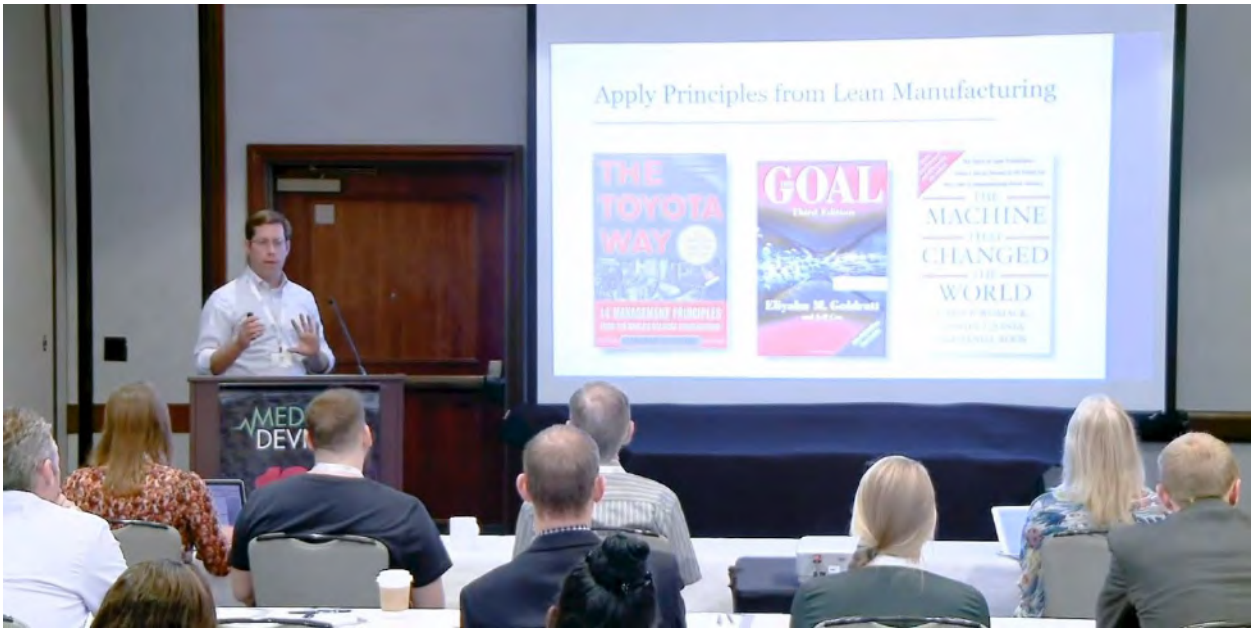

Continuous Integration: Bring Medical Device Software Development Out of the Dark Ages

by Jeff Gable, Founder and Principal, Gable Technology at 10x for Design and Manufacturing

Jeff Gable: Hello, everyone. My name is Jeff Gable and I am embedded software consultant. I help companies develop embedded software for critical applications, like medical devices. I have also done some work for the aerospace and automotive industries.

The topic of my talk today is continuous integration. This is a software development practice that's kind of taken the web development world by storm, and I want to evangelize it and bring it to medical device software development because I think it can help us a lot.

Putting medical devices to the side for a second, let's talk about traditional IT organizations that would make web applications for their internal business processes. You know, circa 2005, you have lots of engineers working on different features, you have these development branches running in parallel. Merging them together is very, very painful so people avoid it. They keep going longer and it gets more and more painful.

There was very much a "throw it over the wall" attitude from the software developers to the IT operations people. You would often see that marketing has set a deadline for this application – to be released on Monday. It gets thrown to the IT operations people at 5pm on a Friday and, "Have fun deploying this over the weekend."

There were a lot of manual error-prone developments, probably poorly written instructions, and people didn't have much of a process how to get applications to deployment.

The Results
That results in what you think it would: very low-quality software. The iteration time was measured in months. A lot of these applications are on a release schedule where every nine months they come out with a new version of the software. That means that if someone in marketing or the business side has an idea that they want to test with the customer, they must wait months to get an answer.

Arlen was talking yesterday about getting answers: He wanted to do simulations to get answers faster on the technical side. Likewise, on the business side, if you have a hypothesis you want to test with the customer, you want to get answers quickly. This very slow, high-latency software development process just kills that effort.

There are a lot of people who've done work in the manufacturing industry on speeding up the manufacturing process and reducing the latency of the manufacturing process. You can respond to the market needs more quickly.

If there are any MBAs in the room, you may have heard of these books. There's the Toyota Way. Toyota was a company that really espoused lean manufacturing methods and developed a lot of groundbreaking efforts.

