



Dr. Dobb's DIGEST

The Art and Business of Software Development January, 2010

Editor's Note by Jonathan Erickson	2
Techno-News Programming from "Scratch" Graphical programming language developed to aid programming fluency.	3
Features 9 Ways To Avoid Open Source Pitfalls by Andrew Till Yes, you can avoid problems and still benefit from open source software. Proactive Management Required by Ralph Priestner Open source can be a powerful tool for cutting costs, improving time to market, reducing risk, and increasing innovation.	5
Selenium: Cross-Browser Website Testing by Alexander Sirotkin Automated testing can make a web developer's life a lot easier	7
SQL Server Solutions in the Cloud by Brent Ozar There are two types of SQL Server solutions in the cloud right now, and they're as different as espresso and instant coffee.	8
Introduction to Functional Programming by John Puopolo Functional programming gives us new ways of developing scalable systems.	11
Bug Opportunity Levels by Peter Eastman A systematic approach to writing robust, bug free code.	14
CPU-Z: Under the Multicore Hood by Gastón Hillar CPU-Z 1.53 delivers more information about multicore CPUs.	17
Columns Conversations by Jonathan Erickson As director of intellectual property strategy for the Linux Foundation and an attorney at Choate Hall & Stewart, Karen Copenhaver knows a bit about open source software licensing.	20
Book Review by Mike Riley Mike Riley examines <i>Android Wireless Application Development</i> by Shane Conder and Lauren Darcey.	22
Other Voices by Pankaj Malviya Why perfecting every pixel in IT applications can be a waste of time	24
Effective Concurrency by Herb Sutter This month, Herb talks about designing concurrent APIs.	25
Flames by Michael Swaine Michael doesn't offer you a prize to wonder about Apple's plans.	27
	30

Top 5 App Dev Changes In 2010

If it's true that change is good, then 2010 ought to be a banner year for software developers, at least according to a new report entitled "The Top Five Changes For Application Development In 2010" (www.forrester.com/rb/Research/top_five_changes_for_application_development_in/q/id/55646/t/2?cm_mmc=Forrester-_-RSS-_-Document-_-55646&src=RSS_2) from Forrester Research. So what kind of change can programmers expect? According to analysts Mike Gualtieri, John Rymer, and Jeffrey Hammond who authored the report, five changes will trump all others:

- **Developers need to embrace cloud computing as an emerging platform.** In particular, developers need to focus on public clouds like that from Amazon Web Services and salesforce.com right away. Why? Because public clouds will let you:
 - Speed delivery of custom applications.
 - Scale up and down.
 - Access clouds from three entry points — IaaS, PaaS, SaaS.
 - Cloud computing is well suited to big data and many Web apps.

Moreover, according to the report, access to clouds like Microsoft Azure, included with the Visual Studio 2010 license, are available now and don't cost anything but time to get up to speed with.

- **Even big shops need to think — and act — like little shops.** Enterprise-level application development groups need to operate like startups. Successful startups:
 - Have limited resources, focus on what is most important, and have little room for missteps, and every line of code they produce needs to solve a business problem and be linked to revenue.
 - Successful startups don't adopt elaborate processes and expensive platforms. They have just enough business-focused process and technical tools and platforms to get them to a solution as quickly as possible.
 - Respond quickly to changing customer requirements and competition, sometimes making big shifts very quickly.
- **Flexibility and cost are more important than platform loyalty.** .NET and Java EE are the top dogs, but Forrester is seeing low-cost alternatives like Adobe Flex, Apache Tomcat, The Dojo Foundation, Drupal, Google Web Toolkit (GWT), Open Source Matters Joomla!, Red Hat JBoss, Ruby on Rails, SpringSource, and the Zend Framework gaining traction. Dev shops need to assess existing projects and determine whether they are overbuying software and hardware in the name of standardization.
- **User experience will be increasingly important.** Users increasingly expect and demand an experience that's valuable, easy to use, aesthetically pleasing, and emotionally satisfying. The user experience has definable business benefits, including:
 - Customer will be willing to purchase. Forrester Research has found that, on average, companies that provide a superior customer experience have 14% more customers who are willing to consider them for another purchase.
 - More customers will resist doing business with competitors. Compared with companies that offer a poor experience, companies that offer the best experience in their industry have 16% fewer customers who are likely to consider doing business with a competitor.
 - More customers will recommend you. Companies with the highest experience scores have 17% more customers who are likely to recommend their products or services compared with their lowest-scoring competitors.

Forrester suggests that developers immediately start making user experience design best practices part of their app development process.

- **Find, coach, and keep the best software developers.** Forrester has found that the top qualities of great developers is their understanding of the business domain and their technical skills. Great talent has both. You find top developers by:
 - Raising your standards beyond just skills. You need application developers who know how to devour business requirements, break down a problem, find creative solutions, and write clean code.
 - Finding talented application developers who may be hiding in plain sight. Review your current application development professionals through the lens of both skills and the other skills they bring to the table.
 - Assessing smarts, not just skills and experience. Smarts can often trump experience and skills.
 - Empowering application development talent in the business.

According to Forrester, each of these five changes will force considerations of broader change, but lead to a more efficient application development organization.



[Return to Table of Contents](#)



By Jonathan Erickson,
Editor In Chief

Programming from "Scratch"

Graphical programming language developed to aid programming "fluency"

When most of us think about computer programming, we think of typing in lines and lines of code with semicolons and complex syntax. This common perception about computer programming helps keep people from getting excited about computer programming, yet many professions need the complex problem solving and analytical skills that programming can build. Changing this perception could help get people, especially young people, interested in computer science.

Mitchel Resnick and his colleagues at the MIT Media Lab (<http://www.media.mit.edu/>), supported by the National Science Foundation (<http://nsf.gov/>), are focused on getting young people excited about computer science and changing this outdated stereotype of the field. Their goal is to encourage young people to use technology as a means to express themselves in creative ways, including through computer programming.

Resnick and his team saw the need for a new computer programming language when they were working with their Computer Clubhouses, community centers that help inner-city youth gain access and experience with new technologies. Students at the clubhouses, ages 10–16, learn software and create their own artwork and multimedia presentations. Many students at these Computer Clubhouses wanted to create animations and games, but existing software programming languages were difficult to learn and not appropriate for the age range of 'the Computer Clubhouses' students.

Resnick and his colleagues saw an opportunity to create a new computer programming language that would be appropriate and fun for children. This new computer programming language would help the students have an enjoyable educational experience in learning math, computation and problem solving skills while helping them create animations and games.'

Scratch is the name of this new computer programming language. The core audience of Scratch is children ages eight to 16, but it has something for everyone.

In Scratch, coding is done with graphical blocks, not with syntax and those semicolons. A student writes code by snapping together blocks, much like LEGO bricks or pieces of a puzzle. Additionally, the blocks are designed to fit only in ways that make syntactic sense. This eliminates the dreaded syntax errors that often frustrate and discourage young computer programmers. To create a program, students drag-and-drop the blocks to create procedures.

For example, a student could code a procedure that would make a figure dance. Scratch is easy for people to get started, but still provides the complex environment that allows people to design more complicated projects. This is the ideal combination for encouraging novices, while still providing a challenging environment for Scratch experts.

When Resnick and his team launched Scratch in 2007, they also launched a Scratch website that allows Scratch programmers to publish their Scratch projects on the web and share it with others. This creates an online community where people share and collaborate on Scratch projects.

The results have been dramatic. Since its launch, there have been almost 800,000 projects uploaded to the Scratch website. Although the number of users and Scratch projects are impressive, the researchers have been most excited about the diversity of projects and the level of sharing and collaboration that exists within the Scratch community.

Dr.Dobb's

EDITOR-IN-CHIEF

Jonathan Erickson

EDITORIAL

MANAGING EDITOR

Deirdre Blake

COPY EDITOR

Amy Stephens

CONTRIBUTING EDITORS

Mike Riley, Herb Sutter

WEBMASTER

Sean Coady

VICE PRESIDENT, GROUP PUBLISHER

Brandon Friesen

VICE PRESIDENT GROUP SALES

Martha Schwartz

AUDIENCE DEVELOPMENT

CIRCULATION DIRECTOR

Karen McAleer

MANAGER

John Slesinski

DR. DOBB'S

600 Harrison Street, 6th Floor, San

Francisco, CA, 94107. 415-947-6000.

www.ddj.com

UBM LLC

Pat Nohilly Senior Vice President,
Strategic Development and Business
Administration

Marie Myers Senior Vice President,
Manufacturing

TechWeb

Tony L. Uphoff Chief Executive Officer

John Dennehy, CFO

David Michael, CIO

John Siefert, Senior Vice President and

Publisher, InformationWeek Business

Technology Network

Bob Evans Senior Vice President and

Content Director, InformationWeek

Global CIO

Joseph Braue Senior Vice President,

Light Reading Communications

Network

Scott Vaughan Vice President,

Marketing Services

John Ecke Vice President, Financial

Technology Network

Beth Rivera Vice President, Human

Resources

Fritz Nelson Executive Producer,

TechWeb TV

techweb™

The social aspect of the Scratch community is an important piece in attracting young people to computer programming. The online community has created expert Scratch programmers whose projects are followed by the rest of the community. Compared to traditional computer programming, Scratch is an easy language to learn and is readily shared with other programmers.

Scratch is being used in the classroom as well. Resnick and his team are creating support materials and working with educators on how best to use Scratch in classroom settings. Additionally, they are disseminating that information online so it can be used as a reference tool for other teachers.

Resnick and his team think it is important that Scratch is a tool to introduce students to computer science concepts, like critical thinking and complex problem solving skills, both in introduction to computer science courses, as well as in non-computer science classrooms.

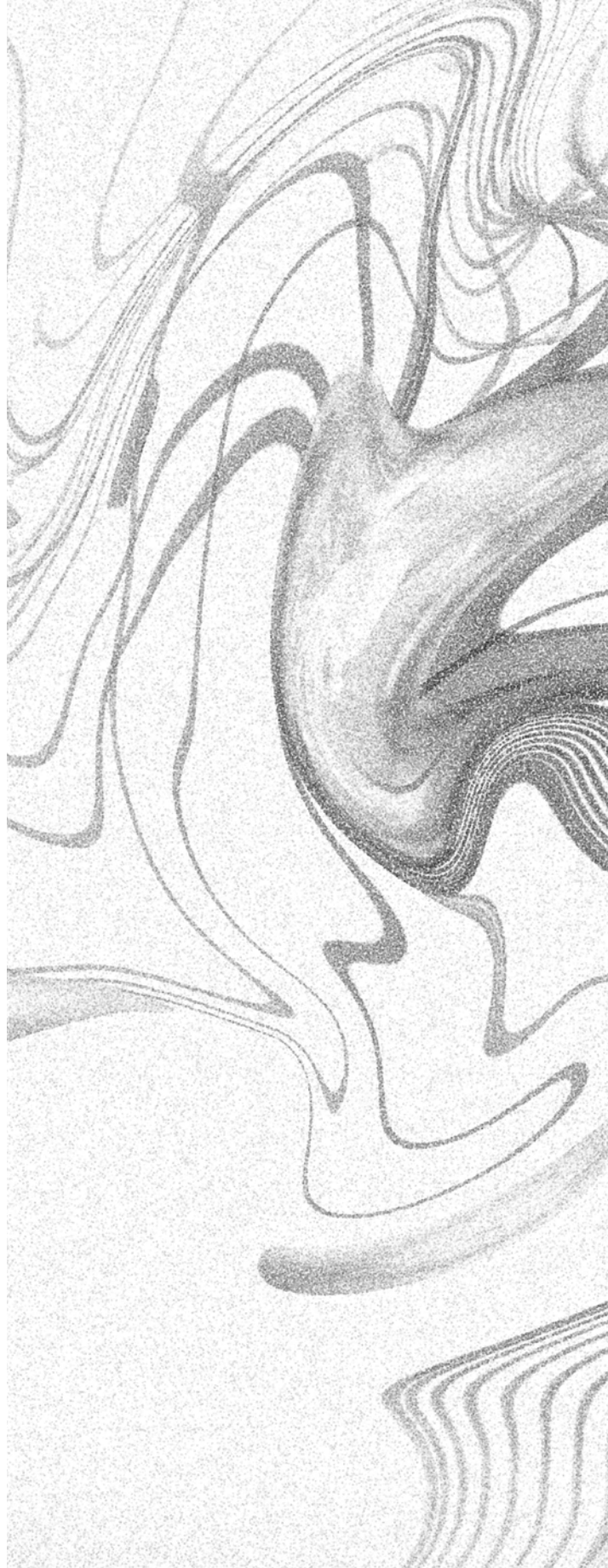
For example, students are using Scratch in English courses as a tool to use in their book reports. Since Scratch is a graphical computer programming language, it has the potential to be used in any kind of classroom. This would expose more students to the skills learned by computer programming and provide additional opportunities for the students to use computer programming to express themselves in creative ways. And, injecting computer programming into non-computer science courses more accurately reflects the present day world where computer-programming skills are needed in a diverse set of professions.

