5 IT Operations Cost Traps and How to Avoid Them

Key Takeaways

- Decisions during the initial development or integration phase for new solutions impact future operations and maintenance costs heavily.
- Wrong assumptions regarding midterm funding for feature development might result in wrong decisions in the area of sourcing. Devops and #noproejcts organizations are especially at risk since they implicitly assume a more constant staffing than the old “project vs. operations” approach.
- Various cost traps relate to tensions between stakeholders such as vendors and customers regarding middleware choices or between agile teams and the senior management regarding centralizing operational and support tasks.
- Planning to switch the funding to a different Organization once the application is live increases the risk for unexpected high operations and maintenance costs.
- Sustainable cost optimizations for IT application landscapes requires cost models linking architectural decisions with future (!) maintenance and operations cost.

Many companies would love to shift budget from IT maintenance and operations to innovative IT projects. But, they struggle to identify new savings options, because recently most organizations have gone through many cost-cutting, outsourcing, and offshoring initiatives. We present a fresh perspective for such organizations — five decision areas for projects, which significantly impact later operations ("traps"). Avoiding these traps is a simple way to free up money for real innovation.

Software developers enjoy delivering new features on a daily, weekly, or monthly basis, using Agile and DevOps methodologies. While teams working on crucial applications in innovative companies might work in this mode for years, the reality is different for most engineers and applications. Many smaller software projects start with an intense initial phase that lasts from a few months to a year or two. Then, the application delivers, hopefully, the expected business value. Development continues, but after some more months (or years), the project reaches a turning point. The business realizes that investing more money for new features doesn't add much business value. They cut the funding, meaning less work for engineers. Most of them transfer to other products or projects. DevOps becomes "DevOps without development work," i.e., old-fashioned maintenance and operations. Occasionally, the remaining engineers implement a few changes or smaller upgrades, as shown in Figure 1. The rest of the time, they deal with identifying bugs, dealing with user issues, and maintaining roles.

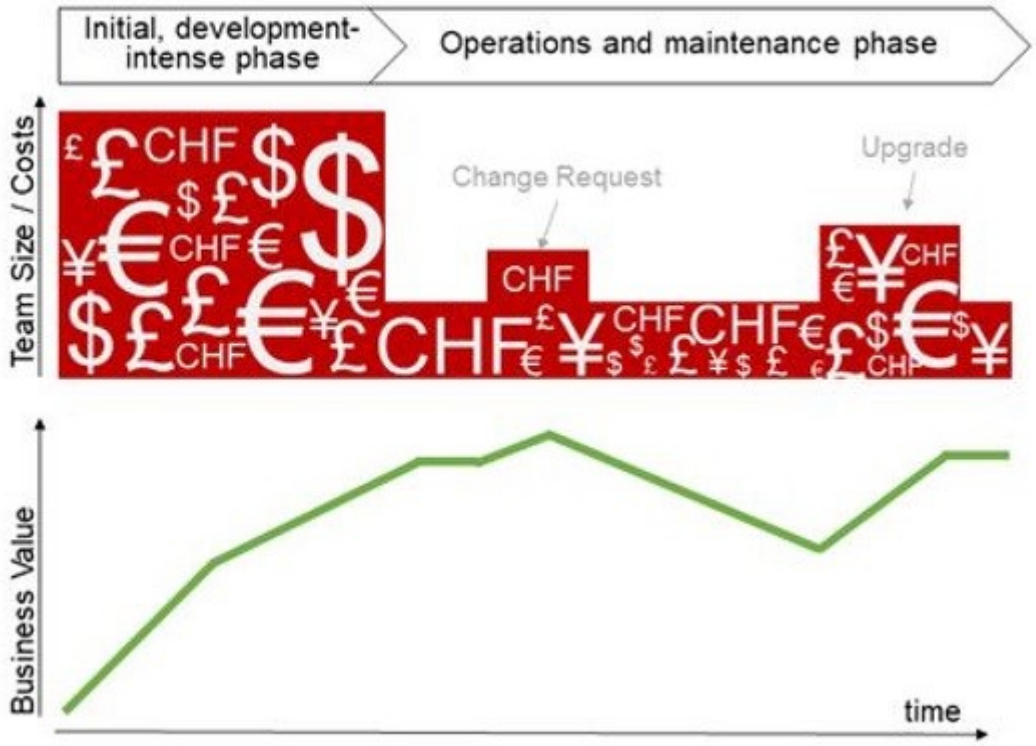

Figure 1: Over time — when adding new features does not increase the business value much — DevOps becomes maintenance only with just a few upgrades.

