

# Scalable Distributed Organizations for Ultra-Large-Scale Software

Daniel J. Paulish  
Siemens Corporate Research, Inc.  
Princeton, NJ 08540 USA  
daniel.paulish@siemens.com

## Abstract

*Achieving ultra-large-scale software systems will necessarily require new organizational structures and processes. This position paper suggests an approach in which large systems are decomposed into smaller components that can be implemented by small teams. A collection of small development teams is then organized such that communications are profiled and controlled as reflected by the system architecture. Although such organization structures are far from optimal, they can help us achieve a next step and gain insights for implementing future ultra-large-scale software systems.*

## 1. Introduction

The number of features within a requirements database for a Siemens software product in the last few years has been typically in the 3000-5000 range. Recently, software products are being developed with feature counts in the tens of thousands. These large, complex, feature-rich software products are foreshadowing the ultra-large-scale software systems that will be more commonly developed in the future.

Marketing, sales, and product managers are putting extreme pressure on software engineers to develop new products quickly in order to reduce time-to-market. Although there are expectations that new tools and technologies will substantially increase software engineering productivity, today's organizations often attempt to reduce time-to-market by adding more software engineers to a project. There is a wide variety of skill and experience levels among individual software engineers and a correspondingly wide variety of development productivity levels.

It is well known that adding software engineers to a project results in lower overall productivity [1]. When new team members are added to an existing project structure they can slow down development as new staff are trained and integrated by the existing staff.

Today's agile processes seem to work best when team sizes are 10 engineers or less [2].

A recent trend in many software organizations is the increased use of outsourced or offshore resources. Often these organizations are attempting to reduce overall development costs by utilizing engineers that live and work in lower-cost countries; e.g., India, Brazil, China, or the countries of Eastern Europe. Such cost savings sometimes don't materialize, since additional communications are necessary with remote teams. Depending on the specific project situation, labor cost savings may be offset by increased training, travel, or telephone costs. But, as schedules get shorter and products become more complex, it's not likely that a single development site can recruit and house all the software engineers necessary to deliver and maintain future products.

## 2. Distributed Development Challenges

This paper explores some of the issues involved in forming scalable distributed software development organizations and provides some short term guidance on organizing new projects for global development. The basic problems that must be overcome for such organizations are related to time and distance. When large teams are spread across different locations in different time zones, communication among the engineers becomes more difficult. Not only do larger teams have more communication paths, but when they are distributed, the communication paths are longer. That is, the informal communications and familiarity that are present within a small collocated team are lost in distributed organizations.

As ultra-large-scale software systems are developed with large distributed development teams, the probability that many of the team members will have worked together on prior projects, know each other, or even meet each other decreases. Team members will likely reside in different countries, have different work habits, have different mother tongues, etc. Thus, communications difficulties will increase and must be

compensated for. These difficulties can generally be compensated for through design, where components interact in standard ways, or through technology, where communications among team members are cost effective and flexible for different individuals.

### 3. Global Software Development Approach

An approach to global software development (GSD) has been widely practiced in Siemens called the “Extended Workbench Model” (Figure 1). Using this approach, a small central team does early phase activities (e.g., requirements analysis, architecture design) and project planning while the remote teams focus mainly on construction activities starting with detailed design. Thus, the central team plans and allocates the work packages of the remote teams. The system architecture is designed as a collection of loosely coupled components, and integration is done continuously as features are implemented as vertical slices of the architecture [3, 4].

For the Extended Workbench Model, remote teams usually communicate frequently with the central team, but only communicate with other remote teams when they must work across a component boundary. The effect is that small agile development teams can often work more efficiently without the large communications overhead of every team member needing to communicate with every other team member across the entire project.

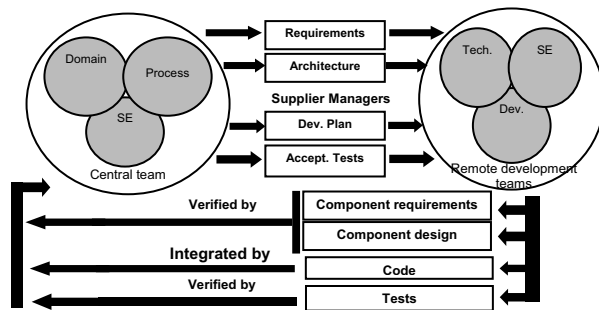


Figure 1. Extended Workbench Model

For ultra-large-scale projects with many distributed team members, there is a need to break the development work down into smaller components such that the project is organized as a collection of coordinated small teams, each of which is highly skilled and productive. We have set guidelines such that these small teams should be no larger than ten developers and there should be no more than ten teams at a development site or an organization of about 100 software engineers [5]. An example organizational

structure consisting of a central team and multiple remote teams is given in Figure 2.

We’ve set guidelines that a system architecture design should consist of no more than about 150 components and no component should be larger than 100 KLOCs of C++ code. Thus, such an approach to global software development could be applied to software systems up to 15 MLOCs. A system of this size is probably not considered ultra-large-scale. Thus, additional research and experimentation is necessary in the areas of software architecture, collaboration communication, and organizational structures to implement ultra-large-scale systems. For example, perhaps such systems should be designed as a loosely coupled collection of independently developed subsystems, each of which is smaller than 15 MLOCs.

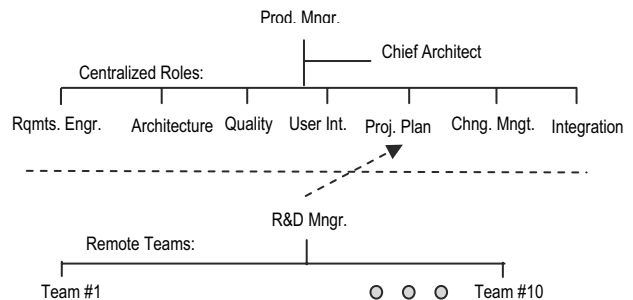


Figure 2. Example project organization

### 4. Research Questions

There are a number of questions that must be investigated before we will be able to organize to successfully implement ultra-large-scale software systems. Some of these questions may be answered for today’s development projects, but very different processes and tools may be used for future projects.

