Testing and Developing Supportability Roadmaps for Independent Software Vendor Applications

Microsoft Corporation
Published: *October 2010*

Authors: *Allan Hirt (Megahirtz LLC) and Maxwell Myrick (Microsoft)*
Reviewer: *Art Rask (Solid Quality Mentors), Anu Chawla (Microsoft)*

Applies to: *SQL Server 2008, SQL Server 2008 R2*

Abstract

This document is intended for independent software vendors designing, selling, and supporting applications. It is often a challenge to keep up with supporting your own applications, let alone changes to the underlying platforms, such as Microsoft SQL Server, that the application supports. In the face of these changing conditions, the only way to ensure that your application remains functional, reliable, and high-performing is through ongoing testing. ISVs can also mitigate risk by publishing a supportability roadmap to help customers navigate what is and is not supported.

Microsoft

Copyright Information

Table of Contents

Overview

Customers expect an independent software vendor's (ISV's) application to work seamlessly with many different software and hardware solutions. Customers also want to be able to update or upgrade each of these separate components without their application experiencing any adverse effects or lengthy downtime. Upgrading should not be a pain point for customers or the ISV.

One of the hardest things any ISV has to do is manage the risk associated with the need to accept change, whether that change is made to its own application or underlying platforms. When it comes to Microsoft SQL Server, as more time passes, the delta of change between older and newer versions becomes larger and larger. If an application has not been tested against newer versions of SQL Server released since the application was introduced, not only is the ISV facing huge risk with possibly years of change to account for, but supportability may have been compromised for both the ISV and its customers.

The only way to minimize risk is for an ISV to perform ongoing testing against the application for changes and updates to the underlying platform and to publish a roadmap to help customers navigate what is and is not supported. This paper introduces the different kinds of updates that affect SQL Server (and subsequently the application) as well as how those updates should be tested on an ongoing basis so that upgrading becomes a relatively painless operation. We will then discuss how to create a supportability roadmap for an ISV application.

Types of Updates

Before we discuss testing or the roadmap, it is important to understand the different kinds of updates from Microsoft that may affect an application over its lifecycle. Microsoft updates include product version upgrades, hotfixes (which include cumulative updates), and service packs. Other updates that are not SQL Server-specific can also potentially affect a SQL Server installation. This section describes the various update types so that you know what you need to test as an ISV.

> **IMPORTANT** Each Microsoft product handles hotfixes and other kinds of updates differently and may implement cumulative changes differently than SQL Server.

Major Version

When you develop an application against a database platform such as SQL Server, you generally have a version you use as a starting point, such as SQL Server 2008. When a new Microsoft product ships, that version is generally referred to as a *major* version. Examples of a major version for SQL Server include SQL Server 2000, SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2. SQL Server ships new major versions about every 3 years.

> **NOTE** If you currently support a down-level major version of SQL Server, a major version could be considered an upgrade (a specific kind of update).

With a major version of SQL Server, Microsoft introduces new features and may enhance or alter existing ones. These changes can be significant. Thus, the SQL Server development team thoroughly tests both functionality and performance of all major releases.

New features should not pose significant risk to an existing application that does not use them. Although there may be a learning curve to implement a new feature (if desired), it represents additional functionality that should have little impact on existing application functionality unless the new SQL Server feature replaces one you are already using.

The most challenging part of a major upgrade for an ISV is handling changes to existing SQL Server features that your application uses or changes in the Database Engine's behavior that may result in different query results. Your customers expect your application to behave as it did before the upgrade, so it is your responsibility to ensure a consistent experience. Even improvements, such as a query running faster with no tweaks required to the application code, represent change that needs to be evaluated and possibly documented. If a change has an adverse effect on your application, your team needs to fix the problem. Testing is the only way to detect any adverse effects to the application so that you can mitigate them before they reach customers.

To help ISVs and customers get a jump start on testing new versions, Microsoft regularly provides full, pre-release versions of its products so that they can be tested at different phases of development. For SQL Server, Microsoft generally ships two to four *community technology preview* (CTP) releases during the latter stages of developing a major version so that testing can happen along the way instead of only after the product is shipped.

Over the course of producing a major version, a lot of changes can happen to the product. An early CTP may have some, but not all, of the new features fully implemented, for example, and may not be completely optimized for performance yet. If there is more than one CTP, each release should be more complete and tuned than the previous one. Besides CTP releases, SQL Server usually provides at least one *release candidate* (RC). An RC is exactly what it sounds like – if it meets the testing and quality standards when put through its paces and there are no major late changes, that version could potentially become the final release.

In addition to the various pre-release versions, Microsoft also accepts select ISVs and customers into what is called the Technology Adopter Program (TAP). TAP allows the SQL Server development team to work closely with participating ISVs and customers to ensure that when SQL Server reaches RTM, their application or product fully supports that new major version of SQL Server from day one. Some ISVs take advantage of this opportunity, others do not. Being part of TAP is a serious commitment from both Microsoft and the participating company, but both benefit greatly.

Hotfixes and Bi-Monthly Cumulative Updates

In contrast to a major version, a *hotfix* is the smallest unit of change for SQL Server. Hotfixes are pretty much what they sound like: If a critical problem is detected (either by Microsoft or by a customer via a support incident), it is fixed and a corresponding installation package for the fix is generated. A hotfix works only with a specific version of SQL Server (for example, SQL Server 2008 Service Pack 1 (SP1) or

SQL Server 2005 SP3). This is because a SQL Server hotfix contains only one set of files, which is appropriate for a specific version of the product. If an issue affects more than one product version and there is customer need, Microsoft releases separate hotfix packages for each SQL Server version that contain the appropriate files for that version. (For example, you would have a hotfix package for SQL Server 2008 SP1 and a separate hotfix package for SQL Server 2005 SP3.) Be aware that hotfixes have a trade-off: They are made and shipped quickly, so they do not undergo the same level of testing as a new major version or SP.

SQL Server follows a purely cumulative servicing model, which started with SQL Server 2000 SP3 hotfixes. A purely cumulative model means that each hotfix package contains all the changes that were previously shipped. For example, a SQL Server 2008 SP1 hotfix will contain the files for that specific hotfix as well as for all previous SQL Server 2008 SP1 hotfixes that shipped before it. This model has two main advantages:

1. You always know exactly what hotfixes you have installed.
2. You always know that Microsoft has tested this exact combination of hotfixes.

The purely cumulative model also greatly simplifies management of SQL Server because you do not have to install each individual hotfix. Whether you need one or 100 hotfixes, all can be installed in a single step. Should a later hotfix be needed to resolve a different issue, when you run that hotfix, you will get everything you need because it will contain everything prior to it, including the hotfix you already applied.

> **TIP** For a given SQL Server instance, you will never need to install more than one hotfix at any time, which is the highest package that includes the highest number fix required. If you are installing more than one hotfix, you are probably adding an unnecessary step.

SQL Server also ships a special category of hotfixes that are of great benefit to ISVs for maintaining a simplified update strategy for hotfixes. Originally, hotfix packages shipped on a more frequent basis (from daily to weekly), but each package contained only one or, at most, a handful of individual changes. Starting with SQL Server 2005, Microsoft introduced a special process of releasing essentially the same overall number of hotfix-level changes but doing so in a smaller number of shipped hotfix packages. This special category of hotfixes is known as the bi-monthly *Cumulative Update* (CU) and ships on a regular schedule. Each CU is a hotfix package containing all of the changes to that point – no different from a standard hotfix – but the CU has the added benefit of being more thoroughly tested because there is more time to do so. Even with the introduction of the CU, individual hotfixes are still made and shipped by the SQL Server team. These hotfixes are classified as Critical On Demand (COD) hotfixes. However, most ISVs should take advantage of the six CU packages that ship each year, which will allow them to schedule their testing in advance.

> **TIP** For any given hotfix you want to use for production systems, strongly consider installing a CU package that contains it. CU packages include the same change as the COD hotfix but receive more testing.

However, keep in mind that a hotfix or CU should be recommended for customer deployments only if you are confident it will resolve a specific issue with your application and there is not a reasonable workaround. If a hotfix is required by your application, it is imperative that you perform sufficient testing to ensure that the hotfix does not cause any regressions.

