App Compat Recommendations

Windows as a Service

Prepared by

Chris Jackson

Sr. Architect

Revision and Signoff Sheet

Change Record

Date	Author	Version	Change Reference
6/27/2016	Chris Jackson	1	Initial draft for review/discussion

Reviewers

Name	Version Approved	Position	Date

Table of Contents

1 Overview

Windows 10 introduces a host of features, both new and familiar. Customer excitement has been high, and enterprise deployments have been accelerating. Microsoft has also changed the engineering approach to how Windows is delivered. Rather than releasing Windows once every 3 to 5 years, and requiring significant upgrade projects to adopt a new version (typically 12 – 18 months or more), we are accelerating time to market of our innovations. By releasing Windows as a Service, new features that could benefit you today become available as soon as they are completed. This increases the amount of value you are able to derive from the innovation by extending its timeline and relevance.

Of course, deploying Windows as a Service requires new, more agile processes, while still managing risk. While some aspects of deployment are operational, the biggest gate historically has been ensuring that the updates to Windows don't negatively impact application compatibility (app compat). This paper discusses both the onramp to Windows 10, as well as the agile processes to continue to deploy updated versions of Windows 10. It is designed with the modern enterprise in mind, to provide you with the guidance Microsoft has collected through our interactions with enterprise customers around the world on what other customers have used to successfully transition to, and thrive within, a Windows 10 / Windows as a Service environment.

2 Pragmatic App Compat Approach

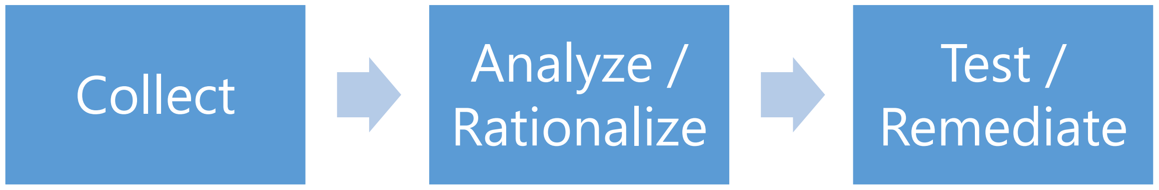
App Compat is all about management of risk. In previous operating system migrations, it was one of the largest sources of risk. Organizations strive to understand and manage that risk.

2.1 Managing High Risk Transitions

For most organizations, moving from Windows XP to Windows 7 was their last experience in migrating to a new operating system, and in this transition, we saw an average of 21% of applications require some form of touch. While many touches were simple (for example, modifying the application installer to remove a check for the version of Windows you are running), the challenge was nonetheless often hard because of:

- The sheer volume of applications – many organizations found that they had thousands, and sometimes tens of thousands of applications in the environment
- The occasional complexity of remediation – often, an organization would (sensibly) begin with its largest and most complex applications, and project that the complexity of solving that problem might be representative of the complexity of the rest
- The support conundrum – determining whether or not an app worked sometimes didn't matter when support was required for the capability but not provided in that version of the app

Many customers chose the approach that Microsoft recommended:



This approach required a significant project effort, discovering, prioritizing, and testing applications across an environment. But, given the risk profile, it tended to work better than other alternatives.

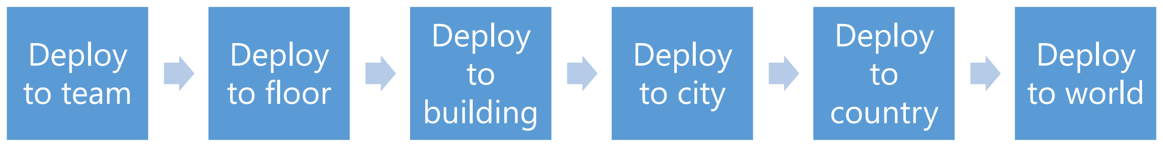
Other customers chose a different approach: they invested less in managing risk, and often ended up struggling to cope with any remaining compatibility issues reactively. Consequently, Microsoft has heard many statements such as "I'm not making that mistake again, next time I'm going to be extremely thorough."

