Socio-technical Aspects in Software Systems

Alberto Bacchelli

What brings you here?

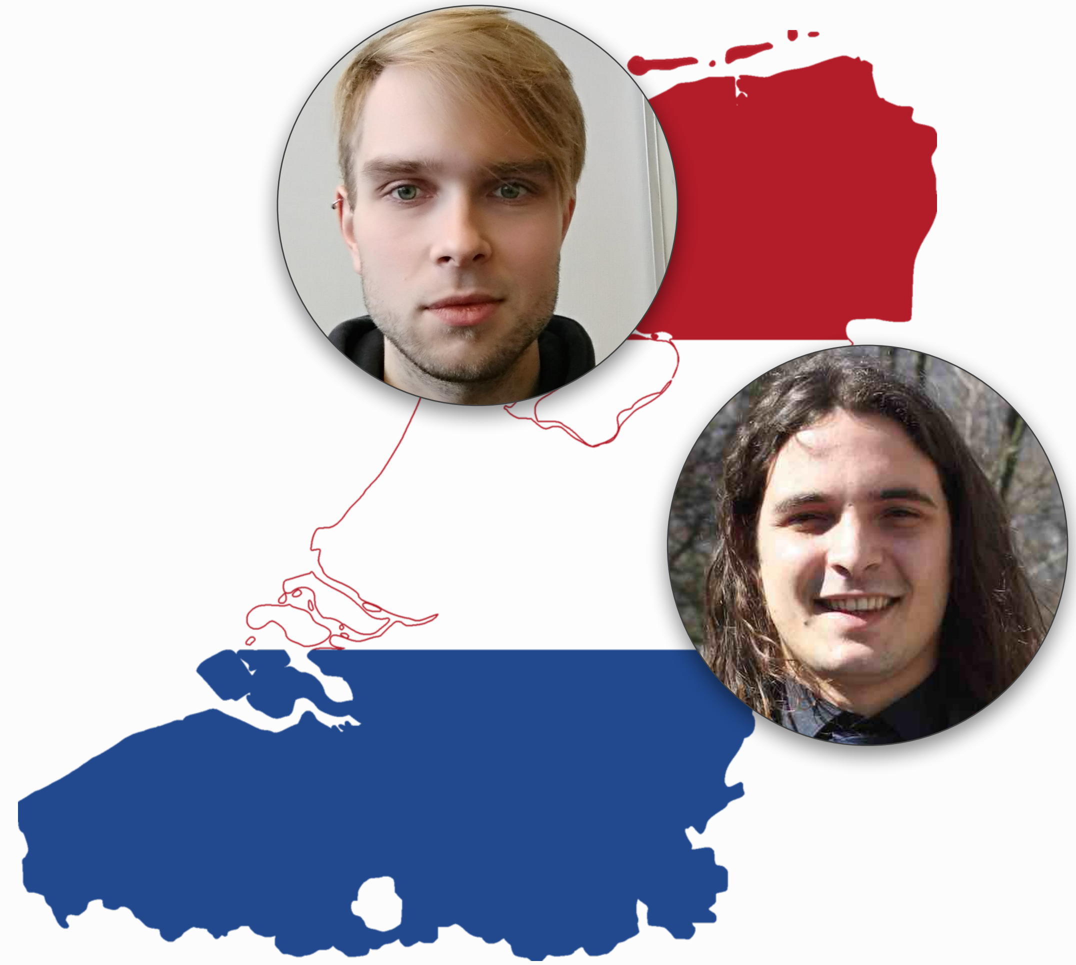
- Let us do a quick round of introductions and let us/me know:
  - your name,
  - what you expected from enrolling in this course,
  - what you would ideally like to do after your M.Sc. studies.

ZEST: Zurich Empirical Software engineering Team

Institut für Informatik
University of Zurich
Switzerland

FONDS NATIONAL SUISSE
SCHWEIZERISCHER NATIONALFONDS
FONDO NAZIONALE SVIZZERO
SWISS NATIONAL SCIENCE FOUNDATION

Computer Science Department
Delft University of Technology
The Netherlands



…and awesome regular collaborators

U. Hannover | CMU | Microsoft | Mozilla | Google

WHAT is important for SOFTWARE QUALITY and WHY?

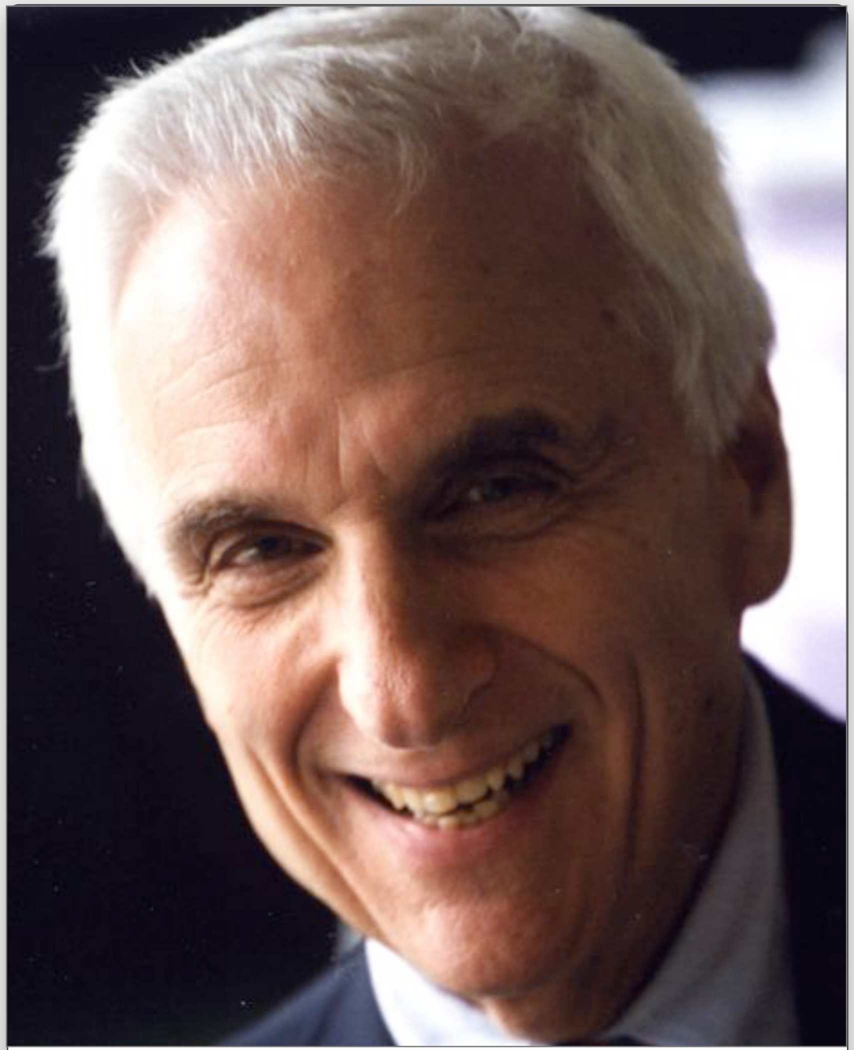
WHAT is important for SOFTWARE QUALITY and WHY?

- When you write software and you care about software quality(*)..
  - What are the aspects that you always try to keep in mind and/or the behaviors that you try to have?
    - Form groups of 2/3 students, discuss the answer to the question above, and come up with a list of top-3 *elements* and their rationale.
      Let us discuss the results in 5 minutes.
  - Do these aspects/behaviors change when you know that you are working in a team?
    - Let us find 5 *elements* that matter and their rationale.

yes, but.. how do you know?

