Application Compatibility Framework - Building Software Synergy

Ashish Khandelwal (Ashish_Khandelwal@McAfee.com)
Shishira Rao(Shishira_Rao@McAfee.com)
Amrita Desai(Amrita_Desai@McAfee.com)

Abstract

Building a healthy inter-product alliance in a software ecosystem requires a great deal of effort. Our paper focuses on simplifying a product compatibility testing and helps a tester to find compatibility defects early in the testing life cycle. It demonstrates a "Framework driven approach" for improving Product Quality from compatibility standpoint which is proven, tested and sustained over releases of a product.

On the basis of underlying testing methodology, we have divided our compatibility approach in three testing types.

a) OS compatibility Testing
b) Third Party Compatibility Testing
c) Endpoint Compatibility Testing

As a standard practice, we follow compatibility model described in our paper and make sure that our product passes the product compatibility testing procedure. With these three testing types in place, not only we have identified risky areas early in the test cycle but also succeeded in influencing product management decisions based on our results and analysis. Here are few of the strategic points which a compatibility tester can leverage upon from this paper.

a) Align to a standard compatibility model
b) Systematic understanding of three compatibility testing types
c) Overview of sample results, case studies and analysis

Biography

Ashish Khandelwal *has more than 6.5 years of Software Testing experience. He holds a B.Tech degree from IIT Kanpur and works as a Senior QA Engineer with the McAfee Host DLP product solution group. He has contributed to multiple international conferences and published papers on Compatibility & Security Testing.*

Shishira Rao *has more than 7 years of Software Testing experience. She holds a B.E. degree from VTU and works as a Senior QA Engineer with the McAfee Host DLP product solution group. Her testing and QA experience includes a focus on Process improvement, Compatibility testing and Strategic planning.*

Amrita Desai *is a QA Engineer at McAfee and works with the Host DLP Product solution group with more than 3 years of experience. She has a Bachelor's Degree in Computer Science from RGPV University. Her testing and QA experience includes a focus on Black box testing, Compatibility testing and Soak testing.*

1. Introduction

How many times have you encountered that your customer has escalated issues in their environment after shipping your product? Have you made an analysis of the number of sustaining patches and hotfixes that have been released in order to maintain the synergy between different applications in the customer environment?

Compatibility testing is the process to determine the impact of conflict between multiple applications in a computing environment and to maintain information system functionality as intended. It is a part of software non-functional testing , which is conducted on the application to evaluate the application's compatibility.

So, how does compatibility testing relate to synergy?

In a technical context, synergy in an environment is established when different applications working together produce results, which cannot be obtained by an application alone. For instance, Open Office is an application suite that can be considered as a fair replacement of Microsoft Office in low budget projects. But do you think the alternative would have worked if Open Office does not support compatibility with Microsoft Office files? Definitely not!!

The ability of these two office suites to work together in harmony makes up this synergized environment.

This paper talks about developing a software ecosystem where applications not just work without conflicts but also work for each other. Here, we are trying to explore the use of model-driven testing techniques and tools to increase quality and make the testing process more systematic and efficient. This presentation will describe the introduction of such technology.

2. Application compatibility – Need of the hour

Historically, in case of functionality testing, corners get cut and often the last phase “Stabilization” suffers the impact of lack of compatibility testing. Customers are dissatisfied, teams are exhausted and quality is compromised in the drive to meet the promised date. To overcome these issues, compatibility testing plays a vital role in minimizing the consequences of lack of stabilization.

“Every Product is subject to Risk”. Let us consider a couple of real life scenarios where compatibility testing could have played a major role in identifying the risks much before it has turned into calamity.

XP Alternative to Vista PCs

While Microsoft is still pushing Vista hard, the company is quietly allowing PC makers to offer a "downgrade" option to buyers that get machines with the new operating system but want to switch to Windows XP. One of the biggest challenges, for both consumers and businesses are Vista’s incompatibility with other applications, intermittent crashes, memory needs and hefty graphics.

Notably the effects could have been alleviated by employing measures that would have detected most of the compatibility issues with other Microsoft applications and thus taken care of memory needs.

Website render wrong in some browsers

Even with all the recent strides in more standardized browsers, we've learned that you still can't let your guard down when it comes to testing cross browser compatibility

Few web based products maintain their independent spirit. Lack of consideration to cross browser compatibility testing brings a bad impression to the users and thereby hampers the growth of the company.

