The Pragmatic Bookshelf

Software Sleuthing

PragPub

The Second Iteration

IN THIS ISSUE

Issue #75
September 2015

Contents

DEPARTMENTS

Published monthly in PDF, mobi, and epub formats by The Pragmatic Programmers, LLC, Dallas, TX, and Raleigh, NC. E-Mail webmaster@swaine.com. The editor is Michael Swaine (michael@pragprog.com). Visit us at http://pragprog.com for the lowdown on our books, screencasts, training, forums, and more.

ISSN: 1948-3562

On Tap

The Testing Issue

You can download this issue at any time and as often as you like in any or all of our three formats: pdf [U1], mobi [U2], or epub [U3]. You can also download our special issue on teaching kids to code in all three formats: pdf [U4], mobi [U5], or (you guessed it) epub [U6]. If you subscribed through an app and want to also get the pdf, mobi, and epub file download links emailed to you every month, just ask [U7].

Software Sleuthing

Adam Tornhill suggests you view your code as a crime scene [U8]. Testing has a lot in common with forensic science, and when you're doing it you may find yourself thinking like a bad guy — or like a clueless user. Today, when you might be developing apps for a spectrum of mobile devices or building a high-traffic always-on ecommerce site, deploying in the cloud and tapping into remote web services or writing robot-control software for a single-board computer, the detective work of software testing is getting more diverse and subtle.

This month we have two articles on testing, and links to almost two dozen more.

Keith Stobie focuses on testing for mobile development. Mobile app development is one of the main fields of software development today, and will just get bigger tomorrow. Testing is critical, but is often given inadequate attention in mobile development. Keith addresses this problem with a crash course in testing for mobile development.

If Keith Stobie has staked out the low end of the spectrum of platform and app size, Ryan Neufeld makes the other end of the spectrum his own. He explains simulation testing, an approach ideally suited to large-scale production systems generating tons of revenue, where the stakes are high.

We've been publishing articles on testing since 2009, nearly two dozen articles. In this issue we link to all of them, from an introduction to TDD from Jeff Langr to articles on testing for iOS, the cloud, web services, and the Arduino.

Also in this issue: Johanna Rothman and Andy Lester offer advice on hiring, Marcus Blankenship gives some tips to the developer recently given management responsibilities, and James Bonang concludes his series on teaching kids to code. We've got a bit of Swift code, a roundup of new programming books, and a puzzle.

So welcome to this issue of *PragPub*.

Who Did What

photo credits: Cover: Holmes [U9] by Graham C99 is licensed under https://creativecommons.org/licenses/by/2.0/ [U10]. Page 12: Detective [U11] by olarte.ollie is licensed under https://creativecommons.org/licenses/by/2.0/ [U12]. Page 27: Software Testing Service [U13] by testbytes is licensed under https://creativecommons.org/licenses/by/2.0/ [U14]. Page 29: Database Plan [U15] by tech_estromberg is licensed under https://creativecommons.org/licenses/by/2.0/ [U16]. Page 40: chimney swift [U17] by Ed Schipul is licensed under https://creativecommons.org/licenses/by/2.0/ [U18].

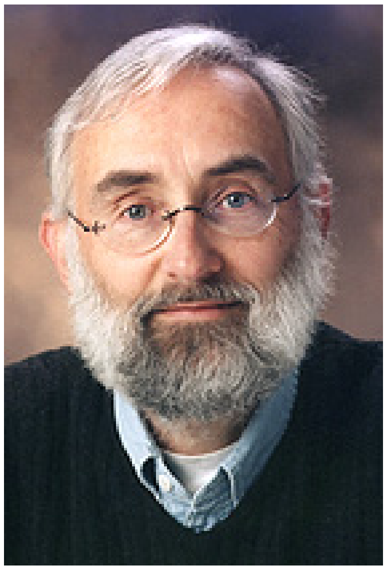
editing and production: Michael Swaine • editing and research: Nancy Groth • customer support, subscriptions, submissions: webmaster@swaine.com • staff curmudgeon: John Shade (john.shade@swaine.com)

Swaine’s World

Belly Up to the Bar

by Michael Swaine

Some choice bits to nosh on.



“It never rains in a pub.” — Traditional Irish saying

Here are a few choice bits to nosh on. Tweets. Links. Our Pub Puzzle. That sort of thing. Cheers.

#noestimates

- *Alternative to estimates: do the most important thing until either it ships or it is no longer the most important thing.* — @KentBeck
- *“We haven’t got time to automate this stuff, because we’re too busy dealing with the problems caused by our lack of automation.” —Everyone* — @bitfield
- *“It’s a heisenrequirement. We know exactly what it is until we ask for details.”* — @channingwalton
- *Just start referring to “estimates” as lies. “how long will that take?” “well, if I had to lie, a week?”* — @trek

The Pub Quiz

Here’s this month’s brain-teaser.

		T			O			
			I	R				
	I	S	T		Q		C	R
	R		K		C			
				O				
S						T		I
O								
			C					K
	K					Q	U	

It’s a sudoku, but with letters. Using only the nine letters that appear in the grid, fill in the empty cells so that every row, every column, and each of the nine 3x3 boxes contains all nine letters.

And it's also an anagram. Properly arranged, the nine letters in the puzzle will reveal a secret word every programmer should be familiar with.

Solution further on in the issue; don't peek!

Google Good, Alpha Better

You had me at A, Alphabet.

- *Google release notes: - Our new name is Alphabet! We hope you like it. - fixed a bug where we were paying too much tax.* — @hondanhon
- *The new parent company will be able to pursue new areas of interest that had previously been off-limits to Google. pic.twitter.com/cQQ6A6cOKO* — @tsrandall
- *The English language has a cruel sense of humor. Do you realize how many anagrams there are for dyslexic?* — @pragpub

Stop Making Sense

- *When did full stack developer start to mean "JavaScript all the way down"?* — @bphogan
- *You think email doesn't scale? I agree. It doesn't. But you know what scales exponentially worse? Chat.* — @codinghorror
- *Unstructured metadata (pizza edition) AKA: "Everything goes in the CDATA field, eventually."* — @doctorow
- *I read that the French government is developing a French language-only version control system. Could this be legit?* — @pragpub
- *"The end point of rationality is to demonstrate the limits of rationality." —Blaise Pascall #zenmonday* — @PragmaticAndy
- *Enjoy the blue moon tonight. But if you'd like it to be another color just let me know. Seriously it's no trouble.* — @BoredElonMusk

Etc.

- *The industry's dirty secret: The #1 thing keeping developers from quitting is the horror of setting up the dev environment in a new job.* — @raganwald
- *The we we were would never understand the we we are. "Dogfooding is a powerful resource allocation heuristic."* — @pragpub
- *Could be "turn spellcheck on", could be "underline every word in red." Hard to tell the difference sometimes.* — @PragmaticAndy

The Punters

Here are the folks I followed this month: John Arundel, Jeff Atwood, Kent Beck, Bored Elon Musk, Reginald Braithwaite, Dan Hon is typing, Cory Doctorow, Trek Glowacki, Brian P. Hogan, Andy Hunt, Christopher Mims, Tom Randall, Michael Swaine, and Channing Walton. But fair's fair. You can follow me at www.twitter.com/pragpub [U1]. Or visit my blog at swaine.com [U2].

- *And how many cars did you sell in your downtime while building a rocket company?* — @BoredElonMusk

Rothman and Lester

Finding the Right People

by Johanna Rothman and Andy Lester

Managing your career is a job in itself. Fortunately, Johanna and Andy have seen all the career mistakes people make and can steer you past the hidden rocks.

Johanna: In our August column, “Selling the Job,” we discussed how you might define your development job so candidates can screen themselves in or out. In this column, we’ll offer you ways to find the right people who fit your open req.

If you’ve tried to find people for your open positions, you might have encountered these problems: too many people apply for the job; no one is quite right; everyone looks like a carbon copy of everyone else; and more problems. You can avoid these challenges by changing the way you look for people.

Andy: That “carbon copy” issue is a big one. If everyone you hire looks like you and is as old as you and acts like you, you’ve got a department that is just one person, multiplied. A healthy working group has different people with different backgrounds, different ages, different levels of experience. We tend to like people who are more like us, but that makes for a monoculture that doesn’t have the benefit of diversity.

“Diversity” isn’t a code word for “you should hire people other than white men.” So when you start looking for people, consider what actually matters to you in a candidate, and not just what makes you feel more comfortable. Unfortunately, human nature makes this easier said than done.

Johanna: I have been quite successful when I hired people who were quick to learn, even if they had no domain expertise in our kind of product. That allowed us to make better decisions than we might otherwise would have made. (See Great People Create Great Products [U1].)

As Andy said, we like to hire people more like us. If you are hiring anyone [U2], be aware of your biases, so you can decide what to do about them.

One of the things I like to do is to vary where I find people. Even if I use a recruiter, I often attend meetings where the people I might like to hire will be. I might go to a Java user group for a Java developer, a PMI meeting for a project manager, an agile group of some sort when agile experience is the most important thing to me. I’m an equal-opportunity meeting participant!

I find that I meet a variety of people at these meetings. When they ask for announcements, I say, “I’m hiring a developer. If you would like to work at my company, please see me afterwards. I’ll stick around until they kick us out.” I bring the job description on a flyer, so people can take it home. I have found that it’s easier for me to hire unique people that way.

Andy: User groups are great. I once hired a guy who came up to me at a Perl Mongers meeting to show me some code he’d written. He knew I was the creator of the WWW::Mechanize Perl module [U3], and he’d used it to write a program to scrape his book checkouts and overdues from his local public library, and then display them on his TV. He was still in high school, but I went back to my boss and said "Can we make an internship for this guy over the summer?" That kind of enthusiasm is hard to find.

Even if you're not at a meeting, keep your eyes open. Back around 2000, I was on a commuter train and saw a woman reading *Webmaster in a Nutshell.* You don't often see someone reading an O'Reilly book out in the wild, and we had just opened a new programmer position in my department. I introduced myself, handed her a company business card, and said we were hiring. A few days later, she emailed HR a re sume and we eventually hired her.

Be open to networking in all ways, not just in formal networking situations.

Johanna: Another problem is when we find candidates, and they aren't quite "right." Maybe they meet all the qualifications of the job analysis you did. And, they still are not quite right. In that case, you might not have accounted for your organization's culture.

Culture is what you can discuss, what you reward, and how people treat each other. I've had times when I had candidates who wanted to discuss loudly and the group was soft-spoken. The group had a terrific way of working through conflict, but it wasn't loud.

I've also had experiences where candidates expected to perform and to be rewarded for heroics, but the team didn't work that way. The team thought it was more important to prevent problems rather than solve them. That was a culture mismatch.

Andy: So how do you handle that? Do you have a way to "make them fit," to help the candidate become a better fit for the group? Or is it something that can't be changed, and you just have to say goodbye to the candidate?

Johanna: I've discussed this with candidates. I've asked behavior-description questions such as, "Tell me about a time you felt you had to work overtime." I hear about that. Then I ask, "Have you ever been in a situation where you could prevent problems rather than work many hours?" If they have tried prevention first, I ask, "What kinds of things did you do in your most recent project to prevent craziness?"

Some people work in reactive mode at work, because that's how it is there. In that case, I ask them how they solve problems at home. "Can you provide me an example of a time you worked on a project — any project — that you prevented problems rather than dealt with them as they arose?"

If I can find evidence of a reasonable fit in their personal lives, and if all else is equal, I'll proceed. If it looks as if even in their personal lives they don't work the way we need them to, I'll say goodbye.

Andy: Do you have that written up somehow? Do you have some sort of standard vision of how they should work? That seems like something you can tell them to let them know what is expected. I know that candidates will often say what it is that we want to hear, but I think it also makes sense to spell out explicitly what the expectations are. "We typically work a 40-hour week, but twice a year during conference season, we'll do 50-60 hours a week. Does that sound like something you can handle?"

Johanna: Yes, this is part of the job analysis that we discussed back in "Selling the Job." Sometimes people leave jobs because they don't work in the job the way they want to work.