(*) e.g., that the software is maintainable and reasonably defect free

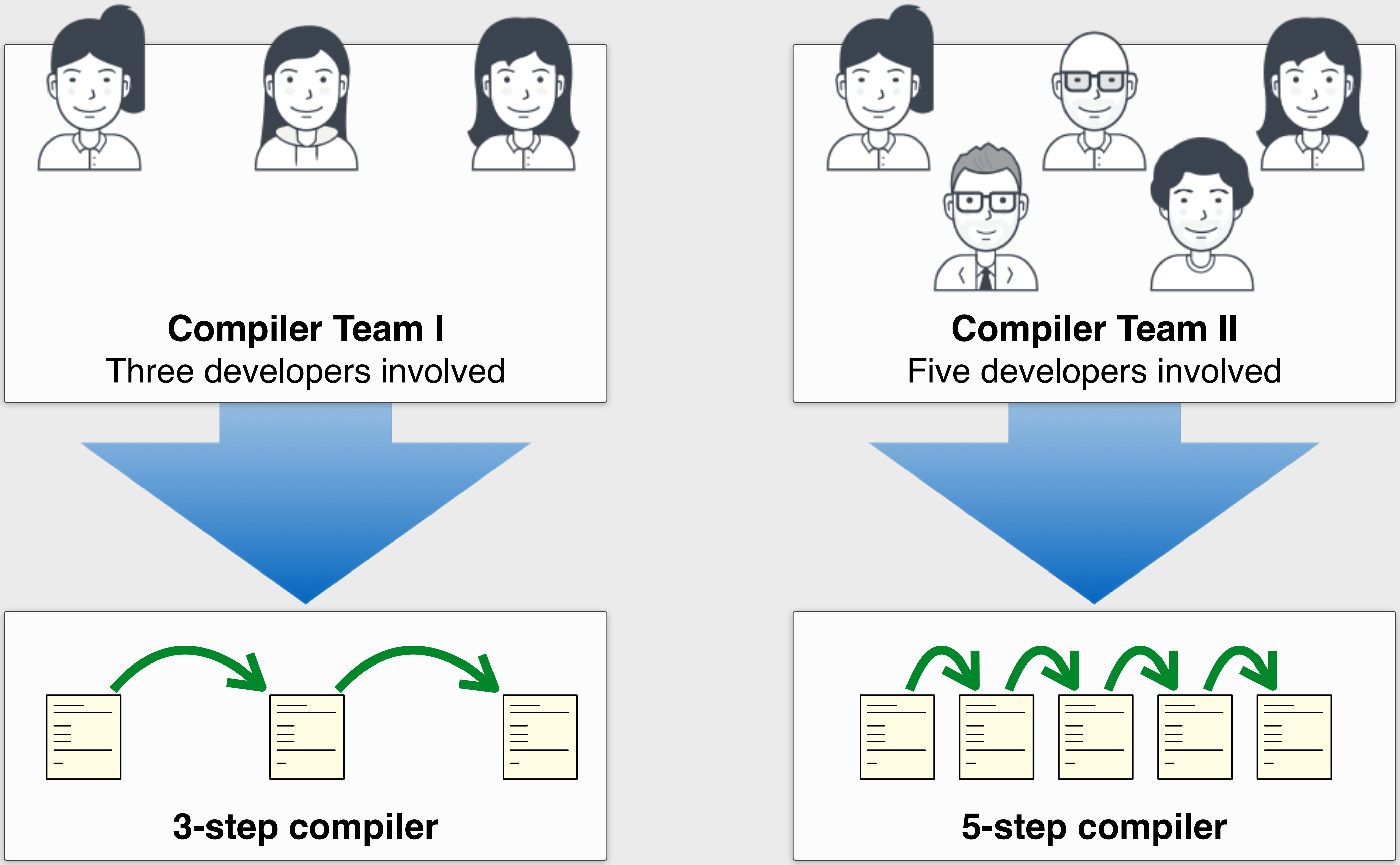
Conway's Law



Melvin Conway

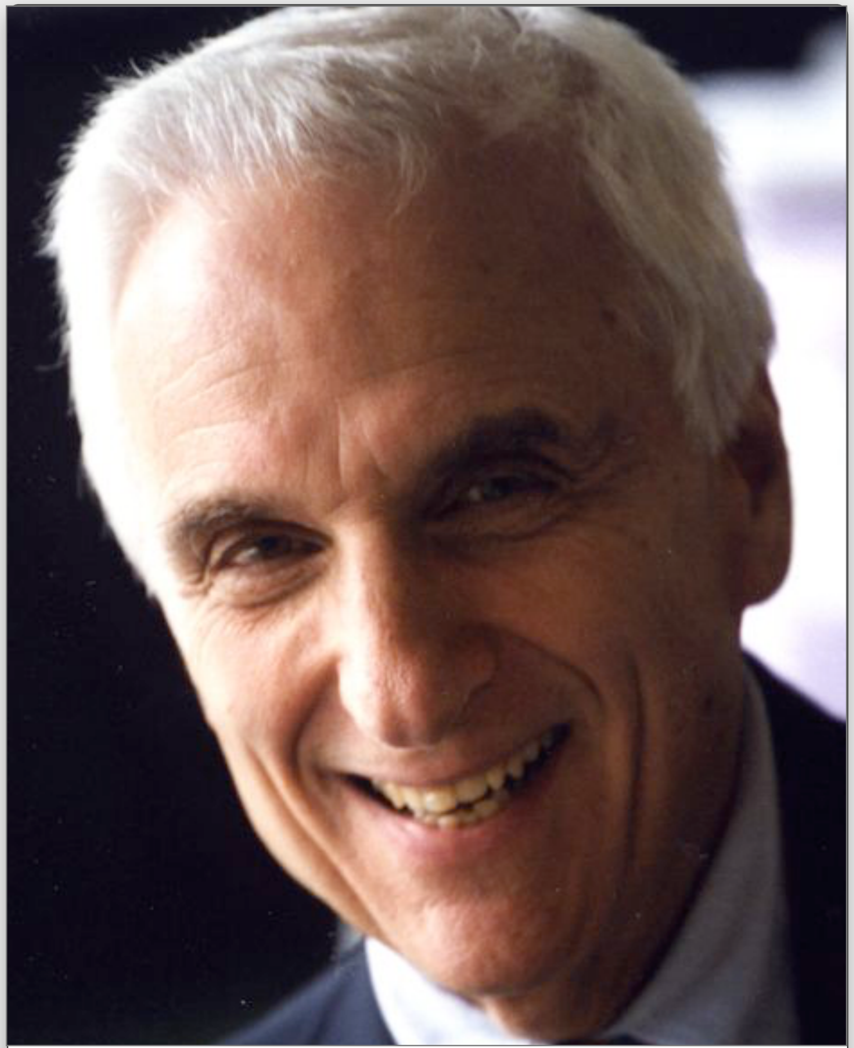
Any organization that design a system will produce a design whose structure is a copy of the organization's communication structure.

Conway's Law — Anecdotal Evidence: Building a compiler



yes, but.. what happens over time?

Conway's Law — A potential corollary



Melvin Conway

Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure.

A software system whose structure closely matches its organization's communication structure works "better."

Conway's Law — A positivist take: Studies to increase our confidence

Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools

Marcelo Cataldo[1] Patrick A. Wagstrom[2] James D. Herbsleb[1] Kathleen M. Carley[1]
[1] Institute for Software Research International
[2] Department of Engineering and Public Policy
Carnegie Mellon University
Pittsburgh, PA 15213
{mcataldo,pwagstro}@andrew.cmu.edu jdh@cs.cmu.edu kathleen.carley@cmu.edu

Coordination & Productivity

Don't Touch My Code! Examining the Effects of Ownership on Software Quality

Christian Bird, Microsoft Research, cbird@microsoft.com
Nachiappan Nagappan, Microsoft Research, nachin@microsoft.com
Brendan Murphy, Microsoft Research, bmurphy@microsoft.com
Harald Gall, University of Zurich, gall@ifi.uzh.ch
Premkumar Devanbu, University of California, Davis, ptdevanbu@ucdavis.edu

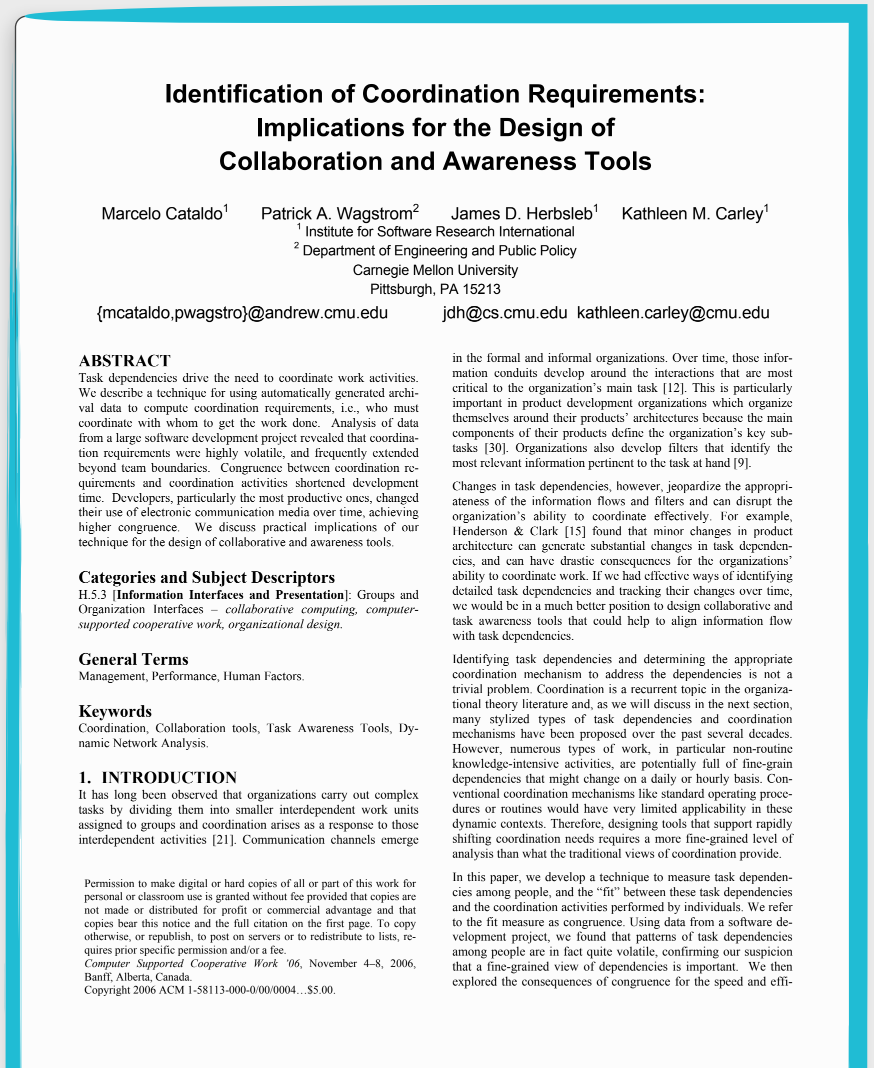
Coordination & Quality

Latent Social Structure in Open Source Projects

Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov and Premkumar Devanbu
Dept. of Computer Science, Kemper Hall,
University of California, Davis, CA, USA,
cabird,dspattison,rmdsouza,vfilkov,ptdevanbu@ucdavis.edu

Coordination & Success

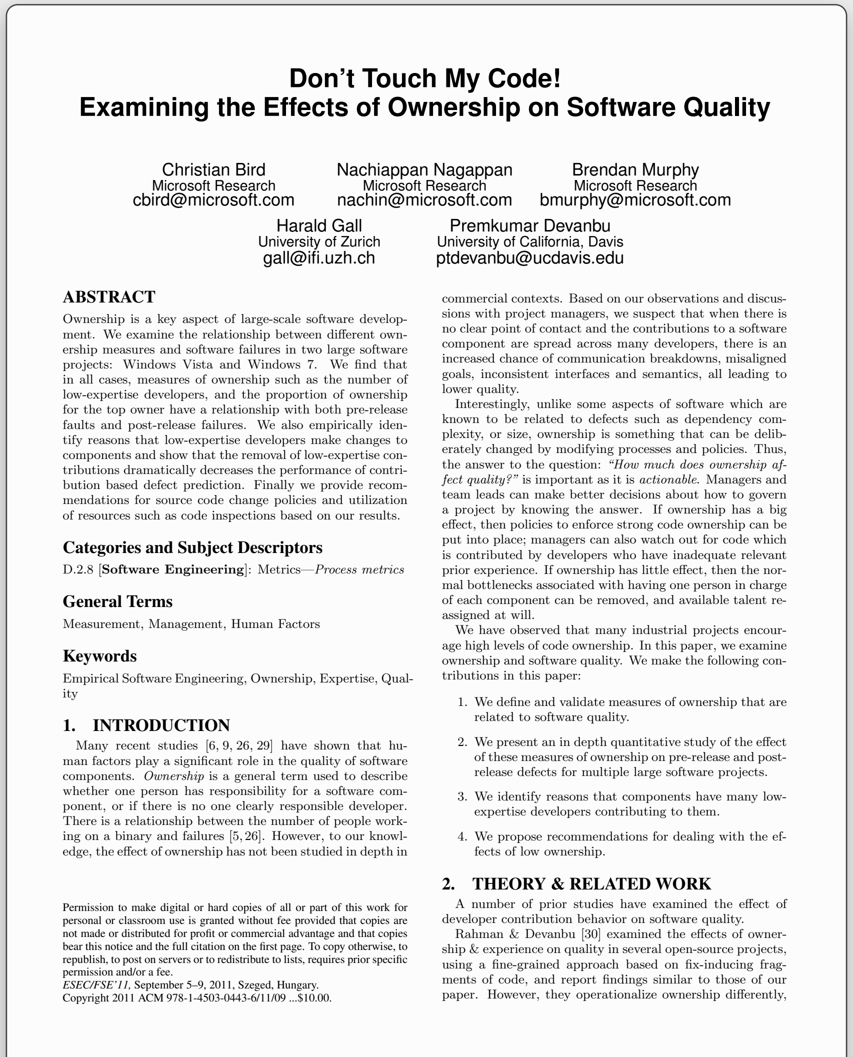
Conway's Law — A positivist take: Studies to increase our confidence



Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools

Marcelo Cataldo[1], Patrick A. Wagstrom[2], James D. Herbsleb[1], Kathleen M. Carley[1]

Coordination & Productivity



Don't Touch My Code! Examining the Effects of Ownership on Software Quality

Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, Premkumar Devanbu

Coordination & Quality

Latent Social Structure in Open Source Projects

Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov and Premkumar Devanbu

Coordination & Success

Don’t Touch My Code! Examining the effects of ownership on software quality

Don’t Touch My Code! Examining the Effects of Ownership on Software Quality

Christian Bird
Microsoft Research
cbird@microsoft.com

Nachiappan Nagappan
Microsoft Research
nachin@microsoft.com

Brendan Murphy
Microsoft Research
bmurphy@microsoft.com

Harald Gall
University of Zurich
gall@ifi.uzh.ch

Premkumar Devanbu
University of California, Davis
ptdevanbu@ucdavis.edu

ABSTRACT
Ownership is a key aspect of large-scale software development. We examine the relationship between different ownership measures and software failures in two large software projects: Windows Vista and Windows 7. We find that in all cases, measures of ownership such as the number of low-expertise developers, and the proportion of ownership for the top owner have a relationship with both pre-release faults and post-release failures. We also empirically identify reasons that low-expertise developers make changes to components and show that the removal of low-expertise contributions dramatically decreases the performance of contribution based defect prediction. Finally we provide recommendations for source code change policies and utilization of resources such as code inspections based on our results.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics—*Process metrics*

General Terms
Measurement, Management, Human Factors

Keywords
Empirical Software Engineering, Ownership, Expertise, Quality

1. INTRODUCTION
Many recent studies [6, 9, 26, 29] have shown that human factors play a significant role in the quality of software components. *Ownership* is a general term used to describe whether one person has responsibility for a software component, or if there is no one clearly responsible developer. There is a relationship between the number of people working on a binary and failures [5, 26]. However, to our knowledge, the effect of ownership has not been studied in depth in commercial contexts. Based on our observations and discussions with project managers, we suspect that when there is no clear point of contact and the contributions to a software component are spread across many developers, there is an increased chance of communication breakdowns, misaligned goals, inconsistent interfaces and semantics, all leading to lower quality.

