DevOps *for the* Modern Enterprise

Winning Practices to Transform Legacy IT Organizations

Mirco Hering

Foreword by
Dr. Bhaskar Ghosh

IT Revolution
Portland, Oregon

PDF COMPANION TO THE AUDIO BOOK

IT REVOLUTION
25 NW 23rd Pl, Suite 6314
Portland, OR 97210

First Edition
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Cover design by Devon Smith
Book design by Devon Smith
Author photograph by [Name]

Library of Congress Catalog-in-Publication Data

Names: Hering, Mirco, author.
Title: DevOps for the modern enterprise : winning practices to transform
legacy IT organizations / Mirco Hering.
Description: First edition. | Portland, OR : IT Revolution Press, 2017. |
Includes bibliographical references and index.
Identifiers: LCCN 2017053420 (print) | LCCN 2017056532 (ebook) | ISBN
9781942788201 (ePub) | ISBN 9781942788225 (Kindle) | ISBN 9781942788195 (trade pbk.)
Subjects: LCSH: Computer system conversion. | Operating systems (Computers) |
Computer software--Development. | Management information systems. |
Information technology—Management.
Classification: LCC QA76.9.C68 (ebook) | LCC QA76.9.C68 H47 2017 (print) |
DDC 004.068—dc23
LC record available at https://lccn.loc.gov/2017053420

ISBN TP: 978-1942788195
ISBN ePub: 978-1942788201
ISBN Kindle: 978-1942788225
ISBN PDF: 978-1942788218

Figures, Tables & Exercises

Preface and Introduction

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 8

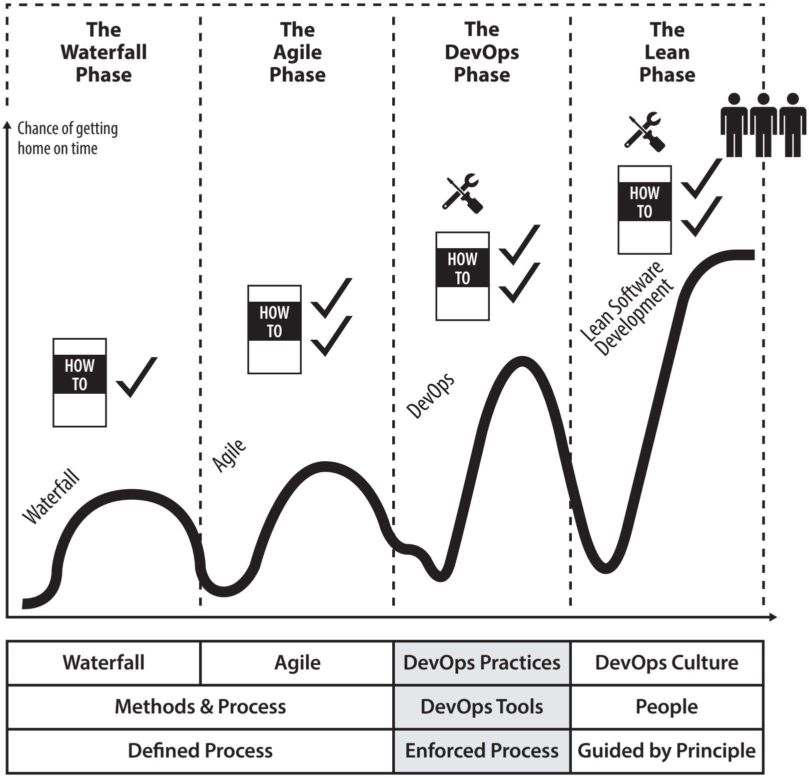
Chapter 9

Chapter 10

Chapter 11

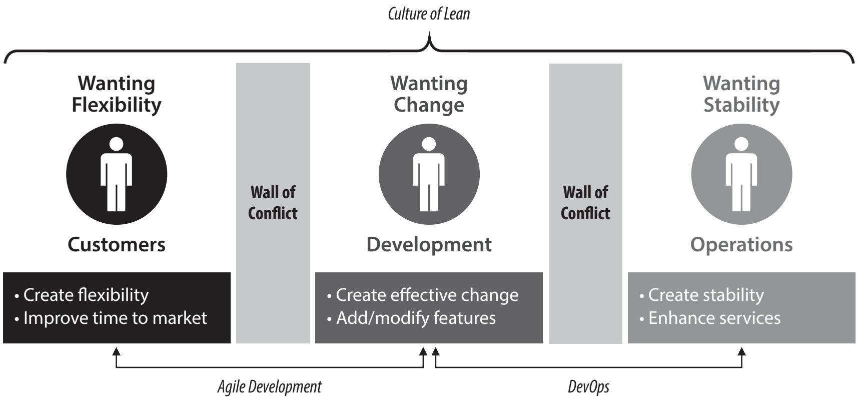
Chapter 12

Appendix



Waterfall	Agile	DevOps Practices	DevOps Culture
Methods & Process		DevOps Tools	People
Defined Process		Enforced Process	Guided by Principle

Figure 0.1: How Mirco's understanding of organizational change evolved



BENEFITS OF AGILE
Alignment between business & IT
Flexibility
More effective solutions
Reduction in risk
(Speed to market—through smaller batches)

BENEFITS OF DEVOPS
Speed to market
Increased throughput
Reduction in risk
Faster feedback
(Reduced cost)

Figure 0.2: Relationship between Agile and DevOps: How the principles Lean, Agile, and DevOps relate to each other

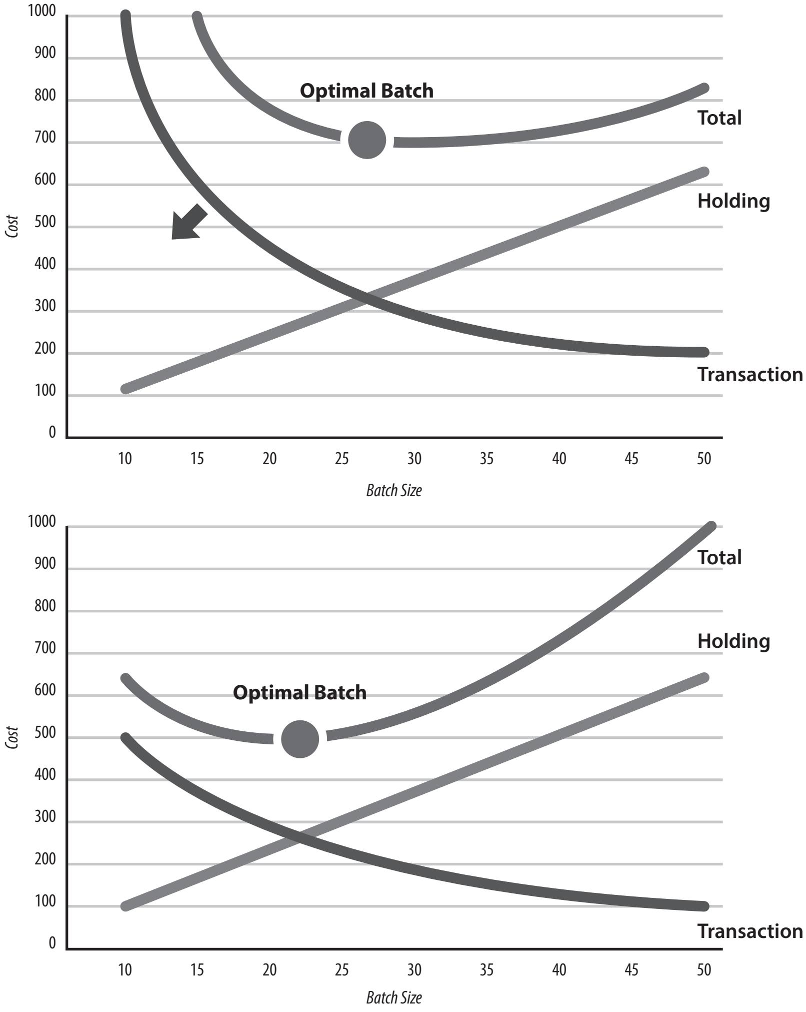


Figure 0.3: Relationship between transaction costs and batch size: Reducing transaction costs allow for smaller batch sizes

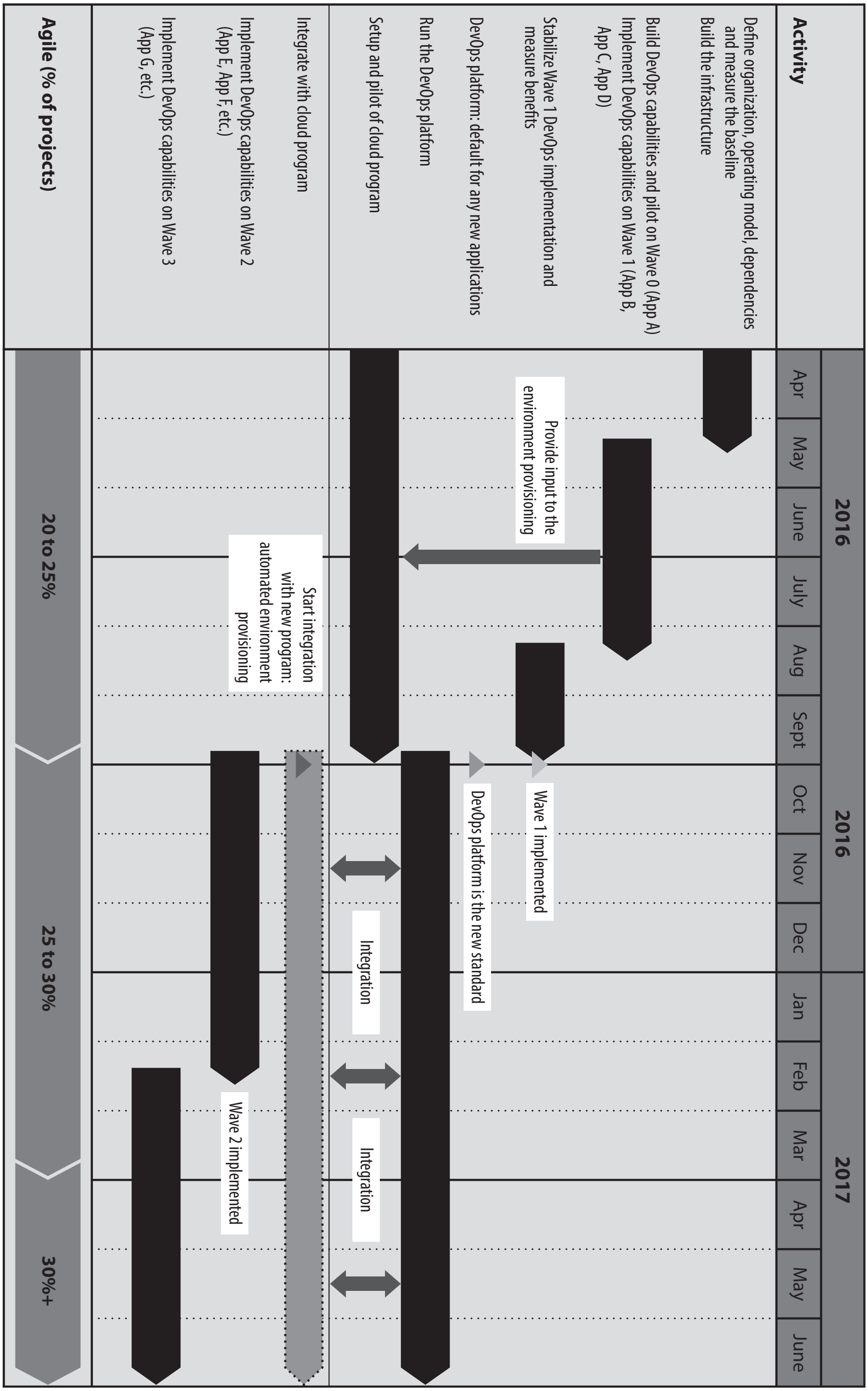


Figure 1.1: Roadmap example showing waves of applications and capabilities

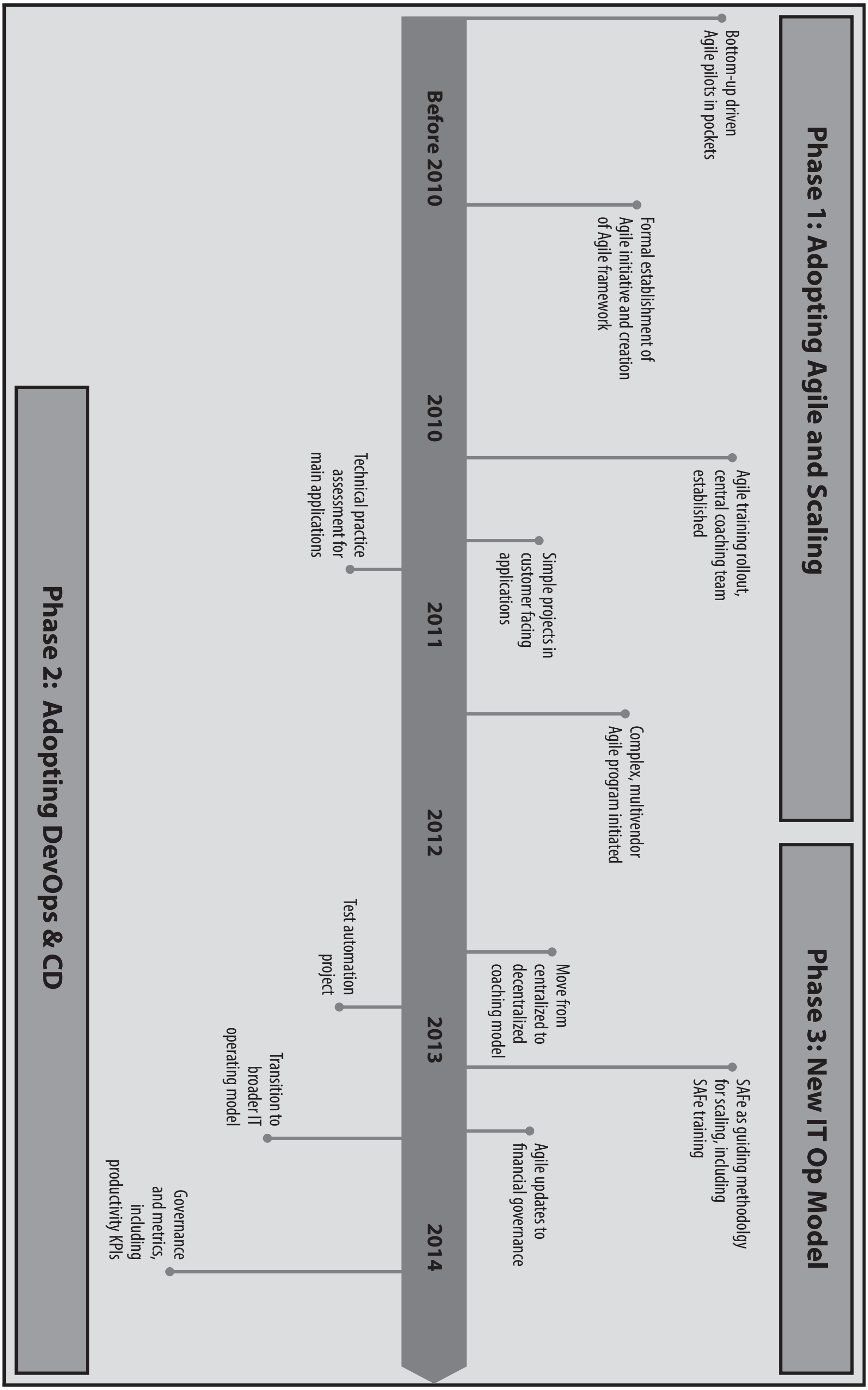

Phase 1: Adopting Agile and Scaling
Phase 3: New IT Op Model
Phase 2: Adopting DevOps & CD
Before 2010
2010
2011
2012
2013
2014
Bottom-up driven Agile pilots in pockets
Formal establishment of Agile initiative and creation of Agile framework
Agile training rollout, central coaching team established
Technical practice assessment for main applications
Simple projects in customer facing applications
Complex, multivendor Agile program initiated
Move from centralized to decentralized coaching model
Test automation project
SAFe as guiding methodology for scaling, including SAFe training
Transition to broader IT operating model
Agile updates to financial governance
Governance and metrics, including productivity KPIs