Who will do compatibility testing?

Certainly, there are no pre-requisites for a tester who wants to perform compatibility testing. The right amount of attitude with zeal to learn could be the starting point for a compatibility tester.

We would like to introduce you to our product environment. We are a team of 8 QA Engineers and 12 Developers distributed across multiple location. The skill-sets and experience levels vary across the team, ranging from Software Engineers working in Quality to QA Analysts with deep domain knowledge. As an initiative, we (a team of 3 functional testers) took up the challenge to find compatibility flaws in our product. The next section describes the foundation of ***A****pplication* ***C****ompatibility* ***F****ramework.*

Let us now start with defining and describing our Application Compatibility Framework.

3. Application Compatibility Framework

This framework highlights the way we perform product compatibility testing and how it fits into the product life cycle. It includes three types of compatibility testing model, OS, 3rd Party & Endpoint, and depicts the correlation between them.

The selection of type of compatibility model in a release cycle is based on various factors such as new OS version support, major or minor release of the product, affected modules of the product and endpoint release dates.

OS compatibility model comes into picture when product management announces the support of an Operating System or a service pack of a pre-supported Operating system is released. In few cases, when application's new modules are affected by OS changes, we perform a regression cycle of OS compatibility suite.

We ought to consider 3rd Party Compatibility testing in case of major releases of the product. However, in some cases with small releases we take prioritized list of applications to be tested.

Any host-based product managed by a centralized server is defined as an Endpoint. Endpoints release date is a key factor in defining plan for Endpoint Compatibility model. We identify the release dates of endpoints which are in the time frame of our product release cycle and strategize to perform endpoint compatibility testing.

Below figure shows the relation and overview of the compatibility framework.

