Finding alternatives in developing software

Comparing free development tools and libraries with proprietary ones

Martin C Brown

Developing software within the free software model can be achieved with all sorts of different tools, but choosing the right tools can make a big difference to the success of your project. Even if you are developing a proprietary solution, there are benefits to using free software tools to achieve it. But what free software tools are available? In this article I'm going to look at the development tools available, from languages and libraries to development environments, as well as examining the issues surrounding the use of free software tools by comparison to their proprietary equivalents.

Development languages

The languages used in your development are generally free. Despite attempts by companies to protect their language specifics from the development community, the majority of those used today fit into the free software category. A lot of these languages are free not necessarily because they were outwardly designed that way but because, like human vocal languages, there was a need for an agreed set of standards.

> Choosing the right tools can make a big difference to the success of your project

Unlike the other components in the development process, the language you use must have a very specific set of semantics. This is because, as the basis and groundwork for your applications, there must be a commonly agreed structure. Now imagine if everybody that supposedly used C used a slightly different format for specifying a function. Essentially the structure of the language wouldn't be standard, and if the structure wasn't standard then we'd end up with thousands of different languages that might differ apart from the use of a semicolon to terminate lines.

Because of the need for standards, specific languages have specific rule sets and by their very nature these rule sets need to be published for people to program in that language. Ultimately, this means that most languages are technically free software or there is a free software solution for that language available.

Returning to the topic of the available free software languages, off the top of my head, I could probably list 15 commonly used languages that are essentially free, starting with C. C was originally developed as part of the Unix development process and the design specifics and semantics of the language were published; thus we have an open standard and one of the languages with the largest user bases and associated tools, libraries and environments. Most operating systems and their components are written in C and that includes both free software solutions like BSD and Linux and commercial offerings such as Solaris and Windows. This has the benefit that it's possible to easily extend the operating system by writing more C applications without the need for brand new libraries for basic functionality. Even more ironic is the fact that most other languages now tend to also be written in C and, in most cases, the libraries for those

Fig. 1: The Mono Project, .NET without Microsoft

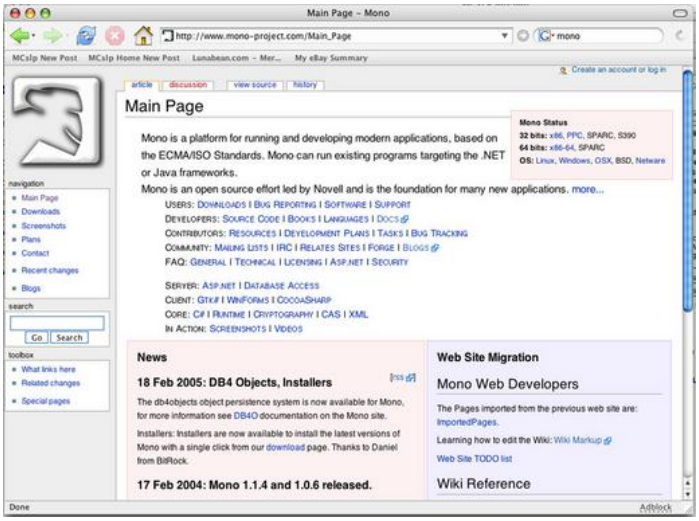
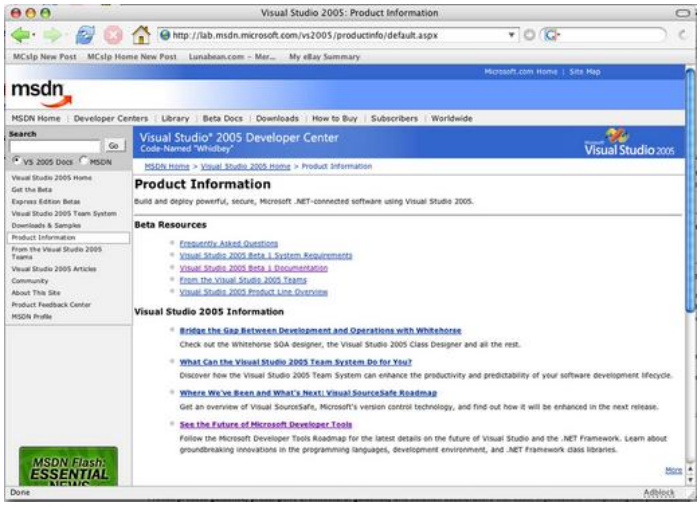


Fig. 2: Visual Studio .NET 2005 (Whidbey), Currently in Beta



languages ultimately interface to the C equivalents within the operating system.

