Software as an Engineering Discipline

SE Radio 574: Chad Michel

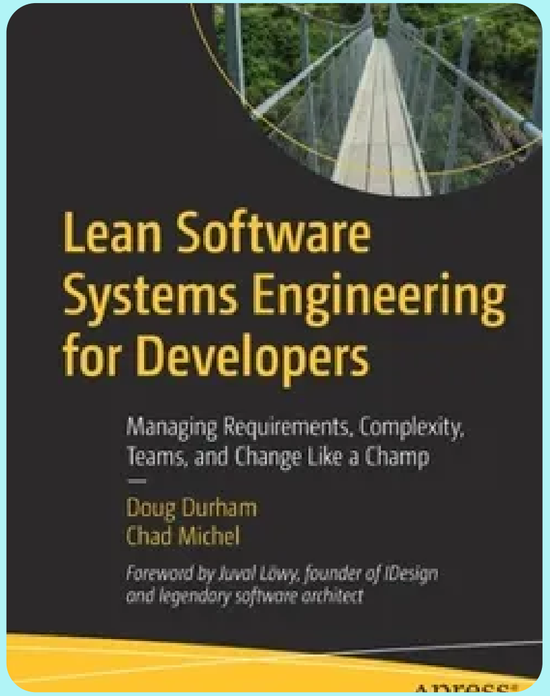
by: Alba González Arango
Lara Haya Santiago
Daniel Fernández Cabrero
Umut Dolangac

Contents

Introduction & Foundations of Software Engineering

Chad Michel’s Background

- Degree in Computer Engineering
- Master in Computer Science
- Head of Engineering at Don’t Panic Labs
- Co-author of Lean Software Systems Enigneering



Software Development vs. Software Engineering

Difference between both approaches

Software development and software engineering differ in their level of **rigor**. **Software development** focuses on quickly creating functional applications, often prioritizing **speed** and **convenience**. **Software engineering**, on the other hand, treats software creation as a structured process, emphasizing **planning**, **discipline**, and best practices.

Software Development

- Easy to start
- Lacks rigor
- Less quaility overall

Software Engineering

- Discipline
- Customer needs
- Long-term thinking

Importance of Rigor in Software Engineering

Engineering vs. Scientific rigor

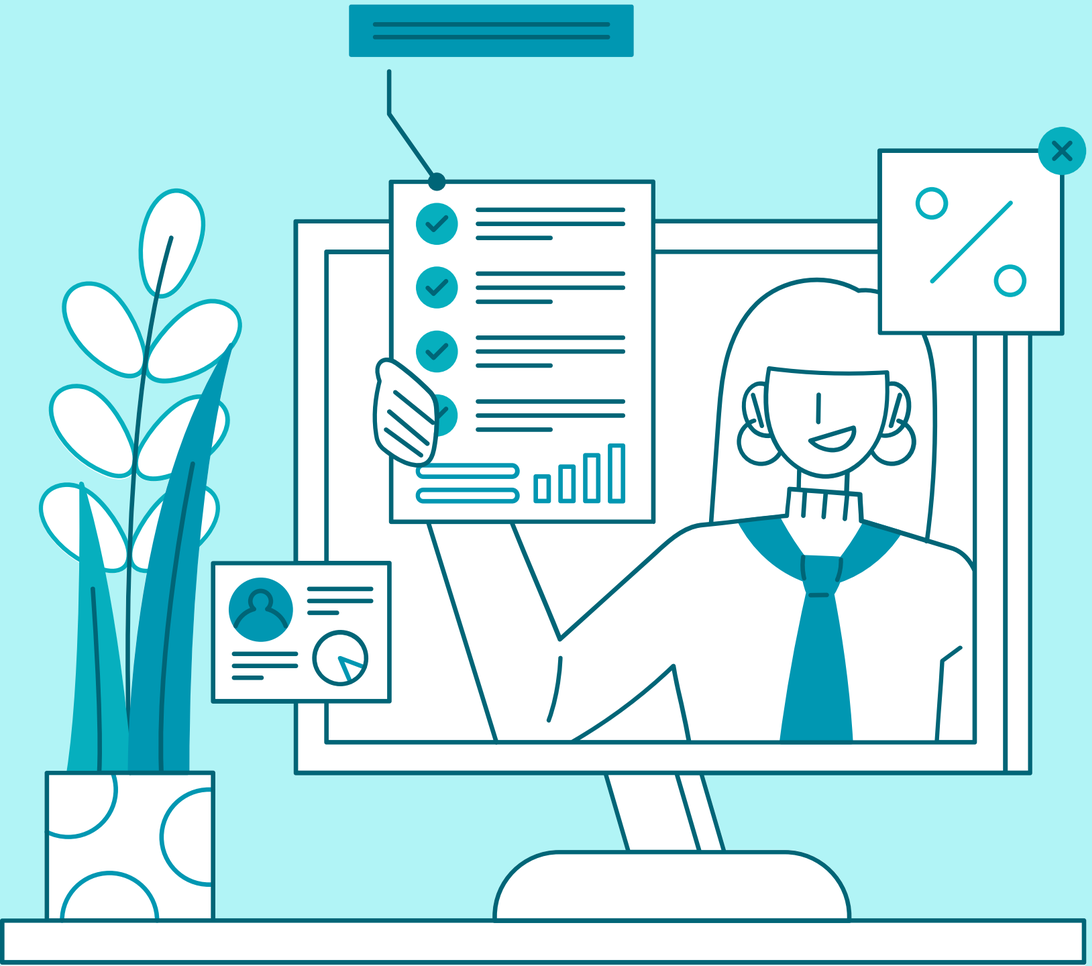
In **science**, they are building to learn something, while in **engineering** we are trying to learn how to build things.