Figure 1.2: Common transformation blueprint: Changing your organization takes time as you adopt different methods

Metric	Definition	Measurement
Release Cycle Time	The average time it takes to approve a work package (user story, feature, set of requirements) and release it	Usually measured as the time difference between work item states in your work tracking system
Cost of Release	The effort it takes to release new functionality, measured as effort for all release activities performed for go-live (a variation of this only counts effort outside of business hours)	Typically based on timesheets
Regression Duration	Time it takes to validate that a change has not caused regression	The time between deployment and the "all clear" from either an automated or manual validation
Production Availability	Percentage production is available to perform the right service	Measured by a percentage of time production is functionally available or percentage of successful transactions
Mean time to Recovery	Time it takes to rectify any production issue	Measured from time of occurrence until full user functionality is achieved
Longevity of Teams	The average duration teams stay together	Measured as months before teams get disbanded and restructured for new projects

Table 1.1: Baseline metrics: These metrics have proven to be successful in guiding transformations

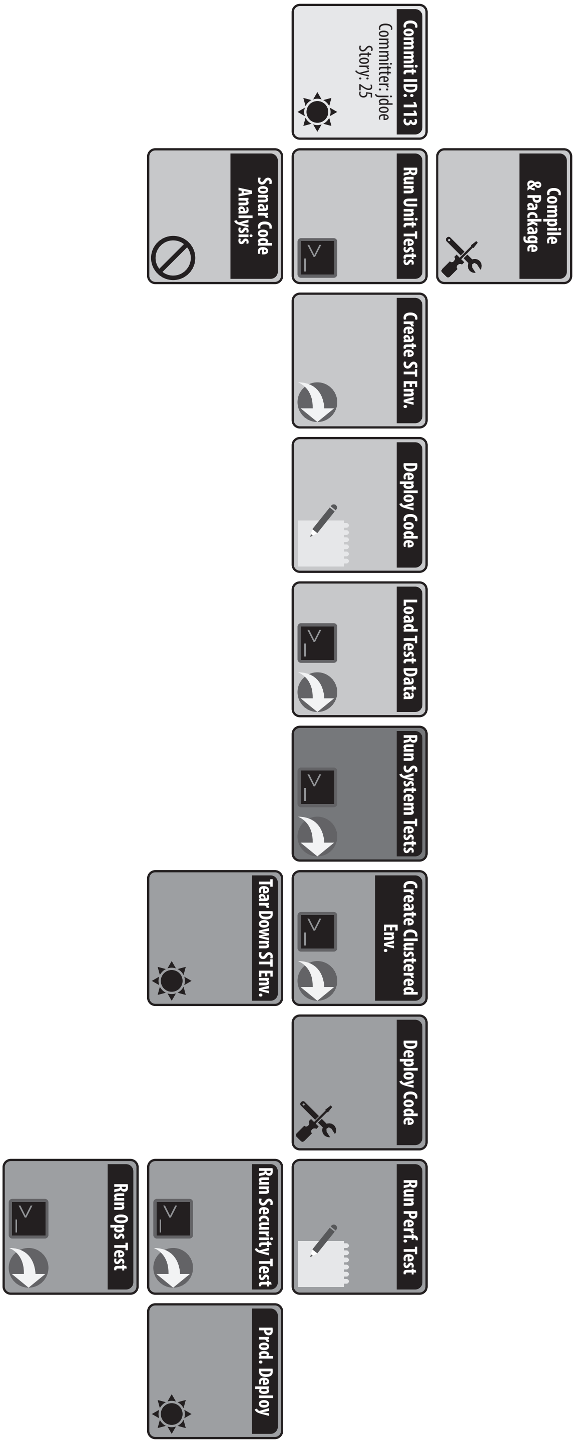


Figure 1.3: Deployment pipeline example: Accenture DevOps platform provides a look into the deployment process

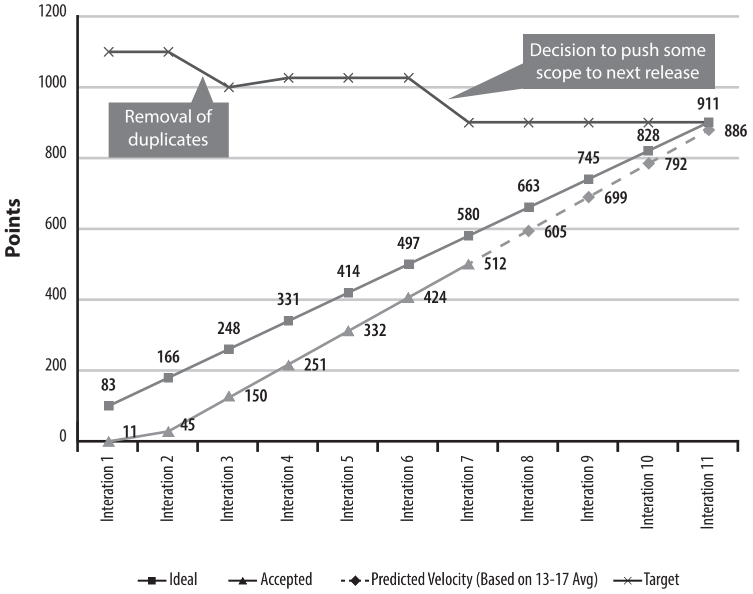


Figure 1.4: Annotated burnup chart: Burnup charts provide an annotated status of the project

First Steps for Your Organization

There are three exercises that I find immensely powerful because they achieve a significant amount of benefit for very little cost: (1) value stream mapping of your IT delivery process, (2) baselining your metrics, and (3) reviewing your IT governance. With very little effort, you can get a much better insight into your IT process and start making improvements.

Value Stream Mapping of Your IT Delivery Process

While there is a formal process for how to do value stream mapping, I will provide you with a smaller-scale version that, in my experience, works reasonably well for the purpose that we are after: making the process visible and improving some of the bottlenecks.[‡] Here is my shortcut version of value stream mapping:

1. Get stakeholders from all key parts of the IT delivery supply chain into a room (e.g., business stakeholders, development, testing, project management office (PMO), operations, business analysis).
2. Prepare a whiteboard with a high-level process for delivery. Perhaps write “business idea,” “business case,” “project kickoff,” “development,” “testing/QA,” “deployment/release,” and “value creation” on the board to provide some guidance.
3. Ask everyone in the room to write steps of the IT process on index cards for fifteen minutes. Next, ask them to post these cards on the whiteboard and work as a

[‡] It is worthwhile for you to pick up *Value Stream Mapping* by Karen Martin and Mike Osterling if you want to formalize this process further.

group to represent a complete picture of the IT delivery process on the whiteboard. Warning: you might have to encourage people to stand up and work together, or you may need to step in when/if discussions get out of hand.

4. Once the process is mapped, ask one or more people to walk the group through the overall process, and ask everyone to call out if anything is missing.
5. Now that you have a reasonable representation of the process, you can do some deep dives to understand cycle times of the process, hot spots of concerns for stakeholders due to quality or other aspects, and tooling that supports the process.
6. Get people to vote on the most important bottleneck (e.g., give each person three votes to put on the board by putting a dot next to the process step).

In my experience, this exercise is the best way to make your IT delivery process visible. You can redo this process every three to six months to evaluate whether you addressed the key bottleneck and to see how the process has evolved. You can make the outcome of this process visible somewhere in your office to show the improvement priorities for each person/team involved. The highlighted bottlenecks will provide you with the checkpoints for your initial roadmap, as those are the things that your initiatives should address.

Baselining Your Metrics

Because having a baseline of your metrics is such an important part of the transformation governance, I want you to spend a few minutes filling out your own Table 1.2. Identify the metrics you care about now and in the future, and identify the mechanism you will use to baseline them. There are a couple of ways to identify the baseline. The baseline approach can be based on surveys, time-in-motion studies, or, ideally, existing and historical data. Where this is not possible, you should think about investing in an automated way to measure this metric. Where that fails, you can run a manual investigation and measuring process (e.g., time-in-motion studies), but those are less reliable and more time consuming.

Metric	Definition	Measurement Mechanism	Baseline Approach	Baseline Value
Release cycle time	The average time it takes for a story to go from a "ready" state to being deployed in production	Extract of date and time from Agile life-cycle-management system	Historical analysis of the last six months of user stories that were successfully deployed in production	168 days

Table 1.2: Metrics definitions example: Metrics should have definitions, measuring mechanisms, and baseline values

Reviewing Your IT Governance Process

There is a lot of talk about automation to help improve the IT delivery process when it comes to speed of delivery and quality. One thing that people underestimate is how much they can influence by just improving their governance process. Here is a short checklist that you can use to review your governance process. Ask these questions to guide where IT delivery governance is really required. Based on the answers, you can evaluate the impact and risk of removing the process step, ideally even with an economic model reflecting monetary impact and risk probability.

IT governance checklist:

- How often has someone rejected a submission to the checkpoint based on reasons other than process compliance?
- What would really happen to the process if an incorrect decision was made?
- What value is being added by the person approving this checkpoint that a computer could not provide automatically based on a number of inputs?
- How much time and money are being spent on this governance process (including the usual wait time that initiatives encounter while waiting for approvals)?
- Is this governance step based on objective measures or a subjective measure? How do you know?

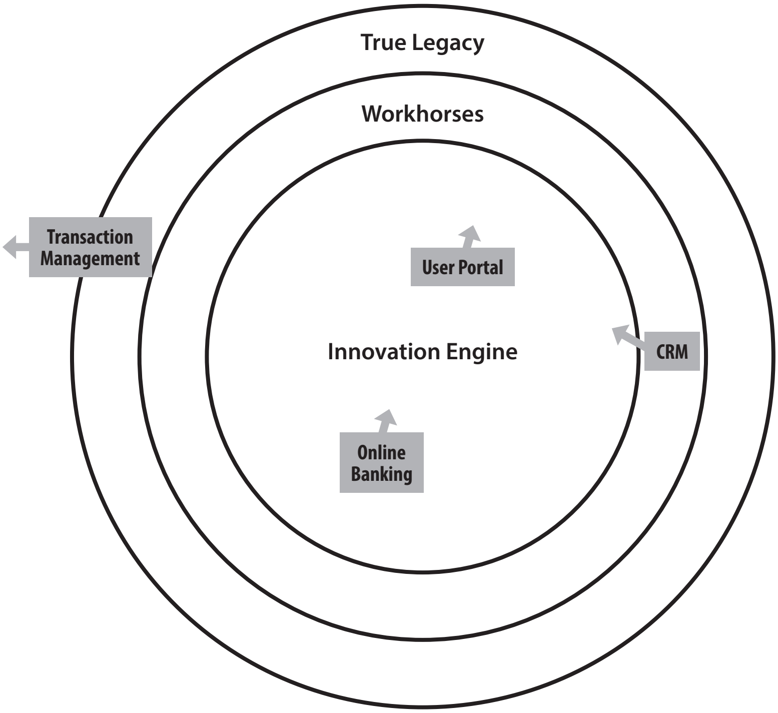


Figure 2.1: Application radar: Makes the status of each application visible

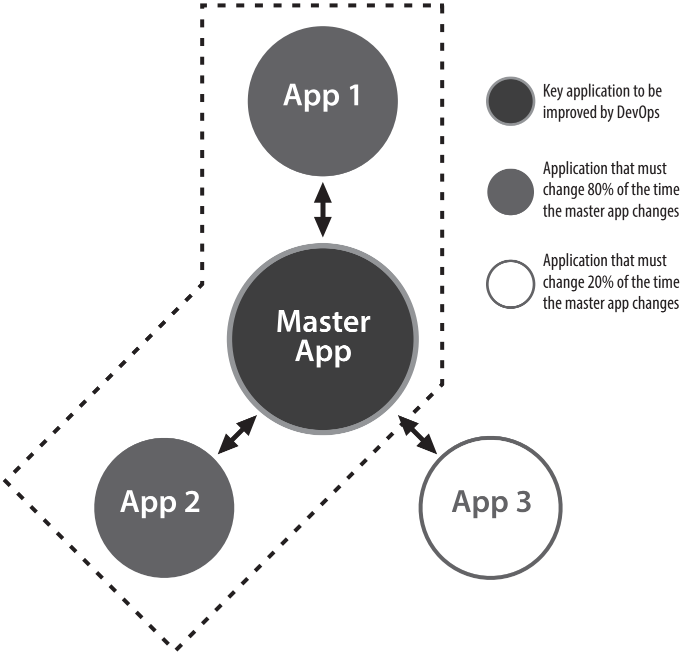

App 1
Master App
App 2
App 3
Key application to be improved by DevOps
Application that must change 80% of the time the master app changes
Application that must change 20% of the time the master app changes

Figure 2.2: Minimum viable cluster: Applies system thinking to application analysis

