;login: logout

EXCLUSIVE ELECTRONIC EDITION JULY 2013

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Rikki Endsley
rikki@usenix.org

PRODUCTION
Arnold Gatilao
Casey Henderson
Michele Nelson

USENIX ASSOCIATION
2560 Ninth Street, Suite 215,
Berkeley, California 94710
Phone: (510) 528-8649
FAX: (510) 548-5738
www.usenix.org

;login: logout

Mobile Computing Research Is a Hornet's Nest of Deception and Chicanery

JAMES MICKENS

James Mickens is a researcher in the Distributed Systems group at Microsoft's Redmond lab. His current research focuses on web applications, with an emphasis on the design of JavaScript frameworks that allow developers to diagnose and fix bugs in widely deployed web applications. James also works on fast, scalable storage systems for data-centers. James received his PhD in computer science from the University of Michigan, and a bachelor's degree in computer science from Georgia Tech. mickens@microsoft.com

Mobile computing researchers are a special kind of menace. They don't smuggle rockets to Hezbollah, or clone baby seals and then make them work in sweatshops for pennies a day. That's not the problem with mobile computing people. The problem with mobile computing people is that they have no shame. They write research papers with titles like "Crowdsourced Geolocation-based Energy Profiling for Mobile Devices," as if the most urgent deficiency of smartphones is an insufficient composition of buzzwords. The real problem with mobile devices is that *they are composed of Satan.* They crash all of the time, ignore our basic commands, and spend most of their time sullen, quiet, and confused, draining their batteries and converting the energy into waste heat and thwarted dreams. Smartphones and tablets have essentially become the new printers: things that do not work, and are not expected to work, and whose primary purpose is to inspire gothic conversations about the ultimate futility of the human condition. People buy mobile devices for the same reason that goldfish swim in their tiny bowls: it's something to do while we wait for death. When researchers talk about mobile computers, they use visionary, exciting terms like "fast", "scalable", and "this solution will definitely work in practice." When real people talk about mobile computers, they sound like they're describing a scene from the Dust Bowl. It's all ellipses and gentle, forlorn shaking of the head. "I tried to load the app...I don't know what went wrong...I'M SO TIRED AND DUSTY AND BOWLED." Let me describe just a few of the problems with mobile devices:

Mobile browsers: When I use a mobile browser to load a web page, I literally have no expectation that anything will ever happen. A successful page load is so unlikely, so unimaginable, that mobile browsers effectively exist outside of causality—the browser is completely divorced from all action verbs, and can only be associated with sad, falling-tone sentences like "I had to give up after twenty seconds." The only reason that I use mobile browsers is that I hate myself and I want to be attritioned into unconsciousness by the desperate, spastic gasps of my browser as it struggles to download the 87 MB of Flash and JavaScript that are contained in any website made after the Civil War. Of course, some web pages are "mobile-enabled," meaning that they only contain 63 MB of things that I don't care about, instead of 87 MB of things I don't care about. To discover whether a page has a "fast-loading" mobile version, you can try to load the regular version, and then see if you get stuck in a hurricane of HTTP redirects, redirects whose durations have been carefully selected to make the load time of the mobile page *completely equivalent* to the load time of the standard, redirect-free version. If the Buddha intervenes and somehow coerces

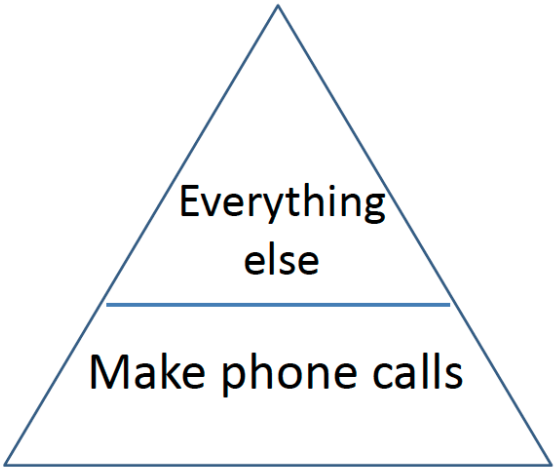
the mobile version of the page to load, you will be rewarded with a "phone-optimized" page that contains 1.5 visual elements (note that the most boring thing in the world has 3 visual elements). The vast majority of your mobile page will be advertisements for a newly discovered herb from South America that causes amazing weight loss. The amazingness of the weight loss will be demonstrated by a three-frame animation that depicts a fat person wearing a wife beater, a marginally less fat person wearing a wife beater, and a skinny person who, for inexplicable reasons, is still wearing a wife beater, even though he is now free to date supermodels, wear polar bear jackets, and do all of the other exciting things that skinny people presumably do when they're pumped full of South American mystery herbs. Importantly, the advertisements on your phone will position themselves in strategically disrespectful places, carefully obscuring the 0.25 visual elements that you actually want to see. When you scroll the page, the ads will engage in a frantic dance to reposition themselves in a maximally infuriating way. You will eventually give up and close the browser, having spent 45 minutes to unsuccessfully load a web page about dogs that look like cats that look like other, different cats.

