Editor: **George Fairbanks**
gf@georgefairbanks.com

Did Your Manager Choose Your Architecture?

Timothy J. Halloran

MOST ENGINEERS WOULD cringe at the idea of their managers choosing their system's software architecture, but back in 1968, this is exactly what Melvin Conway found to be true. Across many companies, Conway found that the organizational chart, chosen by managers, was replicated in the software systems of those companies.

In my 35 years of software development experience, I found that managers may not know the term *Conway's law* or read *The Mythical Man-Month*, but they know how their organizational chart impacts the company. By the end of this article, it's my hope that you have a better understanding of how managers do indeed influence software architecture and promote other factors by tweaking the organizational chart.

The idea of significant manager control over software specification and design is horrifying to many in-the-trenches software engineers. They might cry that such nontechnical manager involvement puts project success at significant risk, perhaps spelling doom for the work. The best real-world parable of this manager involvement is bad view that I know is from shipbuilding in the 15th century. The *Vasa* was a Swedish warship built between 1626 and 1628 that was richly decorated on order of the king, our story's manager, who, it seems, might have meddled with the design. The resulting ship was too tall and top heavy. On her maiden voyage, the ship rolled over and sank into Stockholm harbor in full view of hundreds of stunned onlookers.[1] I believe that this tragedy of engineering is not the rule for manager impact on software design—managers are not always the villains of the story (but they can be). Let's dig into how managers impact software design and grow our understanding of Conway's law.

Managers as Software Designers

Published in 1968, Conway's adage states that "organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations."[2] A humorous statement of this idea is, "If you have four groups working on a compiler, you'll get a four-pass compiler." In *The Mythical Man-Month*, Fred Brooks referred to this observation as *Conway's law*, and the name has stuck. Brooks noted that Conway's law predicts that the organizational chart "becomes intertwined with the interface specification."[3] Who defines this organization? Managers and executives. Wow, Conway says our specification and design work is being shaped by our managers.

Managers impact software design via their control over organizational structure, but the connection of your organizational structure to software design is subtle. In an organization, you'll have teams, subteams, and team members, which results in a hierarchy (Conway used the terms *committee*, *subcommittee*, and *coordinator*). Parts of the design are given to teams (and subteams) to work out in detail. Conway observed that the organizational and design structures match. Why? Organizational constraints on communication paths. If an organization is large and distributed, then every engineer cannot talk to every other engineer about everything, otherwise nothing would get done. For

Digital Object Identifier 10.1109/MS.2022.3180884
Date of current version: 22 August 2022

example, if two communicating top-level teams, A and B, design and build portions of a large system, then those two components will communicate (perhaps through a programming interface). But a component deep within the design of team B will not interface directly with a portion of the design deep within team A's design.

What this causes is a system design that mirrors the communication structure of the organization. Conway related several examples, the most famous of which (and the genesis of the humorous restatement of his adage) is compiler design: "A contract research organization had eight people who were to produce a COBOL and an ALGOL compiler. After some initial estimates of difficulty and time, five people were assigned to the COBOL job and three to the ALGOL one. The resulting COBOL compiler ran in five phases, the ALGOL compiler ran in three." This was a small-scale project. Conway also related two military services after "great effort" produced a joint weapon system design that was a copy of their organizational chart.

My start-up dealt with smaller systems. We built tools within Java IDEs that one or two programmers could fully understand. In the U.S. Air Force and in big tech companies, I worked on large, complex systems built by large and distributed teams. A key criteria to this discussion is that communications are constrained by the organizational structure. There are engineers or teams that don't communicate due to their places in the organizational hierarchy. It's in these cases that Conway's law applies.

It is reasonable to consider why every team or engineer doesn't communicate with every other engineer. They could, right? Perhaps just send an email. It seems like this would be a great way to work. The problem is scale. Conway stated, "Even in a moderately small organization it becomes necessary to restrict communication in order that people can get some "work done." Communication doesn't scale beyond a certain point. Meetings and emails (either one on one or among teams) take up time and effort. If you, as a software engineer, spend all your time on meetings and emails, then little engineering or "work," as Conway put it, can get done on your tasks.

