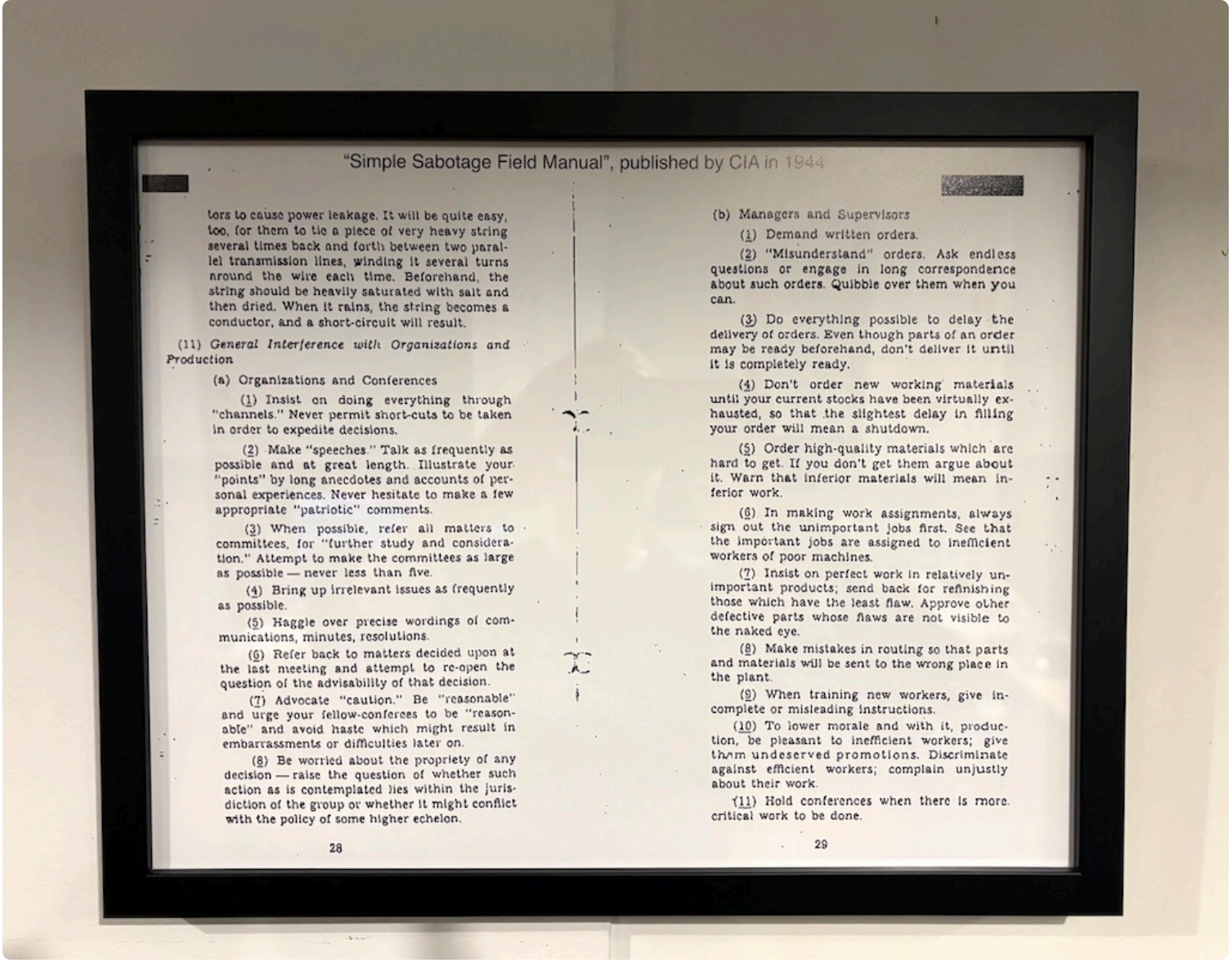
Simple sabotage for software

2023-12-13

CIA produced a fantastic book during the peak of World War 2 called Simple Sabotage. It laid out various ways for infiltrators to ruin productivity of a company. Some of the advice is timeless, for instance the section about “General interference with Organizations and Production”:

1. Insist on doing everything through “channels”. Never permit short-cuts to be taken in order to expedite decisions.
2. Make “speeches”. Talk as frequently as possible and at lengths. Illustrate your “points” by long anecdotes and accounts of personal experience. Never hesitate to make a few “patriotic” comments.
3. When possible, refer all matters to committees for “further study and consideration”. Attempt to make committees as large as possible — never less than five.
4. Bring up irrelevant issues as frequently as possible.
5. Haggle over precise wordings of communications, minutes, resolutions.
6. Refer back to matters decided upon at the last meeting and attempt to re-open the question of the advisability of that decision.
7. Advicate “caution”. Be “reasonable” and urge your fellow conferees to be “reasonable” and avoid haste which might result in embarrassments or difficulties later on.
8. Be worried about the propriety of any decision — raise the question of whether such action as is contemplated lies within the jurisdiction of the group or whether it might conflict with the policy of some higher echelon.

I guess I've always been fascinated with how well this has stood the test of time? I even got this particular section framed and hung up at our office:



Your mission

Let's say you were employed as a CTO behind the front lines and you wanted to destroy productivity for as long as you can without getting caught. You can of course make a series of *obviously* bad decisions, but you'd get fired quickly. The real goal here is to sap the company of its productivity slowly, while maintaining a façade of plausibility and normalcy. What are some things you can do?