Touchscreens: When touchscreens work, they're amazing; however, touchscreens are the only commodities which depreciate faster than automobiles. As soon as you unwrap your phone or tablet, t*he touchscreen starts to die.* Make no mistake, your initial touchscreen romance will be lovely and full. Hark!—as you effortlessly move neon blobs of information like a character from "Tron"! Behold!—as you zoom into and out of a dynamically resizable thing that contains additional-but-only-partially-resizable things! Such experiences represent the springtime of your love, and the initial weeks of your touchscreen romance will be like a young Led Zeppelin, intense and grandiose and punctuated by extended guitar solos. However, at some point, you will drop your phone or your tablet, and this will mark the beginning of the end. When you drop a touchscreen, you initiate a complex series of degenerative processes that corrupt the touchscreen and turn its will against you like a pet lizard who has learned that dinosaurs were real BUT IT'S JUST A STATE OF MIND. Note that, when I say that you will "drop" your touchscreen, I do not mean "drop" in the layperson sense of "to release from a non-trivial height onto a hard surface." I mean "drop" in the sense of "to place your touchscreen on any surface that isn't composed of angel feathers and the dreams of earnest schoolchildren." Phones and tablets apparently require Planck-scale mechanical alignments, such that merely *looking* at the touchscreen introduces fundamental, quantum dynamical changes in the touchscreen's dilithium crystals. Thus, if you place your touchscreen on anything, ever, *you have made a severe and irreversible life mistake.* Slowly but surely, your touchscreen will develop a series of tics and glitches, behaviors that you will initially explain away as "technology is quirky," but that you will quickly begin to describe using extraordinary and significant profanities that are normally employed by Marines and people who work with radioactive waste. On your touchscreen, your swipes will become pinches, and your pinches will become scrolls, and each one of your scrolls will become a complex thing never before seen on this earth, a leviathan meta-touch event of such breadth and complexity that your phone can only respond like Carrie White at the prom. So, your phone just starts doing stuff, all the stuff that it knows how to do, and it's just going nuts, and your apps are closing and opening and talking to the cloud and configuring themselves in unnatural ways, and your phone starts vibrating and rumbling with its little rumble pack, and it will gently sing like a tiny hummingbird of hate, and you'll look at the touchscreen, and you'll see that things are happening, my god, *there are so many happenings,* and you'll try to flip the phone over and take out the battery, because now you just want to kill it and move to Kansas and start over, but the back panel of the phone is attached by a molecule-sized screw that requires a special type of screwdriver that only Merlin possesses, and Merlin isn't nearby, and your phone is still rumbling, and by this point, you can understand the rumble, it's a twin language that you and your phone invented, and the phone is rumbling, and it's saying that it's far from done, that it has so much *more* that it wants to do, that there are so many of your frenemies that it wants to "accidentally" call and then leave you to deal with the social ramifications, and your phone, it buzzes, and you think that you see it smiling, and you begin to realize that land-line telephones were actually a pretty good idea.

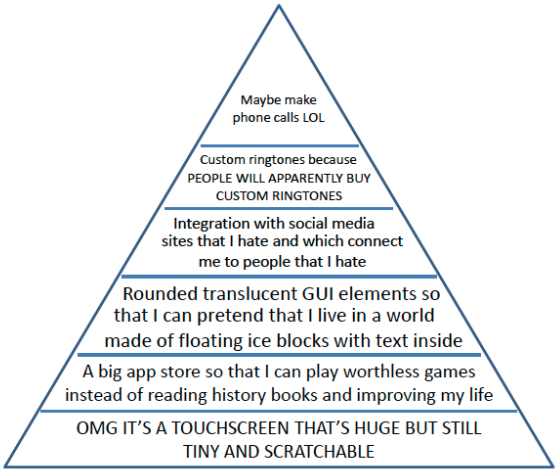
Call quality: Interestingly, a mobile phone *should be able to make phone calls while it moves through time and space.* I derived this provocative concept from basic notions of adjectives and nouns. For example, if I am a gregarious jellyfish, I praise my friends for their wardrobe choices (gregarious) while I repeatedly stab them with my poisonous tentacles (jellyfish). I am a gregarious jellyfish. That is my way. I may be misunderstood by polite society, but as a gregarious jellyfish, my dramatic tensions respect the standard semantics for adjectives and nouns. Similarly, a mobile phone should be able to PHONE PEOPLE while being MOBILE. However, I have never had a successful conversation on a mobile phone. Whenever I talk to people on a mobile phone, they always sound distant and/or creepy, like they're trapped in an echo-filled cave, or a windy cave, or a cave that makes people sound like pedophiles. These are not good caves to be in, to the extent that it's ever good to be in a cave. Nobody takes their honeymoon at Persistently Distracting Echo Cave. Nobody has their Bat Mitzvah at Windy Cave's 80% Packet Loss Ballroom. You may, in fact, find online travel deals for Pedo-Cave, but these are all traps that have been set by "To Catch A Predator." My point