Technology is becoming ever present in our society, but are young people really fluent with technology? This is an important question, as many careers, including careers traditionally outside of technology like the creative arts, need a background in computer science. Although most young people are very good at sending text messages and surfing the Web, that does not mean they are fluent with the technologies.

In "Scratch: Programming for All" (<http://web.media.mit.edu/~mres/papers/Scratch-CACM-final.pdf>), Mitchel Resnick uses a language analogy to question what it means to be digitally fluent. He states that to be fluent in a foreign language, one must be able to explain a complex idea or tell a story, not just be able to say a few phrases in the language. Analogously, he claims that to be digitally fluent, one would need to be able to construct things with the digital tools, not just be able to use them.

The concern is that we have a young generation with access to technologies, but who are not digitally fluent with those technologies. As the 21st Century world becomes more dependent on the skills gained from computer science, such as complex problem solving and analytical thinking, the lack of computer science knowledge may put our young people at a disadvantage. Scratch has the potential to help make more people digitally fluent.

[Return to Table of Contents](#)



9 Ways To Avoid Open Source Pitfalls

Yes, you can avoid problems and still benefit from open source software

by Andrew Till

Microsoft recently rather sheepishly said it would open source its Windows 7 USB/DVD Download Tool, a utility that lets users create bootable copies of Windows 7 on flash drives and DVDs. The company didn't really want to do this, but there wasn't much choice once developers discovered that the software was built using open source code protected by licenses based on General Public License v2, which requires that the source code be made available to anyone who receives the object code. Microsoft isn't alone: The Software Freedom Law Center has identified one new GPL violation like this per day since late last summer, says Bradley Kuhn, the center's technical director.

Neither occurrence should be a surprise to anyone who follows open source software. I've been engaged in a wide range of open source software projects and communities, including the Android OpenMoko, and Maemo Linux-based mobile platforms. Violations are commonplace. However, if you follow the nine tips I present here, you can avoid problems, while still benefiting from what the open source software community has to offer.

1. Ask Why, Rather Than Why Not

It's always important to know why you're pursuing a particular course of action, and using open source software is no exception. Far too often, I hear "Why not?" as the answer when asking CIOs their motivations for engaging with open source communities. Using open source software doesn't just affect a company's R&D organization; it touches functions throughout the business as well, so it's vital to have a clear understanding of why leveraging open source software is the right option. This also helps shape individual actions, provides developers with guidance, and helps the company shape its activities with the open source community.

2. It's Open, Not Free

A common misconception about open source is that it's free. Yes, you can gain access to millions of lines of code at no cost, but that doesn't imply a finished product. Common development tasks

such as integration, regression testing, and the development of feature enhancements still need to be performed. And given that everyone — including your competitors — has access to the same code, your challenge is to build a high-value differentiated product with open source code alone. While leveraging open source lets you redistribute where you spend your development dollars, it typically doesn't reduce that spending. However, it should let you invest more in the innovation and differentiation that you build on top of the open source code, and lower spending on core plumbing and building-block software.

3. A Different Development Model

Leveraging open source software isn't just about finding free code on the Internet and integrating it into your project. It has implications for your overall development model. Instead of traditional waterfall development models widely in use at large companies, open source projects often use Agile or Scrum iterative development methodologies where cross-functional teams collaborate to come up with requirements and solutions.

4. Define A Contribution Strategy

It's important to define a community engagement strategy and contribution policy that says if, when, and how you plan to give something back so you're not viewed as a "free rider." Simply taking and never giving is an open source no-no. Consider whether you'll contribute complete components or bug fixes. Determine whether you'll contribute during development, at service launch, or after launch. You also need to decide if you will provide maintenance and integration support for your contribution — especially if you're seeking to have your code integrated within future baseline releases. Failure to define a contribution strategy and follow through on it can result in significant costs. For instance, if you're building a Linux-based platform and developing additional components, say a software stack that you know will become standard, then you run the risk that a competitor may contribute if you don't. This can result in core baseline software containing components that are less functional or core

APIs that are no longer fully compatible with the components your team has developed. You may then need to make modifications or reintegrate onto each new baseline release, stealing valuable development hours from creating the next great thing for your company.

5. Select Your License With Care

Understanding the licensing models and their implications for your development work can be a challenge. It's worth investing resources to understand which licenses your company should use and which aren't appropriate for your development objectives.

Typically, when coming to open source for the first time, the focus is on GPL or "copyleft" licenses that often require derivative works be licensed under the same terms and conditions, and that source code be made freely available along with a number of other key requirements (private use excepted). However, not all open source licenses are GPL — two other main categories are License Contracts and Permissive License Notices. These licenses are less restrictive and tend to be easier for commercial product developers to use.

Android, the Linux-based mobile device platform developed by the Open Handset Alliance, uses an Apache 2 license that lets companies use the source code in devices and make modifications without the need to contribute or open source the code, provided they include the license text. It's important to fully understand which licenses can be used in combination, as not all licenses can coexist within a common software block.

6. Don't Contribute Bad Code

It's just as important to test and debug code contributions before releasing them to the community as it is to do that before releasing software internally. Contribute buggy code and you'll quickly be seen as a poor community member and your code may not be integrated into future baseline releases. Buggy code can make it more difficult to attract the right talent to your business, especially if you're hiring from within the open source community. If you're developing internal products for your company, it can also affect the confidence the rest of the company has in the development group and if you're developing commercially, it can affect your overall brand. That said, releasing alpha and beta code is acceptable, provided you make it clear that that's what it is and document known issues. Keep it clean, protect your reputation, and you'll become a respected member of the community.

7. Pick The Right Business Model

A common open source debate is how to make money using open source software in a way that's acceptable to the community. A wide range of valid business models can happily coexist using open source software or even providing support services to those already engaged with the community. A number of companies build full software distributions, usually based on Linux, that include a range of open source software but fully integrate it into a usable platform. They provide consulting, support, maintenance, and adaptation services around these distributions. Other compa-

Open Source Defined

For software to be classified as open source, the Open Source Initiative requires that the distribution terms:

- **Not restrict anyone from selling or giving it away as part of an aggregate software distribution**
- **Include source code and allow distribution in source code as well as compiled form**
- **Allow modifications and derived works**
- **Permit distribution of software built from modified source code**
- **Not discriminate against any person or group**
- **Not restrict anyone from making use of it in a specific field**
- **Apply to everyone it's redistributed to without the need for additional licenses**
- **Not tie the program to a particular software distribution**
- **Not restrict other software that's distributed with the licensed software**
- **Be technology neutral.**

nies provide integration services that take open source components and integrate them in product baselines. Relying on the base open source components to build products with significant additional functionality often isn't realistic, so offering services designed to provide customization and extensions to these components is a perfectly valid business model.

8. Don't Require A Ransom

While there are many valid business models, there are also many that can alienate the open source community. One such model is referred to as "asking people to pay ransoms" — that is, contributing core engines or components to the open source community with the intent of making users dependent upon you for overly expensive customization and extension services. The community will quickly reject this model, and your components will likely be removed from the baseline in favor of alternative contributions. It can be surprising how fast the community can move when faced with being held for ransom.

9. Maintenance Isn't Free

While there are justifications for making open source contributions, expecting free maintenance isn't one of them. Most open source projects and communities rely on the good will of the members, be it their personal time or employer sponsorship. However, you can't contract with these sorts of communities to provide service-level agreements for maintenance support, achieve commitments for defect analysis, fix bugs, or set bug-fix priorities.

Get To Work

Putting tips such as these into practice is the first step in developing an effective and rewarding model for leveraging open source software. The core theme is simple: Plan in advance and consider your overall engagement model with the wider community, as opposed to just determining the software components you want to use. Leveraging open source has ramifications that ultimately touch many, if not all, areas of your business.

— Andrew Till is VP of solutions management at Teleca, a mobile technology software provider.

[Return to Table of Contents](#)

Proactive Management Required

Open source can be a powerful tool for cutting costs, improving time to market, reducing risk, and increasing innovation

by Ralph Priester

Every business needs to focus on cutting costs, improving time to market, reducing risk, and increasing innovation. Open source software is a powerful tool to achieve these goals — as long as you manage its use. Unfortunately, many development groups don't know what code sources they have in their codebases, which opens the door to compliance and security issues.

With these risks in mind, Halliburton Landmark instituted manual processes to track code sources and identify open source within DecisionSpace, our core product for the oil and gas industry. However, the time and staffing resources needed to do this significantly hindered our development lifecycle. Manual processes are also prone to human error. We needed an automated way to track baseline code sources that would let us more efficiently manage compliance, minimize error, and proactively use open source in development. Luckily, there are a number of vendors that provide this kind of support, including Black Duck, OpenLogic, Palamida, and Protecode.

We opted for Black Duck Suite, with its database that automatically scans code to check licenses. In addition to heading off potential IP and license issues, Black Duck lets us identify third-party encryption algorithms that require a filing with the Department of Commerce if the code is exported from the United States.

Black Duck has automated the process of tracking and scanning our code base, analyzing approximately 325 million lines of code in 12 months — a process that would have taken more than five years if done manually.

This approach is crucial for the Agile developing we do at Landmark. Automated scanning lets Agile teams identify licensing and compliance issues up front and correct them before they become embedded. Nothing slows the Agile process more than having to backtrack to solve a licensing problem. Programming teams at Landmark also use automated scanning to scan existing open source code for compliance and reuse.

— Ralph Priester is director of intellectual capital, configuration management, and third-party compliance at Halliburton's Landmark Graphics.

[Return to Table of Contents](#)

Selenium: Cross-Browser Website Testing

Automated testing can make a web developer's life a lot easier

by Alexander Sirotkin

A web developer's life has never been easy. Browser market fragmentation, poor standards compliance, and the sheer number of web programming languages and tools all add up to challenges when trying to satisfy user demands and keeping up with ever changing web technologies and trends. And with what appears to be a second browser war underway, the situation is probably not going to get better any time soon. As if this isn't, the advent of cellular broadband access brought to the collection of browsers that you have to test against a handful of new ones, sometimes with limited functionality and non-standard screen resolution. In short, these days you can no longer afford to test your site with just your favorite browser.

Enter Selenium Remote Control (RC), a tool that lets you programmatically simulate user behavior, launch a browser, open a URL, type some text, click on a button, wait for the website response and check the browser state (see <http://release.seleniumhq.org/selenium-remote-control/1.0-beta-2/doc/java/>). Although implemented in Java and JavaScript, Selenium RC supports a variety of languages for test programming, including Java, C#, Perl, Python, and PHP. And last but not least, Selenium is a free open source project with a large and active community of developers and website testers.

Figure 1 shows a typical Selenium test system in which the Selenium client library communicates with the Selenium server via TCP/IP.

A single client can drive a number of browsers via one or more Selenium servers. The number of servers a single client can work with is limited by HW resources and Selenium scalability issues, the rule of thumb is up to 5. Since the Selenium client and the server communicate via TCP/IP they can run on different machines with potentially different operating systems — Selenium runs on Windows, Linux, Mac OS and probably pretty much every OS that has Java support. The most important part of Selenium is the Selenium

core which is implemented in JavaScript and runs inside the browser.

Now that you have some understanding of what Selenium is and how it works, it is time to write some code. The following Selenium RC example makes a simple Internet search with Google. It is implemented in Java (as all the rest of the examples in this article):

```
Selenium selenium = new DefaultSelenium(
    "localhost", 4444, "*iexplore",
    "http://www.google.com");
selenium.start();
selenium.open("/");
selenium.type("g", "42");
selenium.click("btnG");
// Do some stuff
selenium.stop();
```

The code is rather self-explanatory, so I will only describe the important arguments of each command. This example assumes that both the client and the server (Selenium server, not the web server) run on the same machine, i.e. *localhost* and the server is configured to listen on the port number *4444*. The **iexplore* parameter tells Selenium to use a certain Internet Explorer profile (more on this later). Commands *type* and *click* accept HTML ele-

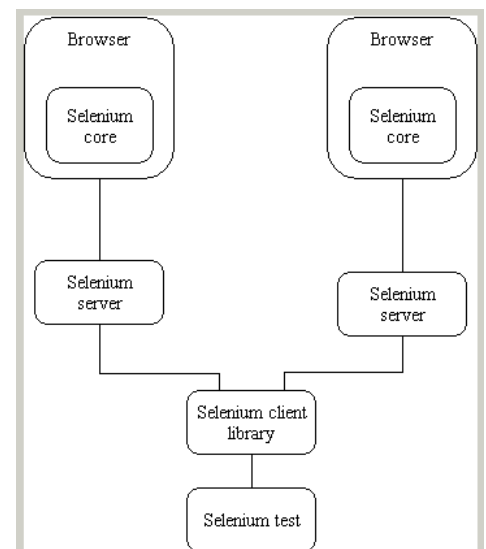


Figure 1: Selenium Remote Control Architecture

ment name or ID as their first argument, which you, as a website developer of your own site under test, should know.

The “BrowserString” parameter (**iexplore*) is very important and unfortunately poorly documented. If you only have to test with Firefox and Internet Explorer you can simply use **firefox* and **iexplore* respectively and forget about this. However, if you need to test with additional browsers, read on.