Other languages which are free including C++, Perl, Python, PHP, Ruby, Pascal, Modula-2, ML, Fortran, Cobol, Lisp, Smalltalk, Tcl, awk/gawk and many others.

Now there are a couple of notable languages which I haven't mentioned, specifically Java and the 'sharp' series from Microsoft, including C# and J#. The first, Java, has free software compilers available for it (notably gcc), but the language itself is a proprietary project of Sun Microsystems. If you want to make full use of the Java language you really need the full Java libraries and associated compatibility, and that only comes from the Java runtime and development environment, which comes from Sun.

Languages which are free including C++, Perl, Python, PHP, Ruby, Pascal, Modula-2, ML, Fortran, Cobol, Lisp, Smalltalk, Tcl, awk/gawk and many others

Similarly, C# and J# are projects developed and managed by Microsoft. Both are object-oriented languages based on C, C++ and Java (and I really do mean a combination of all three, to varying degrees, in each case), which are specially designed to work with the .NET environment.

Like Java there are free software solutions available; the Mono project have developed a C# compiler and an almost complete .NET library suite which enable you to write, build and deploy C#/.NET applications on Unix and other platforms. But, C# and J# are proprietary languages with the development effort driven by Microsoft.

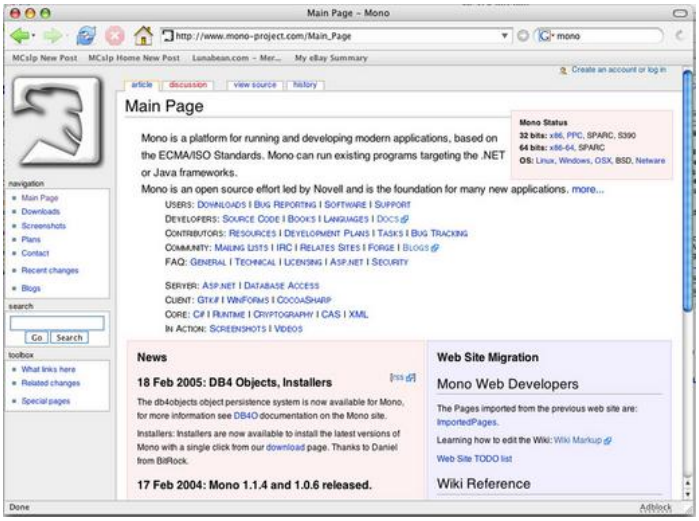
As you can see, on the whole the development language - the core of any development process - is a free software component, largely by its very design and existence. Now let's move on and see the effects of other components within the development process.

Development tools

One of the key reasons I write and develop free software is because of the tools and environments available to me. After 15 years as a developer, 12 of them professionally, I still use the same environment I've always used; emacs, make or Ant and whatever compilers or tools I need such as C, Perl or Java.

This might seem a little archaic compared to some of the tools that are available; for example when developing under Windows surely the best choice is one of the various integrated development environments (IDEs) such as Microsoft's Visual Studio. The problem with these tools is that they expect you to work in a specific way and they often tie you down to that way of working. Within Visual Studio for example you have to use their editor, particularly if you want to use features like code completion and expan-

Fig. 3: The Eclipse IDE in action



sion. Visual Studio also expects you to use the libraries and tools in the .NET framework for your development, and it also provides its own build and testing environment. All of these items become constricting components, designed to keep you using the IDE for all of your development and therefore allowing the proprietary developers to force you down a particular development and deployment route.

The benefit of emacs is that it's an environment I'm familiar with, one which is supported on multiple platforms and one which, with a little more command-line work than others, provides a flexible solution irrespective of the language, and more importantly, the platform on which I am working.