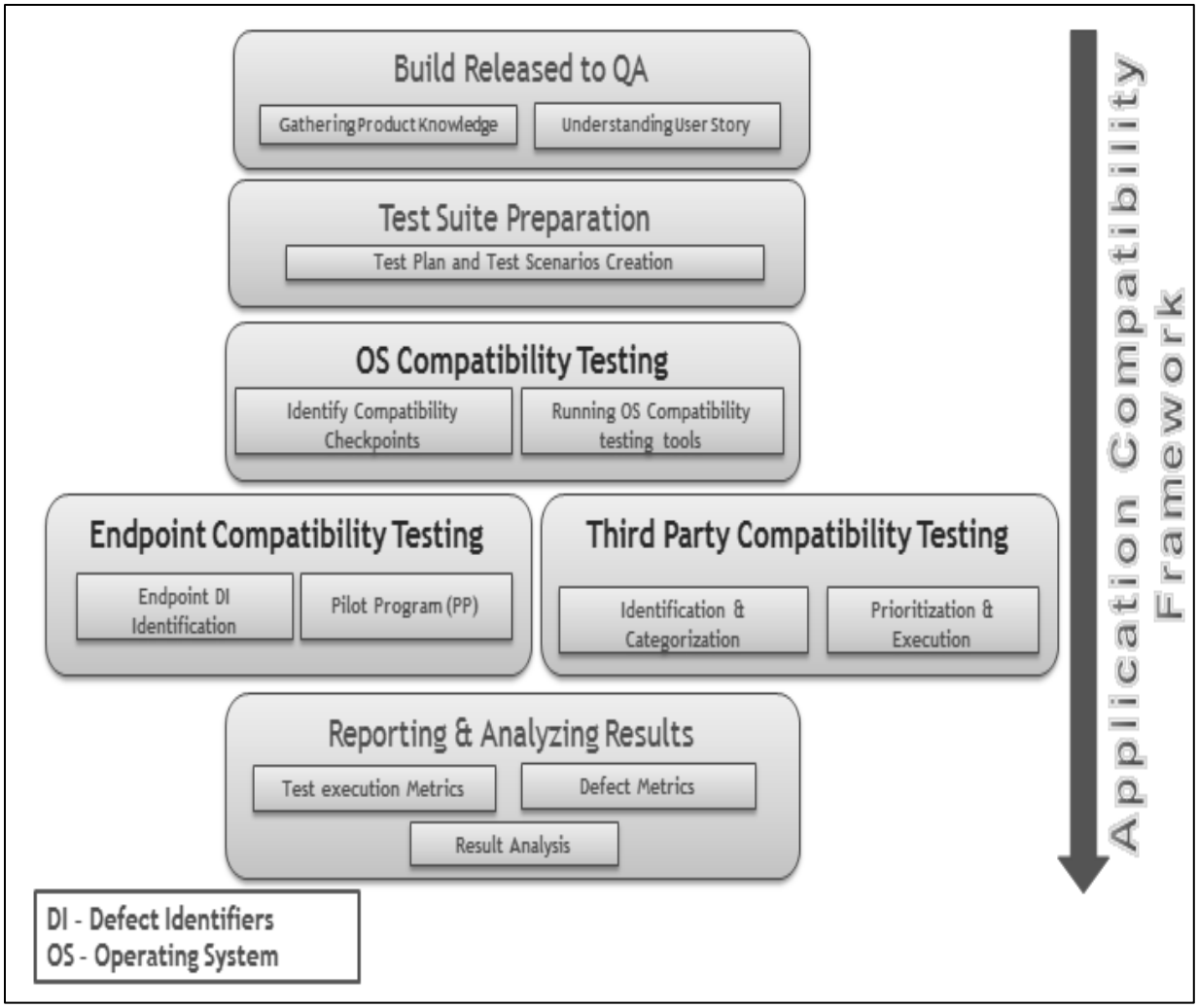

Figure 1 Application Compatibility Framework

In every major release, we plan to ensure that there is a decrease of at least 25% of defects found on customer site concerning compatibility and at least 10% growth in number of compatibility defects logged via compatibility framework. We also aim to build our 3rd Party application portfolio such that no defect occurs outside those applications.

Next three sections will talk in detail about the three compatibility-testing models and the individual role they play in defining the framework.

4. Operating System Compatibility Testing Model

Operating System (OS) design change can propagate numerous compatibility defects in a product. The purpose of this testing type is to uncover such issues that occur due to these enhancements.

For instance, from Windows XP to Windows Vista, Microsoft designers have performed major changes in the system architecture. Although these changes were incorporated to improve the performance and design, they also lead to multiple compatibility issues due to inability of other products to cope with the changing environments.

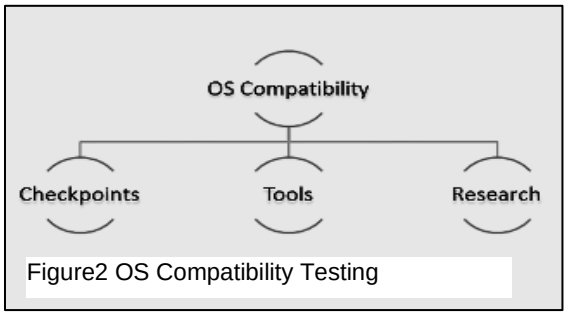

Figure2 OS Compatibility Testing

As a result, with frequent kernel upgrades and service packs release, it has become essential to strategize the compatibility testing approach that uncovers these issues early in a product cycle.

So, how should we perform it? What are the various parameters that will help uncover the issues related to changes in OS?

Figure2 highlights the various characteristics of OS Compatibility testing as explained below:

Checkpoints

These are the various features in OS which can be point of conflict with the application under test. For e.g., User Account Control (UAC), AppLocker, BitLocker.

Research Portability

Research done on Operating System and Service Packs is independent of the knowledge of the product. Later, compatibility checkpoints are identified based upon the application behavior.

For instance, Product A extends the support for Windows7. Thereby, research data gathered by a test engineer for Windows7 will contain a common study that can be shared across other teams, who can utilize them to gather their product specific checkpoints in their extended release for Windows7.

Hence, we can say that identification and study of checkpoints is portable across all products, provided they share a common support for an OS.

Tools

OS compatibility tools play a major role in identifying potential conflicts. OS vendors release them on purpose, as they anticipate compatibility issues. Few of the tools are provided by Microsoft like Driver Verifier, Application Verifier and IE Compatibility test tool.

Let us understand OS compatibility model with the help of a case study. Below you can see four stages of the model.

4-Step OS Compatibility Testing Process

Identify

In this stage we identify the OS or Service Pack for which our product enhances the support.
Let us say, Application XYZ, which contains add-ons for Internet Explorer & Microsoft Office and drivers for device blocking and file filter, starts support for Windows Vista.

