freiheit.com
technologies
hamburg / lisbon

Why Agile Is Not Enough: How to Build and Run a Large Software Engineering Organization

freiheit.com Webinar
Hamburg, 30th July 2020

1983

FIRST LINE OF CODE ON AN APPLE][


1999
FREIHEIT.COM FOUNDED

freiheit.com

FOUNDED IN 1999

ACTUALLY IN OCT 1998, SAME MONTH AS GOOGLE

150+ EMPLOYEES

LISBON / HAMBURG

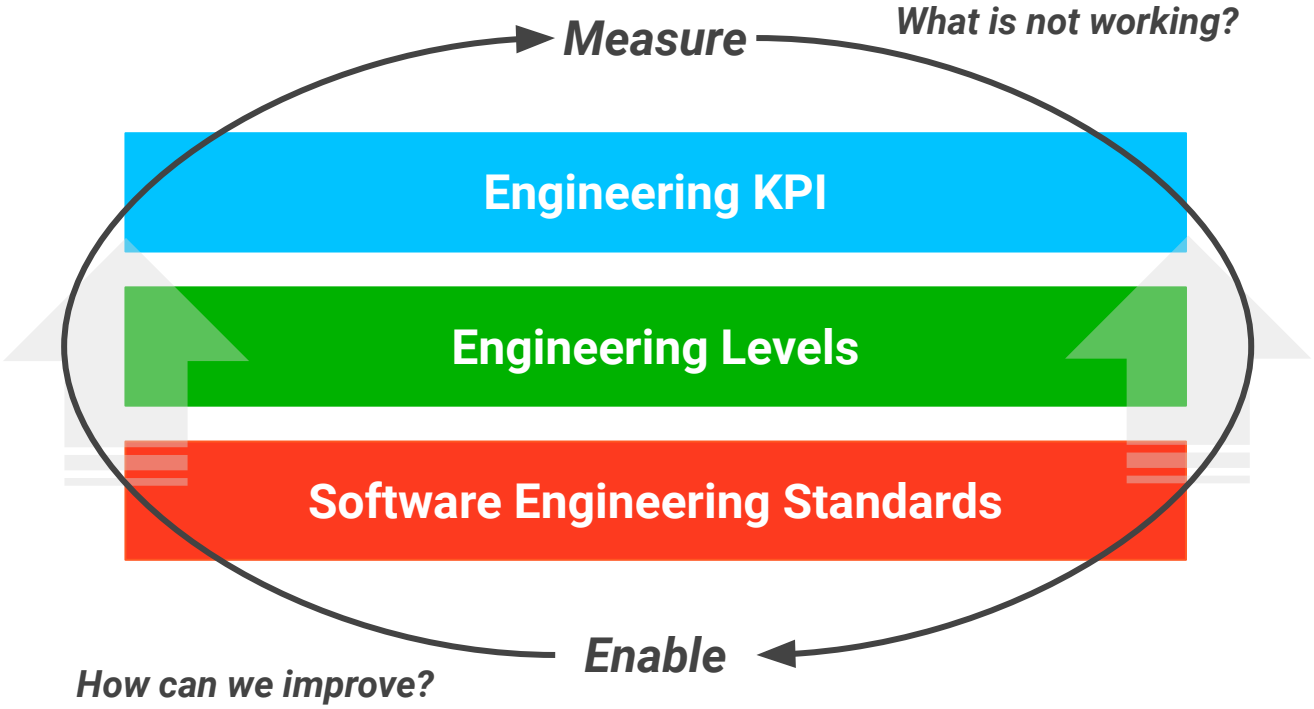
300+ LARGE-SCALE SOFTWARE PROJECTS

90% SOFTWARE ENGINEERS

STRONG MACHINE LEARNING BACKGROUND

20+ YRS EXPERIENCE, TECHNICAL KNOW-HOW, RELIABILITY

Today we want to talk about how to build and scale a data-driven software engineering organization.



What is *software*?

A computer is a universal machine that is used to build specialized machines.