For those that want IDEs there are numerous potential choices available in the free software space. Although emacs is not best known as an IDE, it actually has most of the IDE components available, including project management, source control and building tools. For many of these components, the functionality is actually provided by another free software tool. For example, when building an application you could use GNU make, source control can be managed by the Concurrent Versioning System (CVS) or Subversion and the actual compilation can be processed by gcc.

IBM tried an interesting approach to the idea of proprietary development tools and turned it into one of the best known and acknowledged free software development environment. The Eclipse project was designed by IBM as their new IDE and was set to replace their existing development environments. Eclipse is entirely designed on the basis of plug-ins. There is a very minimal "kernel" to the Eclipse platform, everything else is essentially a plug-in to this kernel, extending and expanding the functionality of the application as it goes. The major benefit of the extensible architecture is that it makes it easy for a developer to modify and adapt the platform to work the way he or she likes. It also means that Eclipse can be used to develop any application for any language and environment. This eliminates one of the major complaints about most proprietary development platforms.

The libraries you use in your application are as important as the application itself. One of the key issues with libraries is that they will affect where your applications can be used and displayed

At a cost of US$30 million, Eclipse didn't come cheap but the result was an incredibly flexible and extensible environment. After the initial development, Eclipse was released to the community and is now developed in the same manner as other free software projects like emacs, Linux and GCC. IBM still use Eclipse as the basis for their commercial development applications - the new Rational Software Development Platform is based on the Eclipse environment and the IDE that is used within the WebSphere development suite (WebSphere Studio Application Developer) is an Eclipse application. Both suites make heavy use of the plug-in architecture of the Eclipse environment to add additional functionality and value to the products.

Eclipse is not the ultimate utopia. Ironically for a system that has been designed to be so flexible and open, it is actually written in Java. Also, Eclipse works on the basis of specific projects and workspaces for controlling the development process.

Development libraries

The libraries you use in your application are as important as the application itself. One of the key issues with libraries is that they will affect where your applications can be used and displayed. This is the issue of portability, and I'll get on to it later in this article.

The other issue is that they affect development from the point of view of the functionality of your application. Most

Fig. 4: The Simple DirectMedia Layer

proprietary libraries are developed with very specific goals and ideals in mind, often to fit the interests of the developers in question. For example, graphics libraries are often developed with the ideals of the graphics hardware in mind, rather than providing an easy to use and flexible graphics library for general use. Using a proprietary library in this situation leads to the development of an application that only works with very specific hardware and that in turn leads to a closed and limited application.

There are, in fact, a wide range of free software libraries available that cover just about every avenue you could want. For example, for multimedia the key technology on Windows would be DirectX, a proprietary solution from Microsoft that provides full multimedia capabilities, from playing music and video to displaying 3D graphics for games. DirectX is only supported on the Microsoft Windows platform but does have the advantage of providing an abstraction layer to a wide variety of underlying hardware.

The Simple DirectMedia Layer (SDL) library provides similar functionality, but does so through a free software model and provides the same functionality across a wide range of platforms. You can write an SDL application that operates on a number of platforms with no changes to the code; something impossible with a proprietary solution such as DirectX.

Better still - and this particularly applies to libraries like no other part of the development process - if there is some element of the library which does not fit in with your plans but which you think would be useful to others you can expand and add the functionality and make it available to others.

Other free software libraries, that can be used in place of proprietary equivalents, also exist. At the basic level, the GNU C library (glibc) provides the core functionality for the C language and libstdc++ is the standard C++ library.

There are database libraries, for example Berkeley DB in place of Microsoft's Jet and numerous networking libraries in place of the proprietary networking solutions.

Portability

I touched briefly upon this subject earlier in regard to the potential for lock-in with libraries. Surprisingly few companies think about this when developing their software, and then get stuck when the vendor drops support for the library. Developers relying on proprietary solutions also find that the lack of flexibility becomes a problem.

Furthermore, the effect can be felt elsewhere in the development process. For example, develop an application using Visual Studio .NET and the chances are that porting your application to a different development environment will be incredibly difficult, because all of your code will be designed to work within the .NET framework and use functions and facilities only available to your .NET developed applications.