2.2 Managing Low Risk Transitions

It's important to note and observe that not all operating system transitions experience the same risk exposure. In fact, most organizations have even more experience dealing with lower risk transitions, as they get to practice this transition once a month: Windows Updates.

While generally focused on patching security issues, Windows Updates are nonetheless modifying Windows components, and have a small, but non-zero risk of app compat impact. But since the risk is sufficiently small, most organizations don't conduct exhaustive testing. Given that the typical app compat assessment project in a large organization would often take 12-18 months, it's hard to fit that into a monthly cadence!

Instead, a common approach to testing for Windows Updates would often look like this:

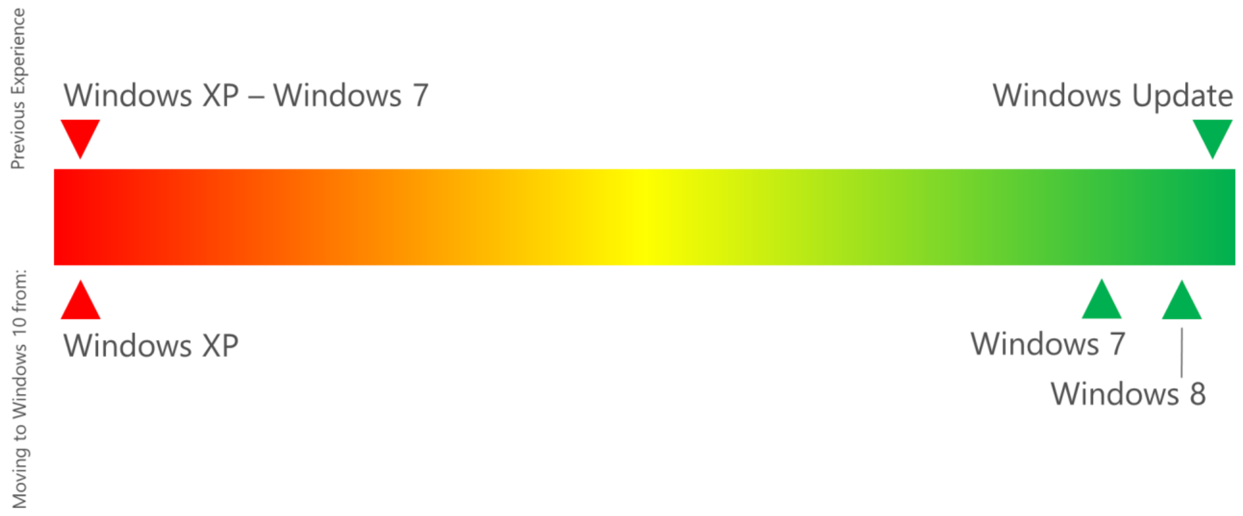


This approach was significantly more pilot-driven than lab-driven, and significantly accelerates the deployment of critical security patches, managing the risk by containing the users impacted to a number more easily rolled back in the event of a compatibility issue.

2.3 Building a Risk Continuum

Given the different approaches presented, what Microsoft recommends is disrupting existing approaches. We believe it no longer makes sense to have two discrete approaches to managing app compat: the Upgrade and Patch approach. This is an artificial separation, after all – patches are just a subset up upgrades. The introduction of a new box, or of a new version number, doesn't add risk – only the introduction of code which impacts app compatibility adds risk, and this can happen anywhere (or nowhere).

Here, for example, is the risk continuum we have observed for transitions to Windows 10 so far:



For customers who are moving to Windows 10 from Windows XP, the risk is very similar to the risk of moving to Windows 7 from Windows XP. In fact, for some organizations, it can be even higher; most Windows XP devices today are running that OS not because of unknown compatibility impact, but because of known incompatibilities.

