Observations on Conway's Law in Scientific Computing

Jorge Aranda, Steve Easterbrook, and Greg Wilson
Department of Computer Science,
University of Toronto
Toronto, Canada, M5S 2E4
{jaranda, sme, gvwilson}@cs.toronto.edu

ABSTRACT
We describe the structure of organizations and products of scientific computing projects using Conway's Law as a lens to guide our observations. Our organizational findings include highly unconventional work structures, loose project membership and roles, low team coordination and awareness, and the dependence on liaisons for geographically distributed development. These characteristics are reflected in the ambiguous goals and requirements, organic growth, and loose boundaries of the resulting products.

1. INTRODUCTION
The emerging literature on socio-technical congruence tends to focus on commercial and open-source software development groups. Although this focus is convenient due to the immediate applicability of its observations to the software industry, it ignores other essential, but less conventional, software development domains. One such alternative domain is scientific computing, which is subjected to the same coordination principles of more conventional projects, but differs in many qualitative attributes.

One principle that software projects have been observed to follow is Conway's Law [3]. Briefly stated, "the structure of a product resembles the structure of the organization that designed it". Over the years, this phenomenon has been observed, expanded, and analyzed in depth (for instance, see [5]). It is, perhaps, one of the few statements that can be seen as general principles in our field and beyond – Conway argued that it is true of all kinds of designed products, software-based or not, since it addresses the essence of the task of coordinated design.

In this paper we look at scientific computing projects through the lens of Conway's Law. Based on an ongoing case study of such projects, we draw observations about the coordination and structure characteristics of the groups and products of this domain.

STC 2008, May 10, Leipzig, Germany.

2. SCIENTIFIC COMPUTING
We consider scientific computing projects to be those that are integral to the process of discovery and validation of scientific knowledge. Examples include simulators of climate models and visualizers of medical phenomena. We exclude trivial programs, proof-of-concepts, and non-trivial software written by scientists for non-scientific purposes, such as derivations of their work for commercialization.

Modern research depends on these scientific computing projects to a surprising degree. It is difficult to find a field of study that does not develop software at least tangentially; for many fields (Physics, Biomedicine, Meteorology, to name a few) it has become an absolute necessity.

Scientific computing is unique due to the combination of several factors. It is routinely performed by people without software training, on a part-time basis, over long periods of time. It has ambiguous and changing goals. It addresses specialized, thorny, unexplored problems. It tends to be computationally expensive. It presents extraordinary challenges for quality assurance since, often, the "right answer" is unknown, and defects are hard to detect. There is rarely an external market for its products, and its main users are often its own developers.

However, for all the differences that scientific computing presents in comparison to the software industry, it still holds design and coordination challenges common to software projects in general. We can therefore hypothesize that coordination phenomena observed in commercial and open source software, such as Conway's Law, should be found in scientific computing projects as well.

3. CASE STUDY
We began to interview scientific groups as part of a larger multiple-case exploratory case study [7] on describing how software is developed "in the wild" [1]. Our goal was to discover how scientific groups coordinate their software systems, and to compare their practices to those of commercial groups of similar sizes. Our interest in scientific computing grew out of the realization that its bottleneck is not computing power, but the skills, coordination, and tool adoption of its developers (see for instance [6] and [2]).

3.1 Methodology and execution
We consider our units of analysis to be the individual scientists that form part of scientific computing efforts (as we

will see, projects would have been a more problematic unit of analysis). To fit our selection criteria, scientists: (a) had to be developing scientific computing software (according to our previous definition); and (b) for convenience, needed to be located in Toronto. Although the large majority of our cases came from the academic environment (as opposed to, for instance, government research laboratories or other scientific bodies), this was not one of our selection criteria.

We interviewed fifteen people in eight different scientific domains. We began our data collection in the summer of 2007. Our interviews were semi-structured, lasted between one and two hours, and covered contextual characteristics of each project, as well as project goals, requirements elicitation, project management and lifecycle, team structure and roles, documentation practices, tool use, and quality assurance processes. We did not conduct our interviews with an examination of Conway's Law in mind, and in most cases we did not examine the software that our participants produced. These are threats to the internal validity of our results. However, the topics we covered allowed us to draw several preliminary but nonetheless useful observations.