Examples:

There are special machines to

- sell goods (eCommerce, Amazon),
- sell ads (Google),
- drive cars (Waymo),
- control a dishwasher (Bosch, Miele),
- control a vacuum cleaner (iRobot).

So any program is just a machine.

Not a physical (hardware) machine, but a “software” machine.

For any machine, there are three rules that are always true.

1. The machine should have as <u>few parts</u> as possible.
2. The parts should be as <u>independent</u> as possible from each other.
3. Same things should <u>work the same</u>, i.e. use the same mechanism.

As few parts as possible:
The Ford Model T had 1,481 parts and its assembly was a process of 7,882 distinct tasks.

As independent as possible:
You don’t have to remove the tires to fix the motor. And the motor will not break if you work on the rear seat.

Same things work the same:
The doors and the wheels use the same parts and have the same mechanism. If a mechanic knows how to assemble one wheel, they know how to assemble the other wheels, too.

This is the art of building a great hardware or software machine:

As few independents parts and mechanisms as possible.

And this is incredibly hard to do!

How can we transfer this to our understanding of software?

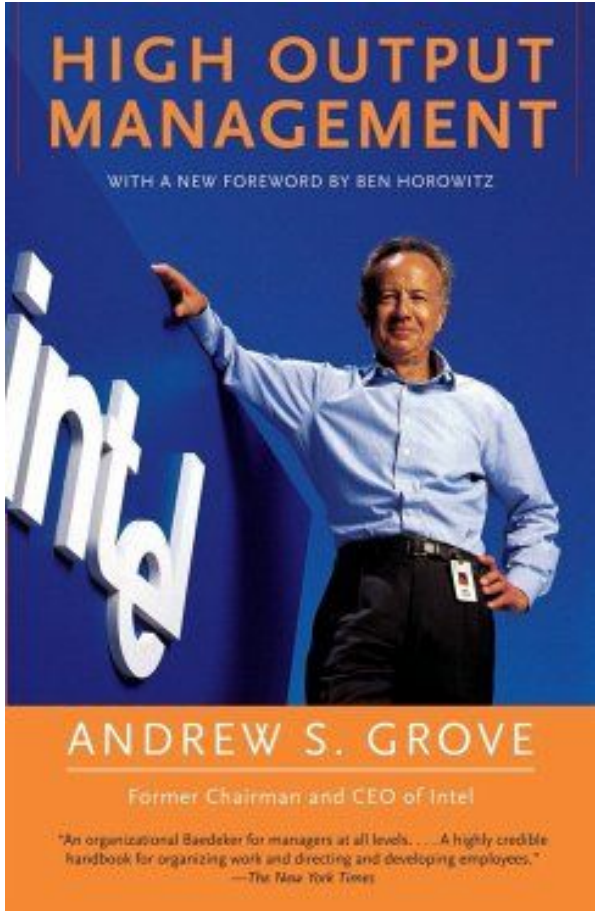
1. The parts of software are the libraries and frameworks that are used, the programming languages and any other components or concepts that are needed to create the expected result (“to build and run the specialized machine”).
2. The independence of the parts can be measured by how many of the used parts/components are directly interacting with each other.
3. Examples for mechanisms are the way a database is accessed or how logging and error handling is done.

When people talk about “agile”, they often say you should just “trust your team” and everything will be fine.

But is it enough to just trust a team that is able to build simple machines?

The answer is <u>no</u>. People must have the skills and expertise to work independently.

This is what Andy Grove called “task-relevant maturity”.



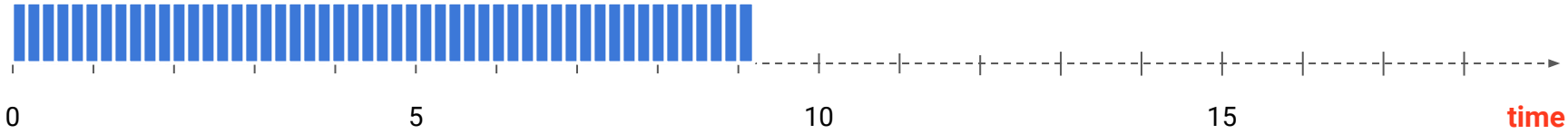
We can safely assume, that most people don’t have the skills and the experience to build simple machines like that.