Interestingly, unlike some aspects of software which are known to be related to defects such as dependency complexity, or size, ownership is something that can be deliberately changed by modifying processes and policies. Thus, the answer to the question: *“How much does ownership affect quality?”* is important as it is *actionable*. Managers and team leads can make better decisions about how to govern a project by knowing the answer. If ownership has a big effect, then policies to enforce strong code ownership can be put into place; managers can also watch out for code which is contributed by developers who have inadequate relevant prior experience. If ownership has little effect, then the normal bottlenecks associated with having one person in charge of each component can be removed, and available talent reassigned at will.

We have observed that many industrial projects encourage high levels of code ownership. In this paper, we examine ownership and software quality. We make the following contributions in this paper:

1. We define and validate measures of ownership that are related to software quality.
2. We present an in depth quantitative study of the effect of these measures of ownership on pre-release and post-release defects for multiple large software projects.
3. We identify reasons that components have many low-expertise developers contributing to them.
4. We propose recommendations for dealing with the effects of low ownership.

2. THEORY & RELATED WORK
A number of prior studies have examined the effect of developer contribution behavior on software quality.

Rahman & Devanbu [30] examined the effects of ownership & experience on quality in several open-source projects, using a fine-grained approach based on fix-inducing fragments of code, and report findings similar to those of our paper. However, they operationalize ownership differently,

ESEC/FSE’11, September 5–9, 2011, Szeged, Hungary.

Coordination & Quality

Don’t Touch My Code!

Don’t Touch My Code! Examining the Effects of Ownership on Software Quality

Christian Bird
Microsoft Research
cbird@microsoft.com

Nachiappan Nagappan
Microsoft Research
nachin@microsoft.com

Brendan Murphy
Microsoft Research
bmurphy@microsoft.com

Harald Gall
University of Zurich
gall@ifi.uzh.ch

Premkumar Devanbu
University of California, Davis
ptdevanbu@ucdavis.edu

ABSTRACT

Ownership is a key aspect of large-scale software development. We examine the relationship between different ownership measures and software failures in two large software projects: Windows Vista and Windows 7. We find that in all cases, measures of ownership such as the number of low-expertise developers, and the proportion of ownership for the top owner have a relationship with both pre-release faults and post-release failures. We also empirically identify reasons that low-expertise developers make changes to components and show that the removal of low-expertise contributions dramatically decreases the performance of contribution based defect prediction. Finally we provide recommendations for source code change policies and utilization of resources such as code inspections based on our results.

commercial contexts. Based on our observations and discussions with project managers, we suspect that when there is no clear point of contact and the contributions to a software component are spread across many developers, there is an increased chance of communication breakdowns, misaligned goals, inconsistent interfaces and semantics, all leading to lower quality.

Interestingly, unlike some aspects of software which are known to be related to defects such as dependency complexity, or size, ownership is something that can be deliberately changed by modifying processes and policies. Thus, the answer to the question: *“How much does ownership affect quality?”* is important as it is *actionable*. Managers and team leads can make better decisions about how to govern a project by knowing the answer. If ownership has a big effect, then policies to enforce strong code ownership can be

Don’t Touch My Code! — Research goal

- How much does ownership affect quality?
  - How to define and validate measures of ownership that are related to software quality?
  - How to operationalize software quality?
  - How to define effect of ownership metrics on software defects in quantitative terms?

Don’t Touch My Code! — Operationalizing ownership

- **Software Component**: A unit of development that has some core functionality
- **Contributor**: Someone who has made commits/software changes to a component
- **Proportion of Ownership**: Ratio of number of commits that the contributor has made relative to the total number of commits for that component
- **Minor Contributor**: Ownership is below 5%
- **Major Contributor**: Ownership is at or above 5%

Don’t Touch My Code! — Considered metrics by software component

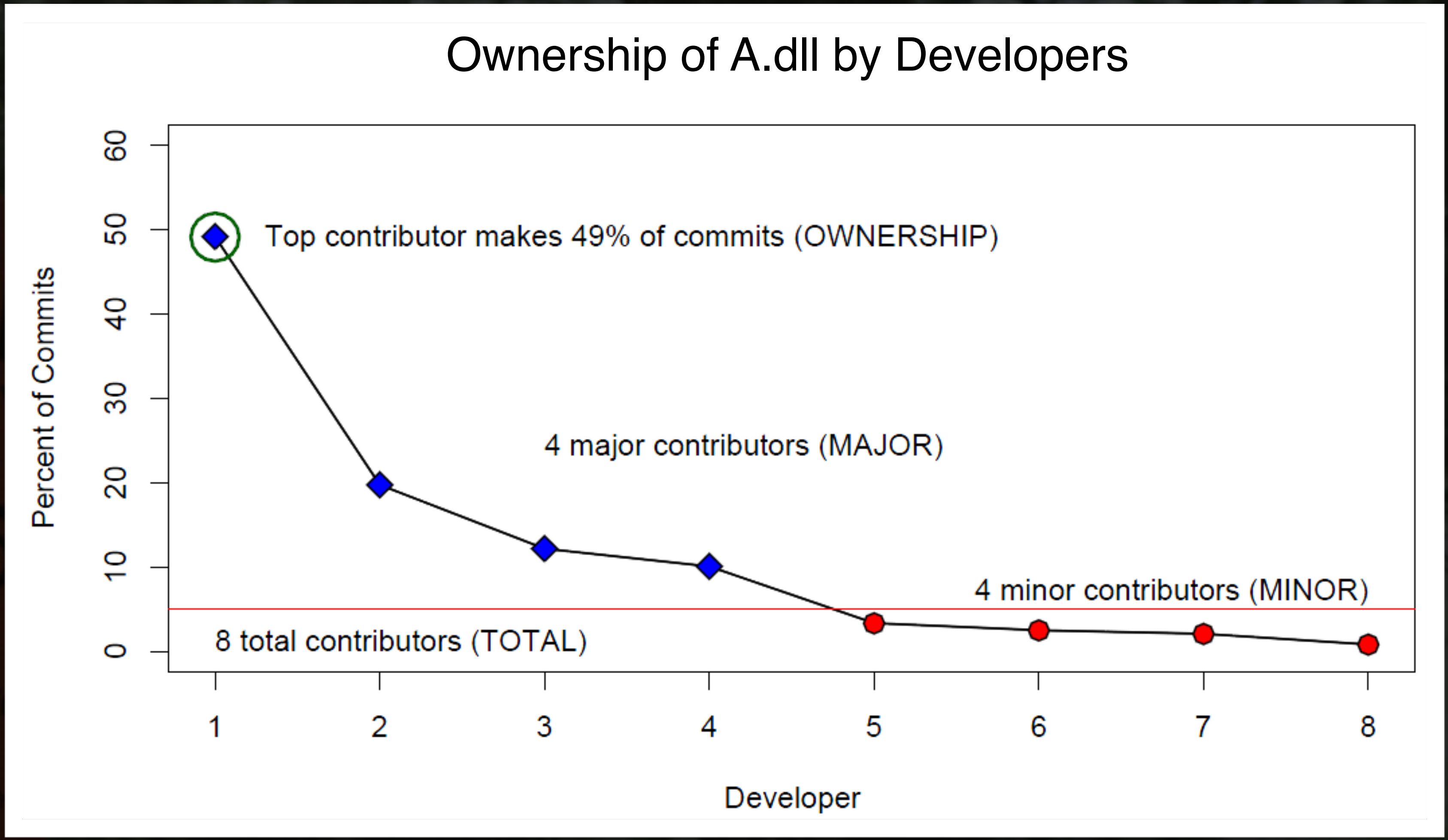
- Minor: Number of minor contributors
- Major: Number of major contributors
- Total: Total number of contributors
- Ownership: Proportion of ownership for the contributor with the highest proportion of ownership.

- Subject systems: Windows Vista and Windows 7

- Subject components: Executable files (.exe), shared libraries(.dll) and drivers (.sys), and their changes recorded in the VCS (changed components, change author, time of change, log message).

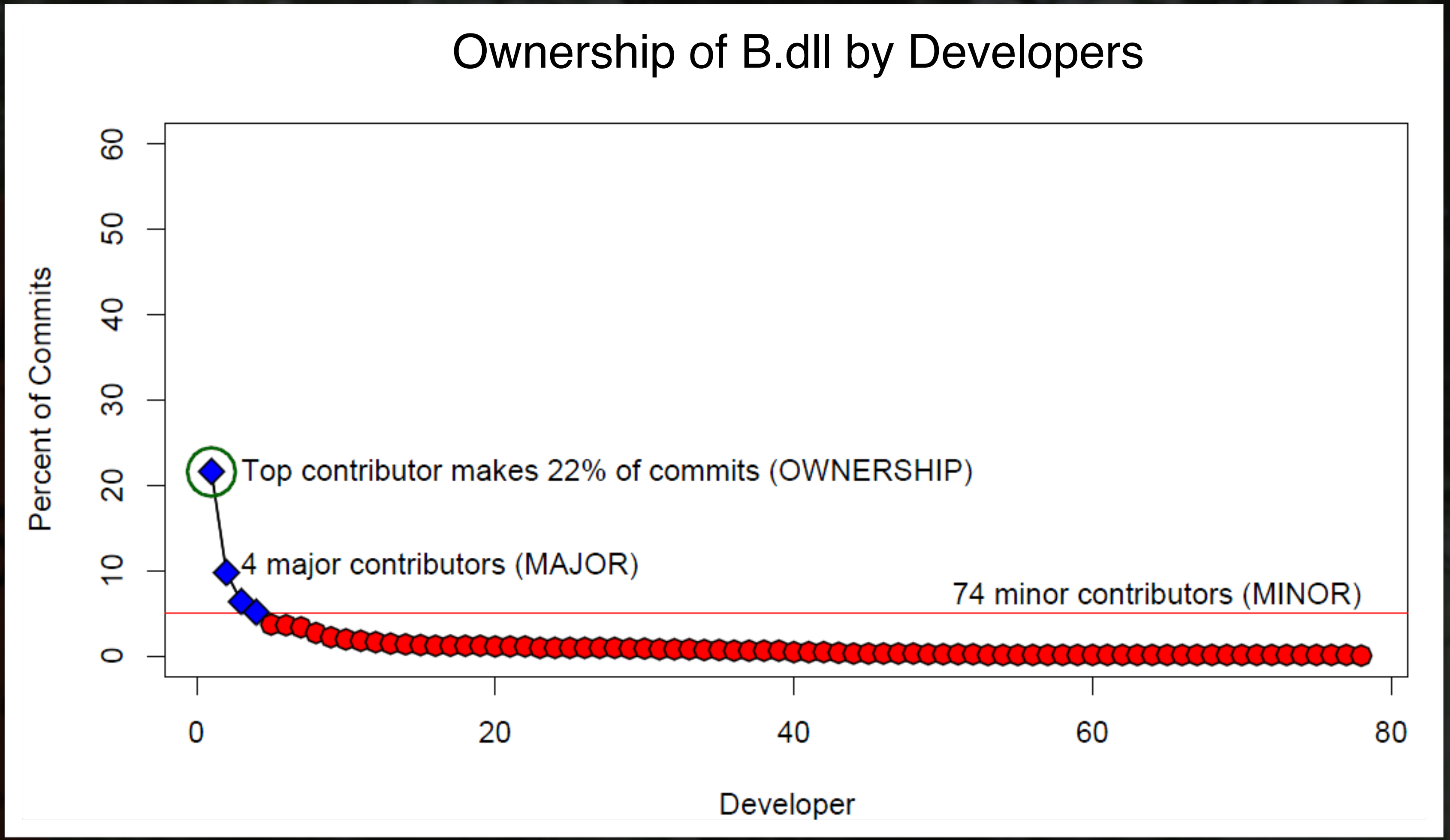
Don’t Touch My Code! — Example metrics for a software component

Metric	Value
Minor	4
Major	4
Total	8
Ownership	49%



Don’t Touch My Code! — Example metrics for a software component

Metric	Value
Minor	74
Major	4
Total	78
Ownership	22%



- Software Quality: Software defects!

