How many emergency releases does it take to change a lightbulb?

Preceded by dropped jaws, wringing hands, and cries of agony, the call for an emergency release is rarely a joyous occasion. The act is more than likely an admittance of failure of the project team, followed by a "holy crap" call to action. Wouldn't that be the most awesome definition? I can see it in the Webster's dictionary right now.

> **Emergency Release** (i-ˈmər-jən(t)sē ri-ˈlēs)
> *the "holy crap" call to action followed by the admittance of failure.*

Nobody wants to go there, and we pray we never have to make that call. If you've been running software projects for more than a day, you've probably had to make that call, and it's usually not pretty. Right after that emergency release is ordered, it is followed by "How did that happen?" or "Why didn't we see that?" The final resolve is **never to let anything like that happen again**.

You take that to heart until the next time when something entirely different causes another emergency release. You kept true to your word, but there has to be a way to keep these emergency releases less frequent. We have three tips geared toward the prevention of those horrible emergencies.

1. Take your stand at the very beginning – with requirements

Ahh, business requirements! Whether you love them or hate them depends on the group you work with. If you've got a group of leaders that is new to the requirements gathering game, don't be fooled.

They may be thinking that they know everything you need to know about what this software needs to do. The challenge is that the problem is that leaders paint with a broad brush for requirements because they see a bigger picture. That isn't a fault with leadership. The big picture is their focus, leaving them wearing blinders for the fine details.

The one tasked with gathering functional requirements should know to NEVER stop at their broad description. Pairing a rookie Business Analyst with a rookie leadership team guarantees a few significant changes down the line. If you're lucky, that pesky goat gets a taste of your Gantt chart. If you're not so fortunate, you're in for one or three emergency changes down the line.

Internal software

When you get to the leadership team, you can first gather their functional requirements, because it does help to have a solid understanding of the big picture. Next, you'll also want some people from the trenches to be "*voluntold*" to share with you the nitty-gritty details. They are the ones to share the wants, needs, and reasons why, so you can build solid user stories. And, in your Agile grooming sessions, y~~Y~~ou also will want to capture all dependencies, especially when data and functionality are shared or transferred across different systems.

Pro Tip: Lighthouse can ~~review~~ inspect your requirements/user stories and help you write acceptance criteria and test cases prior to your developers building the code. In this way, the developers will know the answers to the test! Contact us for more on this because it's an inexpensive way to save a ton from rework and missed requirements.

Customer-facing software

Customer-facing requirements require a little more finesse. Some places to gather information may be with customer service because they are most familiar with the user experience. Another resource may be data from the help desk for known issues that the project is designed to correct or upgrade. The last great source of requirements is from your sales team. Ask them what competitors are doing, and sales leads are asking for.

Another action would be getting the voice of the customer. You can do this by using focus groups to get their thoughts, grievances, and ideas first hand. Again, it's critical that you capture the details in well-documented user stories.

2. Get it right the first time – or as close as you can

Yeah, it's a pipe dream. How do we know? We know because software testing is a booming industry. Nonetheless, the sooner you catch a defect, the easier (and cheaper) it is to fix that defect.

Automation testing is your first line of defense. Automation tools test the code right after it's written, giving coders a chance to make corrections before proceeding with the next task. These immediate corrections then cut the level of defects significantly, for the code is corrected before it can affect any dependencies. In other words, automation testing catching one error can prevent a dozen mistakes down the line of following dependencies. **-Psst! We do this for a living, so if you want test automation, let's talk! Ok?**

Another promising technology evolving in the testing world is artificial intelligence (AI). AI bots recognize changes in code and know to adapt testing to those code changes. The result is efficiency improvements with fewer redundancies (because the bots take care of it) and a higher rate of accuracy.

The next consideration is the quality vs. quantity of testers, especially those writing the testing scripts. Experience is what brings them wisdom for identifying common pitfalls with software developments, and they know just the script to write for it. They will use your user requirements as their compass, so you see even several layers down how essential those business requirements are to the team.

A highly-skilled team of professionals will have proven their value time and again in quality, ~~and~~ efficiency, and a risk-based approach. They understand software testing principles~~,~~ and have the experience to change gears quickly as needed, along with the wisdom and judgment to work autonomously. **-Hey! We can provide an audit of your team and processes if you need an expert second opinion. Let's talk more in a 1:1.**

3. Understand your risks

Even with the best professionals, expect some defects to slip through to production. Ideally, they did with your consent after a full assessment of Technical Debt. For most business systems, we recommend targeting a 95% defect removal efficiency (DRE). Unless you are building a system that risks life and limb, it's not generally worth the extra time and money to target anything higher. (Reach out if you want to understand more about DRE or other core metrics)

Somewhere along the line, though, be it at the beginning of development or the activities leading to the revolution of needing development, someone considered risks. There should be an FMEA or some risk analysis out there somewhere in this case. When you are in a time crunch though and facing a list of defects, it's time to consult your FMEA.

Pro Tip: You may need to make adjustments specific to your defects to map out any problems to anticipate downline from the defect.

~~Your business requirements document~~A requirements traceability matrix (which is still very applicable with Agile user stories) can ~~also~~ shed light on dependencies of functionality and criteria to help gauge what can stay and what should be corrected right away. Assessing your technical debt is your last and most effective line of defense from the dreaded emergency change.

Circle back to tip #2 – Lighthouse lights the way

Lighthouse Technologies brings modern solutions to your testing needs, from testing automation to the top software testing professionals in the industry. How do we know? We ran them through The Gauntlet -our own trial by fire for all who work on our teams.

Contact us and let's talk about your upcoming 2020 projects because we want to ensure they end on time and on budget!