Communication and Collaboration in Global Software Development

Emil Folino Nielsen (emini745)
IDA, Linköping University,
Advanced Software Engineering, TDDD30

May 17, 2012

Abstract

Using Conway's Law to initiate a discussion this paper looks at the difficulties and best practices of global software development. Firstly the validity of Conway's Law in the 21st century is confirmed through two recent case studies. Secondly a review of three papers highlight the problems introduced by distributed software development and suggest best practices for conducting projects without delays and difficulties. Lastly the problems and best practices of global software development are discussed. In this discussion I contribute with my own experience in distributed software development. The paper conclude that despite the limited amount of literature used a lot of the difficulties and best practices are similar and that the best practices from both the literature and my own experience can be used as a starting point when development teams create their own best practices.

1 Introduction

In 1968 Melvin E. Conway published How Do Committees Invent? [1] where he coined what later came to be known as Conway's Law:

> "Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure."[1]

In his paper Conway used linear graphs to exemplify connections between systems and the organizations designing the system. Conway concludes that if two subsystems communicate then two sub organizations designing the subsystems will need to communicate and this will inevitably make the system reflect the organization designing it.

This paper is divided into two parts; in the first the validity of Conway's Law in the 21st century is discussed by reviewing two recent papers investigating how communication and organization influence software systems. The first paper is an empirical case study done by Microsoft Research using Windows Vista as the case and the second paper use a sample of matched-pair products to compare open-source to commercial software. The second part looks at the difficulties arising when distributed teams are used in contemporary software development by reviewing three articles. The reviews are used as a starting point for a discussion of the problems and best practices in global software development.

My contribution to the discussion will be my own experience from working at two danish software development companies while living in Sweden. My contribution will mostly be in 4.1 and 4.2 together with table 4.

The rest of the paper is organized as follows: section 2 is concerned with the validity of Conway's Law, section 3 is a review of contemporary papers discussing difficulties and best practices in distributed development, section 4 is the discussion where my own experiences are used in a discussion of problems and best practices in global software development, and section 5 is the conclusion.

2 Validity of Conway's Law

In this section two papers are reviewed to validate Conway's claim that the structure and design of software resembles the structure of the designing organization.

2.1 The Influence of Organizational Structure on Software Quality: An Empirical Case Study

Nagappan et al. investigate in The Influence of Organizational Structure on Software Quality: An Empirical Case Study [2] the relationship between organizational structure and software quality. The authors are affiliated with Microsoft Research and has a unique possibility to investigate software quality in huge releases by Microsoft. This paper is on a empirical case study of Windows Vista and post release failures are used as the quality variable to validate the eight organizational metrics proposed in the paper. The eight metrics are shown in table 1:

Table 1: Summary of organizational measures. Partly adopted from [2, Table 1]. The Name column is added by the author of this paper for clarity.

Assertion	Name	Metric
The more people who touch the code the lower the quality.	Number of Engineers	NOE
A large loss of team members affects the knowledge retention and thus quality.	Number of Ex-Engineers	NOEE
The more edits to components the higher the instability and lower the quality.	Edit Frequency	EF
The lower level is the ownership the better is the quality.	Depth of Master Ownership	DMO
The more cohesive are the contributors (organizationally) the higher is the quality.	Percentage of Organization Contributing to Development	PO
The more cohesive is the contributions (edits) the higher is the quality.	Level of Organizational Code Ownership	OCO
The more the diffused contribution to a binary the lower is the quality.	Overall Organization Ownership	OOW
The more diffused the different organizations contributing code, the lower is the quality.	Organization Intersection Factor	OIF

Nagappan et al. used the people management software at Microsoft and the version control system to extract the data needed to calculate the organizational metrics. The data was collected for Windows Vista with 3404 binaries and more than 50 million lines of code. In table 2 the organizational metrics are compared to traditional fault prediction metrics:

Table 2: Overall model accuracy using different software measures. Adopted from [2, Table 4].

Model	Precision	Recall
Organizational Structure	86.2%	84.0%
Code Churn	78.6%	79.9%
Code Complexity	79.3%	66.0%
Dependencies	74.4%	69.9%
Code Coverage	83.8%	54.4%
Pre-Release Bugs	73.8%	62.9%

The authors conclude that the organizational metrics proposed in the paper are efficient at predicting post release failures, but they are hesitant to draw general conclusions from the results. They realize that more research is needed to validate these results. In the context of Microsoft and the Windows operating system the authors have confidence that their results can be used and teams at Microsoft have started to use the results of this research when teams are constructed.