A transition from a staff-intensive initial phase to a cost-effective maintenance and operations phase has one major challenge: all tasks and "knowhow" need to be transferred to a small number of remaining engineers or moved to other teams in the organization. Newer approaches, such as those in the #noprojects movement, can ease some challenges, but the commercial reality remains the same: Fewer development tasks implies fewer engineers with hands-on experience who can step in to solve issues or implement changes.

Such a transition is not only crucial for many business cases for new software, it's also crucial to be able to innovate. Less money spent on software in the operations and maintenance phase means more funds available for launching the next innovation project. Clever organizations plan ahead, because they understand this causality and avoid five costs we discuss in the following.

Trap 1: Centralization doesn't fit the zeitgeist

A classic cost-savings approach is consolidating and centralizing middleware and database ecosystems. This approach has three dimensions:

- Centralizing the expertise — one central database administration team supports the organization.
- Shared installations and instances — one shared database, instead of one installation per application.
- Reducing the technology zoo — there is one technology (or two...) as the company standard, and it has to be used. Other similar solutions are phased out and not used in new projects. For example, IT can opt for MariaDB and Oracle as standard, and discontinue the use of DB2, Microsoft SQL Server, and MySQL.

On a first look, centralization contradicts the spirit of DevOps and Agile. Agile teams want to be self-sufficient. They want to have all needed skills on their team so they don't depend on external, centralized help to deliver their sprints. While such self-sufficiency is a guiding principle, DevOps teams always rely on some centralized teams. Hopefully, no DevOps team considers building their own data centers or trying to manage the OS level with all virus scanning and patch management by themselves. So, the real questions are — what must be sourced to a centralized team for cost, compliance, or other reasons? In which areas are project or product teams free to choose to do the work themselves, even if there is a centralized team for this topic? Figure 2 below illustrates this ecosystem of standard services.

