Issue 009: Books, Blogs, Podcasts And Conferences

Adrian Kosmaczewski

June 3rd, 2019

Welcome to the ninth issue of *De Programmatica Ipsum*, dedicated to the subject of *Books, Blogs, Podcasts and Conferences*. In this edition without a guest writer:

- Graham exhorts for a change[1] in the way knowledge is shared in the industry.
- Adrian argues that we have been manipulated by evangelization[2].

Enjoy this issue! Please let us know if you have any feedback[3] and get our free newsletter[4] to stay updated about new releases. If you want to support us, subscribe[5] for a month or a year, and let us know if you would like to write with us[6].

Cover photo by Matt Botsford[7] on Unsplash[8].

[1] https://deprogrammaticaipsum.com/the-software-industry-must-rethink-knowledge-sharing/
[2] https://deprogrammaticaipsum.com/less-evangelization-more-honestization/
[3] https://deprogrammaticaipsum.com/feedback/
[4] https://deprogrammaticaipsum.com/newsletter/
[5] https://deprogrammaticaipsum.com/subscribe/
[6] https://deprogrammaticaipsum.com/write-with-us/
[7] https://unsplash.com/@mattbotsford?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[8] https://unsplash.com/search/photos/podcast?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

The Software Industry Must Rethink Knowledge-Sharing

Graham Lee

June 3rd, 2019

It seems like we've come full circle, doesn't it? We published our first issue on hype[1]. Now here's an issue on books, blogs, podcasts, conference talks...this is the same thing, yes?

Yes.

Knowledge-sharing activities in the software industry have a financial model that follows a long-tail distribution. Almost every software podcast you listen to costs its presenters money in bandwidth fees. A few of them make revenue from advertising. Most of the software podcasts that exist, though, are the ones you *don't* listen to, and probably haven't even heard of.

Almost every software book you read took months of full-time effort to write (either in lieu of, or addition to, a "day job") and made a small amount of money as an advance on sales. A few of them break into profitability. Most of the software books that exist, though, are the ones you *haven't* read, and probably haven't even heard of.

Almost every software conference talk you have seen was delivered for free, perhaps with some honorarium to cover expenses. A very few speakers break into the world of the paid keynote speaker, and a small fraction of those give the same talk often enough to amortise the preparation costs and make money from the talk. Most of the software talks that exist, though, are the ones you *haven't* seen, and probably haven't even heard of.

[1] https://deprogrammaticaipsum.com/issue/issue-001-hype/

And of course those talks are given at conferences. Almost every software conference you have seen cost the organisers money, even where you were paying to attend. A few manage to break even, usually with sponsorship. A few of them break into profitability. Most of the software conferences that exist, though, are the ones you *haven't* attended, and probably haven't even heard of.

To recap, most of this activity, certainly the "at-scale" activity, pertaining to sharing knowledge and experience in the world of software development, comes at a cost to those who perform it. Presumably, then, the people involved have another reason for putting time and other resources into this work. Often, that's promotion.

A consultant can give a prospective client a business card that says they can help with some business activity. It's much more impressive to give that prospect a book about the topic, with the consultant's name on the cover. Life gets even better if you don't have to pursue the prospect at all, and they come to you after listening to your podcast or watching your talk at a conference.

What about salaried staff who write books, give talks, present podcasts? Some are hyping themselves, playing the long game of boosting their profile so that future opportunities, interviews, or promotions come easier. Others work for their marketing teams, hyping their employers, who are banking on a little evangelism going a long way in either promoting their product or their workplace. MongoDB famously had a successful sponsored "grass-roots" evangelist network long before they had[2] a working database.

Much of the software world is written for money, so where we don't see money we naturally look for economic externalities. With knowledge-sharing work, the externality is self-promotion.

No.

Helpfulness is also an externality. De Programmatica Ipsum exists because Adrian and I wanted it to exist. We wanted to read it. We wanted *you* to read it. You could not read it if nobody created it, so we created it. People write blogs because they found out things that were interesting, or would have been helpful to know. You read them because they are interesting, or it is helpful to know what was written. Plenty of people who write or speak do so because they want to help their community. In a way, this also defines and shapes the community. A blog written for mobile app testers defines its readers as mobile app testers, or people who are interested in mobile app testing. That's more restrictive than, say, software testers. It's less restrictive than, say, Flutter app testers. The blogger has defined the size of the group, and who is within or without that group.