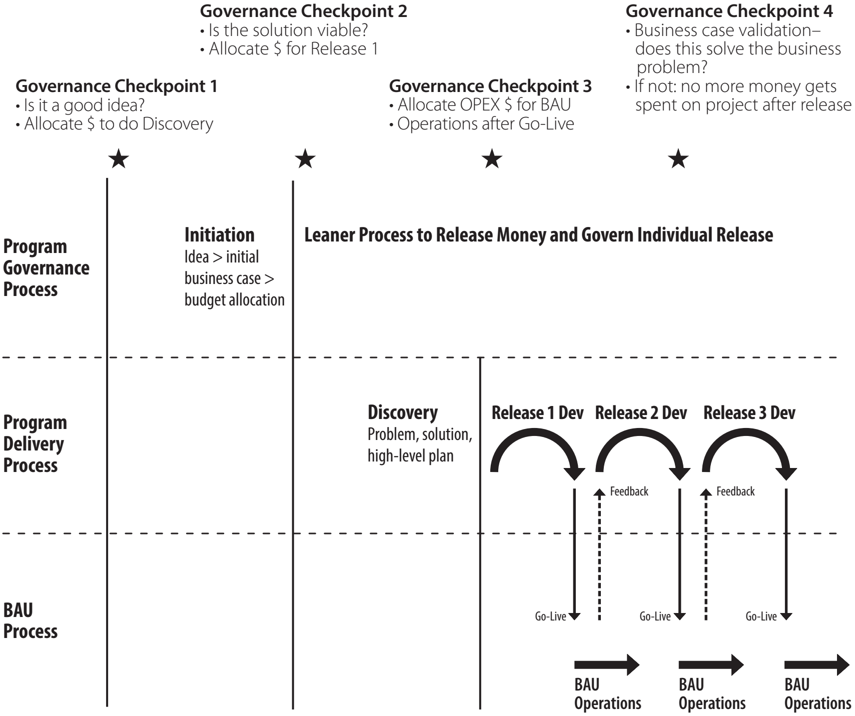

Governance Checkpoint 2
• Is the solution viable?
• Allocate $ for Release 1
Governance Checkpoint 4
• Business case validation– does this solve the business problem?
• If not: no more money gets spent on project after release
Governance Checkpoint 1
• Is it a good idea?
• Allocate $ to do Discovery
Governance Checkpoint 3
• Allocate OPEX $ for BAU
• Operations after Go-Live
Program Governance Process
Initiation
Idea > initial business case > budget allocation
Leaner Process to Release Money and Govern Individual Release
Program Delivery Process
Discovery
Problem, solution, high-level plan
Release 1 Dev
Release 2 Dev
Release 3 Dev
Feedback
Feedback
BAU Process
Go-Live
Go-Live
Go-Live
BAU Operations
BAU Operations
BAU Operations

Figure 2.3: Governance checkpoints: An Agile governance process with four checkpoints

First Steps for Your Organization

To support you in adopting what I have described in this chapter, I will provide two exercises for you to run in your organization. This time, both of them are highly related: the first is an analysis of your application portfolio and the second is the identification of a minimum viable cluster of application for which a capability uplift will provide real value.

Application Portfolio Analysis

If you are like most of my clients, you will have hundreds or thousands of applications in your IT portfolio. If you spread your change energy across all of those, you will likely see very little progress, and you might ask yourself whether the money is actually spent well for some of those applications. So, while we spoke about the IT delivery process in the chapter 1 exercises as one dimension, the application dimension is the second dimension that is important. Let's look at how to categorize your application in a meaningful way.

Each organization will have different information available about its applications, but in general, an analysis across the following four dimensions can be done:

- Criticality of application: How important is the application for running our business? How impactful would an issue be on the user experience for our customers or employees? How much does this application contribute to regulatory compliance?
- Level of investment in application: How much money will we spend in this application over the next 12–36 months? How much have we spent on this application in

the past? How many priority projects will this application be involved with over the next few years?

- Preferred frequency of change: If the business could choose a frequency of change for this application, how often would that be (hourly, weekly, monthly, annually)? How often have we deployed change to this application in the last 12 months?
- Technology stack: The *technology stack* is important, as some technologies are easier to uplift than others. Additionally, once you have a capability to deliver, for example, Siebel-based applications more quickly, any other Siebel-based application will be much easier to uplift too, as tools, practices, and methods can be reused. Consider all aspects of the application in this technology stack: database, data itself, program code, application servers, and middleware.

For each of the first three dimensions, you can either use absolute values (if you have them) or relative numbers representing a nominal scale to rank applications. For the technology stack, you can group them into priority order based on your technical experience with DevOps practices in those technologies. I recommend using a table with headings much like the one in Table 2.1. On the basis of this information, you can create a ranking of importance by either formally creating a heuristic across the dimensions or by doing a manual sorting. It is not important for this to be precise; we are aiming only for accuracy here.

It's clear that we wouldn't spend much time, energy, and money on applications that are infrequently changed—applications that are not critical for our business and on which we don't intend to spend much money in the future. Unfortunately, just creating a ranking of applications is usually not sufficient, as the IT landscape of organizations is very complex and requires an additional level of analysis to resolve dependencies in the application architecture.

#	Application	Technology	Strategic Application	Frequency of Charge	Size of the Application in the Investment Portfolio
95	App A	Java, .NET, Oracle	4–Critical	9	4–Very High

Table 2.1: Application analysis example: A table like this will help you structure the application analysis

Identifying a Minimum Viable Cluster

As discussed above, the minimum viable cluster is the subset of applications that you should focus on, as an uplift to these will speed up the delivery of the whole cluster. Follow the steps below to identify a minimum viable cluster:

1. Pick one of the highest-priority applications (ideally based on the portfolio analysis from the previous exercise) as your initial application set (consisting of just one application).
2. Understand which other applications need to be changed in order to make a change to the chosen application set.

3. Determine a reasonable cutoff for those applications (e.g., only those covering 80% of the usual or planned changes of the chosen application).
4. You now have a new, larger set of applications and can continue with steps 2 and 3 until the application set stabilizes to a minimum viable cluster.
5. If the cluster has become too large, pick a different starting application or be more aggressive in step 3.

Once you have successfully identified your minimum viable cluster, you are ready to begin the uplift process by implementing DevOps practices such as test automation and the adoption of cloud-based environments, or by moving to an Agile team delivering changes for this cluster.

		Product A	Product B	Product C
Functionality	Func. Area 1			
	Func. Area 2			
	Func. Area 3			
Architecture	Auto Scaling			
	Self-Healing			
	Monitoring			
	Changeability			
Engineering Capability	Source Code			
	APIs			
	Modularity			
	Cloud Enablement			
In-House IT Capability				

Table 3.1: Example scorecard: New applications should be evaluated on four dimensions, not just functionality

First Steps for Your Organization

Determine Guidelines for New Applications

Based on the sample scorecard provided in the Table 3.1, derive a scorecard for your organization. Take the next product decision (or a historic one if nothing is coming up) and apply this scorecard to see how it differs from your current process. Determine whether or not this scorecard, with an architecture focus, provides a different result. I would recommend inviting stakeholders across the organization to a workshop to discuss results and next steps to change your evaluation process going forward.

Strengthen Your Architecture by Creating an Empowering Ecosystem

So, you already have software packages in your organization like so many others. In the previous chapter, we did an analysis of your application portfolio, which you can leverage now to determine which software packages are strategic for your organization.

1. Based on the previous application portfolio analysis (or another means), determine a small subset of strategic applications (such as the first minimum viable cluster) to devise a strategy for creating an empowered ecosystem around them.
2. Now pick these strategic packages and run the scorecard from this chapter. You can largely ignore the functional aspects, as they are used more for the choice between package and custom software. You could, however, use the full scorecard in case you are willing to reconsider whether your current choice is the right one. Given that

you are doing this after the fact, you will already know how suitable the package was by the amount of customizations that your organization has already made.

3. Where you identify weaknesses in your software package, determine your strategy for them. How will you work with the software vendor to improve the capabilities? Will you work with them directly? Will you leverage a system integrator or engage with a user group?
4. Results take time. Determine a realistic review frequency to see whether or not your empowered ecosystem is helping you improve the applications you are working with. You can leverage the principles for measuring technical debt from the previous chapter as a starting point if you don't have any other means to measure the improvements in your packaged applications.

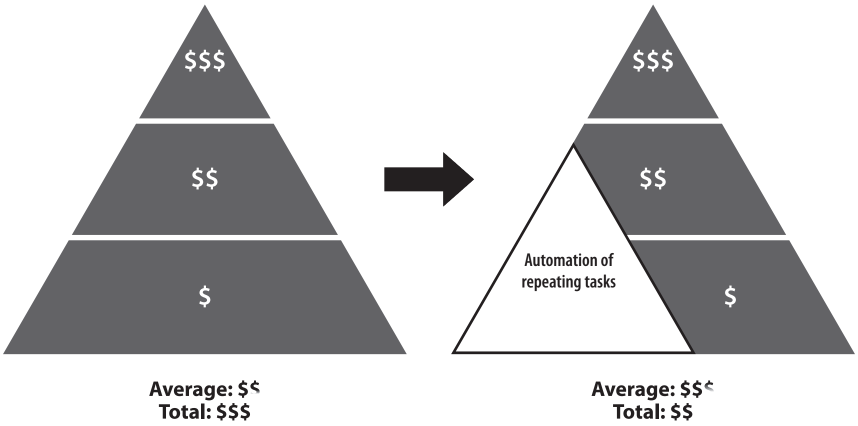


Figure 4.1: Overall costs versus daily rates: Automating work reduces overall cost but increases average cost rate

First Steps for Your Organization

Horses for Courses—Determining the Partners You Need

This whole chapter is about finding the right partner that fits your ambition and culture. But the truth is that you probably need different partners for different parts of your portfolio. If you have done the application portfolio activity in chapter 2, this exercise will be easier. There are three different types of applications for the purpose of this exercise:

- Differentiator applications: These applications are evolving very quickly, are usually directly exposed to your customers, and define how your company is perceived in the marketplace.
- Workhorses: These applications drive the main processes in your organizations, such as customer relationship management, billing, finance, and supply-chain processes. They are often referred to as enterprise or legacy systems. They might not be directly exposed to the customer, but the company derives significant value from these applications and continues to make changes to them to support the evolving business needs.
- True legacy: These applications are pretty stable and don't require a lot of changes. In general, they tend to support your more stable, main processes or some fringe aspects of your business.

Based on these classifications, review your partner strategy to see whether you need to change either the partner itself or the way you engage with the existing one. For the first two categories, you want to engage strategic partners. For legacy applications, you are looking for a cost-effective partner who gets paid for keeping the system running. The incentives for your strategic partners are different. Your partners for the workhorse applications should be evaluated by the efficiencies they can drive into those applications; for the differentiator applications, you want someone who is flexible and will co-invent with you. The outcome of this activity will feed into the second exercise for this chapter.

Run a Strategic Partners Workshop for Your Workhorse Applications

Organizations spend the majority of their money on their workhorse applications. This makes sense, as these applications are the backbone of the business. For this exercise, I want you to invite your strategic partners who support your workhorse applications (and possibly the differentiator ones) to a workshop. You can do this with all of the partners together, which can be more difficult, or by running separate workshops for each partner. It is important to tell them to assume that the current contract structure is negotiable and to be open-minded for the duration of the workshop.

The structure of this workshop should be as follows:

- Explain to your partner what is important for you in regard to priorities in your business and IT.
- Discuss how you can measure success for your priorities.

- Let your partner explain what is important for them in their relationship with you and what they require in their organization to see the relationship as successful.
- Workshop how you can align your interests.
- Brainstorm what the blocks are to truly achieve a win-win arrangement between your two organizations.

The key to this workshop is that both sides are open-minded and willing to truly collaborate. In my experience, it will take a few rounds of this before barriers truly break down—don't be discouraged if all of the problems are not solved in one workshop. Like everything else we talk about, it will be an iterative process, and it is possible that you will realize that you don't have the right partners yet and need to make some changes in the makeup of your ecosystem.

Do a Quick Self-Check about Your Partnering Culture

A quick test to evaluate your DevOps culture with your system integrator:

- Are you using average daily rate as indicator of productivity, value for money, and so on?
  +1 if you said no.
- Do have a mechanism in place that allows your SI to share benefits with you when he improves through automation or other practices?
  +1 if you said yes. You can't really expect the SI to invest in new practices if there is no upside for him. And yes, there is the "morally right thing to do" argument, but let's be fair. We *all* have economic targets, and not discussing

this with your SI to find a mutually agreeable answer is just making it a bit a too easy for yourself, I think.

- Do you give your SI the “wiggle room” to improve and experiment, and do you manage the process together?

  +1 if you said yes. You want to know how much time the SI spends on improving things by experimenting with new tools or practices. If she has just enough budget from you to do exactly what you ask her to do, then start asking for an innovation budget and manage it with her.
- Do you celebrate or at least acknowledge the failure of experiments?

  +1 if you said yes. If you have an innovation budget, are you okay when the SI comes back to let you know that one of the improvements didn’t work? Or are you just accepting successful experiments? I think you see which answer aligns with a DevOps culture.
- Do you know what success looks like for your SI?

  +1 if you said yes. Understanding the goals of your SI is important, not just financially but also for the people who work for the SI. Career progression and other aspects of HR should be aligned to make the relationship successful.
- Do you deal with your SI directly?

  +1 if you said yes. If there is another party involved, such as your procurement team or an external vendor, then it’s likely that messages get misunderstood. And there is no guarantee the procurement teams know the best practices for DevOps vendor management. Are you discussing any potential hindrance in the contracting space directly with your SI counterpart?

If you score 0–2 points, you have a very transactional relationship with your SI and should consider getting to know him or her better to improve the relationship. If you score 3–4 points, you are doing okay but with room for improvements, so you could run a partner workshop to address the other dimensions. If you score 5 or 6 points, you are up ahead with a real partnership that will support you through your transformation. Well done!

