

Reliability: Analyzing Solution Uptime as Business Needs Change

Herbert H. Thompson, Ph.D.

November 2005



187 Ballardvale Street, Suite A260. Wilmington, MA 01887

Tel: (978) 694-1008 Fax: (978) 694-1666

info@securityinnovation.com www.securityinnovation.com

Executive Summary

”Reliability” is often cited by IT Professionals as a key requirement for any technology because delivering reliable IT services is a non-negotiable expectation business decision makers and end-users have for IT departments. Unfortunately, measures of reliability that help IT professionals make technology choices that meet this expectation have been a much greater challenge than establishing it as an IT goal.

Historically, measures of “reliability” have boiled down to benchmarks taken in isolation, pitting components such as the Linux kernel against BSD or the Windows operating system. Measures such as these treat a system as a collection of individual components, and assume that measuring “reliability” as “uptime” of components in isolation can support broader conclusions about the reliability of an entire system. Comparisons like these fail to capture “reliability” from a real-world business perspective, however, because they do not address *IT pain*. The “pain” of *unreliability* stems from two scenarios:

- Business solutions not being available when needed—this encompasses not only IT meeting expectations day-to-day under steady-state conditions (think of email each morning), but also consistently and predictably delivering new capabilities to meet changing business requirements (think of a company adding personalization to their e-commerce site to keep pace with competitors); and
- IT being able to meet business needs only through a combination of “heroic efforts” to overcome unpredictable or failure-prone behavior by technologies, leading to long hours or short-term fixes that meet current needs but put IT’s long-term ability to deliver reliable solutions within budget at risk.

These scenarios often occur in tandem, with one feeding the other, and taken together over time they can cripple IT productivity and the ability of the business to compete and grow. Conversely, “reliable” technologies have characteristics that contribute to *both* consistent, predictable availability of business solutions *and* IT efficiency and control over the complexity of the environment.

In researching this issue, we have met with many CSOs, CTOs, CIOs, researchers, programmers, quality assurance engineers, project managers and other project stakeholders. We looked at the causal factors for business solutions failing and the hands-on challenges of IT departments. We found that measuring reliability in a meaningful way meant understanding *solution availability as systems change over time* with patches and updates, new business requirements, and other environmental changes. It became clear that the key to managing *reliability* was to choose platforms and applications that enabled IT to be *efficient* and facilitated a *simpler* environment over time.

In this report, we present the results of a study designed to compare two platforms—Microsoft’s Windows Server System and Novell’s SuSE Linux Enterprise Server—under

evolving business requirements over an extended period of time. Our goal is to help IT administrators make informed and business-savvy deployment decisions by identifying meaningful differences in real-world reliability between these platforms. We also look at the broader implications of the model used by the two vendors and how the 3rd party community builds upon those systems.

What differentiates the methodology presented here from component-based studies is that it considers the reliability of a system *holistically*. A system is a collection of smaller parts, each of which can be benchmarked and studied in isolation but assessing the true “reliability” of that system means understanding both how those components interact and their evolution over time. The methodology presented here does this by *simulating* evolving solution requirements. Specifically, it allows us to compare two systems by adapting those systems to fulfill new business requirements over time and measuring solution failures and downtime.

We conducted an experiment pitting Windows 2000 Server against SuSE Linux Enterprise Server 8, simulating the one year period from July 1, 2004 to June 30, 2005, which was the most recent full year at the time the study began. During this period, we simulated the evolution of an ecommerce company that has changing business requirements while continuing to maintain security through patch application. At the end of the period, both systems are then transitioned to the more recent versions of their respective operating systems, Windows Server 2003 and SuSE Linux Enterprise Server 9. Security patches were applied in 1 month increments, while new business requirements appeared at three month intervals. The experiment was conducted by three expert Windows administrators on the Windows side and three expert SuSE Linux administrators on the Linux side.¹ To be clear, the focus of this paper is to present a comparison methodology that can be applied and built upon. The sample, although too small to provide conclusive statistical comparisons, illustrates the methodology and begins to shed light on some key model differences between the platforms. A welcomed next step would be a more expansive study based on this foundational methodology with a larger sample size, additional business requirement scenarios and that looks at a wide array of platforms.

The results of this initial study showed some interesting patterns. One of the most heavily touted benefits of Linux is its high modularity and the granularity of control that administrators have over a system. In the experiment, we found that such flexibility also leads to ambiguity for administrators in terms of paths to follow when resolving conflicts. On the Linux side, each administrator pursued vastly different paths to resolve dependency conflicts that arose when new components were installed. The result was solutions that grew in complexity and heterogeneity rapidly over time.

¹ Each SuSE Linux administrator had at least 5 years experience administering Linux in an enterprise setting. We also required 2 years minimum experience administering SuSE Linux distributions and at least 1 year administering SuSE Linux Enterprise Server 8 and half a year administering SLES 9 (released in late 2004). Windows administrators all had at least 5 years experience administering Windows servers in an enterprise environment. These administrators also had at least 2 years experience administering Windows Server 2000 and at least 1 year administration experience with Windows Server 2003.

In contrast, Microsoft has pursued a philosophy it calls “integrated innovation” where much of the core system functionality is incorporated with the operating system itself. During the experiment, all Windows administrators followed a fairly homogeneous route to both install patches and apply component upgrades for the simulated changing business requirements.

The results show how different technologies contribute to real-world reliability:

- ***IT is able to deliver solutions predictably and consistently:*** Two of the three Linux administrators were unable to meet all business requirements within the time constraints of the study; in contrast, all three Windows administrators met all business requirements. Greater complexity of component dependencies which had to be tracked down was a key contributor to failure to meet business requirements for the Linux solution.
- ***IT is able to deliver reliable service efficiently*** Using ‘stop loss’² times (maximum allowed time of 10 hours per task) for those Linux administrators unable to deliver on a particular requirement, on average the three Linux administrators were about 70% slower than their Windows counterparts to fulfill business objectives. This was in part driven by more system failures experienced by the Linux administrators (14 compared to 0 for the Windows administrators) and a greater number of patches that needed to be applied to the Linux systems (in total, 187 compared to 39 for Windows).
- ***IT is able to keep control over the complexity of the environment and ‘keep it simple’:*** The only Linux administrator who was successful in meeting all requirements installed components and component versions that were not directly supported by the vendor (and in some cases custom compiled) that effectively put his *system* into an unsupported configuration³. While the configuration did meet functionality requirements, the administrator is now “on his own” to resolve potential future system failures. It has also increased the IT administrative burden given that any future patches to the unsupported components would now have to be gathered from alternate sources and in some cases edited at the source code level and recompiled. On the Windows front, the system was maintained by components provided either from Microsoft or from the 3rd party component vendor and all configurations were within the boundary of support.

² For one of the milestones, two of the three Linux administrators were unable to complete a component upgrade because of cascading package dependencies. A “stop loss” time of 10 hours, the maximum time allowed, was counted as the total time taken for that task.

³ When a Linux distribution ships it contains a snapshot of versions of constituent components like Apache, MySQL, PHP, etc. When 3rd party solution providers produce their applications, they must also choose a set of components they build against, often forcing enterprise users to “upgrade” individual components in their distribution and taking the platform out of its vendor-supported configuration.

This study provides insight into how different technology choices may contribute to or prevent reliability pain as experienced by IT departments and businesses. This initial foundational research takes into account the real-world pain points of organization such as:

- *IT and Organizational implications:* It is clear that organizations feel pain from solution downtime but some of the biggest pain as an organization can come from not adapting quickly to changes in the market. Any meaningful reliability study therefore needs to consider how IT systems, especially those apart of a customer-facing business solution, must adapt dynamically. For example, during one of our research interviews, the CTO of a major e-retailer recounted his biggest personal pain point of the last year: a worm that took their system offline for a period of a few hours. But it was clear that the biggest pain point for the *organization* was a period of several weeks when their main competitor had personalization features that were drawing their customers away. Once they managed to implement their own, the pain disappeared. Conclusion: systems must be adaptable yet remain robust.
- *Solution perspective:* The behavior of individual components will clearly contribute to the overall reliability of a solution. However, it is also clear that this impact is small compared to the characteristics of the ‘glue’ that ties together components into a working solution. As one of our research subjects said: “The components themselves are the simple part. It’s the stuff that ties them all together that keeps me up at night.” Conclusion: we need to look at a system in its entirety to understand solution reliability.

This study was scoped to simulate the evolution of a system over a one-year period within a time-limited, lab setting. By examining the performance differences between the two platforms studied here in terms of time to deliver a stable solution meeting business requirements, we can investigate the difference in performance on a percentage basis. Figure 1 below summarizes the results obtained during our simulated 1 year platform analysis under evolving requirements.

The methodology presented here for making platform comparisons on reliability maps directly to IT pain and cost. This methodology has been reviewed, commented on, and validated by academics, industry analysts and IT thought leaders. The results speak to system and solution deployments in many typical business environments where keeping pace with changing market and environmental conditions is necessary. The process is highly extensible and our hope is that it will serve as a basis for platform evaluations on reliability beyond the Microsoft Windows / Novell SuSE Linux Enterprise Server cases considered here.

