Taming Complexity with Reversibility

Kent Beck, 28 de julio de 2015
https://m.facebook.com/notes/kent-beck/taming-complexity-with-reversibility/1000330413333156

In 2002, Professor Enrico Zaninotto, Dean of Economics at the University of Trento, gave a keynote at the Extreme Programming conference. It was the clearest technical talk I have ever seen, even though it was delivered by a non-programmer in an unfamiliar language. What set the talk apart was the clarity and depth of the thought behind it.

The goal of the talk was to explain Extreme Programming in economic terms. I believed him at the time, but I can't say I took his message fully to heart. It has taken me thirteen years to digest what he said. Now I can apply it to explain Facebook's engineering culture, which by Prof. Zanninotto's standards is beyond Extreme Programming.

Complexity

As a system scales, whether it is a manufacturing plant or a service like ours, the enemy is complexity. If you don't confront complexity in some way, it will eat you. However, complexity isn't a blob monster, it has four distinct heads.

- *States*. When there are many elements in the system and each can be in one of a large number of states, then figuring out what is going on and what you should do about it grows impossible.
- *Interdependencies*. When each element in the system can affect each other element in unpredictable ways, it's easy to induce harmonics and other non-linear responses, driving the system out of control.
- *Uncertainty*. When outside stresses on the system are unpredictable, the system never settles down to an equilibrium.
- *Irreversibility*. When the effects of decisions can't be predicted and they can't be easily undone, decisions grow prohibitively expensive.

If you have big ambitions but don't address any of these factors, scale will wreck your system at some point. One of the heads of the Complexity Beast must be chopped off and the others controlled to some degree. For example, Henry Ford chose States.

Henry Ford -- Statemeister

Henry Ford had a big dream--realize the potential of the "automobile" (literally: self transportation) by getting everyone into a car. Massive economies of scale were required to reduce the cost of the car. Scale spawned the Compexity Beast.

Ford chose to tame complexity by severing the States Head. Each station in the factory would do exactly one thing and the stations would be joined in a rigid assembly line. Since each station was in only one of a few states and the relationship between the stations was fixed, the system as a whole could only be in a tractable number of states. Several techniques limited the number of states. Foremost among them are:

- Standard products. "Any color as long as it's black" was not a statement of arrogance, it was necessary to get any cars off the assembly line at all.
- Standard parts. The fewer kinds of parts you require, the fewer states each station can be in. "I'm out of bolts" is much simpler to manage than "I am running low on 3/8ths bolts but I have too many 5/16ths".
- Division of labor. When each person does exactly one job, their only states are "done" and "not done".
- Assembly line. Overall, either the line is running or it is not.

Reducing the number of possible states made mass manufacturing tractable for Ford. Over time he reduced the number of states further by standardizing more parts and refining the division of labor. As a result, the price of a Model T dropped by almost two thirds over its life.

The cost of all this state slaying was rigidity. If any station along the line wasn't working, either the line didn't work or someone had to make costly repairs downstream. Because each feature of a car was deeply entwined with both all the other features and with the design of the assembly line itself, making changes was increasingly expensive as the number of states dropped. As soon as the market for cars was driven by something other than price, Ford had a hard time competing.

Facebook: Irreversibility Slayer

From the beginning of Facebook, no one could predict how it would appear a few years hence. No one from Planet Earth could have predicted how fast it would grow. The mission was always to connect the world, but how and when were a mystery. Which head could we sever?

Cutting off the Uncertainty Head was not an option. Because very few computer systems have ever been built at Facebook scale, it is impossible to cut off the Interdependencies Head a priori. As surprising things go wrong, Facebook engineering gives Interdependencies a shave and a haircut from time to time, but surprising interdependencies, good and bad, were and are daily business. The States Head is inevitable, whether measured by the states of servers, disks, and networks or by the activities of the people responsible for changing and running the service. We can prune excessive states (for example, we automatically take unresponsive servers out of service) but this head is just going to be there.

That leaves Reversibility as the only vulnerable head. Reversibility is absurd in the Fordist assembly line world but it's at least possible for computer systems. Reversibility appears all over our development culture and tooling:

- Development servers. Each engineer has their own copy of the entire site. Engineers can make a change, see the consequences, and reverse the change in seconds without affecting anyone else.
- Code review. Engineers can propose a change, get feedback, and improve or abandon it in minutes or hours, all before affecting any people using Facebook.
- Internal usage. Engineers can make a change, get feedback from thousands of employees using the change, and roll it back in an hour.
- Staged rollout. We can begin deploying a change to a billion people and, if the metrics tank, take it back before problems affect most people using Facebook.
- Dynamic configuration. If an engineer has planned for it in the code, we can turn off an offending feature in production in seconds. Alternatively, we can dial features up and down in tiny increments (i.e. only 0.1% of people see the feature) to discover and avoid non-linear effects.
- Correlation. Our correlation tools let us easily see the unexpected consequences of features so we know to turn them off even when those consequences aren't obvious.
- IRC. We can roll out features potentially affecting our ability to communicate internally via Facebook because we have uncorrelated communication channels like IRC and phones.
- Right hand side units. We can add a little bit of functionality to the website and turn it on and off in seconds, all without interfering with people's primary interaction with NewsFeed.
- Shadow production. We can experiment with new services under real load, from a tiny trickle to the whole flood, without affecting production.
- Frequent pushes. Reversing some changes require a code change. On the website we never more than eight hours from the next schedule code push (minutes if a fix is urgent and you are willing to compensate Release Engineering). The time frame for code reversibility on the mobile applications is longer, but the downward trend is clear from six weeks to four to (currently) two.
- Data-informed decisions. (Thanks to Dave Cleal) Data-informed decisions are inherently reversible (with the exceptions noted below). "We expect this feature to affect this metric. If it doesn't, it's gone."
- Advance countries. We can roll a feature out to a whole country, generate accurate feedback, and roll it back without affecting most of the people using Facebook.
- Soft launches. When we roll out a feature or application with a minimum of fanfare it can be pulled back with a minimum of public attention.

- Double write/bulk migrate/double read. Even as fundamental a decision as storage format is reversible if we follow this format: start writing all new data to the new data store, migrate all the old data, then start reading from the new data store in parallel with the old.

Exceptions

Not all decisions are reversible. Decisions affecting people's privacy, trust, and money; and decisions affecting the Facebook brand aren't reversible. Such decisions need to be evaluated up front to the degree possible. Learning to distinguish reversible and irreversible decisions early is a key skill, as the two kinds of decisions need to follow completely different workflows.

So What?

Knowing that we rely on reversibility to keep running and growing, we can predict activities that will create value: defending the reversibility we have and increasing reversibility where it doesn't yet exist. Some examples:

- Project management. If a project is one big, irreversible decision, the project will be more valuable if sliced into a sequence of reversible decisions.
- Development tools. If we can detect the need to reverse a decision automatically, that's a win. If we can automate reversing the decision, that's a double win.
- Deployment. More frequent deployments enhance reversibility. We need to reduce the cost per deployment, but, in general, reversibility trumps cost.
- Programming style. Converting "code push reversibility" into "dynamic configuration reversibility" creates value. Putting in runtime reversibility makes sense where it is reasonably cheap. The design of the system can be refined to encourage runtime reversibility.

Conclusion

If you intend to scale, you are going to have to draw your sword and sever one of the heads of the Complexity Beast. To attack the Reversibility Head, list decisions you regret. What changes--technical, organizational, or business--would you have to make to identify such decisions earlier and make reversing them routine?