Let's summarize. If you want to find great people, make sure you stay away from "mini-me's." Look in different places such as a variety of user groups or professional groups. Be aware of people you meet socially who sound like they might be interesting. Understand your culture and how you might look for evidence of that culture in a candidate.

About the Authors

Johanna Rothman helps leaders solve problems and seize opportunities. She consults, speaks, and writes on managing high-technology product development. She enables managers, teams, and organizations to become more effective by applying her pragmatic approaches to the issues of project management, risk management, and people management. She writes the *Pragmatic Manager* email newsletter and two blogs on www.jrothman.com [U4].

Andy Lester has developed software for more than twenty years in the business world and on the Web in the open source community. Years of sifting through résumés, interviewing unprepared candidates, and even some unwise career choices of his own spurred him to write his nontraditional book Land The Tech Job You Love [U5] on the new guidelines for tech job hunting. Andy is an active member of the open source community, and lives in the Chicago area. He blogs at petdance.com [U6], tweets at @petdance, and can be reached by email at andy@petdance.com.

External resources referenced in this article:

[U1]	http://www.jrothman.com/pragmaticmanager/2012/12/great-people-create-great-products/
[U2]	https://pragprog.com/book/jrgeeks/hiring-geeks-that-fit
[U3]	http://search.cpan.org/dist/WWW-Mechanize/
[U4]	http://www.jrothman.com
[U5]	http://www.pragprog.com/refer/pragpub51/book/algh/land-the-tech-job-you-love
[U6]	http://petdance.com

New Manager's Playbook

How to Really Put Down the Tools and Get Back to Being a Manager

by Marcus Blankenship

You're a developer, but you're at a point in your career where you find yourself managing others. Marcus shares tips on how to be as good at managing as you are at your "real" job.

This month: putting down the tools.

New tech agency owners make all sorts of mistakes. Some are small and private, some are very public — and embarrassing.

But there's one mistake all managers make during their first year, and it's this: they don't put down the tools [U1].

The problem is that when the principals insist on trying to be programmers as well as managers, things fall apart. You can't work on the assembly line, try to keep it running, and maintain your sanity.

But, But . . . What if I Can't Put Down the Tools?

We all fall into the trap of trying to be the hero when the house is burning down. It's a great and noble intention.

And in times of stress, it's easy to try to depend on your own uber-geek brain for a hands-on solution. I understand.

However, when you know this cardinal rule and you pick up the tools anyway, there are only a handful of reasons. If you see yourself in one of these examples, never fear. You can get the work done and be an effective manager.

Why You Think You Can't Put Down the Tools

I understand the pressures of being a leader while producing high-quality code under the scrutiny of your team and your clients. Please understand that I'm not placing blame — I'm shining a light on false beliefs that will handicap you until you give them up.

1. *You don't want to*. Even though you're running the whole show, you still have the impression that your primary value to your company is in your abilities as a programmer. Unless you're coding, you don't feel like you're actually working.
2. *You just can't help yourself*. No matter how hard you try, you can't step away from the IDE. It's a compulsion, plain and simple.
3. *You believe that coding should be a significant part of your job description*. You'd like to focus on your management duties instead of your coding duties, but you're afraid you'll lose respect or your skills if you do. You may have worked in an organization that required managers to code, and you're still clinging to that culture.

Breaking Free of Coding

Seeing the problem is great, but having real solutions is even better. After years of hunting for decent role models and experimentation, here's my go-to list of strategies to stay out of the IDE.

You don't want to stop… but you could

Admit that you love coding, you're proud of your skills, and you're afraid they're going to deteriorate as you spend less time in front of a screen.

Yes, eventually you will see other team members' skills bypass your own, but you now have the ability to build teams, not just software. *There's so much you can accomplish even though your hands aren't creating software every day*.

Take a good look at why you wanted to own your own company in the first place.

Was it to do even more coding? Of course not. Seek out the places where you can identify and remove roadblocks for your team. You want to train yourself to think like a manager and business owner, not a programmer.

Remember that this is a process

If you are committed to the idea of being a great leader (are you?), you will have to put down the tools more and more, especially if you are managing a team of more than two people. Even if it feels awkward now, lay in a plan to release more and more of the programming to your team and dedicate that time to management activities.

You want to stop . . . but you keep getting sucked back in.

Watch for behavioral triggers

Pay attention to what you're thinking and saying just before you stalk over to the computer — again. Chances are, this is what's going through your brain:

- "They don't get it!"
- "I'm the only one who understands this stuff."
- "Why can't anyone else figure this out?"

Sound familiar? When you hear this kind of blaming and frustration, it's a clear signal that you need to invest more in your team. *If their skills are lacking, then it's on you to get them up to speed*.

Think of it this way: If you don't take the time to teach them these skills, what's going to happen when you go on vacation or have an emergency? Take off the hero's cape and build a team that can support you.

Become a force multiplier

It's easy to jump back in, thinking that one more body = more productivity.

But it doesn't.

If no one has an eye on the production line, then details get dropped and small problems get out of control — quickly. Yes, even if you have a very lean team of two-to-four people, a dedicated manager can make a massive difference.

With a leader in place, keeping downtime and preventable errors to a minimum, those two or three programmers' productivity can increase 10-to-25% (I've seen it!), completely surpassing the contribution you could make in your distracted and stressed-out state.

You can't shake the idea that you need to code

It takes a lot of courage to overcome previous habits, but it can happen.

When I drew a line in the sand and made a plan to pull back on my coding responsibilities, it was frightening. I thought I would throw production into chaos. I decided to try a 3-month experiment, which completely proved that coding less made my team stronger. But if your situation doesn't allow for that …

Get some perspective

Ask other agency owners how they wedge in coding with their other responsibilities. I'm guessing you'll hear that they're near the breaking point as well. But if they're doing well, ask them to mentor you.

Prioritize the team's needs over your own

No, this doesn't mean that you should work yourself to the breaking point. I'm suggesting that you focus on the activities that make the team more effective, and put your efforts there instead of coding.

Maybe that means having a stand-up meeting every day or making sure salespeople and clients talk to you instead of your team members. Find the black holes and manage them.

Keep a log of time spent coding

Find out exactly how you're using your time, then start investing more of that time in management activities, like training, delegating, and communicating.

Management Must-Dos

The best way to keep yourself out of coding's powerful tractor beam is to commit yourself to management activities. Just as smokers need to replace the satisfying rituals they used before lighting up, you need to develop alternate habits to keep from falling back into a programmer mindset.

Start your day with some management activity to put yourself in that headspace first thing in the morning. Don't get sucked into dealing with bugs right off the bat.

Schedule meetings for code review, planning, resource allocation, etc. Actually put them on your calendar so you drive home their importance.

Make a plan to reduce your coding time to 0% so you can manage 100%. Follow your instincts when you find yourself thinking, "I should communicate with my team more." Please don't be a skeptic.

I know it’s hard to believe that letting go of coding responsibilities doesn’t equal lost production and expertise, but leveraging your time for better team dynamics will make everyone’s life easier.

About the Author

Nearly 20 years ago I made the leap from senior-level hacker to full-on tech lead. Practically overnight I went from writing code to being in charge of actual human beings.

Without training or guidance, I suddenly had to deliver entire products on time and under budget, hit huge company goals, and do it all with a smile on my face. I share what I’ve learned here in *PragPub* and here [U2].

External resources referenced in this article:

[U1]	http://marcusblankenship.com/chapter-3-putting-down-the-tools-avoiding-the-1-mistake-all-new-software-company-owners-make-in-a-crisis/
[U2]	http://marcusblankenship.com

A Crash Course in Mobile Testing

Everything You Want to Know but Were too Busy to Ask

by Keith Stobie

Testing for mobile app development is rarely done, or done well. Here's how to change that.

Mobility is where it's at. A quarter of all IT budgets are slated to be spent on mobility by 2017. Mobile app development is one of the main areas of software development today. And whether you're a consumer app creator or an enterprise developer, you recognize the critical importance of quality in mobile development. We all do.

So how come so many mobile teams perform little or no testing?

Whether it's lack of tooling or a lack of awareness of the unique challenges of mobility, there's a testing gap. What I hope to do in this article is to try to narrow that gap by looking closely at the tools and the processes needed to create a testing program. I'll do this from the perspective of a business entering mobile development for the first time. Consider this a crash course in testing for mobile development.

Manual and Automated Testing

I'm sure you're clear on the difference between manual and automated testing, but it's important to understand the distinctive roles they play in mobile development. Manual testing introduces creativity and the flexibility of human observation, but it is costly to scale across platforms and subject to human inconsistencies. Automated testing provides repeatable benchmarks and scales well, but it can easily miss subjective or out-of-scope issues manual testers would notice. The two approaches have their respective strengths and weaknesses, and you need to use them in a complementary fashion to leverage their strengths while minimizing their weaknesses. And that balance will be different for each industry and type of deliverable.

Let's get specific. Testing typically involves at least six distinct steps. These steps can be represented by the acronym SEARCH.

- Setup: Getting the environment into the desired state to start
- Execute: Applying inputs and gathering outputs
- Analyze: Using a test oracle to determine if results appear correct
- Report: Aggregating the results of testing
- Cleanup: Returning resources to efficiently allow other testing
- Help: Documentation about the test

Any of these can be automated or manual. For example, you could automate execution but have a human analyze screenshots for correctness. Likewise, you could automate setup and cleanup, then manually execute and record the test session itself.

So that's the breakdown by steps, but an organization's testing efforts mature over time and become more sophisticated, typically following a five-stage evolution:

- Stage One: Basic manual and automated testing aimed at reaching broad functional coverage
- Stage Two: Supplementing simulator-based testing with a range of real devices
- Stage Three: Integrating testing more deeply with development processes
- Stage Four: Addition of more advanced concerns, such as data leakage and security, as well as new data inputs from outside the QA lab
- Stage Five: Extending outputs of testing to guide development and product planning

The choices you make in each step of a testing program, especially how you balance automated and manual testing, can differ depending on the stage of evolution your company is in. Let's look at stage one, where your brand-new mobile startup may be.

Stage One: Basic Manual and Automated Testing

The first step toward quality often begins by creating the broadest amount of coverage as quickly as possible. Many teams begin with manual testing, since testers are already product and domain experts, and can begin testing functionality immediately.

Manual tests are a natural starting place with a low barrier to entry, which is good. But when you use them exclusively, if you aren't careful, the result can be repetitive, low-return results. Manual testing works best when driven by explicit expectations of very targeted outcomes, such as verifying the usability of a certain feature. You should also leverage the essential creativity of human testers by scheduling manual exploratory testing using charters, tours, or other techniques.

Most Agile teams already know they should incorporate testing up front using both Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD) or Behavioral Driven Development (BDD). How do these approaches work in mobile development?

Unit testing techniques and frameworks are well suited for mobile TDD, but ATDD can be a challenge. But behavior can be described using a Gherkin-like language like Cucumber or a lower-level BDD language like RSpec or NSpec, and you can build interfaces using common programming languages such as Ruby or C#.

Automated UI testing in the mobile realm frequently focuses on navigation of the UI. But behavior can be driven below the level of the UI, at a model (instead of presentation) or business-logic layer. This usually lowers maintenance costs, as the UI typically changes faster than underlying business logic. Use Page Object Pattern to minimize maintenance cost when driving behavior through the UI.

In the earliest stages of a product, feature sets and user interface can change rapidly, making maintainable UI automation impractical and costly. At the start of a project, developers should focus on automating discrete unit tests, then automate larger UI tests as the app takes shape. While you can easily use a capture-replay for throwaway automation, designed automation needs some level of stability to be cost effective.

At this level, simulators play a large role in testing efforts, due to their ease of setup and control. Developers often use simulators as a pre-check before committing code. They typically catch problems that simulators miss either through beta-testing, ad hoc checks on real hardware, or (worst case) user feedback after launch. Incident response is typically lightly structured, often including direct delivery of point fixes to affected users.