What? Why is that?

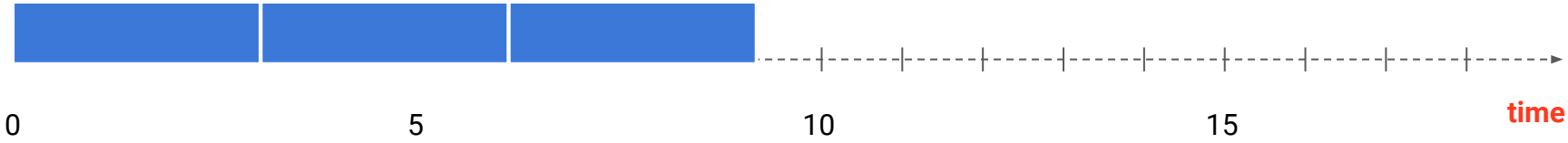
- Software engineering is a very young discipline.
- Software technology is rapidly changing and evolving. Adding good and bad ideas.
- To become a programmer you need a lot of practice. Not years but decades.

It is not for how long you are using a programming language. It is more about how much of the software engineering lifecycle you have experienced for yourself.

Many small projects: Has never dealt with long-term effects of technical decisions.

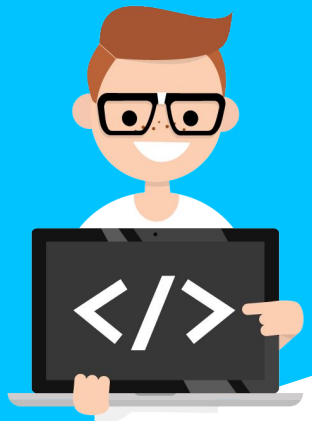


Few larger projects: Has seen different teams and stacks, has built and run a larger system.



Engineering Levels are needed to set expectations on what kind of skills, experience and level of influence are required for each level.