Mobile Computing Research Is a Hornet's Nest of Deception and Chicanery

is that mobile phones are not phones. They are just pocket-sized things that are more expensive than the vast majority of other pocket-sized things. In a rational world, the Maslow's Hierarchy of Needs for Mobile Phones would look like this:



However, in today's world, the hierarchy looks like this:



This is why, when you try to talk to someone on a mobile phone, you are thrown into a frantic world of on-the-fly lossy decompression, like Nicholas Cage in that movie about Navajo code talkers (the only movie that managed to simultaneously offend Native Americans, cryptographers, and people who are neither Native Americans nor cryptographically inclined).

Conclusion

In the minds of mobile computing researchers, humanity is nearing a final, glorious stage of Darwinian evolution, in which mankind and smartphones emerge from a shared chrysalis and transform into shapeless, omnipotent joy clouds of excellence and victory, unconstrained by conventional morality or finite battery life. In reality, you could go to the Middle Ages, find a random person, and take whatever is in his left pocket, and you would have something that is more useful than a modern mobile device (although it may be covered with Bubonic plague or antiquated notions about the stoning of random villagers with respect to the actual size of the witch population). When you purchase a mobile device, you are basically saying, "I endorse the operational inefficiency of the modern bourgeoisie lifestyle, even though I could find a rock and tie a coat hanger around it and have a better chance of having a phone conversation that doesn't sound like two monsters arguing about German poetry." So, I encourage you to throw your tablets and your mobile phones into a fire, and then hide from the angry monsters who no longer have a way to discuss the work of Klaus von Beckenbauer, the acclaimed poet who wrote "The Unsurprising Laments of the Gregarious Jellyfish," "Seriously, Todd, You've Got To Stop Stabbing People If You Want To Get Married," and "Yes, Jellyfish Have Names and My Name is Todd."

Monitor the Monitors

SELENA DECKELMANN

Selena Deckelmann is a major contributor to PostgreSQL and a data architect at Mozilla. She's been involved with free and open source software since 1995 and began running conferences for PostgreSQL in 2007. In 2012, she founded PyLadiesPDX, a portland chapter of PyLadies. Deckelmann founded Open Source Bridge and Postgres Open, and she speaks internationally about open source, databases, and community. She is an advisor to the Ada Initiative, an organization dedicated to increasing the participation of women in open source and technology communities. You can find her on twitter (@selenamarie) and on her blog at chesnok.com. selena@chesnok.com

A colleague asked: What tools do people use for monitoring the monitors? His question was inspired by "Dead Man's Snitch," a web-based tool that detects and reports out when a regularly scheduled job hasn't run. This got me thinking about all the kinds of systems teams set up to detect silent failures. Most involve turning silence into noise.

When asked, friends had quite a few different names for these strategies: meta-monitoring, the "everything's ok" alarm, a "canary in the mine," Dead Man's Switch, heartbeat, watchdog, high availability response, and the entertainingly painful OOBETET (out-of-band-end-to-end test).

You could divide that list up in a few different ways:

- Automated vs. manual execution
- Automated vs. manual response
- Destructive vs. non-destructive response
- Monitoring vs. monitoring of notification services

Classifying the strategies seems fairly simple. The Dead Man's Switch is automated and contains a destructive response, at least in the movies. An out-of-band-end-to-end test is often manual both in execution and response, and non-destructive.

Definitions become a bit fuzzier, however, when we consider "monitoring vs. monitoring of notification services". For example, how do you ensure (within reason) that text message alerts actually made it to the staff who can fix a problem?

This kind of testing seems to fall under "out of band" checks. These are verification routines we don't include in our primary monitoring systems, just in case the primary systems aren't available.

Teams have lots of informal—and sometimes formal—ways of managing these issues. For example, when on-call switches over, the phone company may send an initial alert text message to the new shift's phone number.

In my first job out of college, our team sent out a test alarm at 9 a.m. every day to the on-call team member, signaling that the paging service was still working. If you were on call and didn't receive the page, you knew to contact your manager to get things fixed as soon as possible.

