Unlock the GAME CHANGING MINDSET *of* SUCCESSFUL ENGINEERS

by Mihai-Valentin Curelea

Mindset

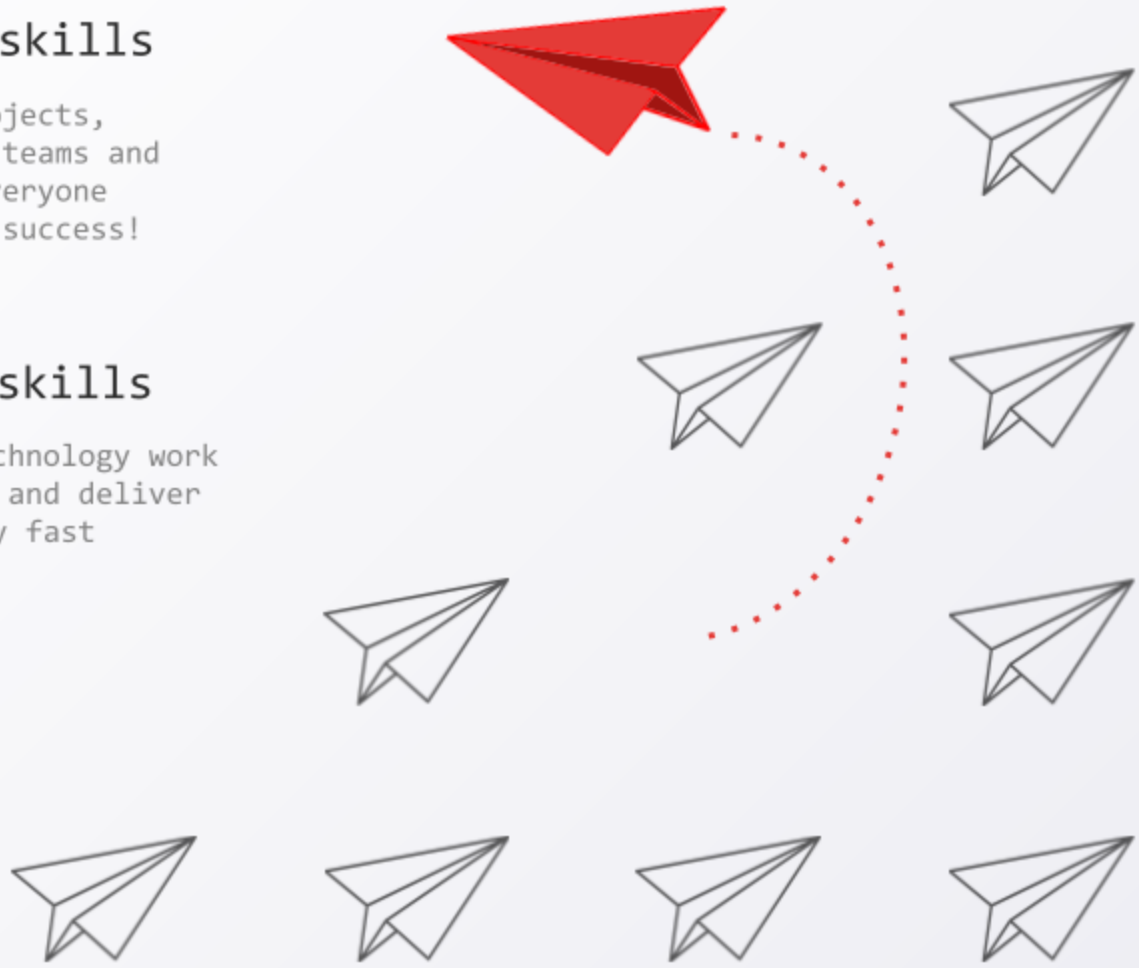
Focus on results and drive your projects to success!

Soft skills

Lead projects, people, teams and drive everyone towards success!

Tech skills

Make technology work for you and deliver insanely fast



Unlock the game-changing mindset of successful engineers

Copyright © 2021 **Mihai-Valentin Curelea**. All rights reserved.

This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review.

Disclaimers for "Unlock the game-changing mindset of successful engineers" book, by Mihai-Valentin Curelea

All the information in this book is published in good faith and for general information purpose only. Mihai-Valentin Curelea does not make any warranties about the completeness, reliability and accuracy of this information. Any action you take upon the information you find in this book "Unlock the game-changing mindset of successful engineers" is strictly at your own risk. Mhai-Valentin Curelea will not be liable for any losses and/or damages in connection with the use of this book. Although the publisher and the author have made every effort to ensure that the information in this book was correct at press time and while this publication is designed to provide accurate information in regard to the subject matter covered, the publisher and the author assume no responsibility for errors, inaccuracies, omissions, or any other inconsistencies herein and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause.

This book is meant as a source of valuable information for the reader, however it is not meant as a substitute for direct expert assistance.

From this publication, you can visit other websites by following hyperlinks to such external sites. While we strive to provide only quality links to useful and ethical websites, we have no control over the content and nature of these sites. These links to other websites do not imply a recommendation for all the content found on these sites. Site owners and content may change without notice and may occur before we have the opportunity to remove a link which may have gone 'bad'.

Please be also aware that when you leave this publication, other sites may have different privacy policies and terms which are beyond our control. Please be sure to check the Privacy Policies of these sites as well as their "Terms of Service" before engaging in any business or uploading any information. Our Privacy Policy was created by the Privacy Policy Generator.

Consent

By reading this publication, you hereby consent to our disclaimer and agree to its terms.

I am an ex-Facebook Tech Lead with more than 10+ years of experience in the industry. Worked at probably the largest scale in the world. Leadership experience in driving initiatives across multiple teams, strategic planning, reducing duplicate efforts, mentoring engineers.