For customers who are moving from Windows 7, working with early adopters, we have observed an average failure rate of around 2%. This is dominated by hard coded version checks for Windows 7. Note that this number is still early and may not be representative of your findings, but is nonetheless instructive when it comes to strategy.

For customers who are moving from Windows 8, working with early adopters, we have observed an average failure rate of a fraction of a percent. This difference from Windows 7 is explained by the fact that, beginning with Windows 8.1, we began lying about the version of Windows by default. Again, these are early findings with relatively low sample sizes, but should nonetheless be instructive in developing smart risk management decisions.

Given this continuum of risk, what we want to establish is a more systematic way to approach management of that risk. Before, we presented only two scenarios – how can we broaden this to discrete intermediate steps? To do this, we will next describe our taxonomy for both assessing application risk and for testing applications.

2.4 Assessing Application Risk

Defining and assessing application risk is often not trivial. If you ask most business owners to identify the criticality of the software they are using, we have found that the most typical result is to discover that 85% of all software is defined as business critical.

We believe this is mostly driven by a lack of agreed upon criteria for software criticality, combined with the abstract fear that, if not defined as critical enough, either testing or future funding is potentially at risk. Regardless of the motivation, the results are predictable: by defining everything as important, you are defining nothing as important. We prefer to differentially invest in software based on risk.

We have successfully used this taxonomy, along with a recommended distribution of software within this taxonomy, to help drive a deeper understanding of where app compat risk exists. It’s important to note that we’re not trying to define business impact – lower ranking here doesn’t mean the app is less important, it merely means that it’s less at risk for app compat failure!

App Compat Risk	Characteristics	Typical Distribution
Critical	Typically, complex custom development or highly customized commercial software. App failure can lead to loss of life or irreparable organizational harm. No acceptable workarounds for failure are identified.	2%
High	Typically, complex custom development or highly customized commercial software. App failure can lead to significant loss of productivity or ability to execute organizational mission. Workarounds can take days or weeks to implement, and/or come at a significant cost.	18%
Medium	Typically, commercial software that is current or a recent version, and is commonly used at other organizations using the target platform; or custom development following straightforward patterns. App failure can be addressed with reasonable workarounds that take minutes to hours to implement and which use standard helpdesk procedures.	50%
Low	Typically, commercial software that is up-to-date and supported on the OS you are moving to, or is custom development that was created and/or tested on the target platform. Vendor is accountable for remediating compatibility issues, or organizational mission can be accomplished without the software on a temporary basis. App failure can be addressed with simple workarounds through standard helpdesk procedures.	30%

Leveraging this taxonomy, we have seen customers driving much more consistent behavior and smart investment against the risk of app compat impact in a migration.

2.5 Defining Testing Approaches

We now need to define the testing approaches we might leverage to help to manage that risk. Being able to define a variety of means to perform testing is important to be able to invest commensurately with risk.

This is the most typical taxonomy we see used to define testing approaches:

Approach	Description
Regression	Users or business owners perform end to end testing to validate all critical scenarios driven by the application. App must pass all tests to be considered compatible. Typical accuracy rate 99%.
Smoke	Users or business owners come to labs or run on their own devices and test a few scenarios to get a gut feel of whether the software appears to be working. Typical accuracy rate 95%.
Automated	Application is run through test automation or static analysis (where available) to predict compatibility based on attributes of the application. Typical accuracy rate 20% - 60%.
Pilot	Application is deployed to pilot users or group technical coordinators to integrate into daily workload, while backup devices are made available in the event of application failure. Typical accuracy rate 85%.
Reactive	Application is deployed to end users on the new platform, with guidance and help desk scripts for responding to production failures that leverage existing helpdesk procedures. Typical accuracy rate 99%.

Note that, while Pilot and Reactive have high accuracy rates, they find app compat issues post-deployment, while the other 3 approaches have an objective of discovering issues prior to deployment. So, it's important to work that into your strategy based on where you can sustain temporary outages, whether in pilot or production scenarios.

