.com/sponsorpages/itko.php
8. http://www.prowidesoftware.com/en/wife-documentation.html

For more information, contact askus@infosys.com

Stay Connected