Harnessing Conway's Law

Managers and executives control the organizational structure and (indirectly) the restrictions that it places on communications. Many factors are in play, including team size and location and the communication culture in the organization. Is a manager's impact on software design good or bad?

Vasa-like experiences do not have to be the norm in software design. My experience has shown me that most managers care deeply about the organization and want to improve it. Managers take definition of the organizational structure extremely seriously. They might not recognize the terms *Conway's law* or *The Mythical Man-Month*, but if you ask them whether the organizational structure impacts work outcomes, you will get a resounding yes. Managers fine-tune their organizational structures over time for better outcomes, including software design.

Managers make mistakes, and spectacular *Vasa*-like software disasters will continue to occur. Not all failures are due to Conway's law, but some are. I have some lessons for managers wanting to avoid their software project sinking in full view of a crowd. These are summarized in Table 1 and presented in more detail in the following section.

Lesson 1: Align Your Organization With Your Architecture

My experience in the U.S. Air Force was on large software projects that support planning and military operations. Military leadership (the managers) are focused on the organizational structure and its leadership, fine-tuning the organizational chart over many decades. They take this seriously. The organizational chart, however, is optimized for warfighting and readiness, not software development. In my experience, the military organizations that were best at software design realized the importance of software to their mission success and organized to support it. In particular,

Table 1. My Conway's law lessons for managers.

Do this	Avoid this
Lesson 1: Align the software-producing portions of your organization with the software architecture.	Lesson 2: Don't fix a bad or dysfunctional organization with a software system. Software cannot solve organizational problems, managers can.
Lesson 3: Be alert to product/software architecture problems caused by the organization and promptly adjust one or the other. Fine-tune over time (reorg).	Lesson 4: Don't avoid necessary reorgs. Manager meetings to cajole design changes are not strong enough to drive architecture changes.

this involved defining an organizational structure with strong communication among the teams responsible for major portions of an architecture.

I worked on many software modernization efforts in the U.S. Air Force. Many of these projects were targeted at migration of mainframe systems, often from the 1960s or early 1970s, to UNIX or Windows systems. Shifts in programming languages, tools, and databases were also often a goal. Overall, the idea was to save costs by using more modern software and hardware to run the system.

An approach I learned that was strongly correlated with overall project success was to run portions of the modernized system in production as soon as possible. I'll coin this *modernization in production*. Component by component, my team replaced parts of the old system and deployed the hybrid system until the old system was gone and we joyfully watched the mainframe roll out of the building. By confronting production complexities early, we reduced the chance of total project failure, and we were forced to understand the requirements. Piecewise replacement of components in production had the cost of significantly reducing our design freedom. The new system design had to closely mirror the old design.

In retrospect, my approach leveraged Conway's law to improve outcomes. The new system mapped well into the existing organizational structure and its communication paths because it closely mirrored the old design. Today, I wonder which facet helped the project outcome more. Risk reduction due to facing and understanding actual production system requirements (my focus at the time) or a clear mapping of the design into the existing organizational structure (which I hadn't considered). My approach worked because the goals of these software modernization projects was cost reduction, not producing the best new design.

> There are engineers or teams that don't communicate due to their places in the organizational hierarchy.

Lesson 2: Software Cannot Fix a Bad Organization

I've experienced software project failures when managers viewed software systems as a clever way to fix an organizational problem. Managers were thinking, "*We have several stovepiped portions of our organization that are politically tough for us to fix/reorg and by building a software system we'll get them to work together much better.*" This is a terrible idea when Conway's law is considered. A strong corollary of Conway's law, in my mind, is that you cannot fix an organizational problem with a software system. It's doomed because if the teams responsible for components of the design do not collaborate on interfaces (or do a poor job), the resulting system is unlikely to meet its objectives. Managers need to fix the organizational structure.

I recall the modernization of a logistics software system development that failed due to a "stovepiped" organizational structure that resisted change. Management prescribed a software solution to improve organizational collaboration and save costs. The project failed after years of work and was trying for all involved. I recall listening to a wing commander tell me about how painful the software field trials were for junior airmen who were trying to use the system. He cared deeply about the airman's plight, and this experience had soured him on software systems.