Note on Automated Testing

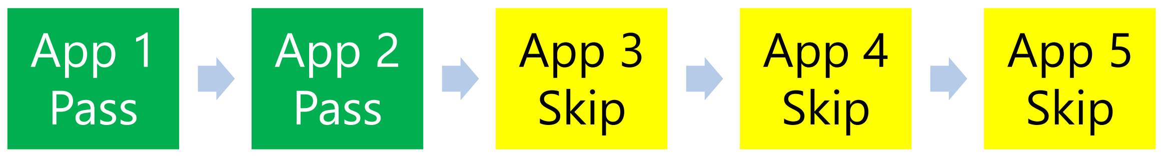
Automated testing can be performed either at runtime or statically (analyzing a binary at rest). This is a testing approach that really came into prominence in Windows XP retirements, as it provided good accuracy for relatively low cost and effort. However, the approach is not universally effective – it's only statistically predictive, and important to baseline against each migration's risks. The ability to cheaply obtain assessments that are

> only accurate 20% of the time is often not helpful in accelerating migrations or managing risks. Baseline the current statistical accuracy of predictions, and keep up to date, as tools are always changing and improving.

With these tools in your arsenal, we can now begin to align tools to the spectrum of risk.

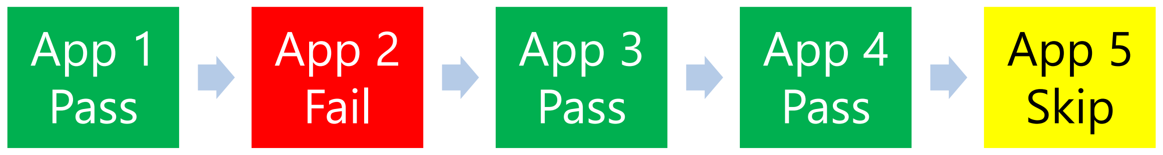
2.6 Grouping Applications by Risk

In the next section, we'll be looking at assigning testing approaches. You'll note that we differently treat applications with the same risk category. This is an approach that we pioneered within Microsoft IT – an approach that groups applications together based on similarities predictive of outcomes. Applications that are written with the same tools by similar teams are more likely to behave similarly regarding compatibility. Thus, if we have 5 applications, we may choose to test the first 2, and then test more based on outcomes. For example:



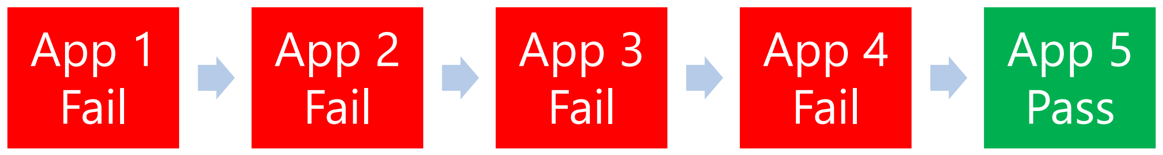
Here, the first couple of apps passed, so we wouldn't bother to test the rest using the defined testing approach – the apparent risk doesn't justify it.

If, however, we found that one of the applications had an app compat failure, we might test more:



Here, we tested more, but when we didn't discover more widespread application failures, we again stopped, as the measured risk didn't justify the investment.

A final example:



Here, we saw rather widespread failure, which could be attributed to poorly defined application standards which were consistently applied. But, again, the sample we put first helped us to detect this risk and manage it appropriately.

We often refer to this approach as the “Canary in a Coal Mine” approach.

2.7 Assigning Testing Approaches

With our risk taxonomies now clearly defined, we can start to assign testing approaches to each of our risk profiles.