1. Given that the technical artifacts that are delivered to the remote component development teams are not adequate in specifying the precise work to be done, in which ways are they deficient? What methods can we use to increase the precision, completeness, and understandability of such technical artifacts? Can specifications, for example, be sent to a remote team or teams and good quality code will be returned that meets the specifications?
2. What strategies do the remote teams employ (and to what effect) to compensate for the deficiencies found in the received technical artifacts? Are there collaboration methods and tools that can help distributed teams quickly resolve issues and answer

questions?

3. What are the early warning signs that an organizational or project problem is imminent? Can communication patterns, for example, between the central and remote teams be used to predict future component integration problems?

One technique that has been useful for measuring and understanding the communications patterns among team members in a distributed software development project is Social Network Analysis (SNA) [6]. As illustrated in Figure 3, the nodes represent team members and the lines represent the communications between them. Note that the bottom three teams communicate only among their own team members and with the central team, while the top four teams communicate with each other. For this project, the top four teams were collaborating on a subsystem development and they were at two locations within the same time zone. The bottom teams were working on components that were loosely coupled from each other. Misalignment between the SNA diagram and the system architecture module diagram may indicate that project problems may be developing; for example, the system architecture is changing or inefficient or costly communications are impacting productivity.

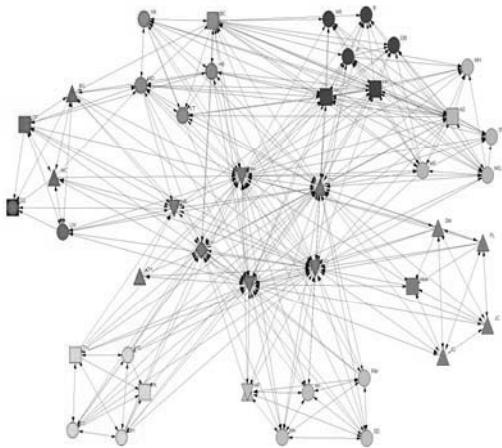


Figure 3. Social Network Analysis diagram

When we start posing some answers to these research questions, we will need to experiment with using new processes and organizational structures. Since it may not be practical to do such experiments on industrial projects, we may have to apply such new approaches to simulated projects, perhaps using university students distributed at multiple universities. Such an experimental project has been organized on a

small scale during the Global Studio Project (GSP) [5]. To increase understanding of the issues and practices of global software development (GSD), Siemens Corporate Research (SCR) has initiated a three-year experimental research project using university students to simulate a real Siemens GSD project (Figure 4). Development teams were set up at 6 universities with about 30 developers within 5 countries in 4 continents spread across 11 time zones [5]. The students implemented a “light” version of a Siemens software product, and they developed and delivered 67.5 KLOCs of operational product and test code.

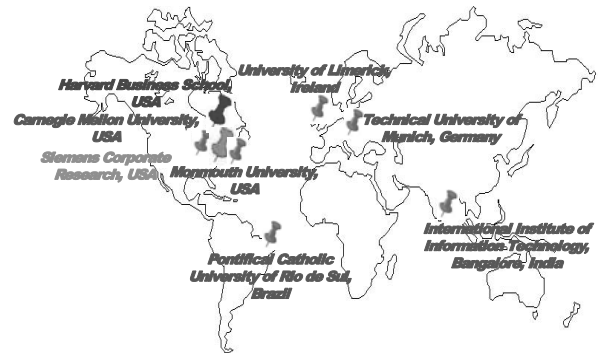


Figure 4. Global Studio Project

If a substantial number of distributed universities would collaborate with each other on an experimental large scale development project, we may begin to understand the organizational issues associated with building such large software systems. We may also be able to learn from the state-of-the-practice of other engineering disciplines; for example, from the organizational structures used to develop complex electro-mechanical products such as airliners and automobiles.

## 5. Conclusions

Future software systems will clearly become larger and provide increased functionality. We have assumed that the organizations required to develop these ultra-large-scale software systems will necessarily become larger. For practical space and staffing reasons, we have assumed that these future development organizations will be widely spread across multiple sites, spanning nations, time zones, and continents. Distance, time, language, and cultural differences will make communications among team members more difficult. Our ultra-large-scale software systems of the future will likely be developed by strangers who may never have worked together before. Informal

communications among team members around the water cooler will cease to exist.

We envision that the agile processes trend will continue such that individual component development teams will remain small and be part of a large coordinated project; i.e., a team of teams. This will require new organizational models that compensate for the loss of informal communications and exploit the advantages of new system architectures, tools, and technologies.

## 6. References

- [1] Boehm, B., *Software Engineering Economics*, Prentice Hall, Upper Saddle River, NJ, 1981.
- [2] Schwaber, K., *Agile Project Management with Scrum*, Microsoft Press, Redmond, WA, 2004.
- [3] Paulish, D., *Architecture-Centric Software Project Management*, Addison Wesley, Boston, MA, 2002.
- [4] Hofmeister, C., Nord, R., and Soni, D., *Applied Software Architecture*, Addison Wesley, Boston, MA 2000.
- [5] Sangwan, R., Bass, M., Mullick, N., Paulish, D., and Kazmeier, J., *Global Software Development Handbook*, Auerbach Publications, Boca Raton, FL, 2007.
- [6] Scott, J., *Social Network Analysis: A Handbook*, Sage Publications, Thousand Oaks, CA, 1991.