Service Pack

A *service pack* (SP) is an infrequent update released by Microsoft that has a moderate number of changes and receives a medium level of testing, which means more testing than with a hotfix but less than with a major version. An SP is also the most familiar form of SQL Server update for customers and ISVs alike. It is basically a collection of all previous hotfixes in a new package that is tested more thoroughly. The hotfixes that shipped prior to the SP form the basic content for any SQL Server SP.

As with other SQL Server updates, SPs are cumulative. For example, you do not need to apply SP1 before SP2; you can just apply SP2 and you will get everything in SP1 as well. As with a major version of SQL Server, Microsoft will often release at least one CTP version of an SP so that those interested can start doing early testing.

The following examples of bug fixes and the hotfixes (including CU changes) shipped in SQL Server SPs give you a good feel for the included hotfix-related content:

- List of the bugs that are fixed in SQL Server 2005 Service Pack 3
- List of the bugs that are fixed in SQL Server 2008 Service Pack 1

Note that SPs may contain hotfixes and additional bug fixes not contained in any CU or hotfix release. Although it is somewhat rare, a SQL Server SP may also include new functionality or features. Examples where Microsoft shipped new SQL Server functionality in an SP include:

- SQL Server 2005 SP1, which enabled database mirroring
- SQL Server 2005 SP2, which included smaller features spread over much of the product (see What's New in SQL Server 2005 SP2 for details)

It is also important to note that an SP may not contain all of the hotfixes included in some CUs because at some point, the SQL Server development team has to freeze the code to be able to test and ship it. This means that a recent CU for an earlier SP may have hotfixes that are not contained in a later SP (or major release). If a specific hotfix is required for your application, you may not be able to recommend and support a SQL Server SP until a corresponding CU containing the required hotfix is released for that SP.

Consider this example from the Microsoft Knowledge Base (KB) article Cumulative Update package 1 for SQL Server 2008 R2:

> "Cumulative Update 1 for Microsoft SQL Server 2008 R2 RTM contains only hotfixes that were released in Cumulative Update 5, 6, and 7 for SQL Server 2008 Service Pack 1 (SP1). Cumulative Update 1 for SQL 2008 R2 RTM is only intended as a post-RTM rollup for Cumulative Update 5-7 for the release version of SQL Server 2008 SP1 customers who plan to upgrade to SQL Server

> 2008 R2 and still keep the hotfixes from Cumulative Update 5-7 for the release version of SQL Server 2008 SP1. No new hotfixes have been included in this cumulative update."

This means that the SQL Server 2008 R2 code stopped incorporating changes from SQL Server 2008 SP1 somewhere around or after SQL Server 2008 SP1 CU4. Thus, any hotfixes released for SQL Server 2008 SP1 starting with CU5 would need to be rolled forward to SQL Server 2008 R2.

> **TIP** If you need a hotfix on a given version of SQL Server, you can use the published information in the KB articles to determine if a higher version of SQL Server includes this hotfix or, if not, which hotfix to apply on that version to retain the hotfix functionality.

Other Updates

SQL Server is built and dependent on other components. The two major Microsoft dependencies for SQL Server are the Windows operating system and the .NET Framework. Some applications may include other dependencies such as Internet Information Services (IIS) if using SQL Server Reporting Services, for example, or System Center Operations Manager for service monitoring. All of these supporting services have their own updates and patches that are independent of SQL Server. These updates could potentially affect your SQL Server application. As an ISV, you need to manage and test these updates as well, based on the recommended configurations you publish as part of your supportability roadmap.

The process used to test and validate your own application code changes between versions should be similar to the process used to validate SQL Server updates and all other updates (including the examples above) that affect your application system. Use the SQL Server-specific testing processes described later in this paper as guidelines for addressing the needs of the overall infrastructure of your application.

If you license or use any third-party components (or even subcontracted code) from other software companies as part of your application, you must adhere to their support policies – including what version(s) of SQL Server they support. This is no different than the position your customers find themselves in when they implement your application.

Understanding SQL Server Versioning

Now that you understand the different types of SQL Server updates, how do you determine what SQL Server version you are supporting for your application? At a high level, your particular version of SQL Server will always have a major version and an SP version, and it may also have a hotfix version. The version will apply to an instance of SQL Server. This section will describe in more detail how to understand the differences in versions and what they mean to an application.

SQL Server Versioning

There are many ways, both programmatic and visual, to determine the full version of a SQL Server instance. One method is to execute the query

```
SELECT @@VERSION
```

Another would be to look at the properties of the instance in SQL Server Management Studio (SSMS). However, what you really need to do is understand how to read the version number you will get from those techniques. The resulting version will fully describe your installation of SQL Server by defining all three of the update types: major version, SP version, and hotfix version.

SQL Server version numbers will generally look something like this: 10.00.2746.00. To understand how to read the version number, it is best to think of the number as having the following format:

MA.Ma.BULD.XXXX

The first two parts (MA.Ma) represent the major version of SQL Server. The major version of SQL Server can be easily mapped to the MA.Ma format as shown in Table 1.

Table 1 - SQL Server Major Version Numbers

Major Version	Version Number
SQL Server 7.0	07.00
SQL Server 2000	08.00
SQL Server 2005	09.00
SQL Server 2008	10.00
SQL Server 2008 R2	10.50

The reason SQL Server 7.0 is included in this table is that you may have heard people refer to some later SQL Server versions as SQL Server 8, 9, or 10, which are SQL Server 2000, 2005, and 2008, respectively.

> **TIP** If you have trouble remembering that SQL Server 2008 is version 10, just count up each major version from SQL Server 7.0, which is version 7.

BULD, short for "Build," is the most commonly used part of the overall version number. People will often say something like, "Our SQL Server is on build 2746." Of course, this statement implies that the person on the other end of the conversation already knows the major version of SQL Server, because each major version could have a build version of 2746. Since the major version of SQL Server changes only every few years, this shorthand is rarely problematic when used with co-workers.

> **NOTE** XXXX may or may not be used and/or represented. It is only shown for completion's sake and would represent a further clarification of the SQL Server build.

One benefit of SQL Server's build number is that it effectively describes both the SP version and the hotfix version (if any) that have been applied. Although this concept might initially seem complicated, it is easy to understand. The key is to remember that a hotfix is an update beyond the SP level. That means that any hotfix will be newer than the version it replaces. In version numbers, the newer number is always higher, so the hotfix will always have newer files than the corresponding SP it applies to. Any given hotfix build version states that the installed SP must be the nearest SP version that is below the hotfix version number.

Table 2 provides a mapping of SQL Server SP versions to their corresponding build ranges. If the build number of the system matches the build number shown in the "Service Pack Build" column, then the system is running on that particular version and has not yet installed any post-SP hotfixes. If the build number of the system doesn't exactly match one of the build numbers in the "Service Pack Build" column, then a hotfix has been applied to the system. In that case, find the build number with the build ranges given in the "Hotfix Range for This Service Pack" column of Table 2. Once you've located the build number within the hotfix range, the corresponding "SQL Server Version" value on that row is the SP installed on that system. This should make sense because each hotfix on SP1 will have a higher build number than SP1 but a lower build number than SP2. Thus, the first possible hotfix on SP1 is the build number of SP1 plus one, and the last possible hotfix build number on SP1 is the build number of SP2 minus one.

Table 2 - Mapping of Build Numbers to Service Packs and Hotfixes

SQL Server Version	Service Pack Build	Hotfix Range for This Service Pack
SQL Server 2000 SP3/SP3a	08.00.760	08.00.761 to 08.00.2038
SQL Server 2000 SP4	08.00.2039	08.00.2040 and up
SQL Server 2005 RTM	09.00.1399	09.00.1400 to 09.00.2046
SQL Server 2005 SP1	09.00.2047	09.00.2048 to 09.00.3041
SQL Server 2005 SP2	09.00.3042	09.00.3041 to 09.00.4034
SQL Server 2005 SP3	09.00.4035	09.00.4036 to TBD *SP3 is the latest update for SQL Server 2005 at the time of this writing*
SQL Server 2008 RTM	10.00.1600.22	10.00.1601 to 10.00.2530
SQL Server 2008 SP1	10.00.2531	10.00.2532 to 10.00.3999
SQL Server 2008 SP2	10.00.4000	10.00.4001 to TBD
SQL Server 2008 R2 RTM	10.50.1600.1	10.50.1601 to TBD

> **IMPORTANT** Every SQL Server instance has an SP version. When no SP has been installed, Microsoft refers to this as "running the RTM version" and lists RTM as the SP version.