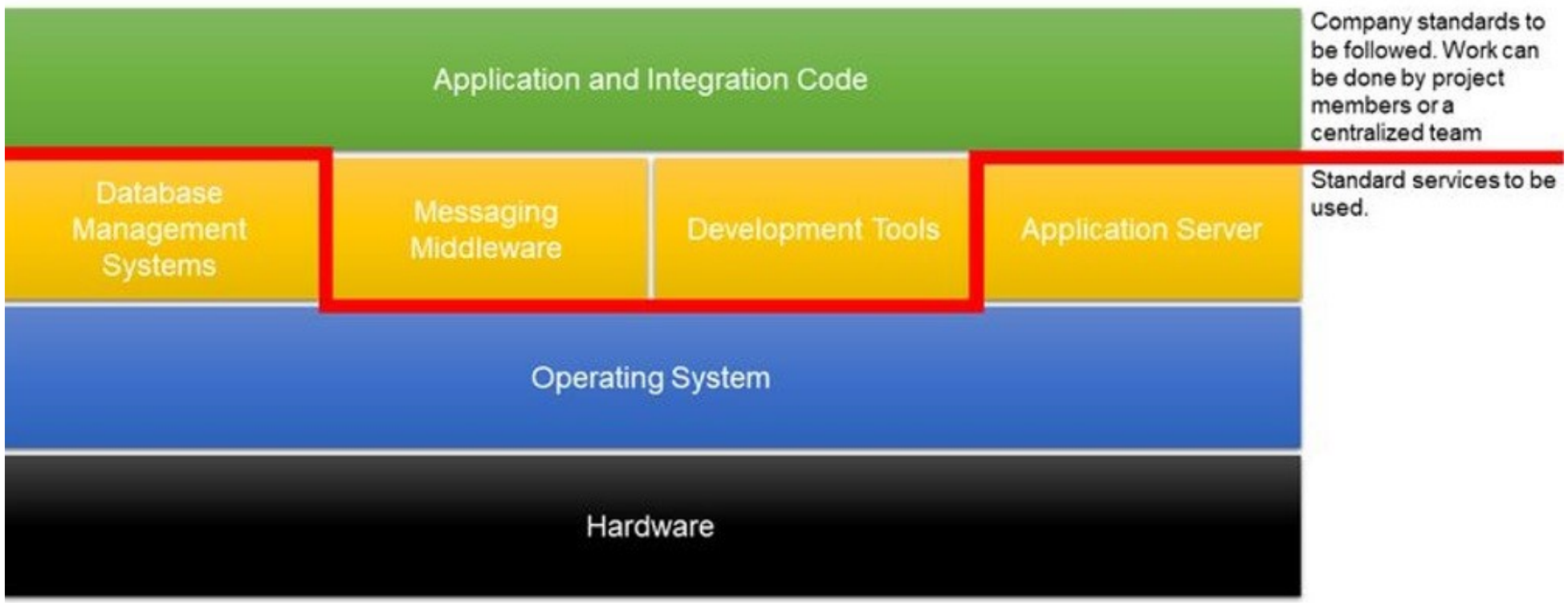

Figure 2: This illustration shows how IT organizations can define where teams have to follow company standards and where they are free to make individual choices. Services below the red line are centralized: hardware layer, operating system layer, and certain middleware services (databases, application servers). Tasks regarding the application and integration layer and for dedicated services such as messaging and development tools can be taken care of by the team. However, they are free to use centralized services if available.

Ultimately, every company and IT organization has to ensure that teams, Agile or not, perform activities and make decisions in line with overall company goals and the CIO's strategy for IT. They define the boundaries within which all Agile or non-Agile and DevOps or old-fashioned development and operations teams act.

A more fine-grained approach differentiates between who does the work (centralizing expertise), whether a central installation has to be used (sharing installations), and whether the organization has mandatory guidelines and standards applicable for all project and product teams (reducing the tech zoo). In the latter two cases, a project or product team can perform the work themselves, as long as they use a shared instance and/or respect company standards.

However, this independence can be a cost trap. There might be enough work for a database administrator or a data scientist in the current phase, but the situation might be different in half a year. Once the project or product switches to an operations and maintenance phase, the team size shrinks dramatically. Specialized tasks must be moved to dedicated centralized teams, to reduce costs without harming quality and stability needs. If this task shift is not possible, and costs remain high, unwanted management attention and actions might be the result.

Trap 2: Avoid over-specification

It's costly to blindly follow the vendor requirements and installation guidelines. In one of my projects, the software vendor expected the bank to use an open-source database they ship with their product. At the same time, the bank had a large, working Oracle database with support from database administrators.

Just think of the cost and quality difference:

- The effort to put a small schema in an existing shared database is minimal.
- The effort to train a 2-3 person application team in a new database product is considerable, and still they'll only have minimal hands-on experience compared to full-time database admins.

The reason for such over-specification is simple: software vendors aim for economies-of-scale. Thus, they restrict the number of support middleware options (vendors and releases) so they reduce the number of configurations to test. They over-specify the configuration by stating, for example, that RedHat 7.6 with Java 13.0.1 and SQL Servers 16 SP2 is certified, but any other combination is not. Obviously, the application works with more or less all Linux distributions and SQL databases. They just don't want to spend time and money testing all of them. This approach conflicts with the needs of IT departments. They want to standardize their applications and centralize tasks (see trap 1). All applications should run on the same release. They don't want to have a different database or Linux installation for each of their hundreds of applications. Figure 3, below, illustrates this conflict.