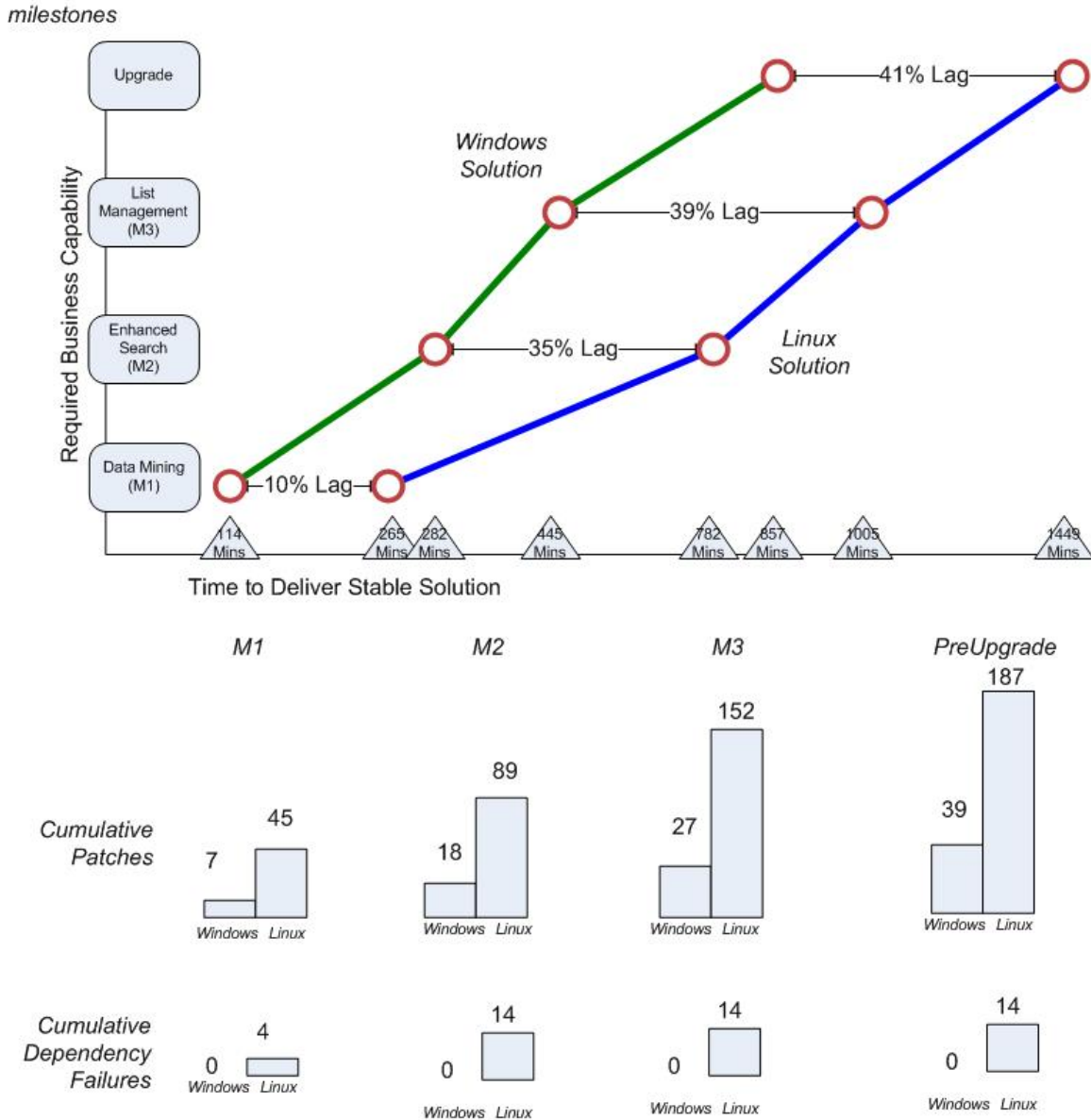


Figure 1 - Summary of the experiment evolving a Microsoft Windows server and a Novell SuSE Linux Enterprise server solution in the one year period July 1st 2004 to June 30th 2005 (the most recent full year at the time the study began) with the same changing business requirements. In the experiment, participants were tasked with applying patches available over that period as well as adding 3rd party enhancements at 3 month intervals (such as a search tool and a data mining tool) to help the solution meet business needs.

Acknowledgements

This study and our analysis were funded under a research contract from Microsoft. As part of the agreement, we have complete editorial control over all research and analysis

presented in this report. We stand behind our methodology and execution of that methodology to determine objective results that will be useful to customers, industry experts and analysts.

We encourage others to examine, thoughtfully analyze and comment on this work. Our goal was to understand some of the key model differences between Linux and Windows in terms of their reliability over time and to perform an analysis based on fact (not speculation) using a transparent and meaningful methodology. During the course of this research we have met with many CSOs, CTOs, CIOs, researchers, programmers, project managers, academics and industry analysts and have incorporated their feedback into the final methodology presented here. Our goal was to provide a general framework for analyzing solution reliability that is meaningful to enterprises and our hope is that others will take this methodology, expand upon it, and conduct their own analysis.

We'd like to especially thank numerous analyst organizations and academics for their conversations, time and insights on this work. We'd also like to thank several individuals we spoke to at ecommerce companies and financial institutions whose PR policies do not allow them to be named but whose time and insights were greatly appreciated.

Introduction

In late 2004, Security Innovation embarked on a study to analyze the pain that organizations have suffered under unreliability. We found anecdotal accounts, benchmarks of certain software components, and extensive speculation on the topic. In most, reliability was measured in terms of components, like the operating system kernel or a web server process. In researching this issue, we have met with many CSOs, CTOs, CIOs, researchers, programmers, quality assurance engineers, project managers and other project stakeholders and found that one of their biggest pains stemmed from solution downtime - the failure to meet business goals - and that the cause of solution downtime was broader than the failure of individual system components.

We stepped back to think about the pain that stemmed from unreliability and wanted to track down its root cause. We quickly found that pain from unreliable software manifests in many different ways from minor annoyances for users to substantial lost revenues for corporations. Smart corporations recognize that the pain they feel in building and deploying systems translates directly into pain experienced by their users. An organization that recognizes and manages its pain is likely to have a user community that is happy, loyal and won't bolt to the competition. The organization that ignores or is unaware of their pain puts its fate in jeopardy.

It seems appropriate, therefore, that in attempting to measure reliability of any sort, a good place to start is to identify and understand the pain points of IT organizations.

Our research indicates that the primary methods of computing reliability are overly simplistic as indicators of real IT pain. For example, *kernel uptime* is commonly cited as a metric of overall platform reliability but as a real pain point for IT organizations the reliability of a single component, even one so central as the operating system kernel, is rarely the largest source of pain. Real pain is caused by the solution itself failing to meet its business requirements.

IT departments are more concerned about *solution uptime* as opposed to *component uptime* because, as a business, they experience pain when their system, website or service is no longer functioning. Certainly the kernel going down will cause the overall solution to cease to function but so will many other things. Although IT organizations will be concerned about the root cause, from a business perspective, the key concern is that the system they rely on to do their business is down. All the contributing reasons for such catastrophes are part of their pain. It is only by understanding this pain, and the many reasons for it, that we may gain insight into the reliability question.

This study looks takes a hard look at reliability pain as business and technology requirements evolve over time. In our paper "Understanding Reliability: How to Measure Reliability and Minimize IT Pain"⁴ we laid the foundation for measurement of quantitative aspects of *system reliability* as opposed to *component reliability*. Specifically, we developed a methodology for reliability assessment based on the

⁴ http://www.securityinnovation.com/pdf/Understanding_Reliability.pdf

evolution of a system over time. We presented an initial proof of concept requirements set that was based upon extensive research of large e-commerce vendors. In this study, we conduct empirical comparisons of solutions based on a distinct requirements set deemed representative based on our customer research. This requirements set tracks the evolving needs of an ecommerce company and has been validated by numerous companies, IT experts, and analysts as representative of real world evolution. While these requirements illustrate only one typical customer case, we urge you to focus on the methodology behind it. The methodology itself has been vetted extensively by the community and we believe it speaks to the heart of solution reliability measurement. Our hope is that others will adopt this methodology for comparison and use it to compare platforms through alternate scenarios.

For true reliability we need to measure solution uptime as it satisfies a set of requirements, as well as measuring user pain while transitioning a solution to meet new requirement sets. Figure 2 illustrates the process of evolving requirements. Here we assume that an IT project begins with some requirement set S_n . For a while, solution requirements are static, but the system still undergoes changes necessary to maintain that solution (patches, etc.). Some interesting elements to measure here are forced upgrades, incompatibilities, downtime due to patch application, time needed to apply a patch, etc.

From an experimental standpoint, these issues can only be measured over some period of “time” to encompass patches and other environmental changes. In our scenario, we chose the most recent year time period from July 1st 2004 to June 30th 2005. As an organization’s needs change, requirements for a solution evolve. In our model we consider *transition processes* where, T_{n-1} represent the process of transitioning from requirements state S_{n-1} to requirements state S_n . Little work has been done to measure solution downtime and customer pain during this transitional period and we believe that studying and measuring this transition process will be extremely valuable for customers making long term technology adoption decisions. Metrics here will include overall transition time, solution downtime due to upgrade incompatibilities, dependency failures, versioning issues, etc. In order to measure transition however, we must first define what it means for a solution transition to a new requirements state to be “complete”. For each requirements state we therefore define *acceptance criteria*, based on new requirements that verify that the solution meets new demands.

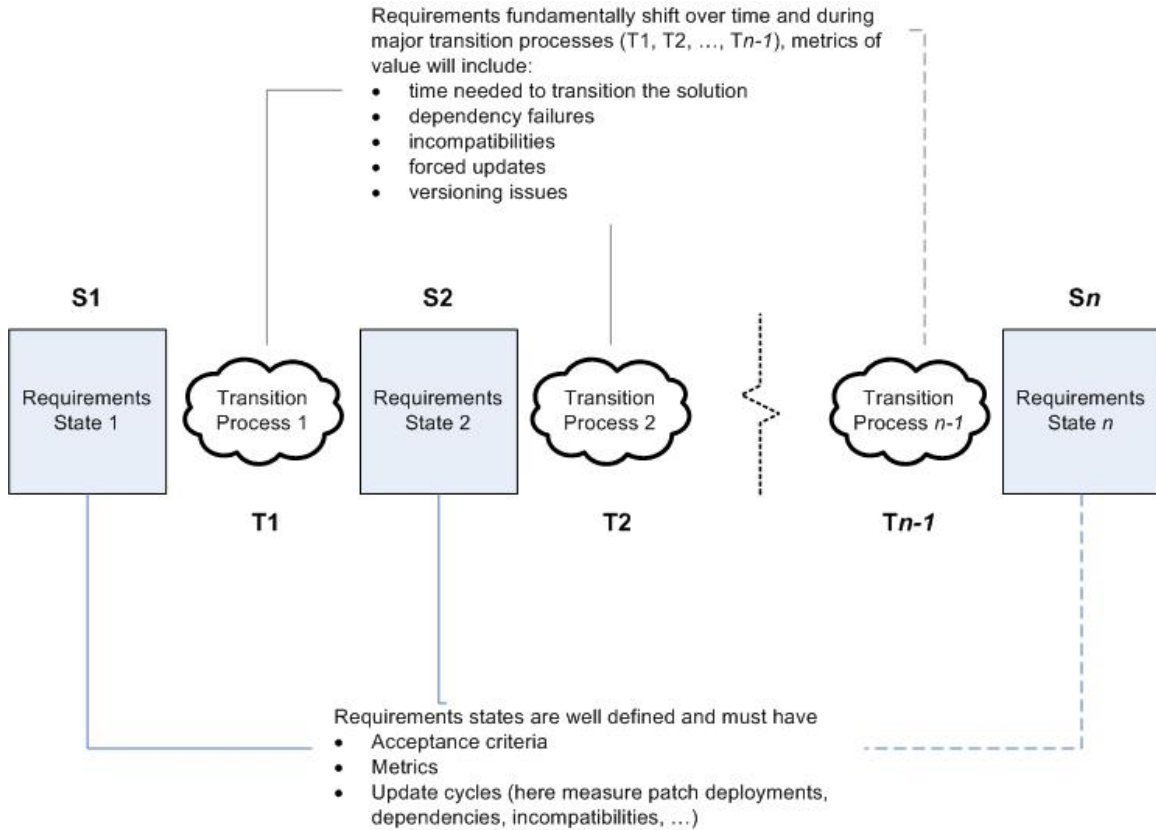


Figure 2 - A model of evolutionary requirements

Analysis Note: SLES vs. RHEL

Red Hat and Novell are currently the leading enterprise distributions of Linux. When we began this study we needed to pick a distribution to use for the experimental part of our analysis, and the choice came down to these two market leaders. Given Novell’s substantial bet on Linux as a platform we began with SLES. Many of the discrepancies in reliability between SLES and Windows however stemmed from fundamental architectural differences that transcend Novell and SuSE. On the Linux side, the results of the experiment showed that pain experienced by administrators stemmed from larger model issues in the platform such as components evolving outside of the operating system distribution, multiple sources for patching after forced upgrades, the “personalization” of installations that comes from a highly flexible and modular architecture, and the support challenges of being forced to go “off distribution” for required versions of components.. These issues weren’t SuSE specific, and speak to the Linux model more broadly. We strongly encourage others to repeat this process with Red Hat and other platform distributions.

For our analysis, we fielded both a Microsoft Windows solution and a Novell SuSE Linux Enterprise Server solution and we tracked administrators through the update and

evolution process to verify configurations, measure downtime, and encapsulate and measure those aspects of maintaining and changing the solution that leads to downtime. There are fundamentally two types of *technological change drivers* though:

- those that come from *evolving functional business requirements* and
- those that are forced due to changes in the *operating environment*

It is worthwhile here to look at the nature of both of these sources of change:

Technology change from evolving business requirements: The mapping here is fairly clear, if business needs require a capabilities that cannot be achieved with the current component set then system must be evolved. More explicitly, as business requirements are translated into application logic, if the new incarnation of the application cannot be supported under the current platform component set then a change is required. An example of these new requirements may be to support more robust search capabilities on an ecommerce system.