For each SQL Server update, the key to understanding what it provides is knowing what fixes are included in the package. Microsoft has made this task straightforward by publishing cumulative KB articles for each version of SQL Server, as shown in Table 3. Note that each cumulative KB article applies to only one specific major and SP version of SQL Server (e.g., SQL Server 2008 SP1 or SQL Server 2005 SP3).

Table 3 - List of SQL Server Cumulative KB Articles

Product Version	KB Article
SQL Server 2000 SP3/SP3a	810185
SQL Server 2000 SP4	894905
SQL Server 2005 RTM	N/A[1]
SQL Server 2005 SP1	913090

SQL Server 2005 SP2	937137
SQL Server 2005 SP3	960598
SQL Server 2008 RTM	956909
SQL Server 2008 SP1	970365
SQL Server 2008 SP2	N/A[2]
SQL Server 2008 R2 RTM	981356

[1]Article does not exist

[2]SQL Server 2008 SP2 was just released at the time this whitepaper was published yet; check the support website

> **TIP** All of the above tables are useful, but you can find a single source that points to most of the summarized information for SQL Server 2005 and 2008 in KB article 957826. This KB article is continuously updated to provide links to the cumulative KB articles for the currently supported versions of SQL Server. Those cumulative KB articles will effectively give you the SP build numbers as well.

To use the SQL Server cumulative KB articles, simply find your build number on the list of build numbers provided in the corresponding KB article. Every hotfix listed in the KB article with a lower build number than the one installed is included in your hotfix, and every hotfix with a higher build number is not included in your hotfix. To get the later hotfix, you would have to upgrade to that higher hotfix build number, and in doing so, you would receive every other hotfix listed in the article between the build number you are currently on and the build number you are moving to.

Now that each segment of the version number has been explained, let's consider the sample SQL Server version number mentioned at the start of this section: 10.00.2746. Looking up the MA.Ma value of "10.00" in Table 1 shows that the major version is SQL Server 2008. Looking up build number 2746 in Table 2 shows that this does not exactly match any of the RTM or SP build numbers listed, which means the system has had a post-SP hotfix installed. Using the "Hotfix Build Range for This Service Pack" column from Table 2, you can see that build 2746 is in the range of hotfixes that apply only to SQL Server 2008 SP1. That means this system has SP1 of SQL Server 2008 installed. Finally, by looking at the cumulative KB article for SQL Server 2008 SP1 (KB 970365), you can see that the specific hotfix package that was installed is KB 975977, which was the 5th Cumulative Update (CU) package for SQL Server 2008 SP1, released November 2009. Therefore, this SQL Server is running SQL Server 2008 SP1 with SP1 CU5. Again, using the cumulative KB article for this build (KB 970365), you can see every hotfix that is installed and quickly determine if a hotfix you want is not installed.

Multi-Instance and Versioning

Where there is a single instance of SQL Server on a standalone server or cluster, the versioning story is straightforward. When multiple instances of SQL Server exist, the story can be a bit more complex.

Consider this example: A standalone server has two instances of SQL Server installed. Both instances are currently at SQL Server 2008 RTM. Each instance is used for different applications. One application requires as part of its update that SQL Server 2008 be upgraded to SQL Server 2008 SP1. Microsoft fully supports this configuration – one instance can be at one version, and another instance installed on the

same server can be at a completely different version. However, there is one fundamental concept that must be understood here: While the dedicated binaries for each instance as well as the internals will be at their respective versions (RTM or SP1, in this case), any shared components used by both instances will be upgraded to SP1. The shared components generally represent administration tools such as SSMS and thus present minimal risk because they have nothing to do with the functionality of the application.

GDR Branching Basics for SQL Server

Generally Distributed Release (GDR) branching may be new to many of you. At a high level, it represents a choice that determines how changes will affect your SQL Server-based application. This section will describe GDR branching and why it is important for you to understand.

It is a given that all customers will need to update their SQL Server installations at some point, whether by installing a hotfix, CU, or SP. In fact, security updates, a special form of hotfix, are often mandated to be installed at most customer sites. These customer-driven environment changes are yet another reason why testing and certifying updates is crucial for any ISV. To understand what makes SQL Server security updates different from any other SQL Server hotfix, you must learn a bit about the servicing branches.

Think of the SQL Server update types we covered earlier – hotfixes, CUs, and SPs. To release an SP, Microsoft has to branch code at some point so that hotfixes can still be released for a specific version of SQL Server while the SP is in development. This process occurs again and again as more versions and branches of code are created (for example, SP1, SP2, and so on). GDR branching plays into this.

Each time a new SP is released, two new branches are created: one for hotfixes and one for security updates, also known as the GDR branch. When a customer needs a security update to SQL Server, they will choose to use either the hotfix package or the GDR package.

Many customers will be able to use SQL Server without installing hotfixes and only need security changes. These customers would install the GDR package. Customers who need a particular hotfix to address an issue they experienced in their environment and also need the security change will install a hotfix that includes both the hotfix and the security update.

> **IMPORTANT** Because SQL Server updates are cumulative, if you are on the GDR branch today and install a higher GDR package, you get that change plus the rest of the changes on the GDR branch up to that package level. If you are on the hotfix branch and you install a higher hotfix branch package, you get all the changes on the hotfix branch up to that package level. Both GDR and hotfix packages are cumulative on their respective branches.

Why do you need to care if you are on the GDR branch or the hotfix branch? The choice of branches gives you the power to control your rate of change. This impacts your test scenarios and the upgrade guidance you should be providing to your customers.

There are four things you need to understand to allow you to take advantage of having a choice in packages:

1. You always start on the GDR branch.
2. You stay on the GDR branch until you install your first hotfix.
3. You are then on the hotfix branch until you install the next SP or upgrade to a higher major version. At that point, you have moved back to the GDR branch.
4. You can determine if you are on the GDR branch by using the process we described earlier for determining your SQL Server version.

The first three rules are enforced by the SQL Server hotfix setup programs, so you do not have to worry about making any mistake that will render your systems unsupported. As an ISV, the only thing you want to do is actively choose which set of packages to use. When you test SQL Server updates to make customer recommendations or to add to your supportability roadmap, it is always better to stay on the GDR branch when possible. You can always go from the GDR branch to the hotfix branch, but the only way to get back on the GDR branch is to go to the next SP or major version.

As we mentioned, detecting whether you are on the GDR branch files or the hotfix branch files is simple using the tools in the above section on how to check your SQL Server version. The implementation for SQL Server GDR branching reserves the first 99 build numbers following each SP for GDR branch packages. Hotfix packages use build numbers at least 100 above the SP build number. Using Table 2, if your build number is equal to the SP build number or if it is less than the build number plus 100, you are on the GDR branch. If your build number is equal to or greater than the build number plus 100, you have applied a SQL Server hotfix and are on the hotfix branch. The same information can be looked up in the cumulative KB articles listed in Table 3. All GDR packages listed in the cumulative KB articles will have the lowest build numbers shown and will be in the range of build numbers between the build version of the SP and the build of the SP plus 99.

Risk vs. Change

Change represents potential risk, and as an ISV, your job is to mitigate that risk. It does not matter if the change is within your application or in the underlying platform. The longer you wait to test things such as SQL Server hotfixes, CUs, SPs, or major versions, the more risk as well as work is involved – plus, supportability could be compromised.

As an ISV, you must test for changes to the underlying platform at regular intervals to lower risk to customers and improve the stability of your application going forward. You also need to test changes for another major reason: supportability. As an ISV, you should be ensuring that your customers have the ability to apply the updates from Microsoft in a reasonable time frame after they are released. Since SPs (which includes RTM versions) have a supportability end date, it is your responsibility to ensure that customers have the ability to stay supported by both you and Microsoft. Whether they choose to apply the updates is something that may be beyond your control, but by testing and documenting the change in your supportability or lifecycle roadmap, you are doing your duty.

There are two ways to think about testing risk versus the amount of change. The first involves the platform (or platforms) you are currently supporting. As noted above, SQL Server ships different kinds of updates. Some are predictable (the bi-monthly CUs), and others come at different intervals (hotfixes,

SPs). As a general rule, there is more change early on than there is as a product matures. This is why you will see more frequent SPs earlier in a product's lifecycle and less frequent ones over time. The first SP often (but not always) is shipped within 6-8 months of RTM, with one more coming on its heels within 12 months. SQL Server 2005, for example, fits this pattern. This means that early on as you support the platform, you will need to be more aggressive in your testing because Microsoft may introduce changes more frequently. However, as time goes on, you should have less testing to do.