There was the Theory of Constraints. These principles from lean manufacturing have been applied to great effect in the manufacturing world to increase the responsiveness of the manufacturing process to the market, and some very smart people looked at that and said, “Hey, why don't we start applying these principles to software development?”

This revolution that came out, you may have heard of the term DevOps – that's development and operations smashed together.

I've heard a lot over the course of this conference about breaking down silos, trying to get people from regulatory and development to work together. One of the tenants of systems engineering, as Martin said, is getting these different stakeholders in the room and having them all contribute to the process. That's the core of this DevOps revolution that has taken the web development world by storm.

The Three Principles
One is to apply systems level thinking, and the system I'm talking about is the business system. You have ideas that for the business that you want to test in the marketplace. And you want to reduce the amount of time that it takes for that idea to get in front of a customer, so that you can validate that hypothesis.

Second is to improve the feedback loops. So again, getting it to the customer. You can test that hypothesis, getting that feedback back up to the business, but also various stages in the process that there any defects that are passed down stream, you want immediately to, to flow back upstream that information to fix the process and not allow this to continue.

Third is to foster a culture of continuous improvement. This last one is doing things like prioritizing the flow of daily work almost as much as the daily work. A lot of people think, we're

too busy, we're late on our projects, we're behind, we don't have time to invest in making our process better.

There are some good rules of thumb in the software world, that you invest 20% of your effort in every development cycle to improving your process. So that's kind of a good metric to follow.

That way 80% of your time is spent getting work done, so you're not, you know, completely ignoring that. But every development cycle, you're investing 20% your effort into improving your process.

Let's get concrete.

At least from the software development and operations side, when a software developer has written code for some feature, and they commit it to version control, there are a lot of steps to happen before that's deployed to the customer.

You may need to provision a server, they need to install a bunch of software packages, they need to configure the network settings, whatever it is.

There needs to be a unit test or acceptance tests. It needs to go to staging for a little while for testing, and then go to the production servers.

These different steps that go there and again, back in the day most of them were manual and error prone.

What people have started to do is just automate that. All of it. And it's surprising how much of the process you can automate.

Particularly you might think that something like provisioning a server is hard to automate.

Well, nowadays, there are software tools that people have developed as part of this effort where you describe your infrastructure as code, you have human readable text configuration files that say, “I want this server and I want it to have these libraries, and they better have this minimum version.”

And nowadays, with all the servers being in the cloud, you almost must do it that way. These configuration files that describe your infrastructure, again, they're stored in version control with your application code.

You really can't say, “I have an application and there's some dependencies such that if they're not there, the application doesn't run right, so I'm going to have some written instructions…” No, those should actually be stored in text right along with the source files for code. What this gives you is repeatable, guaranteed deployments of applications.

As a result, what you get is high quality software, and if you can automate all this, your iteration time can be measured in hours. Going back a few years at a typical enterprise, you would have this deployment frequency of every nine months, you would get a new version of software.

If there were an urgent business need, you had to go through all of those manual steps over months. Because you had this very slow feedback time, you had low quality, low responsiveness, low reliability. You couldn't get quickly get fixes to the application if you discover them in the field.

This chart is from a book I'll show you later called the Phoenix project. But this was, I think in 2012, Amazon was doing 23,000 deployments a day. What that really means is that each deployment is extremely small. I see a bug, I fixed it, I kick off this process, and a few minutes later, it's done, it's live, it's going to users.

Now, in order to do that, you must have really, really solid testing. But that's part of your responsibility as engineers, is to make sure that all the testing that would you would do to do your due diligence to make sure that your code is okay to go in front of customers, that all of that is built into this automated process.

You can imagine what that does for the business in terms of enabling new experiments to be run. A marketing person come up with an idea and you can have code in front of customers the very next day testing that out to see if it works. It really changes the business side of things, speeding up software deployment so quickly, and it's now the industry standard in web development. If you're developing a web application, and you're not doing this, you are way behind the times and you're going to go out of business because you can't keep up.

I'm on a mission to take these lessons and apply them to software development for medical devices. I want to evangelize how we can apply these lessons to solve a real demand for medical devices, because I've rarely seen it done.

Prerequisites for continuous integration

One, your software has to be in version control. If you're not version controlling your software, you're in big trouble and you've got bigger problems, so please come see me afterwards.

All version control is not alike. There are more modern version control systems that are much less painful to work with for developers. Git is the most popular one in the world right now. It's been popularized by sites like GitHub.

The second prerequisite is that your build process – building your executables that actually get flashed onto your device – that build process has to be fully automated.