Worse still, your application not only becomes locked-in to your development environment, it also, ultimately, becomes locked into your development platform. Creating an easily portable piece of code that can be used on Windows and, say, Mac OS X becomes a mammoth task. There are obvious differences between the two platforms, but the common elements within an application clearly remain the same.

Take everybody's favourite proprietary application, Microsoft Word. There are versions available for Microsoft Windows and Mac OS X and the two applications on the two platforms share about 90% of the same functionality (with the rest being taken up by differences like Entourage and the obvious interface specifics). Now you can't develop Mac OS X applications within Visual Studio .NET, but migrating what is core code between the two platforms can't be an easy task.

As a developer, how do you get round this?

Well, using free software is an obvious answer. In general, free software solutions don't lock you in to a particular style

of working and they will be based on open standards and interoperability. If you have developed an application within the GNU framework, for example, then portability is a key part of the development process. GNU tools like autoconf, configure, make and gcc exist and are used for the purposes of making software available on as many platforms as possible.

If you are a developing a free software project then using free software tools is vital because of the ease with which you will be able to share information and the project itself. Imagine trying to get everybody to contribute to a project if they had to obtain a closed set of tools to do it. The chances of people getting reliably involved are slim, and probably non-existent. The more people you can get to aid in the development of your project, the better it will become; and using other freely available tools is the way to do it.

Spotting a free software development environment

I should highlight the fact that not all development environments were created equal, even free software ones, and the specifics of the environments can be difficult to spot. For an excellent example of this look no further than Mac OS X.

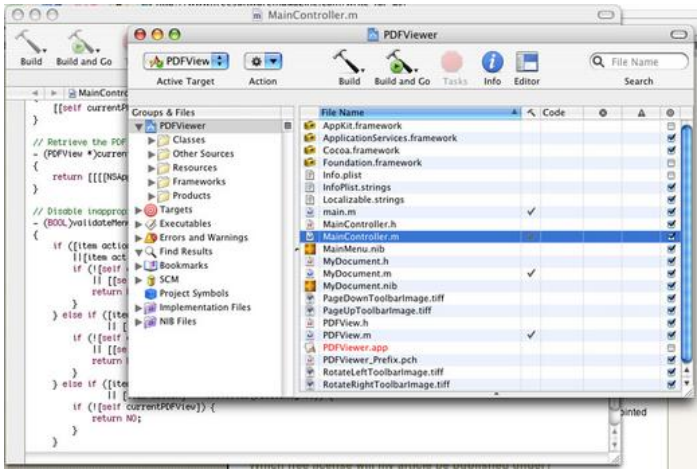
At a fundamental level, Mac OS X is a proprietary operating system based on a free software platform, and its development environment is based on a free software platform, supported by a proprietary development tool. Confused?

Well, Mac OS X is basically a combination of the Darwin operating system and the Cocoa windowing environment that gives OS X its look and feel. Darwin is based on the BSD operating system, a free software project and one of the oldest.

The main development environment on Mac OS X is Xcode. While Xcode is a proprietary IDE designed to work only on the Mac OS X operating system, its underlying code and toolsets are based on the GNU suite. Your applications are compiled using gcc, linked using GNU ld and rely on some of the libraries provided by the Free Software Foundation.

What we have here is a situation where you can develop software, for either a proprietary environment or a generic Unix environment, using a proprietary IDE that builds and compiles applications using a free software development environment.

Fig. 5: Xcode, the proprietary IDE based on free software tools



Does that make Xcode a proprietary solution? Unfortunately yes, but the key is, that underlying this, is the fundamental use of free software tools and therefore a resounding endorsement of the free software development ideals.

So how do you spot a free software development environment? Plain and simple: research. The chances are, if a development environment uses free software tools then it will shout about it. If it doesn't, it's probably closed and proprietary.

Copyright information

About the author

Martin "MC" Brown is a freelance writer and consultant, he works with Microsoft as an SME, is the LAMP Technologies Editor for LinuxWorld magazine, a founding member of AnswerSquad.com, Technical Director of Foodware.net, and has written books on topics as diverse as Microsoft Certification, iMacs, and free software programming.