We fired our top talent. Best decision we ever made.

medium.com/free-code-camp/we-fired-our-top-talent-best-decision-we-ever-made-4c0a99728fde

1 de octubre de 2018

"You will never be able to understand any of what I've created. I am Albert F***ing Einstein and you are all monkeys scrabbling in the dirt."

And so our resident genius, our Dr. Jekyll, explosively completed his transformation into Mr. Hyde.

He declared this in front of the product design team, developers, management, and pre-launch customers. One of our project sponsors had the temerity to ask when the problem crippling our product would be fixed.

Genius is a fickle beast. Sometimes you have the good fortune to work with a mad genius. Other times you are doomed to work with pure madness. There are also times when it is hard to tell the difference.

This story is about the fall from grace of an extremely gifted team member with a deep understanding of our product's architecture. He had an uncanny ability to forecast future requirements, and a ton of domain-specific knowledge.

He was our top contributor. He was killing our flagship project.

We'll call this person "Rick."

You don't want this guy on your team. (image © Warner Bros.)

Rick was universally recognized on the team as the top talent. He was the lead developer and architect of our software projects.

Any time anyone had a question about code or needed help with a task, they would go to Rick. Rick had a giant whiteboard installed in his office used only for this purpose. It was always cluttered with the ghosts of past discussions that wouldn't quite erase.

Any time there was a particularly challenging problem, Rick would handle it. Rick had a server with the same specs as our production server installed at his desk. He used this to run the entire application stack independently and troubleshoot every layer at once.

Rick didn't need anybody else. Rick preferred to work alone in his private work-space.

Rick didn't need anything anybody else built. He built everything he needed from scratch because it was infinitely better than the paltry offerings of mere mortals.

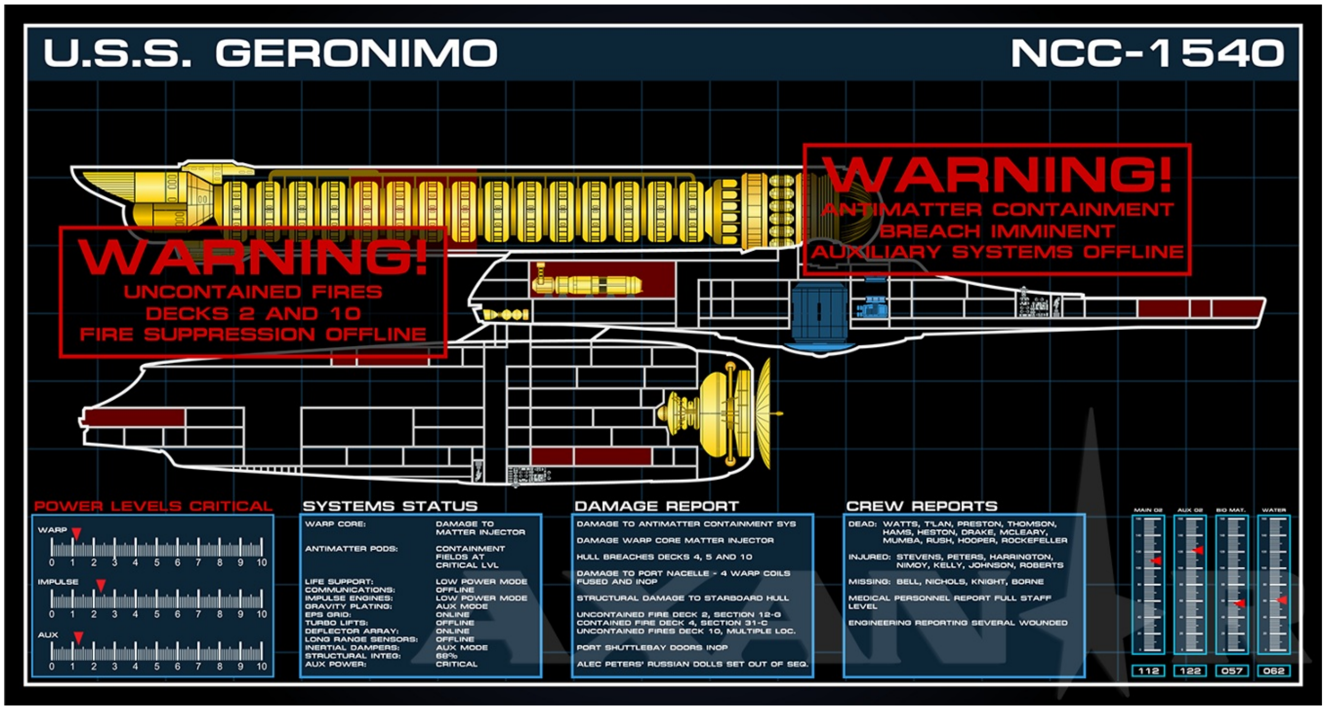
Soon, Rick stopped attending meetings. Rick didn't have time for meetings any more because there was too much to code.