Research

In this stage, we research about the compatibility checkpoints using various resources such as

a) OS Release document for new features
b) OS known compatibility issues published by the vendor
c) ACT Tool community/vendor assessment (only applicable to windows OS)
d) Online OS Release forums and newsgroups

We gather details of all compatibility checkpoints that may conflict with the application under test. For instance, in our case study analyzing checkpoints for Windows Vista leads to the following list:

a) Windows Vista UAC
b) BitLocker
c) Internet Explorer Protected Mode
d) Windows Resource Protection
e) Fast User Switching
f) User Interface Privilege Isolation

Later, we design the scenarios to test based upon these compatibility checkpoints.

Perform

All scenarios that are designed in previous stage are executed here. Here, we also use help of compatibility tools to uncover compatibility issues.

Coming back to our case study, Application Verifier can be used against XYZ to uncover few OS compatibility risks based upon different checks such as I/O Verification, Low resource simulation and deadlock detection. Likewise, we can use IE Compatibility test tool to verify if the changes in OS Architecture did not hamper any functionality of application's add-on for IE.

Report

Lastly, we log all the issues encountered and record the results for further analysis.

5. Third-party Compatibility Testing Model

"Third-party software is software which can be used in conjunction with some software or hardware but is not associated with either the manufacturer or the user. The testing of Third-party software is Third-party compatibility testing"

Let us take a quick example that explains 3rd party compatibility issues. Add-ons that get integrated in Internet Explorer (IE) are specially designed to perform extended functional tasks. Likewise, IE provides a common platform where these add-ons sit and can be seamlessly integrated. However, IE and add-ons will have a common code base that could result in potential compatibility issues.

Most of the 3rd Party Application incompatibilities arise due to:

1. Widely use of Internet for activities such as browsing, social networking and so on.
2. Changes in the implementation model of 3rd Party applications
3. Adoption of newer industry standards which invalidates older mechanism

With 3rd party compatibility model, we stay on top of these issues and try to mitigate the long-term effects they may cause.

Application Identification

Millions of applications exist but is it necessary to test with all of them? Do you know which application could affect your product and vice versa? Does your inventory include the applications installed on your customer environment?

Application Identification answers your questions where we prepare list of applications, which we see can cause a potential conflict with our product. The data is gathered from various sources including:

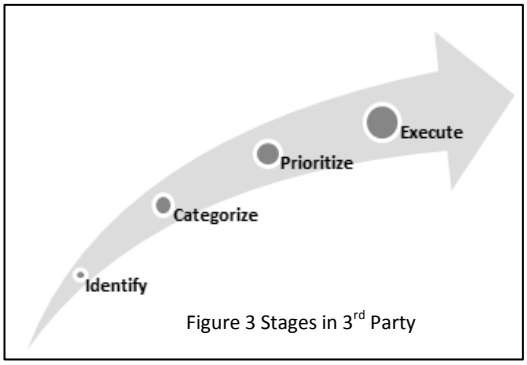

Figure 3 Stages in 3rd Party

1. Application Compatibility Toolkit (ACT) to identify applications installed on a specific system.
2. Tools to identify application installed in customer environment
3. Internally reviewed and installed product repositories
4. Any Enterprise Inventory tools

In practice, with the above criteria, we identified close to 200 3rd party applications. Noticeably, we discovered that 90% of the compatibility issues identified till now in our product involve these 3rd Party applications.

Application Categorization

Now that we have an inventory list from Application Identification, how do we test with these applications? We definitely need some guidelines and a common criterion based upon their functionality and features.

Generally, the applications will fall into one or the other category due to its functional behavior. Using Application Categorization, we group applications into different categories based upon their underlying technology. For instance, IE, Firefox & Chrome will fall into "***Browser***" category and WinRAR, WinZip & 7zip will fall into "***Archiver***" category.

Once we have categorized the identified applications, next steps include:

- Creation of guidelines per category to create a test suite. This includes identifying the conflicting areas between applications.

- Preparation of Compatibility plan and creation of scope based upon release cycle. This includes the high-level plan keeping in consideration resource and time availability.

Application Prioritization

Is it practical to test all the applications that you categorized in a release cycle? Will you have that bandwidth to test with all pre-selected products from last phase?

We follow here the basic principle of prioritization to optimize the output with minimum effort. In Application Prioritization phase, we prioritize applications based on three factors that streamline efforts to cover most critical applications early in testing cycle. They include

a) Popularity Hits
b) Defect Fraction
c) Customer Escalation Fraction

Let us see below each in detail with the help of a case study on Browsers Category. To simplify, we have identified only three most popular browsers viz. Firefox, Chrome and IE.