Why rigor matters for long-lasting systems?

Creating **durable** software requires a **structured** approach. **Rushing** into development without proper research often leads to **fragile** solutions that fail over time. Since software exists to fulfill business objectives, considering factors like maintainability, cost, and risk from the beginning is crucial to ensuring its long-term success.

Business & Complexity in Software Engineering

Business language in Engineering



- **Schedule**

Meet the deadline

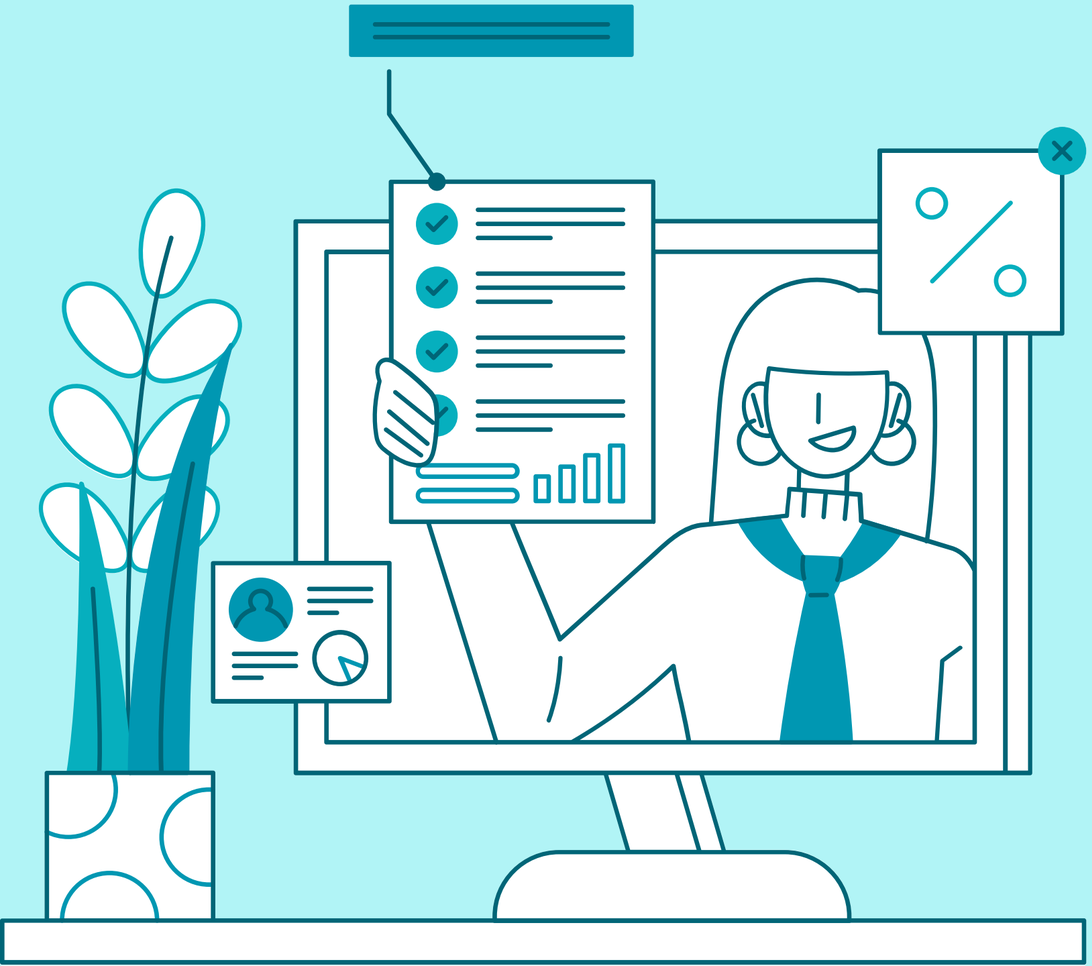
- **Cost**

Shouldn’t be much more elevated than expected

- **Risk**

Provide certainty to the business

Planning and trade-offs



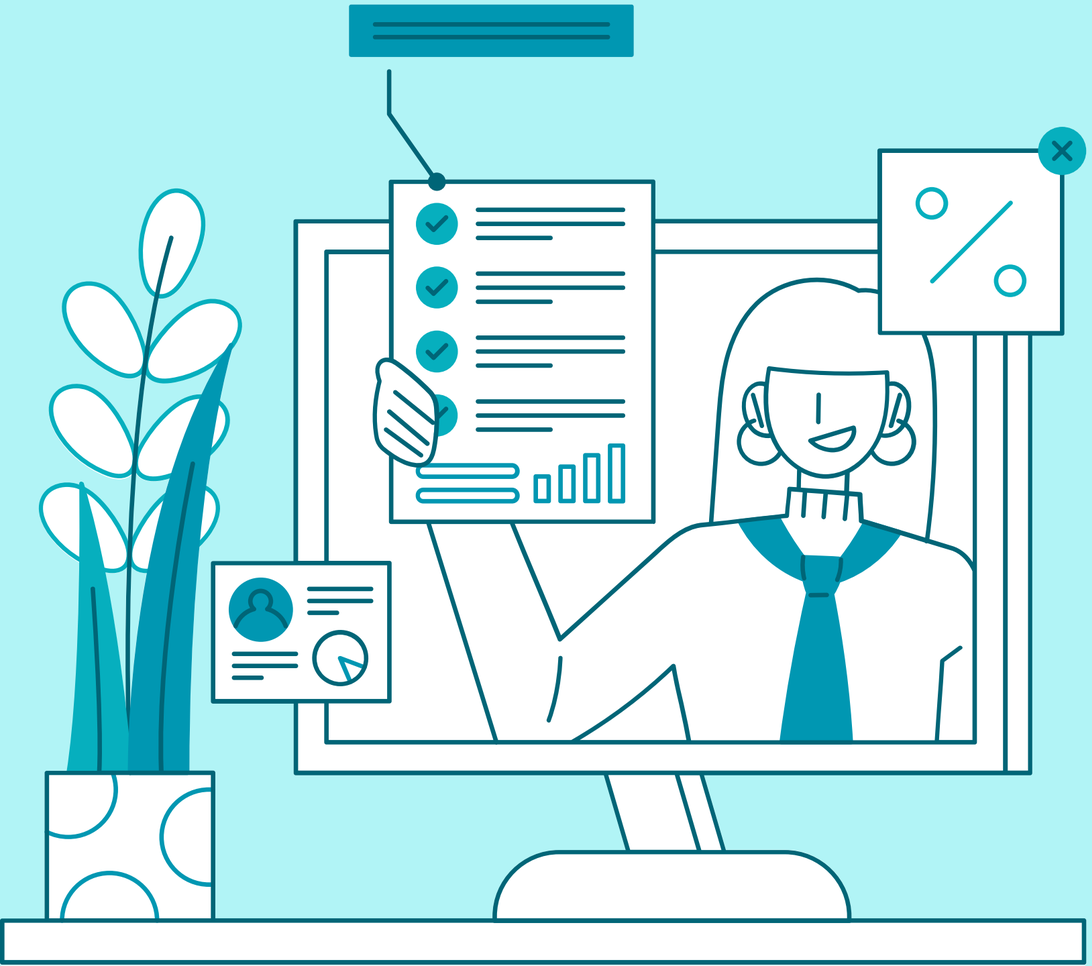
- **Trade-offs are necessary**
  - No unlimited budget
  - No unlimited time

- **Use estimates**
  - Effort to create, modify and mantain
  - In hours and points

- **Planning is essential**

“Plans are imperfect but a good enough plan can be better than no plan at all”