Engineerings Levels @ freiheit.com

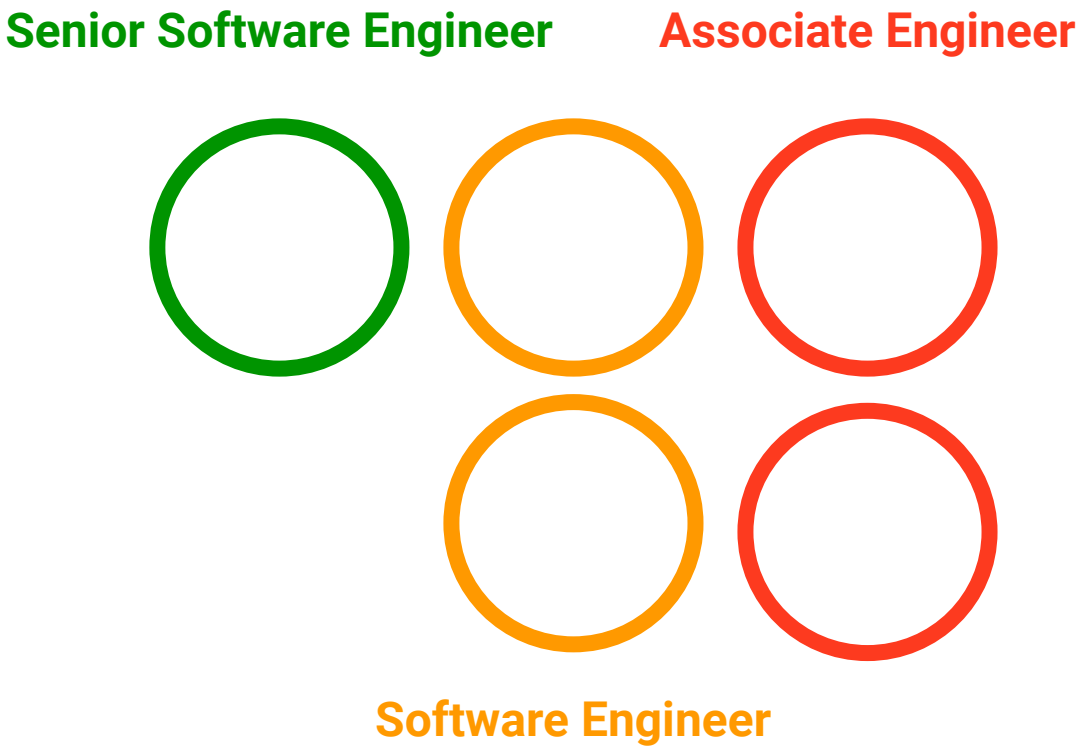


Associate Engineer	Software Engineer	Senior Software Engineer	Principal Software Engineer	Senior Principal Software Engineer	Engineering Manager
Learn	*Do*	*Run*	*Shape*	*Inspire*	*Scale*

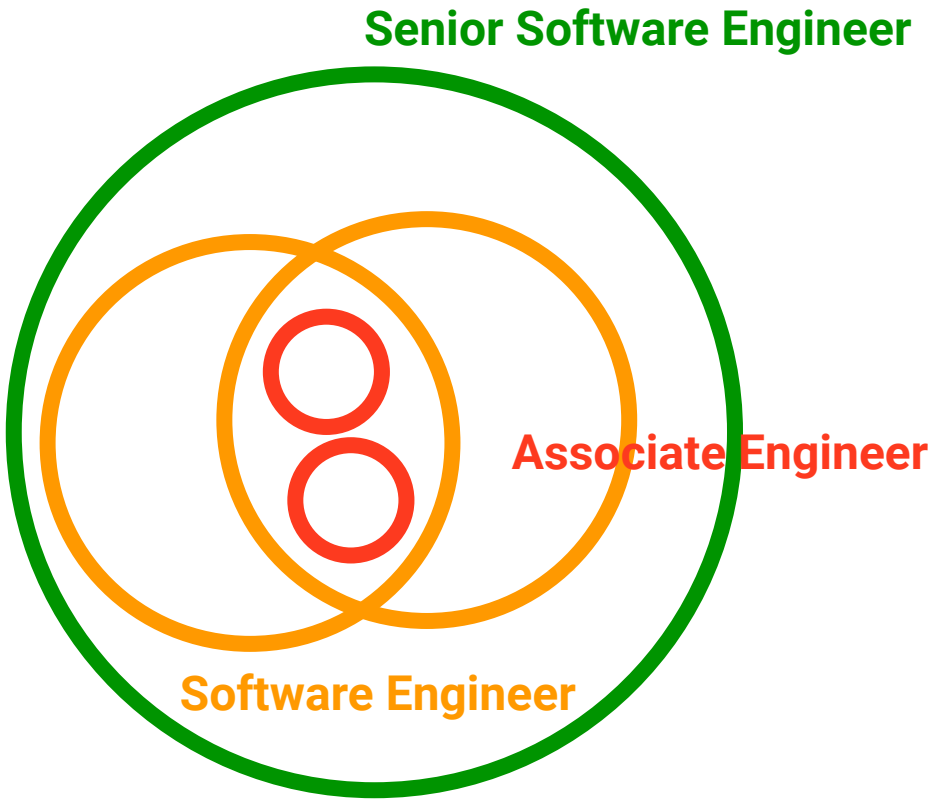
Areas: technical, analytical, execution, culture, influence, anti-patterns

Example: Senior engineers must be able to influence and inspire the engineers around them about how to build the simple machine.

You don’t want people to just work side by side ...



… you want people to take over the end-to-end responsibility for their code base and products.



the next level of experience and skills.