Technology changes needed to maintain the solution: These constraints may be static within a requirements state (S1 for example), however, it is the *environment which is changing*. If environmental change forces the application outside of its requirements, typically a technology change is required to restore balance. These changes may be a new vulnerability discovered in the operating system for example, which means that a patch must be deployed to maintain “security”. More specifically, consider the case of a basic ecommerce application that allows searching and payment processing. Expected requirements may be:

- I’m not vulnerable to any known security flaw in my application code
- I’m not vulnerable to any known platform security issues (judged by security patches available from vendors)
- I’m in-line with the performance of my competitors (judged by benchmarks)
- My application is functioning (judged by acceptance tests)
- My system is manageable in an acceptable way (judged by various management/cost metrics)

Examples of environmental changes that may cause our solution to violate those expected requirements may be:

- A SQL Injection vulnerability is discovered in your web site
- There is an exploitable buffer overflow in the web server you are using to host your application
- Benchmark results on transaction are published in a magazine which finds your performance lacking compared with your competitors
- The shipping company for your goods (FedEx/UPS) changes the way it exports data that your application uses

- Other areas of the organization are migrating to newer technology versions, and for cost of management reasons, it makes sense to upgrade the solution's platform operating system to aid manageability.

While some of these environmental changes require code enhancements, like fixing a flaw found in the PHP code, most require churn in system components. Fixing point problems like an OS vulnerability will mostly take the form of applying patches to system components. More systemic issues such as performance problems, enterprise manageability, or a component going out of support from a vendor, require more involved system changes. These overarching needs are one of the biggest drivers for platform upgrades such as migrating from Windows 2000 Server to Windows Server 2003 or moving from SuSE Linux Enterprise Server version 8 to version 9.

In this study, maintaining the functional requirements of security means that vendor security patches must be applied to the system in a reasonable timeframe. We also, however, consider the broader needs of lifecycle support, performance and management which favor underlying platform upgrades. During the 1 year period simulated we therefore apply all relevant security patches in 1 month intervals and then finally migrate the system at the end of the period to the latest platform version.

Methodology

This section outlines the methodology to conduct experiments of evolutionary requirements and measure associated pain that enterprise administrators may experience as a result of this evolution. At a high level, this process involves creating an initial "state" of a system. This state will serve as the basis state, or S1, which satisfies a specific business requirements set at a distinct point in time. Over some period of time, T1, we transition to the next requirements state, S2. This process can be repeated over multiple state transitions such that the transition period Tn represents the evolution process from requirements state Sn-1 to requirements state Sn. In the initial state, we begin with a system that both meets the requirements of S1 and has all released patches applied up till the point in time associated with S1. For example, we can define S1 as a basic ecommerce application running on the Windows Server 2000 operating system, written in ASP, hosted by IIS using the SQL Server 2000 database that is operating on June 1st, 2004. Similarly, we define S1 on the Linux side to be a basic ecommerce application running on Novell SuSE Linux Enterprise Server 8, written in PHP, hosted by Apache using the MySQL database engine. In this case, we assume that the operating system and all of the running applications on that system have been patched up to date as of July 1st 2004 applied to the system. We may then define requirements set S2 to be a more elaborate ecommerce application, one that uses purchasing history and buying patterns to create a more personalized shopping experience for the user. We assume that this new system, with enhanced requirements will be operational on June 30th 2005. During the transition period between these two requirements sets we make the following assumptions based upon standard enterprise customer practices:

- All new security patches in the time period will be applied to the system in addition to any patch marked as critical or recommended. Patches marked as “optional” by the vendor will not be applied. This application will happen in groups corresponding to calendar months in the transition period. For example, if the transition period from S1 to S2 were a calendar year, there would be 12 patch group applications corresponding to the patches available at the end of each month.
- We will transition to the new requirements set incrementally over time, which means making feature enhancements every three months. These feature enhancements will be simulated by adding best-of-breed third party components to the system that meet new requirements⁵. In the running ecommerce example, this could mean adding a new shopping cart component or an add-in data mining tool. In many cases there will be multiple 3rd party products that satisfy functional requirements. Our selection among these alternatives will be made strictly based on largest market share among enterprise customers.
- Whenever changes are made to the system (patches or new component) we apply a set of test scripts and manual acceptance tests to ensure that the system is functioning properly. These tests will correspond to the current set of acceptance criteria.
- There are two drivers for evolution: requirements changes and environmental changes. We assume that requirements changes are driven by changing business needs for functionality and are well-defined in S1 and S2. Environmental changes are driven by the need for increased reliability, security, scalability, manageability, supportability and performance. While requirements change drives the addition of new features, environmental conditions push upgrades of systems and components – the plumbing underneath functionality. We assume that during the transition period, core components of the solution are only upgraded if new requirements (or 3rd party applications installed to meet new requirements) require the upgrades.

Below is a detailed breakdown of the methodology as applied over a 1 year period:

1. Define acceptance tests for each requirements state S1 (beginning of the period) and S2 (end of the period). These acceptance tests will draw directly from the requirements and will by nature be very application-specific. In the course of this study, these acceptance tests will be used as validation that the administrator has not broken functionality through the application of patches and has successfully transitioned to the new requirements state.

⁵ There are many ways to judge “best of breed” for 3rd party components. In our analysis we chose the components that had the largest enterprise market share in their particular areas. Other criteria could have been used however such as largest vendor, etc, but we believe that market share and market leadership is a highly relevant decision criteria given that it is one of the most commonly used adoption criteria by the enterprise. For more details on the 3rd party components, see Appendix 5.

2. Define the package additions that are made incrementally (every 3 months) which move the system towards requirements set S2. These new components will only be added if they move us towards S2 and the specific technologies or packages that meet these needs will be chosen based on market leadership positions on both platforms (see Appendix 5).
3. Define incremental acceptance tests that will be applied after each system patch update and after a new component has been added. These tests will be applied to ensure that nothing was broken in the application by applying the change and that it is functioning according to current requirements.

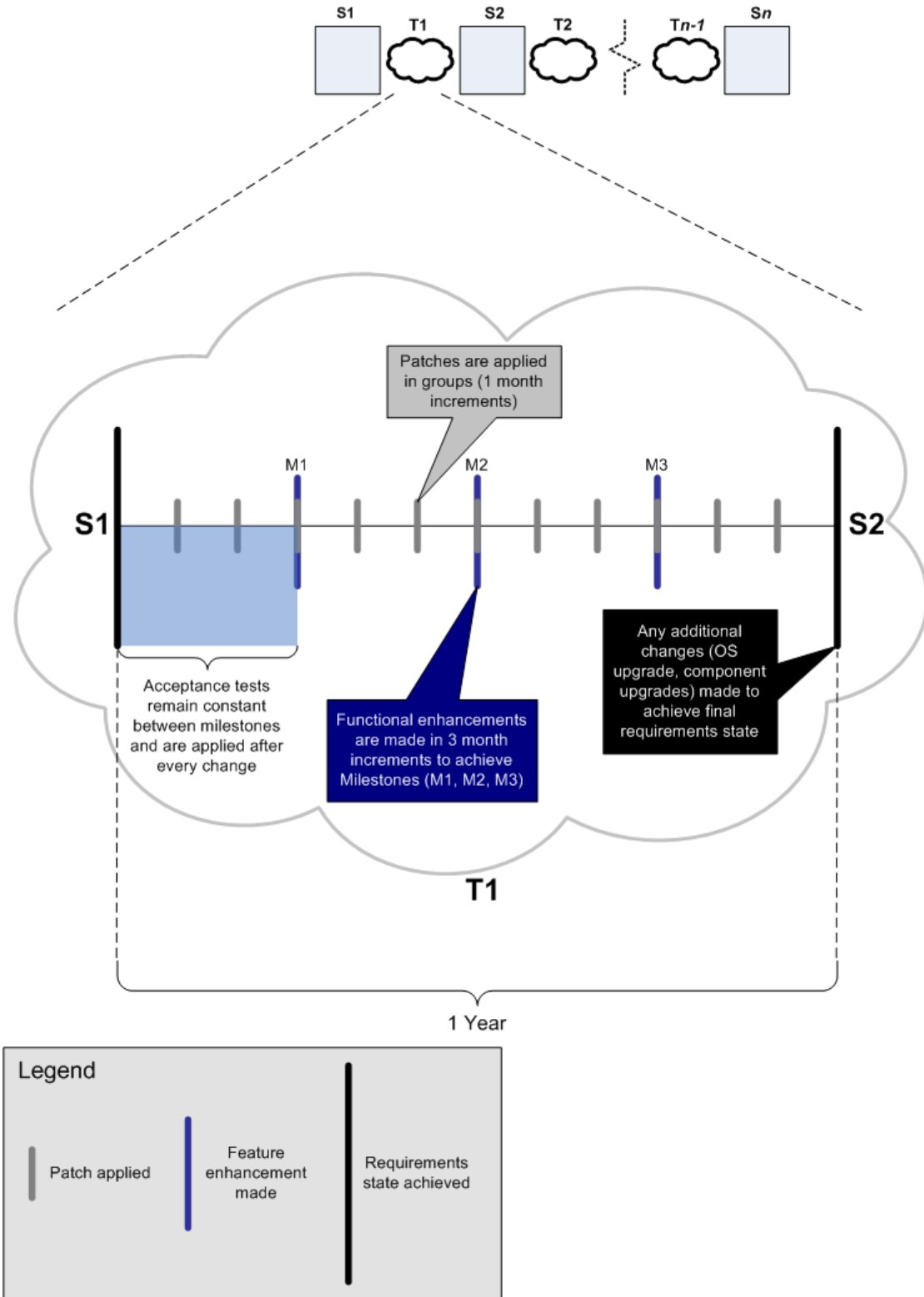


Figure 3 - Looking at the transition process between states, there are two types of system changes: patches and feature enhancements. In this figure we consider a 1 year transition period between states with patches applied monthly and feature enhancements applied quarterly.

4. Install the application, its host operating system and any additional components needed to meet the initial requirements.
 - a. Guide platform installation decisions (like which OS components are installed) based on solution requirements.
5. Apply acceptance tests for application to ensure that it meets its functional requirements of S1.
6. Apply security patches in one month groups in order of release. The only patches to be applied are security patches and those marked critical or recommended by the vendor. Patches marked as “optional” will not be applied.
7. After each patch is applied, perform acceptance tests defined earlier to ensure that functionality is not broken. For each patch group measure impact:
 - a. System reboot – Reboots relate to solution uptime in a measurable way. We thus record if a particular patch group requires a reboot or not. “Requiring” a reboot means that the solution is still vulnerable to the patched issue until the component or solution is rebooted.
 - b. Install time / total resolution time – This is the time taken from the beginning of the patch process until the system is fully functional under the new patch set. To allow an equitable comparison, we install binaries of patches as opposed to any modules that require compilation (for open source) where available. This time will not include the time needed to procure the patch from patching servers due to the potential high variability of the download process across time and connection types.
 - c. System functionality after patch/upgrade – To assess system functionality, a series of functional acceptance tests will be applied to the system. In the case of a web server for example, a sample application will be tested to ensure that its functionality remains intact. In the experimental trials section of this report, we consider an application server which is hosting an ecommerce site. Acceptance tests here will involve conducting several commerce transactions which establish unbroken functionality of core web server and database components.