This parameter not only specifies the browser Selenium will work with, but also the method Selenium uses to control the browser and the mode of communication between the Selenium server and the Selenium core running inside the browser. And yes, there are multiple modes for some browsers, not all modes are implemented for every browser and some Selenium commands will not work with certain modes. To add to the confusion, the default modes sometimes change between different Selenium versions; currently (Selenium 1.0) **firefox* is an alias to **chrome* and **iexplore* to **iehta*. Both **chrome* and **iehta* browser profiles implement a “native” approach in which Selenium uses whatever method is best for each browser to “inject” the Selenium Core JavaScript code and to control the browser, as opposed to a “proxy injection mode” that is generic and, at least in theory, should work with all browsers.

When Proxy Injection (PI) mode is enabled, the Selenium RC which has a built-in HTTP proxy server, configures the custom profile of the browser under test to work with this local proxy. To every http request returned by this proxy server it “injects” the Selenium Core JavaScript code into the <head> HTML element. If you take a look at your website content (using the “view page source” option of your browser) you will see, in addition to your

Same Origin Policy

Same origin policy is the security concept of most modern browsers which defines a clear separation of scripts and content originating from different sites. In a nutshell, it prevents Javascript code downloaded from one domain to interact with the script from a different domain. In particular, it prevents Javascript from navigating your browser to a different domain.

code, the following code along with other scripts, which comprise the Selenium Core.

```
<script type="text/javascript" src="/
selenium-server/core/scripts/selenium-browserbot.js"></script>
```

By design the Proxy Injection mode works with all browsers, even these that are were not tested with Selenium, as long as the browser supports JavaScript and HTTP proxy. Not only that, but it also allows to circumvent the “same origin policy”, i.e. test different sites on different domains during the same Selenium session for browsers for which the “native” mode is not complete; Opera, for instance.

At this point you may think that this the mode you should use for you tests, but unfortunately during the Selenium development people discovered that some important functionality is very hard to implement in this mode and gradually most developers switch to the so called “native” mode, even though it requires a separate implementation for every browser. As a result, the PI mode is poorly maintained and quite buggy. For instance, during my experiments I was surprised to discover that the PI mode does not handle correctly encoding other than Unicode!

If you decide to try the PI mode anyway you may think that the way to enable it would be to use the Proxy Injection mode profiles **pifirefox* and for Firefox and IE, respectively. In practice it should be enabled in the server and the profile argument actually has no effect — Selenium would switch from **firefox* to **pifirefox* and vice versa, depending on the server configuration. Did I already say that Selenium mode configuration is confusing? The fact that PI mode switch is a server runtime argument introduces another significant limitation — the Selenium server working in the PI mode can only handle one concurrent session.

To summarize: As a rule of thumb, remember that you should try to avoid using the Proxy Injection mode and use **firefox* and **iexplore* modes instead. However, if you have to test with a less popular browser for which there is no good native mode implementation and the same origin policy is an issue, be prepared that some of Selenium functions will simply not work.

There appears to be a consensus in the Selenium community that “native” modes are the preferred way to “drive” browsers under test. Selenium 2.0, which is currently being actively developed, will merge with the Google WebDriver project (<http://code.google.com/p/webdriver/>), which provides another indication of the Selenium community to go “native”.

Now that you have a basic understanding of what Selenium is and how it works, let's take a brief overview of the Selenium API, then continue to a more complex Selenium programming example.

Table 1 is a short and definitely not comprehensive list of essential Selenium API calls. I intentionally omitted the less fre-

start	Launches the browser with a new Selenium session
stop	Kills the browser
type	Sets the value of an input field, as though you typed it in
typeKeys	Simulates keystroke events on the specified element
click	Clicks on a link, button, checkbox or radio button
waitForPageToLoad	Waits for a new page to load, usually used after <i>click()</i>
waitForCondition	Runs the specified JavaScript snippet repeatedly until it evaluates to “true”
getEval	Gets the result of evaluating the specified JavaScript snippet
select	Select an option from a drop-down using an option locator
open	Opens an URL in the test frame
openWindow	Opens a popup window
selectWindow	Selects a popup window using a window locator; once a popup window has been selected, all commands go to that window
selectFrame	Selects a frame with in the current window
isTextPresent	Verifies that the specified text pattern appears
isElementPresent	Verifies that the specified element is somewhere on the page
dragAndDrop	Drags an element a certain distance and then drops it

Table 1: Abbreviated list of Selenium API calls.

quently used commands as well as command arguments to make it more readable:

Table 1 demonstrates the richness of the Selenium API, but if you need something extra, take a look at the `getEval()` method that effectively allows you to extend the Selenium functionality by running any JavaScript code on the test page. If you find yourself using this quite a lot, you may find the `-userJsInjection` Selenium server option handy (it allows you to specify a JavaScript file that will be injected into all pages).

Now let's take a look at a bit more complex web test that requires actual programming logic. This is where Selenium really shines. The following sample code tests for a common and frustrating error that I'm sure you've seen at least once — when you fill a long form online and press “Submit” only to discover that you made a mistake in CAPTCHA or password verification and have to fill the same form again. The code below verifies that the Yahoo e-mail registration page preserves all entered fields if the CAPTCHA check fails:

```
String url = "https://na.edit.yahoo.com/registration";
Selenium selenium = new DefaultSelenium("localhost",
    4444, "*firefox", url);

selenium.start();

selenium.open(url);
selenium.windowMaximize();
selenium.type("firstname", "Vasya");
selenium.type("secondname", "Pupkin");
selenium.select("gender", "value=m");
selenium.select("mm", "index=6");
selenium.type("dd", "10");
selenium.type("yyyy", "1917");
selenium.select("country", "value=il");
while (true) {
    try {
        Thread.sleep(200);
    } catch (InterruptedException e1) {
    }
    if (selenium.isElementPresent("cimg"))
        break;
}
selenium.click("IAgreeBtn");
while (true) {
    try {
        Thread.sleep(200);
    } catch (InterruptedException e1) {
    }
    if (selenium.isElementPresent("cimg"))
        break;
}
String firstname = selenium.getValue("firstname");
String secondname = selenium.getValue("secondname");
String gender = selenium.getValue("gender");
String month = selenium.getValue("mm");
String day = selenium.getValue("dd");
String year = selenium.getValue("yyyy");
String country = selenium.getSelectedLabel("country");

System.out.println(firstname + " " + secondname + " " +
    gender + " " + month + " " + day + " " + year +
    " " + country);
selenium.stop();
```

If you run the above example, you will see that the Yahoo registration page works as expected. As with the first example, this code does not need an extensive explanation, but some Selenium tricks may need clarification.

After filling all the fields (I omitted some of them to make the code more readable), the test application waits for the element `cimg` to appear. This is the CAPTCHA image, and if “the user” clicks before it is loaded, Yahoo assumes that something's wrong with your browser JavaScript. After pressing the “Create my

Selenium Projects

There is more to Selenium (<http://seleniumhq.org/>) than just the RC. Selenium Core is the Javascript engine used by Selenium RC, but you can also deploy it standalone. It is actually more flexible than the RC in some ways, mostly because it can work with all browsers, but the downside is that you have to install it on the web server. If you find yourself running Selenium tests overnight than you may want to take a look at the Selenium Grid, which would allow you to run many tests in parallel on different machines in an easy, efficient and scalable way. Since the machines do not have to run the same OS you can also use it for cross browser/cross operating system tests. The last and certainly not the list project is Selenium IDE which is a Firefox add-on that records user actions such as clicks and text typing and produces a Selenium test that you can play back. This record can become a basis for your website tests.

account” button, the test waits for the new page to load. You might think that the `waitForPageToLoad()` Selenium call is the right command to use here, and you would be right, at least according to the documentation. In practice, I've found this function to be unreliable and it is usually safer to test for the appearance of a certain element on the page in all but very simple cases. This is the reason there is a second piece of code that waits for the element `cimg`.

The last piece to the Selenium puzzle is a straightforward instruction on how to run this test. First, you should add “`import com.thoughtworks.selenium.*;`” at the beginning of your code, put “`selenium-java-client-*.jar`” somewhere in the path, and don't forget to run the Selenium server by issuing the “`java -jar selenium-server-*.standalone.jar`” command.

Selenium is a powerful tool for writing website tests. It is under active development, and I expect the Selenium community to add more exciting features in the near future — and hope that they will also find the time to fix some of its bugs. As it often happens with open source projects, implementation of new functionality gets more attention than boring tests and bug fixes.

However, Selenium is probably not the only automatic website testing tool you will ever need. Even though it is often praised for its multiple browser-testing capability; in reality, browsers other than Firefox and Internet Explorer are poorly supported. At the end of the day, if you have to ensure that your website works correctly under all important browsers, your best bet is still a simple browser screenshot tool such as BrowserShots (<http://browsershots.org/>) or BrowserSeal (<http://www.browserseal.com/>). The former is a free web-based screenshot service supporting a wide range of browsers, that unfortunately suffers from long response time; while the latter is a screenshot application that has the advantage of speed and potentially can be integrated into automatic test scripts.

— Alexander (Sasha) Sirotkin works on the LTE (Long Term Evolution) Project at Comsys Mobile. Alexander can be reached via e-mail at sasha.sirotkin@gmail.com.

Return to Table of Contents



Click here to register for Dr. Dobb's M-Dev, a weekly e-newsletter focusing exclusively on content for Microsoft Windows developers.

SQL Server Solutions in the Cloud

There are two types of SQL Server solutions in the cloud right now, and they're as different as espresso and instant coffee

by Brent Ozar

If you need all the power and features of Microsoft SQL Server, you can get the real thing in the cloud. Providers like Amazon EC2 (<http://aws.amazon.com/ec2/>) and Terremark (<http://vcloudexpress.terremark.com/>) offer Microsoft SQL Server (<http://www.microsoft.com/sqlserver/2008/en/us/default.aspx>), including 2008, in virtual machines for rent by the hour. The biggest advantage of these full-strength cloud databases is that they're 100-percent compatible with SQL Server — you can take existing applications and simply deploy them to the cloud without changes. Any application that connects to SQL Server can work with a database stored on an Amazon EC2 or Terremark server. Just as developers can reuse their skills as-is, DBAs can administer their cloud-based SQL Servers without a steep learning curve.

Database administration is slightly different in the cloud, however, because the DBA has to account for some new dangers. Amazon's virtual machines can drop offline at any time, and the DBA needs to plan for that by using a combination of high availability and disaster recovery. One method is to use conventional log shipping, but, instead of log shipping within the same datacenter, use an Internet-based file storage service to another provider. Even though Amazon offers its own file storage services (Amazon S3; see <http://aws.amazon.com/s3/>), and network traffic is free between EC2 and S3, it represents a single point of failure. If Amazon S3 is unavailable, then both the EC2 virtual machines and the backups will be unavailable. Instead, consider backing up to a second provider.

There is some debate as to whether this type of solution really qualifies for the term “cloud,” or whether it's more akin to traditional virtual server hosting. To add more CPU power or memory to a server, the server must be reprovisioned from scratch. The user must create a brand-new SQL Server and move the data from the old server to the newer, faster server. All OS and SQL Server configuration changes are lost. With careful scripting and planning, DBAs can minimize the work involved, but this work still doesn't line up with the public's perception of cloud scalability.

If you don't need the fully-cafeinated strength of SQL Server, though, you can compromise on power to gain scalability with the other cloud-based database option.

SQL Azure: Instant Coffee

Just as instant coffee sacrifices flavor for portability, Microsoft's own cloud-based database solution, SQL Azure, sacrifices features in exchange for easier deployments. SQL Azure isn't completely compatible with SQL Server, and most applications that were already built to store their data in SQL Server probably won't work as-is in Azure. SQL Azure offers fewer datatypes, stored procedure options, and cross-database querying, and less of what SQL Server has built up over the years.

Unfortunately, one of the things it sacrifices is the ability to handle large databases. Azure's database size is capped at 10GB per database. When the developers need to store more than that, they have to buy multiple databases and figure out how to spread data between each of them. This presents challenges for typical SQL Server developers who are used to simply pouring all their data into a single

database. Sharding, the technique of splitting data across databases, isn't an easy skill to master and presents plenty of design challenges.

The production DBA will have challenges as well, since Azure doesn't support the BACKUP command. It's up to the DBA to figure out how to get the data out of Azure consistently and reliably. If Azure becomes unavailable — as it has for short times in the past — either the application will be completely unavailable, or the DBA will have to create a failover option using SQL Server.

The two options both have their own pros and cons, and the best way to illustrate that is to show a use case for each technology.

Cloud Use Case 1: Bursts of Analytics

StackOverflow.com is a question and answer site for programmers with several interesting twists. Questions, answers, and comments can all be voted up or down, and user reputations rise and sink with the quality of their work. Since questions can be tagged with technology names like `sql-server` or `c#`, readers can

quickly find users who are highly rated for a particular technology. Number freaks will love the next twist: the data is licensed under Creative Commons. Every month, the StackOverflow team exports the database in XML format and makes it publicly available via BitTorrent. Anyone can download the data, import it into their database of choice, and slice and dice the numbers to look for trends.

