MIMA Summit Outline

1. What are we going to cover today?

A few questions in front of us I want to look at:

- What is a developer?
- What does the answer to that question mean for each of us?

2. What is my context for talking about this?

- I run the largest code school in the country. We are 10 campuses strong.
- In 12 weeks, we teach people how to write code and then help place them in jobs after the program.
- We have to keep a keen eye on both sides of the equation:
    - What skills do we need to equip our graduates with?
    - What skills are companies hiring for?

- Those two things aren't always perfectly aligned and we've learned some lessons on our quest to close this gap

3. So, back to the original questions:

- What is a developer?
- What does the answer to that question mean for each of us?

4. What is a developer?

We all have our own idea of what a developer is...

- If you're not a developer, you may think being a programmer looks something like this (money gif)
- If you are a developer, you might think being a programmer looks more like this (frustrated gif)
- If you don't know how you ended up in this room on the developer track, you maybe responding to this question like this (cool guy gif)

Let's explore the answer to this question. We'll start by looking at where we have been.

From hard skill and qualifications to curiosity and problem solving

The [remnant] demand for hard skills

I used to work in marketing and let me tell you the types of questions that were NEVER asked in an interview:

- Can you name the various stages of the product life cycle?
- Can you correctly identify the name of the trough in Gartner's chart on hype?

Interestingly, though, this mindset still reigns in the world of software development.

> "Every software engineer has experienced a job interview in which he or she was forced to act more like a dictionary than a human being. There's a prevalent misconception that, in order to properly develop software, one must know all the minute details of any programming language listed in the job requirements. It's a remnant of the days when developers received their knowledge from physical manuals."

- Add to this that languages were young enough and small enough to attain that level of knowledge to some extent. Today, the number of languages and libraries make that impossible.

Job descriptions are a great example:

> ...if you look at just about any Facebook job posting for an engineer, you’ll see a B.A. or M.A. in computer science listed as a “requirement.” It isn’t, according to Serkan Piantino, who leads Facebook’s New York office.
>
> “Part of our recruiting strategy is to be pretty agnostic to the things that don’t matter,” Piantino told Quartz in an interview earlier this summer. “Things like somebody’s prior background, whether they went to a top [computer science] program or never graduated high school, if they’re a good fit for Facebook we just try to focus on that.”

In other words, hard requirements are less important than the person's ability to solve the problems they'll face in the context of the company they're working for.

A move towards tool-agnostic problem solving

> "Job opportunities for web developers are expected to be good. Those with knowledge of multiple programming languages and digital multimedia tools...will have the best opportunities" - Bureau of Labor Statistics

Why is that? This could be an entire presentation in and of itself, but here's are a few significant drivers:

- The number of technologies available to developers has grown at a blistering rate and it will continue to do so
- The creation of new tools (i.e., languages and libraries) creates entirely new ways of solving problems, adding to the accelerated rate of change

This means that the definition of expertise is changing. The pace of change and available tools mean that people are increasingly required to get good at *thinking*, not using a specific tool. Don't hear me wrong, deep knowledge of a discipline is extremely valuable, but an emphasis on *problem solving* allows for rapid adaptation to changes in technology.

To get some feedback on this idea, I asked a question to one of our instructors at The Iron Yard. This person is qualified to teach 3 or 4 different programming disciplines at a high level. I asked him, *"You're valuable because you know so* *many different languages so well, correct?"*

His response:

> I'm not sure I would say my grasp of languages/tools are what make me valuable, but more that I have a set of patterns and mental models that 1) allow me to learn new things quickly and 2) allow me to synthesize different topics into one coherent whole.

Brett Victor summarizes this idea really well:

> Learn tools, and use tools, but don't accept tools. Always distrust them; always be alert for alternative ways of thinking. This is what I mean by avoiding the conviction that you "know what you're doing".

All of these things have been brought about in large part by the open source software movement.

The open source movement as our pathway forward

Part of this is a numbers game. The reality is that use of open source software is growing at an exponential rate.

From a 2008 paper: http://dirkriehle.com/publications/2008-2/the-total-growth-of-open-source/

- 1995-2006

Here's where it's going: [graph from open source presentation]

The numbers alone suggest that in order to best grasp the changes that are happening on the bleeding edge of technology, participating in open source projects is your front-row ticket to the action.

Practically, though, there are reasons that people are attracted to open-source.

First, you get better, faster.

> I would say that with closed source tools, it's really hard learn from them. I can only use them. Whereas, with open source (Rails, Ember, Angular, whatever), I am able to look at their source code and learn from people much better than me.

Second you can develop software in a community where the people *using* the technology are also *building* the technology.

> Closed source communities are divided between those that create the tools, and those that use the tools. In open-source, they're basically the same community. So I can get to know the person who wrote this code.

Let's stop there for a second: you can literally get to know the people who *invented* the tools you're using.

These trends in open source are reflective of a larger trend in the democratization of learning programming

I won't spend too long here, but the short story is that code education is being democratized. Hundreds of online tools allow you to learn for free or almost-free. If you talk to anyone at Treehouse, for example, you'll quickly learn that the heart of their mission is making technology education available to as many people as possible.

- Georgia Tech offers a masters in computer science online for $6,600
- Journalism schools are beginning to incorporate programming into their curricula
- Online programs like Thinkful offer curriculum *and* person-to-person mentorship
- Programs like The Iron Yard can take someone from zero experience to a job paying $60-80k in a matter of months

This democratization of learning has allowed people to look at the world around them and realize they have the tools to solve the problems they see or build the things they want to experience.

The only limitations are curiosity and time.

So...what is a developer?

Here's a working definition:

> Someone who understands a real-world problem and solves it using code-based technologies.

What does this mean for you?

Most people in this room fall into three categories:

1. You are a developer

Briefly:

As Bret Victor said, over-reliance on specific tools is dangerous and diversifying your skill set has never been more important. As we interact with more and more companies, we're seeing that developers who, while really good at one discipline, have experimented with lots of other technologies are in the highest demand and make the most money.

It goes back the problem solving principle—companies don't want to hire one-trick ponies, they want to hire polyglots who can take a fresh look at how to make something work in the best way possible.

Again, I believe that diving headlong into open source projects is the absolute best way grow as a programmer.

Examples? Ryan Poplin, Todd Heidenreich, etc.

2. You hire and/or lead a team with developers on it

First, allow for exploration.

- "this is just the way we do things" doesn't work when the people who

invented the technology you're using are still alive, are accessible and are still pushing boundaries
- we've worked with many companies who have legacy stacks and are losing developers because they don't budget for allowing their engineering teams to experiment with new technologies (even if they aren't shipped to production in the short term)

Re-think your hiring process

- break the mindset that "we need to hire a PHP dev" and think, "we need to hire a creative problem solver"
- what do you think about PHP? "It sucks" means you don't have an informed opinion
- how well do you know X on a scale from 1-10? "if they say 8 or 9" that's BS. if agencies started to ask those questions they would be way, way better off

Prepare to fight hard over talent (if you aren't already)

- The appetite for exploration caused by the rise of open source technology and democratized education has, for better or worse, given developers looking for work more leverage.
- Because developers are in such high demand, salary expectations are escalating
- Because the supply of jobs is so abundant, emphasis on company culture and other job perks (like working remotely) is increasing
- Shifting preference and interests means that many companies will (and already are) experiencing a higher churn rate among their engineering teams. Good or bad, we need to plan for it.
- The ability for developers to build and sell their own products means that that we'll see an increasing number of programmers who want to pursue an entrepreneurial path. Retaining talent might mean creating that path inside of your company.

3. You work with a developer in some capacity

Two areas of responsibility: those who work with developers, and the developers themselves

First, those who work with developers

Just because you don't write code doesn't mean you don't need to educate yourself on this stuff.

> If you understand code, you strengthen your ability to act as a valuable liaison, no matter what your role. It makes you a valuable asset if you’re working within a company, and a stronger leader if you’re building your own. You will also remain empowered when hiring and managing developers or outsourcing efforts. Because you are familiar with the same tools they are, you will know what is feasible, what is efficient, and the true level of effort for any task.

Open source is your best path forward as well, and open source doesn't just mean software. Open source means making things better by opening your work to the world and allowing other people to contribute.

- For example, at The Iron Yard we built a short curriculum that covers the ins-and-outs of freelance and contract work so that our graduates who wanted to follow that path would have a running start.
- Instead of making it proprietary, I open-sourced the project on a site called GitHub so that other developers and project managers could contribute to the project and make it better. There are now over 100 people keeping track of the information and resources.

The reality is that many of us have sold things that we have no idea about. For example, you maybe working on a WordPress site for a client and recommend a plugin that the developer wants to use, but have you ever spun up a WordPress site yourself and tested different plugins? You could do that in less than an hour and the return you'll get from the experience is invaluable.

You have a responsibility to educate yourself, for the sake of your company, your clients and yourself. The days of being able to grow without at least understanding the basics of the tech your team is working on are fading. The good news is that you can dive in at little to no cost and get started right away.

Second, the developers those people work with

For the developers in the crowd:

You have a responsibility to ensure that your staff is technically literate. Who else is going to do that? If you're complaining about other people on the team "not understanding this technology or process," do something about it. If your boss doesn't like the idea of trying a new technology, it's partially your responsibility to educate them well on the merits of the decision you're making.

Fight a tech-buzzword downspiral

One way to pro-actively solve this problem is to punish use of tech vocabulary that people don't understand.

If you use a buzzword and you can't explain what that actually is, you have to put 5 bucks in the jar and everyone drinks on it.

Closing

This is an amazingly exciting time to be in the world of programming. We've just seen the tip of the iceberg as far as what smart, talented problem solvers are going to do with all of the amazing tools available.

In the end, though, the people and companies who invest in educating themselves and participating in the open source community will emerge as the dominant players.