If the system is not functional after patch application, we will assess impact. Issues must be resolved before moving on in the process but several items will be recorded that relate to potential customer pain:

- i. Patch back-out possible? – In many systems, if a patch or patch group is deployed on production systems it is usually first tested and certified in a staging environment. One of the most acute pain points of companies, however, is when the “certified” patch is then

deployed on a wide scale and only later is it discovered that something in the environment is broken. We will therefore record whether a patch backout is possible for broken functionality and the time/effort/tools involved in the process.

- ii. Why did the system fail? – For each failure we will do a root cause analysis to determine its source. These causal factors will be written up and documented in our analysis. Specifically, we will capture metrics around dependency failures, version demand conflicts and other potential sources of failure.
 - iii. Were there any forced upgrades of dependencies not included with the patch? – An interesting pain point is the forced upgrade of system components when a new patch or package is added to the system. Forced upgrades can sometimes cause a cascading update cycle that significantly impacts system uptime. We will record forced upgrades, the number of components/packages involved, the time associated with applying those upgrades and overall system impact. Forced upgrades can also violate system support agreements in some cases. We will consider these implications in our analysis.
8. Apply system milestone upgrade – Milestone upgrades are applied specifically to meet new requirements. Milestone upgrades include any new components or 3rd part products added to the system.
 9. Apply acceptance tests for the upgrade – During experiments, these new components will be added and the system acceptance tests will be updated to include tests which verify that both the original application and the new components are functioning properly. Participants/researchers will not modify the application code to incorporate these new features into business applications but tests will be done to ensure that these components are working properly and that meshing them into the application is only a matter of application code change. Metrics here will follow the same format as Step 7.
 10. Go back to Step 6 until time to apply final system upgrades to achieve the final requirements state (S2)
 11. Apply final system upgrade or migration to satisfy S2 requirements.

Analysis Note: Migration vs. Upgrade

In the experiment scenario outlined below, we measure and report on migration. Participants though were required to perform both a migration and an upgrade in order to measure differences between the two from a timing and complexity standpoint. The majority of enterprises we spoke to in the course of this study indicated that they performed migrations as their primary means of moving to a new operating system version. Several, however, indicated that upgrades were performed occasionally and we

thus wanted to capture both strategies in our results. Other than small timing variations, no material differences in the migration and upgrade scenarios were found during the experiment.

Specific Requirements Scenario – E-Commerce

The internet has forever changed the way businesses sell and market their products and services. Many companies, motivated both by a wider market and pushed by competition, have digitally extended their enterprise through ecommerce sites. In this experiment we will look at the case of a broker of physical commercial goods through an e-commerce site. Initially, this site offers basic product search capabilities, shopping carts, user logins, and payment processing functionality. The company is currently moving to a more personalized marketing and sales model, allowing users to process returns online, view order history, manage shipments, and use this historical data to make more personalized marketing pushes such as featuring specials on the landing page tailored to the user based on ordering and browsing patterns.

We consider the evolution of this company from basic ecommerce requirements through the infrastructure to deliver history-based targeted marketing. In defining both the initial and final requirements for this company, we tapped the expertise of leading retailers and the insight of business analysts. This represents set of changing business requirements for such companies that will serve to illustrate the evolving demands of servers that must deliver these business solutions. Below we define these “states” and the specific requirements of solutions at each stage.

Initial Requirements State (S1) – Basic Ecommerce Functionality

State 1 (S1) represents an ecommerce site that has basic search and payment processing facilities. The site is *stateful* in that it stores personal data such as credit card information, and remembers the items in a user’s shopping cart so they can return to the site later and continue a shopping session. Many modern ecommerce sites fall into this stateful category, but that is beginning to change. Many of the larger and more mature e-commerce sites are now turning toward new features and strategies to lure customers in the face of increased competition and a growing population of Net-savvy customers. We consider the evolution of such a company in this experiment.

The business requirements for this initial state are outlined below:

- Search capabilities
- Storage of user-supplied product reviews
- Login capability for account management (credit card number storage, etc)
- Payment processing capabilities
- Supports secure transactions (SSL)

Implementing these business requirements require underlying technology such as servers and robust applications which run on those servers. Below is a list of the technical requirements to achieve S1 business requirements:

- Platform server OS
- Web server (e.g. IIS, Apache, Tomcat)
- Integration of web server with backend database server (access components for OLEDB, ODBC for example)
- Backend database server
- Scripting platform (ASP.NET, PHP, Perl)
- Certificate support (SSL transactions)

Final Requirements State (S2)- History-Based Targeted Marketing

Some of the leading ecommerce sites are moving from stateful to a truly personalized experience for customers. They are adding features not only to recognize return customers but to remember what the customer likes and dislikes in order to provide them with a more individualized shopping experience. Below are the business requirements for the company who wishes to capitalize on order history and trends to implement history-based targeted marketing.

- Track orders (interaction with shipper sites)
- View history
- Manage returns
- Subscribe to product update information
- Customized landing page
- Marketing sent based on purchasing trends (genre, time of year, location based, etc.)
- New product alerts created based on trends
- Cross-marketing with partners

To achieve these business goals the company determines the following technical requirements:

- Enhanced application code
- Enhanced data storage and retrieval needs (enterprise class search)
- Support for direct access queries to external data sources
- Customer buying and trend tracking
- Data mining and reasoning technologies
- Support for cross site, real-time data flow management
- Support for mailing list management and automated targeted email marketing campaigns

We assume that the organization moves to these requirements iteratively during its transition period. Specifically, we will consider evolution over the 1 year period beginning July 1st, 2004 (with the system meeting S1 requirements) and ending on June 30th, 2005 (with the system meeting S2 requirements). There are essentially two types of changes that need to be made to the system to achieve this:

- System component addition/changes - “plumbing” to support the evolving business requirements
- Application code changes – enhancements to the code to use new components and features

While we believe that the study of application code evolution and the ease with which a skilled programmer can adapt code on different platforms is an interesting area of study, in this analysis however we focus strictly on the user pain and downtime associated with system component additions and changes. We believe that such changes reflect pain that is felt widely, will be felt fairly homogeneously across organizations and can be measured repeatedly. These are pains that all users must face. As requirements change, components must evolve to keep pace. Component change though is also driven by environmental pushes towards increased user expectations of performance, reliability and security.

Specifically, we will consider the case of an ecommerce vendor who is moving from S1 to S2. We study the evolution of two platform implementations, one using Windows and the other using SuSE Linux Enterprise Server. We will consider the initial application (meeting S1 requirements) being hosted on both Windows 2000 Server SP4 and SUSE Linux Enterprise Server 8. We assume the organization has the following IT policies:

- The only non-requirements mandated system changes are those implemented through security patches released by the vendor during the period
- Security patches released in any given calendar month are applied to installed components at the end of that calendar month
- In moving to S2, the company adds features needed to achieve S2 incrementally over time in *milestones* (defined below) at three month intervals
- The organization uses best-of-breed third party components such as data mining tools and search components to enhance the functionality of their site and these components are added at milestone markers
- At the last milestone, M4, the organization makes the final additions to achieve S2. This includes broader performance, reliability and inherent security requirements which require upgrading the underlying operating system.

The specific milestones are described below:

Analysis Note: Building vs. Buying Components

In this study we use best of breed 3rd party components (see Appendix 5). Another option is to build these capabilities internally through custom application code. From an

experimental standpoint, however, custom code is highly variable in that skill of the programmer can vary extensively and is difficult to quantify. We therefore chose to look at solutions that were already developed by 3rd parties and have proven themselves in the marketplace. In each product category (search, data mining, list management) there are many competing products that would be suitable to add the functionality required. During the experimental phase of this project our selection of specific products was guided by their market leadership position, strong reference customers, and support for both platforms. After countless conversations with analysts, CIOs, CTOs, academics and IT managers, we strongly believe that these criteria are some of the most widely applied in the industry for choosing among competing products. Each product had a strong adoption rate among the Fortune 500. A fairly extensive list of reference customers for the products is included in Appendix 5. Additionally, we imposed the added constraint of selecting products that were developed for both Windows and Linux environments to ensure the most meaningful and consistent comparisons. Our primary goal in this study is to understand how the Linux and Windows models differ when it comes to the typical evolution of business requirements. We both encourage and welcome others to repeat this process using other products and look forward to seeing such results published and discussed in an open forum.

Initial Ecommerce Application

The initial ecommerce application was written to meet the requirements specified in S1. Developers with at least 4 years experience respectively in ASP and PHP development were solicited to develop an ecommerce solution that conforms to the basic requirements set out in S1. Both sites were functionally equivalent, allowing users to browse items, add them to a shopping cart, purchase them through a secure connection and write reviews for those products that could be viewed by others. Each application was required to pass the acceptance tests outlines in Appendix 1 both at the beginning of the experiment and then again after every system change. The application is used as a yardstick of business solution functionality, meaning that if the acceptance tests pass we consider the system and the solution to be in a functioning state. In real world deployments, this application code would likely also be modified at milestone boundaries to make use of the functionality provided by 3rd party components such as mailing list managers. Solution breakages due to code change, however, are as strongly related to the skill of the coder as they are to inherent incompatibilities. Instead, we chose to keep the experiment as objective as possible by keeping the functionality of the test ecommerce application constant and then augmenting acceptance tests with additional checks to ensure that the 3rd party component was also functioning after system changes.

Milestones

After every three months, administrators were required to add new functionality to the solution through the installation of a 3rd party component. These “milestones” represent enhancements of the solution driving towards a personalized ecommerce experience. In

each category we went with the market leader in the enterprise space (see Appendix 5). The specific milestones are defined below:

Milestone 1 (M1) – Data Mining

The first milestone enhancement to our ecommerce solution is a data mining utility that allows the organization to both capture and extract browsing and buying trends of users. This is a critical step towards true targeted marketing and requires that additional data be stored in a highly available and immediately reference-able manner.

Milestone 2 (M2) – Enhanced Search

Search is a critical element of any electronic commerce site. As the site moves towards history-based targeted marketing, the solution needs search capabilities that scale with product and service growth and is able to accommodate a wide range of dynamic data. The search market is highly competitive but can be divided broadly into two classes: solutions that are hosted on the service provider's hardware and sites, and solutions that are self-contained and can be installed and supported directly at client sites. We chose to focus on the latter – the self-contained and installable offerings – after multiple conversations with CIOs and CTOs. While several large ecommerce vendors have chosen to outsource search, we found that a greater number preferred the control of an internally run (and sometimes internally developed) solution.

Milestone 3 (M3) – List Management

As an ecommerce company evolves, so to does its need to promote its products and services. Specifically, the company would like to leverage its knowledge of a particular customer's buying or browsing habits to market to them in a personalized and effective way. A critical piece of this targeted marketing solution is a mailing list manager which handles the organization, classification and deployment of email marketing campaigns. When tied in with the data mining component, list management provides a powerful mechanism for promoting good and services with the maximum likely return on investment. A good solution here is critical because one cannot squander client lists by underutilizing them or by inundating them with information on products that are unlikely to be of interest.