Figure 5.1: Discovery versus delivery: Discovery is like the first outline of a picture; delivery fills in the details

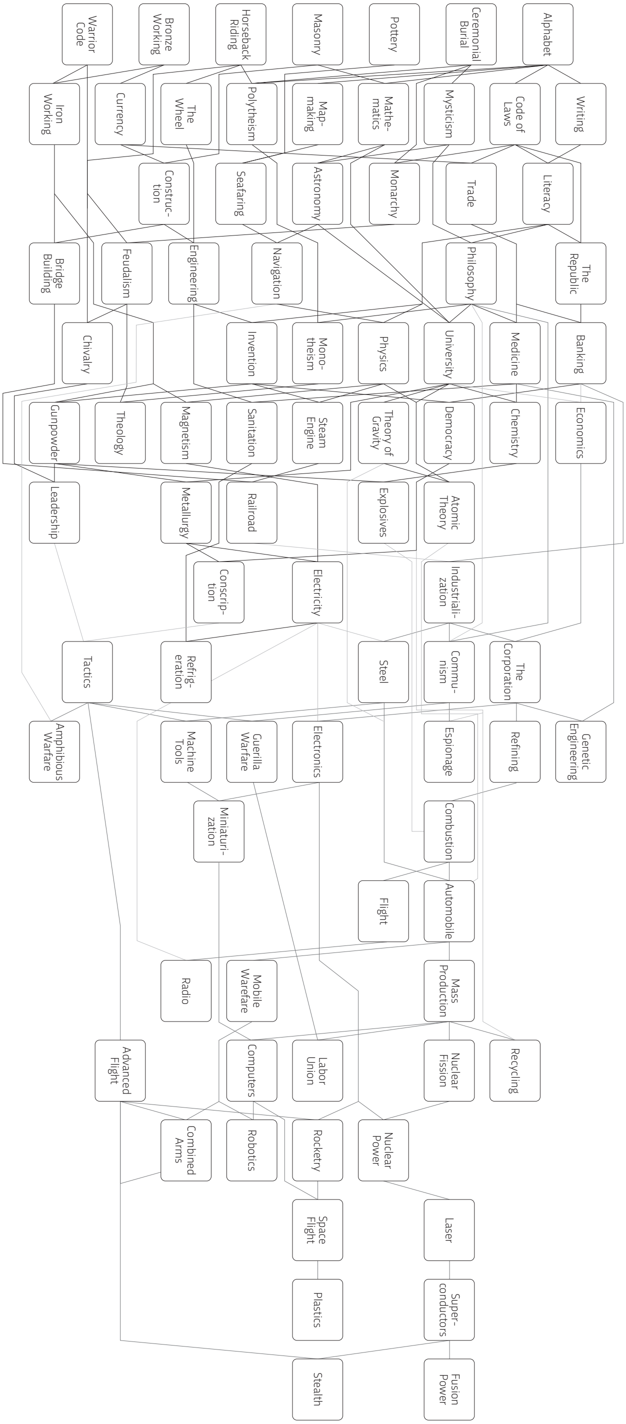


Figure 5.2a: Technology tree: Example showing dependencies between technologies

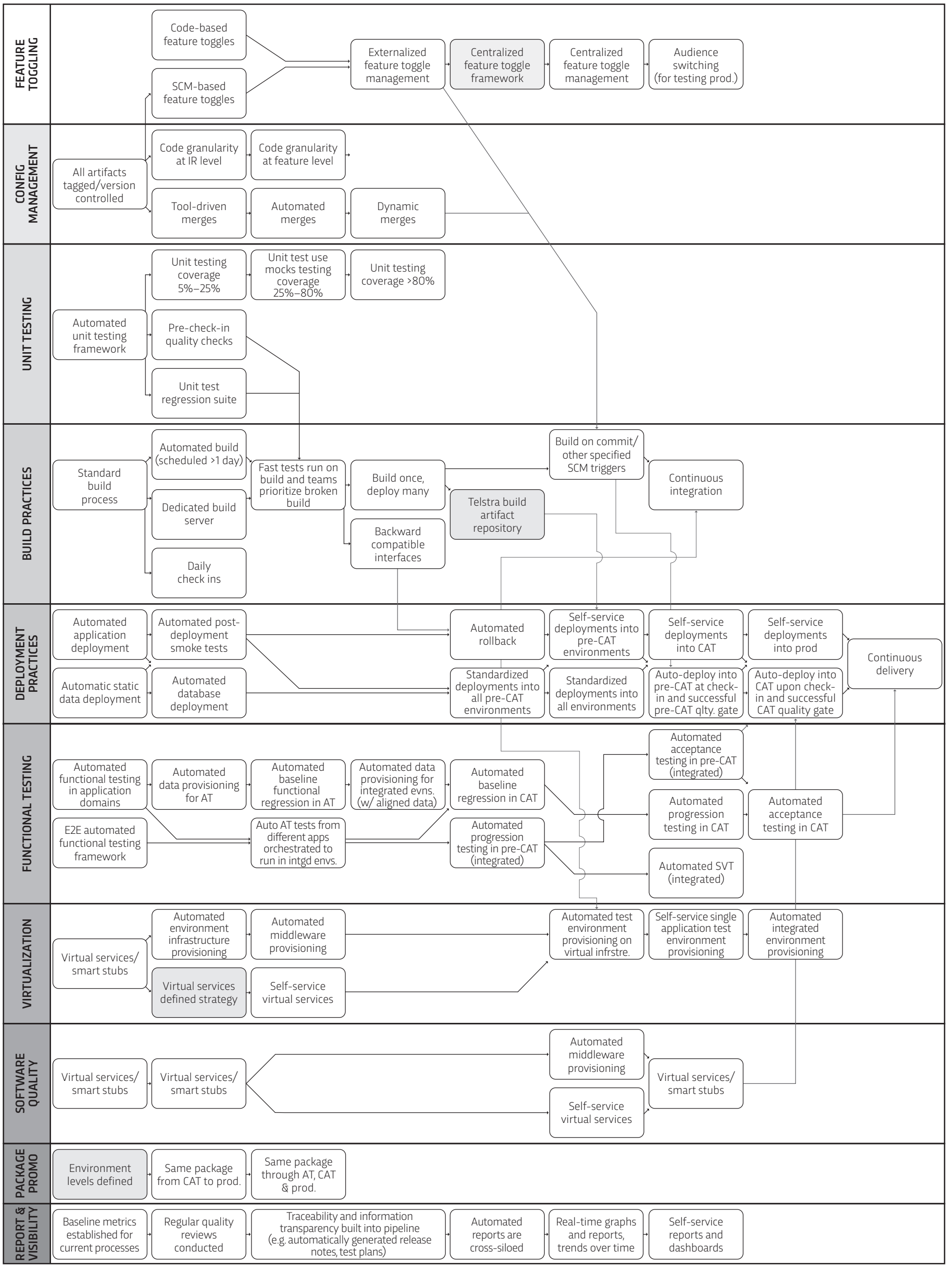

FEATURE TOGGLING
Code-based feature toggles
SCM-based feature toggles
Externalized feature toggle management
Centralized feature toggle framework
Centralized feature toggle management
Audience switching (for testing prod.)
CONFIG MANAGEMENT
All artifacts tagged/version controlled
Code granularity at IR level
Code granularity at feature level
Tool-driven merges
Automated merges
Dynamic merges
UNIT TESTING
Automated unit testing framework
Unit testing coverage 5%–25%
Unit test use mocks testing coverage 25%–80%
Unit testing coverage >80%
Pre-check-in quality checks
Unit test regression suite
BUILD PRACTICES
Standard build process
Automated build (scheduled >1 day)
Dedicated build server
Daily check ins
Fast tests run on build and teams prioritize broken build
Build once, deploy many
Backward compatible interfaces
Telstra build artifact repository
Build on commit/ other specified SCM triggers
Continuous integration
DEPLOYMENT PRACTICES
Automated application deployment
Automated post-deployment smoke tests
Automatic static data deployment
Automated database deployment
Automated rollback
Standardized deployments into all pre-CAT environments
Self-service deployments into pre-CAT environments
Standardized deployments into all environments
Self-service deployments into CAT
Auto-deploy into pre-CAT at check-in and successful pre-CAT qlty. gate
Self-service deployments into prod
Auto-deploy into CAT upon check-in and successful CAT quality gate
Continuous delivery
FUNCTIONAL TESTING
Automated functional testing in application domains
E2E automated functional testing framework
Automated data provisioning for AT
Automated baseline functional regression in AT
Automated data provisioning for integrated evns. (w/ aligned data)
Auto AT tests from different apps orchestrated to run in intgd envs.
Automated baseline regression in CAT
Automated progression testing in pre-CAT (integrated)
Automated acceptance testing in pre-CAT (integrated)
Automated progression testing in CAT
Automated SVT (integrated)
Automated acceptance testing in CAT
VIRTUALIZATION
Virtual services/ smart stubs
Automated environment infrastructure provisioning
Virtual services defined strategy
Automated middleware provisioning
Self-service virtual services
Automated test environment provisioning on virtual infrstre.
Self-service single application test environment provisioning
Automated integrated environment provisioning
SOFTWARE QUALITY
Virtual services/ smart stubs
Virtual services/ smart stubs
Automated middleware provisioning
Self-service virtual services
Virtual services/ smart stubs
PACKAGE PROMO
Environment levels defined
Same package from CAT to prod.
Same package through AT, CAT & prod.
REPORT & VISIBILITY
Baseline metrics established for current processes
Regular quality reviews conducted
Traceability and information transparency built into pipeline (e.g. automatically generated release notes, test plans)
Automated reports are cross-siloed
Real-time graphs and reports, trends over time
Self-service reports and dashboards

Figure 5.2b: DevOps technology tree showing dependencies

First Steps for Your Organization

Run a Discovery Workshop for an Initiative of Your Choice

I described the discovery workshop in some detail in this chapter, and I encourage you to run a session in your organization. Here is a sample agenda for a two-week long discovery workshop, which you can adjust as required for larger or smaller initiatives. The activities highlighted are just an example; there are many additional activities that you can embed in your discovery sessions to enhance the experience.

Part 1: Explore the Business Problem (Two Days)

- briefing by initiative sponsor (leveraging the business case where possible)
- current customer/stakeholder experience
- to-be customer experience workshop
- creating the mission statement
- success criteria definition
- in-scope and out-of-scope brainstorming
- stakeholder management workshop
- prioritization-scheme discussion
- risks, issues, and dependencies identification and mitigation

Part 2: Solutioning (Three Days)

- high-level to-be process design
- high-level technical architecture to support to-be design
- breaking the scope up into features
- identification of minimum viable product (MVP)
- selected deep dives into technologies and processes

Part 3: Planning for Delivery (Five Days)

- Agile training (if required)
- team structure definition
- estimation of scope
- high-level roadmapping
- technical preparation (environment, quality, configuration management strategies)
- setting up delivery governance
- social contract
- prepping the backlog
- discovery showcase to the organization
- planning workshop (PI planning when using SAFe)

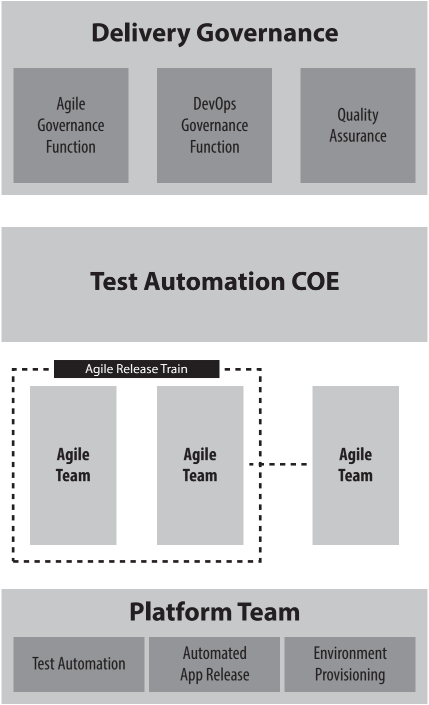
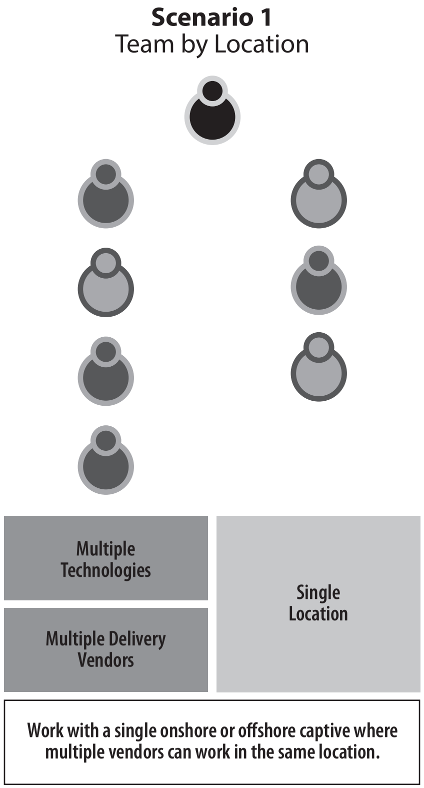
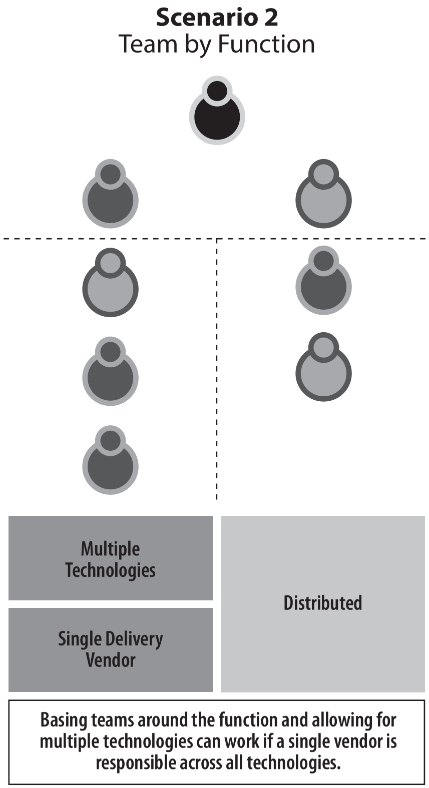


Figure 6.1:
Organizational structure starting point: This layered organizational structure works well in larger organizations



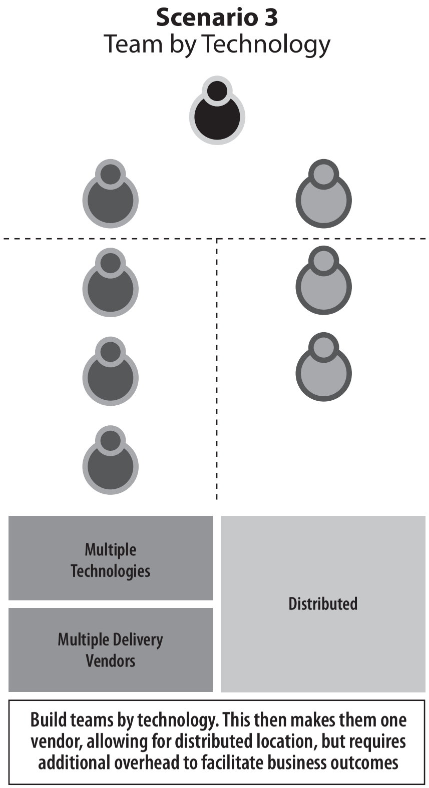
Work with a single onshore or offshore captive where multiple vendors can work in the same location.

Figure 6.2a:
Agile team scenario 1: Agile feature teams in one location can be from multiple vendors and support multiple technologies



Basing teams around the function and allowing for multiple technologies can work if a single vendor is responsible across all technologies.