Here is a sample of one assignment grid, where we were specifically attempting to manage the risk of applications we were **migrating from Windows XP to Windows 10**:

App Compat Risk	Regression	Smoke	Automated	Pilot	Reactive
Critical	100%				
High	20%	30%	50%		
Medium		10%	20%	30%	40%
Low			10%	20%	70%

We built a very similar grid, but instead shifted to accommodate the different risk of **migrating from Windows 7 to Windows 10**:

App Compat Risk	Regression	Smoke	Automated	Pilot	Reactive
Critical	100%				
High		20%		70%	10%
Medium		5%		35%	60%
Low				10%	90%

A couple of things to note in these distributions.

First, our critical apps get 100% regression tested in both scenarios. The assumption here is that there is truly no room for failure. These are systems that have a direct impact on lives, financial viability of the company, and so forth. We suggested very small percentages in our guidance on categorizing, and this simply re-emphasizes that guidance. If you're not willing to invest in full regression testing for even a low-risk migration, then consider not labelling this a critical app.

Second, we push progressively more down to Reactive testing as the risk declines. This means that we have to have a great plan in place for providing reactive support! If you push an application to a user, they need to have a means to raise issues they may encounter that minimizes friction.

Third, today we are not currently recommending automated testing for Windows 7 to Windows 10. As discussed earlier, this is purely based on statistics. Today's tools simply don't justify the investment in tools based on their accuracy. But, as previously stated, we recommend continuing to observe the evolving space to determine if this begins to make more sense in the future. At the same time, we do recommend considering these tools when migrating from Windows XP, where they have been proven relevant and effective.

2.8 Test and Verify

We have built out a risk profile and responses to this. As with all predictions, we recommend tracking your outcomes to determine how closely your results align with industry averages. If you find that you have a higher than normal failure rate, you probably want to test even more, promoting more apps into a higher touch testing scenario. Similarly, if you are finding a lower than normal failure rate, you may want to do exactly the opposite. But following these approaches helps you define a spectrum of compatibility testing which we have found helps organizations become significantly more agile and respond to what are truly very different risk profiles in a pragmatic and efficient way.

Additionally, we have found greater success when incorporating the following into the testing approach:

2.8.1 Build a Centralized Testing Team

There is often a strong temptation to crowd source, and broadly distribute testing responsibilities across a large number of teams. The idea that a large number of hands can make the problem smaller is extremely tempting. However, in observing our most efficient testing organizations, one commonality is that they were able to test their most critical apps much more efficiently with a centralized team than other organizations who distributed this responsibility.

2.8.2 Perform scenario-based testing

Regression testing, by definition, covers all business scenarios. For smoke testing and pilot testing, however, there is a huge variation in accuracy between customers who merely ensure that you can log in and the app "looks good", and those who define "golden scenarios" and specifically target those scenarios in their testing and usage.

2.8.3 Leverage Virtual Machines

Microsoft's own IT department says this best: "I cannot express how important VMs are to our success." Leveraging virtual machines for regression and smoke testing has many advantages, including:

- Setup to Test time is minimized
- Hardware variables are removed from results
- Easy to provide access to developers when issues are found

3 Migrating to Windows 10

Microsoft's overall recommendation is to follow the Pragmatic App Compat Approach, which we have found dynamically scales to fit the various scenarios and risk profiles that customers have encountered – which is what drives the recommendation.

In the context of a Windows 10 migration specifically, there really are two phases to prepare for: the initial migration, and the subsequent servicing.

The transition of Windows from a periodic major release to much more frequent updates, often called Windows as a Service, is better managed when you come prepared to handle the follow-on changes in process. This is truly the end of once- or twice-a-decade big migration projects. But, depending on where you are, the transition into Windows 10 may vary in size, and might be your final "big project" for app compat in readiness for deployment.

3.1 Technical Debt

The single biggest determinant in the amount of effort you need to invest in app readiness for a Windows 10 migration is whether the previous migration took on technical debt.

