Code rant

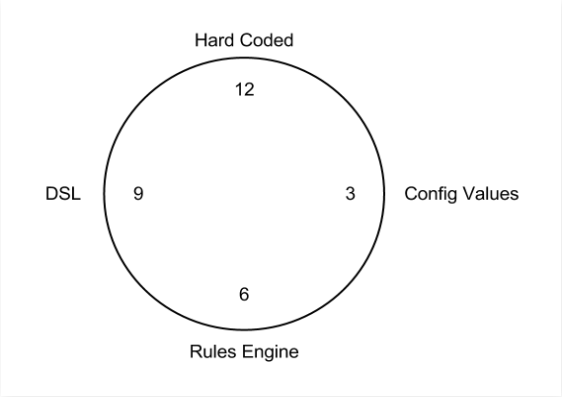
Life as a mort.

Monday, May 07, 2012

The Configuration Complexity Clock

When I was a young coder, just starting out in the big scary world of enterprise software, an older, far more experienced chap gave me a stern warning about hard coding values in my software. “They *will* have to change at some point, and you don’t want to recompile and redeploy your application just to change the VAT tax rate.” I took this advice to heart and soon every value that my application needed was loaded from a huge .ini file. I still think it’s good advice, but be warned, like most things in software, it’s good advice *up to a point*. Beyond that point lies pain.

Let me introduce you to my ‘Configuration Complexity Clock’.



This clock tells a story. We start at midnight, 12 o’clock, with a simple new requirement which we quickly code up as a little application. It’s not expected to last very long, just a stop-gap in some larger strategic scheme, so we’ve hard-coded all the application’s values. Months pass, the application becomes widely used, but there’s a problem, some of the business values change, so we find ourselves rebuilding and re-deploying it just to change a few numbers. This is obviously wrong. The solution is simple, we’ll move those values out into a configuration file, maybe some appsettings in our App.config. Now we’re at 2 on the clock.

Time passes and our application is now somewhat entrenched in our organisation. The business continues to evolve and as it does, more values are moved to our configuration file. Now appsettings are no longer sufficient, we have groups of values and hierarchies of values. If we’re good, by now we will have moved our configuration into a dedicated XML schema that gets de-serialized into a configuration model. If we’re not so good we might have shoe-horned repeated and multi-dimensional values into some strange tilda and pipe separated strings. Now we’re at 4 or 5 on the clock.

More time passes, the irritating ‘chief software architect’ has been sacked and our little application is now core to our organisation. The business rules become more complex and so does our configuration. In fact there’s now a considerable learning curve before a new hire can successfully carry out a deployment. One of our new hires is a very clever chap, he’s seen this situation before. “What we need is a business rules engine” he declares. Now this looks promising. The configuration moves from its XML file into a database and has its own specialised GUI. Initially there was hope that non-technical business users would be able to use the GUI to configure the application, but that turned out to be a false hope; the mapping of business rules into the engine requires a level of expertise that only some members of the development team possess. We’re now at 6 on the clock.

Frustratingly there are still some business requirements that can't be configured using the new rules engine. Some logical conditions simply aren't configurable using its GUI, and so the application has to be re-coded and re-deployed for some scenarios. Help is at hand, someone on the team reads Ayende's DSLs book. Yes, a DSL will allow us to write arbitrarily complex rules and solve all our problems. The team stops work for several months to implement the DSL. It's a considerable technical accomplishment when it's completed and everyone takes a well earned break. Surely this will mean the end of arbitrary hard-coded business logic? It's now 9am on the clock.

Amazingly it works. Several months go by without any changes being needed in the core application. The team spend most of their time writing code in the new DSL. After some embarrassing episodes, they now go through a complete release cycle before deploying any new DSL code. The DSL text files are version controlled and each release goes through regression testing before being deployed. Debugging the DSL code is difficult, there's little tooling support, they simply don't have the resources to build an IDE or a ReSharper for their new little language. As the DSL code gets more complex they also start to miss being able to write object-oriented software. Some of the team have started to work on a unit testing framework in their spare time.

In the pub after work someone quips, "we're back where we started four years ago, hard coding everything, except now in a much crappier language."

