# The six key approaches for business-critical software updates and rollouts

Qentinel

# Introduction

Does the success of your business depend on the flawless operation of software and information systems? It does for most businesses today.

Stop to think for a minute. What do you know about the systems your business relies on? Where do they come from? How is their quality managed? How do you know if something goes wrong? How can you reduce the risk of something going wrong? What are the biggest risks that may materialize? How likely will the biggest risks materialize?

Most business executives say they protect against information system risks by sanctioning their IT suppliers and integrators. Think again! If something goes wrong with your business and your customers suffer, what are the IT sanctions worth? Will they cover your losses? Will they bring back lost business? Will they reassure disappointed customers?

Given all the systems, applications, and integrations, there may easily be more than 1,000 version updates a year, and each of them may affect your vital business processes. Unfortunately, these updates are not as independent as they may seem. An individual app or system may work perfectly alone, but when you combine it with those 300 other systems, the entire ecosystem may become unstable. The devil is in the dependencies.

“Perfectly correct IT” or “100% quality assurance” only exists in fairy tales. You cannot fully control and assure your IT operation. In fact, you don’t even know where all your critical business systems are, or how and when they are being updated. You can, however, systematically reduce and manage the risk associated with your business-critical information systems.

This blog series discusses the six key approaches you should take.

And finally, remember who’s in charge: it’s you.
Because your business is only as good as the software driving it.



Qentinel

# How to apply new control attitude to your digital backbone

Every decent executive wants to have situations under control. But information systems, and particularly their inter-dependencies, have already grown beyond our ability to control them. Seeking full control is, unfortunately, a fool's errand.

The recipe for optimal and realistic control is simple but not always easy to execute:

1. Identify your critical business processes
2. Identify your core systems
3. Identify your critical dependencies
4. Freeze what you can
5. Focus testing on critical integrations

Because you are still in business you certainly know what your critical business processes are and how information systems contribute to them. You may have a myriad of apps and systems but probably only a handful of them are really core systems. A core system is characterized by one or more of the following:

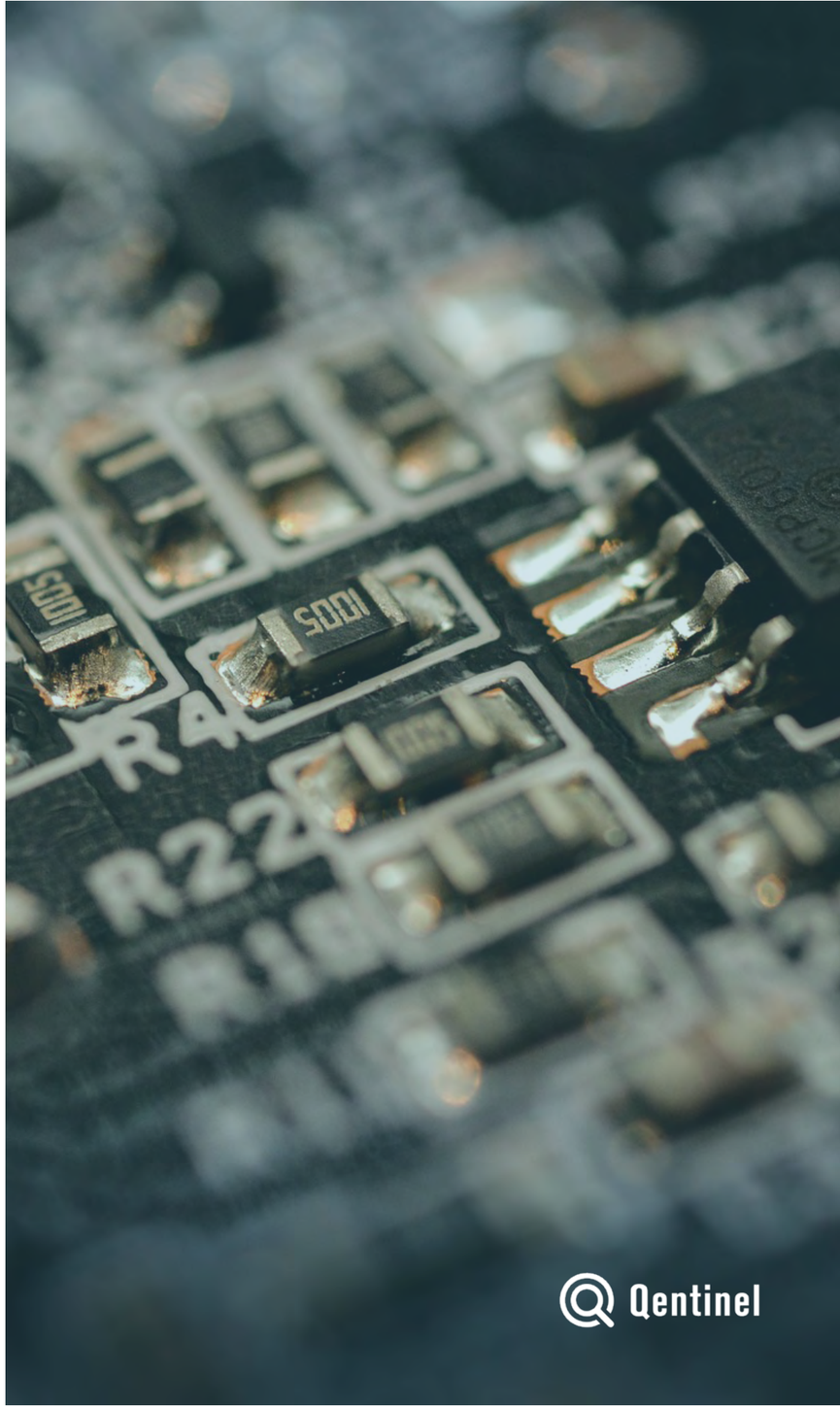
1. Necessary for a critical or core operational process. Manufacturing systems (MES) typically meet this criterion
2. Holds large amounts of critical business data. ERP and CRM systems typically meet this criterion
3. Actively used by a large number of people. Office systems, order processing systems, helpdesk systems, and self-service portals may meet this criterion
4. Serves as an integration hub in enterprise architecture

The fourth criterion is particularly interesting here. Some businesses have built everything around SAP. Some may have a "hidden integration hub" in a very old information system whose database everyone else relies on. Some may have a dedicated integration bus that everything else builds on.

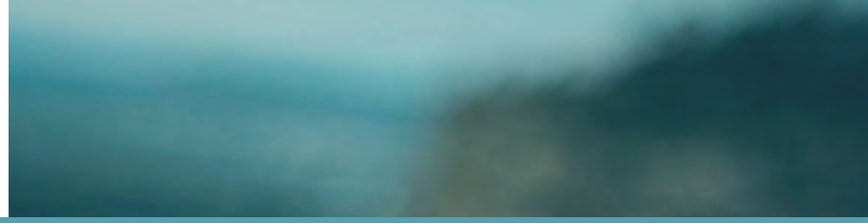
Because software is intangible, fact-based analytical discussion about it may be challenging. The borders of core systems are sometimes hard to draw. For example, the name of the user interface tends to become the everyday name for anything accessed through that user interface – even if that interface is only a thin shell around huge information systems.

Information system problems are more often caused by dependencies and integrations between systems than by any single system alone. Since business moves at the speed of light, its digital backbone needs to evolve almost as rapidly. Therefore, there are a number of not-always-so-predictable system updates underway all the time. The problem is, those who update one system rarely know or care about the other systems it integrates with.

Once you know what your core systems and critical dependencies are you will be able to exercise stricter control and quality assurance over them and let everything else evolve more freely. In practice this means fewer changes in core systems and critical dependencies, more thorough analysis of the impact of those changes, and more thorough QA before the changes go live. It is guaranteed that there will still be problems and there will still be angry people. But at least critical business operations will be less vulnerable.



