



Choosing and Using Open Source Components

Diomidis Spinellis

THE DEVELOPERS OF the SQLite open source database engine estimate that it's deployed in roughly half a billion systems around the world (users include Airbus, Google, and Skype). Think of the hundreds of thousands of open source components, just one click away from you. If you know how to choose and use them effectively, your project can benefit mightily.

Choosing

Say you're looking for a library to evaluate regular expressions, an embeddable scripting language, or an HTML rendering engine, but your search returns a dozen candidates. How do you choose among them? It's actually quite easy: all you have to do is apply a few simple criteria associated with the soft-

ware's legal status, fitness, and quality. Start by considering the open source software's license. Some licenses, like the BSD, Apache, and MIT ones, are quite liberal and basically let you use the software any way you want as long as you attribute the original developer. Others, like the GNU licenses, play well with other software licensed as open source but make life difficult for proprietary offerings. This is especially true if you want to distribute your work to others as a shrink-wrapped package, such as Microsoft Office, or as an embedded software product, like a set-top box. In such cases, the only GNU-licensed components you can easily use are unmodified dynamically linked libraries licensed under the so-called GNU Lesser General Public License (LGPL). You get considerable more leeway with GNU-licensed software if you don't distribute a product but instead offer a service (like Google) or simply use your system privately within your organization.

Next, you need to decide whether you'll adopt the component as a precompiled binary or if you're going to

development, you have the basic capabilities to proceed on your own. In this case, additional factors will affect your choice. What tools and libraries are required to build the software? Is it easy to obtain and maintain them? Are they compatible with the rest of your infrastructure? Is your team comfortable with the component's technology, such as its programming language?

Many components are often available in binary form, but only for widely used platforms, such as Windows or Linux. Building from source lets you port the component to your own platform; if your platform is unsupported, you have to consider the cost of porting. Porting from Unix-like platforms, such as AIX, *BSD, GNU/Linux, Mac OS X, and Solaris, is generally easy. But porting from Windows can be tricky, as is the porting of native GUI applications between proprietary incompatible platforms. Components written in Java or a widely available scripting language travel well—those tied closely to a particular platform spell trouble.

Having narrowed down your search to components you can actually use, your next step involves a beauty contest to select the best one. There's no need to embark on sophisticated research; the quality differences between open source components are either stark enough to be obvious or small and practically irrelevant. Ask colleagues

Will you depend
on the whims of an autocrat?

build it from source. Precompiled binaries shield you from the complexity of building the software, which includes obtaining and maintaining the appropriate tools, libraries, and configuration settings. If you build from source, you buy insurance that no matter what happens with the software's upstream

...continued on p. 95

...continued from p. 96

whose opinion you respect. Work your way down the candidate list by popularity, which you can easily determine from question and answer sites, the number of downloads, or Google's search results. Start by looking at the project's documentation—it's a fast and easy way to separate the wheat from the chaff. Is there a user manual? Is it complete, readable, and well-organized? Can you find technical documentation? Will it help you build and modify the code? Some superb open source products come only with mediocre documentation, but I've yet to see a well-documented project that wasn't outstanding in all its aspects.

Now look at the project's heartbeat, its release history. How often do new releases come out? How recent is the last one? Does the project offer a separate cutting-edge and stable release cycle? A project that appears dormant for years is a bad sign: its developers might have lost interest and abandoned it. Thus, when you need an update—for instance, a new version of your platform requires one—you'll be unlikely to get one from the project's developers. While there, also read the descriptions for each release. Do developers fix existing bugs, or are they just piling on new flashy features? Do they respect their user base, or do they break backward compatibility with each new release? Is the project's direction compatible with yours?

Your next stop involves the family you're marrying into, the project's community. This is important because for many open source projects, the community could be your first and only line of support. Look at the project's contributor acknowledgments, forums, mailing lists, issue tracker, wiki, and commit log. Is there a real community behind the project, or will you tie the knot with a one-man show? Is the community working together as a team or constantly fighting? Do developers cooperate under a well-defined democratic process, or will you depend on

the whims of an autocrat? The project's users are also important. Are they supportive, answering questions, and going out of their way to make newcomers feel welcome, or are they insular, arrogant, and rude?

If the code you're looking for is going to be a strategic part of your offering, you should also examine the project's openness—the ease of pushing upstream the changes that you've made locally. Obstacles here mean you'll face the unpalatable choice between reintegrating your changes with each new release of the project or staying behind in its release cycle. Is there a process for your team's developers to acquire commit rights to the project's code base? Many firms establish a deep relationship with an open source project they depend on by having some of their engineers become part of the project's development team.

If a component passes muster with all the preceding simple tests, the final step involves looking at the actual code. Does it follow a consistent style? Is it well commented? Does it use descriptive names for identifiers and files? Can it be easily built and installed? Is the API well-designed, intuitive, and easy to use? Pick parts of the code you might need to adapt, and see how easily you can understand them. If you find the code too complex, chances are that the problem lies with the code, not with your lack of familiarity.

Using

Once you've singled out the component you wish to adopt, you need to decide how to reuse it. The choices range from copying a few lines into your project's source code base to having a complete system running on a separate box in the datacenter. In between lie the options of reusing complete classes or files, using a prebuilt library, or having your code communicate with a separately running process. Focus on the project's recommended choice, but if you face

trouble, explore the other possibilities.

Although it's tempting, try to avoid modifying the open source code to fit your needs; you don't want to end up maintaining another large component on your own. Instead, satisfy your requirements by tweaking configuration settings or by adapting your own code. If you can't avoid modifying the project's software, keep changes to a minimum, keep them localized, have them follow the project's code style, and contribute them back to the project. Work through your version control system (you use one, right?), and import each new release of the open source project on a separate vendor branch. This allows you to easily reintegrate your local changes with each new release. Then plan how you're going to track the open source project's progress. At the very least, you should handle security updates in the same way you handle your own. Finally, decide whether you'll track a project's stable branch or a cutting-edge one and the process through which you'll stay current.

Walking down the aisle with an open source component is simple, and, if you follow the rules, it will be the beginning of a long, prosperous relationship. 

DIOMIDIS SPINELLIS is a professor in the Department of Management Science and Technology at the Athens University of Economics and Business and the author of *Code Quality: The Open Source Perspective* (Addison-Wesley, 2006). Contact him at dds@aeub.gr.

Post your comments online
by visiting the column's blog:

www.spinellis.gr/tools