3.2 The cases

The following brief descriptions of our cases are only intended to convey a basic impression of their nature.

Particle Physics: We talked to six people in this area. Most of them work with the ATLAS project, an experiment on high-energy particle collisions based on the accelerator in CERN, the European Organization for Nuclear Research. It is one of the largest scientific efforts in history: CERN reports that there are 2,100 participating physicists, including 450 students, from more than 167 universities [4]. Five of our interviewees were graduate students either contributing directly to the ATLAS software or adapting pieces of it to suit their own research needs. Our sixth interviewee was a manager of computing services for this group.

Forestry: Our Forestry interviewee was a Ph.D. student researching, along with his advisor, how forests develop under different management strategies. He uses computer-based stochastic simulations to model forest development.

Urban planning: We had one interviewee in this area, a Ph.D. student working, in collaboration with two professors and several other graduate students, on an urban traffic simulator. The aim is to model city-wide effects of public transit policies. The team needed to build its simulator from scratch, as one of its main innovations is the increased level of detail of its model, which is not supported by other available simulators.

Oncology: The oncology project involves the development of minimally invasive procedures to cure cancer. It determines the amount of heat to be applied to a tissue, through a laser, in order to heal the corresponding organ. The core of the software was written eight years ago by a professor as a prototype; to date it continues to be expanded and refined by people at the research division of a local hospital. A total of six people have worked on it throughout this period.

Medical Imaging: This project consists of the processing of MRI images to automatically detect some forms of cancer. It began as one component of a Ph.D. student's thesis project; a professor and a summer student have contributed in different capacities.

Atmospheric Physics: The Ph.D. student that we interviewed in this area is developing a simulator to help explain the formation of atmospheric currents and clouds in Jupiter. As with other cases in our pool, the simulator itself is part of the innovation in this project, since it implements atmospheric vectors previously not used in the field.

Biotechnology: There were two separate biotechnology projects in our pool. The first is a medium-scale geographically distributed effort to develop software to examine protein interactions visually. Our interviewee was a professor whose group contributes to this project significantly. The project has evolved a steering committee layer, and is one of the prominent software systems in the field.

The second Biotechnology case is a protein interaction database portal that concentrates information on protein interactions. Although it started as an element towards the Ph.D. thesis of its main (and then only) contributor, it quickly grew due to the demand for such a database portal in the community. Currently, five people participate in its development, in different capacities.

Earth Sciences: We interviewed an Earth Sciences professor whose students have developed a series of graphically-intensive rock and soil engineering tools that analyze the stability of various kinds of structural formations. Several of the tools that his group developed over the years became commercial products after significant modifications.

4. OBSERVATIONS

We make the following observations on the structure and coordination dynamics of our cases.

4.1 Organizational structure

The term "organization" might be a misnomer to refer to the groups of people that drive the projects we studied. With this we do not mean that they are disorganized, but that they are less formally defined than traditional software development projects. They are still, however, organizations in the sense that they are groups of people working together. We list some of the characteristics that set them apart:

Loose membership and boundaries: It is often difficult to determine who is a member of a scientific computing organization and what constitutes membership. There are several kinds of roles in the membership borderline: occasional contributors, past contributors available for relatively frequent consultation, non-coders that collaborate by providing guidance or by creating new scientific approaches to an issue facing the project. Even current code contributors present a membership problem: we can see the ATLAS people as forming part of the same large project, but it might be more convenient to picture them as developing independent projects that happen to be based on the same software platform.

Roles and hierarchy: These teams do not tend to follow the conventional, relatively rigorous hierarchy of commercial organizations. Developers are usually also graduate students and professors, and they have flexible roles and a less stringent hierarchy. Developers possess a greater design autonomy. Quality assurance, build coordination, and requirements analysis, among other specialized roles, are absent from most projects. However, as projects grow, the roles of system and database administration may be formally assigned to a (usually non-academic) member. Project leadership does not depend on hierarchical position or seniority, but on proximity to and specialization in the relevant research problem.

Team structure: There are, arguably, at least two team members in most of these teams: a junior academic figure, such as a graduate student, in charge of the development, and a senior academic figure, such as a professor, rarely involved in programming tasks, but participating through guidance and some design activities.