The next way to think about risk versus change involves a major new release of SQL Server, such as SQL Server 2008 R2. As with your own application, development starts at one point and ends with final release, but there are multiple milestones, builds, and releases that may occur before the ship date, as Figure 1 shows. In the case of Microsoft, before customers get to see anything, there will be some internal milestones, which will continue to occur even if there are some public releases. SQL Server generally releases CTP versions of SQL Server long before its ship date; these CTPs could even appear a year before the actual RTM. Earlier CTPs may not be feature complete or fully tuned for performance, but they provide a great opportunity for an ISV as part of ongoing testing to start seeing if any existing functionality breaks. Since SQL Server is at the CTP stage, if problems are found and reported, there is also a chance of them getting fixed. The longer you wait to test, the less likely it is that any changes will get into RTM. By the time a public RC is available, the product can be assumed to be feature complete. Barring any major problems, there will be little (or no) change between the RC and the RTM version, and the window of opportunity to affect change has effectively been closed.

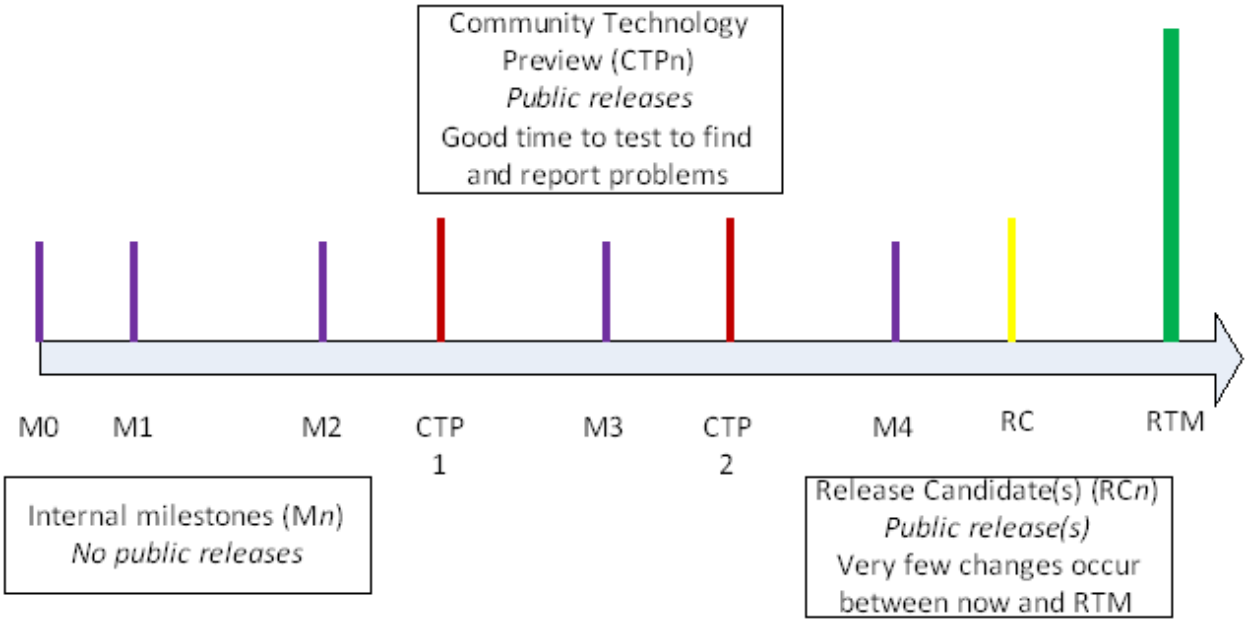

Figure 1 - Example of a release cycle for a new version of SQL Server

Performing Ongoing Testing

Why do you need to perform ongoing testing? As evidenced above, all applications are in a constant state of change. In the case of SQL Server, there are hotfixes, CUs, and SPs as well as major releases that

are released throughout your application's lifecycle, so your work does not end the day you ship the application. The work is just getting started.

The more testing you do along the way, the less painful it is to support each new version and update. That may not be intuitive, but here's the rationale: By testing at smaller intervals (such as at CUs), you are testing many, if not all, of the changes that eventually go into SPs as well as the next major version of SQL Server. If you are testing at each level of update along the way, testing an SP, for example, should be no big deal because a lot of what is in the SP has already been assessed. The number of changes that could affect your application goes up, not down, as time goes on. If a problem occurs somewhere along the way and, for the sake of argument, needs a fix because it breaks your application, it is easier to deal with the problem in a smaller increment than in a more comprehensive package after you've waited months or years between testing of the underlying platform. Although such frequent testing may seem like a lot of work, if you have good automation, the process should not place a great burden on your organization.

This section covers how to approach and perform ongoing testing for an application that uses SQL Server 2008.

> **TIP** Your application may support different SQL Server versions and editions, such as Standard or Enterprise. The testing methodologies described in this section cover all supported platforms.

Testing SQL Server Updates

Let's begin by discussing how to test the various SQL Server update types against your application so that you stay ahead of the curve and make testing a relatively painless process.

Major (Upgrade) Version Testing

Major versions of SQL Server come along every few years (anywhere from 2 to 5 years, if you measure the various releases since SQL Server 7.0). That gives you a lot of time to plan, but you will still have to address a major version upgrade at minimum every 5 years. The challenge from an ISV perspective is that a major upgrade requires the most work to test; it is not just an update to an existing version of SQL Server that you already support. As mentioned earlier, Microsoft may change some behaviors or the way something works, so you will have to run your full set of tests against this version to ensure that everything still functions properly. Whether your customers perform an in-place upgrade or are deploying a new installation of the application with this version of SQL Server, your work is exactly the same.

Microsoft may introduce a new major version of SQL Server while you are between your own major application version releases. If the decision is made to support that new version of SQL Server, there are really two testing efforts you need to gear up for: testing SQL Server fully against the current version, and getting that version of SQL Server into the pipeline for the next version of your application, which is most likely already in development.

Testing coverage for a major version of SQL Server will most likely be a combination of manual and automated testing. For a major upgrade, you will probably want to start with a period of manual testing.

This is an opportune time to evaluate the new features and functionality of SQL Server to determine if your application should take advantage of them now or in a future version. Some features will require no modifications to your application and can be supported "out of the box." A good example of this type of enhancement is the backup compression introduced with SQL Server 2008 Enterprise and included in SQL Server 2008 R2 Standard, Enterprise, and Datacenter editions. Other features need to be coded and incorporated into the application and would generally not make it into the current version of your application. That should not prevent you from certifying the new version of SQL Server to work with your application; it would just mean that to take full advantage of these features, customers would need a later version of your application (which would be a great reason for them to upgrade). A good example of this type of feature would be the new data-tier application (DAC) functionality in SQL Server 2008 R2, which can help standardize how your application is deployed against SQL Server.

> **TIP** As you start to evaluate the new version of SQL Server, part of your testing should include verifying the installation process itself. Ensuring that SQL Server is installed correctly is paramount to a stable ISV application. A broken installation or configuration could cause a variety of issues that appear to be application problems. Start by verifying that the product installed/upgraded successfully, especially with the earlier CTP or beta releases.

As you are doing the initial manual tests against SQL Server, think about the automated testing that will ultimately be needed. Can you reuse existing automated tests? Will you need to devise new ones? What will the effort for automated testing look like end-to-end for a new version of SQL Server? At this point, you should make the changes to automated testing because it will save you time when evaluating later pre-release versions. After the manual testing, proceed to broader functional testing using automation. Here, you are just looking to confirm that any core functionality you rely on is not broken or altered; you are not yet concerned about performance – that is a separate test. Each ISV will have its own certification process for a platform that satisfies its requirements for officially supporting the platform for its application.

> **IMPORTANT** Remember to design test coverage and automation to encompass major version upgrades, which are the broadest case. The more test coverage you have for a major version of SQL Server, the better. Then pick and choose an increasingly smaller subset from that testing to create the test suites you will use for testing SPs, hotfixes, CUs, and security updates, respectively.