Don’t Touch My Code! — Data sources

- Subject systems: Windows Vista and Windows 7
- Subject components: Executable files (.exe), shared libraries(.dll) and drivers (.sys), and their changes recorded in the VCS (changed components, change author, time of change, log message).
- Subject defects: Pre-release defects and post-release failures

Don’t Touch My Code! — Two hypotheses

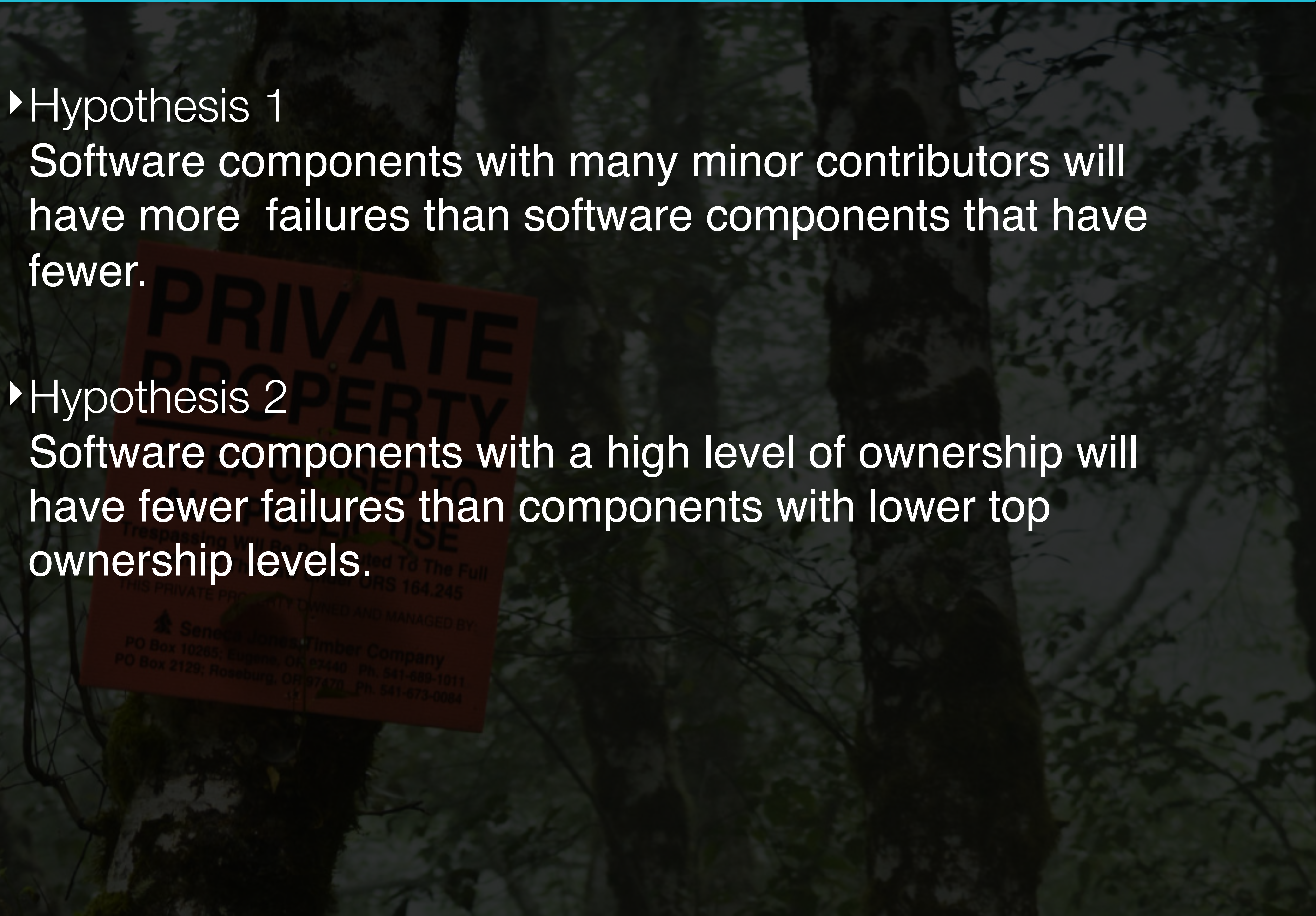
- Hypothesis 1
  Software components with many minor contributors will have ______ failures than software components that have fewer.

- Hypothesis 2
  Software components with a high level of ownership will have ______ failures than components with lower top ownership levels.

more or less?

based on Conway: Fill the blanks!

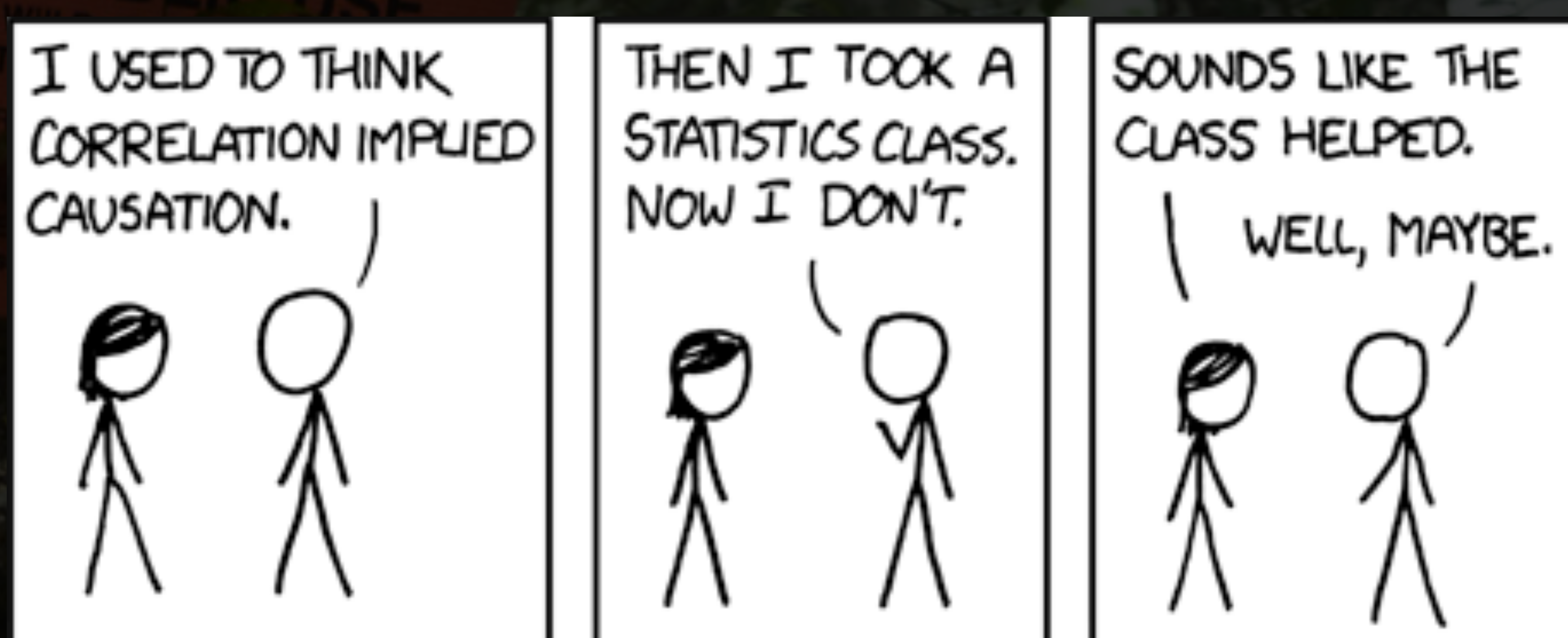
Don't Touch My Code! — Two hypotheses



- Hypothesis 1
  Software components with many minor contributors will have more failures than software components that have fewer.

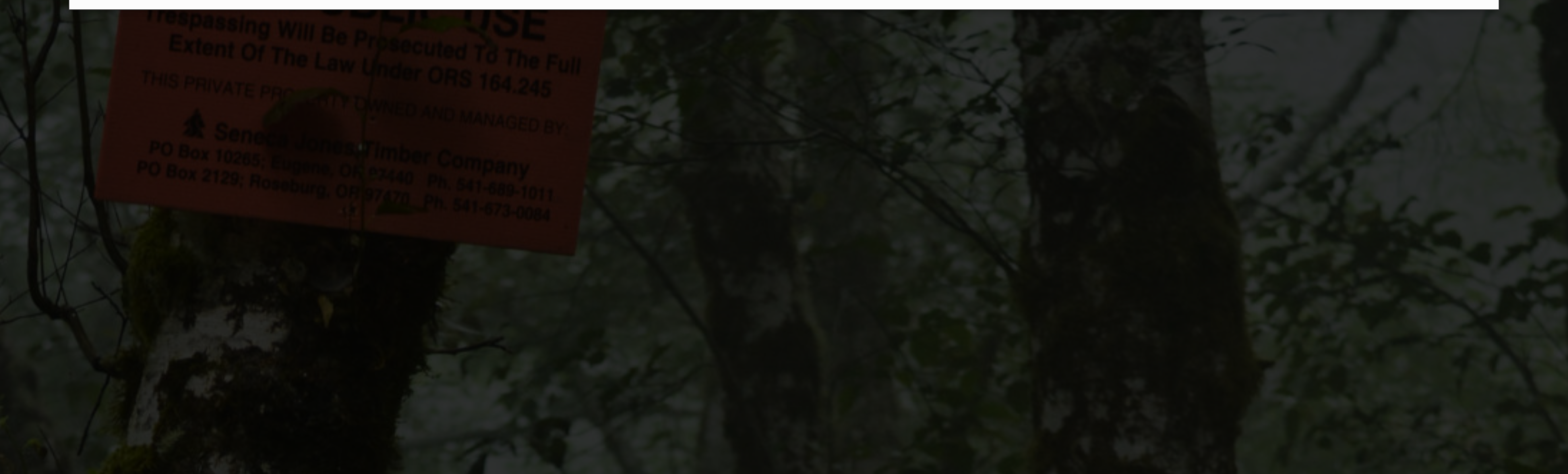
- Hypothesis 2
  Software components with a high level of ownership will have fewer failures than components with lower top ownership levels.

- Correlation analysis:
  - Is there any relationship between ownership and software quality?
  - How strong is the relationship?
- Regression analysis:
  - Is there any effect of ownership variables on failures?
  - Is ownership important when controlling for other factors?



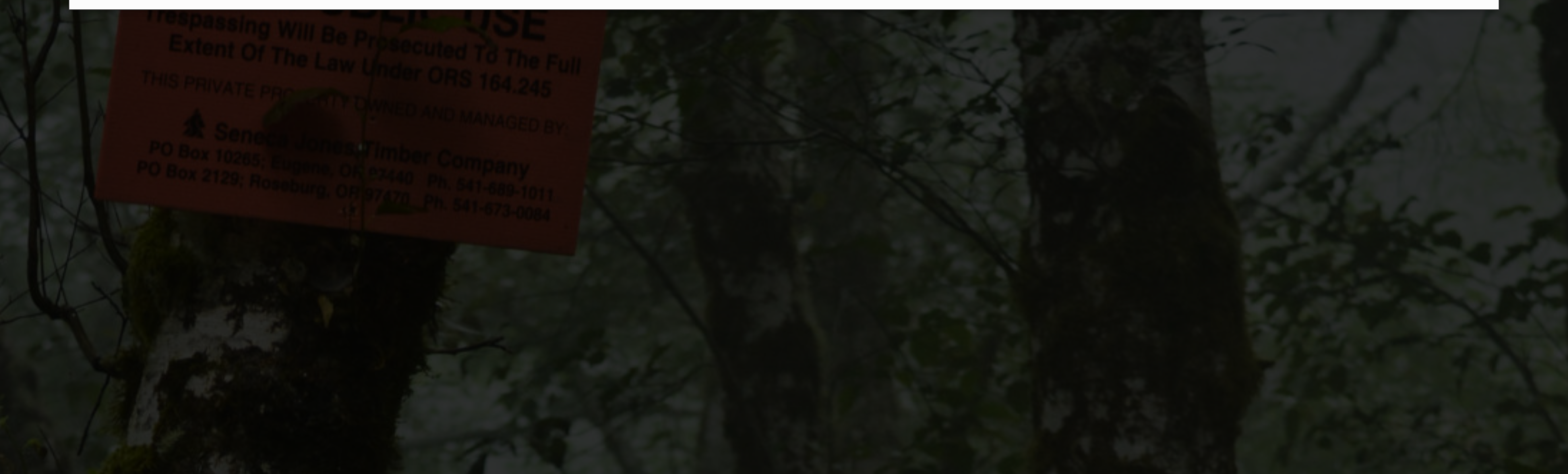
Don’t Touch My Code! — Results: Correlation Analysis