My active-duty experiences are from long ago, and I expect they are less common today. Why? Organizational structures are fine-tuned over time by managers for better outcomes, and the military is no exception. They have adapted to improve outcomes. Military leadership understands software development much better. To humorously illustrate this point, in my first U.S. Air Force assignment, when my new commander—an amazing pilot—first visited his new office, he came out and asked his secretary why there was a TV on his desk. It was a PC computer. This would be unimaginable today.

Lesson 3: Fine-Tune Your Organization and Architecture

At big tech companies, I've worked on very large software systems, often even larger and more complex than the U.S. Air Force systems I developed. Leadership at big tech companies, similar to that of the U.S. Air Force, takes curation of the organizational structure seriously. Leadership makes organizational changes, referred to as *reorgs*, to drive product innovation or improve reliability. The idea of doing this is not new, nor at all unique to big

ABOUT THE AUTHOR

TIMOTHY J. HALLORAN is a software engineer at Google, Pittsburgh, Pennsylvania, 15206, USA and a retired U.S. Air Force Lieutenant Colonel. Contact him at hallorant@gmail.com.

tech. In fact Brooks, in *The Mythical Man-Month* noted, "Conway goes on to point out that the organizational chart will initially reflect the first system design, which is almost surely not the right one. If the system design is to be free to change, the organization must be prepared to change."[3] Thus, Brooks (and I) advise that changing the organization can lead to improved designs and better outcomes.

There can be an architectural cost to reorgs. When significant software systems exist in production that are being maintained, a reorg can significantly change the communication structure such that the current design, which reflects the old organization, is no longer maintainable. This is sometimes called, wrongly I think, *technical debt*. It takes time for the software architecture to catch up to the new organization. Talking to senior managers, I've observed that they view the lifecycle of software as being pretty short, and that every few years major system-rewrite projects are somewhat inevitable. This was a bit shocking to me as I observed multidecade software lifecycles in the U.S. Air Force. Conway's law can help explain their view when reorgs are considered. A newly reorganized team C looks at their system, which was designed to be split between two parts of the organization as odd and unnatural: abstraction boundaries no longer make sense and the components and programming languages may differ. These rewrite projects tend to shift the system architecture back into harmony with the current organization.

Lesson 4: Drive Architecture With Reorgs

Managers, I'd argue, should use the power of the reorg to shift the software architecture. One bad approach is to avoid a reorg and instead form a "council" or "technical steering committee" made up of senior managers that meet to enthusiastically push for change on the system architecture without making the corresponding organizational changes. In my experience, some progress can be made, but it is often resisted and isolated to teams that become "targets" or the "focus of attention" of the counsel. Conway's law predicts this limited outcome. A better approach is for managers to propose a reorg that aligns the organization with the desired goals.

Reorgs are powerful, but do not overuse them. The risk is that your teams spend all their time rewriting existing code—to shift the architecture back into harmony with the organization—rather than innovating.

There's a strong connection between Conway's law and Brooks' law: both are savvy observations about the communication paths within organizations. Brooks' law states that "adding people to a late software project makes it later" because new people adds new communication paths. Conway's law states that a team's designs will mirror the communication paths set up by its organizational structure.

Both Conway and Brooks caution against thinking of software design or engineering as being linear, a fallacy Brooks called the *mythical man-month*. One engineer working for a month cannot be interchanged with 30 engineers working for a single day. That's because software tasks cannot be trivially partitioned, which Conway humorously expressed the following way:[2]

> *Assumptions which may be adequate for peeling potatoes and erecting brick walls fail for designing systems.*

As you manage teams and guide their work, I encourage you to consider the advice of Conway and Brooks. Keep in mind that managers have a tremendous influence on the architecture, but this need not be a bad thing if you design your organizations as carefully as you design your software systems.

References

1. A. M. Squires, *The Tender Ship: Governmental Management of Technological Change*. Cambridge, MA, USA: Birkhäuser, 1986.
2. M. E. Conway, "How do committees invent?" *Datamation*, vol. 14, no. 5, pp. 28–31, Apr. 1968.
3. F. P. Brooks Jr., *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA, USA: Addison-Wesley, 1982.