Because SQL Server releases CTP and RC versions prior to RTM, it is in your best interest to test your critical application functionality early and often. By reporting any issues to the SQL Server development team early in the process, you have a much greater chance of these being fixed in the released product instead of fixed in a post-release hotfix, CU, or SP. It is important to note that early versions of a new major SQL Server release may not be completely optimized for performance, so it's a good idea to test functionality on the earlier pre-releases (e.g., CTP 1) and move on to your performance testing on the later pre-releases (e.g., RC). By the RC release, the version is pretty much considered final, and very few changes are going to happen between then and RTM. If you've done testing along the way, testing the

RTM release should be easy. If you have not even looked at the release prior to RTM, it's going to be a lot of work to do in a short period of time.

> **TIP** Apply the same evaluation process to major version upgrades of Windows for application servers that use a Windows operating system.

Table 4 provides a concise view of testing recommendations for major versions.

Table 4 - Major Version Testing Guidance

Test Item	Testing Advice
Test New Major Releases Early	The goal is to find any bugs or regressions in pre-release versions such as CTPs and submit them so the fixes for your application get into RTM. Remember that CTP testing supports later usage of GDR branch packages. For earlier CTPs: • First ensure that the installation works correctly, as a setup issue could result in a later functional issue. By confirming setup first, you save time on later problem troubleshooting. • Test application functionality. • Test new features you think your application will utilize to provide early feedback to the development team. For later CTPs: • Include performance testing. • Test features you use today that were updated in this release.
Functionality Testing	The major version would represent a new supported platform, so all application functionality must be retested. Test the RTM version fully even if you tested against early CTP releases. The possibility exists that something may have changed. Automate as much testing of core application functionality as possible. These tests will be used over and over again. Where you will be using features new to this release or already use features that are significantly changed in this release, start with manual testing, which is more efficient for finding unpredictable issues.
Performance Testing	Since the major version is a new representation of the SQL Server engine, a full performance test suite needs to be run to ensure that there are no performance regressions. One of the advantages of this testing is that you may receive a performance improvement for your application – effectively for free – which you can then market to your customers during your own upgrade. If performance regressions are found, they can be dealt with before customers see them.

	It's a good idea to use a mix of scripted load and recorded load during your performance testing to expose any possible issues. Although SQL Server is a self-tuning database, Microsoft occasionally introduces "switches" that can be tweaked. If there are any new parameters in the new major version that may improve your application's performance, testing them will allow you to confirm and take advantage of these new benefits.
Upgrade Testing	Migration may be by a clean-install upgrade or an in-place upgrade. • For clean-install upgrades, focus testing more on functionality that could be affected, such as networking and authentication. Also ensure all the proper configuration settings are migrated from the old system. • For in-place upgrades, focus testing more on setup and performance.

Service Pack Testing

In terms of your application's supportability roadmap, SPs will be the one type of update that you definitely have to test along the way. The published Microsoft support policy for SQL Server SPs is to support the prior version for 12 months after the release of the new one. Every product has a different policy, but the SQL Server policy is the most relevant to discuss here.

For example, assume your application supports SQL Server 2008. SQL Server 2008 RTM expired on April 10, 2010. By that point, all of your customers should have already implemented or have planned for upgrading to SQL Server 2008 SP1. The only real way they could do that is if you have done your part and tested the application and fully support it. If you did not do the work and did not plan on certifying or supporting SQL Server 2008 SP1 until after April 10, 2010, you put your product as well as your customers at risk. Either way, your supportability roadmap should have informed customers about when you planned to support this release and when they could also plan to support it.

> **TIP** SPs will generally ship more frequently and with a higher rate of change in the period following RTM. As a product becomes more mature, the interval between SPs gets longer, and the SPs will tend to have fewer total changes. Scale your test efforts accordingly.

Another time saver when it comes to incremental testing is testing the CUs or hotfixes themselves (see the next section for more details). For example, if you were supporting each successive SP of SQL Server as well as the bi-monthly CU packages, you would already have tested a significant portion of the changes going into the new SP.

Table 5 provides a concise view of testing recommendations for SPs.

Table 5 – Service Pack Testing Guidance

Test Item	Notes
Test Early	Test a CTP build of the new SP if one becomes available. If no CTP build is available, test the latest CU released for the previous SP of the same major version instead. This will have much of the same content. Ideally, do a broad functional test pass and a shorter performance test pass on pre-release SP builds. Save the full performance testing for the RTM version of the SP.
Verify Installation	SP setup is normally reliable, but it is always worthwhile to ensure that the product was installed successfully – especially if installing on many instances or a failover cluster. Also check that all components of SQL Server are upgraded as needed. It is rare but possible for one component (SQL Server Analysis Services, SQL Server Reporting Services, SQL Server Integration Services, etc.) to experience a separate setup issue while the other components, including SQL Server, install successfully.
Confirm Required Hotfixes	If you are using a recent hotfix on the current SP, your hotfix may not be included in the next SP. Use the SP fix list to determine this, and use the cumulative KB articles to determine the post-SP hotfix required for your supportability roadmap. While reviewing the included hotfix list, see if anything else you may be getting as part of this hotfix may affect the application either positively or negatively, and test for that condition. If no post-SP hotfix is required, you may utilize GDR branch packages. It is a good idea to decide if you will attempt to use these or will regularly adopt CU builds instead.
Performance Testing	For SPs, generally focus performance testing on memory and CPU usage as these are the most likely quick indicators of an issue. After completing this testing, move on to the application performance. Use performance test harnesses that can give statistics on individual query performance. You may need to tune after the SP or make specific recommendations to customers for tweaking their databases (such as indexing suggestions). Remember that performance testing may discover improvements to configuration settings (e.g., index tweaks) that can be leveraged for your application on other versions as well.
Check Trace Flags	Trace flags may change their default behavior as part of an SP.

	Ensure that no changes affecting the trace flags you use have occurred. Test your scenarios for specific trace flags to ensure there were no changes (and whether the trace flag is still required for your application in this version).
Functional Testing	Perform a broad functionality test based on the nature of changes. Only do deep testing of features used by the application that were significantly updated in this release.
Group SP Upgrades	Most Microsoft products ship about one SP per year, and many of these upgrades can be combined into a single test pass to reduce costs. Since SPs represent enough change to require a broad functional and performance test pass, the validation process is not trivial. However, there is little detailed testing required. This means that the testing to cover a Windows SP is essentially the same as coverage of a new SQL Server SP. More interestingly, testing BOTH a Windows and a SQL Server SPs simultaneously may add no additional cost. If using this method, your supportability roadmap can communicate the requirement that customers do both upgrades together as well. Although single-instance SQL Server SPs rarely require a reboot, many other SPs do. By combining these, you may also minimize the total system downtime.
High Availability Testing	It is worthwhile to re-validate each of your high availability scenarios on the new SP. This includes failover scenarios, database mirroring, and backup and restore. Since SPs usually occur less frequently than other updates, they are a good reminder that you should test the high availability features on a periodic basis.

Hotfix and Cumulative Update Testing

Hotfixes and CUs should be the least intrusive in terms of your testing, and they may not even formally be represented on your supportability roadmap. The good thing about CUs is that unlike major versions, SPs, and even hotfixes, they have a predictable, set release schedule (every 2 months). How often you choose to test is up to you, but the more frequently you do it, the less work you should have down the road. (The next section discusses security updates, which are a special form of hotfix with special testing considerations.)

A hotfix or CU should require the least amount of testing, but you need to provide enough coverage that you know the application does not break. Remember that hotfixes and CUs are cumulative, so when you test one, you test all the ones before it on that specific branch of code. Because you are probably testing hotfixes or CUs on a more frequent basis, any changes or problems would show that the issue was introduced in the latest hotfix or CU because you have the data showing the previous tests were

successful. Nearly all of the testing for hotfixes and CUs should be automated because you will use this testing regularly. You may also design your automation to run functional and performance testing in parallel for hotfix testing. This is normally not done for larger release types because a functional regression may invalidate the performance test results. In addition, the end-to-end signoff time is usually more important for these periodic releases than for SPs and major releases.

It is a good idea to start hotfix testing by repeating the testing from the most recently tested hotfix. Once this hotfix testing completes without errors, proceed with testing the new hotfix. This process will detect any changes in your test environment or test harnesses that may have happened since the previous tests were completed. Because the testing should be fully automated, there is little associated cost.

Table 6 provides a concise view of testing recommendations for hotfixes and CUs.