2.2 Exploring the Duality between Product and Organizational Architectures: A Test of the "Mirroring Hypothesis"

MacCormack et al. compares six matched-pair products in Exploring the Duality between Product and Organizational Architectures: A Test of the "Mirroring Hypothesis" [3]. The matched-pairs consist of an open-sourced product and a product created by an organization. The authors calculated the visibility of each source file for all 12 products and compared the two populations using a Mann-Whitney-Wilcoxon test of differences in means. The results are shown in table 3:

Table 3: Propagation Cost Measures for each Matched Pair. Adopted from [3, Table 4].

Product Type	Loosely Coupled	Tightly Coupled	MWW Test Statistic: In-Degree ($p<0.1\%$)	MWW Test Statistic: Out-Degree ($p<0.1\%$)
1: Financial Mgmt	7.74%	47.14%	U=194247 z = 12.6	U=189741 Z = 11.6
2: Word Processing	8.25%	41.77%	U=410832 z = 8.3	U=549546 Z = 22.9
3: Spreadsheet	23.62%	54.31%	U=174030 z = 12.3	U=180024 Z = 13.6
4a: Operating System	7.18%	22.59%	U=49.4Mn z = 25.6	U=65.0Mn z = 69.5
4b: Operating System	7.21%	24.83%	U=594522 z = 6.2	U=786574 Z = 20.8
5: Database	11.30%	43.23%	U=90814 z = 3.3	?U=126564 Z = 14.1

The authors conclude that a product's architecture mirrors the structure of the designing organization. The loosely coupled open source community produce far more modular designs than more tightly coupled organizations. Table 3 shows a significant difference between the matched pair products and the visibility of each product's source files.

3 Review of Articles

In this part of the paper three articles from 1999, 2009, and 2011 are reviewed. In these papers the coordination and communication problems of geographically distributed are discussed. The criteria for choosing these articles were the number of times an article was cited with a focus on the most recent articles.[1]

3.1 Architectures, Coordination, and Distance: Conway's Law and Beyond

In Architectures, Coordination, and Distance: Conway's Law and Beyond [5] James D. Herbsleb and Rebecca E. Grinter conduct a case study at a Lucent Technologies Department distributed in United Kingdom and Germany. The case study was conducted in two phases. First 10 managers and technical leads were interviewed and the main communication problem was identified as the integration part of software development. In the second phase eight additionally interviews were done with focus on integration. A thorough project retrospective done earlier by the company was used to support claims from the interviews.

The authors identified five major areas where the communication problems between the dispersed teams seemed to arise: Unplanned contact, Knowing whom to contact, The difficulty of initiating contact, The ability to communicate effectively, and Lack of trust. The first two areas can be contributed to the fact that

[1]The articles were chosen using Scopus (http://www.info.sciverse.com/scopus) as the search tool and the citations are the citations in Scopus.

geographically dispersed employees do not share the same relations as with the co-located employees. The rest of the areas are mostly with regard to the different cultures and native languages at the two sites. The British and German cultures are very different and the culture differences created a lot of problems for the teams during integration.

The authors conclude that it is very important to be aware of and adhere to Conway's law and try to design systems in a way so that they resemble the organization designing them. These modular designs will make development easier because teams can focus on specific modules without limiting the whole system. Furthermore the authors recommend that the dispersed teams meet in the beginning of the project to make the other teams more than just an email address and a voice in a phone. This helps cross-site problem solving because the teams now know who and how to contact the other team.

3.2 Communication, Knowledge and Co-ordination Management in Globally Distributed Software Development: Informed by a scientific Software Engineering Case Study

The case study reported in Communication, Knowledge and Co-ordination Management in Globally Distributed Software Development: Informed by a scientific Software Engineering Case Study [4] was conducted with the development of ePCRN[2] as the case subject. The ePCRN project consisted of two software development teams, one scientific computing team, and a domain specific team from three different university distributed in the United Kingdom and United States.

The authors report on several difficulties throughout the project mainly in three different areas. Firstly the development teams had problems eliciting requirements from the scientific computing and domain specific teams, because these teams experienced difficulties in explaining their needs. Secondly the limited overlap between teams due to time difference made communication slow and even non-existing. Furthermore unplanned encounters were far more unlikely to happen between members of the distributed teams. This lack of communication lead to a lot of duplicate and redone work. Lastly staff changes in both project management and development teams lead to inconsistent work and limited knowledge transfer. This lead to a low comprehension of project and task status, creating even more delays and decision making was hindered by the changes in project management.

