Overcoming organizational obstacles to robust software, data, and ML systems

The *people*-problem underlying short-term thinking: **Misaligned incentives**

- Incentives of employees should be aligned with this overall organization’s objective
- Organization's objective: Maximize (expected value of) discounted cash flow (subject to some constraints)
  - Of course, receiving money now is always better than receiving the same amount in a year. (Put another way, paying a given cost in a year is always better than paying that same cost now.)
  - If we don't have a good way of measuring an organization's health, this creates incentives for employees to claim success by realizing gains or avoiding costs now at the expense of the long-term health of the organization.
  - Note that this goes beyond ML and software systems, and mostly applies to even government bureaucracies and other nonprofits:
    - E.g., underinvestment in productivity tools (e.g., old and slow computers), burning out employees, damaging value of the brand by eroding product quality,…
- In most organizations, employee’s **incentives** are misaligned with organizational incentives:
  - **What gets measured gets done**!
    - Easy to observe/measure: Features completed; financial indicators (revenue, spending)
    - Imperfectly measurable (but often not even attempted): Employee satisfaction, hours worked per employee (sustainable pace?)
    - Hardest to measure: technical debt, health of the codebase (software quality)

  -> excessive focus on the short-term

The underlying *people*-problem : How to **align individual incentives with organization’s goals?**

- Solution:
  - 1) Try to make competing goals measurable.
    - Visualize these KPIs on a dashboard for decision makers
    - Hold teams and decision-makers accountable for these metrics. (Reward should be proportionate to how much a given metric is under direct control).
  - 2) Automatically enforce quality gates wherever possible
    - e.g., CI/CD pipeline should enforce test coverage, type annotations, code complexity thresholds, linting, security checks, etc.
- Even if the best we can get is an imperfect measurement, it is better than none at all.
  - The main point of these KPIs is to serve as an early warning signals of declining quality

The underlying *people*-problem: How to **align individual incentives with organization’s goals?**

- By using a **multitude of independent metrics**, we reduce the danger of gaming the system
  - e.g., if we just measured test coverage, it is easy for teams to put in faux tests that always pass
  - ... but if we also look at more direct outcomes (e.g., change failure rate and other common DevOps metrics), we provide incentives for teams to instead create tests that actually do their job.
- Still, we want a **combination of direct and indirect metrics**
  - If humans were rational, direct metrics would be enough. But humans are not rational…
  - Indirect metrics assess whether a team follows best practices. These provide a guide of *how* to achieve the ultimate goal.
  - Simplest solution: Create a checklist of specific best practices, and answer each with a yes or no.
    - Remember: Wherever possible, these should be enforced automatically (e.g., in CI/CD pipeline)
- Problem: Comparing metrics
  - Outcomes are not comparable *across teams*: Applications vary in terms of their inherent complexity
  - Outcomes are usually not even comparable *for a given team over time*: Complexity of a given application is not static (adding features; changing environment)
- This is another argument for *indirect* metrics that check whether teams follow best practices: Since doing so involves clear steps, it is most closely under a team's control.