In practice, Stage One teams often begin their test planning only after app development has begun. As a result, they spend their early cycles playing catch-up, and only begin executing tests in earnest toward the end of their development cycle.

Setting Standards

Then there are those darned standards. While test suites must validate app functionality, there are other equally important concerns they must address. At a minimum, apps need to meet the standards of the marketplace where they will be distributed. Consumer apps need to meet the requirements of Apple's App Store, Google's Play Store, or Microsoft's Windows Store, while enterprise apps may have to meet standards dictated by Legal or Human Resources departments or external partners. Apps of all sorts may also be subject to specialized compliance regulations such as HIPAA, PCI, or SOX.

Stage Two: Real Device Testing

After a few development cycles, testing can use its newfound experience to inform better planning. Even if test execution will still occur later in the process, test planning can begin much earlier and start to address the coverage gaps of previous cycles. One of the most important gaps to address is device coverage.

By this point, it has probably become obvious that what seemed to be sufficient testing on your simulator (and any available local devices) is insufficient for understanding real-world users' issues on diverse devices and environments. Your test automation framework must incorporate compatibility testing by using real devices. Remote Device Access (RDA) or cloud services are a big help here, providing access to a live device over the Internet and enabling you to load your app and test it on the real device remotely.

For both manual and automated testing, you should focus on a set of real devices that best reflects your current or target user base. The right mix will vary by market: North America has more iOS users than Europe and its most popular operator uses an entirely different network technology. Chinese consumers use a different balance of devices and many of their most popular devices exist nowhere else. The fast-growing markets of Southeast Asia are in flux, without any truly dominant platforms.

You must balance the cost of covering more devices with the risk of issues arising from not covering them. You're probably not going to find it a viable option to manage this cost and complexity in an internal device lab. Acquisition, maintenance, and device orchestration are all cash- and resource-intensive, and your needs may shift with changing strategy or commercial trends.

You may also be able to incorporate your automated functional testing against real devices into your Continuous Integration (CI) mechanism, discussed in Stage Three. Even if your company has not yet adopted CI, you should lay the groundwork by incorporating passing of an automated test suite on real devices into your build acceptance plan.

Manual Testing in Stage Two

It's important to remember that while you're spinning up automated testing in the cloud, the manual testing resources you put to work in Stage One remain an essential part of your strategy. Manual and automated approaches work best in tandem.

One such hybrid approach is to run an exploratory manual test and record it, then run the recording in automated fashion against other devices. This can help in discovering crashes on platforms you don't have time to test manually, and with proper tools and setup, you can manually review screenshots of results to identify further avenues for exploratory testing. You can also run manual tests while the system is under load from automated tests. This blend of functional/load testing often reveals flaws that would not surface under typical pre-production conditions.

What to Look For

Many experienced testers are still inexperienced in mobile development and may therefore overlook some mobile-unique errors. Many of the more common areas of concern include:

Installation: The first interaction users have with your app is installation. Too many teams under-test installation, and that's fatal. A failed installation is a lost user. Install your app on real devices if you want to have a hope of fleshing out these onerous issues. And don't forget to verify uninstallation. Incomplete or difficult uninstallation can get you nasty reviews.

Mobile-specific UI: Mobile users interact with apps differently in different contexts. At the very least, your test plans should cover landscape and portrait orientation across all use cases, and also integrate multi-touch gestures, zooming, and other UI features native to devices. In many cases, the "fixes" to these issues may actually require policy decisions from the product owner.

Social and connectivity: Does your app sync? Then you need to include sync testing. Check the app sync intervals. Many apps now involve social networks, location (GPS), and other mobile device sensors (camera, motion, etc.). Pay attention to verifying their functionality. For example, try to send an email, post, or comment without configuring the companion account (email, Facebook, Twitter). Try the same after configuring the accounts. Set up mock GPS for GPS-related functionalities.

Error handling: Check whether proper error or success messages are displayed, especially during first launch of the app with no network or location or social authentication. First launch has been a common issue for many apps. Throughout your testing consider the following common situations:

- Network connection not available
- Network connection weak or sporadic
- Third-party errors (often from social media or email) that impact app behavior

Stage Three: Beyond Functional and Usability Tests

Congratulations: you've embraced the goals and practices of Stages One and Two and as a result have made great strides. Now you're in Stage Three.

Test planning now occurs earlier and testing is an essential part of build approval. Stage Three raises the bar to include test design up front and testing as an essential part of CI processes. Acceptance tests are created alongside stories or feature approval at the start of development and are essential checkpoints on the way to accepted builds.

At this point, it's important to broaden the scope of tests to ensure comprehensive coverage across all dimensions of app behavior. It's certainly important to get functionality and usability right first. If your app isn't usable or functional, nothing else matters. Skipping functional and usability tests to focus on performance and stress tests before you've confirmed the basics generally backfires. Still, once the basics are covered, a mature acceptance test suite should address more than performance, stress, and interrupt testing.

Performance Testing

Users are rightfully demanding much more than a usable and functional app. Performance is one of the most obvious areas to cover. You can start with simple measures; for example, how long do your automated unit and functional tests take to run? Is your app getting slower over time as new features and functionality are added?

Your test environment should provide a wealth of information for mining. Is the app taking longer to load or change screens or using more memory in later builds versus early builds? In some cases (such as the addition of new functionality), this might be expected, but it should still be monitored. Do you have a performance budget? What is the maximum time to respond? What is the maximum memory to use?

You can also modify functional tests to target performance specifically. You can loop over simple common patterns or provide more data that takes longer to process. Performance testing is a big topic and I can't fully cover it here, but you need to keep in mind that performance is almost always a key criterion that your users will be using to judge your app.

Stress Testing

Stress testing mobile apps is key to identifying many reliability issues users will see once an app is in production. Some common types of stress testing include:

- Capacity/Load/Volume: lots of data or content
- Lots of continuous usage: flow issues and memory usage
- Background processing: little cpu, memory, or network available
- Connectivity: slow network, turning network on/off, switching Wi-Fi/LTE

Again, this needs to be done with real devices, as the memory, CPU, and networking of each device can be different.

Most stress testing must be automated to some degree. As mentioned earlier, manual testing while automated stress conditions are present can reveal issues that automation alone may not recognize.

Another form of stress testing is "robot" or "monkey" testing, in which the test automation performs purely arbitrary walks through the UI. In essence, the automation randomly chooses actions on the screen and the simplest verification is that the app didn't crash and an "Application Not Responding" (ANR) pop-up didn't occur. This is relatively cheap to build and run and even the simplest implementation can (initially) find issues. Robot testing can sometimes also address the continuous usage stress case.

Manual testing can be enhanced with automation in ways other than those already discussed. Setup and Cleanup are also areas that automation can expedite. Creation of automation scripts for log on, data loading, etc. allow manual testers to focus on usability and exploring instead of manual script execution.

Interrupt Testing

Interruptions are far more common in mobile apps than in web or desktop applications. In addition to the network interruptions mentioned above, there are also interruptions due to a number of mobile-specific causes, such as:

- incoming calls
- texts
- notifications from other apps
- power issues (e.g., battery drain or power disconnect)
- airplane mode

Instrumentation

Consider instrumenting your code to capture feedback from real user testing. Instrumenting your apps can provide a substantial boost to tests at all stages of the mobile application life cycle. For internal tests, instrumentation can provide forensic information to be mined later, after testers discover a flaw. It also allows developers to conduct beta tests in which they tie objective app-reported usage data to user-reported survey data. External testers may do exploratory testing of your app in ways you never considered.

Ecosystem Integration

Does your app interact with other apps and services? Even if it doesn't, ensure that installation of your app doesn't damage existing apps or the device.

Admittedly, this is easier said than done. What are the typical apps on your users' devices? What do you share (beyond the obvious CPU, memory, battery, and possibly screen)? Do you interact with GPS, camera, etc. along with other apps?

Stage Four: Advanced Testing — Models

As the simple and obvious defects get removed in the prior testing levels, more sophisticated testing is required to flush out the bugs still lurking. Use data from monitoring your users to guide where to put efforts here. It's easy to run tests that find bugs that no user will ever experience. Your focus has to be on user experience and the return on testing investment.

Security

As consumers become more aware of the value of their own data and businesses become more dependent on mobility as a core platform, security becomes a must-have. See OWASP Top 10 Mobile Risks and Mobile Security Testing [U1] as a primer. Many aspects of mobile security testing focus on network communication between the app and the cloud, but even a local-only app requires security testing.

In many cases, the type of app or the industry it supports will determine where to start with security testing, but there are several types of security that all app developers should address, including:

- Data leakage: Is data leaking to log files, or out through notifications?
- Data validation: App doesn't validate input, allowing crashes and exploits.
- Data flow: Can you establish an audit trail for data? What goes where, is data in transit protected, and who has access to it?
- Data storage: Where is data stored, and is it encrypted? Cloud solutions can be a weak link for data security.
- Authentication: When and where are users challenged to authenticate, how are they authorized, and can you track password and IDs in the system? Be careful about authentication tokens or passwords for online services being stored in the app itself.
- Points of entry: Are all potential client-side routes into the app being validated?

You must also perform penetration testing, including:

- Intercepting and inspecting network protocols with man-in-the-middle attacks.
- Delivering client-side exploits to vulnerable devices, or manipulate captured traffic to exploit supporting backend mobile app servers with traffic insertion attacks.
- Bypassing device passcode use by physically connecting the device to an attack workstation to root or jailbreak the device, exposing the file system data.

- Inspecting commonly sensitive data areas on mobile devices for information such as the Notes, SMS, and browser history databases.
- Looking for stored passwords or other sensitive data from your app.

Security testing is a specialty deserving of a much deeper dive than this article allows, but it is important and should be a focus for every developer once they reach Stage Four, if not earlier. Also consider the Open Android Security Assessment Methodology [U2] or the IOS Application Security Testing Cheat Sheet [U3].

Models

Beyond the random UI walks of Stage Three's "monkey" tests, testers can program more sophisticated navigation testing models using data collected from real usage. Automatically generated models allow deeper exploration than purely random models, with more realistic, longer, deeper sequences. Sophisticated models can check for correct behavior beyond just whether the app crashes. Automatically built models can expose bugs with a high degree of confidence when current app behavior differs sufficiently from prior experience.

Stage Five: Optimized Testing

By Stage Four, teams are building secure, thoroughly tested code in an optimized environment. Stage Five is about extending that optimization through to the production cycle, closing the feedback loop. Look for ways to use testing to guide planning and development. The most well-known is A/B testing, in which developers compare different UIs to determine which provides the best experience or most profit. Multivariate testing allows comparing more than a binary set of changes, but the techniques based on Design of Experiments can let you go further than optimizing the UI. You can optimize performance, battery life, or other attributes.

Conclusion

Although we have presented five distinct stages in a fairly common progression, there is no one path. Some app teams may have started with instrumentation and robot automation, and added manual testing later, while others might have begun with A/B testing for their UI, yet have no reliable test method for verifying the app is working properly.

To make the best use of your time and budget, start with the basics of functionality and usability. Make sure you have compatibility across the vast majority of devices in your market. Enhance with known areas of concern such as performance, reliability, and security. Finally, based on return on investment, fill in more specialized or advanced testing such as model-based testing.

About the Author

Keith Stobie is a Consulting Architect for Xamarin Test Cloud [U4]. Previously he was a Principal Software Development Engineer in Test working as a Knowledge Engineer in the Engineering Excellence group at Microsoft, and Test Architect for Bing Infrastructure, where he planned, designed, and reviewed software architecture and tests. Keith worked in the Protocol Engineering Team on Protocol Quality Assurance Process including model-based testing (MBT) to develop test framework, harnessing, and model patterns. With twenty-five years of distributed systems

testing experience Keith’s interests are in testing methodology, tools technology, and quality process.

External resources referenced in this article:

[U1]	https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks
[U2]	http://oasam.org/en
[U3]	https://www.owasp.org/index.php/IOS_Application_Security_Testing_Cheat_Sheet
[U4]	http://xamarin.com/test-cloud

Stepping Up to Simulation Testing

It's Expensive — and Worth It

by Ryan Neufeld

When you move to the high-rent neighborhood of the testing space.