3.3 Coping with Distance: An Empirical Study of Communication on the Jazz Platform

In Coping with Distance: An Empirical Study of Communication on the Jazz Platform [6] the authors set out to prove or disprove that distributed development teams create delays and communication overhead despite new collaboration tools specifically designed for distributed teams. The authors try to answer three study questions:

- Question 1: Does lack of work-hour synchronicity across different sites introduce delay in project communication?
- Question 2: How do distributed teams cope with communication delays?

[2] ePCRN is a tool for designing and running clinical tests in primary care settings

- Question 3: Do the roles of team members' influence their communication behavior and their social networks?

The subject of the case study is a large distributed team of 300 developers in 35 locations and 19 time zones. The team uses the IBM Jazz platform and the development has been ongoing for 3 years. The study was conducted during a 16 month period where 10,967 work items were created in the system and of these 4,311 had comments from at least two developers. These 4,311 work-items were used in the case study.

In the conclusion of the paper the authors propose a set of best practices to minimize delay. The authors highlight the importance of having work hour overlap when collaborating on work items. Otherwise the delay could extend to several work days because response time skyrockets. Distributing related work items into functional teams residing within the same time zone is the other important finding in this case study and this will significantly decrease delays.

4 Discussion

In this section some of the difficulties introduced with distributed development will be discussed and compared with my own experience in distributed development. First I will outline my own experience and the tools used too overcome the distance and in the second part I will compare and discuss the findings from section 3 with my own experience. The last part of this section will look at the validity of the findings in this paper.

4.1 My Own Experience

As described in the introduction I work for two software development companies in Denmark while living and studying in Sweden. In the following I will describe the companies and tools used briefly and then touch on problems and best practices.

TracTrac: A small company involved in GPS-tracking of sporting events. Started doing tracking for orienteering events, but have moved more towards sailing, running, and cycling. I have been working with a web app used by those arranging the sporting events to setup the races. The company is distributed in three countries. In Spain one of the managers and a full-time developer works, in Denmark the managing director and two part-time developers works and then I am a part-time developer in Sweden.

The tools I use as a developer at TracTrac are Skype and mail for communication, trac for project tracking and tickets and Subversion is used as the version control system.

The problems I have experienced while working for TracTrac are mostly related to two areas. The first issue is code produced by developers no longer at the company and not following any particular standard. This make it particularly hard to fix bugs and understand what the developer was thinking while constructing the code. The second issue is unclear descriptions of tickets. The tickets are normally written by other developers and not a non-technical user which makes most of the tickets technical and again the issue of understanding another developer's thoughts arises. The cultural differences between Spain and the Nordic countries create a problem because of different working hours in these countries.

Things that help the communication and collaboration at TracTrac is the ongoing establishment of development, testing, and deployment procedures. Another good thing is the chain of command - everybody knows their role in the company.

Olesen & Nielsen: Is my own software development company together with a childhood friend living in Aarhus, Denmark. We develop web apps and mobile apps for iOS and Android.

Tools at Olesen & Nielsen: Skype for communication, our own simple project management tool and CRM for project tracking and customer relations and we use Git for version handling.

In Olesen & Nielsen the main problems are divided into three areas. First of all it is hard to justify working for Olesen & Nielsen without pay instead of doing school work or for TracTrac. Despite both of us in Olesen & Nielsen are Computer Science students we have too diverse skill sets to be working on the same projects. We are not good enough too explain and document our ideas and they area often only ideas in our heads.

As with most start-ups the knowledge transfer and vision for the company is vaguely described in the beginning, but during the process of working and discussing we are starting to align our views of the company. We both share a passion for simplicity and this helps in constructing software and building our company.

4.2 Comparison

The subjects of the three case studies in section 3 are all large development projects consisting of several hundred primary stakeholders distributed in several countries. My experiences are in both cases from smaller companies with few primary stakeholders involved. Despite this difference in size a lot of the difficulties and best practices are similar in both the large and smaller development efforts. The difficulties and best practices are summarized in table 4.

At the companies described in 4.1 all employees reside in the same time-zone, but because of the cultural differences between Spain and the nordic countries I have experienced that it is easier to initiate contact with the developers working in Denmark than in Spain. Both [5] and [6] report on problems with culture and time-zone differences and the need for coordinating dependent work items within the organization to maximize work-hour overlap to minimize project delay.

At both of the companies depicted in 4.1 I describe problems with understanding ideas and requirements from other developers and management. This leads to delays for all involved stakeholders because time is needed to described what the work item actually mean and involve. This is an expensive part of development because more than one developer is involved in the clarification and elicitation of the actual requirements of the work item. In [4] these difficulties are described and Taweel et al. propose better management of the work-item requirements and creating methods for eliciting the right requirements as the solution. I agree with the authors of [4] on their conclusion and a lot of the development time could be saved if the right requirements and intents of a work-item are described early in the project. Furthermore these requirements should be determined not by developers or other technical employee, but by the actual users or a person in a product owner role, whether inside or outside the organization.