Figure 6.2b:
Agile team scenario 2: Distributed Agile feature teams ideally consist of one vendor only



Build teams by technology. This then makes them one vendor, allowing for distributed location, but requires additional overhead to facilitate business outcomes

Figure 6.2c:
Agile team scenario 3: Agile component teams in multi vendor, multi location scenarios

First Steps for Your Organization

Identify One of Your Value Streams and the Required Teams to Support It

We have spoken about creating a value stream map before (in chapter 2). What we are looking for here is to identify value streams from a business perspective. Once you have identified the value stream, the next step is to identify the systems that support the value stream.

Looking at your backlog of work or your portfolio of initiatives, identify how much work impacts the systems supporting this value stream. On this basis, you should now be able to create a team structure supporting this value stream. It won't be perfect, and you will have to adjust it over time; but you now have a starting point. Using the SAFe terminology, you have identified an Agile release train, and you can now go on to deliver work through this team structure to support the value stream. Over time, you will be able to shift the budgeting model to support the teams as I mentioned earlier in the chapter.

Identify the Teams That Will Be Impacted by the Move to a Platform Team

The platform team is a concept that is very transformational, and the change required is often underestimated. To help you navigate this change, I want you to identify all the teams that are currently performing functions that would ultimately be performed by the platform team or would be impacted by the platform team (e.g., infrastructure teams, testing teams, database administrators [DBAs]). Invite them to a workshop to discuss what the delivery platform at your organization should look like from a functional perspective. Once you have a level of agreement on that, discuss how the delivery platform should be supported. Hopefully the platform team emerges as something that everyone can agree on. Then agree on next steps to get closer to achieving this end-state vision.

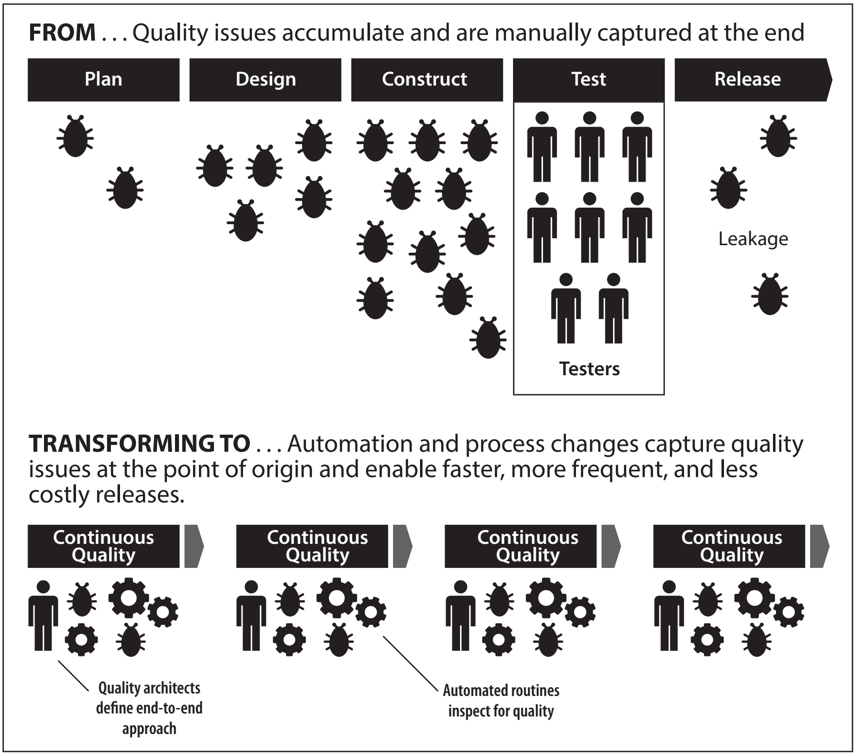

FROM . . . Quality issues accumulate and are manually captured at the end
Plan
Design
Construct
Test
Release
Testers
Leakage
TRANSFORMING TO . . . Automation and process changes capture quality issues at the point of origin and enable faster, more frequent, and less costly releases.
Continuous Quality
Continuous Quality
Continuous Quality
Continuous Quality
Quality architects define end-to-end approach
Automated routines inspect for quality

Figure 7.1: Quality engineering process: Quality engineering shifts the focus to the whole delivery lifecycle

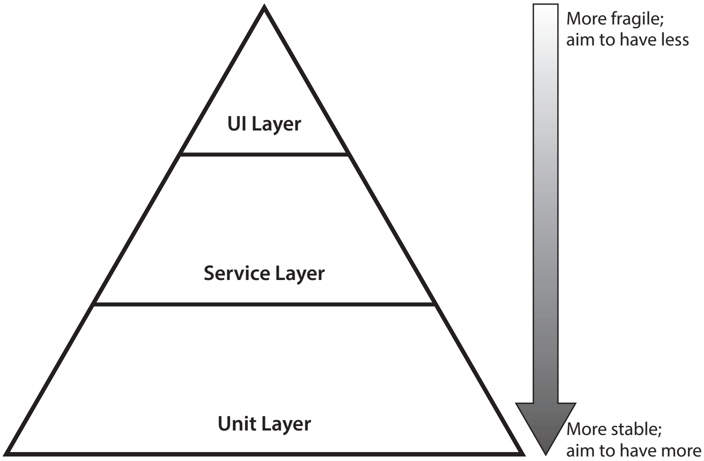


Figure 7.2: Test automation pyramid: The slower the layer, the less we should use it in automating tests

First Steps for Your Organization

Mapping the Quality Process of Your Organization

This activity is somewhat similar to the value mapping we did in chapter 1. Here again, you should prepare a whiteboard or other wall space, and be ready with Blu Tack and system cards.

First, create a high-level work flow on the wall, showing requirements to production, including all the relevant process steps represented on the cards. Then use a different color of cards or pen and list each quality activity where it happens in the life cycle.

To make sure the picture is complete, ask yourself whether you have covered all concerns, including performance, security, availability, and any other concerns. It's okay if you have some aspects missing from the original list; simply highlight these on the side.

As a next step, your team needs to think about automation: What can be automated and hence be done earlier and more frequently? Consider breaking up activities into an automated part that can be completed earlier and more frequently, as well as a part that continues to require a manual inspection. Remember that automated activities are nearly free of effort once implemented, so doing them more often does not add much cost.

Now, you want to identify opportunities to check for quality aspects earlier, as this relates to manual inspection. For each activity think of possible ways to do the full scope or at least aspects of it earlier.

Finally, create a backlog in which your teams can make the required shifts toward quality engineering, and then start making the shift.

Measuring Quality

As discussed in this chapter, measuring quality is not easy. Many measures are only valid temporarily, while you are addressing specific concerns. Sit down with your quality or testing leadership and, on a piece of paper, list out at all the metrics and measures you use to determine quality. Then determine which ones are objective and automated.

If you don't have a good set of automated and objective metrics, then workshop how to get to a small set of these. I think two of the easy ones to agree on are duration for a successful regression test run and incidents found in production per time period. These are pretty noncontroversial and applicable to all kinds of companies, but you will want to define a small number of additional metrics relevant to your business.

First Steps for Your Organization

Set Up One-On-Ones

It is important to find time in your calendar for each of your directs starting two weeks from now, and be sure to make them recurring meetings on a weekly basis. Make them thirty minutes each, and set the agenda as fifteen minutes for the direct first and fifteen minutes for you second. It is common that the direct may run over with his/her fifteen minutes, and that's okay. You can find another chance during the week to update the direct with any information that you didn't have time to share during your portion of the one-on-one. I also highly encourage you to take notes and to follow up on the points discussed in the previous week; this will provide a very rich background of information when it comes to performance discussions, providing a progress measure for your direct report.

Define Culture KPIs for Your Managers

You probably have heard the saying "you get what you measure." Though culture is somewhat evasive to metrics, there are some things you can do:

- Leverage the internal NPS that I highlighted in this chapter, and break it down by team as a good high-level measure.
- Measure one-on-ones of your managers. This allows you to measure whether or not your manager builds effective relationships.

- Spot-check the strength of the relationships of your managers. Ask them about some of the fundamental and noninvasive things a boss should know about their people if they have a positive relationship. For example, do they know the names of their directs' kids and partners, and their directs' favorite pastimes?

And yes, you should do the same for yourself to avoid looking for the proverbial speck of sawdust in your manager's eye while ignoring the wooden beam in your own.

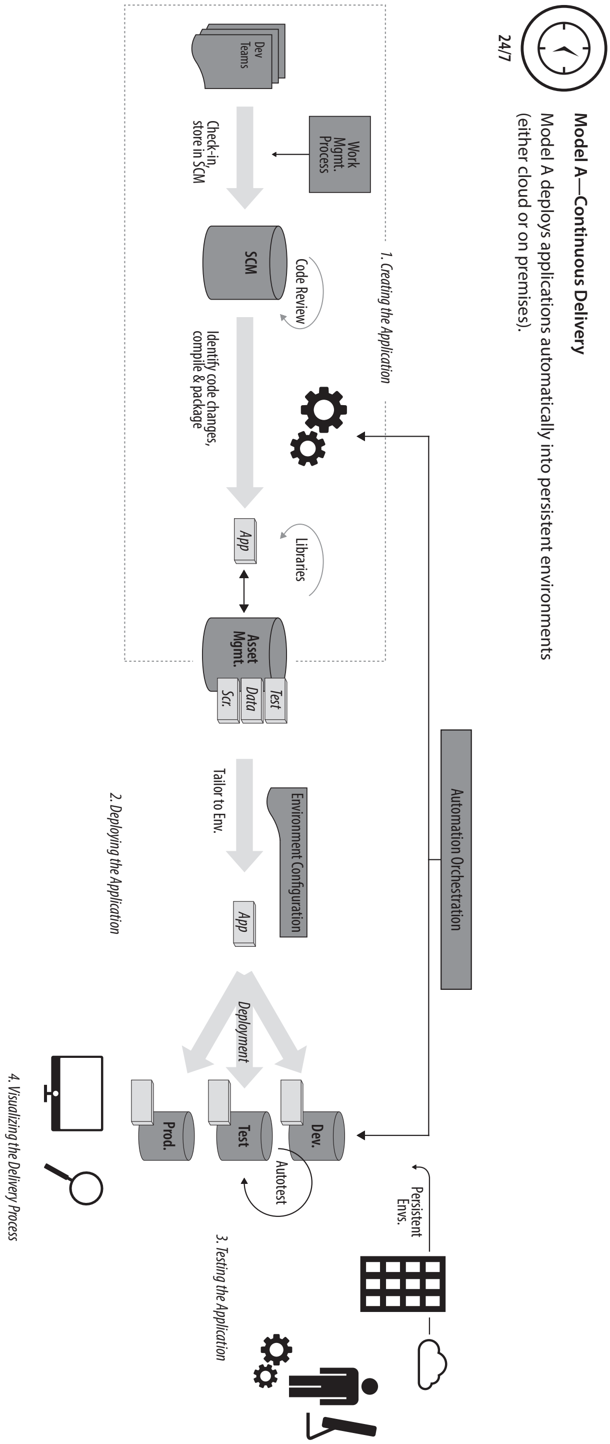

24/7
Model A—Continuous Delivery
Model A deploys applications automatically into persistent environments (either cloud or on premises).
1. Creating the Application
Dev Teams
Work Mgmt. Process
Check-in, store in SCM
SCM
Code Review
Identify code changes, compile & package
App
Libraries
Asset Mgmt.
Test
Data
Scr.
Automation Orchestration
2. Deploying the Application
Environment Configuration
Tailor to Env.
App
Deployment
Dev.
Test
Prod.
Autotest
3. Testing the Application
Persistent Envs.
4. Visualizing the Delivery Process

Figure 9.1: Model A—Continuous delivery: Continuous delivery automates delivery to persistent environments

Model B—Cloud-Enabled Delivery

Model B deploys applications automatically after provisioning a new environment from the cloud or data center.

Main difference compared to Model A is the maturing of the infrastructure practices—infrastructure as code.

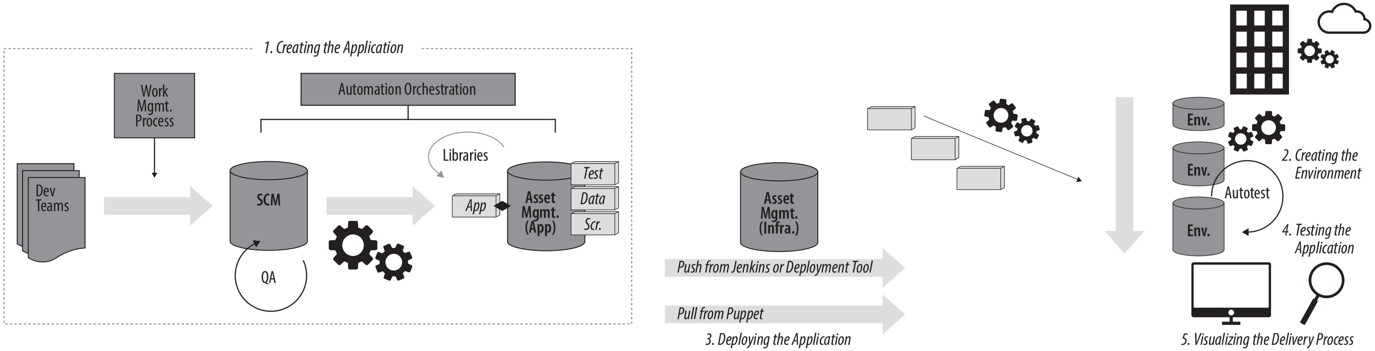

1. Creating the Application
Work Mgmt. Process
Automation Orchestration
Dev Teams
SCM
QA
Libraries
App
Asset Mgmt. (App)
Test
Data
Scr.
Asset Mgmt. (Infra.)
Push from Jenkins or Deployment Tool
Pull from Puppet
3. Deploying the Application
Env.
Env.
Env.
2. Creating the Environment
Autotest
4. Testing the Application
5. Visualizing the Delivery Process

Figure 9.2: Model B—Cloud-enabled delivery: Cloud-enabled delivery creates a new environment with each deployment

Model C—Container-Enabled Delivery

Model C deploys applications as a set of containers into one or more hosts that are dynamically created.

Main difference compared to Model B is the maturity of the container practices and the more modular application architecture.