Table 6 – Hotfix and CU Testing Guidance

Test Item	Notes
Setup Testing	The core function of a hotfix is to install new files on the existing system. It's a good idea to develop an automated test that verifies that at least a few file versions are replaced (enough to catch most errors) for generic usage in maintaining version control in the test environment. Manually test by reviewing the setup log for the hotfix installation at least once on each supported SP level of SQL Server. Perform a specific uninstall test to check the ability to roll back fixes if necessary. SQL Server hotfix installation should not require a reboot in most cases. Use testing to determine the root cause of any locked files, and eliminate these reboots if possible.
Performance Testing	Start with a performance test using recorded load and per-query timing statistics, as most issues will be at the query level (e.g., a changed execution plan) instead of at the instance level of SQL Server.
Functional Testing	Implement an automated test suite that checks many application functions individually and runs less than 2 days. This investment will be returned many times over because this is a highly re-usable basis for ongoing testing. Perform deeper specific testing using manual resources only on hotfix changes desired or hotfixes specifically requested from SQL Server development for your application.
Review Encompassed Fixes	Since hotfix packages are cumulative, a quick review of the other hotfixes that are included (as listed in the appropriate KB article from Table 3) may find potentially beneficial hotfixes or additional

	areas to test.
Branch Testing	Even if you are planning to use only GDR branch packages, it is a good idea to perform periodic testing on the hotfix branch packages just in case it becomes necessary in the future to use a hotfix. SQL Server ships CU hotfixes every other month, so consider testing every other CU (3 packages per year) to ensure that the hotfix branch is a readily available option for your application. Test every shipped GDR package even if you ultimately do not use those packages. These packages will be broadly distributed (often from Microsoft Update), and many of your customers may receive and apply them automatically.

Testing Security Updates

One of the most common updates that most customers will apply to their servers is security updates. Security updates are shipped on the second Tuesday of every month and are documented in their corresponding security bulletins. Security updates may potentially exist for any current Microsoft product if a security issue has been detected. As an ISV, it is your responsibility to ensure that your application works, which includes testing the security issues for the platforms you support for customer deployments. Many times an ISV may not have stringent requirements for a database server because the underlying operating system is dictated by the version of SQL Server, with the application servers being a bigger concern. However, customers should never find issues related to known and possibly required updates before you do. The reason for this is simple: If that update somehow breaks your application, you should already have a fix in place for it. As with any kind of testing, it is always better to be proactive than reactive.

Before exploring the process for testing the security updates, you must understand how to determine which ones are relevant to your application. First and foremost, this paper is about SQL Server, so any SQL Server-related security update would automatically be in consideration. Having said that, the SQL Server team does not often release security updates, so the testing and planning will not consume a lot of time and resources. Even if the area affected is not one that is used by the application, any SQL Server security update should be tested because some of your customers will probably apply it.

Securing SQL Server is not just a function of locking down databases and who has access to data; you also need to secure Windows Server and the other components that SQL Server relies on for core functionality within the Windows platform. Likewise, testing security updates for a SQL Server application also necessarily includes testing security updates for the rest of the platform. Working from the SQL Server layer down, the next component worth evaluating security updates for with a SQL Server-based application would be the .NET Framework. Starting with SQL Server 2005, the .NET Framework is a required part of the SQL Server installation and is used within SQL Server for many essential functions. Therefore, any security vulnerabilities to the .NET Framework should be seriously considered for testing because they may affect SQL Server. As with SQL Server, all relevant .NET Framework security updates should be fully tested.

The layer below the .NET Framework is Windows Server. Most security updates and bulletins will be related to Windows because it now includes many common components and features, including Internet Explorer and Internet Information Services. Not every security update may be applicable to your particular usage of Windows because not all components or features may be enabled.

For example, say that in your application, you recommend a dedicated server for the SQL Server implementation. It is not a desktop, nor is it an application server. The server is behind a firewall and has no direct Internet connectivity. If a particular security update is related to an Internet Explorer vulnerability, as an ISV, that may be one that you do not add to your testing because it would in no way affect your application. A customer may need to test it because they made the server Internet-facing, but as far as you are concerned, it may not be applicable to a typical implementation of your application.

Outside of the direct SQL Server-related components, if your application is using any other Microsoft products such as System Center Operations Manager (SCOM), BizTalk Server, SharePoint Services, and so on as part of the overall application solution, any security bulletins for those also need to be considered for testing.

> **NOTE** While this is a Microsoft-focused paper, if you are using other components in your application (Java, third-party plug-ins, and so on), regardless of what platform they run on, you must keep up-to-date with the security for those as well and apply the same methodology and principles as you do for the Microsoft products.

One big factor in the requirement to validate nearly every security update is that certain industries are highly regulated and must adhere to certain standards. ISVs should know their target industries well and make it easy for customers deploying their software to meet any applicable regulatory requirements.

After you determine the security updates to test, the testing methodology is as follows:

- Set up a test environment with the base configuration.
- Apply the security update.
- Test the application for both functionality and performance.

Why test for performance and not just functionality to ensure that the application isn't broken? For example, if the security update is related to something such as networking, and iSCSI is in use, a change to the network stack could clearly affect I/O performance. Again, as an ISV, you need to ensure that whatever update is applied, it does not cause a regression in performance. Customers tend to notice the bad long before they notice anything good, and a dip in performance would certainly qualify as something they do not want to see.

The value add as an ISV is not seeing if the specific security update fixes a problem; it is to ensure that your application will continue to function properly if the update is applied to a server or component that is running on your recommended configuration. Microsoft has already tested the security update to ensure it fixes the stated problem, but Microsoft does not know your application. You do not code Windows or SQL Server. The customer is relying on everything to work seamlessly.

When it comes to testing security updates, it is strongly recommended that you use an automated test harness. Automated testing not only will reduce the cost of these short test validations each month, it will also ensure that this standard set of tests completes in an agreed-upon amount of time. Remember that your customers are receiving the same security bulletins from Microsoft and probably want to secure their systems by applying these updates. The best practice for ISVs is to establish a default time period, often one to four weeks, during which your company will perform any relevant testing on that month's security updates. In the rare case that your testing uncovers an issue, notify your customers to delay application of that specific security update before this time period expires. Otherwise, your customers will know they are safe to apply these security updates during their maintenance windows.

> **TIP** You can automatically receive a list of the monthly updates by visiting the Microsoft Security Advisories Website and signing up for one or more of the following:
>
> - Subscribe to the Security Advisory RSS Feed
> - Receive the Windows Live Alert: Technical Security Advisory Alerts
> - Receive an e-mail: Microsoft Security Notification Service: Comprehensive Edition

Developing a Supportability Roadmap

One of the most important things you can do as an ISV is to create and maintain a *supportability roadmap* for your application. A supportability roadmap should be a publicly available guide – whether it is a picture, document, or Website – that lets customers know what versions of your application are currently supported, along with the versions of the underlying platform(s) that you support for each version. The roadmap should also clearly indicate when you will consider a version of your application out of support.

Putting together a roadmap is not a straightforward task. It must take the following into account:

- The proposed date for an application release
- Underlying platform version(s) chosen to be the base version(s) to support
- Underlying platform supportability dates as well as known upcoming releases
- Industry tolerance for application updates or upgrades

A well thought out roadmap should:

- Enable customers to develop their own support and lifecycle policies by understanding your application lifecycle (as well as those of the underlying platforms)
- Ensure that both your company's applications and customer deployments keep moving forward, and that the gap between implementations is a reasonable one to minimize risk to your company, the underlying platforms, and customers
- Enable a steady revenue stream for your organization that is fairly predictable

Why should you care about the various SQL Server releases over time? In the case of SQL Server, even if the customer never uses Microsoft for support, the customer is still using the SQL Server database platform. It is always in everyone's best interests to have your customers on a platform that is

sustainable – even if they are an indirect customer of Microsoft because you are selling them the entire solution and providing most, if not all, of the support.

The other reality as of May 2010 is that SQL Server 2008 with SP1 is now the base version of SQL Server 2008 (SQL Server 2008 RTM was retired in April 2010), and new deployments should be able to take advantage of it. Your potential customers may not be happy if they want to implement SQL Server 2008 (or have even standardized on it by this point) and are told they need to use SQL Server 2005. In addition, keep in mind that SQL Server 2008 R2 has RTM'd and is now shipping, and both SQL Server 2005 SP4 and SQL Server 2008 SP2 have been announced. Will your company support all those in your current application versions? If so, what is the time frame to support them in relation to when Microsoft has scheduled ship dates? If those SQL Server releases will not be supported for current versions, what future versions of your application will support them? These are exactly the types of details you need to figure out for your supportability roadmap and communicate to your customers. The supportability roadmap for your application should not be a secret.