From this basic structure, projects grow in two directions. One is towards additional *technical* human resources: hiring a database administrator, or a summer student charged with improving the user interface. This occurs when the technical challenges of the project clearly surpass the capabilities of current team members. The other direction is growth towards additional *scientific* human resources: more researchers getting involved in the project, generally with the purpose of addressing more, similar, research questions. This occurs most often when the team's academic work is centered on a research problem posed by a leading professor, who assigns pieces of the problem to their students.

Communication: Project communication volume tends to be extremely low in comparison to commercial software development. Except for the two largest projects in our pool, coordination almost exclusively has a single focal person: the project lead, the person with the main research problem. Even these communication transactions are scarce and ineffectual – we had instances where our interviewees learned information vital to their projects by talking to us. Team awareness is nearly non-existent. This is not to say that the group does not communicate, but that it discusses scientific challenges, not software development activities.

Knowledge transfer through liaisons: The two largest cases in our pool are geographically distributed. For them, coordinating presents a challenge similar to that experienced by many current global software development projects [5]. These projects rely on the participation of people acting as liaisons among sites – typically post-doctorates or graduate students in an internship. Their role is to transfer tacit knowledge about the technical and scientific problems that the local team encounters. In the ATLAS project, in particular, graduate students report being constantly stumped with the arcane requirements of the software platform until one such liaison contributes the missing piece of the puzzle.

4.2 Product structure

Just as "organization" is a word ill-fitted to describe scientific software development teams, "product" is an inappropriate term to describe the results of their work. Theirs are products that rarely have releases, external users, or clear boundaries. Some of their notable characteristics are:

Undefined requirements and goals: It is rare to find clearly stated targets in this domain. For example, contributors to the ATLAS project report first becoming familiarized with their research problem, with their complex software platform, and with the topics in the mailing lists, and then developing a sense of what "needs to get done next". Requirements documentation is also a rarity, especially when the projects are breaking new ground – and, because of their nature, most of them are. They are never really finished, and though our interviewees used terms such as "eighty percent done", they do not seem to mean that they have truly quantified their progress, but that they sense that there is (always) a little more work to do. Projects are abandoned, rather than finished: when their research problem ceases to be interesting, or when their developers, mostly graduate students, eventually leave after obtaining their degrees and nobody else picks up on their work.

Releases, work backlog, and discipline: On the surface, the previous observations (no upfront planning, adaptive approach) might lead the reader to characterize these groups as performing agile software development. However, they differ from agile development in significant ways: They do not have release cycles. They do not explicitly and periodically prioritize their work backlog (usually they do not even have an explicit work backlog). They do not do test-driven development. They do not hold periodic (daily, weekly, or monthly) meetings to discuss the status of their software projects. All of these are common features of conventional agile development, and they are absent from the large majority of our cases.

Organic growth: The kind of growth that these products experience seems to be the result of their ambiguous goals and team structures and of their low and loose organizational coordination. In general, these projects grow organically, slowly, and often in unexpected directions. Some of them began as side projects or scripts written to get one small problem out of the way; their author shared them with a broader community, sparking its interest, and they took a life of their own, to the extent that they became the main interest of their authors. Some others, built to address a general problem, were picked up, used as a platform, and extended by other developers to address variations of that general problem, or more specific problems, years after. Their contributors have only a rough sketch of the future directions of the project; their strategy seems to be to take development one feature at a time, and to then move on to the feature that "naturally" should follow. This lack of planning appears to affect the architecture of the systems. Their modules rarely, if ever, follow basic architectural principles such as information hiding. Contributions to a platform are sometimes incompatible among themselves, and they easily reach the point where they are of no use to anybody but their authors and, perhaps, their future students.

Loose product boundaries: It is hard to draw a clear boundary delineating these products. Their modules tend to branch out and become incompatible, although formally they may continue to be a part of the same package. They may interact with other external products as a series of workflow pipes to address larger research questions, to the point where they practically cease to be a product themselves and begin to be treated as modules of a different system. They may also be adapted to become commercial tools, in which case they shed many of their features and refine some key functionality. In these cases it is questionable to consider the modified commercial product an "adaptation" of the original scientific product: in our Earth Sciences case reportedly not a single line of code produced in the academic environment survived commercialization.