Rick closed his door. His whiteboard lay fallow. Rick no longer had time to train anyone because he had too much to solve on his own.

A backlog grew behind Rick. Bugs were popping up in old tools he'd built. They sapped his attention from meeting commitments on new product development.

Of course, these bugs were happening because the users had misstated their assumptions. Of course there wasn't any problem in his work. Of course.

On our project dashboard, green flags changed to yellow. Yellow changed to red. Red lights started blinking. One by one, task statuses changed to "Impeded." Everyone was waiting for Rick.


U.S.S. GERONIMO
NCC-1540
WARNING!
ANTIMATTER CONTAINMENT BREACH IMMINENT
AUXILIARY SYSTEMS OFFLINE
WARNING!
UNCONTAINED FIRES DECKS 2 AND 10
FIRE SUPPRESSION OFFLINE
POWER LEVELS CRITICAL
WARP
IMPULSE
AUX
SYSTEMS STATUS
WARP CORE: DAMAGE TO MATTER INJECTOR
ANTIMATTER PODS: CONTAINMENT FIELDS AT CRITICAL LVL
LIFE SUPPORT: LOW POWER MODE
COMMUNICATIONS: OFFLINE
IMPULSE ENGINES: LOW POWER MODE
GRAVITY PLATING: AUX MODE
EPS GRID: ONLINE
TURBO LIFTS: OFFLINE
DEFLECTOR ARRAY: ONLINE
LONG RANGE SENSORS: OFFLINE
INERTIAL DAMPERS: AUX MODE
STRUCTURAL INTEG: 68%
AUX POWER: CRITICAL
DAMAGE REPORT
DAMAGE TO ANTIMATTER CONTAINMENT SYS
DAMAGE WARP CORE MATTER INJECTOR
HULL BREACHES DECKS 4, 5 AND 10
DAMAGE TO PORT NACELLE - 4 WARP COILS FUSED AND INOP
STRUCTURAL DAMAGE TO STARBOARD HULL
UNCONTAINED FIRE DECK 2, SECTION 12-G
CONTAINED FIRE DECK 4, SECTION 31-C
UNCONTAINED FIRES DECK 10, MULTIPLE LOC.
PORT SHUTTLEBAY DOORS INOP
ALEC PETERS' RUSSIAN DOLLS SET OUT OF SEQ.
CREW REPORTS
DEAD: WATTS, T'LAN, PRESTON, THOMSON, HAMS, HESTON, DRAKE, MCLEARY, MUMBA, RUSH, HOOPER, ROCKEFELLER
INJURED: STEVENS, PETERS, HARRINGTON, NIMOY, KELLY, JOHNSON, ROBERTS
MISSING: BELL, NICHOLS, KNIGHT, BORNE
MEDICAL PERSONNEL REPORT FULL STAFF LEVEL
ENGINEERING REPORTING SEVERAL WOUNDED
MAIN O2
AUX O2
BIO MAT.
WATER
112
122
057
062

Don’t worry, Rick will handle it. All of it. (source)

The project manager got a six-month extension from the sponsor. At the end of the six months, production-readiness was estimated to be seven months away. At the end of a year, production-readiness was two years out.

Rick was churning out code faster than ever. He was working seven-day weeks, twelve hours a day.

Everyone knew only Rick could pull the team out of this mess. Everyone held their breath and waited for Rick to invent the miracle cure that would mend this crippled project.

Every day, Rick grew more belligerent and isolated. The mask was coming off. Jekyll was becoming Hyde.

I participated in my first meeting with the project team about two years after the original agreed release date. I’d been aware of the project for a while, because it had grown infamous in my organization, but hadn’t been assigned to it.

I was sent in to see if we could save it.

My first meeting on the project was the aforementioned “Albert Einstein” meeting.

Hmm.

I dove into the source code. Rick was right: no-one could possibly understand what Rick had created. Except for Rick. It was a reflection of the workings of his own mind. Some of it was very clever, a lot of it was copy-pasta, it was all very idiosyncratic, and it was not at all documented.

I went to our CIO with the verdict. Only Rick would ever be able to maintain this product. Plus, every day that Rick worked on the project moved the delivery date back a week. Rick was destroying our product faster than he was creating it.

We sat down with Rick and had a conversation about his role in the project. We reviewed our concerns. We sidestepped his self-comparison to Albert Einstein.

We explained our new strategy. The team was going to collaborate on building a new product from scratch.

