Difficulty scaling software development? [1]

Yogi Schulz

Successful software development projects, often with superior project management, unleash an insatiable demand for more complex software to be developed even faster and preferably cheaper. Completing more ambitious software goals requires scaling up software development.

At the recent Collision conference, Stephen Deasy, the Head of Engineering at Atlassian, explored the issues that affect scaling software development. Atlassian is a well-known supplier of software development tools, including Jira and Trello. He said, "As you scale your software development, you will encounter various dysfunctional patterns such as too many meetings, a push to add another person to the development team or creating a design that mirrors your org chart. Position your projects for productive success by keeping teams small, developing junior people, and adding entirely new teams when necessary." Due to aggressive goals, management often pushes project managers too often to ignore this guidance.

Scaling software development is not straightforward. Most non-IT managers intuitively think that simply adding more capacity, meaning people, will produce more software without adverse impacts on schedule, costs or quality. After all, that's been the long-standing experience with other functions like production, transportation, and warehousing. Unfortunately, this experience is irrelevant and even dangerous when applied to software development.

What should project managers think about when faced with the need to build on software development success? What makes scaling software development different? What are the failed approaches to avoid? What methods can lead to success?

Stephen Deasy reminded his audience of some historical perspectives related to the problem of scaling software development that continues to be valid and relevant in today's software world.

Project managers can use this experience and related research to diplomatically educate management about the best practices that will produce more successful software.

Conway's Law

In 1967, Melvin E. Conway, a computer programmer, stated, "Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure."

Common failures in project organization and software architecture that arise from Conway's Law include:

[1] How to cite this article: Schulz, Y. (2024). Difficulty scaling software development? *PM World Journal*, Vol. XIII, Issue VIII, August.

- Designing the project organization and the software architecture separately. The split will result in poor design choices and integration disconnects. Instead, design the project organization and the software architecture concurrently and expect to revise both as the project evolves.

- Designing a multi-level, hierarchical project organization. This approach will constrain software design to a small set of solutions that will likely exclude the optimum solution. Instead, design the project organization as flat as possible with no more than three levels. A flat project organization is more likely to arrive at the optimum solution.

- Planning for success. Counterintuitively, rigid or detailed planning will produce an inflexible software architecture and software design. Instead, we will achieve more successful software when we expect to encounter failure, assume learning, and recognize the need for repeated adaptation.

- Designing the project organization to involve large teams because the planned system is extensive in scope. Large teams will produce disappointingly little software because of the enormous effort consumed by communication and coordination. Instead, organize several small teams that each own a part of the functionality or architecture the team can develop independently of others.

- Designing the project organization as a set of functional teams such as software developers, business analysts, DBAs or QA staff. Functional teams will create silos that march to their own priorities and don't coordinate well. Instead, organize every team to include all the skills necessary to accomplish the functionality or architecture assigned to the team.

Dunbar's Number

In the 1990s, British anthropologist Robin Dunbar proposed a "cognitive limit to the number of people with whom one can maintain stable social relationships—relationships in which an individual knows who each person is and how each person relates to every other person."

Dunbar proposed that humans can comfortably maintain up to 150 stable relationships. Therefore, a typical person might have up to:

- 4 - 5 intimate relationships.
- 12 - 15 trusted relationships.
- 25 - 35 close relationships.
- 75 - 100 casual relationships.

This observation about social relationships limits the effective team size for many activities, including software development. Experience suggests the upper limit is 6 to 8 software

developers per working group. The project team can be larger in total because it will include other roles such as project manager, business analyst, database administrator, data modeler, and infrastructure administrator. However, increasing the number of software developers will produce less and less software per software developer due to more effort expended on communication and maintaining relationships.

Concepts like pair programming can improve quality, reduce rework, speed the orientation of new developers, and reduce the adverse impact on the project if a valued software developer leaves the project. While this idea adds to the headcount of software developers, pair programming increases the limit to 6 to 8 software developer pairs and cost.

This observation about the human cognitive limit also limits the number of concepts or variables even a talented software developer can juggle in their mind simultaneously. This limit should consciously influence the complexity of the proposed system architecture. The project team should favor designs with a higher number of smaller, crisply defined modules over designs with a smaller number of larger, more complex modules.

The Mythical Man-Month

Fred Brooks, the world's first computer architect and software engineer, first published The Mythical Man-Month in 1975. The book's central theme is that "adding manpower to a late software project makes it later."

This observation is accurate because software development projects cannot be perfectly partitioned into truly discrete tasks that software developers can work on independently. By contrast, manufacturing widgets or transporting materials or finished goods by multiple trucks or railcars are independent tasks.

All software projects require relatively constant communication among the software developers to build software modules and database schemas that can successfully interact with each other. Therefore, assigning more programmers to a project running behind schedule will make it even later. This counter-intuitive result occurs because the time required for:

- Orienting a new software developer on the project detracts from the effort available to the software developer performing the orientation.
- Intra-team communication will consume an ever-increasing percentage of the calendar time available.

This table of example team sizes illustrates how the number of communication channels and the communication effort grow exponentially with team size. In this example, each team member devotes only 1 hour to communicate with each channel or team member in a 40-hour work week.

Project Team Size	Calculation for Number of Communication Channels	Number of Team Communication Channels	Total effort capacity /week	Percent of capacity consumed for communication
5	5*(5–1)/2	10	200	5.0 %
10	10*(10–1)/2	45	400	11.3 %
15	15*(15–1)/2	105	600	17.5 %
25	25*(25–1)/2	300	1000	30.0 %
50	50*(50–1)/2	1,225	2000	61.3 %
100	100*(100–1)/2	4,950	4000	124.0 %
150	150*(150–1)/2	11,175	6000	186.3 %

As the number of project team members grows, their communication effort increases and their output decreases. On a large team, the communication effort will exceed the total team capacity. In this situation, no time is available to perform tasks on the project plan, and the project ceases to progress.

About the Author

Yogi Schulz

Calgary, Alberta, Canada

Yogi Schulz has over 40 years of Information Technology experience in various industries. Yogi works extensively in the petroleum industry to select and implement financial, production revenue accounting, land & contracts and geotechnical systems. He manages projects that arise from changes in business requirements, from the need to leverage technology opportunities and from mergers. His specialties include IT strategy, web strategy and systems project management.

Mr. Schulz regularly speaks to industry groups and writes a regular column for IT World Canada and for Engineering.com. He has written for Microsoft.com and the Calgary Herald. His writing focuses on project management and IT developments of interest to management. Mr. Schulz served as a member of the Board of Directors of the PPDM Association for twenty years until 2015. Learn more at https://www.corvelle.com/. He can be contacted at yogischulz@corvelle.com

His new book, co-authored by Jocelyn Schulz Lapointe, is "A Project Sponsor's Warp-Speed Guide: Improving Project Performance."