What is simulation testing, and why would you want to use it?

You can categorize software testing approaches on two primary axes: *scope* and *level*.

The *scope* spectrum stretches from example-based tests, like:

```
assert(result == true)
```

to property-based tests, like:

```
For all x > 0, assert pos?(x) returns true
```

Property-based tests are more expensive, but provide much stronger guarantees about the behavior under test.

The *level* spectrum ranges from white-box tests, which have intimate access to variables and data, to black-box tests, which have knowledge only of externally-visible state. One type of test is not strictly better than the other, but black-box tests are better at validating *consumer-facing* behavior — the behavior your customers are most likely to pass judgment on you for.

Black-box tests are *also* typically more expensive to build and maintain.

Plotting testing approaches along these axes, we can see that Simulation Testing lies on the intersection of property-based and example-based testing. Simulation testing provides *strong guarantees* about *externally-visible*, *client behavior*. It also lies in the high-rent neighborhood of the testing space.

Tests	Example-based	Property-based
White Box	Traditional Unit Tests	Generative Tests
Black Box	Integration Tests	Simulation Testing

OK, that's where simulation testing lies in the dimensions of testing, but what is it exactly?

Simulation testing is a method of software testing in which we simulate inputs to the system we're testing, using *controlled randomness* to increase our likelihood of hitting all the crucial points in the search space of errors while also keeping our tests repeatable.

Why Simulation Testing?

It's a fair question. As I indicated, both axes increase in cost as they edge toward simulation testing. It's not a cheap approach. So what would necessitate this kind of expenditure?

Generally speaking, systems that warrant simulation testing are large-scale production systems generating hundreds of thousands, if not millions, of dollars of revenue. I've seen a number of such systems in finance and e-commerce, but they're bound to exist at the hearts of a number of service providers.

Having worked with a number of clients in the space, my observation is that they tend to worry most about important *flows* through the application or more general properties of the system (like performance or correctness). Simulation testing helps them dig deep in these parts of the system, and insulates them from costly failures. For them, this justifies the cost.

How Does It Work?

Simulation testing accomplishes these things by way of some very clever destructuring of the concerns that make up testing. Rather than author, run, and validate our system in one pass, simulation testing *pulls apart* these concerns into temporally-disconnected phases.

The Phases

1. Modelling — This is generative testing: we don't prescribe the *exact* behavior our tests will perform, but rather what they *could* perform. This is done in simulation testing by creating a *model* — a probabilistic graph of potential behavior, much like a Markov Chain [U1].

2. Test Generation — From the non-deterministic model, a well-defined set of planned actions is generated. This set of actions is then captured and available to run once, or multiple times, if necessary.

3. Sim Invocation — Executing a generated set of actions constitutes "running" a simulation test. During each planned action, a part of the system is interacted with or observed, and the results are stored for later use.

4. Validation — Finally, at some later point in time, the *action log* of a previous simulation is queried and aggregated to detect any failures or abnormalities. As this is a separate process from running the tests themselves, it also becomes possible to add new validations and "backfill" failures from previous points in time.

In conjunction with one another, these phases allow for an incredibly robust and deep validation of a system. Fortunately, you don't have to roll your own. All of these phases are nicely captured in Cognitect's [U2] Simulant [U3] library. Built on top of Clojure [U4] and Datomic [U5], Simulant leverages Clojure's strong concurrency facilities and Datomic's rich data model to provide these features in a clean, simple package.

Simple doesn't necessarily imply *easy*, however. It can be a bit bewildering to string all the pieces together to form a coherent sim. Throughout the remainder of this article, I'll introduce sim-template [U6], a handy template we at Homegrown Labs [U7] have put together. The generated project is written in Clojure, but you should be able to follow along regardless.

Creating a Simulant Project

Before you get started, there are a few pieces of software you'll need installed on your system, namely:

- Leiningen [U8]: A Clojure project management tool sim-template utilized solely for template instantiation.
- Boot [U9]: A newer project management tool for Clojure that drives the generated Simulant suite.
- Docker [U10] (quasi-optional): A containerization/virtualization tool used to run the sample service the generated suite executes against.
- An active Datomic [U11] transactor (optional): The system of record for a Simulant suite. While you can run the suite in-memory without a transactor while you're getting your footing, one is necessary for persistent testing and result collection.

With the above installed, creating a new Simulant suite is as simple as:

```
bash
$ lein new sim-test org.my-org/sample-sim
```

Run this and you're off to the races.

Surveying the Landscape

If you're not intimately familiar with Simulant, then the slew of files in your generated project will likely be bewildering. At a high level, Simulant achieves the flexibility it affords by breaking the traditional notion of a *test* into four disparate pieces: behavior modelling, test planning, sim invocation, and result validation. All of these phases act in concert to create and execute randomized behavior against a target system.

Here's how each of the generated src/ files relate to those phases:

Model: model.clj where potential behavior in the sim is defined.

Test: test.clj where a concrete plan of action is generated from the non-deterministic model.

Sim: sim.clj where a concrete test is executed against an actual system; actions.clj where general, re-usable *action* behavior is captured; and actions/* where specific actions to take against a target system are defined. (actions/sample.clj defines actions against a sample service, but in practice you will define your own namespaces to test real systems.)

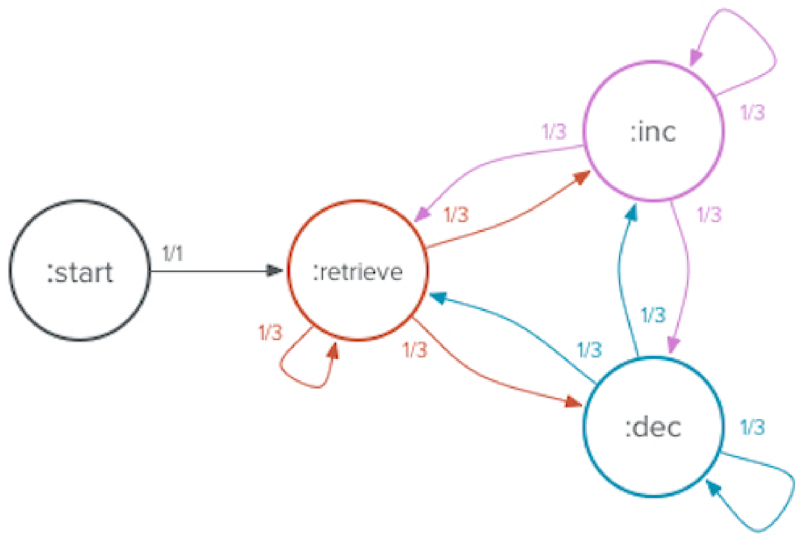
Validation: validations.clj where previously run sims are validated for correctness, etc.

Miscellaneous: repl.clj a namespace for REPL-based interactions with a simulation; db.clj a small namespace for managing database migrations; util.clj utilities for capturing codebase information from the suite and target systems; and resources/schema.edn database schema for simulant's internal Datomic database. Where you'll add new attributes when needed.

With the 10,000-foot view out of the way, let's zoom in on specific phases to see how they work, and how you can start to build your own simulation tests into the project.

Model

As hinted, the *model* phase of a simulation test captures potential behavior of agents acting in a sim. In the generated project, this is defined in generated-project.model/agent-behavior. This data structure represents the *edges* of the following graph, along with a maximum delay per transition. In graph format, this looks a little something like this:



Model Graph

To construct your own models, I would suggest creating a state-transition diagram as above, then transcribing it to an edge-graph when it comes time for implementation. An aside: you're not strictly limited to generating models via *markov models* as above, but it's generally a good starting point for most projects. Since it's data all the way down, you can perform any post processing necessary to adjust behavior for your own domain (often, in test).

Test

The *test* namespace of your generated project has the job of creating a concrete plan of action from your random models and other data generators. This strikes the ideal middle ground between stressful randomized behavior and pre-determined test plans. They're random, and they're repeatable.

In practice, this generally means some housekeeping (setting up agents, domain objects, etc.), followed by generating *action* entities for each state transition your agents will undertake. This is accomplished via the actions-for multi-method, which dispatches based on target state. For the most part, these action entities are static, but if your own actions inject additional random data, this is where you should generate that data. You'll write one actions-for implementation per state in your own model (as well as a corresponding :action.type in resources/schema.edn).

Sim and Actions

The *sim* phase is where the rubber meets the road. In the *sim* namespace, you'll find a number of functions all invoked by run-sim! that organize and initiate running a test plan against a real target system. Most of these functions will remain static for the life of your project (library fodder, I know), but there are two in particular you will interact with more often: setup-actions and setup-system.

setup-actions is simple: this is where you require any action namespace you write (a limitation of Clojure multimethod definitions) — more on that shortly. setup-system is notably empty in the generated project. Alas, it is the place where you should perform any system initialization your suite requires. Think agent accounts, data baselines, etc.

Now, for actions. In the test phase we generated action entities for a number of action types. In the sim phase, you must provide a perform-action implementation for each of those types. For the sample service, these are provided, but for your own actions, you'll need to write action implementations that perform actions and keep records in a similar fashion (this could be its own article, of course).

Validation

Finally, the *validation* phase is where you introspect on action log entries across your sim run to enforce feature and quality invariants. For many types of validations, this is as simple as querying for non-conforming log entries. For more involved validations, you'll find the pattern of querying for raw results, then aggregating and filtering to detect failures to be the best approach. If you're not intimately familiar with Datomic's Datalog, I'd suggest working through Learn Datalog Today! [U12] for developing a good body of knowledge.

Putting It All Together

To put all of these phases together, you have two approaches: In experimentation and development phases, I'd suggest evaluating parts or all of the *repl* namespace in sequence. For long-term installations, I'd suggest using the commands provided by boot-sim [U13] to run through model creation, test creation, sim invocation, and validation, in turn.

I hope sim-template provides you with fertile grounds for building your own simulation tests. Of course, this article barely scratches the surface of simulation testing. In subsequent articles of this series, we'll dig deeper into the what, where, and why of simulation testing. In the mean time, if there's anything specific you're curious about or are having trouble with, feel free to drop a line to info@homegrown.io and I'd be happy to help.

About the Author

Ryan Neufeld is a software developer, author and speaker specializing in broad-scale distributed systems and advanced testing techniques.

While Ryan loves designing and deploying large distributed systems as much as any person, there's one thing he likes doing more: breaking them. To that end Ryan founded Homegrown Labs, a consulting company focused on using simulation testing to help companies deliver better software, faster. You can learn more about simulation testing or get help with your own projects at homegrown.io.

External resources referenced in this article:

[U1]	https://en.wikipedia.org/wiki/Markov_chain
[U2]	http://cognitect.com/
[U3]	https://github.com/Datomic/simulant/
[U4]	http://www.clojure.org
[U5]	http://www.datomic.com
[U6]	https://github.com/homegrownlabs/sim-template
[U7]	http://homegrown.io
[U8]	http://leiningen.org/#install
[U9]	https://github.com/boot-clj/boot#install
[U10]	https://docs.docker.com/installation/
[U11]	http://www.datomic.com/
[U12]	http://www.learndatalogtoday.org/
[U13]	https://github.com/homegrownlabs/boot-sim

Further Reading on Testing

From the PragPub Archives

by PragPub Staff

But wait, there's more: here are nearly two dozen articles we've published on testing.