Complexity in Software Engineering

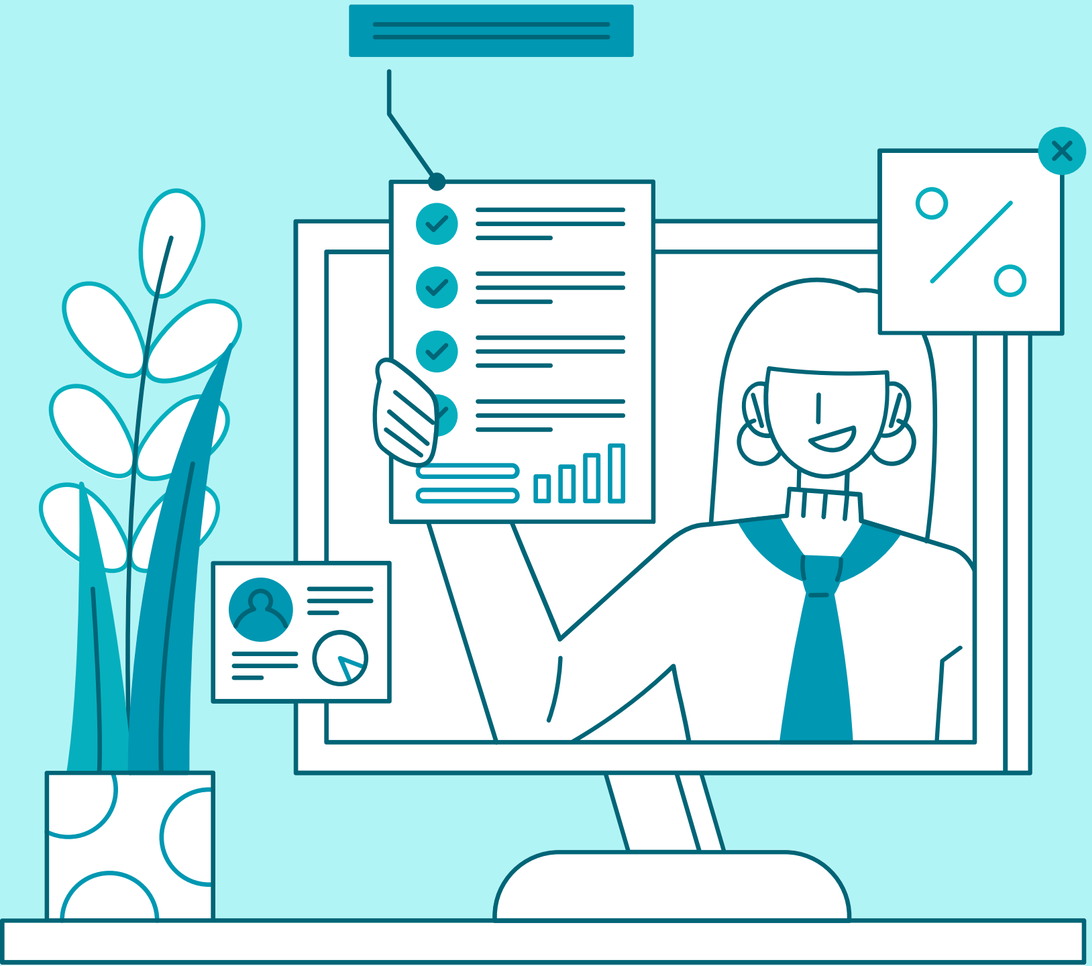


- **Objective complexity**
  - Costumer’s goals
  - Measurables
  - Challenges

- **Requirements complexity**
  - Very important to manage
  - Often duplicated, inconsistent or contradictory
  - Improved by agile tools

- **Solution complexity**
  - Be aware of implications of the delivery
  - Identify solution delivery problems
  - Avoid unexpected issues