Milestone 4 / Requirements State 2 (M4/S2) – Platform Upgrade

In the final transition to the history-based marketing requirements state, our solution needs enhanced performance, reliability, manageability and security. Often a sweeping improvement of these factors, coupled with aging support agreements, prompts enterprises to upgrade the platform OS. In this final transition, we move the two systems to the latest versions of their respective operating systems. There are two primary modes for transition: migration and system upgrade. The majority of enterprises we spoke to in the course of this study indicated that they performed migrations as their primary means of moving to a new operating system version. Several, however, indicated that upgrades were performed occasionally and we thus wanted to capture both strategies in our results

(see analysis note). We therefore consider both the upgrade and migration scenarios here, and transition the Linux and Windows solutions to SuSE Linux Enterprise Server 9 and Windows Server 2003 respectively.

Experimentation

The experimental part of this study serves to illustrate the methodology and provide some initial insight into the impact of some of the model difference between Windows and Linux on solution reliability. We consider the ecommerce scenario described above and implemented across both the Linux and Windows platforms. Evolution of the solution was considered over the time period from July 1st 2004 to June 30th 2005. For the initial requirements state, S1, we use Windows 2000 SP4 and SuSE Linux Enterprise Server 8 as the base operating systems. Six highly skilled administrators were recruited, three experts on Windows and three experts on Linux. Appendices 3 and 4 present the criteria used for selection of the administrators. While the sample size of three administrators on each platform is too small to make sweeping conclusions about the specific systems studied, it does provide some initial insight into some key differences in their models.

At the beginning of the experiment, administrators were presented with the following context document:

Introduction

The overall goal of this project is to get a better understanding of how reliability of a system changes over time and as business requirements evolve. For the next several days, you will be asked to apply patches and add additional components to a functional web server. Specifically, you will make twelve (12) updates to the system which are described in the “Scenario” section below. For each update, there is a checklist of tasks you will need to perform along with a series of questions based on that task you will need to answer. Please be precise as you record timing information and be as descriptive as possible about your experiences. Checklists will be provided for you along with locations of patches, components, etc. that you will need to install. We will consider an update to be “finished” when the components or patches are successfully installed and a series of tests are performed to ensure that these components along with the original ecommerce application are functioning properly. Starting time for a particular update should be recorded after package download. For resolution time (the time from when you begin to install a patch till acceptance tests pass), however, please include the download time of any additional dependency upgrade components, but also make a separate note of how long just the component download time was.

Scenario

An ecommerce company has an established portal used to sell its products over the internet. You have become the system administrator for this site as of July 1st 2004. The company begins with fairly minimal ecommerce functionality and specifically supports the following requirements:

- *Search capabilities*
 - *Storage of user-supplied product reviews*
 - *Login capability for account management (CC number storage, etc)*
 - *Payment processing capabilities*
- Supports secure transactions (SSL)*

The company's policy is to apply security patches to the platform at the end of each month. The only patches to be applied are security related, or those marked as "critical" or "recommended" and sanctioned by the distribution (i.e. Novell or Microsoft) and not from package distributions (i.e. a patch released by the MySQL team not on the Novell/SuSE site). If, however, you are forced to install components unsupported by the platform vendor, you must also seek package distribution patches as well. Therefore, over the one year period, a total of twelve (12) system changes need to be made as outlined below. At the end of the period (June 30th, 2005) the company performs a migration of the underlying platform to the most current version, which will include applying all distribution patches up to and including June 30th, 2005.

Administrators were then given machines with the operating systems installed and all distribution patches applied.

Windows Background

With the introduction of Windows Server 2003 Microsoft has made significant strides in both reliability and security. Many IT houses are now in the process of migrating from Windows 2000 server to Windows Server 2003. In this experiment we consider the evolution of our e-commerce solution as implemented in Windows 2000 server and migrated at the end of the period to Windows Server 2003 as the company moves towards a targeted marketing ecommerce site. In contrast to Linux's modularity, Windows is constructed under a model which Microsoft calls "Integrated Innovation" where a large amount of functionality is integrated into the core operating system. While this offers less granularity of control as compared to Linux, it appears to create a more homogeneous and predictable base for 3rd party vendors to build upon. As compared to the Linux model, most core components of Windows do not evolve outside the OS distribution with gives a certain degree of predictability to deployments. Additionally, those components which do evolve such as the browser Internet Explorer, Windows Media Player, etc. are distributed after compatibility testing with the base OS and their installation leaves the system in a Microsoft-supported configuration.

Linux Background

Linux has a long history as a robust and highly modular operating system. Reliability of the Linux kernel has often been touted⁶ and more general conclusions have been drawn for reliability of the platform based on the results of the kernel. Most of these studies have considered environments which are fairly static, where requirements remain constant over time and very little churn happens in platform components from patches,

⁶ <http://www-128.ibm.com/developerworks/linux/library/l-rel/>

upgrades and enhancements. In real-world deployments, especially for servers that are internet facing, servers routinely undergo changes to maintain security or to adapt to changing business conditions and company needs. In the ecommerce scenario considered in the “Experiment” section of this report, the company obtains and deploys components that transcend the operating system distribution to satisfy business needs. This presents some interesting challenges on the Linux side. When a Linux distribution “releases” a new version of the operating system, what typically happens is that this distribution takes a snapshot of the many diverse core Linux components at a point in time. Consider the often discussed LAMP stack (Linux, Apache, MySQL and PHP) which contains the core components to run a web server. New versions of these individual components are released regularly by the package distribution sites but the distribution “snapshot” remains constant (aside for security and major functionality patches). The key issue is that core components are evolving outside of the operating system. From a 3rd party developer standpoint then, it is particularly challenging because each major Linux distribution bundles different versions of core components. Distribution decisions of which versions of components to package with the OS are usually driven by timing, component stability and the general adoption of those versions. Developers then face the tough decisions of which component versions they should build their applications against and when there is a mismatch between a dependency of the 3rd party software and the version of the component that is shipped with the OS distribution, the user is often forced to upgrade to the version supported by the component.

Patching and Milestone Upgrades

For SLES 8, all required and recommended security patches were applied to the system. The same criteria was applied for Windows patches. These patches were applied in 1 month increments to the system. On the SuSE side, during the one year period under study, patches were released for the core components from multiple sources spanning package developers, individual contributors in the open source community, individuals and corporations. In this analysis, we only consider those patches issued by the operating system vendor (Novell/SuSE). From an enterprise management standpoint, this is the most common scenario given that the chief benefits of using an enterprise Linux distribution is the compatibility testing done by that Linux vendor on patches and the support extended to administrators. By going outside this channel for patches, both benefits are forfeited. In the period from July 1st, 2004 to June 30th 2005 there were 187 patches that were applied to the system. Of these patches, 13 affected the kernel. While kernel patches did not require an immediate reboot during installation, the majority of them need a system restart to immunize the system against a specific vulnerability. In general, patch application on SuSE proceeded well and most patches installed without error or conflict. Beginning at Milestone 1 however, some upgraded components were out of support from SLES 8 and updates for those components had to be obtained from the package distribution sites. As of Milestone 1, MySQL patches were obtained from the MySQL distribution site and as of milestone 2, glibc and directly related packages were maintained through manually applying SLES 9 patches. 3rd party component installations were performed according to the installation procedures specified by those vendors.

Results

Table 1 below presents a summary of the results from the experimental trials. Both the SuSE Linux systems and the Windows systems were adapted to the evolving requirements of the ecommerce scenario described above. On the Windows side, administrators completed all tasks in an average of 857 minutes with administrator 1 finishing in 772 minutes, administrator 2 finishing in 678 minutes and administrator three taking 1123 minutes. On the SuSE Linux side, only one administrator, administrator 2, was able to successfully complete all tasks with a completion time of 1193 minutes. The other 2 Linux administrators both failed to complete the Milestone 2 installation within the stop-loss time for that task of 10 hours (600 minutes). Each milestone component was selected as identical on the Windows and Linux platform and represents the market leader in each space on both platforms (data mining, search and list management for milestones 1, 2 and 3 respectively; see Appendix 5 for details). In the case of Milestone 2 on Linux, the search component required a more recent version of the glibc module on Linux. Each of the three administrators pursued vastly different paths to resolve this dependency with Administrator 3 ending after 10 hours in an unbootable system and administrator 1 running into a cascading sea of broken dependencies and a non-functional solution after 10 hours.

Milestone	Windows				SuSE Linux			
	# of Patches	Average Install Time	Average Number of Dependency Failures	Percentage of Admins Completed	# of Patches	Average Install Time	Number of Dependency Failures	Percentage of Admins Completed
July 2004	7	42.67	0	100%	14	34.67	0	100%
August 2004	0	0	0	100%	12	32	0	100%
September 2004	0	0	0	100%	19	49.67	0	100%
M1 – Data Mining	-	72	0	100%		53.67	4	100%
M1 Conflict Resolution						95.33	0	100%
October 2004	6	49.33	0	100%	11	33.67	0	100%
November 2004	0	0	0	100%	19	29	0	100%
December 2004	4	56.33	0	100%	14	33	0	100%
M2 – Enhanced Search	-	62.33	0	100%	-	23	10	100%
M2 Conflict						398.33	0	33%

Resolution								
January 2005	3	20	0	100%	14	39.50	0	100%
February 2005	7	34.33	0	100%	22	54	0	100%
March 2005	0	0	0	100%	27	70	0	100%
M3 – List Manager	-	108.67	0	100%		60	0	100%
M3 Conflict Resolution						0	0	100%
April 2005	5	34.33	0	100%	17	52.50	0	100%
May 2005	1	14	0	100%	10	58.50	0	100%
June 2005	6	34.33	0	100%	8	40	0	100%
SQL Service Pack Upgrade (Windows Only)	-	61	0	100%	-		0	100%
Migration to new OS version	-	268.33	0	100%	-	292.50	0	100%
Totals	39	857.67⁷	0	-	187	1449.33⁷	14	-

Linux Results

In both Milestones 1 and 2 the challenge of an OS component upgrade was faced by the Linux administrators. This is indicative of fundamental model differences in Linux and Windows. Linux has pursued a model of high modularity and customizability while Microsoft has pursued a model which it calls “integrated innovation” where core OS components are tested together and tightly integrated.

A deeper look at the models can also help explain the deviation in experimental results. One of the key issues is that for Linux, key components evolve outside of the operating system itself. For example, when SuSE releases a new version of Linux, they take a snapshot of components at a point in time. Consider SuSE Linux Enterprise Server 8 (SLES 8) for example. When SLES 8 was released it shipped with (and supported) specific versions of MySQL, Apache and PHP. At the time of SLES 8’s release, all three components were behind the most current versions released by the package distributors. Other Linux distributions such as Red Hat, Mandrake and Debian also had supported released versions of these components with different versions of MySQL, Apache and PHP. This creates an interesting challenge for third party solution developers who must

⁷ “Total Average Time” is the sum of the individual average times per task. In the Linux case, one administrator was not able to complete all tasks so that administrator’s time is only “averaged in” on tasks that were completed.

build and test their components based on a finite number of version configurations of these components. A common practice for 3rd party developers is to use the most recent versions of OS components as a base for their solution. The result is often forced upgrades of OS components when the solution is installed. Consider for example the data mining tool installed during Milestone 1. In the Linux case, the component required an upgrade of the MySQL database component from version 3.23 to version 4.1. Upgrading MySQL means going outside of the supported OS configuration and obtaining the new version from the package distribution site. There is also confusion over *where* to obtain these components from. For example, one administrator went to the SuSE site for the newer version for MySQL that shipped with SLES 9. This version, however, was built against a more recent version of GLIBC which then lead him to the MySQL site for the MySQL upgrade built against SLES 8's existing GLIBC version.