We've been publishing articles on testing since 2009. Here are links to nearly two dozen articles from past issues:

General Testing Articles

- Jeff Langr presents a clear explanation of what Test Driven Development is all about, and what its potential benefits and risks are. [U1]
- Jeff Langr and Tim Ottinger present eight techniques to improve your tests. [U2]
- Kent Beck really doesn't care if you "do" TDD — so long as you know the consequences. [U3]
- Sam Ruby tells how the system that keeps the Rails book on track also aids in the development of Rails itself. [U4]
- Zach Dennis explains when to Mock and when not to Mock. [U5]
- If you've ever had the feeling that you're refactoring without a safety net, you need to read what Michael Rogers has to say about mutation testing. [U6]
- Chris McMahon explains why it's time to take a fresh look at Software Quality Assurance, a discipline that deserves much more respect that it usually gets. [U7]
- Paul Butcher explores how to improve your tests with custom expectations, and does so using parallel implementations in Ruby and Scala. [U8]
- Noel Rappin cautions that, while Mock objects can be very useful in test-driven design, they need to be used wisely. [U9]
- Jonathan Rasmusson looks at automated testing and the automated tester. [U10]
- Adam Goucher points out that test automation, like all of testing, is an inherently heuristic activity. [U11]
- Rachel Davies offers advice on how to deal with managers who don't see the value of testing and code maintenance. [U12]

iOS Testing

- Apple's new language features a nifty way to test out ideas — but Ron Jeffries asks whether undermines test-driven programming practices. [U13]

- Jonathan Rasmusson explains how iOS developers maintain high quality without unit tests. [U14]
- Eric Smith reveals that, while it may require some DIY, TDD on iPhone is totally a thing. [U15]
- Ian Dees shows how to drive an iPhone GUI from a Ruby test script. [U16]
- Rob Holland introduces iCuke, which makes iOS testing even easier. [U17]

Arduino Testing

- Ian Dees brings the testing power of the Ruby-based Cucumber testing library to the Arduino. [U18]

Testing for Cloud and Web Services

- Adam Goucher explains that the three big differences cloud computing brings with it are really just modern twists on old practices. [U19]
- Noel Rappin details the key ideas behind testing for web services. [U20]

Note: there are two kinds of links here. For complicated reasons, links to earlier articles are to the individual article, while links to later articles are actually download links to the entire issue the article appeared in.

External resources referenced in this article:

[U1] https://pragprog.com/magazines/2011-11/testdriven-development
[U2] https://pragprog.com/magazines/2011-04/test-abstraction
[U3] https://s3-us-west-2.amazonaws.com/pp-14-09/September-14.pdf
[U4] https://pragprog.com/magazines/2013-06/keeping-rails-on-the-rails
[U5] https://pragprog.com/magazines/2011-06/practical-mock-advice
[U6] https://s3-us-west-2.amazonaws.com/pp-15-07/PP-15-07.pdf
[U7] https://pragprog.com/magazines/2011-01/rediscovering-qa
[U8] https://pragprog.com/magazines/2010-08/great-expectations
[U9] https://pragprog.com/magazines/2010-05/the-virtues-of-mockery
[U10] https://s3-us-west-2.amazonaws.com/pp-13-09/pragpub-2013-09.pdf
[U11] https://pragprog.com/magazines/2011-10/lightsabers-time-machines--other-automation-heuristics
[U12] https://s3-us-west-2.amazonaws.com/pp-15-07/PP-15-07.pdf
[U13] https://s3-us-west-2.amazonaws.com/pp-14-07/July-14.pdf
[U14] https://pragprog.com/magazines/2012-10/its-not-about-the-unit-tests
[U15] https://pragprog.com/magazines/2010-07/tdd-on-iphone-diy
[U16] https://pragprog.com/magazines/2009-08/iphone-meet-cucumber
[U17] https://pragprog.com/magazines/2010-07/bdd-on-iphone-icuke
[U18] https://pragprog.com/magazines/2011-04/testing-arduino-code
[U19] https://pragprog.com/magazines/2011-03/testing-for-the-cloud
[U20] https://pragprog.com/magazines/2011-03/testing-for-web-services

Making Computer Science Insanely Great

Part 3: The Unknown Unknowns

by James Bonang

Jim teaches kids to code in the most unlikely of circumstances and encounters the Unknown Unknowns.

This is Part 3 of a 3-part story of one developer's experience in teaching kids to code.

Some things you know you don't know. Then there are the Unknown Unknowns.

My dream of teaching computer science at my daughter's high school was abruptly postponed when the U.S. Navy recalled me to active duty and deployed me overseas. However, after an unlikely series of events, I found myself selected as a volunteer assigned to teach a thirty-minute lesson in computer science during a Science, Technology, Engineering, and Math (STEM) event held at a small Department of Defense Education Activity (DoDEA) high school serving the children of overseas U.S. service personnel. I was to introduce my students to programming, give them a useful skill and a sense of accomplishment, and excite their interest in computer science and technology in general. How hard could that be?

Hard. Fortunately, I obtained expert help, not only from the exceptional teachers at the high school but also from authors of the *PragPub* "Teaching Kids to Code" [U1] series of articles. I developed a lesson plan where my students would create Vokis [U2], customized, animated speaking characters, embed them in their own web pages they would write in HTML, and have the characters debate the pros and cons of school uniforms (recounted in Part 1). But to present technology, programming, and computer science in a way that would make it exciting and interesting to high school students I consulted a master — the master. I employed many of the techniques Steve Jobs used in his legendary keynotes into my presentation (recounted in Part 2). Now, I had a solid draft of the presentation completed and felt almost sanguine even though time was running short. I was just about there. But that feeling was misplaced.

Our personal computers have multiple typefaces and proportionally spaced fonts in large part because Jobs happened to audit a class in calligraphy at Reed College in the early 1970s. There, he learned "what makes great typography great." (Isaacson, Walter. *Steve Jobs*, New York, Simon & Schuster, 2011, ISBN 978-1451648539, pp. 40-41) [U3]. He would continue to interweave technology and great design into all his endeavors, welding together C.P. Snow's two cultures [U4]. His presentations were no exception; looking at them, I slowly began to discern that I had missed something. Something vital. Something really important.

The Red Pill

Like Neo in The Matrix [U5], I realized something was wrong with the world. My world. More specifically, my slides. I wasn't quite sure what it was, but I could feel it, like a bullet point in my mind, driving me mad. I watched the

Steve Jobs 2007 MacWorld keynote and studied his slides, then looked at mine. Mine looked … different. A lot different. Why are Steve's slides so appealing? Indeed, what makes a slide clear and aesthetically pleasing? I had to know what it is. That led me to, not Morpheus, but *The Non-Designer's Presentation Book: Principles for Effective Presentation Design* [U6] (Peach Pit Press, 2010, ISBN 978-0321656216, by Robin Williams). Prior to reading this, I thought I had done at least an adequate job of slide design. I swallowed the red pill and read in a state of continuous astonishment. My slides began to appear strange, unclear. I realized Default Layouts and Themes had been pulled over my eyes to blind me from the truth. Now, I could see my slides as they actually were — awful. I had violated nearly every principle of slide design Ms. Williams covered. So, this is your last chance; after this, there is no turning back. Read on — take the red pill — if you dare.

The better your presentations, the more successful you will be in teaching, and in your career. A great presentation needs great slides and crafting great slides takes skill. We'll first look at what needs to go on your slides and what should instead go into your notes and your handouts; a presentation is more than just slides. Next, we'll look at conceptual design, specifically how to make your slides convey your message as clearly as possible. Then we'll cover techniques for visual design; you'll learn how to make your slide look professional and appealing. Presentations require preparation, and I learned the essentials the hard way. Instead, you'll learn those preparation steps right here. Lastly, we'll see how my lesson went, what worked and what didn't. Was it effective?

Fate had not finished with me though; after the presentation one more unexpected event awaited me on my deployment. One where everything I learned about teaching and presenting would come in to play.

Remember, all I'm offering is the truth, nothing more.

Hand Out a Handout

While I violated nearly every one of Ms. William's design principles, I didn't violate all of them. I got the handouts right; well, at least I had handouts. When you create a presentation, there are actually three deliverables: the slides, the printed handouts, and the speaker's notes. You'll typically create the slides and the speaker's notes in presentation software such as Keynote, PowerPoint, or Impress, with the speaker's notes going into the notes section of each slide. While you could use, say, Keynote to create your handout, the handout should not be the presentation. Handouts and presentations serve vastly different purposes. Handouts contain reference information such as web addresses, contact information, and additional reading lists, that audience members should take with them after the presentation. Handouts also include detailed information such as charts, graphs, equations, tables, and step-by-step instructions, which might be hard to see up on the screen. The slide presentation, in contrast, is an accompaniment to your talk. And that is where I went awry: for my handouts, I simply printed out my slides. Indeed, I created my slides to serve as both presentation and handout. PowerPoint even has a button to print your slides as handouts. It seemed so convenient and efficient. *Big mistake*.

A spoon is not a screwdriver; a presentation is not a handout. Why are things so obvious only in retrospect? Some of my slides included just a few keywords to summarize talking points, no detailed information, no references: adequate for the screen but a waste of paper in my handout. Worse, I included webpage URLs on some slides that belonged nowhere but in the handouts.

In some cases, though, you'll place material in both your presentation and in your handout. When teaching a lesson that includes a lab exercise, place step-by-step instructions both in your presentation and in your handouts. The instructions may be too detailed for the screen, so having them in your handout is essential. But having the instructions at least available in the presentation allowed me to discuss some particular point in the instructions in the event several students had trouble at the same point during the lab. I'd point to a spot in the instructions and the students would then look down at their printout and listen. This kept the lab session on track.

If your presentation doesn't include any detailed slides, and your handout needs just references and contact information, you might distribute it by email afterwards (but let your audience know in advance). So, like I did — provide handouts. Just don't hand out your presentation, at least not all of it. There is no spoon!

Note: Use Speaker's Notes

I may have passed with a C- on the handouts, but not so on the speaker's notes: Zero. Point. Zero. I had none. It gets worse. I actually did have speaker's notes of a sort — on my slides, my overly verbose, cluttered, complete-sentences-everywhere slides. *Don't do this*. Write your speaker's notes, the text of what you'll say, in the notes section of your slides. You alone need these; the audience does not. They need you — the speaker. The clear, concise text on your slide serves as a mnemonic; you have to memorize roughly what you'll say on each topic but your slide will cue you to recall it. You won't be perceived as reading your slides; rather, you'll appear as the knowledgeable authority you are. With the speaker's notes tucked away out of sight, your presentation appears streamlined and visually appealing — just like Steve's.

I'll Be Back

Ms. Williams divides presentation design into two broad categories: conceptual and visual. Her approach to conceptual design left me astounded; she pursues clarity with the relentlessness of The Terminator [U7]. "Can the clutter. Get to the point. Simplify. Be specific." My slides did not communicate clearly and she pointed out why: you need to leave things out. In particular, edit the text. Get rid of the extra words. Like the Terminator, she has plenty of weapons. First, avoid lengthy complete sentences; you'll be speaking in complete sentences so the audience needs only the main points on the slide. To eliminate superfluous words, use active voice; in general, active voice takes fewer words. For example:

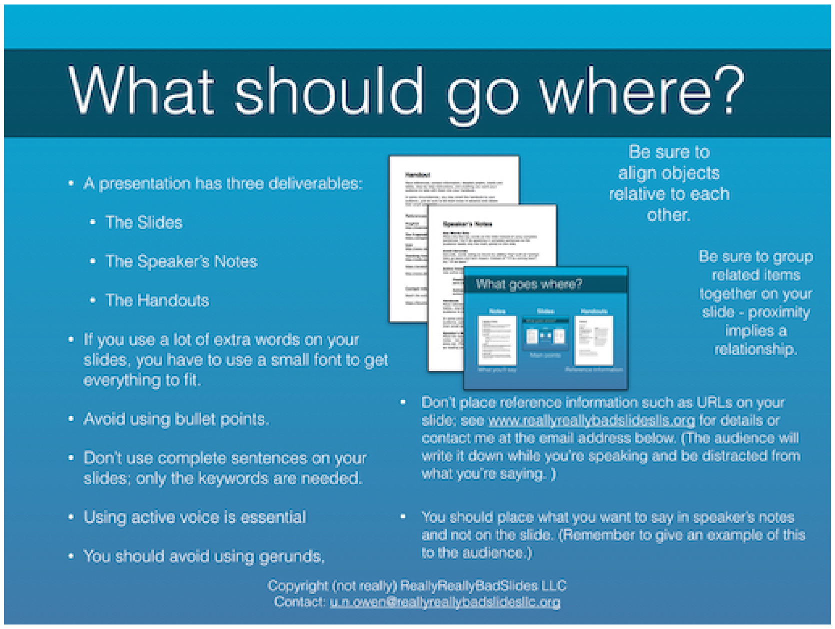
- Passive: If something goes wrong, one can push the panic button.
- Active: When something goes wrong, push the panic button.