Managing agility and control



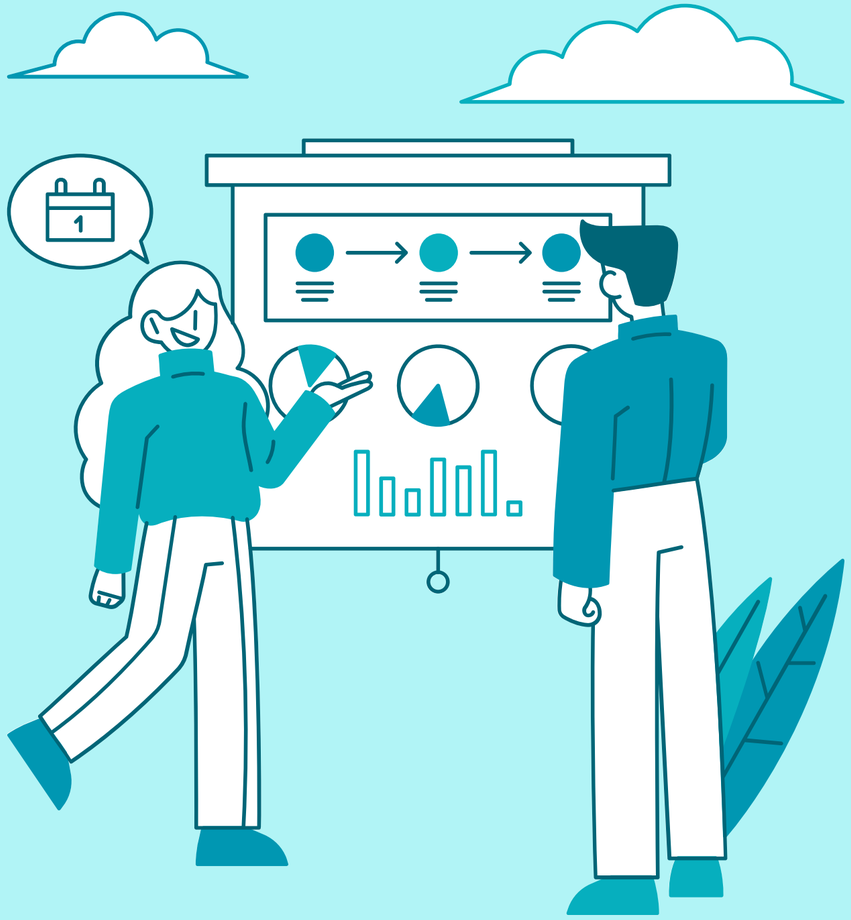
- **Agility**

  Requirements complexity
  +
  solution complexity

- **Design for change**
  - Assume changes will appear
  - Adapt to change
  - A new requirement shouldn’t require big changes

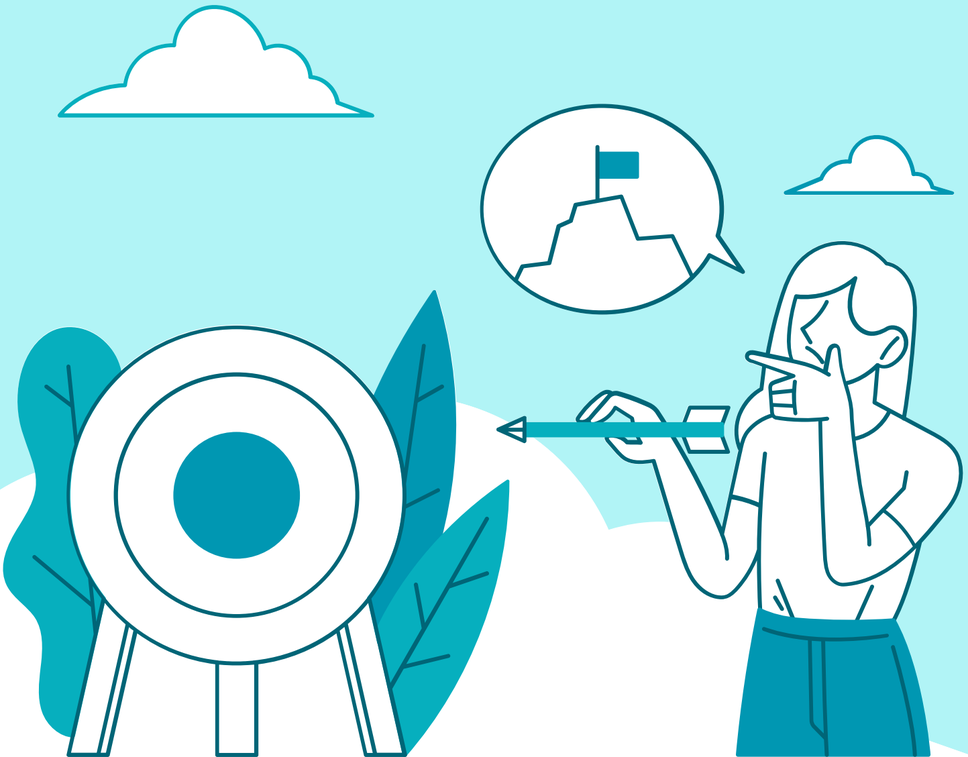
- **Design is iterative**
  - We cannot avoid changes
  - We can ensure to adapt for changes

- **Share the design**
  - Identify weak points
  - Get different perspective



Software Design, Quality, & Engineering Constraints

The importance of early issue detection



Bugs don’t just come from code

Many originate from **unclear requirements** and poor design. About 50% of product bugs stem from bad requirements, leading to costly and stressful fixes later in production.