Popularity Hits (PH)

This section is to measure how popular the application is with respect to other applications in the same category.

We derive this by dividing the number of hits of an application by total no of hits of all the applications in that category. Hits of any application are the number of results of that particular application on a popular search engine. So, in our example let's say

Firefox has 801m, Chrome has 622m and IE has 1.82b hits. Now Priority Links of Firefox can be calculated as

#Hits of Firefox/Total #Hits = [801]/ [801+622+1820] = 0.246

Defect Fraction (DF)

Here, we give priority points to an application based upon its historical defect data. If an application has more defects compared to other applications in that category, priority points of that particular application will be more. It is calculated by

#of Defects of an Application/Total # of Defects of All Applications under that category

E.g. Firefox has 4, Chrome has 3, and IE has 3 defects

Defect Fraction of Firefox = [4]/ [4+3+3] =0.4

Customer Escalation Fraction (CF)

This factor gives priority points to an application based upon the escalations observed from the customer environment. It is calculated by

Of Escalations of an Application from Customer / Total # of Escalations of All Applications under that category

For E.g., Firefox has 1, Chrome has 0 and IE has 1 escalation.

CF of Firefox = [1]/ [1+1] = 0.5

Priority Points

Once we identify all three factors of an application, we calculate Priority Points in the following manner.

$$PP = \left(\frac{PH}{TPH}\right) + DF + CF$$

PP – Priority Point
PH – Popularity Hits (Of Application)
TPH – Total Popularity Hits
DF – Defect Fraction
CF – Customer Escalation Fraction

Figure 4 Priority Points calculation

So, in our example:

Firefox Priority Points = 0.246+0.4+0.5 = 1.146
Similarly, IE & Chrome Priority Points can be calculated as 1.361 and 0.492 respectively.

The higher the Priority Points of an application, more critical the application for us to test. So, in Browser category we have IE, and then followed by Firefox and Chrome in prioritized application list.

Based upon the factors such as availability of resources and time, major or minor release cycle, we can take a call to choose high, medium or low priority 3rd party applications in a category to test.

Execution and Reporting:

This is the final phase of 3rd Party compatibility model. After going through identification, categorization and prioritization, finally we come to this phase where as per the scope we execute the test suite of prioritized products for all categories. During execution we match the actual and expected results and raise an alarm by reporting the issue in case of any deviation.

Let us understand this model with the help of a case study. We have identified two categories Browsers & Antiviruses and applied the different stages of the 3rd party model.