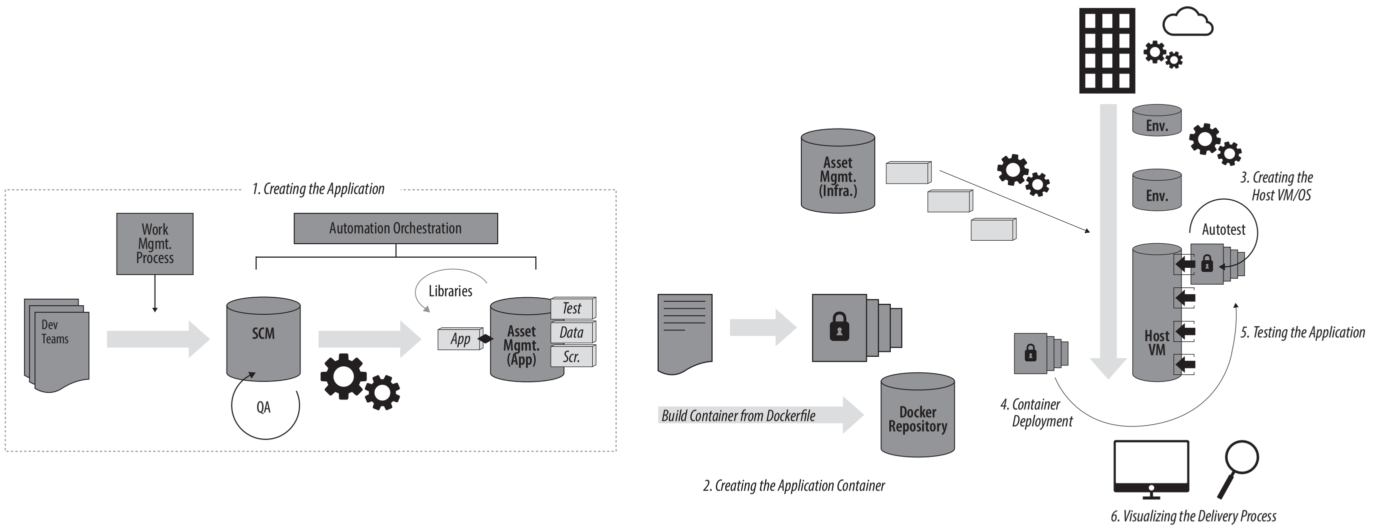

1. Creating the Application
Work Mgmt. Process
Automation Orchestration
Dev Teams
SCM
QA
Libraries
App
Asset Mgmt. (App)
Test
Data
Scr.
Build Container from Dockerfile
Docker Repository
2. Creating the Application Container
Asset Mgmt. (Infra.)
Env.
Env.
3. Creating the Host VM/OS
Autotest
Host VM
4. Container Deployment
5. Testing the Application
6. Visualizing the Delivery Process

Figure 9.3: Container-enabled delivery manages an application in containers

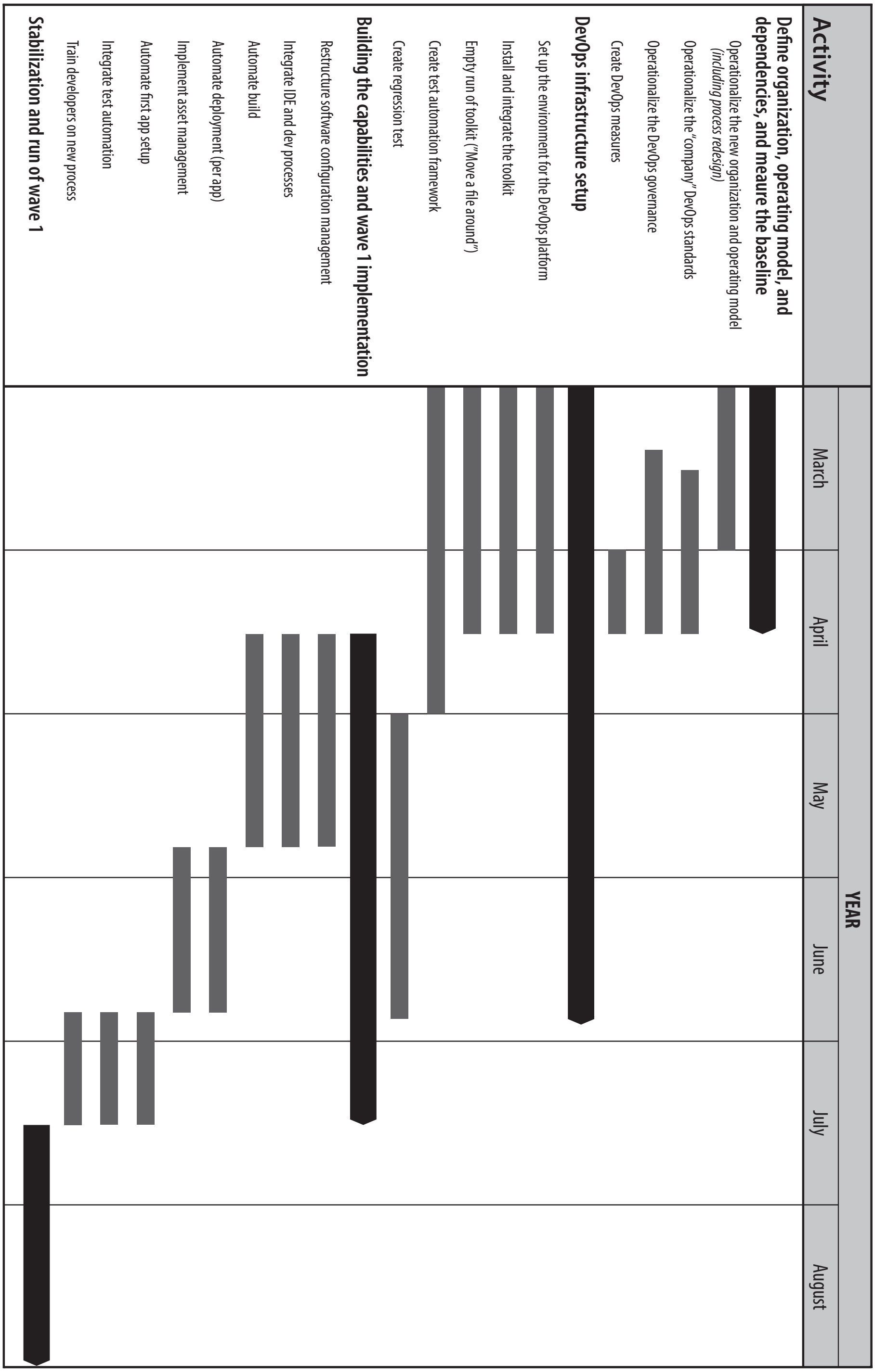


Figure 9.4: Sample plan for initial build of capabilities: Container-enabled changes and infrastructure setup are common first steps

First Steps for Your Organization

Map Your Application Delivery Models

As I described above, it is not advisable to push all applications into a container-enabled delivery model, as it would not be economical or feasible. In organizations with a large amount of legacy, you will probably have the largest proportion, targeting continuous delivery and cloud-enabled delivery with some container-enabled delivery in your digital applications. And that is realistic. Remember that the goal is to get better; too often, we make perfect the enemy of better. With this in mind, run a workshop where you review your applications and define what your current and ideal delivery model is for each application. You will need to bring people from your infrastructure, your architecture, and your delivery organization into the same room for this. Then do a fit/gap analysis of the capabilities required for the delivery model you assign to each application. Brainstorm a set of initiatives to build the capabilities that are missing. Often, you can reuse capabilities for applications of the same technology stack (e.g., Java) once they are built for another application. Identify those opportunities for reuse. With all these in mind, define a six-month roadmap, and review the roadmap and progress on a monthly basis to reprioritize based on the lessons learned so far.

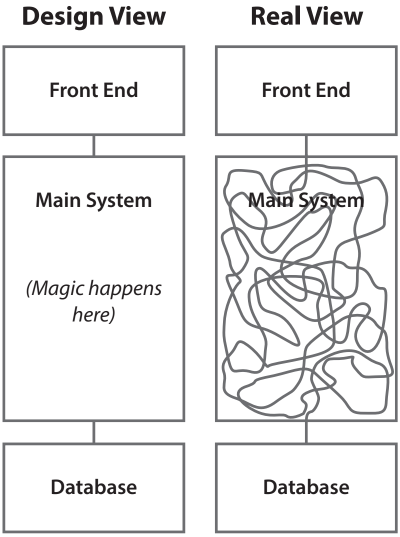


Figure 10.1:
Design view versus real view:
Simple diagrams do not equal simple architectures

First Steps for Your Organization

Identify Your Architecture Evolution Strategy

Invite your architects to a workshop about architecture strategies. Let them explain what the current plan is, and try to map this back to the architecture evolution strategies I have highlighted in this chapter: decoupling your architecture; removing technical debt; creating a new architecture on the side or eroding the old architecture core. Then discuss alternative approaches with them and see whether you can come to an aligned strategy that will provide more decoupled services over time. Make sure that with this architecture evolution strategy, related capabilities are covered too. Refer back to the delivery models and their associated capabilities to make sure your architects are not talking just about the system blueprints but also about the capability build-out to support the architecture with the right engineering practices.

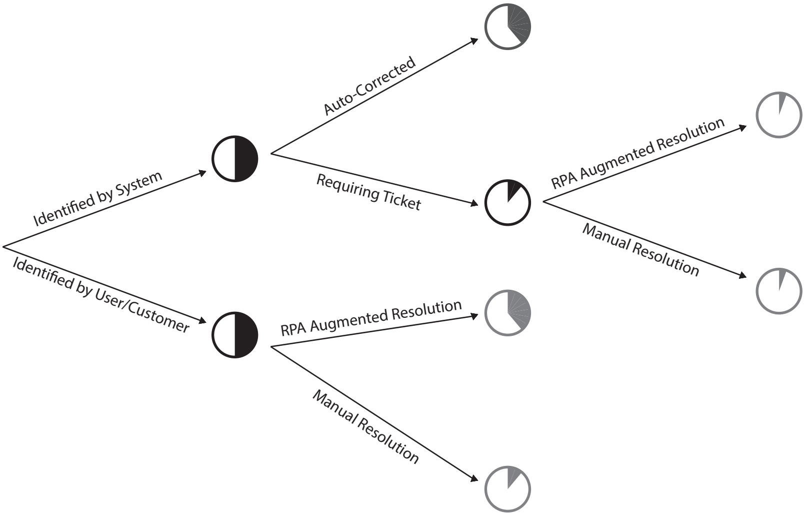


Figure 11.1: Advanced maturity state: Modern operations works based on the principle to minimize work that needs to be done

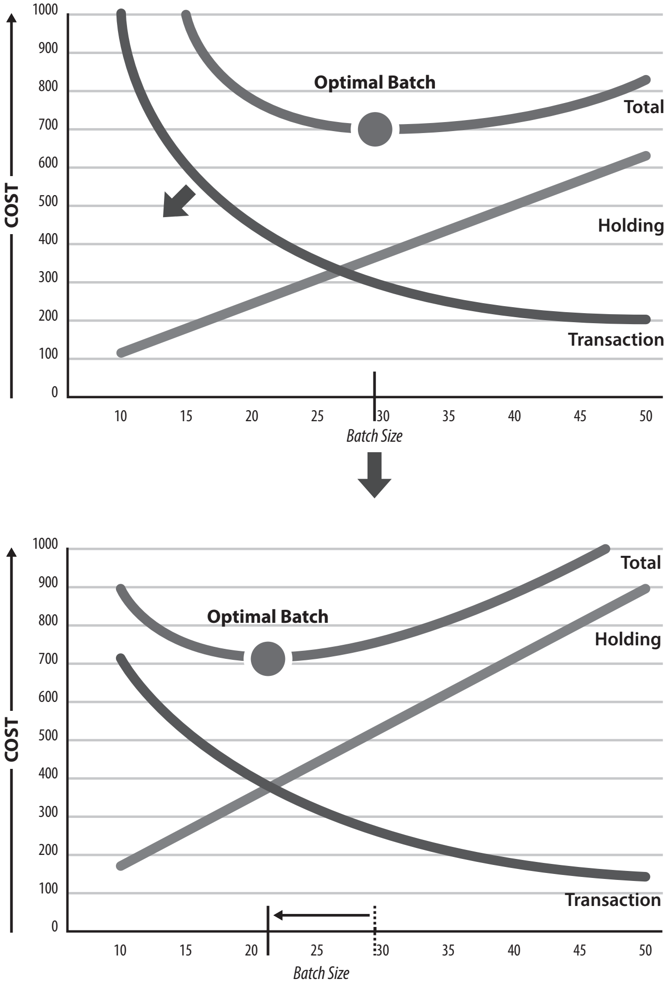


Figure 11.2: Reducing transaction costs enables smaller batch sizes

First Steps for Your Organization

Run Problem-Ticket Analysis

In this exercise, we will look at ways to improve your application operations through analysis of your problem tickets to identify what can be automated. As I said before, you have a lot of data that you don't use to its full potential. Get your hands on this problem-ticket data, and run some simple analysis over it. I suggest using a word cloud to identify common wording (e.g., "restart server," "reset password," "out of storage"); then try to categorize on that basis. Once you have done that, you can go through the ones with the highest count to see what can be done to improve the system by resolving it either automatically or with the support of automation. It usually takes two to three rounds of refinement before the categorization—based on wording and other metadata—is accurate enough for this analysis. This will give you the starting point for your automated production system that can self-correct over time.

Review Your DevOps Tools

With the principles of good DevOps tooling in mind (strong APIs, configuration as code, supportive licensing model), sit down with the architects in your organization and make a list of the tools you are using for DevOps capabilities. You will be surprised by how many tools you have and how many of them are overlapping in regard to their functionality. Analyze the tools for how future-ready they are (utilizing Table 11.1, which you can enhance with further criteria specific to your context), and define your weak spots, where you have tools that are really not compatible with the DevOps way of working and are holding you back. Identify a strategy to replace these tools in the near future.

Criteria	**Tool A**	**Tool B**
API support		
Configuration management		
Multienvironment / code-branch support		
License model		
Data access		

Table 11.1: DevOps tools review: DevOps tools should follow DevOps good practices themselves

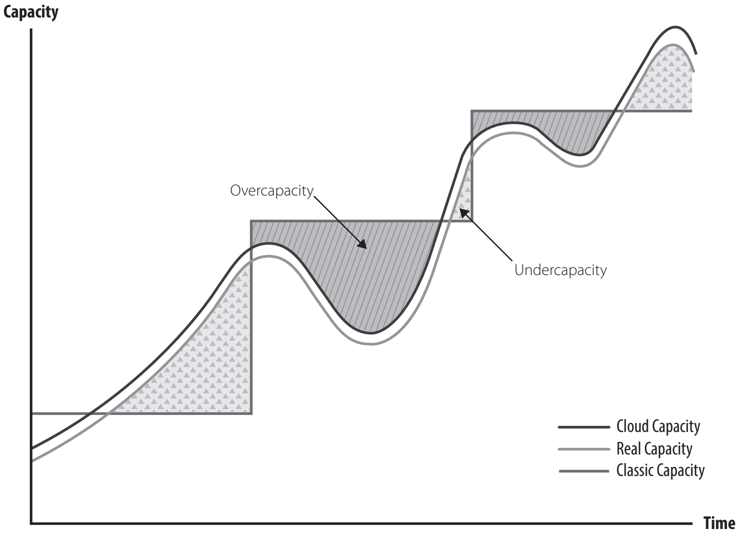


Figure 12.1: Capacity versus time: When done correctly, cloud capacity moves according to the need

First Steps for Your Organization

Review Your Cloud Applications

With the understanding of how the cloud is benefiting you most (based on the two factors—granularity of the architecture and maturity of the application in regard to DevOps practices), review your existing cloud applications (or the ones you are planning to move). To do this, first analyze the architecture to identify the components that are decoupled from each other and have the potential to be scaled independently. Also identify services that should be decoupled for later architectural refactoring. For each of the decoupled components, review their maturity of DevOps practices (SCM, build and deployment management, test automation) to identify gaps that require closing.