Category	Metric	Windows Vista		Windows 7	
		Pre-release Failures	Post-release Failures	Pre-release Failures	Post-release Failures
Ownership Metrics	TOTAL	0.84	0.70	0.92	0.24
	MINOR	0.86	0.70	0.93	0.25
	MAJOR	0.26	0.29	-0.40	-0.14
	OWNERSHIP	-0.49	-0.49	-0.29	-0.02



Don’t Touch My Code! — Results: Correlation Analysis of Controlling Factors

Category	Metric	Windows Vista		Windows 7	
		Pre-release Failures	Post-release Failures	Pre-release Failures	Post-release Failures
Ownership Metrics	Total	0.84	0.70	0.92	0.24
	Minor	0.86	0.70	0.93	0.25
	Major	0.26	0.29	-0.40	-0.14
	Ownership	-0.49	-0.49	-0.29	-0.02
“Classic” Metrics	Size	0.75	0.69	0.70	0.26
	Churn	0.72	0.69	0.71	0.26
	Complexity	0.70	0.53	0.56	0.37

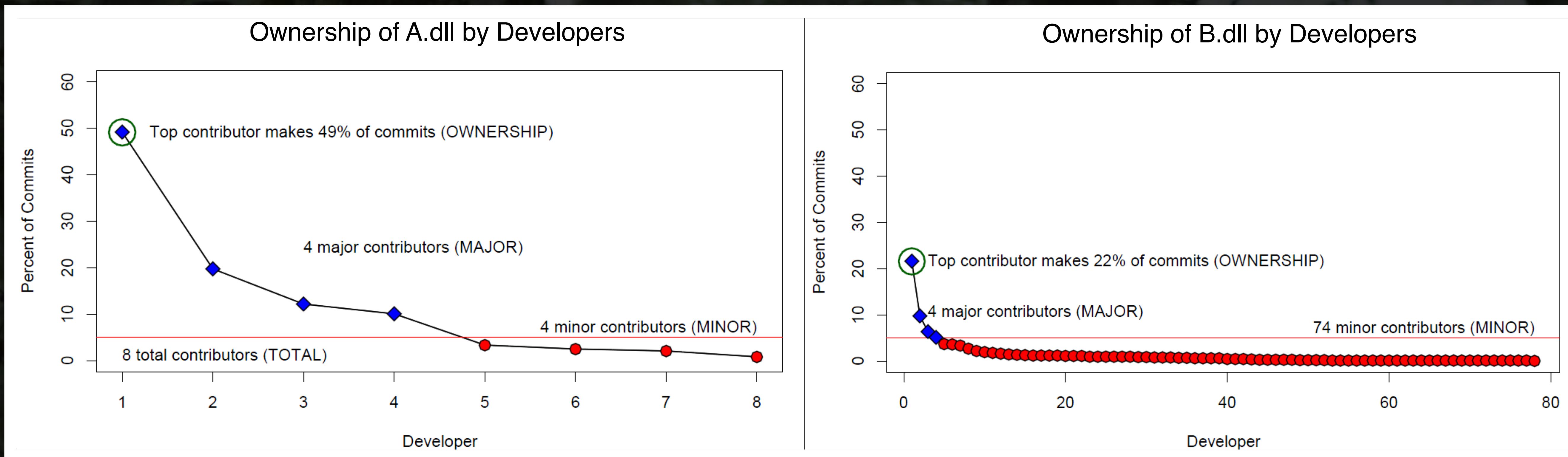


Don't Touch My Code! — Results: Regression Analysis

Category	Metric	Windows Vista Pre-release Failures	Windows Vista Post-release Failures	Windows 7 Pre-release Failures	Windows 7 Post-release Failures
Ownership Metrics	TOTAL	0.84	0.70	0.92	0.24
	MINOR	0.86	0.70	0.93	0.25
	MAJOR	0.26	0.29	-0.40	-0.14
	OWNERSHIP	-0.49	-0.49	-0.29	-0.02
"Classic" Metrics	Size	0.75	0.69	0.70	0.26
	Churn	0.72	0.69	0.71	0.26
	Complexity	0.70	0.53	0.56	0.37

Model	Windows Vista Pre-release Failures	Windows Vista Post-release Failures	Windows 7 Pre-release Failures	Windows 7 Post-release Failures
Base (code metrics)	26%	29%	24%	18%
Base + TOTAL	40%* (+14%)	35%* (+6%)	68%* (+35%)	21%* (+3%)
Base + MINOR	46%* (+20%)	41%* (+12%)	70%* (+46%)	21%* (+3%)
Base + MINOR + MAJOR	48%* (+2%)	43%* (+2%)	71%* (+1%)	22% (+1%)
Base + MINOR + MAJOR + OWNERSHIP	50%* (+2%)	44%* (+1%)	72%* (+1%)	22% (+0%)

Don’t Touch My Code! — Comparing two components



which of these two components is more likely to have defects?

Conway's Law — A positivist take: Studies to increase our confidence

Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools

Marcelo Cataldo[1] Patrick A. Wagstrom[2] James D. Herbsleb[1] Kathleen M. Carley[1]
[1] Institute for Software Research International
[2] Department of Engineering and Public Policy
Carnegie Mellon University
Pittsburgh, PA 15213
{mcataldo,pwagstro}@andrew.cmu.edu jdh@cs.cmu.edu kathleen.carley@cmu.edu

ABSTRACT
Task dependencies drive the need to coordinate work activities. We describe a technique for using automatically generated archival data to compute coordination requirements, i.e., who must coordinate with whom to get the work done. Analysis of data from a large software development project revealed that coordination requirements were highly volatile, and frequently extended beyond team boundaries. Congruence between coordination requirements and coordination activities shortened development time. Developers, particularly the most productive ones, changed their use of electronic communication media over time, achieving higher congruence. We discuss practical implications of our technique for the design of collaborative and awareness tools.

Categories and Subject Descriptors
H.5.3 [**Information Interfaces and Presentation**]: Groups and Organization Interfaces – *collaborative computing, computer-supported cooperative work, organizational design.*

General Terms
Management, Performance, Human Factors.

Keywords
Coordination, Collaboration tools, Task Awareness Tools, Dynamic Network Analysis.

1. INTRODUCTION
It has long been observed that organizations carry out complex tasks by dividing them into smaller interdependent work units assigned to groups and coordination arises as a response to those interdependent activities [21]. Communication channels emerge in the formal and informal organizations. Over time, those information conduits develop around the interactions that are most critical to the organization's main task [12]. This is particularly important in product development organizations which organize themselves around their products' architectures because the main components of their products define the organization's key subtasks [30]. Organizations also develop filters that identify the most relevant information pertinent to the task at hand [9].

Changes in task dependencies, however, jeopardize the appropriateness of the information flows and filters and can disrupt the organization's ability to coordinate effectively. For example, Henderson & Clark [15] found that minor changes in product architecture can generate substantial changes in task dependencies, and can have drastic consequences for the organizations' ability to coordinate work. If we had effective ways of identifying detailed task dependencies and tracking their changes over time, we would be in a much better position to design collaborative and task awareness tools that could help to align information flow with task dependencies.

Identifying task dependencies and determining the appropriate coordination mechanism to address the dependencies is not a trivial problem. Coordination is a recurrent topic in the organizational theory literature and, as we will discuss in the next section, many stylized types of task dependencies and coordination mechanisms have been proposed over the past several decades. However, numerous types of work, in particular non-routine knowledge-intensive activities, are potentially full of fine-grain dependencies that might change on a daily or hourly basis. Conventional coordination mechanisms like standard operating procedures or routines would have very limited applicability in these dynamic contexts. Therefore, designing tools that support rapidly shifting coordination needs requires a more fine-grained level of analysis than what the traditional views of coordination provide.

In this paper, we develop a technique to measure task dependencies among people, and the "fit" between these task dependencies and the coordination activities performed by individuals. We refer to the fit measure as congruence. Using data from a software development project, we found that patterns of task dependencies among people are in fact quite volatile, confirming our suspicion that a fine-grained view of dependencies is important. We then explored the consequences of congruence for the speed and effi-

Coordination & Productivity

Don't Touch My Code! Examining the Effects of Ownership on Software Quality

Christian Bird, Microsoft Research, cbird@microsoft.com
Nachiappan Nagappan, Microsoft Research, nachin@microsoft.com
Brendan Murphy, Microsoft Research, bmurphy@microsoft.com
Harald Gall, University of Zurich, gall@ifi.uzh.ch
Premkumar Devanbu, University of California, Davis, ptdevanbu@ucdavis.edu

ABSTRACT
Ownership is a key aspect of large-scale software development. We examine the relationship between different ownership measures and software failures in two large software projects: Windows Vista and Windows 7. We find that in all cases, measures of ownership such as the number of low-expertise developers, and the proportion of ownership for the top owner have a relationship with both pre-release faults and post-release failures. We also empirically identify reasons that low-expertise developers make changes to components and show that the removal of low-expertise contributions dramatically decreases the performance of contribution based defect prediction. Finally we provide recommendations for source code change policies and utilization of resources such as code inspections based on our results.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics—*Process metrics*

General Terms
Measurement, Management, Human Factors

Keywords
Empirical Software Engineering, Ownership, Expertise, Quality

1. INTRODUCTION
Many recent studies [6, 9, 26, 29] have shown that human factors play a significant role in the quality of software components. *Ownership* is a general term used to describe whether one person has responsibility for a software component, or if there is no one clearly responsible developer. There is a relationship between the number of people working on a binary and failures [5, 26]. However, to our knowledge, the effect of ownership has not been studied in depth in commercial contexts. Based on our observations and discussions with project managers, we suspect that when there is no clear point of contact and the contributions to a software component are spread across many developers, there is an increased chance of communication breakdowns, misaligned goals, inconsistent interfaces and semantics, all leading to lower quality.

Interestingly, unlike some aspects of software which are known to be related to defects such as dependency complexity, or size, ownership is something that can be deliberately changed by modifying processes and policies. Thus, the answer to the question: *"How much does ownership affect quality?"* is important as it is *actionable.* Managers and team leads can make better decisions about how to govern a project by knowing the answer. If ownership has a big effect, then policies to enforce strong code ownership can be put into place; managers can also watch out for code which is contributed by developers who have inadequate relevant prior experience. If ownership has little effect, then the normal bottlenecks associated with having one person in charge of each component can be removed, and available talent reassigned at will.

We have observed that many industrial projects encourage high levels of code ownership. In this paper, we examine ownership and software quality. We make the following contributions in this paper:

1. We define and validate measures of ownership that are related to software quality.
2. We present an in depth quantitative study of the effect of these measures of ownership on pre-release and post-release defects for multiple large software projects.
3. We identify reasons that components have many low-expertise developers contributing to them.
4. We propose recommendations for dealing with the effects of low ownership.

2. THEORY & RELATED WORK
A number of prior studies have examined the effect of developer contribution behavior on software quality.