This scenario lends itself to cloud-based databases for a few reasons:

- High power needed in short bursts. Data processing is only done once per month when the export is made available, and the rest of the month the server sits idle.
- No backups required. There is no original data involved, only imported data from other sources.
- No privacy concerns. The data is already public.

However, the scenario didn't work well with the SQL Azure model because:



[Click screen to link to video](#)

.NET Sensor and Location Library for Windows 7

Yochay Kiraity has been working with the team to create Managed Code APIs making it even easier for .NET developers to take advantage of these features. Watch the video on Dr. Dobb's Microsoft Resource Center.

- The data surpassed Azure's 10GB limit. I would need to shard the data across multiple databases in order to do my analysis.
- Loads would have been slow. To load the entire export every month, I'd need to get the data from BitTorrent, then export records into Azure. Amazon's virtual machines made this much faster because I could have them pull the data from BitTorrent, then insert the data locally rather than over the Internet.
- Azure's costs would be prohibitive. I didn't need to persist the data all month, but, instead, do my analysis once and export it. Getting two 10GB Azure databases would cost \$200 per month, which equates to a lot of EC2 processing time. EC2 would process it faster, and I could then shut the machines down.

In this scenario, I chose to use the cloud, just not Azure. For less than the cost of a pizza, I was able to run a very powerful SQL Server in short bursts — long enough to find the answers I wanted, but short enough to avoid commitment.

Cloud Use Case 2: Shared Data for Queries

Over the past several months, I've been importing Twitter data into a SQL Server database. I wanted to see what the public could do with it, so I made a backup of my SQL Server database and put it on my web site for anyone to download. The same day, SSIS guru Jamie Thomson took that data and put it into SQL Azure (http://sqlblog.com/blogs/jamie_thomson/archive/2009/12/03/twittercache-now-hosted-on-sql-azure.aspx). This let anyone with SQL Server Management Studio connect to the database and do read-only queries.

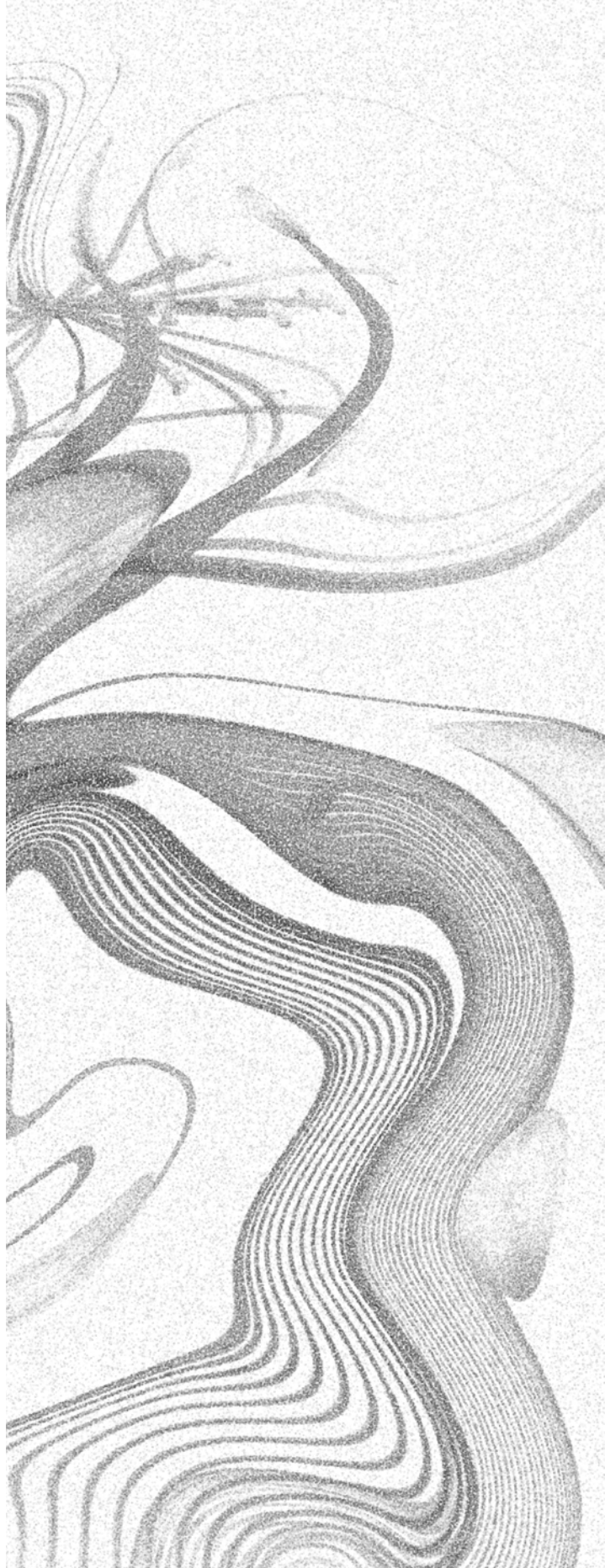
This scenario played to SQL Azure's strengths because:

- It needed public access. Running a publicly accessible SQL Server can be expensive and dangerous, but Azure takes away most of the cost and a lot of the risk.
- It fit within Azure's 10GB limits. And the dataset wouldn't grow past that limit in the near future.
- Anyone could use SSMS to query it. Database developers and administrators could leverage the tools they already owned to access the cloud.
- It could be populated with SSIS. Jamie didn't need the full-blown functionality of SQL Server, just the ability to insert data via an ETL tool and then query it. Azure's subset of features didn't impair him at all.

Espresso isn't for everyone, nor is instant coffee, but they both have their fans. The two cloud-based SQL Server options out there today won't meet everyone's needs either, but they're already being deployed in niche markets. As their feature sets grow and their drawbacks decrease, they'll see more widespread adoption.

— Brent Ozar is a SQL Server Expert with Quest Software, and a Microsoft SQL Server MVP. He blogs at www.brentozar.com.

Return to Table of Contents



Introduction to Functional Programming

Functional programming gives us new ways of developing scalable systems

by John Puopolo

There are many ways to examine problems and model their solutions. In software, we tend to look at problems through the lens of our experiences and the tools that we have at our disposal. In other words, our experiences and the tools we know influence how we approach solving problems. For example, in the days of structured programming, we solved many problems using procedural decomposition and subroutines, focusing on internal cohesion and loose coupling. Building on these successful techniques, we transitioned to modeling solutions using objects and object-oriented techniques, further encapsulating imperative code and its data.

While object-oriented design and object-oriented programming will continue to play a significant role in years to come, I believe we are facing challenges that require new ways of thinking and new ways of implementing solutions. For example, massive computing power via multicore architectures and cloud computing will make developing concurrent systems and parallel algorithms a virtual necessity. Functional programming provides us with many of the tools and technique we need to make this a reality. Additionally, functional programming gives us new ways of developing scalable systems that are easy to debug and test.

What is Functional Programming?

Functional programming is a specific way to look at problems and model their solutions. Pragmatically, functional programming is a coding style that exhibits the following characteristics:

- Power and flexibility. We can solve many general real-world problems using functional constructs

- Simplicity. Most functional programs exhibit a small set of keywords and concise syntax for expressing concepts
- Suitable for parallel processing. Via immutable values and operators, functional programs lend themselves to asynchronous and parallel processing

Functional programming treats computation, e.g., running a program or solving a numeric calculation, as the evaluation of functions. Not surprisingly, in dealing with functional programming, we will use the term function quite often and in many contexts. So, before we continue, let's make sure we understand what a function is:

A function is fundamentally a transformation. It transforms one or more inputs into exactly one output.

An important property of pure functions is that they yield no side effects. This means that the same inputs will always yield the same outputs, and that the inputs are not changed as a result of the function. This idea of “no side effects” is one (of several) that makes functional programming particularly attractive when writing multi-process and multithreaded systems. The key concepts that make functional programming attractive for solving certain types of problems are described in *F# Survival Guide* (www.ctocorner.com/fsharp/book/default.aspx).

From the abstract mathematical theory of functions comes a variety of interesting concepts that have broad application and use in functional computing. For example, functions can take other functions as arguments, and functions can return other functions as results. This ability results in flexible and powerful mechanisms that enable us to implement ideas simply and elegantly — ideas that are difficult to express in non-functional languages.

Functional vs. Imperative

We can think of imperative programming as writing code that describes in exacting detail the steps the software must take to execute a given computation. These steps are generally expressed as a combination of statement executions, state changes, loops, and conditional branches.

In contrast, functional programming looks at a program as a set of nested functions — one “outer” function (the program) calling one or more “inner” functions to calculate a value to be returned as the result of the program. Functions take arguments and calculate return values, but do not maintain a “state of the program” per se.

Most modern programming languages support sufficient logic and data storage as to make them Turing complete (http://en.wikipedia.org/wiki/Turing_completeness). This means that in principle they can be used to write programs to solve any “computable” problem. F# (<http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/default.aspx>) is also a Turing complete language. The crux of choosing a language is not then one of capability, but of inherent applicability.

F# offers some advantages over other languages when working on certain classes of problems. Some general strengths of F# are as follows:

- The signal-to-noise ratio is high. F# is economically expressed, e.g., favoring whitespace over other grouping symbols. As a result, your code ends up being quite succinct.
- Powerful matching logic based not only on Boolean values but the “shape” and type of the data.
- Built-in support for “lazy evaluation” of expressions.
- Built-in support for writing thread-safe and asynchronous applications very simply via asynchronous workflows.

To help make things a little more concrete, and whet your appetite for the material to come, let's consider a few examples that help contrast how we'd look at and solve the same problem from the imperative vs. the functional perspective.

Example 1: Filtering a List

Let's suppose we have a list of 10,000 employee records in a list, and we want to find all the employees with salaries that equal or exceed 100,000 dollars. Taking the imperative approach, we would most likely write a *for*- or *while*-style loop, examining each employee's salary to see if it met our criteria. For each employee meeting the criteria, we would append the record to a growing list of matches. As you can see, we delineate each and every step in detail. In contrast, from a functional perspective, we would describe this solution as applying a filter (function) to a list with the goal of producing a new resulting list.

Purely Functional?

Some programming languages are “purely functional languages.” This means that they follow strict functional rules, e.g., no imperative looping constructs and no implicit way to introduce state changes. For example, Haskell, a pure functional language, disallows state changes and side effects altogether. Programs written in pure functional languages are almost solely made up of functions that accept arguments and return values. Unlike imperative and object-oriented languages, they allow no side effects and use recursion instead of loops for iteration. Languages like Haskell are often criticized as being too extreme in not allowing side effects, giving them limited penetration in mainstream development environments. Operations such as reading a file become awkward to code in a language where no side effects are allowed. In contrast, impure, multi-paradigm languages such as F#, a strongly typed, first-class .NET programming language designed by Don Syme and others at Microsoft Research (<http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/default.aspx>), discourage side effects and imperative constructs, but make them available alongside their functional brethren.

Example 2: Downloading Files Asynchronously

Asynchronous and parallel programming are difficult to do and even more difficult to get right. For example, to asynchronously download HTML pages from the Web via imperative languages requires us to create HTTP channels, issue asynchronous calls, write completion routines, and manage failures. Depending on the implementation, we may need to also manage our own threads. This results in a lot of complex, error-prone code. While we can't escape having to perform these tasks in some way, we can manage them much more succinctly and rigorously in functional languages by structuring and sequencing the calls using a concept called “workflows.”

Example 3: Customizing Algorithms

A classic problem that functional languages address elegantly is that of arbitrary value comparison. When your program needs to sort or compare values, it needs a mechanism to determine whether two values are “equal” or if one value is “less than” the other. When using naturally ordered elements, such as numbers or letters, the natural ordering applies. In many business-style programs, however, we need to compare elements that have no natural ordering, e.g., Employees, Automobiles, Tickets, Movies, Concerts, etc. In these types of applications, it's extremely convenient to supply a comparison or sorting function that can be subsequently executed against two elements. This is one of the most powerful capabilities of functional programming — the ability to not only pass and return simple values, but the ability to pass and return functions.

To solve this problem in a functional language, you merely pass the main comparison function a “sort function” that it can use to compare two elements. Depending on the context of the

When faced with a problem, you need to consider what you have in your “toolbox” and decide the best way to solve it

application, or the types being sorted, the program can pass along different “sort functions” without changing the main routine whatsoever. Functional languages tend to exhibit this compositionality in most aspects of their design and, ahem, function.

To address this problem in traditional imperative languages, we'd need to resort to function pointers or similar constructs. Again, this is possible to do, but not as clean or elegant as it can be. We should note here that C# now ships with some functional constructs, including anonymous functions and lambda expressions, which help to elegantly address this type of problem.