In Section 2.3, we looked at the continuum of expected application risk based on statistical averages. It's important to note that these averages assume that your organization has made the most typical choices when it comes to your previous migrations. We know, however, that some customers have chosen to postpone some aspects of modernization, whether due to ecosystem challenges in their industry or simply time or budgetary constraints. We refer to these decisions as technical debt, and some of it is debt that should be paid back in the context of a Windows 10 migration.

The two most common areas of technical debt we have seen customers take are postponing the transition to standard users, and postponing the transition to 64-bit Windows, though any similar deviation from well-known and proven solutions would similarly qualify.

For details on Microsoft's recommendation to favor default configurations, see: https://blogs.technet.microsoft.com/fdcc/2010/10/06/sticking-with-well-known-and-proven-solutions/

3.1.1 Enabling UAC and Transitioning to Standard Users

Disabling UAC on Windows 10 puts you into an untested and unsupported configuration. It also blocks your ability to run many modern applications. As such, retiring this technical debt is a high priority for most customers.

A bit of background on why many customers postponed this. The single biggest impact on app compat moving from Windows XP to a more current operating system is the change in the default assumptions regarding user privilege. Historically, Windows had a default assumption of assuming the user was a local administrator, which dates back to Windows 1.0 which didn't incorporate a granular security model. Even though the Windows NT family of operating systems provided a granular security model, for compatibility with existing software most organizations did not leverage this. This led to a feedback loop. Developers who were local admins continued to create new software that inadvertently depended on having those admin rights, and users who had admin rights weren't aware that it was happening. Breaking this cycle was critical – with end users as local administrators, there simply isn't enough that can be done to secure the endpoint, given that adversaries have the rights to disable any protections that are put in place!

While many people believe that UAC is what is breaking applications, this is not precisely true. UAC is actually a suite of security technologies that includes functionality designed to remediate failures that come from running as a standard user! Though it is not able to fix them all, hence the perception. As such, the most common reason for an organization disabling UAC is because they are also running as Local Administrators. (Standard Users running without UAC would actually experience a higher failure rate.)

Why is it so important to, at a minimum, enable UAC now? Even if your organization isn't yet planning to transition a larger percentage of users to standard users, enabling UAC should be an important part of your Windows 10 strategy. In addition to providing remediation technology for migrating to standard users, UAC provides additional security infrastructure that assist with application isolation and security containment. **This infrastructure is what is used to better secure Universal Windows Platform (UWP) apps, and without UAC, these applications don't run.** Given that the investments Microsoft is making in new APIs and the application platform itself are concentrated in UWP apps more than legacy Win32 apps, this is a problem that will continue to grow over time as more applications are released on this platform. There are already several apps that are shipped with Windows 10 that are based on UWP, including Microsoft Edge and Calculator.

> Note that while some in-box UWP apps may run without UAC enabled, no testing has been performed in this configuration by Microsoft.

With the idea that enabling UAC is critical, we now need to assess the impact. Since the transition from the default assumption of administrators is what drove the majority of app compat impact in Windows 7 and Windows 8 migrations, we have seen Windows 10 migrations that are retiring this technical debt share many characteristics as a more typical Windows XP retirement. So, looking at the recommendations in 2.7 around how to assign testing approaches, we would suggest starting with the assumption that the Windows 10 retirement will look more like a Windows XP retirement than a Windows 7 retirement in order to pay back this debt.

3.1.2 Transitioning to 64-bit Windows

Unlike Windows Server, which transitioned to 64-bit only in Windows Server 2008 R2, Windows Client – including Windows 10 – is still available in a 32-bit variant. So, why retire this technical debt now? By continuing to run 32-bit Windows, devices are unable to take advantage of all of the memory resources available to them. Memory beyond 4GB is not accessible using a 32-bit operating system, and this will gradually introduce more constraints. This is gradually becoming more problematic, as applications continue to grow in size based on typical memory availability, yet despite larger memory availability the constrained OS isn't able to provide the expected performance improvements the hardware could support. Additionally, more and more testing is conducted in 64-bit Windows by software manufacturers, increasing organizational risk with untested scenarios on new software.