Active voice eliminated two words (*one* and *can*). Robin also terminates gerunds — words acting as nouns by adding "ing" to them. "I'll be coming back" just doesn't have the impact of "I'll be back," and it's longer. Incidentally, unlike the Terminator, Robin does not use bullets; she has bullet points, but eliminates the dot. You'll not only save ammunition, you'll enhance clarity. With a clear, streamlined, exciting presentation, you'll be a popular presenter — you'll be back.

Visual Design

While conceptual design centers on clarity, Visual Design encompasses four principles: Contrast, Repetition, Alignment, and Proximity. When I applied these in retrospect to my amateurish slides they were transformed — a vague clutter became professional and clear. More like … Steve's. Contrast simply means making items that are different look different. To achieve contrast, use a different typeface style such as bold or italic, or highlight the text. You might also change the color of the background or the text to increase contrast. Repetition provides unity to your presentation; it gives your presentation a consistent look. This could be as simple as using the same font across all slides. Align the edge of every item on your slide with something else; don't place anything arbitrarily. Misaligned items look unprofessional. Physical proximity implies a relationship; group related items together on your slide.

Like the Matrix, no one can explain it to you — you have to see it for yourself. Take a look at the two example slides below. The first slide shows what not to do. The second slide contains the same information as the first, after applying a few of Robin's techniques.



What should go where?

- A presentation has three deliverables:
  - The Slides
  - The Speaker's Notes
  - The Handouts
- If you use a lot of extra words on your slides, you have to use a small font to get everything to fit.
- Avoid using bullet points.
- Don't use complete sentences on your slides; only the keywords are needed.
- Using active voice is essential
- You should avoid using gerunds,

Be sure to align objects relative to each other.

Be sure to group related items together on your slide - proximity implies a relationship.

- Don't place reference information such as URLs on your slide; see www.reallyreallybadslideslls.org for details or contact me at the email address below. (The audience will write it down while you're speaking and be distracted from what you're saying.)
- You should place what you want to say in speaker's notes and not on the slide. (Remember to give an example of this to the audience.)

Copyright (not really) ReallyReallyBadSlides LLC
Contact: u.n.owen@reallyreallybadslidesllc.org

Figure 1: *What not to do*.

Now, after I apply the four visual design principles, terminate complete sentences, gerunds, and passive voice, move references and contact information to the handouts, and place information needed only by the speaker in the speaker's notes, how does it look?

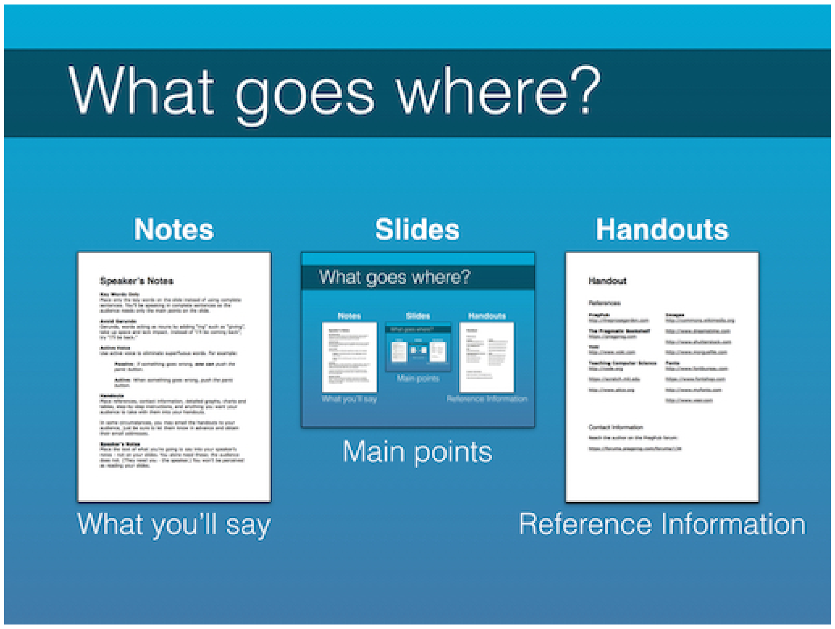


Figure 2: *The slide after applying a few of Robin's techniques*.

I thought the same thing: "Whoa" (pronounced in the manner of Keanu Reeves).

Of course, I've only scratched the surface of visual design and Robin's many other techniques; a treasure trove of ideas awaits you in Robin's book, *The Non-Designer's Presentation Book: Principles for Effective Presentation Design*.

The table below summarizes Robin's principles I found essential when teaching.

Technique	Description
Hand out a Handout	Place references, contact information, detailed diagrams, step-by-step instructions, and anything your audience should take with them in your handouts.
Use Speaker's Notes	Place what you'll say in your speaker's notes, leaving only essential keywords summarizing the main points on your slides
Pursue Clarity	Place only main points on your slides; move all else into speaker's notes or handouts. Use active voice. Don't use bullet points.
Apply the Four Visual Design Principles	Use contrast, repetition, alignment, and proximity.

Table 1: *Robin Williams' Presentation Principles*

This Looks Like a Job For …

If only I had had time to read Robin's book before I taught my class. But just as in software development, I had to ship. I put some last minute touches on the presentation, including a few publicity stills from the popular 2009 science fiction movie Avatar [U8] to provide a familiar and exciting reference when introducing Voki avatars. In the movie, humans control ersatz alien bodies, avatars, to allow them to interact with the inhabitants of an alien world.

Controlling these avatars is like having a superpower; I wanted to subtly convey the notion that programming your own avatar and displaying it on a web page is like having a superpower. As my daughter advised me (in Part 1), every high school student wants a superpower — and wants to show it off.

Images

While Avatar's writer/director/producer James Cameron probably won't complain about the free advertising, including movie still photos, even publicity stills, in a professional presentation is unwise. Fair use of publicity photos in such cases is unclear so I never posted the presentation with these pictures in it on the school website. In any case, avoid using such photographs in professional presentations. Instead, obtain images from sites such as Wikimedia Commons [U9], Shutterstock [U10], and Dreamstime [U11]. Nevertheless, movies like Avatar showcase applications of computer science such as motion-capture technology and raise student interest; I included a still of the motion capture system in use alongside the resulting computer animation. Use images — just be careful where you get them.

The Cavalry Still Comes Over the Hill

Adding exciting images from Avatar required little time, but I still worried whether I could complete everything before showtime; I had already run out of time to revise my slides in accordance with Robin's guidance. Things looked bleak. At this point in a movie (a Western anyway), the cavalry arrives. Now, in my all-too-real situation, … the cavalry arrived — literally, in the form of another volunteer, a U.S. Army captain (sans horse) and fellow software engineer, who developed several Voki avatars for me that visually resembled famous early programmers including Grace Hopper [U12], who programmed the Harvard Mark I, and Elizabeth Holberton [U13], one of the original six ENIAC programmers. He scripted the Vokis to summarize the careers of both programmers in first person. I included the two Vokis at the end of the presentation along with pictures of early computers in a series of slides themed "Programming Pioneers." We added two interesting role models, conveyed a bit of computer history, and provided examples of how the students could use Vokis for in-class presentations. These slides repeated the answer to the question "Why should I care?" Avatars can be used to spice up class presentations and you'll "earn better grades." Remember, Steve Jobs repeated the answer to the question "Why should I care?" several times in his presentations. It's best not to rely on the cavalry showing up to do that for you though.

Getting Ready

When you design your programming lesson, you may need to work around severe equipment and procedural limitations. In my case, school IT staff imposed strict (but necessary) security measures. As a volunteer, I couldn't plug in a USB drive, install software, or even obtain an account on the school's computers. Any software I used in my lesson had to be accessible over the web or already present on the school's computers. I planned for this: students would build their avatars using the Voki website and write HTML in Notepad. Fortunately, the staff would allow me to plug my new Macintosh Pro laptop into the audio video system and I could use Keynote to display my presentation on a large, overhead screen. I also discovered earlier that I needed a Mini DisplayPort to VGA Adapter to connect the Mac and brought one. Alas, even though I began setting up an hour before class, I couldn't get the audio output to the big speakers adjacent to the screen working, or a pair of speakers I brought with me, and had to compensate by setting the Mac's internal speaker to maximum volume. It turned out to be a Mac OS X configuration setting that I wasn't familiar with (the volume for the external speakers was set to zero). Test everything in advance: a little more preparation time and testing would have eliminated this embarrassing glitch.

Worse, I also placed my laptop with the screen facing the audience. Ideally, you want it facing away. I turned away from the students occasionally to glance at the screen to view the next slide displayed in Keynote's presentation view. If the screen faced away from the audience I never would have. Your audience needs your undivided attention as much as you need theirs.

Nothing turns students off like dull, tedious set-up work; I wanted the students to have fun and so had everything ready in advance such as the accounts needed to access the Voki website. A teacher let me type in an HTML file as a contingency against a student being a slow or inaccurate typist (none were). I also positioned my handouts next to the particular computers I wanted the students to use; the computer lab was lavishly equipped with Windows laptops and I needed the students near the projection screen (and my Mac's internal speaker) and seated so I could easily observe their work and diagnose trouble.

Showtime

I arrived at the school right after duty still in my U.S. Navy uniform — apropos as the class would create Vokis to debate whether the school should require uniforms or not (currently the school does not). Shortly after I completed setting up, the students arrived. My six students ranged from 9th to 12th grade and were all girls — the school planned this particular STEM event specifically for girls. Six doesn't seem like a large turnout, but represents a good fraction of the small school's student body. Smiles and interested glances met the photo from the movie Avatar on my title slide: "Create your own Avatar — with Voki." (It worked!)

Showtime.

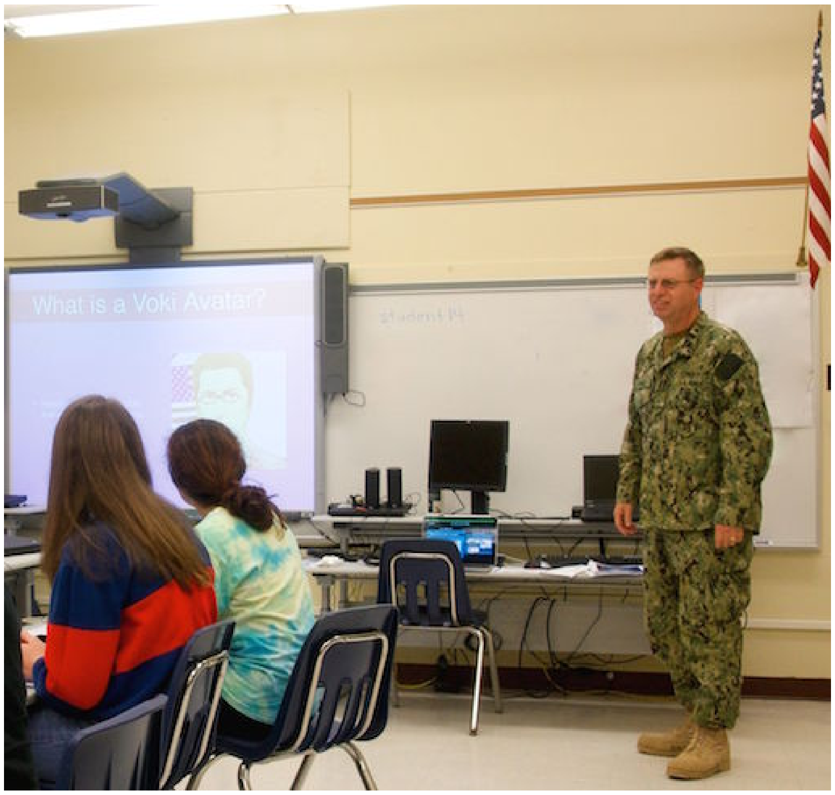


Figure 3: *Jim's Voki avatar explains Voki avatars.*

A Voki avatar of myself (well, perhaps a bit better-looking version of myself) introduced the assignment. The students paid very close attention — they'd never seen a Voki and weren't expecting this. I (the real me, that is) then gave a one-slide overview of how to build a Voki and demonstrated what their web page would look like — with two working Vokis debating the issue of school uniforms. I had their attention, though my "Why should I care?" answer — get better grades missed the mark. It wasn't convincing enough. Thankfully Lyndsey Scott [U14], my role model, salvaged the situation with a much stronger answer. The surprised looks of the students clearly revealed they weren't expecting a supermodel/programmer, especially one who viewed programming as fun and on equal footing with a glamorous modeling job. (Memorable moment achieved too!)