They've gone around the clock and are back at 12.

Why tell this story? To be honest, I've never seen an organisation go all the way around the clock, but I've seen plenty that have got to 5, 6, or 7 and feel considerable pain. My point is this:

At a certain level of complexity, hard-coding a solution may be the least evil option.

You already have a general purpose programming language, before you go down the route of building a business rules engine or a DSL, or even if your configuration passes a certain level of complexity, consider that with a slicker build-test-deploy cycle, it might be far simpler just to hard code it.

As you go clockwise around the clock, the technical implementation becomes successively more complex. Building a good rules engine is hard, writing a DSL is harder still. Each extra hour you travel clockwise will lead to more complex software with more bugs and a harder learning curve for any new hires. The more complex the configuration, the more control and testing it will need before deployment. Soon enough you'll find that there's little difference in the length of time it takes between changing a line of code and changing a line of configuration. Rather than a commonly available skill, such as coding C#, you find that your organisation relies on a very rare skill: understanding your rules engine or DSL.

I'm not saying that it's never appropriate to implement complex configuration, a rules-engine or a DSL, Indeed I would jump at the chance of building a DSL given the right requirements, but I *am* saying that you should understand the implications and recognise where you are on the clock before you go down that route.

Mike Hadlow at 12:24 pm

17 comments:

Richard OD 8:54 am

Great post Mike- oddly I saw it as a link in Google+ from Mark Seemann. Then I thought Code Rant- that sounds familiar. I then woke up. :-)

Reply

Bartosz Adamczewski 2:26 pm

Awesome post :-), it supposed to be called "THE DOOM CLOCK" ;-)

Reply

Marc Ziman 11:26 am

Mike - great post. I think it's dangerous to change config values on the fly. People can get it wrong and will bring down the app. So hardcode the values if you like or put it in a config file or whatever else - just make sure that you treat the change as any code change and run your full suite of tests before you deploy it. If your build and deploy process is solid it is no extra overhead to retest and deploy.

Reply

HolyTshirT 2:18 pm

Great post! Racing around the clock because you fear recompiling and redeploying, making this easy from the beginning, makes most of this problem go away.

Reply

majkinetor 10:40 pm

Nice. Its probably the best to develop some nice devops solution in order to make deployment trivial and hard code things.

You need easy deployments anyway.

Reply

Steve M 6:42 pm

I love it. This is very similar to a pattern I've been thinking about for a while -- call it the Configuration Complexity Pendulum.

Year 0: Everything is done in code.

Year 1: Clever Programmer A realizes that he's writing the same code over and over. He parameterizes the code and moves the parameters into a configuration model.

Year 2: Requirements become more complex. Configuration model becomes more complex, and starts incorporating conditionals, exceptions, triggers...

Year 3: Clever Programmer B realizes that the XML has become a programming language in its own right -- a terrible one that takes 20 lines to say IF A THEN B. She starts writing an API or a DSL...

And thus, we swing back and forth between code and config, forever.

Reply

Unknown 12:18 pm

Great post and I recognise this pattern. These days I use Octopus deploy as part of a one-click deployment pipeline and it does help minimise the pain when you need to either update config or have hard coded values in your app.

Reply

Christopher Hoover 5:51 am

Perhaps if you use Common Lisp, the circle collapses to a point.

Reply

Graham Poulter 10:40 am

The example seems to be assuming a single deployment - one binary running in one place, and the configured variables vary over *time* but not over *space*.

However, if the packaged binary runs in many places, some values vary depending on where it is deployed: at a minimum things like what datacenter, what user, what backends to connect to, what paths to read from, as well as anything else that varies by user.

While "temporal" variations can easily be hardcoded if you have a short release cycle, "spatial" variations are not so easily hardcoded: you end up maintaining a source branch for each active variant.

And then what do you do if the spatial variations are enormous: a huge amount of information that differs between deployment?

Then you might consider hard-coding the spatially varying part separate from main application which does not vary: write a separate piece of code that generates a structured data format with all the config information, which the main application slurps in.