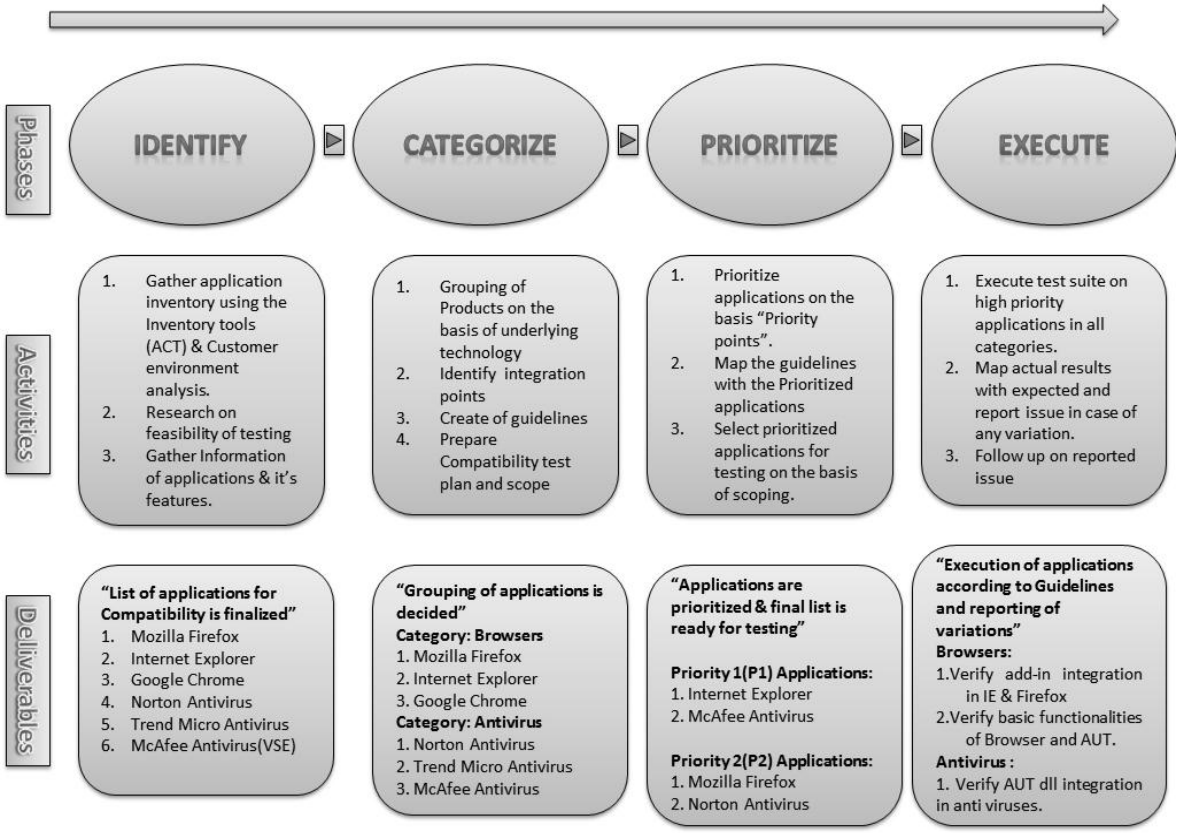

Figure 5 3rd Party Model Case Study for Browsers & Antiviruses

6. Endpoint compatibility testing model

"Endpoint compatibility testing is a process that enables synchronization of endpoint products to run on a computing environment."

In a customer environment, it is important to ensure that the products designed by the same vendor are in harmony. To do this, we have designed endpoint compatibility model that is helpful in identifying the potential conflicts among these products.

To understand this model, one need to understand the following key attributes.

Inter-team collaboration

Products from the same vendor give more scope for doing compatibility testing and that can be leveraged to make your plan effective. We acquire knowledge by internal training, formal or informal discussion with the peer product teams and in-house technical forums that can be used to develop key feature documents (Feature Flash or Knowledge Base Articles) and help us to design compatibility test suites.

Team-Paired Testing

Here, team members from different teams sit together and conduct an exploratory or ad-hoc session to test potential conflicting scenarios between their products.

Defect Identifiers:

Inputs from inter-team collaboration and team-paired testing help in identifying the key areas where the product's functionality ceases to co-exist. These key areas we call as "Defect Identifiers". For instance, a ***Reports*** module exists in a product whose codebase and libraries are shared across all endpoints in an organization. The UI interactions and code libraries of this ***Reports*** module can be considered as defect identifiers.

Pilot program

Large-scale deployment always led to unforeseen issues and companies can prevent these outcomes by installing their products in a simulated environment in-house. This is carried out through pilot program that simulates customer environment in terms of number of nodes and installed applications.

Let us understand endpoint compatibility model with the help of a case study. The diagram below displays the 4 phases in which endpoint model progresses.

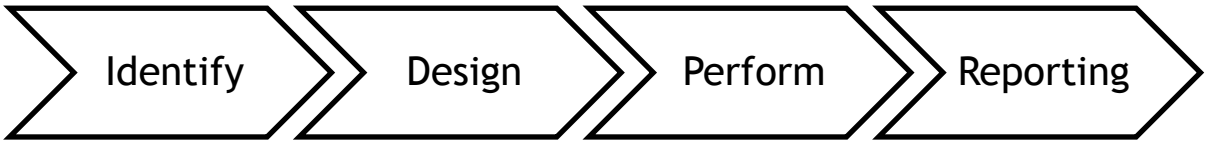

Figure 6 Endpoint Compatibility Model Phases

Identify

The first step in endpoint compatibility model is to identify the endpoint products and their versions that need to be tested. We start with gathering information about environment configuration to test.

Let us take example of two endpoints X and Y managed by a centralized server. X monitors data on the host and alerts the server administrator when any forbidden data is leaked outside the host. Y forbids any malicious file to enter into the host. Although X and Y differ in terms of functionality, they share a common module Z that restricts users logged into the host to tamper with the resources of endpoint X and Y.

Assuming that a new version for endpoint X is released for Windows 7, let us make use of our endpoint model to design the testing strategy for this endpoint X.