If you want a good example of a supportability roadmap, look no further than Microsoft. Microsoft maintains a Lifecycle Website that details the general policies as well as the supportability dates for each product (including things like SPs). This Website should be used when planning for your products and what versions of Microsoft products you will support in a particular version of your application. It also explains the general Microsoft lifecycle policy and how it works.

According to Microsoft's Lifecycle Website, the supportability roadmap for SQL Server is as follows:

- 5 years of mainstream support and 5 years of extended support (for a total of 10 years) for each major version. For example, SQL Server 2005 mainstream support ends on April 12, 2011, and extended support ends on April 12, 2016.
- A given SP level is supported for 12 months (1 year) after the next SP is release. For example, SQL Server 2005 SP1 support expired on April 8, 2008, which is just over a year after SQL Server 2005 SP2 was released (February 19, 2007).

Your supportability roadmap should include similar details for the version(s) of Windows (or any other required Microsoft products) you support or require for your application servers. While this is a SQL Server-focused paper, chances are that your application will most likely need additional Windows Servers somewhere else in the application's ecosystem.

In theory, putting a supportability roadmap together is a matter of synchronizing dates for your application schedule with the release or supportability dates for a product such as SQL Server. The reality is that the process isn't quite that simple. You cannot pick a date for your support of a product version without knowing your testing resources (the number of testers, available test machines, how many other testing activities are going on in parallel, and so on) as well as how long it usually takes to do the testing to certify a version. Without those inputs, you could prematurely announce dates and have to change them. That has three possible outcomes: 1) You look bad, 2) indirectly you make a product such as SQL Server seem like it is not ready for use, or 3) a combination of both. Therefore, a supportability roadmap is largely dictated and should be owned by your testing organization.

> **TIP** When building the supportability roadmap, always start with the biggest changes and work your way down to the smallest changes. This advice applies equally to the changes that you are shipping for your own application as an ISV, to the changes shipped for SQL Server, and to the changes shipped for the underlying Microsoft components such as Windows Server.

As evidenced by the different kinds of updates that can be applied to SQL Server, updates can be much more granular than an SP. Even if you do not formally test every hotfix or CU, a customer may need or want them (especially a CU; they would need to know about or be recommended to install a specific hotfix). So while it is important to work SPs into the roadmap, you should consider devising a formal policy regarding hotfixes, CUs, and security updates. Supporting CUs should not place a significant burden on your organization because they are predictable, and if you have a testing harness, it should be easy to see if a CU breaks your application or not.

Managing an application lifecycle is a long-term investment. It is a safe assumption that no matter what base versions of SQL Server or Windows that you select, there is already another in development or soon to be released. It is also safe to assume that customers may be one or two versions behind even what you are planning to support in the application. This becomes a delicate balance that you have to manage carefully.

> **TIP** As part of your roadmap and an overall versioning strategy, keep in mind that some features in SQL Server may be deprecated. As each new version is released, the best way to determine if a feature has been deprecated and when it will be removed from the product is to refer to SQL Server Books Online. Books Online is continually updated, so you should as a matter of policy make it a point to check every few months to see if a new version is available.

Roadmap Example 1: Best Case Scenario

Let's look at an example that demonstrates a methodical approach to creating a supportability roadmap for a SQL Server-based application that takes into account an ISV's release timeline and testing cycles as well as Microsoft's release cycle and the Microsoft lifecycle policy.

As shown in Figure 2 below, SQL Server 2005 was released on January 14, 2006. In this example, the initial version of your application was slated for an October 2006 release date, so the initial SQL Server version that was planned for support was SQL Server 2005 RTM, which was what was being tested. At that point, SQL Server 2000 support was not considered because you wanted some of the SQL Server 2005 enhancements to the Database Engine that benefitted your application.

SQL Server 2005 SP1 shipped on April 18, 2006. Because development and all other application plans had been put in place by then, the release of SP1 was too late in the development cycle to work into being supported for your October release. The problem is that with the release of SQL Server 2005 SP1, support for SQL Server 2005 RTM ends in 12 months, according to the Microsoft support lifecycle. That only gives you a short window to officially support SP1; otherwise you put your company, the application, and your customers at risk. So in the timeline, you need to immediately have a post-RTM testing cycle to ensure that SP1 is fully supported. Since most of the major testing work was done for RTM, it should not be hard to reuse those tests to certify the application against SP1.

While you are finishing up the first version for October, others are starting to plan the second version of your application to be released in early 2009. That date was chosen because in the industry you serve, customers have a tolerance for new application releases between every 2 and 4 years. At the time of initial planning for Version 2, there was no formal announcement of any new major version of SQL Server or SPs. Shortly after development was started, however, SQL Server 2005 SP2 was released, so your team decided it would be the base version supported by the application.

Fast forward a few months. SQL Server 2008 has been announced and the first CTPs start appearing. Since the release date for the application is slated for early 2009, is there enough time to incorporate SQL Server 2008 into the test matrix for Version 2? This is debated, and your company decides to support SQL Server 2008 within 12 months of its release, meaning SQL Server 2005 SP2 is still going to be the base version tested and supported. SQL Server 2005 SP3 was released too close to the ship date, but because SQL Server 2005 SP2 is fully supported and stable, testing and certification is planned for 3 months post-RTM of the application because the amount of testing should be relatively trivial.

Now take into account the end of life for SQL Server 2005 SP2, which is January 12, 2010. While that date may seem a long ways away when you are doing planning for Version 2 in 2008, it sneaks up on you very quickly. It is imperative that you meet the schedule to not only support SQL Server 2005 SP3, but *also* SQL Server 2008 so that you have valid options for your customers when it comes to SQL Server.

What does this example show? While it certainly will take a lot of thought and persistence to keep up with releases and their expiration dates, you keep moving forward, and you can keep your customers moving forward. There is something to be said for stability, but remaining on a 5- or 10-year-old platform can only do harm in the long run. For example, at some point, updates such as security patches may no longer be made – that puts both the ISV and the customer at risk. The hardest part is lining up your time frames with Microsoft's to provide the best solution for everyone involved.

Think about it from another point of view: Having your customers stay current also has a potential benefit of revenue for everyone involved. Customers do not want to be on a continual treadmill of upgrades and platforms. Many customers would like to get 3 or more years out of their current software and platforms if not more. If you enable them to seamlessly upgrade the back end (SQL Server or Windows) and the front end (your application) – perhaps even independent of each other – both Microsoft and your company benefit long term. One of the biggest problems many software companies face is getting their customers to upgrade or migrate to newer versions, and a big part of that is the pain associated with upgrading. If you eliminate that pain, you will have satisfied, long-term customers.