Example 4: Creating Infinite Sequences

There are certain data sets that are “naturally infinite”, e.g., the set of all prime numbers, making their complete calculation impossible. Functional languages employ mechanisms such as lazy evaluation and delayed computation to make infinite sets both representable and efficiently computed in on-demand fashion. Again, while you can accomplish the same things in non-functional languages, the imperative implementations tend to be “less natural” and more convoluted.

What Problem Spaces are Naturally Addressed by Functional Programming?

Growing up, I'd occasionally help my dad fix things around the house, paint, etc., and from the many things he taught me, “the right tool for the job” stands out clearly. When faced with a problem, you need to consider what you have in your “toolbox” and decide the best way to solve it. Certain problems can be solved eas-

ily with a hammer, whereas with others you need a saw. You need to apply the same thought process to choosing programming paradigms and the languages that support them.

Functional programming is appropriate for applications that exhibit one or more of the following characteristics: require significant computation (compute-bound) as they can be parallelized easily, are themselves parallel in nature, can benefit from asynchronous calls, need to be provable, or require sophisticated pattern matching. This means that functional programs are not generally applicable to typical line-of-business applications that deal with objects and their state over time; however, there may be portions of those programs, e.g., portfolio basket optimization for fixed-income securities, that would benefit greatly from using functionally-based libraries. We typically see functional programming applied to image processing, machine algebra, lexing and parsing, artificial intelligence, and data mining.

Why Should I Care About Functional Programming?

While you can develop concurrent, scalable and asynchronous software without embracing functional programming, it's simpler, safer, and easier to use the right tool for the job. Functional programming enables you to take advantage of multi-core systems, develop robust concurrent algorithms, parallelize compute-intensive algorithms, and to readily leverage the growing number of cloud computing platforms.

In addition, if you've not already explored non-imperative programming, it's a great way to expand your problem solving skills and your horizons. The new concepts that you'll learn will help you to look at many problems from a different perspective and will help you to become a better and more insightful OO programmer.

A note of caution: I am not here to convince you how wonderful functional programming is, or how it will change your life. It is a tool — nothing more, nothing less — that helps to solve a certain class of problems. In *F# Survival Guide* (www.ctocorner.com/fsharp/book/default.aspx), we explore and explain these types of problems, and point out why functional programming applies. We go on to explore functional concepts in depth in Chapter 8, Functions and Functional Concepts (<http://www.ctocorner.com/fsharp/book/ch8.aspx>). To build up to that, we first examine F#'s syntax, primitive data types, control flow structures, loops, and basic collections (<http://www.ctocorner.com/fsharp/book/ch2.aspx>).

— John Puopolo is, with Sandy Squires, the author of the *F# Survival Guide* (www.ctocorner.com/fsharp/book/default.aspx), on which this article is based. The *F# Survival Guide* covers all of the essential elements of functional programming and the F# language.

[Return to Table of Contents](#)

Bug Opportunity Levels

A systematic approach to writing robust, bug free code

by Peter Eastman

If every piece of code worked perfectly the first time, programming would be an easy job. Sadly, that is not the case. In any large project, you will probably spend more time testing, tracking down bugs, and figuring out how to fix them than you spend writing the code in the first place.

Dealing with bugs is an unavoidable part of writing software, but there are things you can do to make the job easier. At every step of the process, you make decisions that influence the risk of creating bugs later on. Experienced programmers understand this and intentionally write their code in a way that minimizes the opportunities to make mistakes. This is sometimes known as “defensive programming”.

In this article I make this practice a little more systematic by presenting a taxonomy of “bug opportunities” — situations in which it is possible to make a programming mistake. I categorize them into five levels, based on the time and effort needed to identify the mistake. And I give examples of each level, and suggests strategies for turning the more severe levels into less severe ones.

A bug opportunity is not the same as a bug. With enough caution, even a minefield can be successfully navigated. But you will make mistakes from time to time, and the more opportunities you have for mistakes, the more mistakes you will end up making. A key to writing bug-free software is therefore to avoid situations in which you might make a mistake; or when that is not possible, to choose your risks carefully so that bugs will be found and corrected as quickly as possible.

Most of the examples in this article are in Java, and I assume familiarity with the Java language.

None of the principles are Java specific, however, and they apply just as well to almost any other language.

Bug Opportunities

Here are the classes of bug opportunities:

Level 0: There is no opportunity to make a mistake. This is the ideal situation, and the one you should always strive for. It does not simply mean that there is not a mistake. It means there cannot be a mistake, because you never have an opportunity to make a mistake. It is not a “bug opportunity” at all, which is why I call it level 0 instead of 1.

For example, consider the “enhanced *for* loop” introduced in Java 5. Suppose you have two arrays of objects. You need to compare every element of the first array to every element of the second one, and take some action when you find a match. Prior to Java 5, you might have written it like this:

```
for (int i = 0; i < array1.length; i++)
{
    Object obj1 = array1[i];
    for (int j = 0; j < array2.length; j++)
    {
        Object obj2 = array2[j];
        if (obj1 == obj2)
            // We found a match, so do something.
    }
}
```

Using the enhanced *for* loop, it can be written more concisely:

```
for (Object obj1 : array1)
{
    for (Object obj2 : array2)
    {
        if (obj1 == obj2)
            // We found a match, so do something.
    }
}
```

The second version is a bit faster to type. That's a nice bonus, but not really a big deal.

Somewhat more importantly, it is easier to read and understand. That will make your code easier to maintain in the future.

The really big difference, though, is that the second version works, while the first one does not. Did you spot the bug? In the sixth line, I used *i* as the index into *array2* when I should have used *j*. This kind of indexing error is very common. With the enhanced *for* loop, there is no opportunity to use the wrong index because no index variable ever appears in the code. By taking advantage of a language construct, we have eliminated any danger of a major class of error.

Level 1: If there is an error, it will be caught at compile time. This is the next best case. Although it is possible to make a mistake, that mistake is guaranteed to be caught before you ever run the program. If your source code editor provides on-the-fly error checking, you may even be alerted to the mistake within seconds of making it.

Many recent developments in programming languages have been oriented toward allowing more errors to be caught at compile time. A good example is the generics facility added to Java in version 5.

Suppose you are creating an *ArrayList* to hold *Strings*. You do not intend that anything but *Strings* should ever be put into it, and you explicitly cast each element to a *String* as you remove it from the list. Nonetheless, prior to Java 5 there was nothing to prevent you from putting a different kind of object into the list. The result would be a *ClassCastException* when you removed it and tried to cast it to a *String*.

In Java 5, you can declare the list to be an *ArrayList<String>*. This informs the compiler that nothing but *Strings* may ever be put into the list. If you try to put anything else in, your code will fail to compile.

Level 2: If there is a bug, it will result in a runtime error describing the problem. This case is worse than the previous one in a couple of ways.

- First, it takes more steps to discover a bug. The cycle of edit-compile must be lengthened to edit-compile-run-test. This will slow down your debugging quite a bit.
- Second, a bug may be in a piece of code that does not get executed every time the program is run, or it may only occur under special circumstances. A lot of testing could be required before it is discovered, and even then, it is hard to be confident that all bugs have been found.

This is the reason that languages with static typechecking, such as Java, are generally preferred when writing large applications. In a dynamically typed language such as Python or Ruby, you can invoke any method on any object. The compiler does not know what type of object is stored in any variable, and just trusts that it has a method with the specified name that takes the right set of arguments. If this assumption is wrong, the result is a runtime error.

To be fair, this can be a very useful tool in certain cases. It also

allows your source code to be slightly more concise, which is convenient when writing simple scripts. But for large applications, those benefits are far outweighed by the lack of compile-time error checking. This is one of the reasons that so many more applications are written in Java than in Python.

A level 2 bug opportunity is definitely worse than a level 1, but it is still not that bad. If all bugs were of this sort, debugging would be a relatively simple and painless process. Now let's consider a case that really is bad.

Level 3: If there is a bug, the program will work incorrectly. In this case, there is no error message to warn you of the problem. The program simply behaves in some way that is incorrect.

This is worse for a couple of reasons:

- First, the behavior might be incorrect in some subtle, non-obvious way. A large amount of careful testing and analysis might be required to detect the bug. In the worst case, you might not catch it at all, and only learn of its existence when a customer reports it to you.
- Second, even when you learn of the existence of a bug, it may not be obvious what is causing it. This is especially serious when you do not know a reliable way to reproduce it. A long, painful debugging process may be required before you finally track down the cause of the problem and fix it.

Suppose you are writing a method to perform some operation, and realize that in certain cases, it is impossible to do exactly what you intend. For example, suppose it is meant to set the value of some parameter. That value is stored both in memory and on disk, and the method is supposed to update it in both places.

What if an error prevents you from updating the value on disk, perhaps because another thread already has the file open for writing? There are two obvious choices for what to do:

- Update the value only in memory, then return.
- Throw an *IOException* and force whatever code is invoking the method to handle it.

Inexperienced programmers will often choose the first approach. They will claim this is “more robust”, since the method partly succeeds instead of completely failing. They will say they are making the code “fault tolerant”. They will say it produces an easier-to-use API, since programmers invoking the method are not forced to worry about *IOExceptions*. They are, however, completely wrong.

In fact, what they are doing is replacing a level 2 bug opportunity — an exception that clearly tells you what has happened — with a subtle, hard-to-identify source of intermittent bugs. Every method should precisely define what its behavior is. If it cannot do exactly what is expected, it should signal this in a way that alerts either the programmer or the user to the problem.

A level 3 bug opportunity is bad, and you should try hard to avoid them, but there is something even worse:

Level 4: If there is a bug, it will cause an unrelated part of the program to work incorrectly. If you have only programmed in Java and other modern, high-level languages, you may have trouble thinking of a case where this would be true. In fact, errors of this sort are rare in Java, which is a tribute to some of the good decisions made in the design of the language. On the other hand, if you have done much programming in C or C++, you probably know exactly what I mean, and have spent far too much time hunting for bugs of exactly this sort.

Suppose you create an array of length 10, then try to store a value into the 11th element of the array. In Java, this will result in an *ArrayIndexOutOfBoundsException*. This is a level 2 bug opportunity — not a big deal.

In contrast, C and C++ do not perform bounds checking on arrays. It will happily store the value into the memory location that would hold the 11th element of the array, if only you had allocated that much space for the array. Everything will seem to work normally, exactly as if the array really had 11 elements.

The only problem is that you have just overwritten whatever information was in that memory location. Some completely unrelated piece of code may have stored something important there. Thousands of lines further on, perhaps minutes later, that piece of code will look at that memory location and find an incorrect value there. The result could be anything from an immediate crash to a subtly incorrect result for some important calculation. Furthermore, the result could be different every time you run the program, since memory layout is not guaranteed to be the same from one run to the next.

Bugs of this sort are extremely difficult to track down, since there is no obvious connection between the error you observe and the root cause of the error. You can easily spend a week trying to fix a single bug.

You may find it surprising that I chose to categorize bug opportunities solely on the basis of how difficult it is to find the bug, not on how severe the consequences are. After all, some bugs are much worse than others. Having the program crash or corrupt user data is far worse than a minor cosmetic bug, like drawing some text in the wrong font.

The reason is that it doesn't matter how serious a bug is, as long as you find and fix it before the program is ever released to users. In practice, of course, you will have a finite amount of time for bug fixing, and may choose to focus that time on the most serious bugs, intentionally choosing to release with the less serious ones unfixed. But you first need to find the bugs before you can prioritize them. A bug that does not get found is guaranteed not to get fixed. Besides, the less time you are forced to spend finding bugs, the more time you will have left for fixing them.

Now that you know about the different types of bug opportunities, what can you do with them? First and foremost, keep them in mind while programming. You should constantly be asking your-

self, "What kinds of mistakes are possible in this design?" Try to find ways to replace higher level bug opportunities with lower level ones. Above all, watch out for level 3 and 4 bug opportunities, and try to transform them into level 2 or lower ones.

Assertions are a popular technique that can be used for doing this. Suppose you have just written some code which calculates a number. You expect that number to always be positive, and if it ever were not, that would indicate that something was wrong. Then go ahead and check that in the code:

```
assert number > 0;
```

What you are doing is transforming a level 3 bug opportunity (the code works incorrectly) into a level 2 one (an exception is thrown clearly alerting you to the presence of a bug). Similarly, it is always a good idea for methods to validate their arguments to make sure whoever is calling them has passed in sensible values.

Unit tests are another popular tool for avoiding bugs. Each time you write a piece of code, you should also write a test case for it that repeatedly invokes it with different inputs and verifies that it behaves correctly. Using the classification system of this article, we can see why unit tests are so useful. They transform a level 3 bug opportunity (the code produces incorrect results) into something like a level 2 one. (It actually is slightly different: Errors are not discovered when you run the program, but rather when you run the test case. As long as you run all your test cases sufficiently often, this may be just as good.)

Make extensive use of checked exceptions to signal errors as soon as they happen. Don't worry that forcing users of an API to handle exceptions might make that API harder or less convenient to use. The fact is, if a routine can fail, anyone calling that routine should be forced to think about that failure and decide how to handle it. The extra work required to write 9 blocks is tiny compared to the cost of dealing with bugs that occur when a routine silently fails and the invoking code doesn't detect or handle it.