Overview

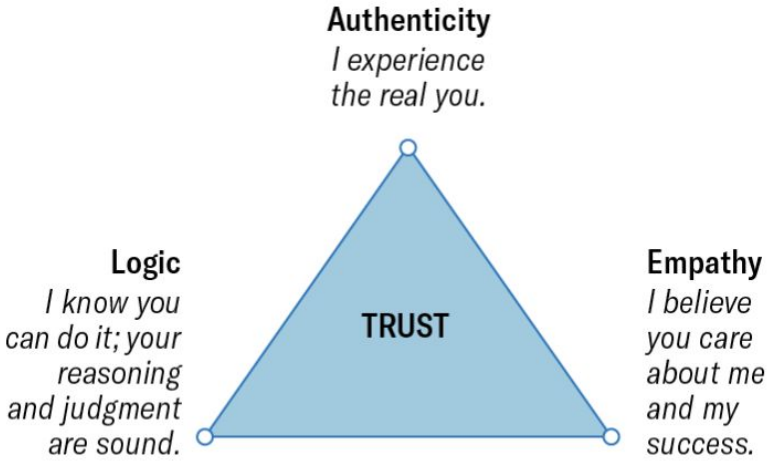
This should give you a very reduced overview over the levels to better understand the intention of each level.

Engineering Level	Shortcut	Focus
Associate Engineer	AE	Have continually displayed that they are able to take over the responsibility for the quality of their own code.
Software Engineer	SWE	Have continually displayed that they are able to take over the responsibility for the quality of the code base in their area of responsibility and their area of expertise.
Senior Software Engineer	S-SWE	Have continually displayed that they are able to take over the responsibility for the quality of the code base of a group of AE and SWE and for the SLO, SLA and Error Budgets of their production system. At least 3-5 years in house experience.
Principal Software Engineer	PSE	Have continually displayed that they are able to take over the responsibility for the quality of the code base of a group of S-SWE - maybe even across different teams - and their production systems. Can start new projects and build up new teams. Mentors S-SWE. Advocates new team standards.
Senior Principal Software Engineer	S-PSE	Have started and delivered several new projects. Mentors and develops PSE and S-SWE. Is responsible for the quality of the

To have influence (without power) people need to trust you. You can not just boss people around ...

The Trust Triangle

Trust has three drivers: authenticity, logic, and empathy. When trust is lost, it can almost always be traced back to a breakdown in one of them. To build trust as a leader, you first need to figure out which driver you “wobble” on.



From: “Begin with Trust,” by Frances Frei and Anne Morriss, May–June 2020

HBR

To build our leaders of tomorrow, we are working together with SNP.

Stay tuned: We are bringing the APM to Germany, together with SNP, the company who invented it with Marissa Mayer!

GOOGLE'S APM PROGRAM

What is Google APM?

Google's Associate Product Manager (APM) program is an amazing opportunity for new grads and early-career professionals to build exciting, global-scale products. As an APM, you will:

- Develop feature ideas that address user needs.
- Work cross-functionally (with engineers, UX designers, marketing, etc) to launch your features.
- Determine metrics to evaluate the success of your features and make improvements.

SNP

It's most important to emphasize to APM's what their contribution to the product is - they are the "hub." Prioritizing between the micro and the macro arena. Developing, maintaining and sustaining relationships between engineering, data scientists, finance, sales, marketing and, most of all, the user based on trust.

The relationship between PM and Engineers are not only important to have at a high level but it's important to understand role expectation as well. PM's drive the "what and why." Engineers drive the "when and how."

Next Webinar!

MARISSA'S MARVELS: The Graduates Of Her Google Genius School

Owen Thomas Jul 24, 2012, 2:03 AM

It's the closest thing Silicon Valley has to Professor X's School for Gifted Youngsters.

For a decade at Google, Marissa Mayer ran the Associate Product Manager program, an elite training regimen for recruiting fresh talent into its

Cultivating creativity. Marissa Mayer/Google+

From PM to CEO: How Sundar Pichai's Background in Product Paved the Way for Becoming CEO at Google

Expectations are not enough. You also have to enable engineers how to reduce complexity.