Rahman & Devanbu [30] examined the effects of ownership & experience on quality in several open-source projects, using a fine-grained approach based on fix-inducing fragments of code, and report findings similar to those of our paper. However, they operationalize ownership differently,

Coordination & Quality

Latent Social Structure in Open Source Projects

Christian Bird, David Pattison, Raissa D'Souza,
Vladimir Filkov and Premkumar Devanbu
Dept. of Computer Science, Kemper Hall,
University of California, Davis, CA, USA,
cabird,dspattison,rmdsouza,vfilkov,ptdevanbu@ucdavis.edu

ABSTRACT
Commercial software project managers design project organizational structure carefully, mindful of available skills, division of labour, geographical boundaries, etc. These organizational "cathedrals" are to be contrasted with the "bazaar-like" nature of Open Source Software (OSS) Projects, which have no pre-designed organizational structure. Any structure that exists is dynamic, self-organizing, latent, and usually not explicitly stated. Still, in large, complex, successful, OSS projects, we do expect that subcommunities will form spontaneously within the developer teams. Studying these subcommunities, and their behavior can shed light on how successful OSS projects self-organize. This phenomenon could well hold important lessons for how commercial software teams might be organized. Building on known well-established techniques for detecting *community structure* in complex networks, we extract and study latent subcommunities from the email social network of several projects: Apache HTTPD, Python, PostgresSQL, Perl, and Apache ANT. We then validate them with software development activity history. Our results show that subcommunities do indeed spontaneously arise within these projects as the projects evolve. These subcommunities manifest most strongly in technical discussions, and are significantly connected with collaboration behaviour.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management—*programming teams*; D.2.8 [**Software Engineering**]: Metrics—*process metrics*

General Terms
Human Factors, Measurement, Management

Keywords
Open Source Software, social networks, collaboration

This work was supported by a grant from the National Science Foundation Grant no. NSF-SoD-0613949 and software donations from SciTools and GrammaTech Corporations.

1. INTRODUCTION
Brooks, in his seminal work *The Mythical Man-Month* [12], noted the scaling issues that arise in large software teams: the number of potential interactions grows quadratically with team size, thus quadrupling when the team size is doubled. Clearly, without organization of some kind, both within the software and the community that develops it, there is a limit to how much projects can be scaled.

In traditional, commercial software projects, the response to the Brooksian critique of large teams is to divide and conquer, *by fiat*. The system is deliberately divided into smaller components, and the developer pool grouped into manageable teams which are then assigned to those components. With well-defined interfaces, the teams' efforts are confined to smaller groups, and the coordination needs are moderated. Software design principles such as separation of concerns [53] play a part in this, as does "Conway's Law" [16], which connects artifact structure with organizational structure.

By contrast, Open Source Software (OSS) projects are not formally organized, and have no pre-assigned command and control structure. No one is forced to work on a particular portion of the project. Team members contribute as they wish in any number of ways: by submitting bug reports, lending technical knowledge, writing documentation, improving the source code in various ares of the code base, etc. It has been observed by Sosa *et al.* [56] that the fixed organizational structure found in commercial settings may lead to misalignment with evolving complex products. Henderson and Clark point out that it may may actually hinder innovation [32]. Thus the lack of a rigid organizational structure may in fact be a boon to OSS projects. However, the absence of any structure at all may be just as harmful. Henderson and Clark [32] found that "architectural knowledge tends to become embedded in the structure and information-processing procedures of established organizations". Modularizing artifacts and mapping artifact tasks onto organizational units is a well known solution to the problem of complex product development in organizational management literature [56]. The question then arises, is the social structure of OSS projects free of such constraints and actually unorganized and free-for-all? Do they stand in contrast to the structured, hierarchical style of traditional commercial software efforts? Or, do OSS projects have some latent[1] structure of their own? Are there dynamic, self-organizing subgroups that spontaneously form and evolve?

[1]By latent, we mean not explicitly stated, but observable.

Coordination & Success

Conway's Law — A positivist take: Studies to increase our confidence

Latent Social Structure in Open Source Projects

Christian Bird, David Pattison, Raissa D'Souza,
Vladimir Filkov and Premkumar Devanbu
Dept. of Computer Science, Kemper Hall,
University of California, Davis, CA, USA,
cabird,dspattison,rmdsouza,vfilkov,ptdevanbu@ucdavis.edu

ABSTRACT

Commercial software project managers design project organizational structure carefully, mindful of available skills, division of labour, geographical boundaries, etc. These organizational "cathedrals" are to be contrasted with the "bazaar-like" nature of Open Source Software (OSS) Projects, which have no pre-designed organizational structure. Any structure that exists is dynamic, self-organizing, latent, and usually not explicitly stated. Still, in large, complex, successful, OSS projects, we do expect that subcommunities will form spontaneously within the developer teams. Studying these subcommunities, and their behavior can shed light on how successful OSS projects self-organize. This phenomenon could well hold important lessons for how commercial software teams might be organized. Building on known well-established techniques for detecting *community structure* in complex networks, we extract and study latent subcommunities from the email social network of several projects: Apache HTTPD, Python, PostgresSQL, Perl, and Apache ANT. We then validate them with software development activity history. Our results show that subcommunities do indeed spontaneously arise within these projects as the projects evolve. These subcommunities manifest most strongly in technical discussions, and are significantly connected with collaboration behaviour.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*programming teams*; D.2.8 [**Software Engineering**]: Metrics—*process metrics*

General Terms

Human Factors, Measurement, Management

Keywords

Open Source Software, social networks, collaboration

This work was supported by a grant from the National Science Foundation Grant no. NSF-SoD-0613949 and software donations from SciTools and GrammaTech Corporations.

SIGSOFT 2008/FSE 16, November 9–15, Atlanta, Georgia, USA

1. INTRODUCTION

Brooks, in his seminal work *The Mythical Man-Month* [12], noted the scaling issues that arise in large software teams: the number of potential interactions grows quadratically with team size, thus quadrupling when the team size is doubled. Clearly, without organization of some kind, both within the software and the community that develops it, there is a limit to how much projects can be scaled.

In traditional, commercial software projects, the response to the Brooksian critique of large teams is to divide and conquer, *by fiat*. The system is deliberately divided into smaller components, and the developer pool grouped into manageable teams which are then assigned to those components. With well-defined interfaces, the teams' efforts are confined to smaller groups, and the coordination needs are moderated. Software design principles such as separation of concerns [53] play a part in this, as does "Conway's Law" [16], which connects artifact structure with organizational structure.

By contrast, Open Source Software (OSS) projects are not formally organized, and have no pre-assigned command and control structure. No one is forced to work on a particular portion of the project. Team members contribute as they wish in any number of ways: by submitting bug reports, lending technical knowledge, writing documentation, improving the source code in various ares of the code base, etc. It has been observed by Sosa *et al.* [56] that the fixed organizational structure found in commercial settings may lead to misalignment with evolving complex products. Henderson and Clark point out that it may may actually hinder innovation [32]. Thus the lack of a rigid organizational structure may in fact be a boon to OSS projects. However, the absence of any structure at all may be just as harmful. Henderson and Clark [32] found that "architectural knowledge tends to become embedded in the structure and information-processing procedures of established organizations". Modularizing artifacts and mapping artifact tasks onto organizational units is a well known solution to the problem of complex product development in organizational management literature [56]. The question then arises, is the social structure of OSS projects free of such constraints and actually unorganized and free-for-all? Do they stand in contrast to the structured, hierarchical style of traditional commercial software efforts? Or, do OSS projects have some latent[1] structure of their own? Are there dynamic, self-organizing subgroups that spontaneously form and evolve?

[1] By latent, we mean not explicitly stated, but observable.

Coordination & Success

Latent social structure in OSS projects

Latent social structure in OSS projects

Latent Social Structure in Open Source Projects

Christian Bird, David Pattison, Raissa D'Souza,
Vladimir Filkov and Premkumar Devanbu
Dept. of Computer Science, Kemper Hall,
University of California, Davis, CA, USA,
cabird,dspattison,rmdsouza,vfilkov,ptdevanbu@ucdavis.edu

ABSTRACT

Commercial software project managers design project organizational structure carefully, mindful of available skills, division of labour, geographical boundaries, etc. These organizational "cathedrals" are to be contrasted with the "bazaar-like" nature of Open Source Software (OSS) Projects, which have no pre-designed organizational structure. Any structure that exists is dynamic, self-organizing, latent, and usually not explicitly stated. Still, in large, complex, successful, OSS projects, we do expect that subcommunities will form spontaneously within the developer teams. Studying these subcommunities, and their behavior can shed light on how successful OSS projects self-organize. This phenomenon could well hold important lessons for how commercial software teams might be organized. Building on known well-established techniques for detecting *community structure* in complex networks, we extract and study latent subcommunities from the email social network of several

1. INTRODUCTION

Brooks, in his seminal work *The Mythical Man-Month* [12], noted the scaling issues that arise in large software teams: the number of potential interactions grows quadratically with team size, thus quadrupling when the team size is doubled. Clearly, without organization of some kind, both within the software and the community that develops it, there is a limit to how much projects can be scaled.

In traditional, commercial software projects, the response to the Brooksian critique of large teams is to divide and conquer, *by fiat*. The system is deliberately divided into smaller components, and the developer pool grouped into manageable teams which are then assigned to those components. With well-defined interfaces, the teams' efforts are confined to smaller groups, and the coordination needs are moderated. Software design principles such as separation of concerns [53] play a part in this, as does "Conway's Law" [16], which connects artifact structure with organizational struc-

Latent social structure in OSS projects — Research goal

- Is there a community structure in OSS projects? How clearly can you define subcommunities within the network?
- Given that the discussions are either project or process related, is the subcommunity structure influenced by the group's discussion goals?
- Do members of a subcommunity work on the same areas of code?
- Do members of a subcommunity have a common focus?

Latent social structure in OSS projects — Operationalizing structures

- Social Structure: Discussion in the development mailing list.
  Nodes: developers; Edges: exchanged messages.

- Technical Structure: Changes to files in the code base of the project.

Latent social structure in OSS projects — Data sources

- Subject systems:
  - Well-organized, hierarchical structure (Apache, Ant)
  - Informal, community driven structure (PostgreSQL)
  - Monarchist, with project leader (Python, Perl)

- Subject artifacts: Source code files (changed files, change author, time of change, log message) and email messages.

Latent social structure in OSS projects — Three hypotheses

- Hypothesis 1

  Subcommunities of participants will form in the email social networks of large OSS projects and the levels of modularity will be statistically significant.

- Hypothesis 2

  Social Networks constructed from product-related discussions will be ____ modular than those relating to non-product related discussions or all discussions.

- Hypothesis 3

  Pairs of developers within the same subcommunity will have ____ files in common than pairs of developers from different subcommunities.

more or less?

based on Conway: Fill the blanks!

Latent social structure in OSS projects — Three hypotheses

- Hypothesis 1

  Subcommunities of participants will form in the email social networks of large OSS projects and the levels of modularity will be statistically significant.

- Hypothesis 2

  Social Networks constructed from product-related discussions will be more modular than those relating to non-product related discussions or all discussions.

