Antipattern: Never change a running system

Author(s): Andreas

“Never change a running system” - probably everybody working in IT has heard this sentence. This can be an anti-pattern when there is the “system” and nobody wants to touch it in fear of breaking something.

The system is either very critical or complex so that this fear of breaking it when changing it comes to exist. Often people are discouraged from even trying to change the system, which leads to monopoly of heads or even no one beeing able to change the system anymore. Also often people start working around the system and wrapping additional layers around it to add functionality or even fix bugs. This then leads to an even more fragile and complex system.

Nobody wants to work on such a system, it will be hard to find developers and hard to motivate them. So the problem will increase even further. When software is no longer maintained and updated regularly, libraries and frameworks will soon be outdated, and they will no longer play nicely with the surrounding systems. Other system elements will start to build anticorruption layers around it. Often this results in reimplementing/reverse engineering the system. This comes with high costs and even higher risks due to the lack of information.

What are some examples?

- System part that is implemented and just working, without real pressure for new features like document management.
- Old system that is not in the center of attraction, where nobody wants to work, because it “doesn’t matter”.
- System written in a legacy language, where it is hard to find new people taking over.
- Old system that does not get updated and has a lot of legacy libraries full of potential security risks.

Why does this happen?

- When a complex system is developed by a project team and then given to a maintenance team that was not part of the development process (often a good handoff does not happen since documentation and handoff are often the first working steps that get cancelled when the deadline comes close)
- the system is disregarded since it is finished and up and running and after some time no-one feels responsible anymore and no-one sees the necessity to maintain the system
- the risk of head monopolies is not as important for the deciding people as saving money and time

How can we avoid getting into the situation in the first place?

- write unit tests and have a continuous integration environment for your software so that you can reduce fear of breaking the system
- Document carefully.
- Implement and maintain testcases
- Update technology and the knowledge regularly
- Have your continuous delivery system check for outdated libraries and libraries with security risks
- have a concept for a handoff when the development team and maintenance team do not overlap or try not to have different teams for development and maintenance at all (“you build it - you run it”)
- Mind there is no part in the overall system that is not important
- Pick on a regular basis a random legacy system that has to implement a minimal change and bring it to production.

What are suggestions to get out of the situation if we ended up in it?

Often the preferred way will be a re-implementation of the system. But this is not the ideal solution due to the immense costs and risks. There might be reasons for such a drastic step, for example if the environment is outdated like COBOL code on a host environment, but most of the time it is better to keep the system alive and kicking. The better way is simply to name but hard to do. Start reducing the technical debt. For that the following steps are normally useful:

- Implement tests or complete tests. Start with the most important ones. This gives back more confidence in the overall system. The test can even be used to prove uncertainties about implemented features and side effects. Sometimes test are no longer functioning – revisit them and update.
- Close gaps in the documentation. Even document what you don’t know (yet).
- Update libraries, find dead outdated code – remove it.
- Refactor the code, etc.
- Start changing the system, try to implement tests on the go and set up a continuous integration environment
- Encourage people to work on the system
- Stop building wrappers around the system to avoid changing it