My areas of expertise are:

- improving business KPIs using technology
- improving developers velocity and confidence
- improving time to market
- leading scalability/automation/resiliency efforts
- promoting data for making data-driven decisions.

https://www.linkedin.com/in/mihaivalentin/

https://mihaivalentin.com/

Opinions are my own and not the views of my employer.

Sample chapters

Get the full book at https://mihaivalentin.com/unlock

Be pragmatic and think strategically

People who are negative and don't get things done are most likely to slow and demotivate others. It is very hard to communicate and reason with these people, as they believe they know it all because they just read one blog post from a big tech company preaching about hammers, and now all they see around them are nails. Even worse, they can insist on adding extra technologies to the project every week, just because they are passionate and up to date with best practices. **They enjoy the journey and not the destination; however, it is the destination that pays everyone's salaries and keeps the lights on.**

If you have people like this in the company, you must work with them to think about results rather than code. Shift their enthusiasm towards delivery.

It is straightforward for engineers to learn new technologies and languages. Still, **it is extremely hard to adopt a new mindset -- the mindset of results**, pragmatism, business/customer-driven, where the technology is a means to an end and not the end itself.

The lack of pragmatism is a hidden effect of the inability to release often and prototype things fast. Because the moment when a feature goes into production seems far away, people feel they need to over-engineer and overanalyze what to do, starting to slow everything down.

To help developers embrace the mindset of pragmatism and results, ask questions like:

- What would you do if this were your company and you'd run out of money in 1 month?
- When would you rather find out that we need to change the requirements drastically? In 1 month (if we move fast), or in 5 months (if we do anything "by the book" as you say)?
- How would you see supporting all our projects in a codebase that uses more than 3-4 technologies?

- This new latest & greatest technology you want to add -- how does this work with the existing infrastructure, data, deployment, and tools we already have in this company?

How to implement this mindset?

If you are an executive:

- Set up teams/designate people to work on an infrastructure to release often and quickly prototype experiments in production. Get data to verify your assumptions. Encourage short release cycles, with feature toggles to release to a small number of users, get feedback, and improve.
- Work with the architects to define a shortlist of technologies and frameworks the company will use. Bring everyone up to speed with them, build tooling to increase developers' productivity, make sure code can be reused as much as possible.
- Make engineers aware of why fast iteration is crucial by putting them in more contact with business and sales. Make them aware of the competition in the market of your company's competitive advantage. Writing code with the business context in mind will help them and help the company in the long run.

Avoid overengineering and unnecessary complexity

Overengineering is one of the worst productivity killers. It not only takes much time during the coding phase, but it also takes a tremendous time to understand an overengineered codebase and add new features to it. Debugging over-engineered code is exceptionally frustrating. But this does not stop at coding. Overengineering also complicates writing new tests for the new features but also keeping the current tests green.

Perfectionists love overengineering. While working with perfectionists is excellent because they tend to foresee all the possible use-cases and failure scenarios, they could end up coding the core use-case in 80 lines and have 500 other lines that manage all the potential failures. Depending on the project you're working on, this can be a great thing (eg payments processing), or a terrible thing (incrementing website visited counters).

You know it's over-engineering when:

- When you're looking for actual code, but all you find are abstractions (lots of classes, interfaces, traits, etc., for something simple). In most cases, other developers will build something on the side just because it is excruciating to understand what's there, let alone try to add their one-liner logic across tons of classes and interfaces.

- When you haven't launched the MVP yet, you create a generic solution that you think could support the next 5 years. It is tough to foresee how the product will evolve in the future, and in most cases, the generic solution will not be too generic at all.
- When you look at code, you are surprised by how much code is there given what the program does.
- When you look at the code and find core infrastructure blocks implemented as code (e.g., load balancer, cache, firewall, API gateway, etc). Unless you know what you are doing (e.g., if you have a particular optimization use case that no off the shelf solution doesn't provide), never do this. Core infrastructure solutions have way better testing coverage, and they have proven reliability under production.

Here are some ways to ensure you don't overengineer:

- The best code is the code you don't have to write. Every line of code that you have to write comes with testing, maintenance, documentation, and operations.
- Use the simple principle from the UNIX world: "smart endpoints, dumb pipes." Make your programs extremely simple and glue them together to achieve more complicated things.
- Don't reinvent the wheel; make sure you evaluate many options before deciding to write something in-house. Writing a system in-house should be your last resort and should be done if you are convinced that existing services can't do what you need.
- Don't implement core infrastructure blocks from scratch.
- Don't write too many layers. I know OOP is cool, but adding 8 classes, 2 abstract classes, and 5 interfaces to support a new API method is not an OOP best practice! It's just overengineering.

Try to also think about things like:

- What would you do if this was your company?
- What if you run out of funding in 3 months?
- How does the project/task I'm working on will benefit the company?
- Would you be able to explain to non-tech people why you need to overengineer? Or why are you stuck?

There are, of course, cases when what we need to do can't be simplified. The only thing we can do about this is to structure our code correctly and encapsulate it in several modules, with clear input/output.

For example, if you need to have lots of files/classes, use code generation tools. Ensure you correctly name generated files so developers become aware of them. Make sure you add comments with which files they need to modify/which scripts they need to run to update them (ideally, write this information as a comment in the first lines of the generated files).

Even if you don't over-engineer if your system is complex, you will have to deal with its complexity. To do that, build tools, build metrics, build dashboards to help you understand how it is running, and also how the changes you're making impact production.

These were 2 sample chapters out of the **25+ chapters** from the full ebook.

Get the full ebook at https://mihaivalentin.com/unlock

This is the list of all the chapters:

Table of contents