At first blush, this high modularity and ability to tweak the operating system seems to be an asset but administrators have now dramatically increased the complexity and management burdens of the solution. First, consider that the primary benefits of an enterprise-class and packaged version of Linux such as SuSE and Red Hat are the compatibility testing performed on the shipped components, the support of the shipped operating system provided by the vendor, and the consolidation and testing of patches as vulnerabilities and bugs are discovered in those components. By installing a newer version of MySQL, the administrator has forfeited the majority of those benefits: the system is in an unsupported configuration, that configuration hasn't been tested by the OS vendor, and perhaps the biggest pain point, administrators must now look to multiple sources for patches. Additionally, confusion exists over which sub-flavor of MySQL 4.1 to install and from where, which leads to significantly divergent systems for the three Linux administrators involved in the experiment. The situation was exacerbated in Milestone 2 where the search component required a newer version of the GLIBC package than the one shipped with SLES 8. There are many ways to resolve such a dependency including trying to get the two versions to co-exist, upgrading the existing version, etc. Upgrading GLIBC directly (a route two administrators took), quickly leads to a cascading sea of dependency failures as other packages on the system that rely on the older version fail. One such failure came in the RPM package installer which left no direct route to re-install the old version of the GLIBC library. Figure 4 below illustrates how the three Linux solutions driven by the same requirements diverged over time as each administrator was faced with a wide range of options to resolve conflicts and dependencies.

Linux Solution – Evolution Over Time

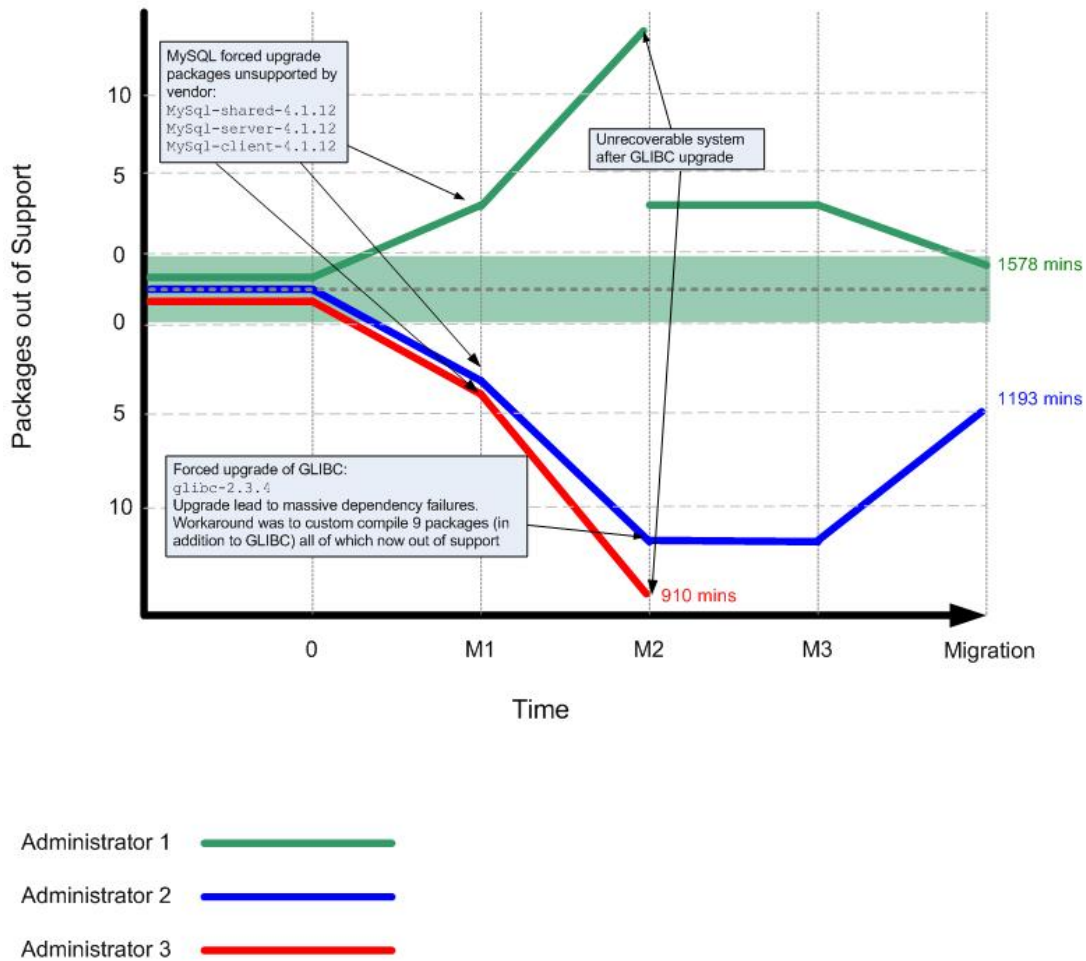


Figure 4 – Evolution of the Novell SuSE Linux Enterprise Server solution over time. Here we can see that the three administrators took vastly different routes to resolve conflicts beginning at M2 with the result for two of the administrators being an inoperable system. Of these two administrators, one was unable to move beyond milestone 2 in the allotted time and the second had their image “restored” to its pre milestone 2 state and continued the experiment without the milestone 2 component installed.

Two of the administrators encountered terminal failure at milestone 2. There are many potential routes to follow when faced with an upgrade of an integral component like GLIBC. One administrator proceeded to upgrade GLIBC to the required version by downloading the version available in that time period from the package distribution site (GNU). After installation, this administrator then suffered a sea of broken dependencies including the package installer RPM. After a day and a half of efforts to recover the system, the administrator’s only believed course of action was to install a clean version of SuSE on a new system and attempt to migrate data on the existing system over. A second administrator obtained a new version of GLIBC from the SLES 9 distribution

(which was available during the period of study) and installed it. The result was many broken applications and modules as was the case with the first administrator. A third administrator was eventually able to recover from the recursive dependencies by obtaining a newer version of the RPM installation tool, replacing individual files (the normal installation method would be to use the RPM tool itself which was broken) and then rebuilding the RPM database. While this administrator was able to get the system running to the point of passing acceptance tests, they faced the ongoing challenge of looking elsewhere for patches that affected the changed modules. For example, an upgrade to RPM obtained through Novell for SLES would now “undo” changes made by the administrator to get the system running and thus result in system downtime while the issue was resolved. Instead, extreme care had to be taken by the administrator to ensure that all patches installed would leave system functionality intact. While the RPM tool could still be used when forced for installation, it now could no longer identify dependencies on components which in turn breaks the YaST Online Update mechanism. During the course of the experiment, this translated into the administrator having to visit several package distribution sites directly to obtain patches compatible with the solution. The administrator was able to do this successfully but commented on the complexity involved in imparting this knowledge to other administrators who may work on such a machine in his absence in an enterprise setting. Figure 5 below illustrates the decisions made by these administrators during the trials.

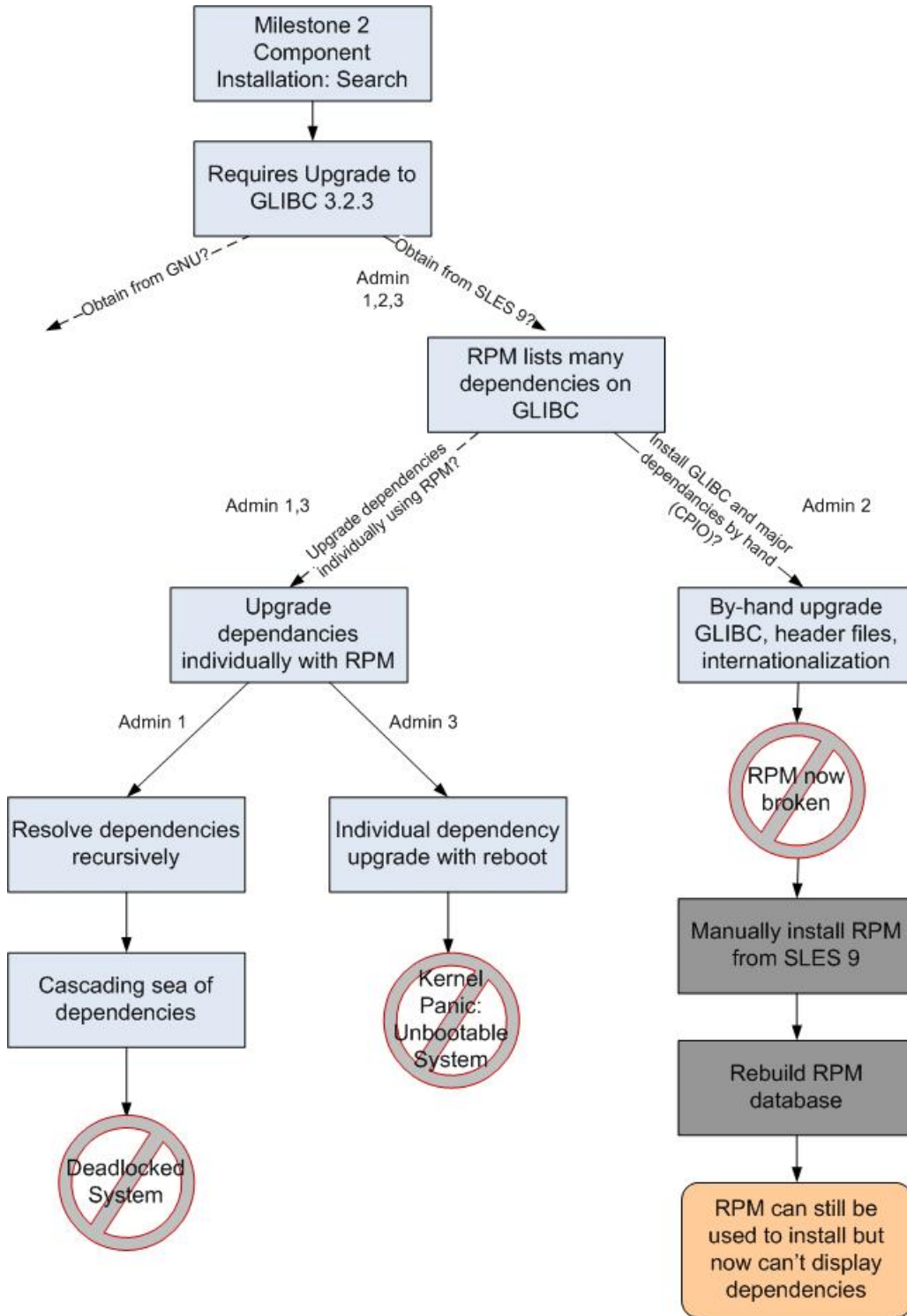


Figure 5 - Administrators took vastly different routes to resolve a dependency on a more recent version of GLIBC at Milestone 2. Two administrators ended in an unrecoverable system while the third was able to maintain a functional solution although abandoning some of the functionality of RPM until a migration/upgrade to SLES 9.

The patching process and the installation of the Milestone 3 (email list management) component for all three administrators was fairly straightforward and proceeded without incident. The systems remained in very different states and configurations however until the final migration to SLES 9. The migration essentially allowed the solutions to be normalized, and served to reset the configuration deviations that surfaced during the experiment. Reinstallation of the old components on the new version of SLES 9 went well and moving to SLES 9 essentially allowed OS components to “catch up” with the versions that the 3rd party applications were built against. This speaks well of the migration scenario for SuSE and indicates that migration to newer releases may be warranted to ebb the swell of complexity that builds over time. It also has some interesting implications for the enterprise. 3rd party compatibility would seem to favor Linux distributions that release fairly rapidly which must be balanced against the vendor’s need to minimize the number of simultaneously supported versions. This underscores the tension between the rapidity of change in community driven projects and business time frames; IT administrators may have to bear the burden of reconciling the two.