In [5] Herbsleb et al. propose a best practice of designing the system by adhering to Conway's law and realizing that the designed system will inevitably end up mirroring the designing organization. This realization is interesting because normally a design of a system would be forced on the developers of the organization, not as Herbsleb et al. propose where it is the other way round. By forcing the design to resemble the organization some of the difficulties of distributed development can be erased. These findings are confirmed in [6] where the authors propose a best practice of co-locating sub organizations working on dependent work items at least within the same time-zone.

Both [5] and [4] report on the fact that unplanned meetings between developers are important to project delays and overall product quality. I find it very interesting that encounters at the coffee machine or at lunch and

Table 4: Comparison of experiences with literature

	Own Experience	Literature
Problems	Code written by developers no longer at the company. Unclear description of tickets. (TracTrac) Too little development time. Too diverse skill set. Plans in our heads. (Olesen & Nielsen)	Little or non existing unplanned contact. Lack of contact persons. Difficulty of initiating contact. Effective communication. Lack of trust. Limited time overlap and communication channels.[5] Problems with eliciting requirements from domain specific teams. Staff changes and limited knowledge transfer leading to a lack of comprehension for project status.[4]
Best practices	Chain of Command. Ongoing establishment of standards. (TracTrac) Passion for simplicity. Better knowledge transfer. (Olesen & Nielsen)	Increase face-to-face time in the beginning of project. Modular designs. Adhering to and designing with Conway's Law.[5] Improve requirement eliciting mechanisms to allow for better management of interrelationships and dependencies. Create methods to elicit the right requirements from domain specific experts.[4] Work hour overlap when working in different time-zones. Distributing dependent work items within the same time-zone.[6]

not the planned encounters are those that solve problems. These types of encounters are hard to create when the development teams are in different locations. Herbsleb et al. highlights in [5] the importance of early encounters between the distributed teams to create bonds and establish real-life contacts between developers creating the possibility for more unplanned encounters despite the distributed teams.

4.3 Validity Threats

Because the amount of literature, studies, and experience used for this paper is very limited the results and findings cannot be extrapolated to all situations and organizations. Most of the best practices and difficulties described in the papers used to validate Conway's law, the case studies used in section 3, and my own experience from smaller development projects are similar and this finding increases the validity of these findings.

5 Conclusion

With Conway's Law from 1968 as the starting point this paper have looked at the difficulties and best practices of distributed teams in contemporary software development. First of all the validity of Conway's Law was discussed and two recent papers [3], [2] both conclude that the design of a system mirrors the organization designing the system. According to these two case studies Conway's Law is as relevant for software development today as it was in 1968.

Throughout the rest of the paper focus is on the problems that arise when distributed teams are introduced and how organizations and scholars have developed best practices on how to cope with distance. The difficulties and best practices are summarized in table 4. As described in 4.3 the literature used for this paper is very limited and further investigation and discussion would be both interesting and beneficial for both the academic as well as the corporate world. Despite the limited literature a lot of the findings were similar in both small and large development efforts and the best practices listed in 4 can help most distributed development efforts. The best practices provide a good starting point for developing the team's own best practices to achieve the benefits of distributed development without the delays and difficulties.

References

[1] Conway ME. How Do Committees Invent?. Datamation, Vol 14, No 4. 1968;p. 28–31.

[2] Nagappan N, Murphy B, Basili V. The Influence of Organizational Structure On Software Quality: An Empirical Case Study. Association for Computing Machinery, Inc.; 2008.

[3] MacCormack A, Rusnak J, Baldwin C. Exploring the Duality between Product and Organizational Architectures: A Test of the ÒMirroringÓ Hypothesis; 2011, working paper.

[4] Taweel A, Delaney B, Arvanitis T, Zhao L. Communication, knowledge and co-ordination management in globally distributed software development: Informed by a scientific software engineering case study; 2009. p. 370–375.

[5] Herbsleb J, Grinter R. Architectures, coordination, and distance: Conway's law and beyond. IEEE Software. 1999;16(5):63–70.

[6] Sindhgatta R, Sengupta B, Datta S. Coping with distance: An empirical study of communication on the Jazz platform; 2011. p. 155–162.

I promise that I have written this term paper myself and that I have handled references and quotations according to the instructions on the course home page.