You can't be working in an IDE (an integrated development environment) where someone uses their mouse pointer to click build and then configuring things… No, you type in a command and it goes and does the rest. Once you have that, you can start having little robots on the internet do that for you all the time. And it's wonderful.

The next thing you do is to start adding automated testing. There's a couple of different levels of this. There are unit tests, where you're testing low level functions of your code, and you can run them on host computer rather than on the target device that maybe is running a different processor with a different architecture.

I've heard a lot of people complain, essentially, no, no, my code runs on this device only. It's dependent on the hardware and it’s just not unit testable.

To which I say: bullshit. It can be done. It is worth the effort and going through that exercise really forces you to have good modular software architecture.

I’ll show you the book “Test-Driven Development for Embedded C” later. It walks you the process. I don't know that many people in this room are going to go through that exercise themselves, but if you hear these excuses from your software engineers, have them call me.

Then, there are integration tests, which are maybe more common these days. This is where you're running code on the actual device.

You can still apply a fair level of automation to that, especially if you build in scripting and data instrumentation capability into your embedded code.

At that point, you can have something running on the host, communicating with the device, saying, “Run this command and give me the data back,” and you can verify that. It gets to be a little messier in the real world because hardware sometimes fails. It's not as predictable, but it can be done, and you can create special test images of the software with very limited capability to exercise the hardware in a certain way.

We see, again, that principle of infrastructure as code. It’s a very common problem in software development for different engineers to say, “Well, it builds on my machine, I don't know why it's not building on yours.”

There again, you can create your build and development environments by describing them with configuration files. These could create a virtual machine that would have the compiler, the flashing tool, and probably your development environment with your debugger, and everything else. I've been working this way for five or six years now, where I do my software development in virtual machine, and it's fast enough to where it works.

At that point, you're guaranteed that everyone who's working on this project has the same development and build environment, and then you reuse that on your build server. This automated build that takes place uses that same configuration.

Putting this into practice

Now, this next part may be getting a little too in the weeds, but at least I'll give you an overview. If you're in an organization, and you say, “Hey, that sounds great. I actually want to get started with continuous integration.”

First, you buy build server, and don't skimp. You make it really, really fast. Lots of RAM, lots of processor speed.

There is continuous integration software out there. For open source options, there is one called Jenkins, and there's another good option called GitLab. There are several others that are paid.

The very first thing is to have it run your builds automatically, every single time you check into version control, and email you if there's a failure. All of these tools come with nice, pretty dashboards, which show you the latest status. Then you slowly, over time, invest 20% of your effort into adding tests, defining more of your infrastructure, and just improving everything that you're capturing there.

This is an example of a dashboard of a continuous integration build. I stole this from someone's talk on the web ["Microcontrollers in Micro-increments" by Mike Ritchie, CppCon 2017].

This is checking out the latest version of code, and then you have different unit tests and you're running it with the debug configuration and the release configuration. If any one of these steps fails, it immediately stops the build and lets you know. Your highest priority is to fix that issue, and only then move on. This way, your code is always in a working state.

And you can see that there's a lot of things you can put in here to do very robust checks of your software, and know that if you check in software, and it passes all these tests and you've done your due diligence, and you've done this properly, by definition, then it is suitable for deployment.

At this point, I have an open question for the group.

Medical devices are not web applications. We are highly regulated. The FDA is an important part of the deployment chain. So, an open question is: How fast can you make that process?

This is all predicated on the fact you did your due diligence and you're comfortable. I've made a software change. I've tested it and analyzed it well enough so that if my spouse or my child were using this medical device, I would be comfortable with the software change.

At that point, and only once you satisfy that condition, what's the fastest you can deploy to your device? If that involves the FDA, what are the steps you could do? Again, it's an open question for the audience. What can you do to speed that process up?

Walt McClay: So, not all deployments require FDA approval. If it's a simple deployment, you can deploy it immediately. Of course, if your device is somewhere out in the field, you have to find it and update it.

Maren Nelson: I'm Maren Nelson, and I have done a little bit of medical device software in my past. As Walt mentioned, there is a straightforward assessment you can do on the software, even before you get to the point of deployment, to know whether you've got to bring the FDA into that or not.

The other part of the deployment chain is the end user. Because if you're making a change that impacts user interface or anything like that, you may have training and other things to address. So, I think it's important to keep the user as part of the deployment chain as well.

Jeff Gable: Absolutely. I wanted to bring up the FDA part, because inherent in that is the assumption that you have done all the analysis that that you were doing the right thing and that you've checked all the boxes for the right reasons, not just to check boxes for compliance.

Walt, you mentioned updates over-the-air updates.