Another externality is enjoyment. Some people enjoy writing, or podcasting, or preparing and giving conference talks, and do it *despite* the cost in the way that other people play music, cycle around town, or compete at football, without expecting to profit from it. Indeed, some combination of all four motivators we've identified so far – money, promotion, helpfuless, and enjoyment – with different "weights", led to the creation of each talk you've watched, each book or article you've read, and each podcast episode to which you've listened.

[2] https://www.nemil.com/mongo/1.html

Software Engineer, Disrupt Thyself

Knowledge sharing is, in a knowledge economy, unsurprisingly very useful. But, surprisingly, not particularly economical. As I have argued here, the activities that define our communities and through which we disseminate information are not ones that will make you money. A programmer can improve through training, education, study, and a connection to a community of practice and expertise. Many of the artifacts of training, education, study, and the community of practice are loss-making and very few are sustained through point-of-use payment.

The software craftsmanship movement[3] claims to be "raising the bar" by "helping others to learn the craft". Towards this end, their manifesto says that craftsmanship means valuing "a community of professionals" and "productive partnerships". There is no evidence in the further reading[4] for the manifesto that this idea of value includes any monetary definition. This is unsurprising: software professionals also make heavy use of free and open source software without contributing to its creators. If it's hard to get software businesses to value software, how would we get them to value things that only *indirectly* become software?

Difficult though it may be, I believe it to be necessary. There are many dimensions along which software construction could improve: access, diversity, usability, quality, fairness, reliability; these are all areas in which we have industry-level problems. If we want industry-level solutions we must be able to pool our collective resources, and build sustainable ways of "uncovering better ways of developing software by doing it and helping others do it". For it to be sustainable, it cannot just be the hobby of some interested individual or a sideshow a consultant uses to drum up business. There must be a way to fund the improvement of our industry.

There is plenty of innovation in the world of documentation *tools*. The OpenAPI initiative[5] defines a specification for formatting web service documentation. IDEs like lighttable[6] and project jupyter[7] allow deeper integration between code and its documentation than before. Publishers have been on a journey from TeX[8] through OpenDoc, LaTeX, and Markdown, making it easier for an author to type a book in to a computer. Much less has changed in the *practice* or *perception* of documentation, training, and knowledge sharing. People have misinterpreted "we have come to value working software over comprehensive documentation" to mean "we do not value documentation, or documenters, at all".

Question-and-answer sites like project-specific forums, Stack Overflow and Quora give people a great opportunity to find the answer to the questions that are blocking them *today*. As with personal blogs and podcasts, the creators who answer questions on these sites do not get paid: indeed they are digital sharecroppers[9] whose work makes money for the site owners.

Collectively, the (small-u) union of software professionals has come to rely more on these landlords and their q-and-a sites than on other forms of knowledge sharing. Unix command-line tools, Alexa skills, and IDE plugins all let us ask questions on Stack Overflow without disrupting the flow. It's become so easy to solve the problem I have *today* that there is no need for me to level up so that I can have newer, higher level questions tomorrow. Where

[3] http://manifesto.softwarecraftsmanship.org/
[4] http://manifesto.softwarecraftsmanship.org/#/en/reading
[5] https://openapis.org
[6] http://lighttable.com
[7] https://jupyter.org
[8] https://deprogrammaticaipsum.com/the-art-of-the-art-of-computer-programming/
[9] http://www.roughtype.com/?p=634

will the better ways of developing software come from if we all get accustomed to the idea that the existing ways will work, given enough time and enough upvotes?

A Solution?

At the outset of this article, I noted that the "content generation" side of the knowledge advancement industry is long-tailed. For every "Learning Python" there are a hundred "Teach Yourself C++ in 21 Days", ten thousand "Get Started with Erlang and Kubernetes" and one million "Advanced Mathematical Computing in Postscript" titles. The reason is that the readership is the inverse of this: for everybody who wants to read an "Advanced Mathematical Computing in Postscript" book, there are one million who will read "Learning Python".