Design

Once we identify the endpoints to test, we move ahead with the design phase where with the help of Inter-team collaboration we distinguish the defect identifiers.

Coming back to our case study, as we understand module Z is shared commonly across endpoints X and Y. Inter-team collaboration among endpoint teams X , and Y leads to identify several defect identifiers such as

- DLL's that protect resources

- Watchdog process that accounts for access protection,
- UI component on the server that allows administrator to configure module Z functionality on the client

Based upon these defect identifiers, we design the scenarios to test compatibility between X and Y. Some of the scenarios can include:

a) Install/Uninstall endpoints X over Y and vice versa and verify the integrity of shared DLL files of module Z
b) Tamper the resources of endpoint Y such as registry entries, program files, processes and drivers while endpoint X is installed.
c) Verify that changing UI components to non-default values reflects to client properly while X and Y are present.

Perform

Now that the test scenarios are written and the execution plan is created, we perform the scenarios on the defined test environment.

These scenarios are executed in phases. In first phase, we run test scenarios on an environment with minimal configuration and record the results. In subsequent phases as the product comes near to beta release, we use help of pilot program and try to uncover issues by running it against a simulated customer environment.

Reporting

We capture our results in the test management software. The RAG (Red Amber Green) status is determined. This is a term used to decide a go-no go in various milestones of a product and is used with respect to compatibility in our model.

7. Lessons Learnt

While we did compatibility testing in our product, we identified several areas mentioned below, which we think would help you to follow our framework:

Admit first

Management can praise or be prejudiced about its testing team capabilities but it does not give them the right to judge the issues found on the customer site. We often see that product management does not take responsibility of a customer issue until it is a major road block or customer persists.

We can improve this situation by accepting more proactive strategies. Also, we should learn to accept our product fault that sets the need to adapt compatibility testing.

Prioritize

Remember that 100% testing is not possible, specially, in compatibility; we always deal with modules of other products. Therefore, prioritization of test scenarios is a key that will help a compatibility tester in maximizing the results keeping efforts constant. The factors like checkpoints, Defect Identifiers and 3rd party application categorization and prioritization will always help you to achieve that.

Always Client First

Applications running on client environment are of prime importance to us. Therefore, gather as much information as possible about most installed applications; frequency of OS distribution, number of licenses procured and any other related information and further use this to build a compatibility testing strategy for the three testing types.

Following the above, testing our product on the customer-simulated environment not only will add to product confidence rating but also will build a fine rapport with the clients.

8. Conclusion

The fundamentals of compatibility testing reveal that you will uncover these issues only when your product interacts with other products. Hence, it is difficult to find these compatibility issues via test scripts or regular functional testing. In that case, to mitigate the gaps with other products one needs to follow a specific framework that aids him in finding flaws early in product cycle.

Application Compatibility Framework promises to empower functional testers to perform compatibility testing and guides him to take the pre-defined path to achieve a better result.

We hope that our framework simplifies understanding of application and optimize the efforts put down by testers in finding compatibility flaws.

In terms of results, we found close to 5% of total defects using this framework, which we otherwise would have not gotten in absence of our model. After this approach has been adopted, we also see a downward trend in number of sustaining hotfixes released to the customer due to compatibility issues.

9. Road Ahead

It has been two years since we adopted compatibility testing and we continue to focus on different areas of improvement.

Compatibility in testing context is a niche area and our efforts were focused till now keeping in mind our product requirements.

Though we have observed 10% growth in defects found via our framework from last release, the customer compatibility issues did not show a substantial decline due to several factors such as rapid changes in product portfolio, unavailability of proprietary applications used by customers and increase in number of clients. Currently we have close to 200 Applications in our 3rd Party portfolio that are prioritized and tested as per release requirements.

In future, we plan to take up tasks such as development of Automated Test Bed, combining Interoperability with Compatibility testing, streamlining Identification & Prioritization based upon qualified factors and enhancing OS Architectural knowledge to develop a comprehensive OS based compatibility model.

References

1. http://blogs.computerworld.com/node/3309
2. http://www.phonescoop.com/news/item.php?n=2336
3. http://news.cnet.com/The-XP-alternative-for-Vista-PCs/2100-1016_3-6209481.html
4. http://www.usertesting.com/blog/?p=242?utm_source=e011&utm_medium=email&utm_campaign=Critical%2Busability%2Bmistakes%2Bof%2B2011