You cannot hope to completely eliminate all bugs from your code, but there is a lot you can do to reduce their number. Every design decision you make affects the risk of bugs later on. By analyzing the options in terms of bug opportunities and picking the design that creates the fewest, lowest level ones, you can make debugging easier and produce better, less buggy code.

[Return to Table of Contents](#)

CPU-Z: Under the Multicore Hood

Version 1.53 delivers more information about multicore CPUs

by Gastón Hillar

If you want to understand modern multicore hardware, it is a good idea to run CPU-Z — the legendary freeware that gathers information on microprocessors, mainboard, memory, and graphics.

What Kind of Questions

While developing software optimized for parallelism and multicore, you need answers to questions such as:

- Does the microprocessor support SSE 4.1?
- Does it support SSE 4.2?
- Does it offer virtualization extensions?
- How many physical and logical cores does it offer?
- Does it offer a Level 3 cache?
- How many Level 2 caches does it offer?
- I need information about the QPI (short for QuickPath Interconnect; see www.ddj.com/architect/217500801) link. Which is the speed offered by the QPI link?

CPU-Z 1.53 (<http://www.cpuid.com/>) provides this information and answers to the aforementioned questions with a simple freeware

application. The new version is compatible with both 64-bits and 32-bits modern Windows versions, including Windows 7 and Windows Server 2008 R2. This new version is also available in Chinese.

You can install CPU-Z or run it without installation. Once you run the application, it takes a few seconds to detect the underlying hardware and it will display a window with many tabs. The first tab, CPU, offers valuable information about the microprocessor(s) found in the computer. For example, Figure 1 shows the information for an Intel Core i7 820QM (mobile Core i7).

As you can see in Figure 1, there is valuable information about the additional instruction set:

Instructions MMX, SSE(1, 2, 3, 3S, 4.1, 4.2), EM64T, VT-x

The aforementioned line means that this microprocessors offers MMX and SSE up to SSE 4.2, EM64T (x86-64) and Intel virtualization extensions (VT-x). In fact, the HTML report that you can save with this utility will show a more complete line:

Instructions sets MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x

As you can see in the same tab, the processor offers 4 physical cores (Cores) and 8 logical cores or hardware threads (Threads). Remember that the number of graphics shown in Windows Task Manager corresponds to the number of hardware threads, as in Figure 2.

If you have more than one microprocessor installed in the computer, you can see the information for the other processors choosing it on the Selection combo box.

The information shown about the clocks correspond to the first core (Core #0). As many new microprocessors, like Core i7, are able to change the multiplier individually, per core, this information doesn't apply to all the cores offered by the Intel Core i7 820QM.

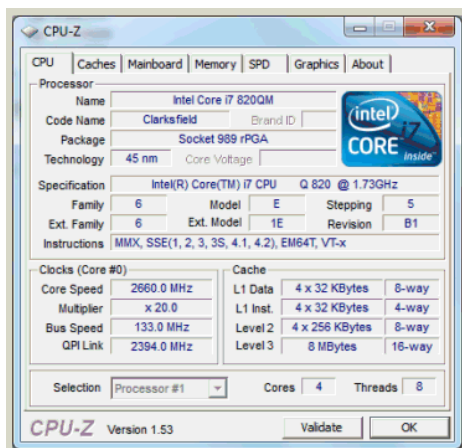


Figure 1: CPU-Z displaying information about a mobile Intel Core i7 with 8 logical cores (hardware threads).

The speed for the QPI link is useful when you're trying to determine potential bottlenecks for multicore designs.

Besides, it shows information about the cache memories included in each microprocessor. However, there is more detailed information in the Caches tab, as in Figure 3:

In this case, as you can see, there is a shared 8 megabytes level 3 cache memory. The four physical cores share this cache. Besides, each core has its own 256 kilobytes (0.25 megabytes) level 2 cache memory. Therefore, CPU-Z adds an x4 to the cache size.

It also offers detailed information about the data and instruction level 1 caches, their number of ways and the line size.

This information is very important to measure the potential multicore power offered by the underlying hardware and the additional instructions to take advantage of parallel execution. Sometimes, a small change in the micro-architecture produces important speedups in certain parallelized algorithms. Therefore, it is very important to take into account the underlying hardware.

CPU-Z also offers detailed information about the mainboard, the memory subsystem, and the graphics card. The information about memory timings is indeed important to understand the problems that you are going to face when the three cache levels fail to deliver your data, because the latencies are still too high, compared to the evolution of the microprocessors' architecture. In fact, this is one of the most important problems that you are surely going to meet whilst trying to optimize your algorithms as much as possible in modern multicore hardware.

The paradigm shift required to translate multicore power into application performance oblige developers to learn the new hardware that runs their software and systems. Tools like CPU-Z are a great alternative to understand the features offered by most modern microprocessors and micro-architectures that power different Windows versions.

— Gastón Hillar is the author of C# 2008 and 2005 Threaded Programming: Beginner's Guide.

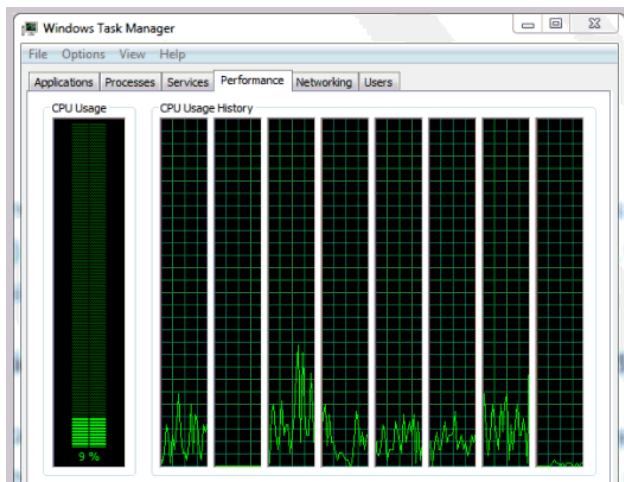


Figure 2: Windows Task Manager displaying one graph per logical core (hardware thread).

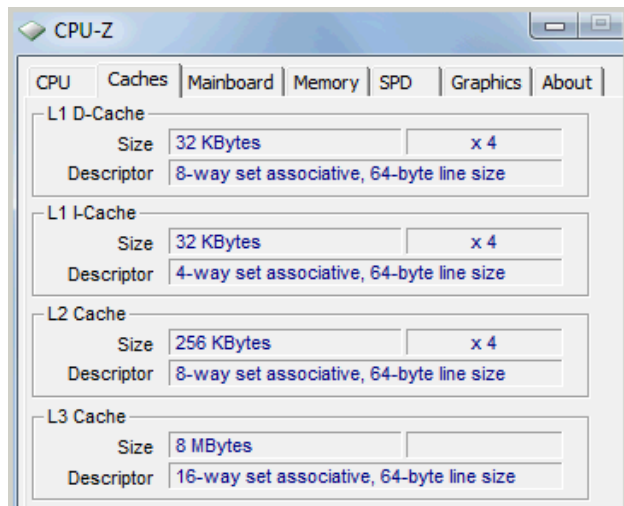


Figure 3: CPU-Z displaying information about the three cache levels for a mobile Core i7.

[Return to Table of Contents](#)

Q&A: It's All About The License

by Jonathan Erickson

As director of intellectual property strategy for the Linux Foundation and an attorney at Choate Hall & Stewart, Karen Copenhaver knows a bit about open source software licensing. She recently spoke with *Dr. Dobb's* editor in chief Jonathan Erickson.



Dr. Dobb's: Are open source and public domain the same thing?

Copenhaver: Not at all. Open source licenses are granted by the copyright holder and the license is an exercise of the copyright. In order to enjoy the benefits of the license, you must comply with its terms. If you don't comply with the license, you're not licensed — and another word for unlicensed use is “infringement.”

Dr. Dobb's: There are usually no signed license agreements with open source software. How, then, can users be subject to licenses?

Copenhaver: Confusion is often caused by the fact that the software is made generally available for download. Users may assume that if something is made available for free, it's thus licensed for use without restriction. That's not the case. A license may be implied where one isn't stated, but an implied license can't contradict an actual stated limited license. The fact that there is no charge for the software is irrelevant.

Dr. Dobb's: What are the main legal issues regarding open source?

Copenhaver: The main issue for businesses is internal control. Businesses have procurement organizations that are responsible for procuring everything the business uses in its operations. Those organizations review agreements and establish terms before they write the check. The internal control is based on the assumption that the business will pay for everything it uses. In order to arrange for payment, you have to go through procurement.

Because open source is made available without charge, this basic internal control fails. The software often comes into the organization without review of the associated obligations or any process to ensure compliance. Companies are in the process of implementing policies and processes that impose basic internal controls over acquisition, use, and compliance of open source software.

Dr. Dobb's: Can you give us a (legal) definition of open source? What exactly is it?

Copenhaver: There is an official definition of open source that can be found at the website of the Open Source Initiative (<http://www.opensource.org/docs/osd>). This definition was developed from the work of Bruce Perens and is used by OSI to determine whether or not a software license can be considered an “open source” license.

But when many people say open source, they are actually thinking about “Free Software” as defined by the Free Software Foundation based on the work of Richard Stallman. For a software program to be Free Software, it must be licensed under terms which assure that all users will have the freedom to run, copy, distribute, study, change and improve the software. Free Software is defined by preservation of these liberties, not by price. As the Free Software Foundation likes to say: “To understand the concept, you should think of free as in free speech, not as in free beer.”

The term “Free and Open Source Software” or FOSS is used to cover both kinds of software licenses.

Free Software requires a “reciprocal” or “enforced sharing” license, such as the GPL, which requires programs which are derived from Free Software to be distributed under the terms of the Free Software license. Open source licenses do not all have these obligations. Open source licenses such as the BSD license permit the incorporation of the software into other programs that may be distributed under non-open source terms.

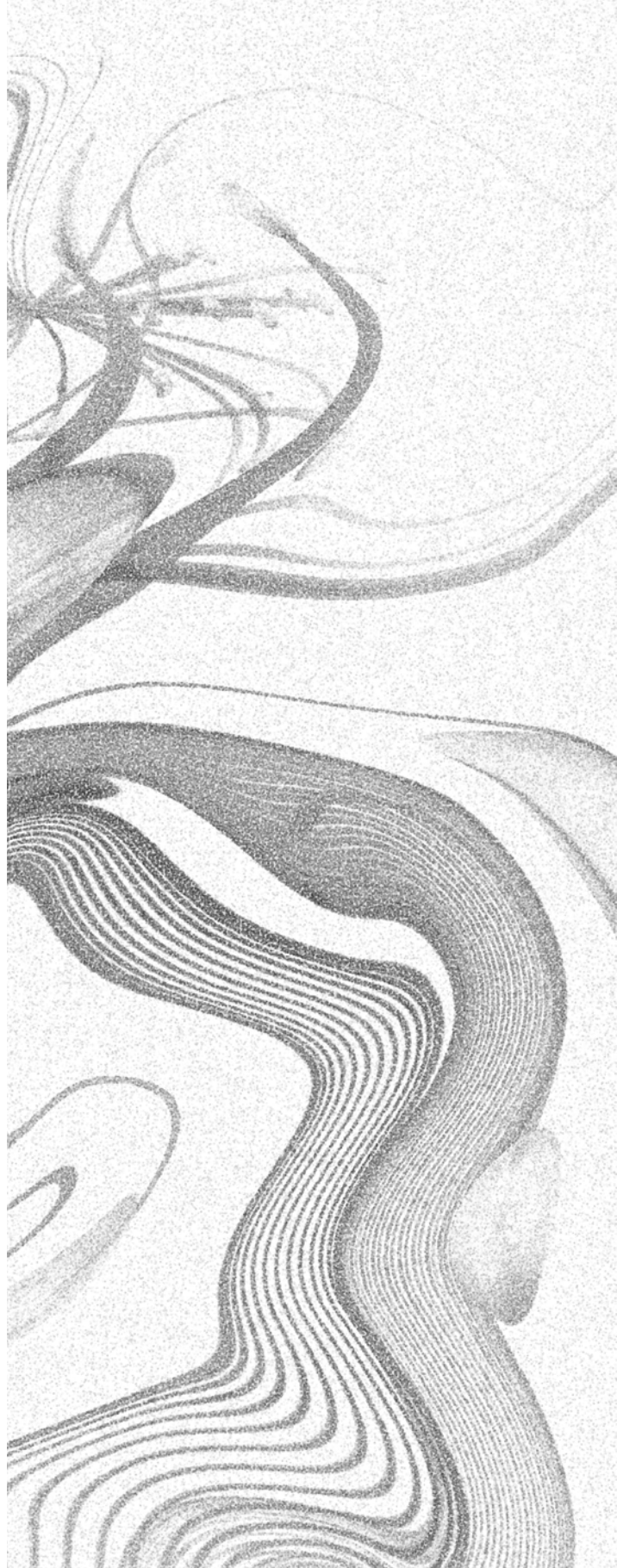
Dr. Dobb's: Is there a danger, legally speaking, in mixing closed and open source in the same project?