To support a transition to 64-bit Windows, you'll again want to leverage many of the techniques organizations used in migrating away from Windows XP. Leveraging techniques to identify any 16-bit applications and addressing issues with hard coded paths will once again accelerate this transition. It adds additional potential risk, with well-known techniques and outcomes. Plan on needing to replace any 16-bit applications and installers, and on relatively straightforward remediation for issues with hard coded paths.

3.2 Enabling Key Features

When planning the transition to Windows 10, many organizations develop a business case to justify the transition. The most popular justification we have seen so far is **security** – Windows 10 offers significant improvements in security. It's important to note that some of these features, particularly security features, might have an impact on app compat. Consequently, it's important to identify which features you intend to use, and enable them minimally in lab and test environments to begin to identify app compat issues. For example, while very rare, Microsoft has been discovering some applications and application installers which are impacted by enabling **Credential Guard**, so enabling this in the test environment is helpful in early discovery.

3.3 Support statements

The other challenge we have seen with Windows 10 migrations, and migrations in general, is the careful delineation of tasks. Technical teams tend to focus on whether or not the application works, but that's not always the only concern. An application might work technically, but if there is a business requirement that the vendor of the application provide ongoing support that will not be sufficient.

We recommend that you first identify which applications you require vendor support for. For most organizations, vendor support is of course desired, but what we want to capture is whether or not it is mandatory. For those applications where it is mandatory, the next step in the process is to investigate whether the version you are using is supported. If not, then the remediation is always to either upgrade to a supported version, or to transition to a competitive solution which offers this support.

Microsoft recommends checking the end of support dates for old versions. We have worked with many customers who have suggested that they need to remain on both an older version of the application in order to retain support. In our experience, many software vendors will stop supporting old versions of their products on old versions of Microsoft Windows at the same time that Microsoft ends support for these old versions. Consequently, we often run into this scenario:

Historical Support		Perceived Support		Actual Support		Preferred Support	
OS	App	OS	App	OS	App	OS	App
Supported	Supported	Unsupported	Supported	Unsupported	Unsupported	Supported	Unsupported

Clearly, in the pursuit of more support, arguing for fewer supported components of the application stack represents more risk, and as such we advise scrutinizing the support statements you have to ensure that the older platform remains supported, and when that support is set to expire.

4 Supporting Windows as a Service

One important difference between moving to Windows 10 as compared to previous OS migrations is that Windows 10 will continue to evolve, rather than remaining mostly static for years at a time until the next OS migration takes place. As such, learning to manage these lower risk transitions using the Pragmatic App Compat Approach is important. There are a few other recommendations that we have to improve these processes.

4.1 App Management

Historically, we observed most customers begin their OS migrations with a Discovery phase – where they sought to sort out exactly which applications they had in order to plot out their app compat testing. While we would often recommend looking at application portfolio management, we didn't see a lot of uptake in investments here.

With Windows as a Service, the important of considering this is magnified. In addition to short-circuiting the Discovery phase (which now happens far more regularly), there's an emerging pattern which makes this even more critical:

The best predictor of whether or not an application will fail on the latest Windows 10 Build is, today, whether or not it failed on the last one.

In other words, in order to better manage the risk of application failure, it's becoming increasingly important to keep track of what failed last time. We have, internally, been referring to this as software brittleness, and applications with a persistent problem are ones that we investigate to understand the root cause of the brittleness, and then look to address this where possible.