This is why you need <u>Engineering Standards</u> to write down your organization's engineering expertise for reuse.

What do we mean by “standards”?

Standards are best practices to decrease complexity, how to build simpler systems.

- What to look for in a code review?
- How to access the database efficiently?
- How to do error handling and logging?
- How to design a cloud system based on numbers (expected traffic, latency, throughput etc.)?
- How to build in observability (e.g. monitoring, metrics, tracing, alerting)?
- How (and if) to versioning your API?
- How to copy really “big” data (import / export)?
- How to handle and validate fast request?

This is not the corporate “architecture board”. Did this ever work?

We are using a Requests for Comments (RFC) process to incorporate expertise from our teams into our standards.

Software Engineering Standards, Part 1

142 mins remaining

1. Introduction
2. General Dos and Don'ts
3. Observability
4. Version control
5. Dependency management
6. Code Reviews
7. Build management
8. Infrastructure
9. Testing
10. Programming Languages

2. General Dos and Don'ts

This chapter contains general and very simple rules that everybody always has to keep in mind, because not doing so often leads to non-reversible situations where it is difficult to decrease entropy in later development stages.

Often we say "Don't do X" and "Never do Y". This was made to underscore the importance. If there are exceptions to be made, they should be discussed with many senior people before going that path.

Optimize for the reader of code, not for the writer

Engineers often try to write code in a way that the number of characters or number of lines of code are minimal to "save time by typing less". The problem with this is, that code is only written once, but it is read many times more often, maybe hundreds or thousands of times more.

One important goal of writing code is that the code communicates its intention well and that a reader can clearly understand what the code is supposed to do. Therefore the goal is to always optimize for the reader and not for the writer of the code. Never do something just because you want to save time typing.

Do things one way

A large code base is easier to maintain the more uniform it is. Uniformity is very important and doesn't happen automatically: You have to actively do work to keep the code base uniform. Practically this means, that a project team has to agree on certain things, practices and patterns and then stick to them.

Examples are:

- Choose one HTTP client library and just use that. Don't introduce a second one
- Define one way how to access a database and stick with it
- Agree on one pattern how you do (structured) logging
- Have a clear definition how error handling is done

Over time, some of your ways of doing something will have to change. But changing it must be a deliberate process: Nothing should ever happen accidentally or because of a lonely decision by one engineer. There are three choices:

- **When you change something, then the old way of doing things is changed in the whole code base to the new way of doing things.** This is the prefered way of handling this situation if the change is mechanically easy to do. One example at freiheit.com was the move from Flow type annotations to TypeScript. Even in a large code base this change can be done in a couple of days. But in the case of more complex changes this could become a large-scale

Report a mistake

Back

Next

[RFC] Software Engineering Standards, Part 1 @ freiheit.com

File Edit View Insert Format Tools Add-ons Help Last edit was 14 days ago

100% Heading 2 Roboto 13

entropy in later development stages.

Often we say "Don't do X" and "Never do Y". This was made to underscore the importance. If there are exceptions to be made, they should be discussed with many senior people before going that path.

Optimize for the reader of code, not for the writer

Engineers often try to write code in a way that the number of characters or number of lines of code are minimal to "save time by typing less". The problem with this is, that code is only written once, but it is read many times more often, maybe hundreds or thousands of times more.

One important goal of writing code is that the code communicates its intention well and that a reader can clearly understand what the code is supposed to do. Therefore the goal is to always optimize for the reader and not for the writer of the code. Never do something just because you want to save time typing.

Do things one way

A large code base is easier to maintain the more uniform it is. Uniformity is very important and doesn't happen automatically: You have to actively do work to keep the code base uniform.