Next, ask yourself whether you would really benefit from the flexibility of the cloud for these applications, because you can leverage the elasticity of the architecture. Only if you have the right architecture and automation capabilities will you be able to fully benefit from the cloud. You should start building these capabilities—either before moving to the cloud or once you are in the cloud—to reduce the cost of your cloud infrastructure and the risk of business disruptions from application issues.

Based on this analysis, you will have a list of applications that are cloud ready and a backlog of work to make more and more applications cloud ready through architecture refactoring and the building of additional DevOps capabilities.

Plan a Cloud Disaster Event

Pick a scenario (e.g., your cloud provider going bust and you losing access to all the systems and data stored on the cloud) and run a full rehearsal of what it would take to come back online. This will include activities such as creating a new infrastructure with a different cloud provider, installing the applications you need to run your business, and restoring data from an external backup. There are two things that you want to do:

1. Identify your weak spots and prioritize them to improve your cloud architecture.
2. Measure the impact and duration of your rehearsal so that you can study how you become better over time.

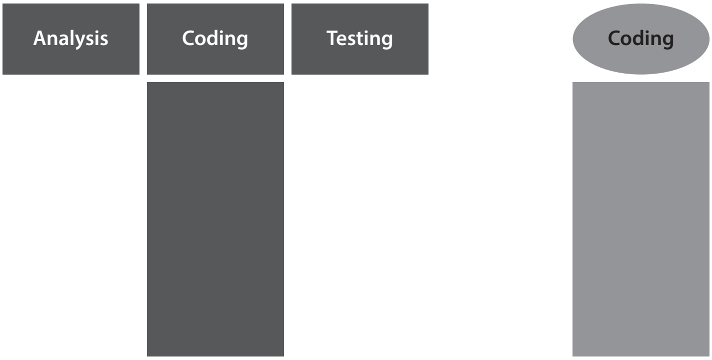


Figure A.1: T-shaped skills: T-shaped employees have broader skills than I-shaped employees

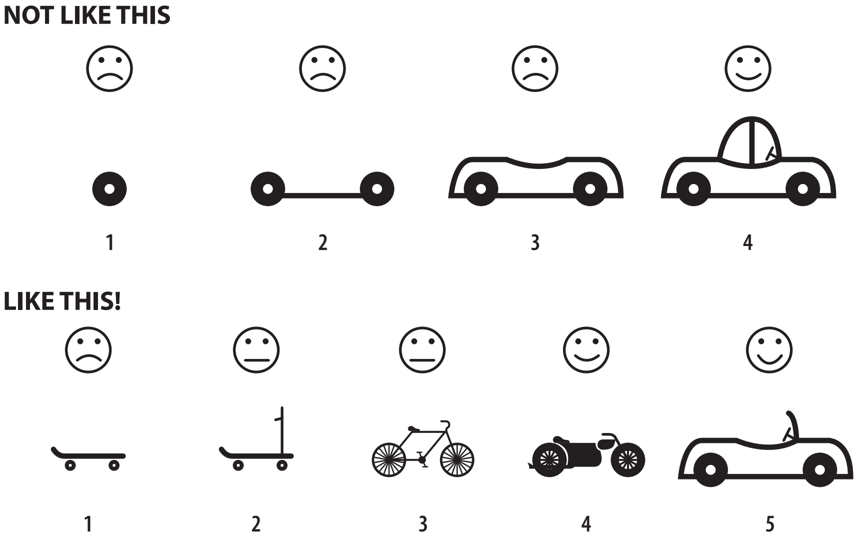


Figure A.2: Iterative versus incremental delivery: Iterative delivery slowly increases the benefits of the product, while incremental requires the full product before it is useful

(Recreated based on image by Henrik Kniberg, "Making Sense of MVP (Minimum Viable Product)—and why I prefer Earliest Testable/Usuable/Lovable," *Crisp's* Blog, January 25, 2016, http://blog.crisp.se/2016/01/25/henrik kniberg/making-sense-of-MVP.)

Resources

Books

Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation by Jez Humble and David Farley: a very technical reference book on how to implement continuous delivery

Leading the Transformation: Applying Agile and DevOps Principles at Scale by Gary Gruver and Tommy Mouser: a great book with very pragmatic advice on how to start the transformation for IT.

The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations by Gene Kim, Jez Humble, Patrick Debois, and John Willis: this book provides a lot of useful guidance on implementing DevOps practices.

The Effective Manager by Mark Horstman: this is a great book on good management that focuses on the people who work for you.

The Goal: A Process of Ongoing Improvement by Eliyahu M. Goldratt and Jeff Cox: an easy-to-read business novel introducing you to systems thinking

The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses by Eric Ries: Eric describes how structured experimentation allows you to better solve business problems.

The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win by Gene Kim, Kevin Behr, and George Spafford: this one is an easy-to-read novel introducing you to DevOps concepts and culture.

The Principles of Product Development Flow: Second Generation Lean Product Development by Donald G. Reinertsen: a great book including some of the best discussion of batch size

Site Reliability Engineering: How Google Runs Production Systems by Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy: learn about modern operations for applications inspired by Google.

Podcasts and Online Resources

The Agile Revolution: an Australia-based Agile podcast (TheAgileRevolution.com)

Arrested DevOps: a podcast that provides information on upcoming conferences as well as discussion on DevOps topics (ArrestedDevOps.com)

Career Tools: a helpful podcast from Manager Tools designed to offer advice to anyone at any point on their career path (https://www.manager-tools.com/all-podcasts?field_content_domain_tid=5)

DevOps Café: a conversational-style podcast about all things DevOps (DevOpsCafe.org)

The Economist Radio: a daily podcast for staying up to date with science and politics (https://radio.economist.com/)

Freakonomics Radio: a podcast of surprising insights from science into the world around us (http://freakonomics.com/archive/)

HBR IdeaCast: a business-focused podcast from *Harvard Business Review* that dives deep into one specific area per episode (http://feeds.harvardbusiness.org/harvardbusiness/ideacast)

Manager Tools: great guidance for managers and directors (https://www.manager-tools.com/all-podcasts?field_content_domain_tid=4)

The Ship Show (now defunct, but episodes are still out there): one of the earlier DevOps podcasts (TheShipShow.com)

Software Engineering Radio: an in-depth podcast on technical topics (http://www.se-radio.net/)

TED Talks: inspirational talks that often cover science and technology (https://www.ted.com/talks)

Glossary

abstract environment configuration: variables like IP addresses and server names need to be abstracted so that configuration files only contain placeholders and not the actual values.

abstraction layer: an abstraction layer decouples two layers of architecture so that they can evolve independently from each other without being tightly coupled and causing dependencies.

access layer: usually a user interface that makes accessing information in the underlying systems easier and more user friendly than direct-access systems.

application programming interface (API): is a set of clearly defined methods of communication between various software components that allows access to functionality from external systems.

average daily rate (ADR): the average cost for a day of work across a team of resources with several daily cost rates per person.

bimodal IT: a concept introduced to demonstrate that newer, more modern IT systems are being developed differently from older systems.

black box mode: a type of IT delivery for which the customer does not care about the means of delivery and is only interested in the outcome.

blameless postmortem: a review technique that focuses on systematic problems and is purposefully not looking to attach blame to an individual.

build artifacts: the result of the build process, often as a binary that can then be used to deploy an application.

business IT isomorphism: the organizational approach to align IT functions with business functions to simplify the engagement model between business and IT.

canary testing: inspired by the canary in the coal mine, this approach deploys into a subset of production to validate the application before rolling it out more widely.

cloud: the practice of using a network of remote servers hosted on the internet to store, manage, and process data rather than using a local server.

cloud native application: an application built specifically to leverage the abilities of cloud computing and hence be more resilient and efficient.

compilers: utilities that “translate” programming code into executable programs.

compute environments: application environments that allow you to execute programs.

concept of error budgets: budgets that, instead of the more traditional costs, allocate a level of errors or outages to teams that they have to manage in order to be seen as successful.

configuration drift: when a server configuration starts to drift away from its intended configuration by either human or system intervention.

consumable services: IT services that can be called upon by other programs and that provide easy-to-consume interfaces as a contract of engagement.

container images: representations of an application in a container that can be deployed into a container engine for fast creation of the application.

continuous delivery: an IT practice made popular by a book of the same name in which software is automatically evaluated for quality and deployed into environments all the way up to production.

continuous integration: is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

cost performance indicator (CPI): a metric that allows you to measure how much work has been performed for a certain amount of cost.

COTS product: commercial-off-the-shelf products are preconfigured IT applications that support certain business processes without much configuration or programming required.

CRM system: a customer relationship management system allows a company to engage with its customers in a consistent way and leverage the relationships for further business cutover.

cycle time: the total time from the beginning to the end of your process, as defined by you and your customer.

dashboard: a visualization of several data points or reports aggregated across several data sources so that the information is easily available.

decoupling of systems: a technique that enables systems to be changed independently from each other by introducing interfaces that remain stable when each of the systems changes.

defect density: a metric that measures how many defects per day of programming or line of code are introduced.

definition of done: an Agile practice that defines the exit criteria under which user stories are considered to be done. (See also *definition of ready*.)

definition of ready: an Agile practice that defines the entry criteria for user stories to be considered for the next sprint. (See also *definition of done*.)

develop-operate-transition (DOT) contracts: a popular contract structure that differentiates between vendors who deliver a solution, vendors who operate the solution, and the state in which the organization transitions the solution back in-house.

DevOps tool chains: the set of tools that supports DevOps practices like configuration management, deployment automation, and test automation, among others.

discovery: a phase in the beginning of an Agile project used to align all stakeholders on the intended outcome of the project and the way the team will achieve this.

discovery showcase: a meeting at the end of discovery in which the results of the phase are shared with a broader set of stakeholders.

end-state architecture: the envisioned end state of the application architecture that will fully support the business of an organization.

ERP system: an enterprise resource planning system is the practice of managing all the resources for the production and fulfillment process of an organization.

front-end team: the team that delivers the front-end experience with which the end customers interact.

function call: the programming technique that allows the usage of functions provided by other applications or parts of the same application.

function points: an estimation technique that aims to provide an objective way to measure work in IT projects.

go-live: the release of new functionality or a new program when it is ready for customer use.

graceful degradation: a practice that allows systems to provide basic functionality even when core processes are not available; this is in contrast to being completely unavailable when one function fails.

green field setting: a project setting in which the team can start from scratch instead of having to consider existing applications.

hardening: an Agile project phase right before production deployment in which additional testing (such as performance and security testing) takes place that was not able to be accommodated in the Agile sprints.

horizontal scaling: a scaling technique in which additional workflows are distributed to more systems of the same size and shape instead of providing more resources to the same systems.

IDE extensions: extensions that are provided for developers in their integrated development environment to support specific programming languages with helpful utilities.

internet natives: companies that were built around solutions that leverage the internet and hence considered internet capabilities in the application architecture from the inception of the service.

iterative character: something that evolves over several iterations, with each iteration bringing it closer to the real answer.

Jenkins: a continuous-integration tool.

lead time: the time between the initiation and the completion of a production process.

legacy: in the context of IT, this describes applications that were built in the past and need to be maintained.n (See also *true legacy*.)

LeSS: Large-Scale Scrum, a scaling method for Agile.

mean time to discovery (MTTD): the mean time it takes to identify that a problem exists. (See also *mean time to recovery*.)

mean time to recovery (MTTR): the mean time it takes to rectify a problem. (See also *mean time to discovery*.)

mental models: the models that humans use to understand the world and make decisions, often leveraging heuristics, as too much information is available for full evaluation.

microservices: an architecture paradigm that tries to identify the smallest possible independent component that can run as a service.

middleware: software that runs in between the operating system and applications (i.e., integration services and data access layers).

minimum viable cluster: the minimum set of applications that can be changed so that a real positive impact can be made by uplifting the DevOps capabilities.

minimum viable product (MVP): a product that has only the absolute necessary features to validate its viability with customers. Additional scope is then built out over time.

monolithic applications: applications that provide many different services that can only be deployed together.

multimodal IT: an IT environment that leverages several modes of delivery across the Agile and Waterfall spectrum.

NPS: the net promoter score is a metric that measures the satisfaction of a constituency with a service provider or organization.

open source: a software distribution model that does not require payment for usage and is based largely on voluntary contribution to the source code.

Perl: a scripting language often used for automating tasks.

PI planning: program increment planning is an implementation of large-room planning in the Scaled Agile framework, which brings all stakeholders together for a combined planning event of the next planning cycle.

PMI: Project Management Institute runs a training and certification program for project managers.

program increment: a planning duration that consists of several sprints/ iterations. Usually consists of around five sprints and is around three months long.

pulling model: an interaction model between services in which the consuming service pulls information rather than providing a queue in which the production service pushes information.

queuing theory: a scientific approach to understanding how queues behave.

refactoring: a programming practice that allows programmers to improve the structure of a program without making functional changes.

regression suite: a set of tests that ensure that previously implemented functionality continues to work.

release train: a superstructure (team of teams) that delivers to a common outcome, usually a team of 3–12 Agile teams.

robotic process automation (RPA): a tooling technique that provides utilities to automate tasks in applications that would otherwise have to be performed manually by human beings.

scale up/vertical: scaling techniques that add additional compute or other services to the same instance instead of distributing workflows to additional instances.