# Five steps you need to apply DevOps now



If you haven't yet adopted DevOps in your application development, start now. Simply put, DevOps is to software delivery what lean is to production processes. DevOps aims to optimize the ratio of time to value – with high quality, of course. Software developers love DevOps because it's considered cool. Unfortunately, a majority of developers treat DevOps the same way they initially treated agile: by only adopting the fun parts.

All your suppliers are going to deliver new releases more frequently than ever. A growing number of them do not allow you to choose when to upgrade. You will be forced to follow their rhythm. Without DevOps it is going to be painful, if not impossible.

First, familiarize yourself with the subject by reading The DevOps Handbook or The Phoenix Project. Both books are easy to read, not too technical, and teach you what DevOps really is about. After reading these books you can make your own judgements without being manipulated by your techies.

Start with a small but motivated team, preferably competent in agile methods. Select a non-critical assignment. Consider hiring an external DevOps coach. Your team is likely to spend a lot of time figuring out the right ways of working, wrestling with tools, fighting among themselves, and delivering little. Learning a new mindset may hurt. Tools may cause pain, too. There's an abundance of nice open source tools available for DevOps teams but putting them all together for the first time is agony.

Be aware that developers tend to misunderstand DevOps. The core idea is to have development, operations, QA, and security working together at an agile pace delivering value through software. While everybody welcomes the idea in principle, developers usually find it handy to exclude the other three from the process as it makes their lives simpler. QA, Ops, and Sec are often happy to be excluded because they are probably even less familiar with the DevOps way than the developers themselves.

To get DevOps right you need to:

1. Prioritize work strictly based on business value.
2. Recognize that delivering business value sustainably requires continuous investment in software architecture and software process, too.
3. Really have your Dev, QA, Sec, and Ops organized together around the same priorities and the same pace of work.
4. Automate process everywhere you can.
5. Measure everything and use the measurements to make decisions about your business.

Many consider the adoption of DevOps a cultural change, and for a reason. Work culture is not magic: it is about habits and beliefs and they change through work structures, processes, practices, and tools.

To get an idea of what it will take to make DevOps work in practice, imagine taking a bunch of freedom-loving artisans (Dev), an artillery team from a disciplined army (QA and Ops), and a rules-driven government officer (Sec) and putting them all to work in a lean Toyota factory. That's why you need to start soon and start small to achieve success in good time.

Learn more about moving from projects to products **Mik Kersten**'s book **Project to Product: How to survive and thrive in the age of digital disruption with the flow framework**.

Qentinel

# What you need to make your IT organization powerful

"From projects to products" has recently become a fashionable slogan – and for a reason. In the past, an information system was a project investment. We spent a year defining it, a couple of years implementing it, and a few years utilizing it. Then we started a new project to replace the system.

In reality, our information system went through quite a bit of "maintenance", "minor development", and an "upgrade" during its active lifetime. In many cases the number of people needed for "maintenance" was almost as large as the number of people required for the original implementation project.

A project is meant to be a temporary organization with a clear and fixed scope, objectives, budget, resources, and duration. A product organization, on the contrary, is thought to be permanent or at least long-lived. Its mission is to deliver subsequent releases of the product and to enhance the product's competitiveness with each release.

There is no doubt that the lifecycle of an information system today resembles a product lifecycle more than it does a project lifecycle. "From projects to products" simply means making a virtue of a necessity.

Start by establishing a product management function. A product manager leads the information system as an incremental investment, where each increment enhances the value of the system. He or she will also remain constantly on top of the business' requirements, prioritize them and plan a constant flow of system releases that deliver the most value-creating new functions in the product. In other words, the product manager manages the value of the information system as a continuum rather than as a one-time investment.

The same goes for the "product development team". The people actively involved in the development of the system will not go away after the "project is done". They will remain to keep the system competitive. Sure, the size of the team may vary over time.