4.3 Linking organization and product

Although these are preliminary observations, the previous subsections have noted significant evidence describing how the structure of scientific software development groups is reflected in the structure of their products. Both the teams and their products have ambiguous boundaries. Coordination and awareness are low, and they tend to radiate from the project lead; the result is a central platform with extensions and modules that grow in disparate directions, eventually becoming incompatible to each other. Cohesion through code or documentation is low; to alleviate this problem geographically distributed teams need to modify their structure by resorting to the establishment of liaisons.

4.4 Other observations

These observations are not related to Conway's Law, but are still relevant for an understanding of the domain.

High-performance computing: Scientific computing is often confused with high-performance computing. However, although the scientists we interviewed sometimes use high-performance tools and equipment, their use and efficiency is far from being their main concern. This might be because they have no need for high-performance tools or, more likely, because they are so difficult to use and take advantage of, even for people with good programming proficiency, that scientists do not invest time to learn to use them in depth.

Tools and practices: Scientific computing is generally performed by people that are not trained in software development and that see such training as a waste of time in comparison to further study in their area of knowledge. They do not use current software tools (version control repositories, defect databases, debuggers, or IDEs) or practices. Quality assurance is an afterthought, partly because of the challenge of testing a process for which the right answer is unknown, but also because of a lack of testing discipline. This results in low efficiency and a high rate of project failure.

Scientific vs. commercial software development: Although most scientific computing projects are small, the contrast with commercial software development in small companies is stark. While small companies present a wide range of approaches to software development [1], scientific computing groups are mostly quite similar. This is unfortunate because they are similar in their general lack of technical expertise, tool adoption, process, and professional discipline. There are exceptions, and our case pool includes one, but this seems to be a pervasive pattern in the domain.

Another difference between the two domains is the lack of market pressures on scientific computing groups. They do not usually respond to external users. The survival of their groups rarely depends on the success of their software projects, and their funding does not depend on code quality. They do not validate their own code, and therefore they only catch the most glaring defects before the publication of their results. Scientists do report feeling the pressure to deliver results – students to their advisors; advisors to their funding sources. But it is a pressure that pushes them towards publication, not towards software quality, which as a consequence is often overlooked.

We have previously observed how the catastrophic tone of the literature on commercial software does not match our observations. It turns out to be much more fitting to scientific computing. Considering our society's dependence on science, and science's dependence on software, this is an area that demands further attention from our community.

5. CONCLUSION

In this paper we analyzed scientific software development through the lens of Conway's Law. We discovered highly unconventional organizational structures and coordination mechanisms, and we traced the effects they have in their resulting products. These observations should be valuable for researchers of socio-technical congruence, as we explore the phenomenon in an unusual setting, and for researchers of scientific software development, due to our focus on coordination rather than on tool efficiency.

6. ACKNOWLEDGMENTS

We thank the participants of our case study for their invaluable collaboration, and Neil Ernst, Jennifer Horkoff, and Jonathan Lung for their feedback on an earlier version of this paper. This research was funded by NSERC.

7. REFERENCES

[1] J. Aranda, S. M. Easterbrook, and G. V. Wilson. Requirements in the wild: How small companies do it. In *RE '07: Proceedings of the 15th IEEE International Requirements Engineering Conference*, pages 39–48, Delhi, India, 2007.

[2] J. Carver, R. Kendall, S. Squires, and D. Post. Software development environments for scientific and engineering software: A series of case studies. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 550–559, Minneapolis, MN, USA, 2007.

[3] M. E. Conway. How do committees invent? *Datamation*, 14(4):28–31, 1968.

[4] European Organization for Nuclear Research. Atlas experiment public information, 2008. `http://atlas.ch/`.

[5] J. D. Herbsleb and R. E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *ICSE '99: Proceedings of the 21st International Conference on Software Engineering*, pages 85–95, Los Angeles, CA, USA, 1999.

[6] G. V. Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94(1):5, 2006.

[7] R. K. Yin. *Case Study Research: Design and Methods (3rd Edition)*. Sage, 2003.