Scaled Agile Framework (SAFe): a popular scaling framework for Agile delivery.

schedule performance indicator (SPI): a metric measuring whether or not a project is on schedule according to a predefined plan.

shell scripts: a popular automation technique based on the UNIX shell.

site reliability engineering: a modern operations approach made popular by Google.

software as a service (SaaS): software that is provided as a service from the cloud and in a per-consumption model.

software configuration management: is the practice of tracking and controlling changes in software that includes version control, branching of parallel development, and maintaining a view of what versions of code are included in a software package.

software delivery life cycle (SDLC): this cycle describes all activities required to implement requirements from idea to production go-live.

stateful calls: a programming technique that requires a service to remember the state of the transaction to successfully complete it over several transactions.

stateless calls: a programming technique that does not require a service to know the current state of the transaction.

story points: a sizing approach in Agile methods that is based on relative sizing instead of absolute sizes, such as days or hours.

strangler pattern: a programming technique that builds new functionality and diverts workflows to the new functionality incrementally until the old program can be turned off.

systems integrator (SI): a company that helps organizations to bring the different components of a system together through implementing,

planning, coordinating, scheduling, testing, improving, and sometimes maintaining the system for the organization.

systems of engagement: systems that users directly engage with and that evolve quickly. (See also *systems of record*.)

systems of record: systems that hold core data and do not need to evolve quickly. (See also *systems of engagement*.)

systems thinking: an approach to analyzing systems as a whole instead of as a sum of their parts.

technical debt: known or unknown parts of programs that are currently suboptimal and should be refactored.

technology stack: all the technologies that are required to support business functions from the operating system up to the actual applications.

telematics: a systematic approach to gathering and using information that is generated during the use of a system.

theory of constraints: a scientific approach to analyzing systems based on the constraints that exist in the system.

to-be process design: the planned state of a process to be implemented that improves on the current condition.

true legacy: systems that are not being updated anymore and are only kept running to support business functions. (See also *legacy*.)

twelve-factor application: an architecture concept that provides twelve criteria for modern applications.

two-speed delivery model: a model that supports two different speeds of delivery to differentiate between fast-changing and slow-changing systems.

upskilling time: the time required for new people to become effective when joining a new team or learning a new skill.

user stories: used in Agile methods to describe the functionality of the system, often in the format of "As a <role>, I want to <functionality> so that <outcome>."

value stream mapping: an activity that maps out the whole legacy tranformation to support a process including contextual information such as timings, tooling, and other actors.

versioning: the practice of storing multiple versions of a program or other artifact so that it is possible to move between known states of configuration.

walled-garden test automation tools: test automation tools that are not enabled through application programming interfaces and that are only possible to be used within a specific technology or vendor context.

Waterfall: an exaggerated delivery approach based on stage containment between requirement, design, development, and test.

weighted shortest job first: is a way to determine the sequence of work by dividing the value one derives from the work by the cost of the work. An example formula for this is based on SAFe: cost of delay/duration.

XaaS: anything as a service.

Notes

Preface

1. Mirco Hering, “Agile Reporting at the Enterprise Level (Part 2)—Measuring Productivity,” *Not a Factory Anymore* (blog), February 26, 2015, https://notafactoryanymore.com/2015/02/26/agile-reporting-at-the-enterprise-level-part-2-measuring-productivity.

Introduction

1. Stefan Thomke and Donald Reinertsen, “Six Myths of Product Development,” *Harvard Business Review*, May 2012, https://hbr.org/2012/05/six-myths-of-product-development.
2. Don Reinertsen, “Thriving in a Stochastic World,” speech, YOW! conference, December 7, 2015, Brisbane, Australia, YouTube video, 56:49, posted by “YOW! Conferences,” December 25, 2015, https://www.youtube.com/watch?v=wyZNxB172VI.
3. “The Lean Startup Methodology,” The Lean Startup (website), accessed November 10, 2017, http://theleanstartup.com/principles.
4. Brad Power, “How GE Applies Lean Startup Practices,” *Harvard Business Review*, April 23, 2014, https://hbr.org/2014/04/how-ge-applies-lean-startup-practices.
5. Mirco Hering, “Let’s Burn the Software Factory to the Ground—and from Their Ashes Software Studios Shall Rise,” *Not a Factory Anymore* (blog), November 9, 2015,

https://notafactoryanymore.com/2015/11/09/lets-burn-the-software-factory-to-the-ground-and-from-their-ashes-software-studios-shall-rise.

6. Mark Rendell, "Breaking the 2 Pizza Paradox with Platform Applications," speech, DevOps Enterprise Summit 2015, San Francisco, CA, YouTube video, 25:26, posted by "DevOps Enterprise Summit," November 10, 2015, https://www.youtube.com/watch?v=8WRRi6oui34.

Chapter 1

1. "The DevOps Platform: Overview," ADOP (Accenture DevOps Platform on GitHub), Accenture, accessed May 2, 2017, http://accenture.github.io/adop-docker-compose.
2. Carreth Read, *Logic: Deductive and Inductive* (London: DeLaMare Press, 1909), 320.

Chapter 2

1. "Gartner IT Glossary: Bimodal," Gartner, Inc., accessed May 2, 2017, http://www.gartner.com/it-glossary/bimodal.
2. Ted Schadler, "A Billion Smartphones Require New Systems of Engagement," Forrester Research, Inc. blogs, February 14, 2012, http://blogs.forrester.com/ted_schadler/12-02-14-a_billion_smartphones_require_new_systems_of_engagement.
3. Martin Fowler, "Strangler Application," *MartinFowler.com* (blog), June 29, 2004, http://www.martinfowler.com/bliki/StranglerApplication.html.

Chapter 3

1. Mirco Hering, "How to Deal with COTS Products in a DevOps World," *InfoQ* (blog), July 24, 2016, https://www.infoq.com/articles/cots-in-devops-world.

Chapter 4

1. Francis Keany, "Census Outage Could Have Been Prevented by Turning Router On and Off Again: IBM," ABC News, October 25, 2016, http://www.abc.net.au/news/2016-10-25/turning-router-off-and-on-could-have-prevented-census-outage/7963916.
2. Mike Masnick, "Contractors Who Built Healthcare.gov Website Blame Each Other for All the Problems," *Techdirt* (blog), October 24, 2013, https://www.techdirt.com/articles/20131023/18053424992/contractors-who-built-healthcaregov-website-blame-each-other-all-problems.shtml.

Part B Introduction

1. Barry Schwartz, “The Way We Think about Work Is Broken,” filmed March 2014 in Vancouver, BC, TED video, 7:42, https://www.ted.com/talks/barry_schwartz_the_way_we_think_about_work_is_broken.
2. Dan Pink, “The Puzzle of Motivation,” filmed July 2009 in Oxford, England, TED video, 18:36, https://www.ted.com/talks/dan_pink_on_motivation.

Chapter 5

1. “PI Planning,” SAFe (Scaled Agile Framework), Scaled Agile, Inc., updated November 11, 2017, http://www.scaledagileframework.com/pi-planning.
2. Paul Ellarby, “Using Big Room Planning to Help Plan a Project with Many Teams,” *TechWell Insights* (blog), November 26, 2014, https://www.techwell.com/techwell-insights /2014/11/using-big-room-planning-help-plan-project-many-teams.
3. Wikipedia, s.v. “Dunning–Kruger effect,” last modified November 11, 2017, 19:01, https://en.wikipedia.org/wiki/Dunning%E2%80%93Kruger_effect.
4. Wikipedia, s.v. “Technology tree,” last modified November 13, 2017, 21:45, https://en.wikipedia.org/wiki/Technology_tree.

Chapter 6

1. Jargon File (version 4.4.7), s.v. “Conway’s Law,” accessed November 14, 2017, http://catb.org/~esr/jargon/html/C/Conways-Law.html.
2. *2016 State of DevOps Report* (Portland, OR: Puppet Labs, 2016), p. 9, https://puppet.com/resources/white-paper/2016-state-of-devops-report.
3. Rouan Wilsenach, “DevOpsCulture,” *MartinFowler.com* (blog), July 9, 2015, https://martinfowler.com/bliki/DevOpsCulture.html.
4. Matthew Skelton, “What Team Structure Is Right for DevOps to Flourish?” ed. Manuel Pais, *DevOps Topologies* (blog), accessed May 2, 2017, http://web.devopstopologies.com.
5. “WSJF—Weighted Shortest Job First,” Black Swan Farming, accessed May 2, 2017, http://blackswanfarming.com/wsjf-weighted-shortest-job-first.

Chapter 7

1. W. Edwards Deming, *Out of the Crisis* (Cambridge, MA: MIT Press, 1982), 29.

2. Kin Lane, “The Secret to Amazon’s Success Internal APIs,” API Evangelist blog, January 12, 2012, http://apievangelist.com/2012/01/12/the-secret-to-amazons -success -internal-apis.
3. Jeff Galimore et al., *Tactics for Implementing Test Automation for Legacy Code* (Portland, OR: IT Revolution, 2015).

Chapter 8

1. Anonymous, private conversation with author, 2004.
2. Dan Pink, “The Puzzle of Motivation,” filmed July 2009 in Oxford, England, TED video, 18:36, https://www.ted.com/talks/dan_pink_on_motivation.
3. Mark Horstman, “Managerial Economics 101,” YouTube video, 4:33, posted by “Manager Tools,” May 3, 2009, https://www.youtube.com/watch?v=gP-RC5ZqiBg.
4. John Goulah, “Making It Virtually Easy to Deploy on Day One,” *Code as Craft* (blog), March 13, 2012, https://codeascraft.com/2012/03/13/making-it-virtually-easy-to -deploy-on-day-one.
5. Mirco Hering, Dominica DeGrandis, and Nicole Forsgren, *Measure Efficiency, Effectiveness, and Culture to Optimize DevOps Transformation* (Portland, OR: IT Revolution, 2015), 14, https://itrevolution.com/book/measure-efficiency-effectiveness -culture-optimize-devops-transformations.

Chapter 9

1. Jez Humble and David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation* (Crawfordsville, IN: Pearson Education, Inc., 2011).
2. *The Netflix Tech Blog*, assessed November 16, 2017, http://techblog.netflix.com.
3. “Hygieia: An OSS Project Sponsored by Capital One,” Capital One DevExchange, assessed November 16, 2017, https://developer.capitalone.com/opensource-projects /hygieia.

Chapter 10

1. Jez Humble, “Architecting for Continuous Delivery,” speech, DevOps Enterprise Summit 2015, San Francisco, CA, YouTube video, 34:17, posted by “DevOps Enterprise Summit,” November 17, 2015, https://www.youtube.com/watch?v=_wnd-eyPoMo.
2. Randy Shoup, “Pragmatic Microservices: Whether, When, and How to Migrate,” speech, YOW! conference, December 2015, Brisbane, Australia, YouTube video,

49:00, posted by "YOW! Conferences," December 30, 2015, https://www.youtube.com/watch?v=hAwpVXiLH9M.

3. James Lewis, "Microservices—Building Software That Is #Neverdone," speech, YOW! conference, December 2015, Brisbane, Australia, YouTube video, 45:55, posted by "YOW! Conferences," December 29, 2015, https://www.youtube.com/watch?v=JEtxmsJzrnw.
4. Wikipedia, c.v. "Conway's law," last modified November 3, 2017, 09:02, https://en.wikipedia.org/wiki/Conway%27s_law.

Chapter 11

1. "About IT4IT," The Open Group, accessed August 4, 2017, http://www.opengroup.org/IT4IT/overview.

Chapter 12

1. Keith Collins, "How One Programmer Broke the Internet by Deleting a Tiny Piece of Code," Quartz Media, March 27, 2016, https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code.
2. Josh Corman and John Willis, "Immutable Awesomeness," speech, DevOps Enterprise Summit 2015, San Francisco, CA, YouTube video, 34:25, posted by "Sonatype," October 21, 2015, https://www.youtube.com/watch?v=-S8-lrm3iV4.
3. Debbi Schipp, "Bonus Bet Offers Peak as Online Agencies Chase Cup Day Dollars," News.com.au, November 1, 2016, http://www.news.com.au/sport/superracing/melbourne-cup/bonus-bet-offers-peak-as-online-agencies-chase-cup-day-dollars/news-story/8e09a39396fb5485cf1f24cbea228ff9.
4. Yury Izrailevsky and Ariel Tseitlin, "The Netflix Simian Army," *The Netflix Tech Blog*, July 18, 2011, http://techblog.netflix.com/2011/07/netflix-simian-army.html.

Appendix

1. Mirco Hering, "Agile Reporting at the Enterprise Level (Part 2)—Measuring Productivity," *Not a Factory Anymore* (blog), February 26, 2015, https://notafactoryanymore.com/2015/02/26/agile-reporting-at-the-enterprise-level-part-2-measuring-productivity.
2. Andy Boynton and William Bole, "Are You an 'I' or a 'T'?" *Forbes Leadership* (blog), October 18, 2011http://www.forbes.com/sites/andyboynton/2011/10/18/are-you-an-i-or-a-t/#2517d 45b351b.

3. Don Reinertsen, “Thriving in a Stochastic World,” speech, YOW! conference, December 7, 2015, Brisbane, Australia, YouTube video, 56:50 posted by “YOW! Conferences,” December 25, 2015, https://www.youtube.com/watch?v=wyZNxB172VI.
4. Gary Gruver and Tommy Mouser, *Leading the Transformation: Applying Agile and DevOps Principles at Scale* (Portland, OR: IT Revolution, 2015), 17.
5. Frederick P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering*, anniversary ed., 2nd ed., (Crawfordsville, IN: Addison-Wesley Longman, Inc., 2010), 25.
6. Brooks, *The Mythical Man-Month*, 17.

Acknowledgments

Like they say about raising children, the same is true for writing a book—it takes a village to do so. And I am sure I will miss people who should really be on these pages. Apologies for that. Grab me at the next conference, and I will buy you a drink instead.

First of all, I have to thank the fantastic team that supported me through the editing process: Todd, Gene, Anna, Leah, and Karen—without you, my thoughts would have never found a presentable and readable form. It was hard work but also a lot of fun working with you.

Then there are the peer reviewers who offered up some of their valuable time to provide feedback: Eric, Yong, Ajay, and Emily—you helped to bring the book into final shape and kept me honest.

I have some special thanks to give to three people without whom this book would have never happened: Eric, Todd, and Gene—you helped me move from "I can never write a meaningful book" to "Hey, I might have something to say that can help people." Your early support kicked off all of this.

I want to thank Accenture leadership—Bhaskar Ghosh, Adam Burden, and Peter Vakkas—for their support of this project and for providing me the flexibility at work to get this “labor of love” done.

And then there is Gary Gruver, who shared many pieces of advice from his book-writing experience and helped me come up with my own writing strategy.

I would also like to thank all the people in the Accenture DevOps practice and other parts of the company who helped shape my approach to enterprise transformations. I want to thank the clients that I have been working with and from whom I always learn something new during the engagements. You might find some of your thinking reflected in this book.

Last but not least, I need to say a huge thank you to my wife, Anjali, who has been a fantastic support as I have been writing this book—and at a time in our life that already has a full schedule thanks to a beautiful little boy, who was born while this book was in progress. Anjali, you are an absolute star!

About the Author

For over a dozen years, Mirco Hering has worked on accelerating software delivery through innovative approaches (what is now called DevOps), and ten years ago, started experimenting with Agile methods. As the Asia Pacific lead for DevOps and Agile at Accenture, he supports major public- and private-sector companies around the world in their search for efficient IT delivery. Mirco blogs about his experiences at NotAFactoryAnymore.com and speaks at global conferences to share what he has learned. You can also follow Mirco on Twitter: @MircoHering.