How important is backwards-compatibility? [closed]

Asked 16 years, 1 month ago Modified 16 years, 1 month ago

Viewed 3k times

6

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 12 years ago.

When you are considering an upgrade in a development tool, how important is backwards-compatibility to you? Would you still purchase Visual Studio 2010 if it required significant changes to your source code? Where is the tipping point for you in terms of trading backwards compatibility for new features?

backwards-compatibility

Share

Improve this question

Follow

edited Nov 20, 2008 at 19:49

Scott Dorman

42.5k ● 12 ● 81 ● 112

asked Nov 20, 2008 at 19:47

JosephStyons

58.6k ● 64 ● 167 ● 237

Could you as an experienced guy explain me why this is not a community wiki post? – Paul Kapustin Nov 20, 2008 at 19:56

Because it's a legitimate question? Community Wiki is not applicable to everything, and I don't fault kogus for not opting into it. – Kyle Cronin Nov 20, 2008 at 20:05

7 Answers

Sorted by: Highest score (default)

13

While you asked this from the point of view of the developer, I think it would be a more interesting question in reference to the software you develop. So I'm going to answer that question instead. :)

Hardware and software that's backwards compatible (and, more importantly, future-compatible) provides a sense of security to your users, especially when buying or upgrading platforms like Windows. If nothing else, Windows is known for its meticulous attention to backwards compatibility. You can run programs written over a decade ago on Windows Vista with only minor

problems, provided they were "well written" (i.e. don't use undocumented APIs).

On the other hand, strict attention to backwards compatibility can tie your hands when you're attempting to introduce new features or to revolutionize the platform. Apple knew it had a dying OS, and in one of its most daring moves, it purchased NeXT and decided to make NeXTSTEP the new MacOS. One of the key things that sold people on the transition was the backwards-compatible layer Classic. Again when Apple decided to switch to Intel chips, a mechanism for running PowerPC apps on Intel called Rosetta, along with the Universal Binaries, allowed people to freely move between PowerPC and Intel without fear of application loss.

One interesting thing is that with the transition to Intel the Classic environment disappeared, but nobody really cares because they had the preceding 5 years to transition away from Mac OS 9. So it is possible to eventually drop support for legacy systems as long as you have an easy way to migrate to the new system and give your users ample time to do so.

answered Nov 20, 2008 at 19:59

Kyle Cronin

79k ● 45 ● 151 ● 167

In a development tool, if it doesn't provide total backward compatibility with my previous code, I won't buy it and I doubt anyone would. Frankly, there's no point. If I already

3

have a compiler that works to build my source code into executable code that works for me, then I'll use that. Why bother changing my code to conform with what is obviously to the toolmaker not a standard? If they force source code changes from one version to the other, why would they bother to make the next version compatible?

100% backwards compatibility with source is a requirement. The only situation where this isn't a total requirement is when the incompatible bits are extensions; i.e. API changes that are specific to the tool, such as Eclipse plugins, etc. Even then, I'd like compatibility, but I realize that it can't be completely expected. But if you provide an API for base application / tool development, and can't be bothered to maintain compatibility; well, then, you clearly aren't serious about your tools, and I won't pay serious money for them.

Share Improve this answer

Follow

answered Nov 20, 2008 at 19:56

Paul Sonier

39.5k ● 3 ● 79 ● 117

4 I suspect you don't really mean that as absolutely as you put it. For instance, C# 2 isn't 100% backward compatible with C# 1. It's close, but it's not 100%. By your logic no-one should have gone with VS2005 or VS2008. – Jon Skeet Nov 20, 2008 at 20:09

I also disagree. I've helped several people migrate from eVC to Studio and code that "worked" under eVC wouldn't even compile under Studio because the compiler is far improved. Once compiling, the newer runtimes also found lots of

problems with the code. So even after weeks of work it was worth it. – ctacke Nov 20, 2008 at 20:41

2

For home projects, backwards compatibility really isn't important. For the office/enterprise, it's absolutely critical.

Share Improve this answer Follow

edited Nov 20, 2008 at 20:03

answered Nov 20, 2008 at 19:56

Brian Knoblauch

21.3k ● 16 ● 61 ● 96

1

It depends on what environments you need to support and what third-party tools are used that may or may not be compatabile.

For example, where I work we upgraded everyone to VS2008 from VS2005 except for our BI group as the SQL Server BI tools were not compatible with VS2008. Once they were updated, they upgraded to VS2008.

When looking at VS specficially, keep in mind that VS2008 can target .NET 2.0, .NET 3.0, and .NET 3.5. The trick is to realize that it actually targets .NET 2.0 SP1 and .NET 3.0 SP1. As such, upgrading the IDE should not require you to make changes to your code.

Share Improve this answer

answered Nov 20, 2008 at 19:51

Follow

Scott Dorman
42.5k • 12 • 81 • 112

1

In genreal if you are developing a platform which will be constantly used by a lot of other users to build their own products, and you plan to develop the application for a long time then it is important. See the PHP, Python, Eclipse and other open source projects who put a lot of importance into backwards compatibility. It is also importan when developing services or other open apis used in the n-tier architecture. You can have all applications in an enterprise breaking all the time when you change your services.

Now, If you are building a shrink wrap application or a bussiness application then it is not so important, beacuse each version is separate from its predecessors.

Share Improve this answer

Follow

answered Nov 20, 2008 at 19:54

Nikola Stjelja
3,687 • 9 • 39 • 47

1

Since lot of changes are happening in the Software and Hardware field, I think it is a good idea to be open for new changes and better tools while you architect your solution. For example we didnt have multi core processors and high end graphics cards or network card back in 90s, So naturally the optimization goals of the compilers and tools were different. But at the same time

Visual studio like tools are doing their best to accommodate the old frameworks and apps.

I think if we are looking forward for a better world, we should be open to a constant change until this industry is super matured. (May not happen in our life time though :))

Share Improve this answer

Follow

edited Nov 20, 2008 at 20:04

answered Nov 20, 2008 at 19:52

Jobi Joy

50k ● 20 ● 110 ● 120

0

Define "significant changes". I'd go for it if the changes could be made with a carefully crafted "search & replace" even if they were extensive.

However, that's what *I* would do. Any company I've worked for would balk at any changes to existing code.

Share Improve this answer

Follow

answered Nov 20, 2008 at 19:51

James Curran

103k ● 37 ● 185 ● 262