The shift from projects to products may sound like a huge increase in costs. Sometimes it is, sometimes it isn't. Don't fall into the trap of creating the product organization alongside the old project organization. You may need to re-shuffle roles and responsibilities – and change some people. Basically, you need someone who owns the definition of the product, someone who delivers the software and someone who operates it.

A project manager's mindset is to deliver according to a plan and budget and then move on. An operations manager's focus is to keep things running and to minimize user complaints. A product manager's task is to keep the product competitive by managing its business case.

And what does "project to product" have to do with testing? Just imagine how differently a person responsible for delivering a project and a person responsible for a product think about quality, and how differently they'd prioritize activities. We'll dig deeper into these differences in the next few posts.

# Why you need to test from coding to production

Software testing is the oddball in information system development. It is generally easy to estimate how long it will take and how much it will cost. Testing is also widely acknowledged to be the most important bottleneck in the software release process.

In traditional IT projects, serious testing began shortly before the planned completion of the project. In the past, it was not easy to test the system before everything was put together. Testing folk call this the “big bang”. With the “big bang” approach, a testing effort initially planned to last a couple of weeks easily extended to 10 months. It is expensive, unpredictable – and still dangerously popular.

There is a good reason for testing early: problems are cheap and fast to fix when identified early on. The root cause may be somewhere deep in the system design. Finding it, correcting it, and verifying the correction may take a lot of effort and may consume even more time. What’s worse, such rectifications are likely to introduce a bunch of new problems.

Assume one programmer made an error and five other programmers wrote new code based on the erroneous code – and assume the new code was correct. Once that single error is found and corrected those five other pieces of code will probably each behave erroneously. Imagine this happening a few hundred times. The result is a huge and often panicky test-debug-correct-retest circus that introduces new errors while correcting old ones. This is how the “big bang” works.

The obvious solution is to shift testing left, i.e. test at much earlier stages of system development and much closer to the code. Defects will then be detected when they have not yet affected the rest of the system. Obviously, they are then faster to fix and verify. When it’s possible to spread the testing effort over a longer period of time, the total number of people needed for the process will be smaller.

Agile development methods, modern tools, and cheap computing capacity have eliminated all the old excuses for not testing early. If you want to know how professional your agile teams are, find out how they test…or if they test.

But the evolution did not stop there. The internet, the cloud, and agile methods have facilitated a whole new way of architecting and releasing software. Information systems are no longer independent, slowly-changing monoliths. They are being put together from seemingly independent elements and released frequently.

For example, your ERP is likely to be linked to your CRM. The ERP may be inside your network but the CRM may reside in the cloud. The ERP probably serves several front-end applications developed independently, and it frequently exchanges information with your suppliers’ and resellers’ systems. Whenever one of these systems changes, your critical business processes may be affected.

Somewhat surprisingly, there is also a need to shift testing right, too. Right-shifting means, in practice, executing tests in the production environment. Often, the fastest and cheapest way to ensure everything is still working is to keep running tests in production and to monitor how the systems behave in relation to the business processes they are meant to serve.

Shifting testing left improves the speed, quality, and productivity of software development. Shifting testing right improves confidence in business process performance among different integrated information systems.

## Read more: Shift-right to production testing if you can’t shift-left

Qentinel

# How to get test automation right

It is practically impossible to operate a rapid software release cycle without automated testing. Manual testing is too slow to keep up with the brisk pace of agile development teams. Later on, during system testing and validating integrations with other systems, both the time spent and the overall workload become intolerable.

Test automation is notoriously expensive to perform and even more expensive to maintain, even with the most advanced tools. Therefore, automation should first be applied to tests that are frequently repeated and infrequently changed.

Test automation, if done well, may save quite a bit of human effort. Bigger gains, however, come from time saved. In a best-case scenario, test automation can accelerate your testing by as much as 90% simply by eliminating waiting times from the process.