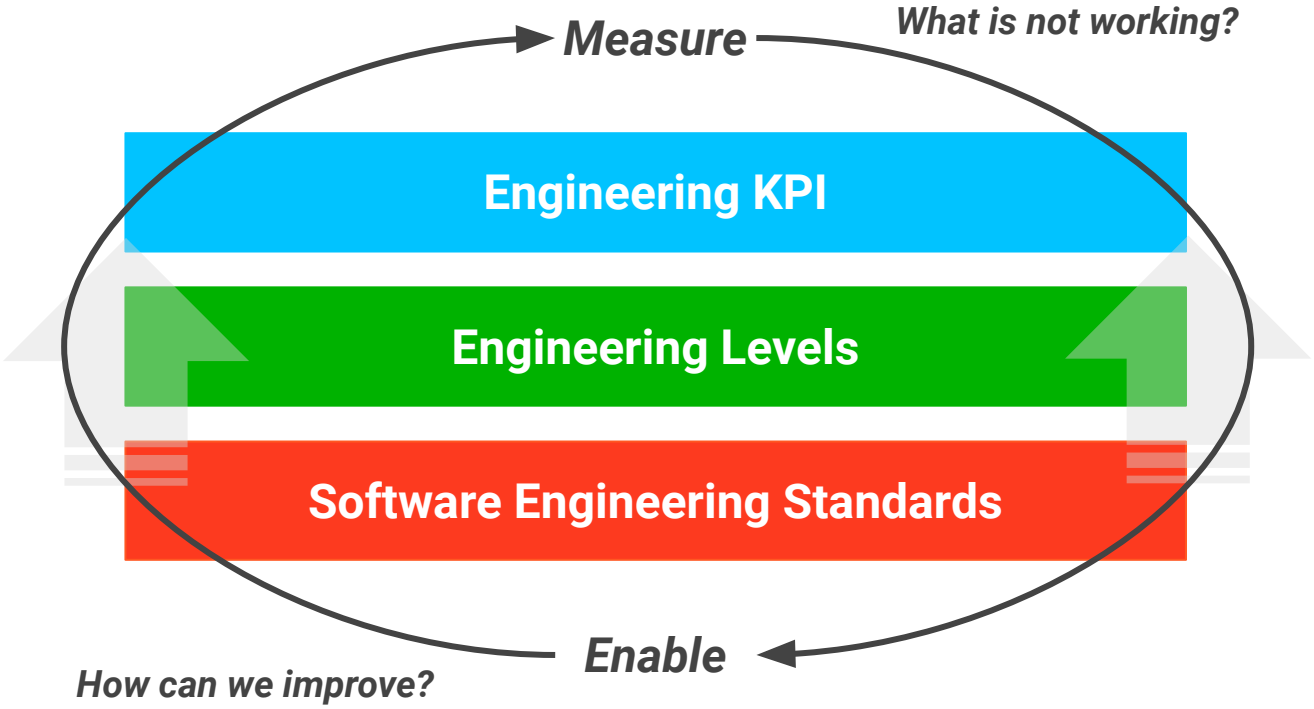
Practically this means, that a project team has to agree on certain things, practices and patterns and then stick to them.

Examples are:

- Choose one HTTP client library and just use that. Don't introduce a second one
- Define one way how to access a database and stick with it
- Agree on one pattern how you do (structured) logging
- Have a clear definition how error handling is done

Over time, some of your ways of doing something will have to change. But changing it must be a deliberate process: Nothing should ever happen accidentally or because of a lonely decision by one engineer. There are three choices:

And on top of this you have to put a KPI system to measure the quality of your systems and the productivity of the teams to understand what needs to be improved.



+++ SAVE THE DATE +++

Why Agile Is Not Enough: How to integrate Product Management and Engineering

with Stefan Richter & special guest

10th September 2020

For updates on our webinars, follow us on LinkedIn!

For further questions, please directly address: birgit.riedel@freiheit.com

THANK YOU

freiheit.com technologies
Budapester Str. 45
20359 Hamburg, Germany

kontakt@freiheit.com
T +49 40 890584 0
www.freiheit.com