I suggest having three marks on the clock for such data:
12: nothing varies spatially, everything in main code
3: separate a few flags varying spatially (--input_path=/somewhere)
6: structured data format for a larger number of spatially-varying parameters (protobuf, INI, JSON, XML).
9: DSL, to generate the structured data. Skip this if at all possible!
12: Full-fledged programming language to generated the structured data.

For the things that vary temporally (having the same value in all deployments), now you have a choice of whether to hard-code it in the main application, or move it in to the config data, based on whether it changes more or less frequently than the release frequency of the main application.

Reply

Jonathan Gilbert 9:01 pm

On UNIX systems, configuration is often done in the form of a script that is technically capable of arbitrary programming complexity but whose base case *looks* declarative.

For instance, on one of my FreeBSD servers, /etc/rc.conf starts out with the following lines:

hostname="laliari.logiclrd.cx"

defaultrouter="184.71.82.245"
ifconfig_vtnet0="inet 10.0.0.1 netmask 255.128.0.0"
ifconfig_vtnet1="inet 184.71.82.246 netmask 255.255.255.252"

gateway_enable="YES"
natd_enable="YES"
natd_interface="vtnet1"
natd_flags=`/root/make_natd_flags/make_natd_flags`

(I'd have used a PRE tag for that but blogger.com doesn't allow it!)

Everything there looks pretty straightforward -- name equals value, repeat as necessary -- except, whoa, what's going on in that last line? I am actually running an external program to generate the flags to pass to natd.

This type of configuration, leveraging a common, system-wide scripting facility, means your users can do simple configuration without ever knowing the system isn't purely declarative, and more complicated configuration can use arbitrary programming techniques. In this example, make_natd_flags is actually a small C application, but it could be absolutely any command-line, or could even be implemented in shell script right inside the rc.conf file.

This mode of configuration has been the de facto standard for a *very* long time, and doesn't seem to be progressing around the clock. Where would you say the clock is stopped for this type of configuration? I find I can't really assign it a position on the clock; it seems to combine most of the advantages of most of the "times" you've described, without suffering unduly from the downsides. The only real problem I see with it is that a user who is terribly inexperienced, or who suffers greatly when it comes to attention to detail, is at greater risk of producing a configuration file with syntax errors, even if the syntax is as simple as 'name="value"'...

Reply

RMKnightStar 2:04 am

This struck nerve with me. I have been in technology for over 35 years and I have to say that I have seen this clock cycle, time after time after time. Though I really had never seen it illustrated like this. When ever I am thinking about a new system, the line between configuration and code generally gets a lot of thought. This is really good food for thought!

Reply

Anonymous 5:09 am

Awesome post !! definitely could relate to it to some of the projects i had been working on now.

Reply

pt 2:41 pm

You left out the step where the config values are put into a database. The boss likes Lotus Notes despite the fact it isn't relational and isn't really a database, so that's where config values go. Nine months later he's gone and the team takes the opportunity to move the data to Oracle, except the two people who quit because they hate Oracle. Nine months later Oracle raises prices so the dB gets migrated to SQL Server. Another developer quits because he hates Microsoft.

Reply

Scott Cooper 8:46 pm

Reminds me of Ivan Sutherland's Wheel of Reincarnation: http://cva.stanford.edu/classes/cs99s/papers/myer-sutherland-design-of-display-processors.pdf

Reply

Xeno PuTtSs 9:13 pm

Just want to say, its the year 2020 now and I am still using this document almost 3 times a year to show this same pattern. This is one of those things that hasn't yet changed.

Reply

Dan Nugent 4:56 pm

I think there's a way out: Use sqlite for storing configurable values. It's a self-contained file format, it has a way to dump to a reviewable, versionable text representation for tracking, it's fast enough to access online so you can modify things during runtime, it's got a lot of tooling support, it uses a real language, it's extensible, there are specific rules for how to migrate data and if you adhere to the generally advised notion of never directly accessing data through tables and only using views, straightforward migrations of the configurations are possible.

If there's a significant drawback it's that the types are a bit too flexible, but not much worse than many other text formats.

Reply

geeklinda 1:15 am

Oh no, it me :(:(

Reply

To leave a comment, click the button below to sign in with Google.

SIGN IN WITH GOOGLE

Note: only a member of this blog may post a comment.