	Database Management Systems							Messaging			
Product	Microsoft SQL Server			MariaDB		MySQL	DB2	Rabbit MQ		Apache Kafka	
Release	10.5	12.0	14.0	10.1.41	10.3.16	8.0.18	11.1	3.7.14	3.8	2.2	2.3
ERP System				Works	Works	Certified		Works	Certified		
CRM Solution	Works	Certified	Works	Works	Works	Works				Works	Certified
Process Engine	Certified	Works	Certified					Certified	Certified		
Printing Solution		Works	Works	Works	Certified		Certified			Certified	Works
Customer Service App			Works	Certified	Works	Works		Works	Certified	Certified	Certified
Central instance and support team in place		✓		✓				✓			✓

Figure 3: Keep operations costs low by considering existing central instances and support teams when choosing middleware solutions and the exact release version to be used. In this example, the blue rows state for each software which middleware it can be used with distinguishing between certified options and options that work but are not supported by the software vendor. The green row notes whether there is already a central instance and support team for a certain software version.

IT departments can avoid the over-specification trap, by deciding explicitly which of the vendor's system requirements they fulfill. New middleware easily induces six-digit costs per year, at least for the next four to five years. Training, adding engineers and/or engineers with more senior profiles, monitoring, patch management, etc., add up.

What should you do if the software vendor or the project insists on a specific middleware, causing unexpected follow-up costs? The answer is simple: rework the business case. Then, the project sponsor needs to decide to continue, modify, or cancel the project.

Trap 3: There is no linear correlation between development and operations costs

Elasticsearch is a great open source component for adding search functionality to a software solution. For me personally, Elasticsearch was an eye opener: development and maintenance costs often do not correlate linearly.

What happened was that an engineering team convinced the business that Elasticsearch is a cost-effective and quick way to add search functionality to the software. The software development project manager was proud of his team and the business loved the proposal as well. Adding complex new functionality with minimal effort using inexpensive or free software as building blocks is great engineering work. However, there is one thing to be aware of. If you add 100 lines of code to an application with already 100,000 lines of code, this has little impact on maintenance and operational costs. If you add 100 lines of "glue" code to integrate one or two "free" components, the driver for maintenance and operational costs is (mainly) the complexity of the added components and of the additional interfaces between components, not the number of lines of code.

To illustrate this: It is clear for everyone that Linux or Hadoop are free, but you need dedicated teams to run them. The same applies, though often less obvious, for solutions such as TYPO3 and MariaDB and integrations of Elasticsearch or for an interface to Google Maps. You might pay no license fees, but you still need a team to maintain and operate such solutions and interfaces to them.

The trap is to assume that one has to maintain only a few lines of code — and to oversee that there are components and interfaces causing a lot of work for the operations staff. The solution? Consider not only the effort needed to maintain self-written lines of code. Ensure to estimate as well the effort to operate and maintain the interfaces and "free" components that will be added to your solution.

Trap 4: Initiators of high operation costs are not accountable

It seemed to be the perfect setup: An innovative IT company develops the software. They are in frequent contact with the business to discuss the needed features in detail. At the same time, the IT department organizes operations which are handed over to a very experienced IT service provider. Such a set-up has quite some charm. Within the company, each organization focuses on what they know best: the business regarding the bank customer needs, and the IT organization regarding sourcing application management and operations to an IT service provider. Also, the bank chose the best company for software development and the best one for operations. However, such a setup is risky. What happened was: The project (and business) focused on features without looking at follow up costs. The IT department was not involved, since they were just talking about features from the business perspective. Obviously, the head of IT was not amused once he realized how expensive operations and maintenance would be in the future. However, the solution was implemented, the damage done.

When taking a step back to analyze such situations — and to understand the risks — a responsibility triangle with three main stakeholders (Figure 4) helps: This triangle consists, first, of the ones doing the engineering work, e.g., software developers. Second, there are the ones deciding about features, e.g., product owners. Third and final, there are the sponsors paying the bills, e.g., the business.