Analysis Note – Change Management

In discussing these results with analysts, one interesting point that was raised is the potential impact of strict change management policies on the Linux systems. In our experiment, administrators were given free reign to resolve conflicts and dependency problems. Many of the breakages that resulted were traced back to the highly divergent and sometimes risky paths taken by the administrators involved in the experiment. Some of this variability would have been mitigated with a change management policy which mandated, for example, that no unsupported components be added to the system. This policy would have left the machines in a more homogeneous state but would have also prohibited any new 3rd party product from being installed that required an upgrade of MySQL, Apache, or any component shipped to a version unsupported by the OS distribution. Such a policy severely limits the choices available from 3rd party vendors however given that many of them build to the latest versions of these components instead of supporting multiple builds to accommodate each distribution’s Linux “snapshot” of components. Keeping complexity, support and functionality in balance is at the heart of change management, IT departments must carefully look at the implications of any policy they adopt.

Windows Results

During the experiment, administrators followed a fairly homogeneous path through both the installation of patches and component installation. There were no component incompatibilities that surfaced during execution. Issues encountered were limited to

resetting database table types (case insensitive to case sensitive), rebooting the machine after the majority of patches, and some configuration issues (conflicting port numbers) during milestone 3. Each administrator was able to complete all installation tasks. Third party customer support was called by one of the administrators to resolve a configuration issue with one of the installed products (milestone 3) and the matter was then resolved. While the number of patches installed under the Windows solution was smaller, the proportion of patches requiring reboots was larger. This was mitigated by the fact that many reboots could be postponed until all patches in the month period were installed. There were also few potentially divergent decisions that needed to be made by administrators and the systems were in very similar states after each milestone was complete as illustrated by Figure 6 below.

Windows Solution – Evolution Over Time

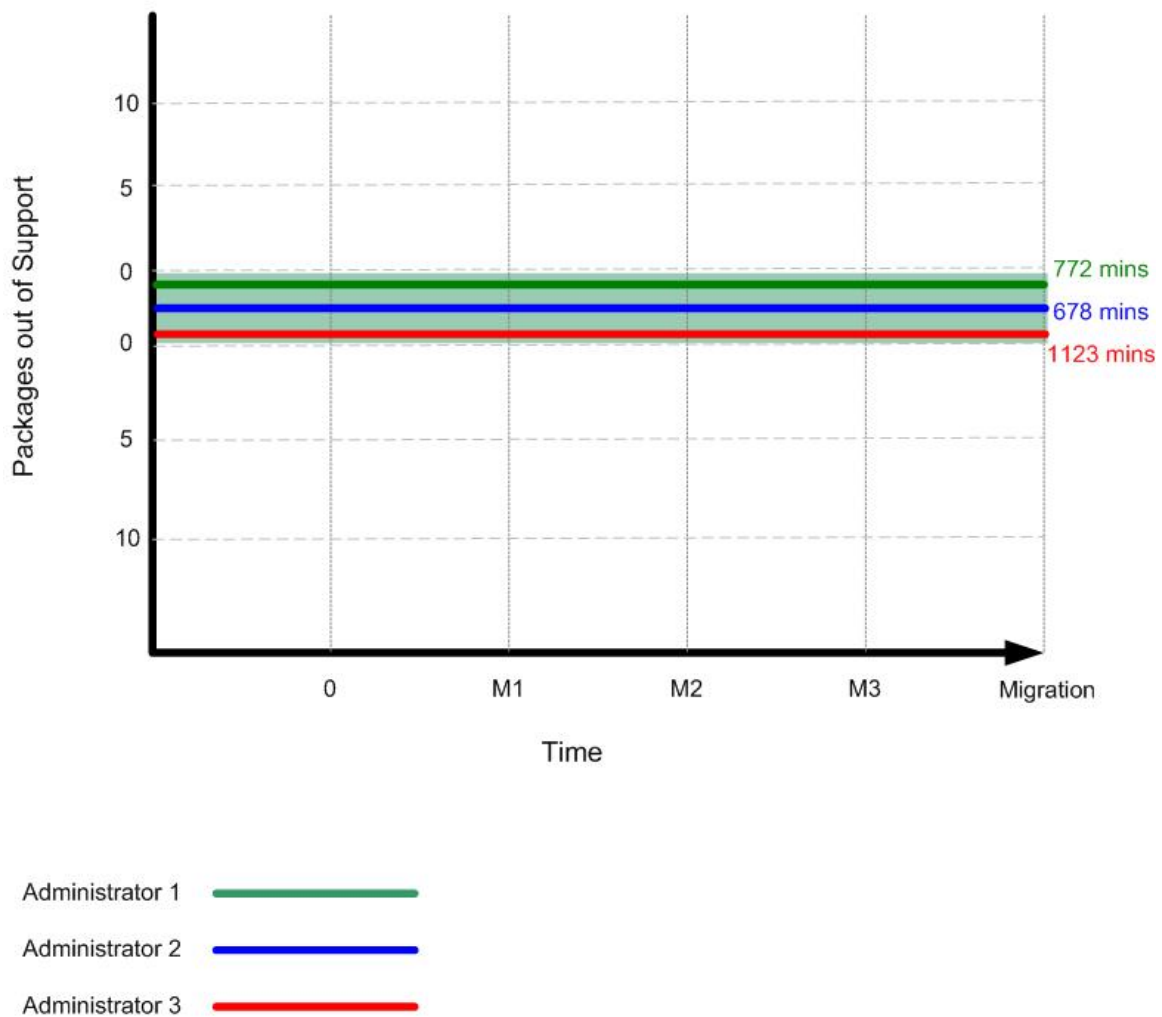


Figure 6 - Evolution of the Windows solution over time. All system administrators for the windows solution followed a fairly homogeneous path through the system. Aside from configuration issues

with one administrator during Milestone 3, the process was fairly consistent among administrators with a fully supported system.

The patching process brought out several interesting model differences between the two platforms. For the Windows solution, patches for the platform and SQL Server were obtained from Microsoft and were readily visible through tools like Windows update. Beginning at Milestone 1, patching the Linux solution required administrators to go outside of Novell and directly to package distribution sites for unsupported, upgraded components. After the Milestone 2 update, RPM was unable to properly identify package dependencies which meant that updates needed to be installed manually and the YaST Online Update tool would no longer be functional. Figure 7 illustrates the patching evolution of the two systems.

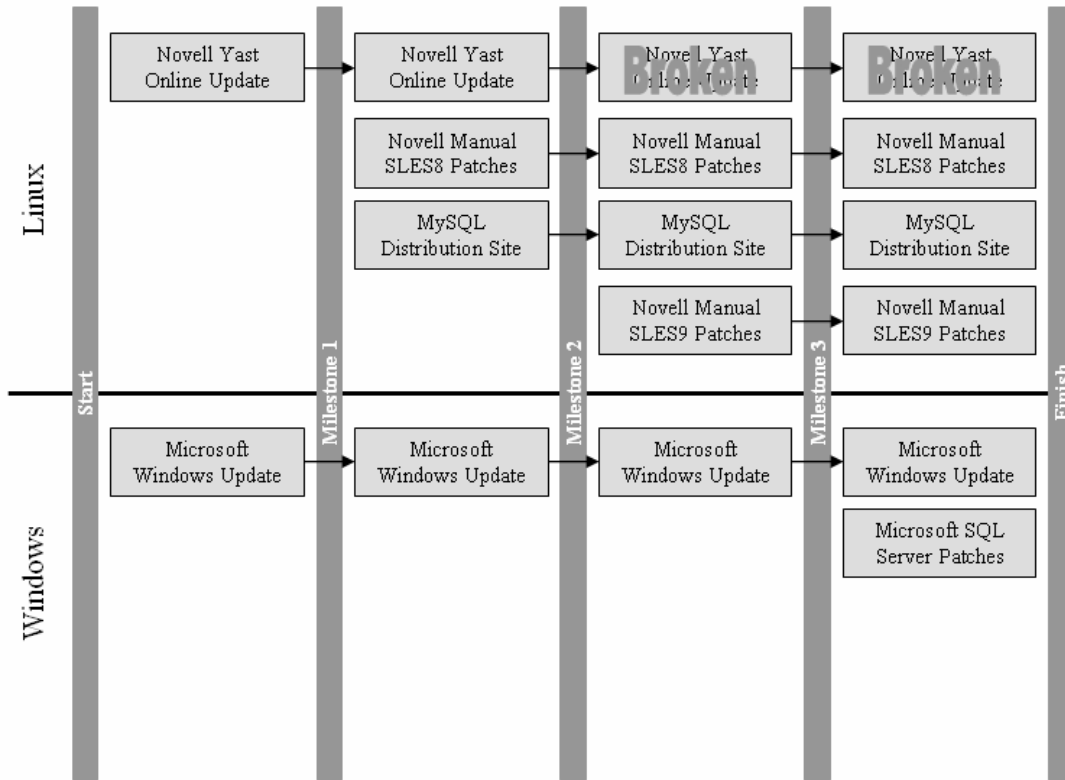


Figure 7 - As the business requirements evolved, Linux administrators needed to go outside of normal support channels to obtain patches on upgraded components. This graph shows this evolution with respect to the milestones.

Analysis Note - Migration

An interesting pattern in the results is that all breakages occurred before migration. In our discussions with industry analysts, one point of discussion has been decision to include a platform migration as part of the one year experiment period. Transitioning to the latest operating system version is often driven by supportability, performance and maintenance needs and it is an eventuality for most server roles. Based on our discussions with customers, it is thus important to understand potential IT pain associated with it. While the results of our study included migration at the end of our experiment, the breakages, dependency failures and incompatibilities all occurred before that final transition was made—we believe this in itself is valuable information for customers and speaks to our intention to provide a more holistic, customer scenario rather than version-based view of reliability. Table 1 provides granular detail on when failures occurred during the assessment.

Conclusions

Businesses and customers care about *solutions* being available to them when needed. They care about reliability in broader terms than kernels and dlls; they care about the pain caused when a *solution* fails. In this paper we present a new model for measuring and evaluating *solution reliability*, one that looks at a system in deployed environments as opposed to components in isolation. Any true measure or predictor of downtime must consider the system as it changes over time which includes updates to maintain security and functionality, but also the evolving requirements of the enterprise. When considering the results here, it is important to understand their scope and applicability. This study pertains to enterprise solutions that have patching needs and changing demands. It is not appropriate to draw any conclusions for embedded devices or systems that are completely static and rarely, if ever, require system change.

In the “Experiment” section of this study we considered one evolution scenario: an ecommerce company that must move its site from basic purchasing to a personalized, history-driven portal. These requirements were created with the help and input of some of the world’s largest e-retailers and the experiment follows the life of two systems, one Windows based and one SuSE Linux-based over a simulated one year period. While the sample size of administrators was too small to provide conclusive statistical comparisons, the results highlight some fundamental differences in the Linux and Windows models. System administrators on Windows followed a fairly predictable and consistent installation path for both patches and add-on components while Linux administrators all followed vastly different paths to both add functionality and resolve conflicts. One of the most striking outcomes of the experiment was the diversity of solutions under Linux. Multiple pathways can be both an asset and a weakness; it can allow highly skilled administrators to solve problems using greatly varied approaches but it also leads to the “personalization” of systems which can make issues like administrator substitution

problematic. The Linux solutions also quickly went out of support from the both the distribution vendor and 3rd party solution vendors as individual components (such as MySQL) were upgraded to meet 3rd party solution needs.