Copenhaver: As I mentioned above, Free Software licenses, such as the GPL, require programs which are derived from Free Software to be distributed under the terms of the Free Software license. In order to assure that all users have the freedom to change and improve the software, Free Software licenses require the source code for the software to be made available to anyone who receives the object code. If you combine closed and open source software in the same project, you may create a derivative work of the Free Software. If you distribute that derivative work, you will have to do so under the terms of the Free Software license and make the source code available to the recipients. It is important to remember, though, that not all open source licenses are Free Software licenses, and not all combinations are derivative works, and not all software is developed for distribution. The GPL is, however, by far the most popular of the open source licenses.

Dr. Dobb's: What's the main thing CTOs and CIOs ought to be concerned about when bringing open source into the organization??

Copenhaver: Education. Developers learn how to program using open source tools and components. Ignoring the availability and usefulness of these tools and components is either disingenuous or denial. If companies don't educate and have honest conversations that support deployment of open source software, they are just driving usage underground. But customers, strategic partners, investors, acquirors all want to know. But the real reason to educate is survival. The best and the brightest development talent does not want to work at a company that does not take full advantage of open source tools and components. The most cost effective companies do not re-write basic functionality. Companies cannot survive when their competitors have learned how to leverage open source while they continue to “Just Say No.”

[Return to Table of Contents](#)



Android Wireless Application Development Book Review

Reviewed by Mike Riley

Android Wireless Application Development
Shane Conder and Lauren Darcey
Addison-Wesley
\$44.99

Industry analysts predict 2010 to be the year of the Android OS. With the success of the Motorola Droid and devices like it on the verge of release, Android will accelerate its adoption curve. Now is the time to learn how to programmatically leverage this modern mobile operating system. Is this the book that can help developers achieve this objective? Read on to find out.

Android Wireless Application Development is the third Android programming book I have reviewed and is also the most comprehensive one I have read to date. While it doesn't cover everything (its discussion of threads was weak, and the use of threads in timers, working with alerts and other such topics were non-existent), the book provides enough solid foundation to construct a practical working knowledge of the platform.

Compared to the other Android books I have reviewed, *Android Wireless Application Development* does not contain a running tutorial building an Android application from the ground up. Rather, it consists of a series of code snippets illustrating a feature or solving a cookbook-style problem. The book, divided into seven sections, sets the stage with a general overview of the pre-Android mobile marketplace followed by a summary of what makes Android so special. The next section covers Android application design principles such as application lifecycle and resource management. Section 3 and 4 cover UI essentials and common APIs, respectively. Section 5 goes into a deeper explanation and examples of Android notifications and services. Section 6 on application deployment discusses the software methodologies best suited for Android development as well as the business cases around market feasibility and leveraging the Android Market as a sales and distribution platform. The final section consists of 4 appendixes detailing the Android emulator, Dalvik Debug Monitor Service (DDMS), Android Debug Bridge (ADB) and a primer on SQLite, Android's data store of choice. A nice convenience is the addition of a CD-ROM containing the book's source code along with several open source Android developer tools. Even though the code can be downloaded from the book's website,

the self-contained package adds to the book's overall value.

Due to the rapid pace of Android OS evolution, the book is already somewhat dated by the fact that it only covers up the 1.5 release of the OS. With the release of Google's own Nexus One running version 2.1, a handful of important changes and new features such as the Text To Speech engine, Google Map API enhancements and the like are obviously not covered in the book. Still, there are enough really helpful examples on a number of capabilities that have remained the same since the 1.x release. One of these that I found particularly helpful was a discussion beginning on page 397 regarding the ability to programmatically monitor battery status. Having written several Android apps that regularly poll network resources, I am acutely aware of the power such activity can consume. In an effort to minimize this impact, I modified the battery monitoring code in the book and reworked it into my apps to reduce the frequency of my program's network events when battery capacity is less than 50%. While I could have just as easily Googled for this code or queried a web site like Stack Overflow, taking the time to read this topic in the book is what ignited my incentive to modify my programs in the first place. It really helped to have the code wrapped in the context of a discussion on battery performance, which in turn, did a better job putting the topic into context of what the code did and why it was important to consider the impact of performance and battery life. Doing so also added a better degree of stickiness to the solution since I took the time to read, think about, re-read and implement the necessary logic to perform such an operation.

Overall, *Android Wireless Application Development* is ideally suited for developers already comfortable with Java and application basics of Android OS. For those new to the platform, I would recommend first starting with Ed Burnette's *Hello, Android* followed by this book for a strong basis to write sophisticated, compelling Android applications.

[Return to Table of Contents](#)

PaaS-onomics

Why perfecting every pixel in IT applications can be a waste of time

by Pankaj Malviya

Most of the effort in developing system functionality in today's business applications is not programming. It's the busy work. It's reinventing the wheel. I might even say that it's a complete waste of time. If you happen to be working on such a project today, I apologize if you take this the wrong way. It is not to say you are a waste as a programmer, but as you might have already realized, the cycles you are spending refining a proprietary business rule engine or an audit trail function are likely not worth the long hours you're putting into them, especially when you cannot leverage that functionality elsewhere. Too much emphasis is on application look-and-feel and building proprietary processing technology.

I recently talked to Izak Joubert, the CTO of NES Financials and a LongJump platform customer, who said that in the past, they had the time, money, tools, and resources to "perfect every pixel" of applications. As developers, they loved the control and the blank canvas to design beautiful applications.

But as businesspeople, that control and dedication wasn't enabling them to finish projects any faster or meet the challenges of a very dynamic business climate. Instead, responsiveness to change became more difficult because there was no framework and too much time was spent on accoutrements that don't matter in the back-office IT world. At the end of the day, NES Financials is a financial solutions company, not a software company, and application perfection was taking away from their core competencies.

We hear many stories about businesses that essentially need development tools to get their work done, but traditionally have only had what constitute designer tools geared towards producing a wide range of commercial software applications. As a result, their developers fall in love

with perfecting those pixels, because they can and because it's fun.

As developers, we rely on our ingenuity and hard work to keep us employed, but the truth is many of today's back-office IT applications don't need the sexy wow factor that commercial applications offer. An overwhelming number of business applications are not 99-cent, millions-of-downloads, mass market apps. After all, we're not releasing the next Guitar Hero or Tap Tap Revenge app. Back-office IT applications are hard-working, bread-and-butter applications. They don't need to be pretty. They simply need to make everyone's work easier, more accurate, and more effective.

Enter the advent of cloud computing and cloud application platforms or platforms-as-a-service (PaaS) — virtualized and scalable IT infrastructure married to adaptable application platforms. In the same way server infrastructure is outsourced and managed in a ubiquitous cloud, applications can have their "guts" outsourced to speed software development — including common building blocks like data modeling, security, data and workflow processing and reporting.

Applications built on these cloud platforms offer up real immediate value. The pre-built interfaces are utilitarian and pragmatic, while the platforms come with flexible data processing and reporting engines, and they're infused with the enterprise class features of workflow, permission management, and business object modeling every enterprise app needs. Add on layers of user self-service customization features, web services integration, API extensions, and versioning and deployment capabilities, and you have everything needed to build robust, serviceable business applications in a small fraction of the time with significantly fewer resources.

The value of building on the platform goes beyond the first application. IT can take advantage of the economies of scale, even in their own private clouds, and do so in a way that promotes agile development and enhanced internal customer service.

For example, using PaaS, you can build an order management application that might encompass data on customers, orders and inventory. Well, in that same platform, you can seamlessly reuse the orders data in your accounting application, and reuse the inventory data in an asset reconciliation application. Different applications for different departments — all using a common set of objects.

The entire organization benefits from less e-mailing of spreadsheets, less time spent building integration points, and less rogue applications to contend with. In addition, this process is building upon a foundational set of data that can be leveraged over and over again. When our customers need changes, most are menu-based configurations that don't involve custom code, so we can have solutions out faster and adapt quicker to new requirements.

Beyond the fiscal and time-saving benefits, there is also a strategic reasoning for PaaS that hits the crux of mission-critical application planning. IT needs to take a serious look at where the weaknesses are when it comes to information flow. Where is the weak link to giving information workers the data analysis and processing they need? Where are companies most at risk for a major breakdown in process, accountability, or compliance? In most cases, the trouble starts where there are no applications or where entrenched rogue applications have introduced chaos into the information landscape.

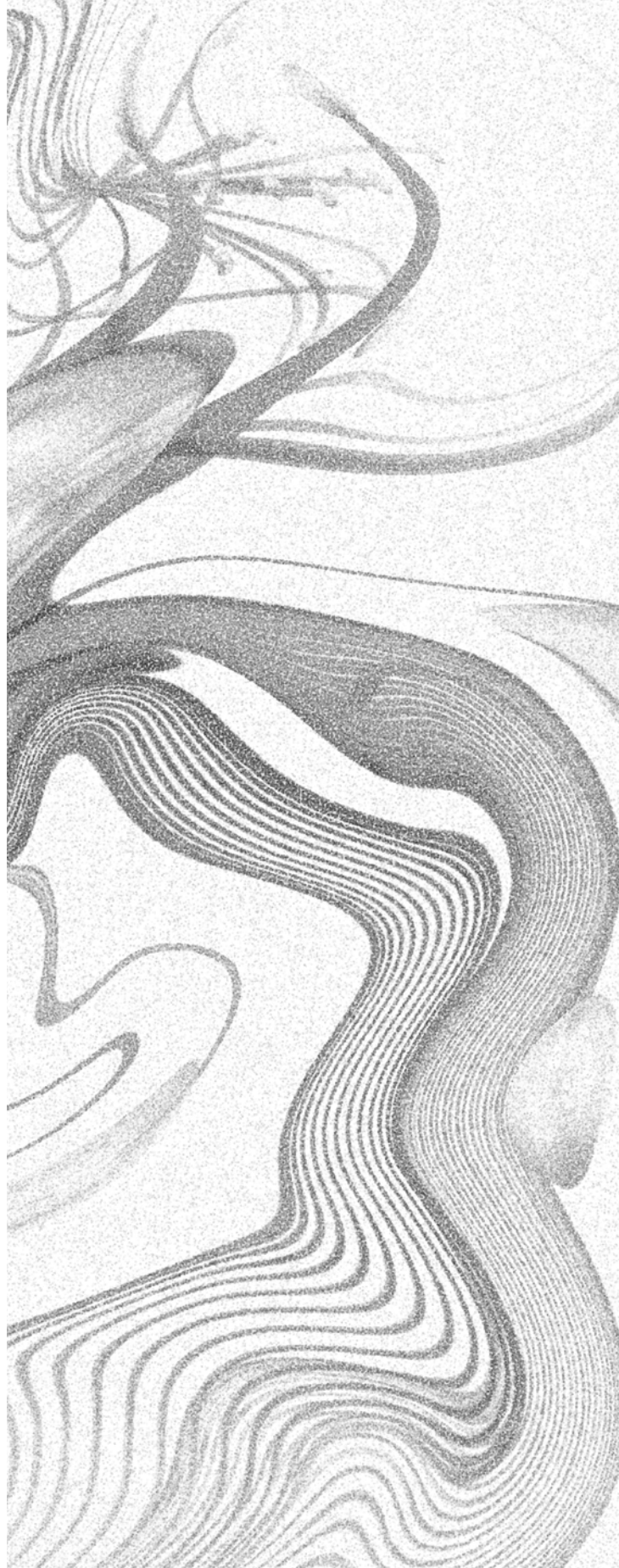
PaaS is essential to establishing an IT strategy for businesses to:

- Control and host applications from a single environment
- Standardize application architectures
- Quickly roll out changes while safeguarding information
- Give IT a leadership role in defining applications rather than having to support them after the fact
- Focus on core competencies

What would IT need to take advantage of a cloud/PaaS arsenal? First and foremost, it means adopting the mindset of solution architects focused on analyzing challenges at the information and business process level, not at the physical and networking levels. It also means wasting less time on what doesn't matter and instead focusing and redirecting resources into what does.

— Pankaj Malviya is CEO of LongJump (www.longjump.com/), a Platform-as-a-Service (PaaS) provider.

[Return to Table of Contents](#)



Prefer Futures to Baked-In “Async APIs”

When designing concurrent APIs, separate “what” from “how”

By Herb Sutter

Let's say you have an existing synchronous API function:

```
// Example 1: Original synchronous API
// that could take a long time to execute
// (to compute, to wait for disk/web, etc.)
//
RetType DoSomething(
    InParameters ins,
    OutParameters outs
);
```

Because *DoSomething* could take a long time to execute (whether it keeps a CPU core busy or not), and might be independent of other work the caller is doing, naturally the caller might want to execute *DoSomething* asynchronously. For example, consider calling code like this:

```
// Example 1, continued:
// Sample calling code
//
void CallerMethod() {
    // ...

    result = DoSomething( this, that, outTheOther );

    // These could also take a long time
    OtherWork();
    MoreOtherWork();

    // ... now use result and outOther ...
}
```

If *OtherWork* and *MoreOtherWork* don't depend on the value of *result* or other side effects of *DoSomething*, and if they can correctly run concurrently with *DoSomething* because they don't conflict by using the same data or resources, then it could be useful to execute *DoSomething* concurrently with *OtherWork/MoreOtherWork*.