Let's consider a simple example. A software team is releasing a new system version for testing. It is Thursday. The testers are still busy verifying some corrections from the previous test release, but they will be ready to start by Monday. You've lost more than three days. The testers will complete the test round in three days, i.e. next Wednesday. That is when the development team will be able to start correcting errors found in the tests. Some of the corrections are easy. Some will require so much work that they will be tested in the next test batch in three weeks. After the three weeks a similar test cycle will begin again. Now the testers will be testing new features as well as corrections to errors they found three weeks ago. And so on.

A testing job that took more than six calendar days could have been completed in a few hours by automated tests. The corrections in the code could have been verified any time by just re-running the tests. The time saved in the feedback cycle is immense.

But test automation is not a silver bullet. While you can reduce the number of testers, you still need people to execute tests manually as well as to design and maintain automated tests. A creative human being is still superior to a machine when it comes to testing new features and figuring out what could go wrong. The machine is superior when it comes to doing the same routine repeatedly. In a world of fast cycles and multiple integrations, "repeatedly" represents a large number.

Test automation is usually perceived as the automation of the execution of tests. This is a somewhat narrow view. Broadly speaking, test automation means

1. Automating execution of the tests
2. Automating preparation of test data
3. Automating creation of test cases
4. Automating setup and configuration of test tools and environments
5. Automating installation and configuration of the system being tested
6. Automating reporting of test results
7. Automating analysis of test results and
8. Automating reporting of test statistics and quality metrics

Test automation will deliver huge benefits if done right. Again, it all starts from understanding the quality risks, understanding the dependencies, and understanding the software process. Automating a poor process still leads to automatic mediocrity.

**Test automation can accelerate your testing by as much as 90% simply by eliminating waiting times from the process.**


**Contact:**
esko.hannula@qentinel.com

Qentinel

# This is how you find metrics that matter

A software process tends to be more difficult to manage and optimize than an industrial manufacturing process. This has little to do with the “intellectual challenge” or “inherent complexity” of software. The history of industrial manufacturing just happens to be longer, its underlying practices, processes, and tools are much more standardized, and there is less variation to manage than with software.

A business that is only as good as the software driving it can stay competitive by continuously improving and optimizing its software process. And no: this has nothing to do with killing innovation. In fact, the better, faster, and more automated your process is, the more you can afford to innovate.

The software process needs rigorous measurement. We need to see if things are developing for better or for worse. We need to rapidly find the right levers to adjust when improvement or corrections are needed. Luckily, the software process is easy to measure. Almost everything from code to helpdesk involves tools that produce a wealth of data that can be used to understand, control, improve, and forecast the quality of the software and the process creating and releasing it.

People in different roles need different views of the same information. Trying to run the software process by focusing exclusively on defect reports is a bit like trying to run a business by looking at accounts receivable and accounts payable only. Likewise, trying to manage and develop a business by reading fiscal statements is a doomed effort, but analysing the trends of crucial financial indicators gleaned from fiscal statements is much more useful.

The purpose of software metrics is to facilitate the best possible decisions at the right time.

Current literature provides hundreds of handy software quality metrics. The metrics that have served me best in software quality management are the following:

1. Pass/fail statistics.
2. Error accumulation
3. Defect detection percentage
4. Time/Phase distribution of errors
5. Architectural distribution of errors
6. Rework accumulation
7. Wait accumulation
8. Code changes per day/week

But this is just a tip of the iceberg. A comprehensive account of how to make metrics work to your advantage is here. https://info.qentinel.com/blog/how-to-set-up-a-no-nonsense-devops-dashboard

A word of warning may be necessary, though. Most people hate being measured. They don’t believe management has good intentions when it comes to metrics. This means any measurement effort that requires people to report things manually is doomed to fail. Therefore, automate your metrics.

Attitudes towards measurement will change over time once people learn to use metrics first to understand, and only afterwards to control.