This effort would be very limited in scope and would only provide the bare essentials to get us to production. The whole team would contribute and be able to support it. No more bottlenecks.

How did Rick react to this?

The only way Rick could. Rick exploded.

Rick wanted no part of this farce. If we couldn't appreciate his genius, that was our fault, not his. Rick predicted that within months we'd come crawling back to him begging him to save us.

Rick screamed that we lacked the basic mental capacity to appreciate genius when it was staring us in the face.

Sadly, after this, Rick rejected months of overtures by leadership. He refused to take time off or allow any work to be delegated. He rejected repeated attempts to introduce free open source frameworks to replace his hard-to-maintain bespoke tools.

He reverted code changes — including tested bug fixes — by other developers. He asserted that he wouldn't be held accountable for supporting other people's work. He continued publicly belittling his colleagues.

We fired Rick.

It took about a week for the dust to settle. It took time for the shocked team to gather themselves after losing their embattled guru.

Then I saw them huddled around a whiteboard.

Collaboration. Rick had never seen this before. (source)

They collaborated. They designed a replacement product. It would be much simpler.

It wouldn't have all the bells and whistles. Nor would it anticipate requirements from five years down the product road map.

Rick's product supported a dynamic workflow with over fifteen thousand permutations. In reality 99% of our use cases followed one of three paths. The team hard-coded the workflow. This removed over 30% of Rick's work.

It wouldn't have custom hand-coded components for every task. They stripped out every bespoke dependency that they could buy instead of build.

This removed hundreds of hours of Rick's contribution. But it also removed thousands of hours of technical debt.

We obtained an agreement from the project sponsor to shut off some edge-case functionality.

This had served only 5% of our pre-launch user group and was responsible for about a quarter of the product's complexity.

We re-released the product to this group. It consisted of 10% of Rick's original code which was pretty stable. It also had a few thousand lines of new code to replace about 150,000 lines of incomprehensible mess.

The team had replaced five years of work in about six months. Over the next few months we expanded from pilot to full customer release.

Not only had we replaced what Rick had built, we sped past him and fully launched the product — all in under a year. The result was less than a fifth the size and complexity of what Rick had built.

It was also hundreds of times faster and nearly bug-free despite having been assembled in a fraction of the time and serving ten times as many customers.

The team went back to Rick's other products. They threw away his old code there, too.

They re-released another product of his after three years in development, with three months of concerted team effort.

There were no Ricks left on the team. We didn't have any mad geniuses building everything from scratch. But our productivity was never higher.

Rick was a very talented developer. Rick could solve complex business logic problems and create sophisticated architectures to support his lofty designs. Rick could not solve the problem of how to work effectively on a team.

Master builders are cool, but skyscrapers are built by teams. (image © Warner Bros. Animation and The Lego Group)

Rick's presence was destructive in several ways.

First, he created a cult of dependence. Any problem eventually became a Rick problem, a myth he encouraged. Developers learned to stop trying and just wait for Rick.

Second, he didn't write maintainable code. He never documented or tested anything, and so failed in spite of his own intelligence. His belief in his personal infallibility trumped common sense.

Third, he was personally destructive. Team members didn't want to speak up and offer their own ideas because he always berated them for it. Rick only respected Rick and went out of his way to make everyone else feel small.

Fourth, he lacked all personal accountability. No failure was his fault. He sincerely believed this, and it prevented him from learning from his own mistakes.

I don't believe Rick started out this way. I saw him at his worst. This was after years of working escalating overtime and facing increasing criticism from customers and colleagues.

It's sad that Rick descended this far. His manager shares in this responsibility. In fact, the original management team was held accountable: they were let go first.

Unfortunately Rick was so far gone that he couldn't, or wouldn't, be brought back. No amount of coaching, feedback, time off, or assignment to other projects changed his toxic behavior.

By this point the whole team knew he was destructive. But the cult of dependence was so strong that everyone believed he was the only option.

There is always another option.

Your team's strength is not a function of the talent of individual members. It's a function of their collaboration, tenacity, and mutual respect.

Focus on building teams that value each other and try to bring the best out of one another.

Together, they'll be able to tackle greater challenges than Rick could ever fathom.

The events described in this essay took place many years ago and do not reflect the opinions or experience of my current employer.

I have published a follow-up story with our lessons learned if you are interested in reading more! You may also be interested in reading about my first job at a startup, which happened to be imploding around me

You can follow me here or on Twitter @jhsolor for more updates.

Note: Some details (such as names) have been changed. I've never actually worked with anyone named Rick.

Jonathan leads enterprise software development and architecture teams.

He earned a Physics degree from Stanford University and has since spent over 10 years working in information systems architecture, data-driven business process improvement, and organizational leadership.