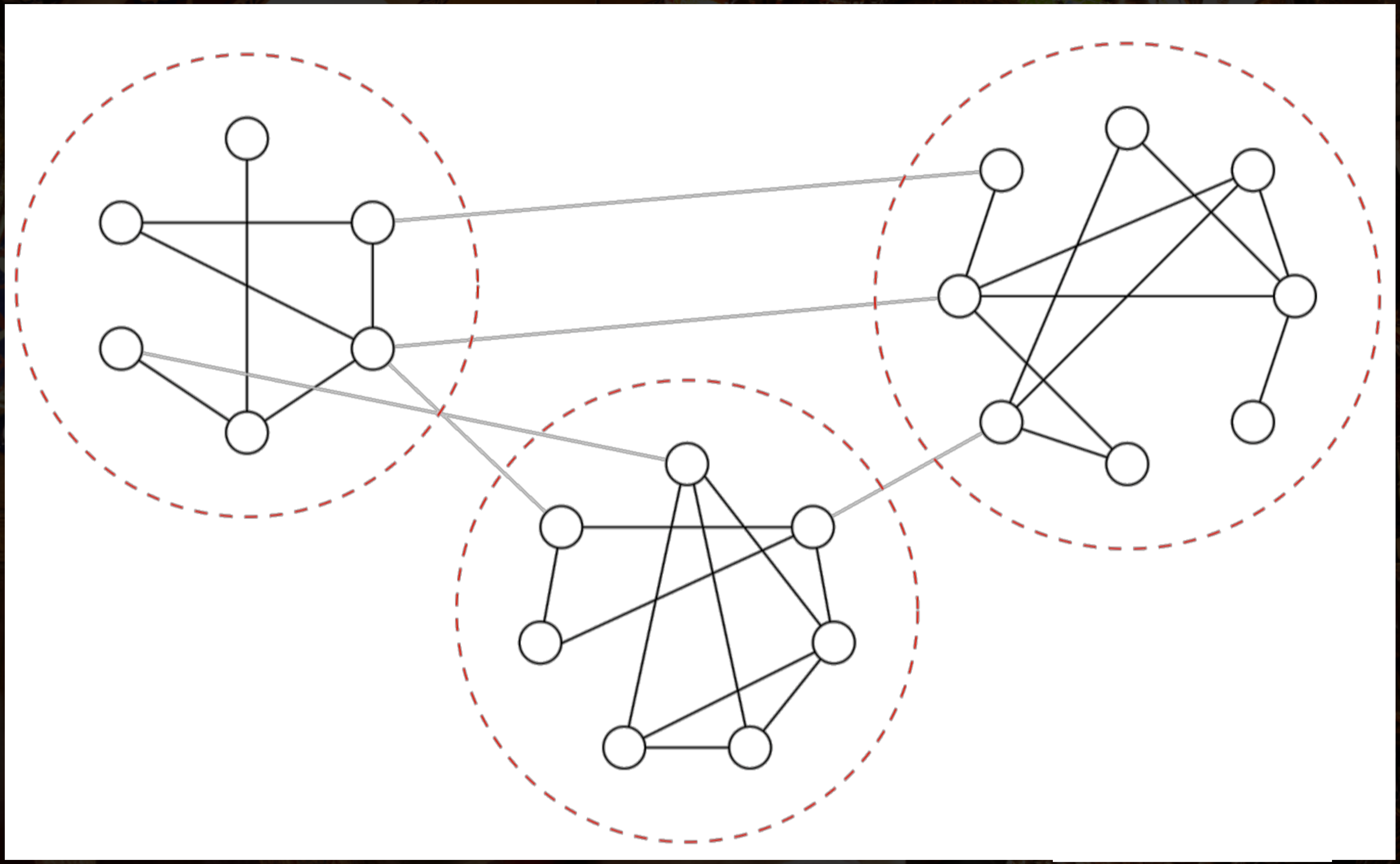
- Hypothesis 3

  Pairs of developers within the same subcommunity will have more files in common than pairs of developers from different subcommunities.

Latent social structure in OSS projects — Three hypotheses

- Hypothesis 1

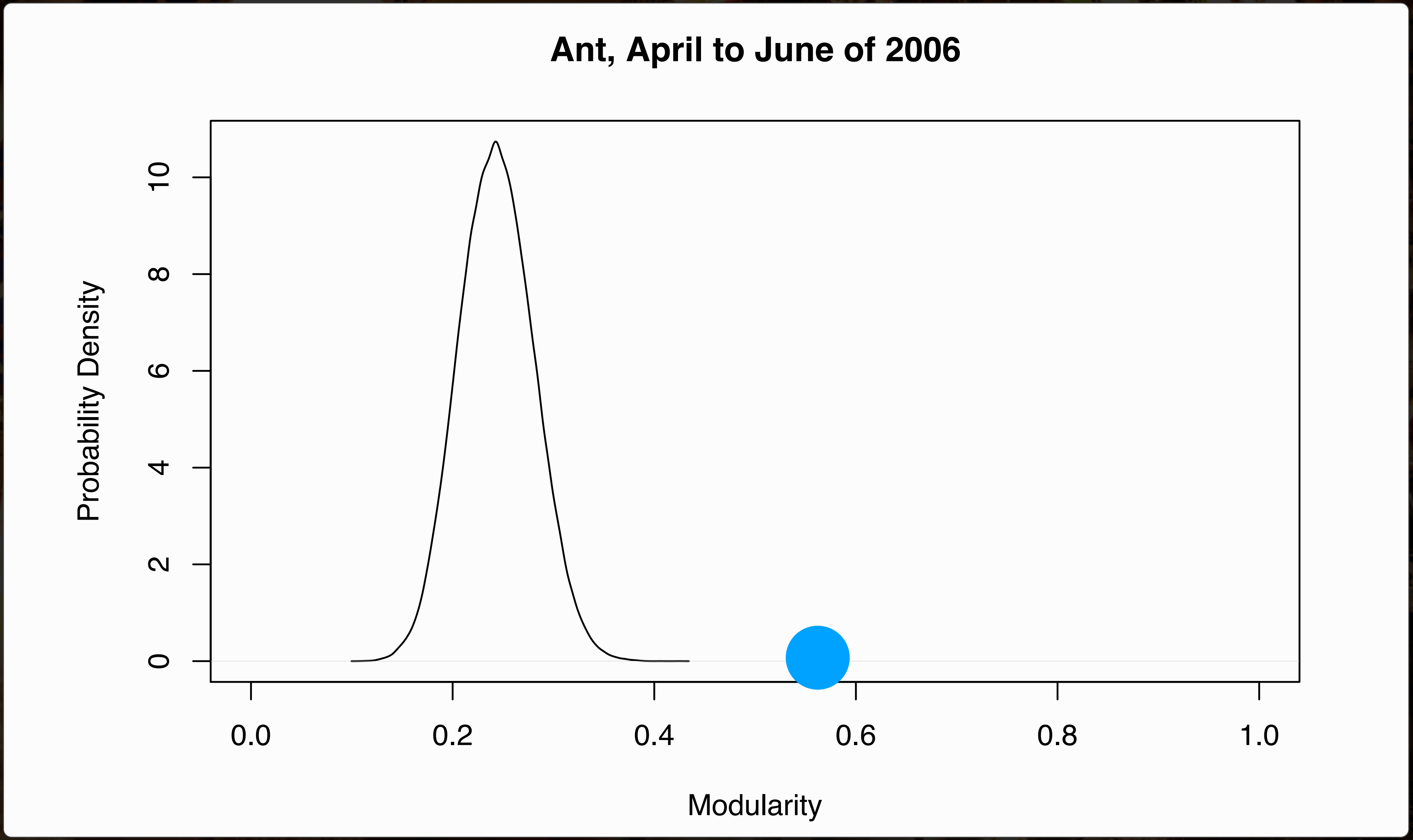
  Subcommunities of participants will form in the email social networks of large OSS projects and the levels of modularity will be statistically significant.



Latent social structure in OSS projects — Three hypotheses

- Hypothesis 1

  Subcommunities of participants will form in the email social networks of large OSS projects and the levels of modularity will be statistically significant.



Latent social structure in OSS projects — Three hypotheses

- Hypothesis 1

  Subcommunities of participants will form in the email social networks of large OSS projects and the levels of modularity will be statistically significant. ✔

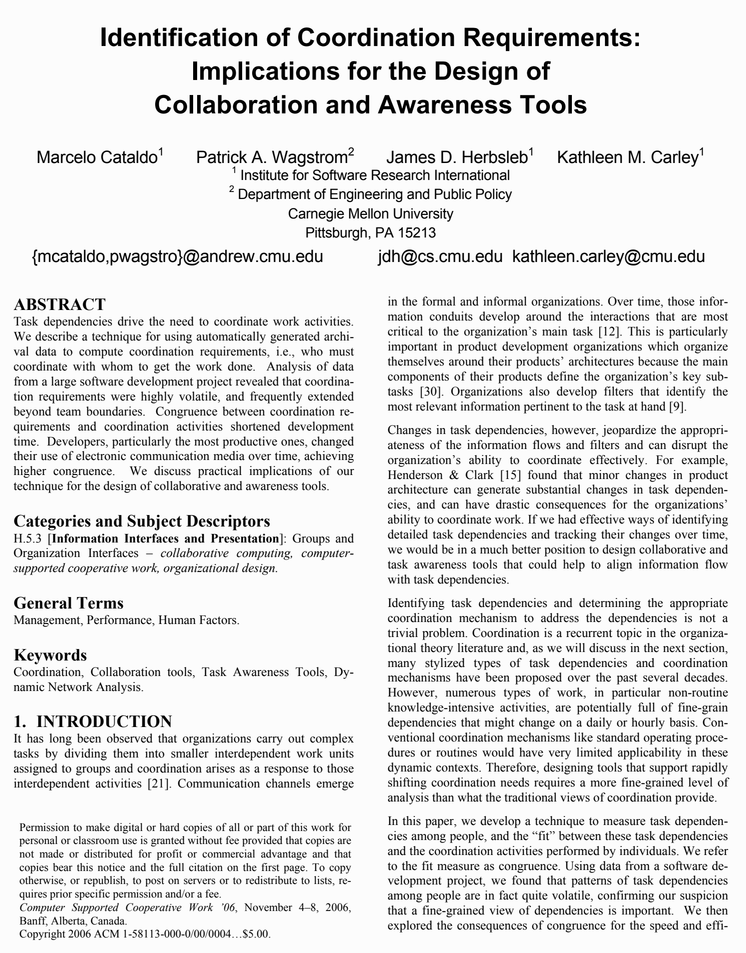
- Hypothesis 2

  Social Networks constructed from product-related discussions will be more modular than those relating to non-product related discussions or all discussions. ✔

Latent social structure in OSS projects — Three hypotheses

- Hypothesis 1
  Subcommunities of participants will form in the email social networks of large OSS projects and the levels of modularity will be statistically significant. ✔
- Hypothesis 2
  Social Networks constructed from product-related discussions will be more modular than those relating to non-product related discussions or all discussions. ✔
- Hypothesis 3
  Pairs of developers within the same subcommunity will have more files in common than pairs of developers from different subcommunities. ✔

Conway's Law — A positivist take: Studies to increase our confidence



Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools

Marcelo Cataldo[1] Patrick A. Wagstrom[2] James D. Herbsleb[1] Kathleen M. Carley[1]
[1] Institute for Software Research International
[2] Department of Engineering and Public Policy
Carnegie Mellon University
Pittsburgh, PA 15213
{mcataldo,pwagstro}@andrew.cmu.edu jdh@cs.cmu.edu kathleen.carley@cmu.edu

ABSTRACT
Task dependencies drive the need to coordinate work activities. We describe a technique for using automatically generated archival data to compute coordination requirements, i.e., who must coordinate with whom to get the work done. Analysis of data from a large software development project revealed that coordination requirements were highly volatile, and frequently extended beyond team boundaries. Congruence between coordination requirements and coordination activities shortened development time. Developers, particularly the most productive ones, changed their use of electronic communication media over time, achieving higher congruence. We discuss practical implications of our technique for the design of collaborative and awareness tools.

Categories and Subject Descriptors
H.5.3 [**Information Interfaces and Presentation**]: Groups and Organization Interfaces – *collaborative computing, computer-supported cooperative work, organizational design.*

General Terms
Management, Performance, Human Factors.

Keywords
Coordination, Collaboration tools, Task Awareness Tools, Dynamic Network Analysis.

1. INTRODUCTION
It has long been observed that organizations carry out complex tasks by dividing them into smaller interdependent work units assigned to groups and coordination arises as a response to those interdependent activities [21]. Communication channels emerge in the formal and informal organizations. Over time, those information conduits develop around the interactions that are most critical to the organization's main task [12]. This is particularly important in product development organizations which organize themselves around their products' architectures because the main components of their products define the organization's key subtasks [30]. Organizations also develop filters that identify the most relevant information pertinent to the task at hand [9].