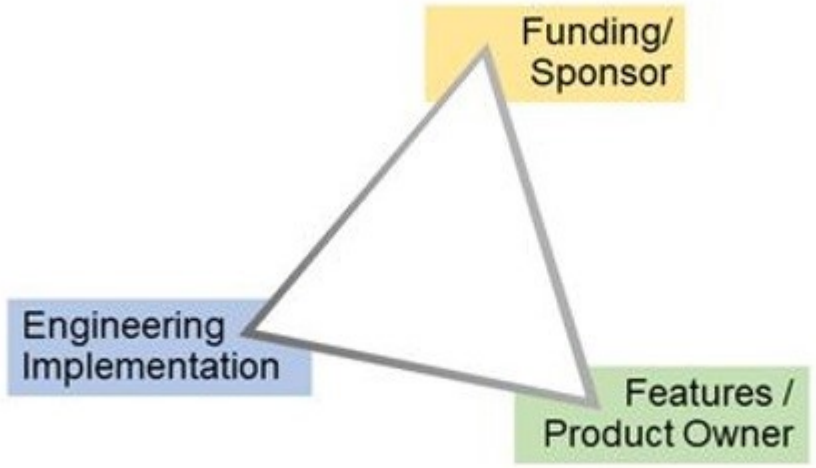

Figure 4: Responsibility triangle with main stakeholders

When the project moves from a development-intense phase to a maintenance and operations-intense phase, and all three stakeholders remain the same, the risk of bad surprises is low. However, especially a change of the sponsor can result in misalignments and operational cost time bombs. While it is easy to blame the project manager or external companies, this is gamesmanship. The real reason is the project culture and/or missing governance: How does the organization measure project success? Just by looking at what was delivered on time and within the project budget? Are the later operational costs reflected as well? Who estimated the impact of changes on operational costs and who approves such a change?

Trap 5: Resistance against simplistic cost models

Have you ever thought about the difference between a good and a great cost/price model? A good model reflects the actual costs and work the IT organization has caused by the services they provide for internal or external customers. This can be an a-posteriori estimation. A great cost model, however, enables customers to simulate, a-priori, how their decisions impact the future costs and, by that, control them.

When you want to simplify an application landscape and reduce IT operations cost in a sustainable way, the key is to link architectural decisions directly to later operational costs. Managers have to know the cost impact before deciding about change requests or architectural variants. This requires simple or even simplistic cost models. You want to add one component with three interfaces? You have to budget CHF 30,000 per year for operations (see Figure 5). If the cost impact on operations is only understood after the implementation, it is too late.

Figure 5: Simplistic cost model for operations costs

The biggest hurdle for such a setup are emotions. Engineers see applications as "their baby," individual, unique pieces of art for which general cost models are too simple and inadequate. Let them implement their proposal, then — and only then —a good operations cost estimation is possible. Another typical statement comes from sponsors having to pay for IT operations. They tend to say, "Oh, it is just a small interface, this causes nearly no extra work. You do not want to rip me off, do you?" Quite a delicate situation in a customer/service provider relationship — I know from my own experience. However, the real trap is an "all inclusive" cost model. It makes inconvenient discussion obsolete since there is no price tag for each component and interface. There is just a flat fee. The side effect: now there is no incentive to optimize. When it is time to recalculate the costs, e.g., after a year, the increased complexity will result in additional costs no one was aware of.

Looking forward

We discussed five typical cost traps for applications which sooner or later move from a development phase to a less-development intense maintenance and operations phase. These five cost traps circle around ownership and governance. Addressing them is often not a question of technical and engineering know-how or funding. It is more a question of whether an organization or management has the will to really work on organizational topics. Get the benefits of speedy, high performing projects and innovative methodologies. Just prepare as well for the day after when no one cares about new features and everything is about costs. Are you ready for this challenge?

About the Author

Klaus Haller is a Senior IT Project Manager and Solution Architect with experience in Data Management & Analytics, Information Security and Compliance, Business Analysis, and Software Engineering and Testing. He likes applying his analytical skills and technical creativity to deliver solutions for complex projects with high levels of uncertainty. Klaus is a Computer Science graduate from TU Kaiserslautern (Germany) and the Swiss Federal Institute of Technology (ETH) in Zurich and publishes frequently articles reflecting his work experience in the IT industry.

Discuss

Please see https://www.infoq.com for the latest version of this information.