Critically, this kind of verification system is not fully automated. Instead, the system is managed by training people to notice a simple signal (in this case, a disruption in a well-known routine) and supplying instructions for exactly how to respond.

Much of the work involved in keeping formal out-of-band checks working and useful is in documentation and training. When systems are designed for human intervention, there's

a temptation to not document the process fully because you know a person must puzzle their way through responding to the alarm anyway.

But there's significant value in documenting even our informal process. Training is much easier when you've got a document to work from, and you can have some degree of assurance that team members will run through steps to fix problems the same way every time. If something changes, you know where to document the change – in the written documentation. You may find that the check and even the fix can be automated. And, staged failures to test the procedure is a lot easier to manage when you have a fix procedure to test.

So, take a few minutes and have a look at your team's out-of-band monitoring. What are the things that might fail silently? Can you turn that silent failure into noise? How will your team detect and respond? And then write it down!

Setting Up a Vagrant Workflow

MATT SIMMONS

Matt Simmons is a 12+ year system administrator who works at the College of Computer and Information Science at Northeastern University in Boston. He blogs at http://www.standalone-sysadmin.com/ and can be reached via @standaloneSA on Twitter. standalone.sysadmin@gmail.com

Fire. The wheel. The Internet. Microwave cheese. These are things that we, as a species, have created and really matter to us in our daily lives, and that have appreciably made the world a better place. I want to cast my vote to add Vagrant [1] to that list.

Remember the bad old days, when you would write configuration management code, commit it to a repo, check it out in the testing environment, reboot a machine, and then a few minutes later figure out that you left out a semicolon, so you'd have to do it all over again? That whole workflow is so 2011.

Not that long ago, I was listening to some trainers talking about offering a Vagrant Box to people attending their classes at conferences. Being the naturally inquisitive sort of person that I am, I rudely interrupted their conversation to ask what they were talking about. I learned that Vagrant was apparently a "thing" that made "VMs" from "images".

Now, I'm more than passingly familiar with the whole "virtualization" deal, so I felt like I had a decent grasp of things from that description. I mean, I didn't think Vagrant was anything revolutionary, but I could kind of see where it was going. I thought Vagrant was something maybe like VMware's marketplace or maybe a nicer way for people to distribute their images or something. In a sense, I was kind of right, but in reality, I was way off. Vagrant is so much cooler than that.

As I found out later, Vagrant is an abstraction layer above virtualization solutions, typically things such as VirtualBox or VMware Fusion. These virtualization products, meant to be desktop solutions, have rather robust back-end capabilities and offer headless solutions that are of absolutely no use to you as a desktop, and they default to a console display if you use their interfaces, which is annoying if you want to use them as a server environment.

Vagrant is a way of automating and controlling the creation and destruction of those machines, but above and beyond that, Vagrant images (or boxes, in the lingo) have certain software installed, configured, and ready to be put to use by you for all of your nefarious (or not) purposes.

Here's how my current workflow looks. Suppose I've got an Ubuntu machine on which I want to play with Vagrant. I'll install it like this:

```
$ sudo install vagrant
```

By default, Vagrant doesn't come with any boxes to make new machines from, so lets add one:

```
$ vagrant box add precise32 http://files.vagrantup.com/precise32.box
```

References

[1] Vagrant: http://www.vagrantup.com/

[2] VagrantUp.com: http://www.vagrantup.com/

[3] Vagrant documentation: http://docs.vagrantup.com/v2/

Setting Up a Vagrant Workflow

This adds a box named precise32, and the source image...err, box...is downloaded from the given URL. We're really close now:

```
$ vagrant init precise32
```

This creates a configuration file that Vagrant will use to build the machine. The defaults will give us a nice clear template to work with. Now, we're ready:

```
$ vagrant up
```

Ta-Da! You now have a machine. You can connect to it like this:

```
$ vagrant ssh
```

It's up and running, with whatever image you wanted. Want to shut it down? Exit from the ssh session just like you normally would, then type:

```
$ vagrant destroy
```

Poof. Gone!

I realize that this is a simplistic example of what's possible, but look through the Vagrantfile and you'll see an entirely new world open before your eyes. You can create a Puppet or Chef configuration and have it run automatically on boot, or run an initialization shell script, or even create multiple VMs at once and build an entire infrastructure in miniature, then destroy it with less effort than it takes to kick down a sandcastle.

To check out Vagrant, I recommend working through the exercises at VagrantUp.com [2] and then read the documentation [3]. Vagrant has completely changed the way that I test my Puppet code, and once you grok it, I'm certain that it'll change yours, too. Feel free to write me at standalone.sysadmin@gmail.com and let me know what you think of it.