For the administrator trials, our choice of requirements was based on a common evolutionary scenario, one vetted and confirmed by analysts and corporations in the ecommerce space. Our choice of 3rd party components was based on market leadership in the enterprise markets. The experiment follows only one scenario with a small set of administrators, but the results highlight some key model differences in Windows and Linux. We strongly encourage others to repeat this analysis under other requirement sets and using a variety of 3rd party products. The industry needs metrics on reliability that are true predictors of organizational pain. Our hope is that the methodology presented here will serve as a base that others can expand on and adapt to their own business needs.

Appendix 1 – Acceptance Tests

Acceptance Tests

Basic Ecommerce Application

Please validate that each task is performed successfully.

- Create a new account; call it TestAcct_# (where # is the task number). If you need to repeat this task call the new account TestAcct_#_2 etc.
- Exit Browser
- Navigate to site and login
- Search for an item that doesn't exist – look for 0 results found
- Search for an item that does exist, verify that it was returned
- Add three new items to cart
- Purchase these items (make sure you are going over SSL for credit card data)
- Add a review for one of the purchased items
- Close browser
- Log back in and make sure credit card information is retained

Note failures at any stage or feel free to add additional comments:

Appendix 2 – Milestone/Upgrade Questionnaire

Questionnaire

Task Number _____

Task Time Period _____

Task Description _____

Administrator _____

OS _____

As you apply the updates, there will be several elements you'll be asked to record. Please read through the following checklist before you begin:

Things to check before you start installing:

- Make sure that logging is enabled for the system logs.
- Capture screenshots of any error messages or conflict messages that come up and save them.
- Before package installation (if on Linux) do an "RPM -qa" and save the results to a file marked beginning_of_phase_#_of_12 (where # corresponds to the stage you are at).

Record the size of the files that needed to be downloaded (e.g. patch/package sizes)

Questions

Record the time you started applying the updates (this time begins after you have downloaded the packages and extends through unpacking, installation and problem resolution.

Start Time _____

Please log breaks during the process

Time out _____ Time test resumed _____

Time out _____ Time test resumed _____

Time out _____ Time test resumed _____

Time out _____ Time test resumed _____

End Time (this is to be recorded **after** all acceptance tests pass) _____

Record the number of patches that needed to be applied. Note patch ID and direct impact of patch (conflict, reboot, recompile, ...). Please feel free to use extra sheets of paper.

Download/Search time interval of **additional components not included in the update**

_____ mins

Was a reboot required during this stage? Yes No

How many reboots? _____

Comments on reboots

Was compilation required for any of the modules (e.g. only SRPMs available)? Yes No

Comments on compilation?

Were there any components that needed to be updated on the system or any dependencies broken? Please elaborate on your answer:

What dependencies needed to be upgraded? From which versions to which versions?

Did the upgraded packages come with the component/patch set or did they have to be downloaded separately? Explain.

Were required upgrades available from the OS vendor or did you have to go to the package site (Vendor's site vs. component site for example)?

Did the component/patch deployment tool clearly indicate what dependencies needed to be updated or did the installation simply fail? Please be detailed in your answer.

Did the system pass acceptance tests on the first go around? If no, please explain.

If there were any conflicts, upgrades or errors please elaborate on what the problem was, what steps you took towards resolution and how the problem was finally resolved:

On exiting:

- Save system event log files
- After patch installation (if on Linux) do an "RPM -qa" and save the results to a file marked end_of_phase_#_of_12 (where # corresponds to the stage you are at).

Appendix 3 – Recruiting Questionnaire for Windows Administrators

Questionnaire – Windows Administration

Name:

Contact Phone Number:

Email address:

General IT Background:

How many years of system administration experience do you have?
4-5 minimal

On which systems and what duration for each?
Looking for at least 4-5 years on Windows

List the certifications you have

MCSE highly desirable.

Other interesting certifications:

MS Certified Solutions Developer

MS Certified Professional

MS Certified System Administrator

MS Certified Architect Program

Windows-Specific:

How many years of experience do you have as a Windows administrator?
Looking for at least 3 years on Windows

How many years with Windows 2000 Server?
Looking for at least 2 years

How many years with Windows Server 2003?
Looking for at least 1 year

How many machines have you migrated from Windows 2000 server to Windows Server 2003?
Looking for 20

Describe your experience in applying system upgrades (hot fixes, service packs, component upgrades)

Typically, how many servers were administered?

We are looking for on the order of 20 servers

How many years experience do you have administering a eCommerce web solution?

Looking for at least 2 years here

Years directly with IIS 5 or 6?

Looking for at least 2 years here

How many years experience do you have administering SQL Server 2000?

Looking for at least 1 year here

Classify your experience with Windows Update (minimal, moderate, extensive)?

Appendix 4 – Recruiting Questionnaire for Linux Administrators

Questionnaire – Linux Administration

Name:

Contact Phone Number:

Email address:

General IT Background:

How many years of system administration experience do you have?

3-4 would be minimal criteria here

On which systems and what duration for each?

Looking for at least 2 years on Linux

List the certifications you have

CompTIA Linux+, Novell Certified Linux Engineer, Linux Professional Institute L1, Red Hat Certified Engineer (RHCE) would all be valuable here

Linux-Specific:

How many years of experience do you have as a Linux administrator?

Looking for at least 3 years on any distro

How many years with any SuSE Linux products?

Looking for at least 1.5 years on SuSE

How many years with SuSE Linux Enterprise Server?

Looking for at least 1 year on SLES

Which versions, and how long with each?

Looking for at least 1 year on SLES 8 and half a year on SLES 9

How many machines have you migrated from SuSE Linux Enterprise Server 8 to 9?

Looking for 20

Classify your experience with YaST and YUM (minimal, moderate, extensive)?

Describe your experience in applying system upgrades (hot fixes, service packs, component upgrades).

Typically, how many servers were administered?

We are looking for on the order of 20 servers

How many years experience do you have administering a web solution?

Looking for at least 2 years here

Years directly with Apache?

Looking for at least 2 years here

How many years experience do you have administering MySQL?

Looking for at least 2 years here

Appendix 5 – 3rd Party Component Selection

During the experimental trials, 3rd party best-of-breed components were chose to satisfy the needs of the solution. Our criteria for selection of components were:

- Support on both Windows and Linux
- Strong and established base of enterprise customers

The purpose of the experiment is to analyze how operating system components evolve over time and the impact of this evolution on solution reliability. It is important, however, to keep in mind that the 3rd party components were chosen purely based on market leadership in their respective fields. The specific 3rd party vendors are not disclosed because the focus of the study is the methodology and not a specific component. Below is a sampling of large customers using each of the solutions selected.

Milestone1 Component (Data Mining)

American Cancer Society	Federal Aviation Administration (FAA)	Liz Claiborne	Progressive Insurance
American Express	Federal Trade Commission	Lockheed Martin	Qwest Wireless
American Red Cross	FleetBoston	L'Oreal USA	Raytheon
AOL Time Warner	Ford Motor Company	Magazines.com, Inc.	Samsung
Applied Materials	GE Medical Systems	Mass Mutual Life Insurance	Seattle Daily Journal
Bayer Corporation	GEICO Direct	Microsoft Corporation	Sony Corporation
Best Buy	Goldman Sachs	Mitsubishi Motors North America	Sprint Corporation
Blue Cross Blue Shield	H&R Block Financial Advisors	Morgan Stanley	Sun Microsystems
BMW Manufacturing Corporation	Harvard Medical School	MotherNature.com	Sunoco
C.N.A. (Cigna) Insurance	Hewlett Packard	Motorola, Inc.	The Walt Disney Company
Campbell Soup	Hitachi, Ltd	Museum of Modern Art (New York)	Time, Inc.
Capital One	Honda Motor Co., Ltd	NASA	T-Mobile
Citi Group	Hoover's Online	National Public Radio (NPR)	Toshiba
CNN	Hotwire.com	Network World	Turner Broadcasting Company
Columbia University	ING	New England Journal of Medicine	U.S. Army
Cornell University	Isuzu Motors America	New Jersey Devils	U.S. Navy
CVS	JC Penney	Nokia Group Finland	Verizon
Daimler Chrysler	Johnson & Johnson	Papa John's	Virgin Mobile

Deutsche Bank	Lands' End Direct Merchants (U.K.)	Pfizer, Inc.	Xerox Corporation
Ernst & Young	Lehman Brothers	Pharmology.com	Yale University

Milestone 2 Component (Search)

American Express	GlaxoSmithKline	Princeton University	The E.W. Scripps Company
American Hospital Association	Home Depot, Inc.	Prudential Securities	The Financial Times
American Red Cross	HP	QUALCOMM	The Iams Company
Arizona State University	International Data Group (IDG)	Raytheon	The U.S. Air Force
AXA Financial Inc.	Iowa State University	Reuters	Thomas Cook
Banco de Mexico	Irish Revenue Commissioners	Revlon	Timex
Boeing	Johns Hopkins Institutions	SAIC	Toys R Us
Bristol-Myers Squibb	Kodak	Salvation Army	U.S. Army
C-SPAN	KPMG LLP	SAP	U.S. Federal Aviation Administration (FAA)
Central Statistics Office Ireland	LG Electronics	Segue Software	UBS Warburg
CERN	MCI	Southern California Permanente Medical Group	UK National Lottery
City of Boston	META Group	Standard Life	United Way
Colgate-Palmolive Company	Middlesex University	Stanford Graduate School of Business	United Way of Roanoke Valley
Computer Sciences Corporation	MillerBrewing.com	State of California	Universal Studios Hollywood
Compuware	Ministry of Agriculture, Fisheries and Food	State of Georgia	UPS
DaimlerChrysler	New Jersey Transit	Superpages.com	Vodafone
Dartmouth College	Nissan	Sybase	Wachovia
E-Trade	Nortel	Texas Department of Transportation	WeddingChannel.com
Ford Motor Company	Ohio State University Comprehensive Cancer Center	Texas Education Agency	Weight Watchers
French Ministry of Defense	Oracle corporation	Thailand National Statistical Office (NSO)	Xerox

Milestone 3 Component (List Management)

ABC News	CMP Media	Library of Congress	Sandia National Labs
Abercrombie & Fitch	CNN	Lockheed-Martin	Scripps
About.com	Delta Airlines	Lucent Technologies	Siemens
AIIESEC	Dow Jones	Massachusetts Institute of Technology	Sony Electronics
America Online	Duke University	McGill University	State of Mississippi
American Bar Association	easyJet	McGraw-Hill Publishing	SUN Microsystems
American Medical Association	Ericsson	MCI	Texas Instruments
American Red Cross	Federal Aviation Administration	Microsoft	Time Warner
Arizona State University	Ford Motor Company	Mitre	Travelweb
AT&T	FOX News	Motorola	UNISYS
Bank One	Georgetown University	MSNBC	United Way
BBC	Harvard University	Nabisco	US Department of Commerce
British Airways	Hewlett-Packard	NASA	US Department of Justice
British Telecom	Hilton International	National Academy of Sciences	US House of Representatives
Brown University	IDG	National Endowment for the Arts	US Patent and Trademark Office
Canon	IKEA	New York Times	Victoria's Secret
Carnegie Mellon University	Intel	Nortel	Wal-Mart
CBS News	Knight-Ridder	Oxford University Press	Wells Fargo Bank
Charles Schwab	Legg Mason	PricewaterhouseCoopers	World Bank
Citicorp	Lexmark	Samsung	World Health Organization