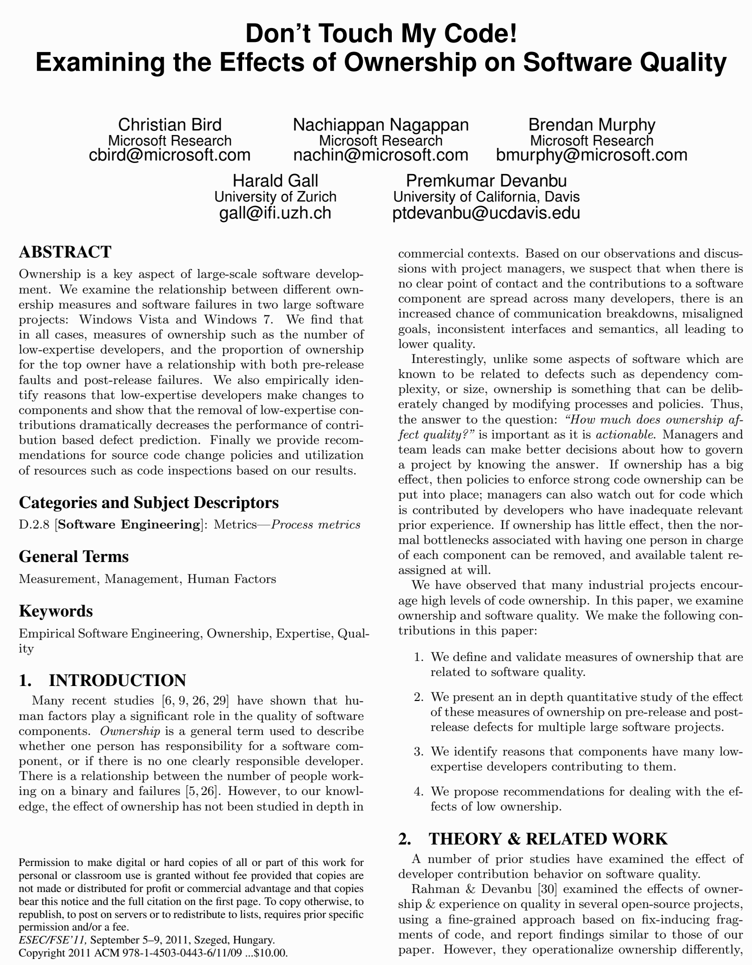
Changes in task dependencies, however, jeopardize the appropriateness of the information flows and filters and can disrupt the organization's ability to coordinate effectively. For example, Henderson & Clark [15] found that minor changes in product architecture can generate substantial changes in task dependencies, and can have drastic consequences for the organizations' ability to coordinate work. If we had effective ways of identifying detailed task dependencies and tracking their changes over time, we would be in a much better position to design collaborative and task awareness tools that could help to align information flow with task dependencies.

Identifying task dependencies and determining the appropriate coordination mechanism to address the dependencies is not a trivial problem. Coordination is a recurrent topic in the organizational theory literature and, as we will discuss in the next section, many stylized types of task dependencies and coordination mechanisms have been proposed over the past several decades. However, numerous types of work, in particular non-routine knowledge-intensive activities, are potentially full of fine-grain dependencies that might change on a daily or hourly basis. Conventional coordination mechanisms like standard operating procedures or routines would have very limited applicability in these dynamic contexts. Therefore, designing tools that support rapidly shifting coordination needs requires a more fine-grained level of analysis than what the traditional views of coordination provide.

In this paper, we develop a technique to measure task dependencies among people, and the "fit" between these task dependencies and the coordination activities performed by individuals. We refer to the fit measure as congruence. Using data from a software development project, we found that patterns of task dependencies among people are in fact quite volatile, confirming our suspicion that a fine-grained view of dependencies is important. We then explored the consequences of congruence for the speed and effi-

Computer Supported Cooperative Work '06, November 4–8, 2006, Banff, Alberta, Canada.

Coordination & Productivity



Don't Touch My Code! Examining the Effects of Ownership on Software Quality

Christian Bird
Microsoft Research
cbird@microsoft.com

Nachiappan Nagappan
Microsoft Research
nachin@microsoft.com

Brendan Murphy
Microsoft Research
bmurphy@microsoft.com

Harald Gall
University of Zurich
gall@ifi.uzh.ch

Premkumar Devanbu
University of California, Davis
ptdevanbu@ucdavis.edu

ABSTRACT
Ownership is a key aspect of large-scale software development. We examine the relationship between different ownership measures and software failures in two large software projects: Windows Vista and Windows 7. We find that in all cases, measures of ownership such as the number of low-expertise developers, and the proportion of ownership for the top owner have a relationship with both pre-release faults and post-release failures. We also empirically identify reasons that low-expertise developers make changes to components and show that the removal of low-expertise contributions dramatically decreases the performance of contribution based defect prediction. Finally we provide recommendations for source code change policies and utilization of resources such as code inspections based on our results.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics—*Process metrics*

General Terms
Measurement, Management, Human Factors

Keywords
Empirical Software Engineering, Ownership, Expertise, Quality

1. INTRODUCTION
Many recent studies [6, 9, 26, 29] have shown that human factors play a significant role in the quality of software components. *Ownership* is a general term used to describe whether one person has responsibility for a software component, or if there is no one clearly responsible developer. There is a relationship between the number of people working on a binary and failures [5, 26]. However, to our knowledge, the effect of ownership has not been studied in depth in commercial contexts. Based on our observations and discussions with project managers, we suspect that when there is no clear point of contact and the contributions to a software component are spread across many developers, there is an increased chance of communication breakdowns, misaligned goals, inconsistent interfaces and semantics, all leading to lower quality.

Interestingly, unlike some aspects of software which are known to be related to defects such as dependency complexity, or size, ownership is something that can be deliberately changed by modifying processes and policies. Thus, the answer to the question: *"How much does ownership affect quality?"* is important as it is *actionable.* Managers and team leads can make better decisions about how to govern a project by knowing the answer. If ownership has a big effect, then policies to enforce strong code ownership can be put into place; managers can also watch out for code which is contributed by developers who have inadequate relevant prior experience. If ownership has little effect, then the normal bottlenecks associated with having one person in charge of each component can be removed, and available talent reassigned at will.

We have observed that many industrial projects encourage high levels of code ownership. In this paper, we examine ownership and software quality. We make the following contributions in this paper:

1. We define and validate measures of ownership that are related to software quality.
2. We present an in depth quantitative study of the effect of these measures of ownership on pre-release and post-release defects for multiple large software projects.
3. We identify reasons that components have many low-expertise developers contributing to them.
4. We propose recommendations for dealing with the effects of low ownership.

2. THEORY & RELATED WORK
A number of prior studies have examined the effect of developer contribution behavior on software quality.

Rahman & Devanbu [30] examined the effects of ownership & experience on quality in several open-source projects, using a fine-grained approach based on fix-inducing fragments of code, and report findings similar to those of our paper. However, they operationalize ownership differently,

ESEC/FSE'11, September 5–9, 2011, Szeged, Hungary.

Coordination & Quality

Latent Social Structure in Open Source Projects

Christian Bird, David Pattison, Raissa D'Souza,
Vladimir Filkov and Premkumar Devanbu
Dept. of Computer Science, Kemper Hall,
University of California, Davis, CA, USA,
cabird,dspattison,rmdsouza,vfilkov,ptdevanbu@ucdavis.edu

ABSTRACT
Commercial software project managers design project organizational structure carefully, mindful of available skills, division of labour, geographical boundaries, etc. These organizational "cathedrals" are to be contrasted with the "bazaar-like" nature of Open Source Software (OSS) Projects, which have no pre-designed organizational structure. Any structure that exists is dynamic, self-organizing, latent, and usually not explicitly stated. Still, in large, complex, successful, OSS projects, we do expect that subcommunities will form spontaneously within the developer teams. Studying these subcommunities, and their behavior can shed light on how successful OSS projects self-organize. This phenomenon could well hold important lessons for how commercial software teams might be organized. Building on known well-established techniques for detecting *community structure* in complex networks, we extract and study latent subcommunities from the email social network of several projects: Apache HTTPD, Python, PostgresSQL, Perl, and Apache ANT. We then validate them with software development activity history. Our results show that subcommunities do indeed spontaneously arise within these projects as the projects evolve. These subcommunities manifest most strongly in technical discussions, and are significantly connected with collaboration behaviour.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management—*programming teams*; D.2.8 [**Software Engineering**]: Metrics—*process metrics*

General Terms
Human Factors, Measurement, Management

Keywords
Open Source Software, social networks, collaboration

This work was supported by a grant from the National Science Foundation Grant no. NSF-SoD-0613949 and software donations from SciTools and GrammaTech Corporations.

SIGSOFT 2008/FSE 16, November 9–15, Atlanta, Georgia, USA

1. INTRODUCTION
Brooks, in his seminal work *The Mythical Man-Month* [12], noted the scaling issues that arise in large software teams: the number of potential interactions grows quadratically with team size, thus quadrupling when the team size is doubled. Clearly, without organization of some kind, both within the software and the community that develops it, there is a limit to how much projects can be scaled.

In traditional, commercial software projects, the response to the Brooksian critique of large teams is to divide and conquer, *by fiat*. The system is deliberately divided into smaller components, and the developer pool grouped into manageable teams which are then assigned to those components. With well-defined interfaces, the teams' efforts are confined to smaller groups, and the coordination needs are moderated. Software design principles such as separation of concerns [53] play a part in this, as does "Conway's Law" [16], which connects artifact structure with organizational structure.

By contrast, Open Source Software (OSS) projects are not formally organized, and have no pre-assigned command and control structure. No one is forced to work on a particular portion of the project. Team members contribute as they wish in any number of ways: by submitting bug reports, lending technical knowledge, writing documentation, improving the source code in various ares of the code base, etc. It has been observed by Sosa *et al.* [56] that the fixed organizational structure found in commercial settings may lead to misalignment with evolving complex products. Henderson and Clark point out that it may may actually hinder innovation [32]. Thus the lack of a rigid organizational structure may in fact be a boon to OSS projects. However, the absence of any structure at all may be just as harmful. Henderson and Clark [32] found that "architectural knowledge tends to become embedded in the structure and information-processing procedures of established organizations". Modularizing artifacts and mapping artifact tasks onto organizational units is a well known solution to the problem of complex product development in organizational management literature [56]. The question then arises, is the social structure of OSS projects free of such constraints and actually unorganized and free-for-all? Do they stand in contrast to the structured, hierarchical style of traditional commercial software efforts? Or, do OSS projects have some latent[1] structure of their own? Are there dynamic, self-organizing subgroups that spontaneously form and evolve?

[1]By latent, we mean not explicitly stated, but observable.

Coordination & Success

Socio-technical Aspects in Software Systems — A take on Conway's Law

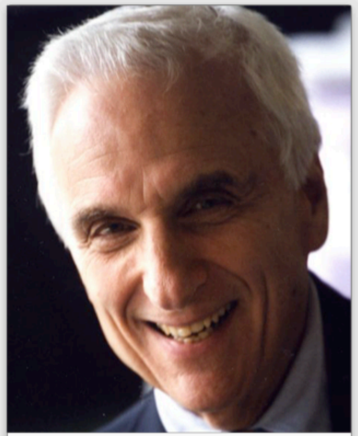
WHAT is important for SOFTWARE QUALITY and WHY?

- When you write software and you care about software quality(*):
  - What are the aspects that you always try to keep in mind and/or the behaviors that you try to have?
    - Form groups of 2/3 students, discuss the answer to the question above, and come up with a list of top-3 *elements* and their rationale.
      Let us discuss the results in 5 minutes.
  - Do these aspects/behaviors change when you know that you are working in a team?
    - Let us find 5 *elements* that matter and their rationale.

yes, but.. how do you know?

(*) e.g., that the software is maintainable and reasonably defect free

Conway's Law — A potential corollary



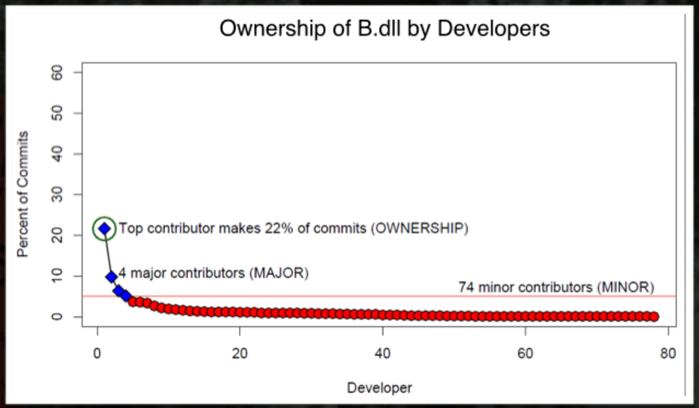
Melvin Conway

Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure.

A software system whose structure closely matches its organization's communication structure works "better."

Don't Touch My Code! — Example metrics for a software component

Metric	Value
Minor	74
Major	4
Total	78
Ownership	22%



Latent social structure in OSS projects — Three hypotheses

- Hypothesis 1

```
Subcommunities of participants will form in the email
social networks of large OSS projects and the levels of
modularity will be statistically significant.
```