The question is, how should we enable that? There is a simple and correct answer, but because many interfaces have opted for a more complex answer let's consider that one first.

Option 1: Explicit Begin/End (Poor)

One option is for the API itself to provide an asynchronous version of the function. This is such a

popular option that there are patterns for it, each of which has been reinvented with variations a number of times.

One form of pattern that is commonly used in .NET (as well as reinvented in other places, such as in C++ code inside Microsoft Office) is called the Frameworks Async Pattern. The idea in this pattern is to split the original method into a pair of methods, one to begin (launch) the work, the other to end (join with) the work. In a moment we'll see how this can get elaborate, with intermediate structures and explicit callback registrations, but here's the simple version:

```
// Example 2: Applying the basic “Async
// Pattern”
//
// The “Begin” part takes the input parameters.
//
IAsyncResult BeginDoSomething(
    InParameters ins
);
//
// The “End” part yields the return value and
// output parameters. Note: The caller must
// explicitly call EndDoSomething.
//
RetType EndDoSomething(
    IAsyncResult asyncResult,
    OutParameters outs
);
```

Here's how the earlier calling code would look, using this *BeginXxx/EndXxx* pattern to call *DoSomething* asynchronously:

```
// Example 2, continued:
// Sample calling code
//
void CallerMethod() {
    // ...

    // Launch, passing the input parameters.
    IAsyncResult ar = BeginDoSomething( this, that );
    result = DoSomething( this, that, outTheOther );

    // These could also take a long time
    // but now run concurrently with DoSomething
    OtherWork();
    MoreOtherWork();

    // Join, then call End to get the return
    // value and output parameters.
    ar.AsyncWaitHandle.WaitOne();
    result = EndDoSomething( ar, outTheOther );

    // ... now use result and outOther ...
}
```

This approach works. However, it has some drawbacks.

First, this pattern is intrusive and bloats the API. The API surface area has now tripled for calls that could be asynchronous, because instead of one function we have three that must be kept in sync. There is also the issue of consistency: The pattern may vary in that it depends on manual discipline for each API author to do it the same way each time. Further, the API designer has to know in advance which methods should support asynchronous calling.

We can eliminate this problem by providing a generalized form of the begin/end pattern that doesn't need to be baked into a specific API. For example, the .NET Framework provides a generalized *BeginInvoke/EndInvoke* that can call arbitrary methods asynchronously. [1] Unfortunately, this does not address the other drawbacks below.

Second, this approach becomes complex for callers, who instead of calling one function must now call three functions: one each to begin, to wait, and to end. It also has more potential points of failure where the calling code can go wrong. For example, if you end up deciding you don't need the result after all because your work is being cancelled or you called the function speculatively or for any other reason, do you have to call the *EndXxx* method? Typically, yes (and on .NET specifically, always yes), because the *BeginXxx* call will allocate resources to launch and track the work, and the *EndXxx* call releases those resources. Hence, the .NET Design Guidelines document says: "Do ensure that the *EndMethodName* method is always called even if the operation is known to have already completed. This allows exceptions to be received and any resources used in the async operation to be freed." This is a common source of errors as even expert programmers, and books written by true gurus, have got this wrong by thinking the *EndXxx* call was optional, or just forgetting to write it on all paths.

Third, this approach actually grows more complex because following this path leads to more complex and ornate styles. In particular, if you look at the actual implementations the begin/end pattern actually grows more parts:

```
// Example 3: The fuller "Async Pattern"
//
// The "Begin" part takes the input parameters
// and optionally a callback to be invoked
// at the end of the work, and a cookie to
// identify this invocation.
//
IAsyncResult BeginDoSomething(
    InParameters ins,
    // + caller can supply a callback notification
    AsyncCallback callback,
    // + extra state to identify "this call"
    Object cookie
);
// The "End" is the same as in Example 2.
RetType EndDoSomething(
    IAsyncResult asyncResult,
    OutParameters outs
);
```

The reason that we grow this requirement is so that callers can provide a kind of "finally do X with the result" handler in cases where the caller wants to return without necessarily needing or wanting to wait for the async call to complete. For example:

```
// Example 3, continued:
// Alternative calling code that doesn't wait
//
void CallerMethod() {
    // ...

    // Launch, passing the input parameters.
    // But don't join, just eventually use the result.
    IAsyncResult ar = BeginDoSomething( this, that,
        (Object myExtraState)=>{
            result = EndDoSomething( ar, outTheOther );
            // do whatever is necessary with result
            // (write to disk, update a GUI text box, ...)
        },
        new MyExtraState( /*...*/ )
    );
    result = DoSomething( this, that, outTheOther );

    OtherWork();
    MoreOtherWork();

    // ... now return without waiting for DoSomething
}
```

Fourth, and finally, all variations of this begin/end approach hardwire a specific way to perform the asynchronous call. The caller can choose to run the work asynchronously, but cannot choose how to run it — whether to run the work on a thread pool thread, on a fresh new thread, as a task on an automatically load-balanced work stealing runtime, on a particular processor core, and so on.

We definitely want to decouple the idea of "run this work asynchronously" ("how" to call) from any given API itself ("what" to call). We partly achieved that much with the generalized form of the begin/end pattern. But we want even more: We want to make the calling code simpler and more robust, and give the caller the flexibility of launching the work in arbitrary ways.

The good news is that we can do better.

Option 2: Decouple "What" From How"

Abstraction to the rescue:

- Use a separate general-purpose task launcher to launch the work. You probably have a number of options already available in your environment, such as being able to write *pool.run(/*task*/)* to run a task in a Java or .NET thread pool, or *async(/*task*/)* in C++0x.
- Use futures to manage the asynchronous results. A "future" is an asynchronous value — think of it as a "ticket redeemable for a value in the future." [2] This abstraction is available in Java as *Future<T>*, in the upcoming C++0x standard as *future<T>*, and also as *Task<T>* in the next release of .NET.

For example, here is a simple synchronous call to *CallSomeFunc*:

```
// synchronous call (this will block)
int result = CallSomeFunc(x,y,z);

// ... code here doesn't run until call completes and result is ready ...

// use result which is already available
DoSomethingWith( result );
```

Here is a corresponding asynchronous call (a mix of C++0x and C# syntax):

```
// asynchronous call
future<int> result =
    async( ()=>{ return CallSomeFunc(x,y,z); } );

// ... code here runs concurrently with CallSomeFunc ...

// use result when it's ready (this might block)
DoSomethingWith( result.value() );
```

The future allows us to decouple the call (launch) from the receiving of the result (join). This allows us full flexibility in how to launch the work without being invasive in either the API which remains synchronous and unaware, or the task handle which is represented as a simple and robust *future*. Also, we do not need to remember to call an explicit *EndXxx* method as the cleanup is encapsulated in the future abstraction (typically, the future object's destructor or disposer method).

Here's how it looks in the context of our original example. Note that there is no change to the original API:

```
// Example 4: Same original synchronous API.
//
RetType DoSomething(
    InParameters ins,
    OutParameters outs
);
// Sample asynchronous calling code
//
void CallerMethod() {
    // ...

    // launch work asynchronously (in any
    // fashion; for yuks let's use a thread pool)
    // note that the types of "result" and
    // "outTheOther" are now futures.
    result = pool.run( ()=>{
        DoSomething( this, that, outTheOther ) } );

    // These could also take a long time
    // but now run concurrently with DoSomething
    OtherWork();
    MoreOtherWork();

    // ... now use result.wait() (might block) and outOther...
}
```

Note: For convenience only, I'm showing the code using C# lambda syntax. If you don't have C++0x or C# lambdas available in your environment, you can still do this: Just replace "()"=>" with a separate Runnable object (Java), delegate (C#), or functor (C++). The lambda is just syntactic sugar for writing a runnable or functor.

What if we don't want to wait for the result but just want some final steps to be performed whenever the call completes, as in Example 3? Easy: Just make it part of the async work, no callback required:

```
// Example 5 (compare with Example 3):
// Alternative calling code that doesn't wait
//
void CallerMethod() {
    // ...

    // Launch, passing the input parameters.
    // But don't join, just eventually use the result.
    async( ()=>{
        DoSomething( this, that, outTheOther );
        // do whatever is necessary with result
        // (write to disk, update a GUI text box, ...)
    } );

    OtherWork();
    MoreOtherWork();

    // ... now return without waiting for DoSomething
}
```

Summary

How should we supply an async version of an API? Often the best answer is to do nothing at all, because a caller can make any call asynchronous externally using features like "async" or task launching together with futures.

If you are providing a framework or library or any other API interface, prefer to keep asynchronous launching ("how") separate from the task to be done ("what"). This path leads to simpler APIs, simpler and more robust calling code, and great flexibility in where and how to execute the work.

Notes

[1] "Calling Synchronous Methods Asynchronously" (MSDN, available online at msdn.microsoft.com/en-us/library/2e08f6yc.aspx.)

[2] In the past, people have sometimes used the word "future" mean both the asynchronous work and its result, but this is conflating two separate things. Always think of a future as just an asynchronous value or object.

—Herb Sutter is a bestselling author and consultant on software development topics, and a software architect at Microsoft. He can be contacted at www.gotw.ca.

[Return to Table of Contents](#)



Second-Guess Apple and Win a No-Prize!

Maybe it's just the giddy air of gadgetophilia that exudes from the Consumer Electronics Show at around this time every year, but it sure seems like permission to think outside the box has been granted to computer manufacturers recently. The box in question being the one on your desk or lap with its too-familiar WIMP interface. It started with the iPhone, of course, which demonstrated that a handheld device can to a large extent replace a computer, without benefit of a standard keyboard or mouse, lacking the obligatory windowing GUI, and while taking a radically different approach to files and applications. The iPhone didn't pioneer any of those things, it just mainstreamed them.

And I'm guessing that this is why we're seeing netbooks finally gaining ground, but netbooks are already looking very last-decade. Now we're being told that this will be the year of the slate, or tablet, or whatever you call it.

The uncertainty over the name of the product category reflects the squishiness of the definition of that product category. Some of the devices being demoed have actual keyboards, some virtual ones. They occupy various positions along the phone-to-computer bridge, many of them up on the railing without a bungee. Some are two-screen devices, with hinges and other moving parts that will probably break if the things ever make it from concept to product stage.

And there is a lot of naked concept dancing around out there and flaunting its stuff. Dazzling concept videos that demonstrate once and for all that the industry has learned how to produce concept videos. There's an amazing teaser for the next-after-the-next-gen OLPC (a tablet), floated before the next-gen OLPC (a laptop) comes out. I'd say that OLPC just pulled an OCC, but OLPC has a unique business model, so maybe they know what they're doing.

It's almost getting interesting.

But this is January, so the only thing we're allowed to obsess about is what universe-shaking something Apple will release. Because rumors are flying that Apple is about to announce that it will hold an event later this month to announce something. I've got tingles.

Will the something have a virtual keyboard that nevertheless provides tactile feedback? I like that prediction. 3D graphics? Post-Avatar, that might seem to be obligatory. Will it continue the evolution of the iTunes Store toward news-stand-slash-movie megaplex?

Speculations and predictions (or are those the same thing in this case?) are all over the map, but by a sort of Wisdom of Crowds logic, some kind of consensus ought to emerge. If it has, though, I haven't discerned it, and anyway, my money would not be on the crowd this time.

Nobody but Steve Jobs knows what the plan is, but that doesn't stop every pundit and his canine companion from guessing. I don't want to do that. I want to get you to do that.

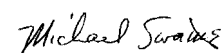
I propose a contest. But one with no prizes. Or, in the spirit of Marvel Comics guru Stan Lee, No-Prizes. I will award a No-Prize to the person who does the best job of answering the following questions about the rumored Apple device (or as Andy Ihnatko calls it, the RAT: Rumored Apple Tablet). These questions may not be the most interesting, but I've tried to focus on the more-objective aspects of the rumored device, so I have some hope of deciding who actually got it right.

So here are the questions:

- Price: Less than \$500, \$500-699, \$700-999, four figures.
- Processor: ARM, Intel, something else.
- Operating System: Basically the iPhone OS, basically the Mac OS, something significantly different.
- Screen Size: 7-inch diagonal, 10-inch diagonal, something else.
- SDK: Full SDK available on delivery or not.
- Name: Something involving Slate, something involving Tablet, something involving Book, something else.
- Ship Date: Before March 1, in March, after April 1.
- Number Sold in 2010: Your guess.

Email me (mike@swaine.com) your guesses and I'll announce the No-Prizes for the best guesser(s) shortly after the announcement. Don't miss your chance to have your name forever enshrined in, er, wherever we enshrine things nowadays. Remember: there is no cost to enter (and no prize, I mean a No-Prize, for winning).

As Stan would say, 'Nuff said.



By Michael Swaine

[Return to Table of Contents](#)