The poor sound quality of my overly long (six minute) "How to Create a Voki" video didn't help. Fortunately, the students were so intrigued with building a computer alter-ego of themselves, they jumped right onto the website, ignored the video after the first minute, and started creating their Vokis. Most students followed the step-by-step instructions in the handout, a few relied on intuition. Thanks to the computer lab layout, I could easily move behind them and helped when they had questions. I displayed specific steps on the main screen when more than one student was stuck in the same place. Everyone succeeded in building their first Voki and had it speaking a few sentences in favor of school uniforms. (Smiles all around at this point.) Visible results were achieved quickly, but was it quick enough? We only had thirty minutes and the hard part was still to come. A "well done" from the instructor helped build confidence for the next step.

Another Voki avatar introduced programming and the Hypertext Markup Language (HTML). I explained HTML in more detail on three slides and provided step-by-step instructions on creating an HTML file and displaying it in a browser. Most students struggled despite my handout. One saved her file under a new name and was mystified when the older version was rendered in the browser. (She needed to update the pathname in the browser's address box.) All of them were reasonably good typists (which shouldn't have surprised me yet it did; even when I was in college many students had to hunt and peck at the keyboard). With a little help, they all successfully had the skeleton HMTL file displayed in their browser. Another visible result accompanied by visible happiness.

For the next step, they would display their Voki avatar in their web page. This involved inserting code to run their Voki's Shockwave video from the Voki website into their HTML file. With minutes left, and a bit of help, they all displayed their first Vokis in their own web pages. One girl exclaimed triumphantly "I did it! I didn't fail! I did it!" — which produced another visible result: a very pleased instructor.

I concluded by showing the students how to spice up presentations using Vokis with the two "Pioneers of Programming" slides and pointed out that they already knew a few fellow programmers. This raised confused looks until I followed up with "they're sitting next to you." Finally, I pointed out they could finish creating the second Voki and adding it to their web page on their own. The supervising teacher called time and the students proceeded to the evening's next event. Mission accomplished.

A PDF file containing the presentation, with all its faults (but not its videos), may be downloaded here CreateYourOwnAvatar [U15].

After Action Report

So, how effective was the class? Did it engender student interest in programming and computer science? Of the six students, one left early to attend soccer practice. She had indicated when signing up for the STEM event that her coach might not let her skip. Sports still trumps academics, computer science included. At the end of the class though, three of the remaining five students took their handouts with them, suggesting interest and intent to complete the second part of the exercise — and who knows what may come of that? A teacher who stepped into the classroom to observe asked if he could take one of the handouts too.

Clearly, Robin's slide design techniques and more practice would have improved the presentation immensely. I should have eliminated the audio glitches in advance and used a shorter (under three minutes) how-to video. The lesson plan worked well, thanks to the *PragPub* authors' advice: visible results quickly, a constrained environment so that getting started is easy, a lesson designed for play (the students really enjoyed designing their Vokis — pure fun) with fundamentals coming later, and an interesting topic with room to grow.

The Unknown Unknowns

I learned alot too. When I started creating my lesson, I didn't know how to give a good presentation. I actually thought I knew how. I thought it was trivial — I didn't know I didn't know. My awakening after taking the red pill and facing the Unknown Unknowns shook me just as it did Neo; I'll never see a presentation the same way again.

I plan on teaching a STEM course at my daughter's high school when I return home and will eventually help introduce a full-semester computer science course. I know how to do that now. More importantly, you now also know.

You know too how to craft your lesson using the techniques of the *PragPub* authors and how to choreograph your presentation like Steve Jobs, techniques you can use for teaching and throughout your career as a software professional. You know how to design your presentation as well — not just your slides, but your presentation. Like Neo, you've left the dreamworld of Bullet Points and Default Layouts and vanquished a few of those Unknown Unknowns.

Just One More Thing

I remained on deployment for some time after that STEM night. Just before returning to a base in the U.S., events again took an unexpected turn. Out of the blue, a senior officer directed me to prepare a briefing — for an Admiral. And, he informed me, an Air Force general would also be attending, along with all the senior staff. Being rather low on the totem pole, I'd never given a briefing to such an audience.

My presentation went very smoothly.

We've created a special issue of PragPub containing all of our articles on teaching kids to code that are referenced in this article. You can download the issue in any or all of our three formats here: pdf [U16], *mobi* [U17], *epub* [U18].

About the Author

Jim Bonang has spent over twenty-five years developing software for a wide range of systems, including satellite terminals, document management systems, medical devices, and many others. He's also worked on several compilers.

External resources referenced in this article:

[U1]	http://theprosegarden.com
[U2]	http://www.voki.com
[U3]	http://www.amazon.com/Steve-Jobs-Walter-Isaacson/dp/1451648537/
[U4]	https://en.wikipedia.org/wiki/The_Two_Cultures
[U5]	https://en.wikipedia.org/?title=The_Matrix
[U6]	http://www.amazon.com/Non-Designers-Presentation-Book-Robin-Williams/dp/0321656210
[U7]	https://en.wikipedia.org/wiki/The_Terminator
[U8]	https://en.wikipedia.org/wiki/Avatar_(2009_film)
[U9]	https://commons.wikimedia.org/wiki/Main_Page morgueFile
[U10]	http://www.shutterstock.com
[U11]	http://www.dreamstime.com
[U12]	https://en.wikipedia.org/wiki/Grace_Hopper
[U13]	http://en.wikipedia.org/wiki/Betty_Holberton
[U14]	http://www.forbes.com/sites/vannale/2014/08/18/meet-lyndsey-scott-model-actress-and-app-developer/
[U15]	https://dl.dropboxusercontent.com/u/68222076/CreateYourOwnAvatar.pdf
[U16]	https://s3-us-west-2.amazonaws.com/pp-x1/Special.pdf

[U17] https://s3-us-west-2.amazonaws.com/pp-x1/Special.mobi

[U18] https://s3-us-west-2.amazonaws.com/pp-x1/Special.epub

Functional Snippets

Map for Optionals

by Chris Eidhof, Wouter Swierstra, and Florian Kugler

Here's the latest installment in our series designed to help you get a grip on functional programming in Swift.

Swift opens up a whole new world of programming. To quote Swift's creator, Chris Lattner: " 'Objective-C without the C' implies something subtractive, but Swift dramatically expands the design space through the introduction of generics and functional programming concepts."

In this series, we use small snippets of Swift code to explore this new world of programming, as well as demonstrate how Swift supports the functional paradigm.

We've already talked about the map function for arrays in a previous snippet. In case you're not familiar with it, map transforms an array into a new array with the same size by applying a transform function to each element. For example:

```
SELECT ALL
let urls: [NSURL] = [ /* a bunch of image URLs */ ]
let images: [NSImage?] = urls.map { NSImage(contentsOfURL: $0) }
```

However, map can not only be defined for arrays. On a more abstract level, map simply unwraps values from a container type, applies a transform, and wraps them again. For example, we can also define it for optionals (it's already in the standard library, but you could easily write it yourself):

```
SELECT ALL
func map<A, B>(x: A?, f: A -> B) -> B? {
    if let x1 = x {
        return f(x1)
    }
    return nil
}
```

Let's say we wanted to transform an optional URL into an optional NSImage, similar to the above array example. One approach would be to use Swift's optional binding:

```
SELECT ALL
let url: NSURL? = NSURL(string: "image.jpg")
var image: NSImage?
if let url1 = url {
    image = NSImage(contentsOfURL:url1)
}
```

By using map, we can solve this in a much more succinct way:

```
SELECT ALL
let url: NSURL? = NSURL(string: "image.jpg")
let image = map(url) { NSImage(contentsOfURL: $0) }
```

map can be defined on many types, including dictionaries, tuples, functions, and your own types.

About the Authors

Along with Daniel Eggert, Chris Eidhof and Florian Kugler created objc.io[U1], a periodical about best practices and advanced techniques for iOS and OS X development. Eidhof, Kugler, and Wouter Swierstra also wrote the book *Functional Programming in Swift*[U2], in which they explain the concepts behind functional programming and how Swift makes it easy to leverage them in a pragmatic way, in order to write clearer and more expressive code.

External resources referenced in this article:

[U1] http://www.objc.io/

[U2] http://www.objc.io/books/

Antonio on Books

Thirty New Books

by Antonio Cangiano

Antonio looks at all the new tech books of note.

Antonio Cangiano is the author of the excellent Technical Blogging [U1] and you can subscribe to his reports on new books in technology and other fields here [U2].

Across North American college campuses, MacBooks absolutely dominate. When it comes to smartphones however, Apple's victory over their competitors is not as clear-cut. In fact, you'll find a good split between Android and iPhone devices. Zoom out of North America, however, and you'll quickly discover that the world at large mostly uses Android devices (around 80% of smartphones, depending on which figures you are looking at).

So if you want to develop applications that appeal to a more geographically diverse audience, Android is certainly the primary target. And if you decide to develop apps for the North American market, it would be wise to have both an iOS and an Android version, as has become pretty much standard for startups.

Either way, the message is that you shouldn't ignore Android. Thanks to this month's pick, you don't have to. The second edition of *Android Programming: The Big Nerd Ranch Guide*, provides an updated guide to the latest best practices and tools (e.g., Android Studio), brought to you by some of the best teachers in the industry.

Our Staff Pick: *Android Programming: The Big Nerd Ranch Guide (2nd Edition)* [U3] • By *Bill Phillips, Chris Stewart, Brian Hardy, Kristin Marsicano* • ISBN: *0134171454* • Publisher: *Big Nerd Ranch Guides* • Publication date: *August 3, 2015* • Binding: *Paperback* • Estimated price: *$31.19*

Applied Minds: How Engineers Think [U4] • By *Guruprasad Madhavan* • ISBN: *039323987X* • Publisher: *W. W. Norton & Company* • Publication date: *August 3, 2015* • Binding: *Hardcover* • Estimated price: *$13.70*

Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka [U5] • By *Vaughn Vernon* • ISBN: *0133846830* • Publisher: *Addison-Wesley Professional* • Publication date: *August 17, 2015* • Binding: *Hardcover* • Estimated price: *$29.97*

Exploring Robotics with ROBOTIS Systems [U6] • By *Chi N. Thai* • ISBN: *3319204173* • Publisher: *Springer* • Publication date: *August 23, 2015* • Binding: *Hardcover* • Estimated price: *$45.52*

Learning Processing, Second Edition: A Beginner's Guide to Programming Images, Animation, and Interaction [U7] • By *Daniel Shiffman* • ISBN: *0123944430* • Publisher: *Morgan Kaufmann* • Publication date: *August 20, 2015* • Binding: *Paperback* • Estimated price: *$40.43*

Fluent Python [U8] • By *Luciano Ramalho* • ISBN: *1491946008* • Publisher: *O'Reilly Media* • Publication date: *August 20, 2015* • Binding: *Paperback* • Estimated price: *$37.49*

Machine Learning with R - Second Edition [U9] • By *Brett Lantz* • ISBN: *1784393908* • Publisher: *Packt Publishing - ebooks Account* • Publication date: *August 3, 2015* • Binding: *Paperback* • Estimated price: *$54.99*