Bugs Exist Beyond Code

- Issues can stem from bad requirements and poor design
- Fixation on code bugs often overlooks early-stage mistakes

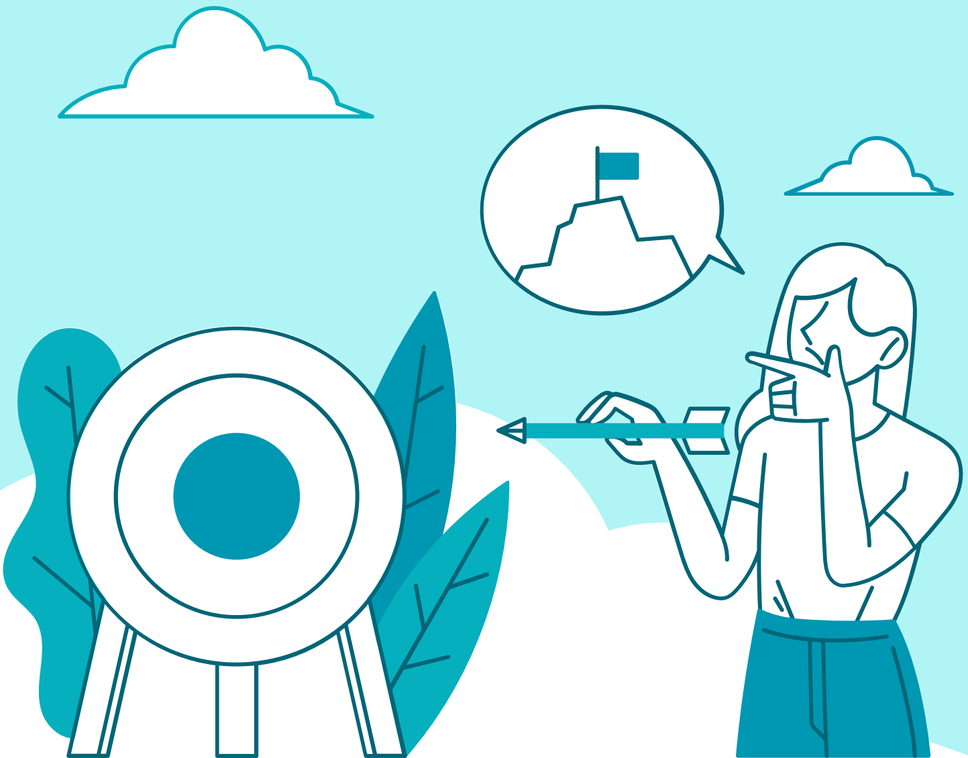
Cost of Late Bug Discovery

- Around 50% of product bugs come from unclear requirements
- Fixing bugs in production is expensive and stressful

Prevention Over Fixing

- Strong planning and clear requirements reduce costly mistakes
- Early detection makes fixes easier and cheaper
- Proper testing prevents new issues while fixing old ones

Quality Assurance vs. Quality Control



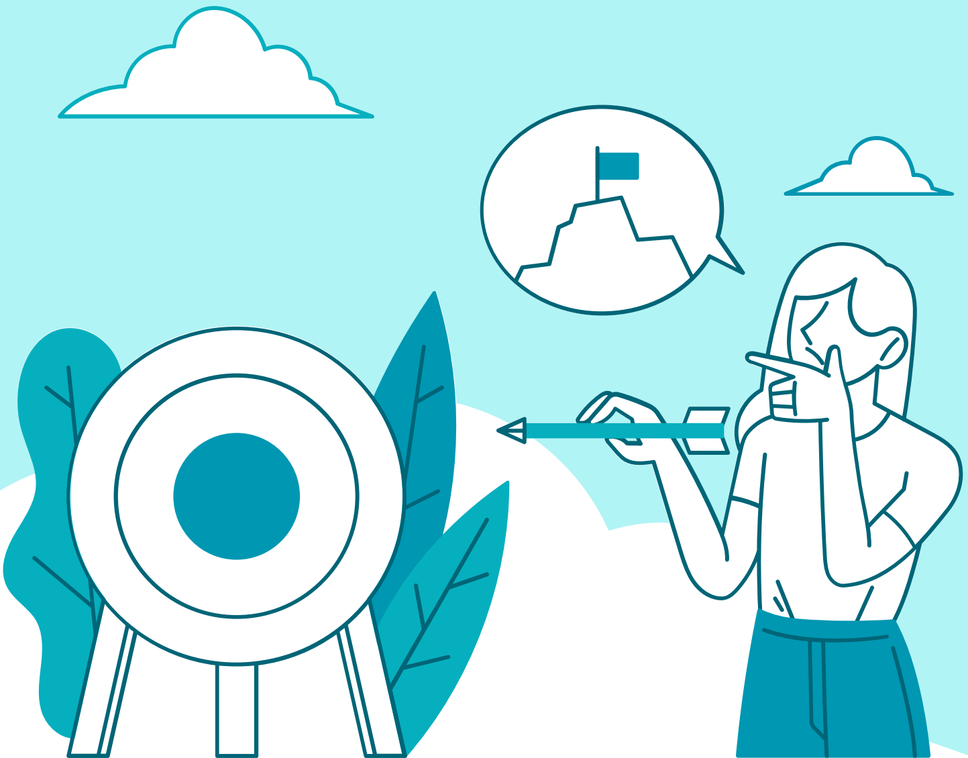
Quality isn’t just about testing at the end

It must be built into **every** step of development. Quality Assurance (QA) is proactive, ensuring good practices from the start, while Quality Control (QC) is reactive, catching issues later.