Within the realm of app management, our experience with customers indicates that the most successful customers don't attempt to always catalog everything – instead, they divide the applications they are aware of into categories of support. A common taxonomy is:

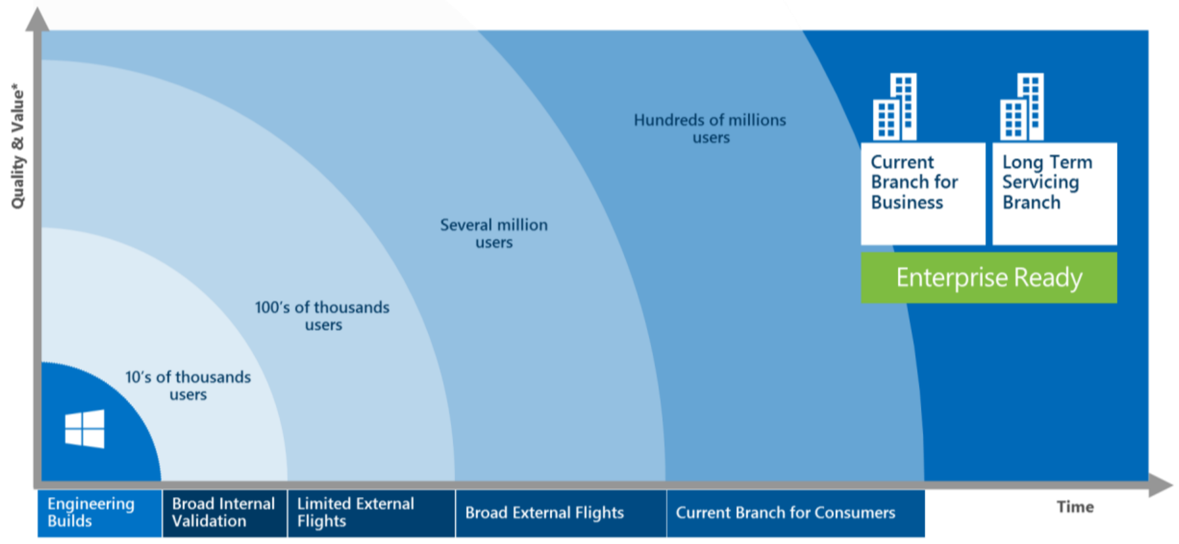
Category	Description
Managed	The IT department takes on accountability for tracking, testing, servicing, and upgrading applications, tracking them over time
Supported	The IT department doesn't hold any accountability for proactive management of applications, but the helpdesk will support business units incorporate these applications into business processes
Unsupported	The IT department doesn't hold accountability for management of applications, and the helpdesk doesn't support any issues with them

Category	Description
Banned	The IT department actively tries to prevent these applciations from running

As you can see, in this taxonomy, the two categories that require active tracking are Managed and Banned – the others require no tracking. Distribution varies based on business unit autonomy, there really is no right answer. However, a key indicator of success is whether or not the IT department is bound to support an application they had no hand in acquiring, no ability to certify it meets quality standards, and often no awareness of its introduction. To use a metaphor, if your IT department is expected to provide insurance on the app, it's of course only fair that they are notified of the asset, have the ability to judge its insurability, and optimally charge a premium for this insurance!

4.2 Running Rings

The Windows as a Service approach incorporates several rings of deployment. Windows is in a state of continual engineering, and at various milestones these builds are released to progressively larger audiences, as illustrated in the following diagram:



Leveraging this ring approach can offer important opportunities to better manage risk. Having users leveraging faster rings, whether Windows Insider builds or the Current Branch (also known as Current Branch for Consumers) can give you early visibility into changes coming to the Current Branch for Business builds.

Microsoft recommends running progressively more devices in higher level rings in order to gain the earliest visibility into changes in the operating system.

> Several customers have suggested moving to the Long Term Servicing Branch for all devices in order to maintain the status quo on upgrade cadence. While appropriate for many devices where change should be minimized, we don’t typically recommend this for general purpose computing. Among other side effects, the absence of in-box Universal Apps can degrade the user experience noticeably.

4.3 Vendor Support for Windows as a Service

When investigating Vendor Support, discussed in Section 3.3, the advent of Windows as a Service also potentially changes how to interpret support statements. We recommend checking to see if there are constraints on vendor support, such as tying support to a particular build or a particular branch, and understanding how that support alignment works with your current strategy.