Unsurprisingly, then, you can look to any publishing house (any that remains in business) to see how we might run an organisation dedicated to knowledge-sharing in the software industry. They have a business model that is tailored to the long tail. Think of the mass market publishers Penguin Random House, who missed a trick by not adopting the name "Randy Penguin" upon their merger. They publish Sir Terry Pratchett's *Discworld* series and E. L. James's *Fifty Shades of Grey*, and the money from those titles lets them take risks on publishing titles into the longer tails like esoteric literary fiction.

Or indeed a software-focussed publishing house like O'Reilly. Their "Learning React" book is much more likely to pay for itself than "Decentralized Applications", but the fact is that "Learning React" probably paid for both.

So the long tail market model of a publishing company fits well the long-tail model of knowledge sharing (obviously: as nonfiction books are representations of knowledge). Our long-tail model is one of industry improvement. Plenty of people want to become programmers, software testers, sysadmins, penetration testers, SaaS marketers, or take on other roles in the software industry. Some fraction of those want to be better at those roles. Some fraction of those want to become experts at those roles. And some fraction of those want to transcend those roles, and redefine computing.

The publisher model also has its problems, otherwise books wouldn't have appeared at the start of this article. Publishers seem to have high enough overheads that the (absolute) revenue going to creators in the long tail is tiny. The distribution is probably entirely equitable: all authors are offered an advance, followed by a percentage of revenue. But that means that no matter how *important* the topics in the long tail, creating for that market is not going to pay well. In academia, the model is even reversed, with authors being charged to publish their articles in Open Access journals, in return for the article being freely available to all. In that world, the authors can write their costs into their funding requests. The problem to be solved back in the software world is that there *isn't* a magic pot of money for content creation.

The combination of a long tail (more novices than juniors than seniors than architects than...) and the increased influence of the individuals out in the tail leads to the conclusion that versioned pricing can address this problem. Beginner material can be cheap, because it will be sold in volume, and then you have the benefit that beginner material becomes *accessible* to those of lesser means. Expert material becomes expensive, which takes advantage of the fact that experts get paid more so can put more in, and provides for the fact that there are fewer people who *can* credibly share expert knowledge with experts, so their skills are rare and deserve recompense.

By the way, this money doesn't need to come from individuals, their employers should be

responsible for their development too. In many places I've worked, the stated budget for training has been much larger than the amount that's actually spent by employees. So something you can do *right now* to improve your own career and the availability of high-quality knowledge-sharing resources is ask your employer how much they set aside for professional development, then go out and spend that money.

As it happens that question, and also the response to your attempts to use the money, are great for finding out to what extent your employer walks the walk when they say that they only hire the best. Someone, somewhere, must be hiring the mediocre and below-average developers, and it's probably the people who turn out not to have a training budget.

Cover photo by Miguel Henriques[10] on Unsplash[11].

[10] https://unsplash.com/photos/-8atMWER8bI?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[11] https://unsplash.com/search/photos/lecture?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Less Evangelization, More Honestization

Adrian Kosmaczewski

June 3rd, 2019

One of the first, and still one of the most well-known evangelists in the history of technology, is without any doubt Guy Kawasaki. In an article in the Harvard Business Review[1] he explained his role and gave an interesting overview of what it is, its consequences and benefits. Written, of course, for the audience of HBR, which might not have much overlap with the audience of this humble publication you are reading right now.

The word, however, is still puzzling to no end. "Evangelizing."A word of greek origin (thanks, Guy) that immediately brings us to the world of religions. The problem with such a word is that, after hearing it, a quick overview of the current state of the world (a savant mix of individualism, hysteria, extremism, and sectarism,) will prompt any sentient being with some remain of empathy to quickly retreat to a remote island in the Arctic, and wish for themselves to be forgotten forever.

And the truth is that, as Guy aptly mentions, in this age of social media, we are all evangelizers.