These days, web applications have continuous delivery. The engineer checks in code, and a few minutes later it is deployed live to real users. Maybe it's in a staged fashion where they're looking for problems and would automatically roll back if there were problems. But instead of continuous integration, that's the next logical evolution. That's continuous delivery.

I don't think that's feasible for medical devices. I'm not suggesting that it is, but you can certainly greatly increase the cadence, the frequency of your releases. What can that enable?

Tesla does a lot of things that I disagree with, but Tesla has shown the automotive world what's possible in terms of, they just push something to your car and now your shocks ride differently, or they've solved an issue in the field.

They are getting reports of a problem and a week later, they have pushed a fix and it is running on everyone's cars.

So again, just use that as inspiration to spark your imagination for what's possible and whether that enables you to work differently, to test business hypotheses in a different way that maybe was not feasible before. All that requires the engineering of over-the-air update. But it's doable, and people are doing it a lot.

Joe Hage: What about legacy systems? I mean, it would be ideal to be able to change everything everywhere. But we've had a system in the field for the last 10-15 years. Does it mean we'll just have to wait till the next generation of product? Because it's just too much to reverse and …

Jeff Gable: A lot of those have no means for updating. Some of them do, but you have to ship USB sticks to your customers. So, what I'm what I'm advocating for is a new way of engineering new products.

Christine Zomorodian: This is happening. There are legacy products in the field for large device manufacturers are already receiving remote service. This is already happening. They do it usually through the hospital it. They're connected to the network.

There are huge issues around cyber-security. Joe and I were present for a presentation in Washington on this topic where we had the manufacturing side and the hospital institution side.

It's happening though, because these guys, especially in imaging systems, they have to get the software out there and they can't afford of when servicing it to deliver a USB stick and they can't hand it to anybody else. It's already happening.

Jeff Gable: Sure, it's just very painful.

You deal with legacy systems the best that you can. What I'm advocating is, for your new products, use this process. Even if you don't worry about the deployment, the increasing the

cadence of release, using continuous integration during the development, until release, speeds up the process so much.

It's breaking down the silos, like in the web world between development and operations, and in the medical device world, it's been between development and QA.

So, involve your QA people's creativity to automate these tests. How can we describe in computer code how you are testing this device, and let's code that up and then we can automate it.

You don't have to run the same manual test every time you do a release. Tests can get run automatically. It's definitely useful during development.

Ryan Carpenter: I thought I'd point out, it's not only software, I'm seeing several hearing aid companies delivering firmware updates via apps.

And, well, I know the old way too; I worked with a product manager in Sydney that at the time, it was shipping CDs for updating firmware.

At any rate, I'm seeing that happen, where several new manufacturers are delivering firmware updates over the air through apps. Many of the legacy devices don't have the same full capability as the most current wireless hearing aids, but they do have older wireless technology, so some of them will use a streaming device to connect to a phone for example.

Jeff Gable: I would say this about over-the-air updates. So, to anyone who is selling a connected medical device.

FDA has issued guidances on post-market cyber security. A lot of those connected medical devices are using open source libraries. Security researchers are pounding those libraries all the time and finding new vulnerabilities.

If you don't have a list of what libraries are on your device, and you're not monitoring the central worldwide database for new vulnerabilities, and you're not getting latest versions, and then patching your device… It's the reason you get updates on your iPhone and your Android phone all the time.

They are constantly finding and fixing new vulnerabilities.

People are still designing medical devices with no way to patch, and so now you have medical devices on hospital networks that are completely vulnerable. It used to be, well, if it's on the hospital network, it's the hospital's problem. The FDA is starting to change their tune, so they expect you to be able to patch for security reasons.

Joe Hage: I know this is a terrific layup for you. But let's say they employed you as their consultant. What are you going to do about it?

Jeff Gable: My role is, I help people develop embedded software better. I have two sides to my consultancy – I do coaching where I help clients improve their process, and also help them early in the product design process, to flush out their development plan.

I bring the technical side to systems engineering, where I have enough knowledge of all the different engineering areas to be able to intelligently spot problems.

Then, on the development side, I specialize in firmware development and control algorithms. Not as much the cyber security, but I know enough to be to be dangerous, and at least spot the problems.

Resources
I'll close by adding, if your software engineers are giving you the excuse that they can't do unit testing on their embedded hardware, give Test-Driven Development for Embedded C by James Grenning. It's wonderful.

The Phoenix Project is a business novel that describes the revolution of DevOps. Like most business novels, it's cheesy, but it is a little bit easier to read than, say, this one [The DevOps Handbook] which is the same information presented in textbook form.

Even not working on web applications, working on firmware, these two books were really eye opening for me, showing me this whole world of capability that's possible.

Thank you!