Predictive Analytics with Microsoft Azure Machine Learning 2nd Edition [U10] • By *Roger Barga, Valentine Fontama, Wee Hyong Tok* • ISBN: *1484212010* • Publisher: *Apress* • Publication date: *August 19, 2015* • Binding: *Paperback* • Estimated price: *$44.79*

You Had Me at Hello World: Mentoring Sessions with Industry Leaders at Microsoft, Facebook, Google, Amazon, Zynga and more! [U11] • By *Dona Sarkar* • ISBN: *0996731113* • Publisher: *Dona Sarkar* • Publication date: *August 22, 2015* • Binding: *Paperback* • Estimated price: *$9.99*

Visual Basic 2015 Unleashed [U12] • By *Alessandro Del Sole* • ISBN: *067233450X* • Publisher: *Sams Publishing* • Publication date: *August 3, 2015* • Binding: *Paperback* • Estimated price: *$34.45*

Data Algorithms: Recipes for Scaling Up with Hadoop and Spark [U13] • By *Mahmoud Parsian* • ISBN: *1491906189* • Publisher: *O'Reilly Media* • Publication date: *August 1, 2015* • Binding: *Paperback* • Estimated price: *$51.96*

Murach's Beginning Java with Eclipse [U14] • By *Joel Murach, Michael Urban* • ISBN: *1890774898* • Publisher: *Mike Murach & Associates* • Publication date: *August 24, 2015* • Binding: *Paperback* • Estimated price: *$41.49*

Creating a Data-Driven Organization [U15] • By *Carl Anderson* • ISBN: *1491916915* • Publisher: *O'Reilly Media* • Publication date: *August 14, 2015* • Binding: *Paperback* • Estimated price: *$20.05*

More Coding in Delphi [U16] • By *Nick Hodges* • ISBN: *194126610X* • Publisher: *Nepeta Enterprises* • Publication date: *August 17, 2015* • Binding: *Paperback* • Estimated price: *$39.99*

AngularJS, JavaScript, and jQuery All in One, Sams Teach Yourself [U17] • By *Brad Dayley, Brendan Dayley* • ISBN: *0672337428* • Publisher: *Sams Publishing* • Publication date: *August 15, 2015* • Binding: *Paperback* • Estimated price: *$25.44*

Spark Cookbook [U18] • By *Rishi Yadav* • ISBN: *1783987065* • Publisher: *Packt Publishing - ebooks Account* • Publication date: *August 3, 2015* • Binding: *Paperback* • Estimated price: *$40.49*

Android Application Development All-in-One For Dummies [U19] • By *Barry Burd* • ISBN: *1118973801* • Publisher: *For Dummies* • Publication date: *August 3, 2015* • Binding: *Paperback* • Estimated price: *$22.40*

On Computing: The Fourth Great Scientific Domain [U20] • By *Paul S. Rosenbloom* • ISBN: *0262528282* • Publisher: *The MIT Press* • Publication date: *August 21, 2015* • Binding: *Paperback* • Estimated price: *$18.90*

Advanced Malware Analysis [U21] • By *Christopher Elisan* • ISBN: *0071819746* • Publisher: *McGraw-Hill Education* • Publication date: *August 13, 2015* • Binding: *Paperback* • Estimated price: *$32.79*

How Software Works: The Magic Behind Encryption, CGI, Search Engines, and Other Everyday Technologies [U22] • By *V. Anton Spraul* • ISBN: *1593276664* •

Publisher: *No Starch Press* • Publication date: *August 21, 2015* • Binding: *Paperback* • Estimated price: *$18.52*

Android Application Development in 24 Hours, Sams Teach Yourself (4th Edition) [U23] • By *Carmen Delessio, Lauren Darcey, Shane Conder* • ISBN: *0672337398* • Publisher: *Sams Publishing* • Publication date: *August 8, 2015* • Binding: *Paperback* • Estimated price: *$22.39*

Ruby Pocket Reference [U24] • By *Michael Fitzgerald* • ISBN: *1491926015* • Publisher: *O'Reilly Media* • Publication date: *August 29, 2015* • Binding: *Paperback* • Estimated price: *$10.72*

Web Development Recipes [U25] • By *Brian P. Hogan, Chris Warren, Mike Weber, Chris Johnson* • ISBN: *1680500562* • Publisher: *Pragmatic Bookshelf* • Publication date: *August 1, 2015* • Binding: *Paperback* • Estimated price: *$23.64*

Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON [U26] • By *Lindsay Bassett* • ISBN: *1491929480* • Publisher: *O'Reilly Media* • Publication date: *August 20, 2015* • Binding: *Paperback* • Estimated price: *$12.06*

Programming in D: Tutorial and Reference [U27] • By *Ali Cehreli* • ISBN: *1515074609* • Publisher: *CreateSpace Independent Publishing Platform* • Publication date: *August 19, 2015* • Binding: *Paperback* • Estimated price: *$25.26*

Mastering Probabilistic Graphical Models using Python [U28] • By *Ankur Ankan, Abinash Panda* • ISBN: *1784394688* • Publisher: *Packt Publishing - ebooks Account* • Publication date: *August 3, 2015* • Binding: *Paperback* • Estimated price: *$44.99*

JavaScript For Kids For Dummies [U29] • By *Chris Minnick, Eva Holland* • ISBN: *1119119863* • Publisher: *For Dummies* • Publication date: *August 24, 2015* • Binding: *Paperback* • Estimated price: *$16.47*

Formal Verification: An Essential Toolkit for Modern VLSI Design [U30] • By *Erik Seligman, Tom Schubert, M V Achutha Kiran Kumar* • ISBN: *0128007273* • Publisher: *Morgan Kaufmann* • Publication date: *August 28, 2015* • Binding: *Paperback* • Estimated price: *$81.89*

Jump Start Foundation [U31] • By *Syed Fazle Rahman, Joe Hewitson* • ISBN: *0992461278* • Publisher: *SitePoint* • Publication date: *August 13, 2015* • Binding: *Paperback* • Estimated price: *$16.41*

Basic Visual Formatting in CSS: Layout Fundamentals in CSS [U32] • By *Eric A. Meyer* • ISBN: *1491929960* • Publisher: *O'Reilly Media* • Publication date: *August 28, 2015* • Binding: *Paperback* • Estimated price: $9.99

Pragmatic Bookstuff

Here's what's up with the Pragmatic Bookshelf and its authors.

Here's what's happening at Pragmatic Bookshelf.

What's Hot

1	NEW	Exercises for Programmers
2	3	Rails, Angular, Postgres, and Bootstrap
3	6	Beyond Legacy Code
4	2	Programming Elixir
5	NEW	Agile Web Development with Rails 4
6	1	Mazes for Programmers
7	NEW	Programming Ruby 1.9 & 2.0
8	NEW	The Definitive ANTLR 4 Reference
9	4	Clojure Applied
10	NEW	The Pragmatic Programmer
11	7	Web Development Recipes 2nd Edition

Who's Where When

And here's what the authors are up to:

2015-09-08	**Treat your Code as a Crime Scene** Adam Tornhill (author of Your Code as a Crime Scene [U1]) Swanseacon, Swansea, Wales, UK [U2]
2015-09-09	**What Makes a Great Test Leader** Johanna Rothman (author of Behind Closed Doors [U3] , Manage It! [U4] , Manage Your Project Portfolio [U5] , Hiring Geeks That Fit [U6] , Manage Your Job Search [U7] , and Predicting the Unpredictable [U8]) SQGNE [U9]
2015-09-17	**Extensions Programming Workshop** Chris Adamson (author of iOS 8 SDK Development [U10]) CocoaConf Boston [U11]
2015-09-18	**Video Killed the Rolex Star** Chris Adamson CocoaConf Boston [U12]
2015-09-23	**Using Agile contracts in Sweden** Mattias Skarin (author of Real-World Kanban [U13]) Upphandla IT, Göteborg
2015-09-24	**Lead organizer** Alex Miller (author of Clojure Applied [U14]) Strange Loop - St. Louis, MO [U15]

2015-09-29 **This is a hands-on workshop where you'll learn how to leverage the capabilities of Kafka to collect, manage, and process stream data for big data projects and general purpose enterprise data integration needs alike.**
Jesse Anderson (author of The Cloud and Amazon Web Services [U16] and Processing Big Data with MapReduce [U17])
Strata NYC [U18]

2015-09-29 **Harness Your Leadership Workshop**
Johanna Rothman
Agile Cambridge [U19]

2015-09-30 **Keynote: Becoming an Agile Leader, Regardless of Your Role**
Johanna Rothman
Agile Cambridge [U20]

2015-10-05 **Treat Your Code as a Crime Scene**
Adam Tornhill
GOTO Copenhagen, Denmark [U21]

2015-10-05 **Swift and Robotics When Swift was announced, a lot of people saw roadblocks. Optionals, generics, why would you want immutable objects…? Our company saw an opportunity to make real changes to our code that would save us money and make our control softw**
Janie Clayton (author of iOS 8 SDK Development [U22])
360|iDev Min; Greenville, SC [U23]

2015-10-07 **Full day Workshop: Code as a Crime Scene**
Adam Tornhill
GOTO Copenhagen, Denmark [U24]

2015-10-09 **Learning Chess from Bobby Fischer I will be speaking about what it is like to learn directly from a master programmer. It's a wonderful opportunity, but there are some downsides you don't always think about that you need to handle.**
Janie Clayton
CocoaLove 2015, Philadelphia, PN [U25]

2015-10-13 **Full day Workshop: Code as a Crime Scene**
Adam Tornhill
Software Architect 2015, London, UK [U26]

2015-10-20 **Lean Kanban conference moderator. Topic: Evolving products and organisations Fast Feedback.**
Mattias Skarin
Lean Kanban Nordic - Stop Starting 2015, Stockholm [U27]

2015-10-25 **Workshop: Scaling Agile Projects to Programs: Networks of Autonomy, Collaboration and Exploration**
Johanna Rothman
Tel Aviv, Israel [U28]

2015-10-26 **Software Design, Team Work and other Man-Made Disasters**
Adam Tornhill
Trondheim Developer Conference, Norway [U29]

2015-10-26 **Workshop: Harness Your Leadership**
Johanna Rothman
Tel Aviv, Israel [U30]

2015-11-04 **Mine Social Metrics From Source Code Repositories**
Adam Tornhill
Øredev 2015, Malmö, Sweden [U31]

2015-11-05 **Extensions Programming Workshop**
Chris Adamson
CocoaConf San Jose [U32]

2015-11-06 **Video Killed The Rolex Star**
Chris Adamson
CocoaConf San Jose [U33]

2015-11-07 **Revenge of the 80s: Cut/Copy/Paste, Undo/Redo, and More Big Hits**
Chris Adamson
CocoaConf San Jose [U34]

2015-11-14	**Intro to Clojure workshop** Alex Miller Clojure/conj 2015 - Philadelphia, PA [U35]
2015-11-16	**Coming soon!** Mattias Skarin Lean Kanban Central Europe 2015

What's Happening

But to really be in the know, you need to subscribe to the weekly newsletter. It'll keep you in the loop, it's a fun read, and it's free. All you need to do is create an account on pragprog.com [U36] (email address and password is all it takes) and select the checkbox to receive newsletters.

Solution to Pub Quiz

Here's the solution to this month's pub quiz:

R	Q	T	U	C	O	K	I	S
C	O	K	I	R	S	U	T	Q
U	I	S	T	K	Q	O	C	R
I	R	U	K	T	C	S	Q	O
K	T	Q	S	O	I	C	R	U
S	C	O	R	Q	U	T	K	I
O	U	C	Q	I	K	R	S	T
Q	S	R	C	U	T	I	O	K
T	K	I	O	S	R	Q	U	C

Sorting the letters from any row, column, or smaller square just right yields QUICKSORT, the classic sorting algorithm.

Shady Illuminations

xxx

by John Shade

xxx

xxx

About the Author

John Shade was born in Montreux, Switzerland, on a cloudy day in 1962. Subsequent internment in a series of obscure institutions of ostensibly higher learning did nothing to brighten his outlook.