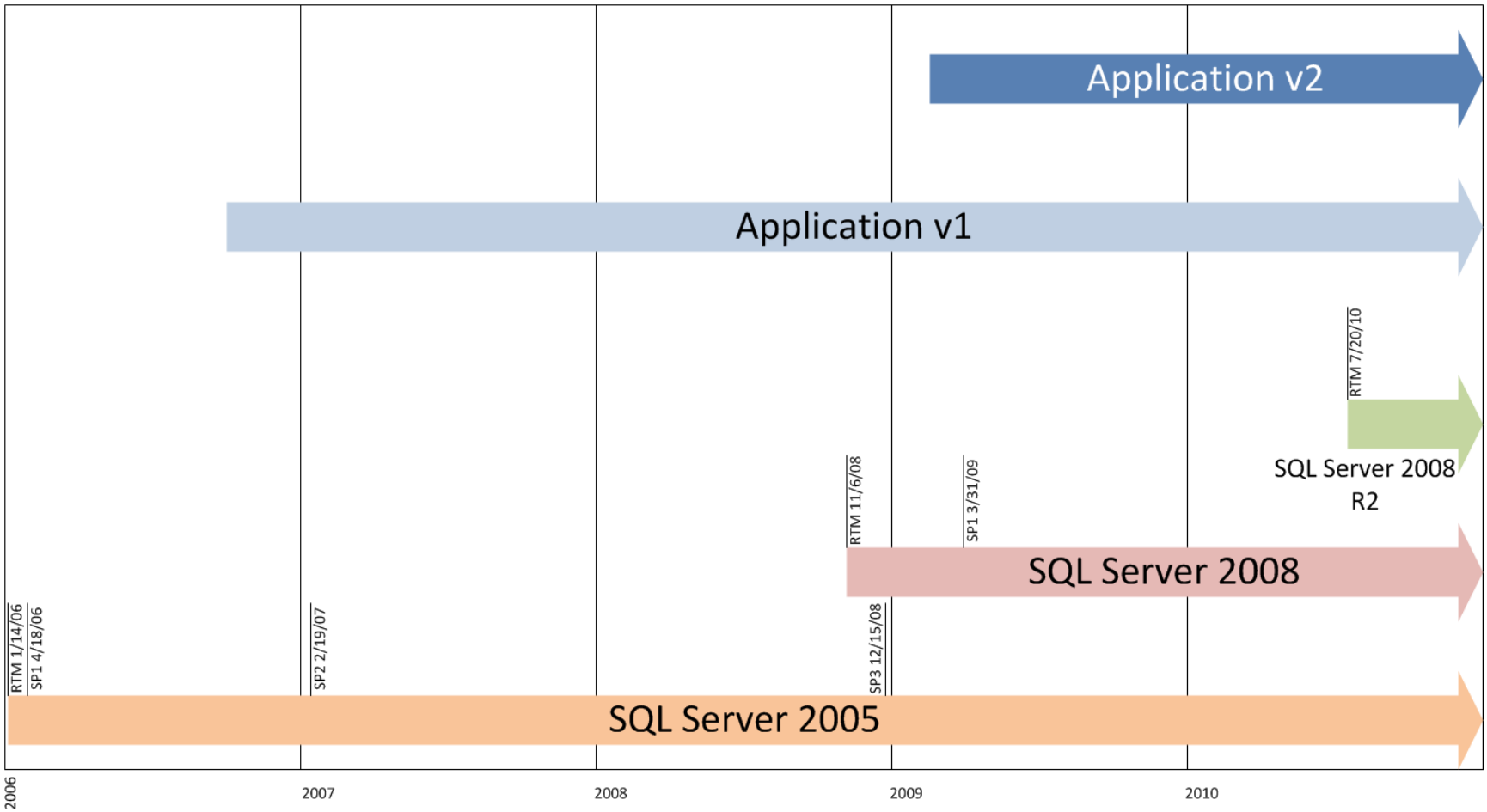

Figure 2 - Timeline of planned application releases versus SQL Server release dates

Roadmap Example 2: Worst Case Scenario

You shipped Version 1 of your application in December 2003 and supported SQL Server 2000 SP3a. No new version of the application shipped until September 2007 with support for SQL Server 2000 SP4. Although SQL Server 2005 was already at SP2 by that point, an executive decision was made to still only support SQL Server 2000. Your company clearly communicated this to the application's customers, explaining that you could not yet support SQL Server 2005 and that you would let them know when this newer version was certified for the application.

You would be asking customers implementing your software for the first time in late 2007 or early 2008 (and beyond) to implement a *7-year-old* platform, when there was a newer one already released and stable and another coming on its heels later in 2008. If you do not even announce support for SQL Server 2005 until sometime late in 2008 or early 2009, you are easily going to be at least two major versions behind by the time SQL Server 2008 R2 ships in 2010 (although from a timeline perspective, SQL Server 2008 R2 would not be known at this point). That is quite a bit of testing and change to have to deal with. While you may be constrained (be it by development or testing resources), falling that far behind can only hinder development, not help.

By going down this path, you also put customers in a bind. They may have already standardized their new deployments for SQL Server on SQL Server 2005, yet with your application, you are forcing them to take a step back with no future upgrade path defined. While SQL Server 2000 SP4 mainstream support wasn't slated to end until April 8, 2008, do you want to put your customers at risk in this manner?

This scenario also represents a potential and significant loss of revenue. If customers perceive your application as not supporting their needs for whatever reason – including platform support – they may look for an opportunity to replace you. Perception is always reality, even if it is not true. Such an outcome also means lost revenue for any partner (including Microsoft) that is part of your application's ecosystem.

Conclusion

Developing and testing applications is an ongoing concern for all ISVs. An application release is just a single point on a line that continues until development ends and a product is completely retired. That means that anything used or related to the application – including its underlying platforms – must be tested continually for every release of the application. It is not Microsoft's responsibility to ensure that your application works with SQL Server 2008 R2 or any of its subsequent updates; it is yours. Testing mitigates risk. Testing ensures stability. Providing documented lifecycles instills confidence and allows customers to plan accordingly. Failing to perform any one of these tasks can mean much more work to do later on, or even a potential loss of revenue for everyone involved. The goal of any ISV should be to provide customers with the best experience possible, including smooth upgrades. Using the information in this paper, you can make that happen.

Testing and Supportability Roadmap Resources

White Paper Section	Link	Description
	http://www.microsoft.com/sqlserver/2008/en/us/	SQL Server Website
	http://www.microsoft.com/visualstudio/en-us/products/2010-editions/test-professional	Visual Studio Test Professional 2010
	http://msdn.microsoft.com/en-us/library/ff649520.aspx	Prescriptive Architecture – Testing Methodologies
	http://technet.microsoft.com/en-us/library/cc506049.aspx	Microsoft Operations Framework
Service Pack	http://support.microsoft.com/kb/955706	KB955706 – List of the bugs that are fixed in SQL Server 2005 Service Pack 3
Service Pack	http://support.microsoft.com/kb/968369	KB968369 – List of the bugs that are fixed in SQL Server 2008 Service Pack 1
Service Pack	http://download.microsoft.com/download/2/B/5/2B5E5D37-9B17-423D-BC8F-B11ECD4195B4/WhatsNewSQL2005SP2.htm	What's New in SQL Server 2005 SP2
Service Pack	http://support.microsoft.com/kb/981355	KB981355 – Cumulative Update package 1 for SQL Server 2008 R2
SQL Server Versioning	http://support.microsoft.com/kb/810185	KB810185 – SQL Server 2000 hotfix update for SQL Server 2000 Service Pack 3 and 3a
SQL Server Versioning	http://support.microsoft.com/kb/894905	KB894905 – Cumulative list of the hotfixes that are available for SQL Server 2000 SP4
SQL Server Versioning	http://support.microsoft.com/kb/913090	KB913090 – A list of the bugs that have been fixed in SQL Server 2005 Service Pack 1
SQL Server Versioning	http://support.microsoft.com/kb/937137	KB937137 – The SQL Server 2005 builds that were released after SQL Server 2005 Service Pack 2 was released
SQL Server Versioning	http://support.microsoft.com/kb/960598	KB960598 – The SQL Server 2005 builds that were released after SQL Server 2005 Service Pack

		3 was released
SQL Server Versioning	http://support.microsoft.com/kb/956909	KB956909 – The SQL Server 2008 builds that were released after SQL Server 2008 was released
SQL Server Versioning	http://support.microsoft.com/kb/970365	KB970365 – The SQL Server 2008 builds that were released after SQL Server 2008 Service Pack 1 was released
SQL Server Versioning	http://support.microsoft.com/kb/957826	KB957826 – Where you can find more information about the SQL Server 2008 builds that were released after SQL Server 2008 and about the SQL Server 2005 builds that were released after SQL Server 2005 Service Pack 3 and after SQL Server 2005 Service Pack 2
SQL Server Versioning	http://support.microsoft.com/kb/975977	KB975977 – Cumulative Update Package 5 for SQL Server 2008 Service Pack 1
Testing Security Updates	http://www.microsoft.com/technet/security/advisory/RssFeed.aspx?securityadvisory	Sign up for the security advisory RSS feed
Testing Security Updates	http://signup.alerts.live.com/alerts/jump.do?PINID=3274	Sign up for the Windows Live Alerts
Testing Security Updates	http://go.microsoft.com/fwlink/?LinkId=51352	Sign up for the security alerts via e-mail
Testing Security Updates	http://www.microsoft.com/technet/security/advisory/default.mspx	Microsoft Security Advisories Website
Developing a Supportability Roadmap	http://support.microsoft.com/common/international.aspx?RDPATH=dm;en-us;lifecycle	Microsoft Support Lifecycle Website