Technology

- When joining, require a 6-18 months rewrite of core systems. Blame the previous CTO.
- Encourage everyone use their own language and frameworks.
- Split systems along arbitrary boundaries: maximize the number of systems involved in any feature.

- Encourage a complex dev setup: running a service mesh with a dozen services at a minimum.
- Make sure production environment differs from developer environments in as many ways as possible.
- Deploy as infrequently as possible. Urge extreme caution about deployments. Leverage any production issue as a reason to “pull the brakes”.
- Introduce very complex processes for code change and common workflows. Blame it on “security” or “compliance”.
- Make sure every task is tracked in a task tracker and has been reviewed, prioritized, and signed off by a group of at least five people.
- Disallow anything outside the scope of the original task, such as code cleanup or other drive-by improvements.
- Build in-house versions of almost anything that's *not* a core competency. Justify it by “avoiding vendor lock-in”.
- Insist on adding abstraction layers on top of everything. Use vendors that are themselves abstractions and then add extra layers of abstractions.
- Encourage technical decisions based on wildly optimistic expectations of scale. Plan for at least 3 orders of magnitude more load than you have.
- Encourage communal ownership of systems. Make sure no one feels responsible for maintenance.
- Insist on centralizing almost everything as a “platform” owned by the “platform team”. Understaff the platform team and prevent other teams from building anythings that the platform might “own”.
- Make the platform team iterate on APIs frequently and mandate that other teams refactor their code to the latest version as frequently as possible.
- Hire “architects” and require even small changes to have an “architecture review”.
- Require even small changes to have a “security review”.

Product

- Dismiss useful metrics on academic grounds (e.g. “biased” or “lagging indicator”).
- Pick vanity metrics with little or no correlation with business value and high amount of noise.

- Insist on anything to be done as a “big bet” and insist on everything to be completely done before deployed.
- Consider every feature a “must-have” and critical part of “version zero”. Do not budge.
- Develop incredibly detailed “strategic” plans.
- Pivot frequently.
- Dismiss obvious improvements as “local optimization”.
- Use latest trends to tie up resources. Kickstart a vacuous “AI strategy” that seems plausible at the surface. Spend heavily on vendors and consultants for these.
- Encourage product managers to spend most of their time on “strategy” and “planning”.
- Make it hard/impossible for engineers and product manager to use the product internally.
- Dismiss users as “stupid” internally.

Leadership

- Link compensation to title, and title to to team size, in order to incentivize bloat.
- Make big talk about strategies, features, or technical complexity.
- Make expensive acquisitions to enter new product areas. Refer to “synergies”. Shut down the acquired product.
- Use lots of dotted lines in the reporting structure.
- As much as possible, have people to report into managers in other teams, locations, or functions. Make sure managers are ill-equipped to supervise their reports.
- Frequently reassign underperformers to other teams.
- Put high performers on highly speculative R&D projects with unclear deliverables.
- Always require a meeting for any decision, no matter how trivial.
- Insist that every “stakeholder” needs to be present in the meeting.

Hiring

- Create a hiring process that seems plausibly objective but in reality subjective.
- Reject the best people based on “poor culture fit” or other vague criteria.
- Hire the weakest candidates based on “potential” or “attitude” or other vague criteria.
- Recruit very expensive senior leaders with large headcount promises.

- Use inflated titles and made-up roles to attract opportunists.
- Hire highly specialized “experts”, then create contrived projects to prevent them from quitting.
- Use specialization as a justification to hire other, complementary people.

Project management

- Require very detailed estimates for any project.
- Encourage projects that span as many teams as possible, ideally in different locations.
- Add new requirements that depend on work done by other teams.
- Frequently make use of expensive agencies. Make the scope ambigious and hand over unfinished prototypes on the in-house team for them to finish.
- Build complex “self-service” systems for stakeholders in other teams.

This is from the 1994 music video Sabotage by Beastie Boys. The lyrics are mostly about technology leadership and developer productivity.

The outcome

It's a hard job to pull it off! But if you can parachute behind the enemy front lines, and land a job as a CTO, you can make this happen.

For the non-saboteur: this is obviously a story about how to get the most out of your team. Productivity in general is a story of a thousand cuts, and none of these things are in themselves the thing that will ruin the productivity. But productivity adds up on a logarithmic scale, meaning that all these things compound in a multiplicative way. Basically, do 100 things that each is a 5% tax on productivity, and you just slowed everything down by 131x! The only way to keep engineers happy is to say no to 100 minor cuts that each sound plausible and specious.

Tagged with: management, software

Want to get blog posts over email?

Enter your email address and get an email (roughly monthly) when there's a new post!

Your email address Subscribe!

Related posts

σ-driven project management: when is the optimal time to give up? 2022-04-05
Optimizing for iteration speed 2017-07-06
What I have been working on: Modal 2022-12-07
We are still early with the cloud: why software development is overdue for a change 2022-10-19
Storm in the stratosphere: how the cloud will be reshuffled 2021-11-30

Erik Bernhardsson

... is the founder of Modal Labs which is working on some ideas in the data/infrastructure space. I used to be the CTO at Better. A long time ago, I built the music recommendation system at Spotify. You can follow me on Twitter or see some more facts about me.