Toyota’s Andon Cord Concept

Inspired by Toyota’s Andon Cord, teams should feel **empowered** to stop development when quality concerns arise, preventing bigger problems down the line.

Institutionalizing Quality



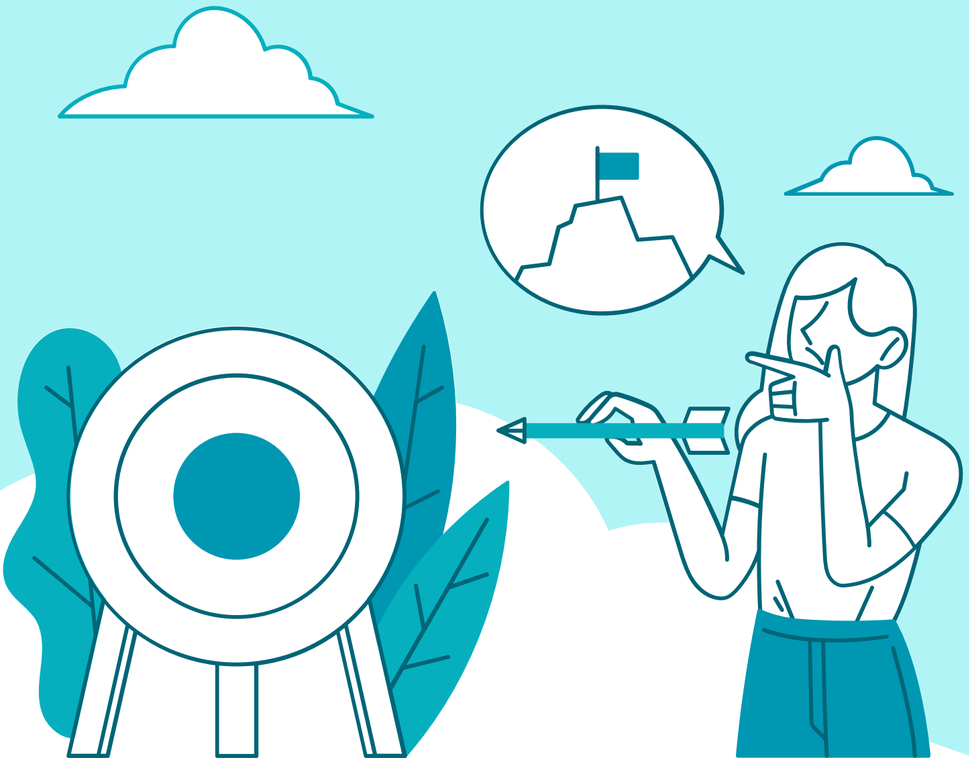
Testing has to be part of the entire team’s mindset

Building a **culture of quality** ensures that developers, designers, and project managers prioritize quality from the start. Addressing issues early prevents last-minute fixes.

The **chief engineer** plays a key role in **educating** and guiding the **team** to **maintain high standards**, especially in fast-paced development environments.