Actually, in a rather sad metaphysical conundrum, both Graham and the author of these lines *are* evangelizing you, our faithful readership, with our own opinions of the state of the industry. Who are we kidding?

Lies

We all know the difficult state of our industry. We are producing more software than ever; the working conditions of those making it are worsening; and the resulting quality of our work is dwingling and breaking havoc. Yet, we still fall for charmismatic speakers on stages,

[1] https://hbr.org/2015/05/the-art-of-evangelism

writing incredibly detailed blog posts, speaking clearly through their Yeti microphones[2] and luring us into yet another wave of hyped technology.

After decades of blindly eating those words and falling into despair, a change is taking place. We do not simply take those words and swallow them without reflection. We have seen the terrible consequences of adopting technology just for the sake of beefing up a resumé. At least, this author wants to believe that those reading these words have asked themselves this questions, and that we fill our coffee mugs every morning with healthy, equal doses of caffeine and skepticism.

Honestization

Let us create a new word. After all, this is one of the greatest skills of our industry, the other being inventing acronyms. Instead of *Evangelizing* a new technology, in whichever medium you choose to do that, maybe we should be *Honestizing* them.

From a practical point of view, that would mean putting forward all the flaws and shortcomings first. Imagine reading a blog post about a new Node module that starts with the enumeration of all its drawbacks, incompatibilities, reduced scope of testing, limited range of applicability, and which would then (and *only then*) start explaining its benefits. This author would personally would welcome such an approach (not only for Node modules, that is, although given the pervasiveness of JavaScript in our industry it might be a great place to start.)

How would Honestization work? We could use something like an MPAA film rating[3] for conference talks, blog posts, podcasts and other social media channels:

- G for introductory talks, mostly harmless pieces of information about team dynamics, project management methodologies, or being a developer at 40[4].
- PG for "Professional Guidance Suggested;" the contents of such talks might have unintended consequences if applied in sensitive environments, like production.
- PG-13 would mean "Professionals Strongly Cautioned," where the "13" means years of experience in the branch. That means, stuff most developers should not care about or be able to apply in their jobs before reaching the very senior age of 35 years old.
- R for "Restricted;" heavy, dangerous advice not to be taken lightly, and only to be applied by white-bearded, senile professional staff, that is, above 40 years old.
- Finally, NC-17 for any tutorial or conference session about the C programming language, whichever its subject or applicability may be.

Of course the writer of these lines will not be holding its breath until this happens, for such a system will never see the day of light, just like there might never be a Hippocratic Oath[5] for software developers.

There is, however, tremendous value in such a system.

Documenting not the benefits of a particular technology or idea, but rather of all its shortcomings. Leaving behind the positivity that is drowning us in lies, with "small print" level of warnings – when they exist at all, or behind a 35-page "License" that one has to "Agree" whenever downloading a piece of technology.

[2] https://www.bluedesigns.com/products/yeti/
[3] https://en.wikipedia.org/wiki/Motion_Picture_Association_of_America_film_rating_system
[4] https://akos.ma/blog/being-a-developer-after-40/
[5] https://deprogrammaticaipsum.com/primum-non-nocere/

Making our industry more honest. Stating openly that things break. Making sure everybody agrees that things do not really work the way they are supposed to, except in some very specific environments and situations. Maybe those environments and situations can be reproduced easily, and thus the true value of a piece of software can unfold in front of us. But in those cases where this is not possible, just be upfront about it.

Do Not Lie To Me

It all boils down to the idea that making a better world with software means being sincere to one another. Avoid the lies. Do not be a hypocrite. Be upfront about the shortcomings of your software. Do not sell it as a solution for anything and everything, and do not shy away from saying what is wrong with your ideas.

Paraphrasing Joel Spolsky, know that whichever abstraction comes up in your mind, they will be leaky anyway[6]. Acknowledging those leakages from the start might be the bowl of air our industry needs to start afresh.

Cover photo by Efren Barahona[7] on Unsplash[8].

[6] https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/

[7] https://unsplash.com/@efrenbarahona3?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[8] https://unsplash.com/search/photos/microphone-conference?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText