Proactive Programming

PJ Hagerty | DevRelate.io | Humio.com
@aspleenic | @DevRelateIO | @MeetHumio

DevRelate.io

Humio.com

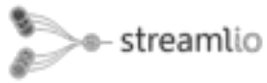
Logging, Monitoring, Observability

Hypi.io

Application Development and Distribution

Ruby Together

Organization dedicated to paying Ruby devs for their OSS work



Streaml.io

Intelligent platform for fast Data.

DevSpotlight.com

Tech content curation, creation, and dissemination.

Tito.io

Ticketing and Event Management for Conferences

Hey You Podcast

An open chat podcast on a variety of topics

CommunityPulse.io

A podcast on Developer and Community Relations

Also…Prompt!

prompt

http://mhprompt.org

team@mhprompt.org

A little about me…

I'm a Developer Advocate, which means I come to speak at conferences, build internal tools, help people with meet ups and stuff like that. Also, I'm a musician, a hockey coach, a dad who lets his kids keynote conferences, and a general people person. I like people, and people are what we are and what makes up the tech community.

“It seems backward, but keeping your mind focused on the present will get you further towards your goals than keeping your mind focused on the goal itself”

Chad Fowler - The Passionate Programmer

Just to let you know, there will be quite a few animated gifs. Prepare yourselves accordingly.

01

Some Definitions

Establishing our terms

Reactive Programming

Reactive programming is a declarative programming paradigm concerned with data streams & the propagation of change.

Reactive programming is the example we are more used to seeing. This is what we see with most modern, non-compiled programming languages, like Ruby, PHP, and Python. The data we wish to manipulate or interact with controls the flow of the program. This means we can make changes to the underlying architecture to see changes reflected in the user interface.

The positives can be numerous, and we will outline those as we move along. There are also a few negatives we need to be aware of.

Proactive Programming

Proactive programming focuses on eliminating problems before they have a chance to appear

Proactive programming is relatively easy to define, if not so easy to execute. The idea is to eliminate problems before they become problems. Sounds simple right? No…not at all. What we are going to focus on in this talk is tools and techniques to make this process easier. That said, we are not talking about some weird psychic ability to determine what needs to be done before it happens. If anyone could do that, none of us would need jobs.

The idea behind proactive programming is pushing the concept of isolating variables that might be easy to mitigate. We will take a look at some of these things as we move on, but for now, we know what the term means, and that's a good start

Feedback Loop

A feedback loop is a term used to refer to a situation where part of the output of a situation is used for new input.

Since this is a DevOps conference, I'm sure we are all familiar with the idea behind the Feedback Loop. It's one of the most important concepts in DevOps - the ability to keep moving forward based on the feedback the process itself has informed us of.

This is the way we understand what we need to do and is a MAJOR part of the proactive concept. We'll look more into the tools on this later in the talk as we know what it is, so it needs a little less attention.

Observability

Observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs

According to Rudolf Kalman who coined the phrase, observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs. Clearly, this definition was developed from an understanding of engineering complex machines where moving parts were internal and their dynamics could only be monitored via extrapolation.

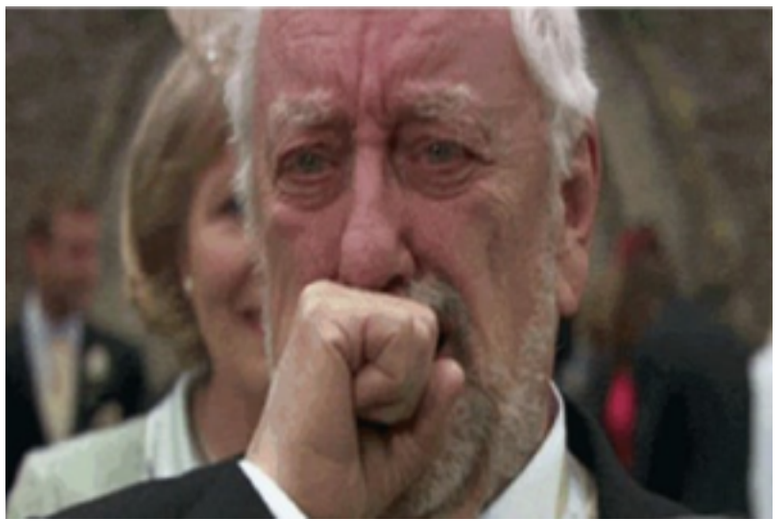
Unfortunately, that definition doesn't apply for modern application development and the infrastructure monitoring necessary to keep the world moving forward in technology. Engineering has experienced a cultural shift that requires live observability and data driven monitoring.

We will take a look at redefining this as we move forward

02

Reactive Programming

Reactive Programming



As we mentioned before, reactive programming is the common model. We build, release, users do something - maybe what we expected, maybe not - and suddenly we need to iterate and rebuild and re-release.

Even with the modern QA cycle, the reactive model is most common. The “get to MVP as quickly as possible” philosophy is mostly at fault here. We live in the land of the VC backed startup - a modern snake oil maybe, or a great, viable business model. That’s a discussion for another time. What we do know is reactive programming is fast and can get us to our goal quickly.

Reactive Programming

“The reactionary is always willing to take a progressive attitude on an issue that is already dead”

Theodore Roosevelt - US President

Reactive programming’s speed is interesting in the fact that it leads to another very Silicon Valley, start up concept: pivoting. If we build something, say an interactive game, but people are really into the chat functionality and not really the game, we pivot. This is how Slack was born.

Many companies work to build a specific vision. The question is raised on how that vision is executed, but also on how the end users and community distort this vision through their own lens. Since we are reacting to their input, there is a necessity to adjust, change, and sometimes completely abandon parts of the vision that may have been laid out initially.

Reactive Programming

In reactive development you solve matters as they arise. This can spark creativity and you can focus on the progress rather than optimizing for millions of users or security threats that aren’t there. When issues come you are expected to have some sleepless nights.

This is where our modern concept of the SRE or Site Reliability Engineer comes in, along with a slew of other support roles and functions. Common ideas such as on-call or pager duty have proliferated due to the inability of reactive programming to be sufficient for the stability of our applications.

Reactive Programming

The problem with reactive development is that most of the control over the development is now outside of the software development or DevOps team. For a simple bit of code with few architectural needs, this user-driven approach has a good chance of succeeding. A fairly verbose user with a good vision on how to really solve their own problems can articulate the interface and data required, leaving the programmers to fill in the blanks. This works so long as the bulk of the programming is primarily business related. It is however, a slow process and the resulting code is disorganized and redundant.

This makes it seem as if reactive programming is NOT quite as fast as we first discussed. And there is the catch. It's a fast first build or MVP, then you are playing catchup with the user. If you users' needs outstrip your ability to keep the code up to date, you will fall behind, lose customers, and eventually experience a slow painful death.

This is not an unusual story. Very few startups have the resilience to build on their mistakes. Heck, very few old corporations are able to move along without somewhat of a pivot. The exceptions, of course, are those organizations so large that there is money in the bank for centuries: these are you Microsofts, your IBMs, your googles. And lest be honest, that is a short, short list.

Reactive Programming

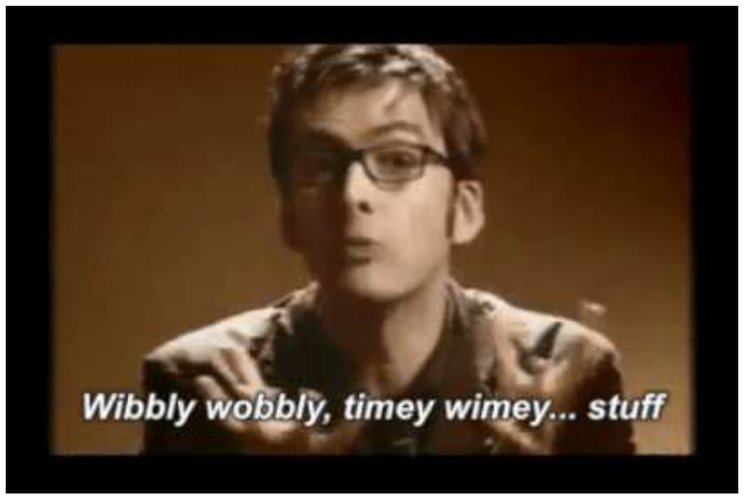
Reactive development approaches have been popular for decades, mostly as an alternative to the failings of big slow long-running projects. When the scope of a project bloats heavily, the various forces involved can accidentally send it off on an unreasonable course. Because the time scales were so long, any misdirection could take a long time to detect and then cost a lot to correct. The reactive idea was that a much larger number of smaller changes driven 'directly' by the users would insure that the users get exactly what they specified. In a very real sense, that's what reactive development achieves.

In short, we can’t just keep doing this because it’s the way we’ve always done it. Reactive programming works in theory, in the same way Agile workflow works in theory. When the rubber meets the road, things change rapidly. That change will determine whether a development team is organized enough to move with the times, or get left behind.

03

Proactive Programming

Proactive Programming

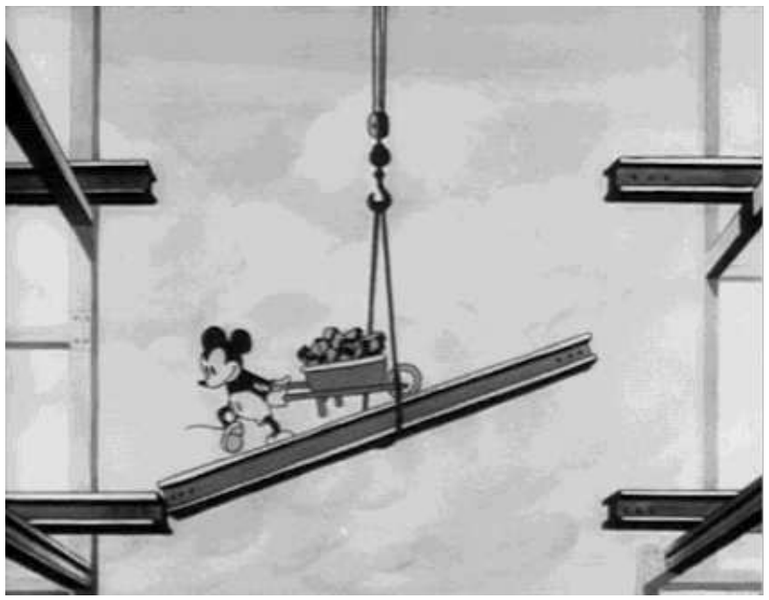


Again, I need to reiterate that Proactive Programming is not being an all-seeing, all-knowing mastermind. It’s nice if you can get the gig, but unlikely any human is qualified for it.

Being proactive means that there is considerable work done first to establish a base for handling the user issues. The initial code doesn't solve problems, rather it sets up the organization necessary to be able to do that in the future.

This doesn’t mean you are giving up the flexibility and iterative nature of reactive programming completely. What it does mean is you will need to build a tool kit to plug holes and patch leaks before they happen.

Proactive Programming



Think of proactive programming like building a house. You start with a strong, solid foundation. Regardless of what happens as things progress, the house will stand and be safe - because you've built it on something that can't be shaken.

In the case of reactive programming, our foundation might be flexible yet far from sturdy. Our idea is built, but it looks more like a pillow fort than a mansion. While it's fun and it looks cool, it's not going to be very helpful when the wind and rain come.

Being Proactive means we are expecting the rain at some point. We may not know when, but we know it's will happen. So we prepare - put in nice double paned windows and dependable roof tiles.

Proactive Programming

In proactive development you solve matters before they become an issue. You generally spend more times on the optimizations (for example, improved security or caching of everything). Proactive development makes developments more stable, but you could anticipate the wrong future and end up spending lots of time on something that isn't important.

This where you hear terms like observability and chaos engineering tossed around. We will talk more about both those things in a bit, but for now, be aware there will need to be schema shift if you are moving from a reactive to a proactive situation. Just like when you abandoned Waterfall for Agile then went all DevOps.

Proactive Programming

When it comes to proactive engineering, the focus is on being as safe and secure as you can possibly be before putting things out into the world. This means having a tight feedback loop.

Feedback loops range from end user feedback to QA teams to CI/CD. We can take a look at some specific tools in a few minutes. And keep in mind as we look at tools - the most important tool is the people you have executing the vision. None of this works if the human factor is made to feel irrelevant.

The key drawback to proactive development is it will take more time in the initial stages. Things can be fast and flexible or sturdy and secure. Pick one and decide what works best in your organization. Chances are security is top of the list, so proactive will be a bigger benefit.

04

The Feedback Loop

Feedback Loop

As we've discussed before, we know the feedback loop is key to what we do. It informs us and allows us to make adjustments and learn from the behaviors of our users, past and present. This means we can work toward proactive development based on our reactive past.

But feedback requires tools. Tools that humans can use. Let's not let our modern star shine too brightly either. Humans have been building tools to make their tasks easier since long before now…actually, since long before humans. That, however is a talk for another time.

Some of the modern tools people are using include monitoring, continuous integration and continuous delivery. QA tools continue to grow as well.

Another piece of the puzzle is the concept of chaos engineering. If you've never heard of it, here's a quick primer.

Feedback Loop

Chaos engineering is the concept of testing systems, live, production systems, at times to see if they are resilient enough to sustain through the chaos. The concept was developed at Netflix and has spread to may organizations since, going so far as companies building chaos engineering as a service around the idea.

What chaos engineering really provides is a way to get a glimpse at what's happening when the worst occurs. Say half your production instances in AWS suddenly fail, or there is a DDOS attack and your site is hit. How well does your system perform? Does your application remain available to most users or is there a complete crash?

Knowing how things will get handled, not only by the application but by your team of engineers, is key to understanding where the flaws and strengths in your application might be.

Feedback Loop

The next key feature in the proactive programming feedback loop is observability. This is the direct feedback from your application, infrastructure, and all other systems that allows you to see how everything is functioning, from sandbox to testing to production.

There was a time when folks would write code, no tests, no QA, and deploy directly into production. That time is past. We are better than that.

Luckily here we have great tools, like Humio, that can see into the system from end to end in real time. But that's not the only tool. Things like Travis CI and GoCD can let you know where the hiccups might be before they hit production. All of this helps complete the loop. But let's look a little further into observability.

05

Observability

Observability

Monitoring methods and tools are now more sophisticated and more widely adopted to handle increasingly distributed and complex systems. While it is now easier to observe code and tests and extrapolate that behavior, it’s also true all bets are off once your code hits the real world.

So for the modern tech world, we’ve seen the need to redefine observability, as we mentioned earlier. For us, observability is the ability for teams to view information and investigate further how a given system is performing in real time, allowing for adjustments and fixes, in order to create better systems and identify threats in any complex computing environment.

Observability

Real-time is critical when it comes to application development and monitoring. Most tools focus on digesting information and kicking it back after a certain period of time. This may not be ideal for many teams, especially those deploying multiple times per week, or even per day. Whether a sandbox environment or a production cluster, systems need to be monitored in concise intervals - near instantaneously.

And we can’t see the full picture without making all aspects of what we are trying to observe readily available. Modern infrastructures generate large amounts of unstructured data but often only sample small parts, due to hardware constraints or high licensing fees. Slow query speeds and long latency between ingest-to-search makes that data not available “fast” enough for quick analysis. And complicated and complex solutions that are not easy to use, query, deploy or manage mean limited usage and pleasure in actually using them.

Observability

This ultimately results in outdated observability for just parts of a system. Removing these obstacles empowers users and teams to quickly and easily query, analyze and visualize all of their data, instantly.

Data-driven observability means that you leverage all of your log data and use real-time streaming capabilities for querying and dashboards to power live system visibility for all engineers, not just Operations or Ops-minded DevOps folks

Observability

Making systems better is the goal of observability. The more we know, the more we are able to improve and adjust. It’s commonly said software development is never “finished”. With this in mind, being able to observe and monitor what is happening out in the world allows our teams to build better, smarter, and get closer to the goals we keep pushing forward.

Observability

From a tooling perspective, this requires handling large data volumes efficiently, and executing ingest and search with sub-second latency, in a simple solution. But observability is about more than just tools.

06

Conclusion

Proactive vs Reactive

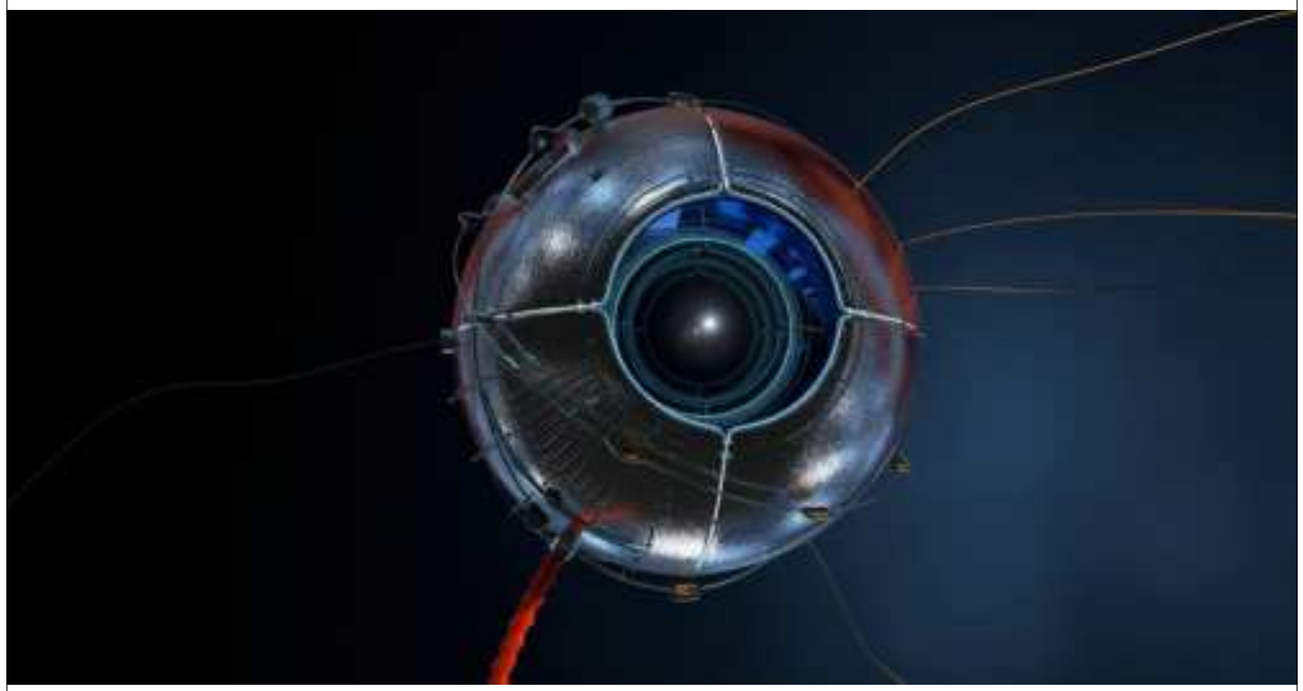
Reactive Programming allows for speed at the cost of stability

Proactive Programming helps mitigate risk

The choice here is up to you and your team. Every organization is different and no single talk at a DevOps Days is going to change the philosophy or workflow for your company.

These are all things to consider as we imagine, create, and distribute software applications into the future.

Observability



Observability is the largest tool in the Feedback loop system to create a proactive environment. Combined with Chaos Engineering, which resides on what I like to call the Observability Tool Belt, there is a way to mitigate and eliminate many issues before they become insurmountable obstacles.

Again, remember, tools are great, but they need to be usable and valuable to the humans who are interacting with them.

“[Observability is] not about logs, metrics, or traces, but about being data driven during debugging and using the feedback to iterate on and improve the product.”

Cindy Sridharan - Distributed Systems Observability

Remember not to get too caught up in tooling. Find a simple thing (not always the best or longest running - Humio is new and easy to use) that allows you to see into your systems and become proactive.

Live system observability is about this data-driven, iterative process for teams that improves the overall health and resiliency of systems.

And mitigating the issue of live system observability is at the top of the heap for every modern company or organization developing an application - be it web or mobile, fintech or funtech. Successful tools need to give developers, DevOps practitioners, security operations, systems administrators, and everyone else insight into how systems are functioning in real time. And they must be built to scale linearly and efficiently store data so users are not wasting their compute resources.

One last thing…

https://www.devopsdays.org/events/2019-buffalo/welcome/

Thank You

PJ Hagerty | DevRelate.io | Humio.com

@aspleenic | @DevRelateIO

pj@devrelate.io