Leadership, Training, & Challenges In Software Engineering

Leadership

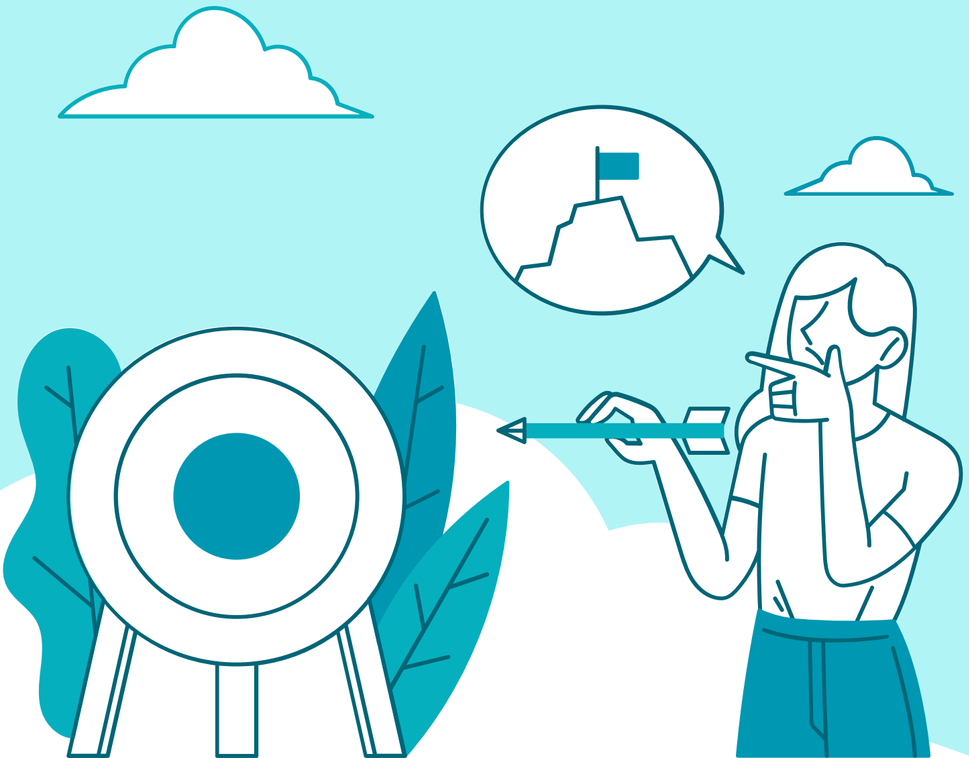


A good leader keeps team members consistently pulling in the same direction.

Chief Engineer:

- Knows lots of differents areas(hosting cloud, managing, security etc.)
- thinks overall
- Guides and educates new team members(kinda like coach)
- Ensuring consistency in engineering teams
- Motivates

Training



training helps engineers develop long-term skills and quickly adapt to new environments.

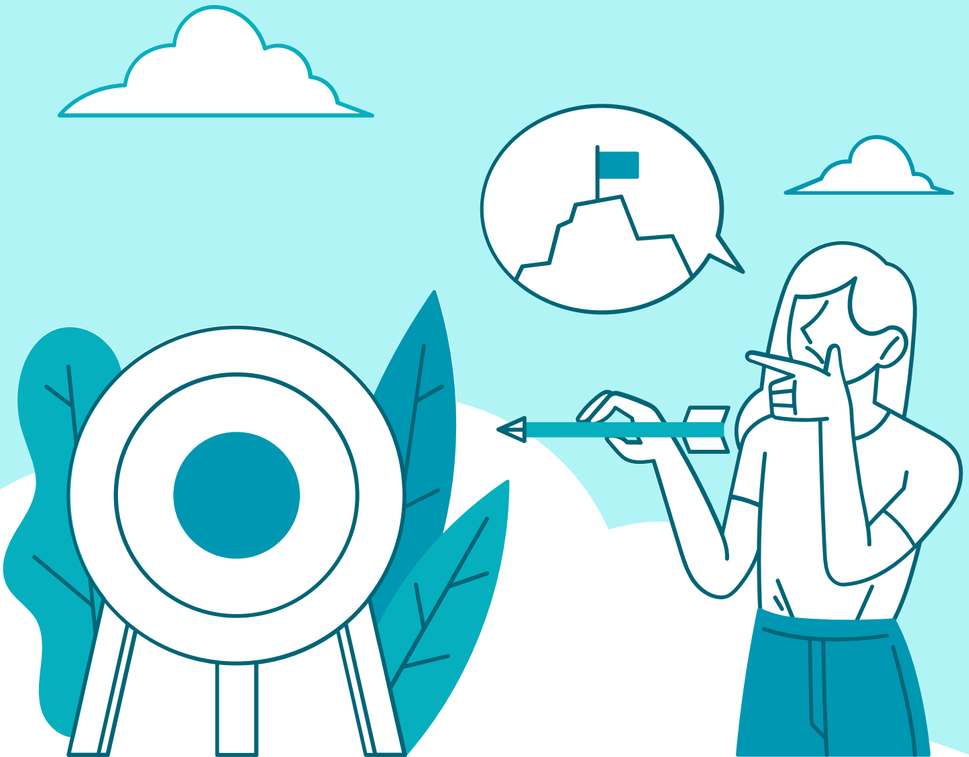
What to do?

- Learn core of systems not just tools
- Learn from others in the community to improve practices.
- Go to internal workshops
- Lots of practice

Challenges In Software Engineering

Software is Hard.

Donald Knuth



Tools are helpul, but using them does not makes you software enginner. Why?

- Without knowing how to design a system, you cannot solve unsolved problems or update old technologies to the current standards.

Other challenges:

- Building architectures that support long-term growth.
- Rapidly Evolving Technology
- Changing Requirements

THANKS!

Time for your questions