APPROVAL SHEET

Title of Dissertation: Measuring Shared Understanding in Software Design Teams

Name of Candidate: Brandt Benedict Braunschweig
Doctor of Philosophy, 2016

Dissertation and Abstract Approved: _____
Dr. Carolyn B. Seaman
Associate Professor
Department of Information Systems

Date Approved: _____

CURRICULUM VITAE

Name: Brandt Benedict Braunschweig.

Degree and date to be conferred: Ph.D., 2016.

Collegiate institutions attended:

2010-2016 University of Maryland, Baltimore County
Ph.D., Information Systems, 2016.

2008-2012 University of Maryland, Baltimore County
M.S., Information Systems, 2012.

1983-1992 Virginia Polytechnic Institute and State University
B.S. Electrical Engineering, 1992.

Professional Publications:

Braunschweig, Brandt, and Carolyn B. Seaman. 2014. "Measuring Shared Understanding in Software Project Teams Using Pathfinder Networks." In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 41:1–41:10. ESEM '14. Torino, Italy: ACM. doi:10.1145/2652524.2652543.

Braunschweig, Brandt, and Carolyn B. Seaman. 2013. "An Examination of Shared Understanding in Free/libre Open Source Project Maintenance." In 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), 113–16. doi:10.1109/CHASE.2013.6614744.

Braunschweig, Brandt, Neha Dhage, Maria Jose Viera, Carolyn B Seaman, Sreedevi Sampath, and A. Günes Koru. 2012. "Studying Volatility Predictors in Open Source Software." In *Proceedings of the 6th International Symposium on Empirical Software Engineering and Measurement*. Lund, Sweden: ACM.

Professional positions held:

2007-2007 Consumer Health Technologies, LLC
Ft. Lauderdale, Florida
Senior Software Architect.

2000-2006	Emdeon Business Services Columbia, Maryland Manager of Product Design and Architecture.
1999-1999	Food Service Technologies, LLC Rocky Mount, North Carolina Principal Software Architect.
1990-1999	Recognition Research, Inc. Blacksburg, Virginia Product Development Manager.
1984-1990	Image Processing Technologies, Inc. Blacksburg, Virginia Lead Programmer.

ABSTRACT

Title of Document: MEASURING SHARED UNDERSTANDING IN SOFTWARE DESIGN TEAMS

Brandt Benedict Braunschweig, Ph.D., 2016

Directed By: Dr. Carolyn B. Seaman, Department of Information Systems

Background

Software engineering teams must have a shared understanding of the system design in order to work independently but successfully integrate their code. These issues of understanding are important to project success but difficult to investigate with current approaches. Current techniques for investigating shared understanding, such as interviews or questionnaires, are limited by the difficulty of team members to externalize knowledge relevant to shared understanding.

Aims

This research has two goals. The first goal is to identify and validate a measure of shared understanding that researchers can use to investigate issues of shared understanding in software design. The second goal is to evaluate the potential for this measure to be used by practitioners to improve the software design process.

Method

A measure of shared understanding was developed by adapting an approach from the Team Mental Models literature. Five student teams and two industrial teams were recruited to evaluate the measure empirically. The validity of the measure, the significance of the differences in understanding found, and the applicability for design process improvement were investigated using qualitative techniques, including group interviews, observation, and questionnaires.

Results

When ranked by the measure of shared understanding, high ranking design concepts were generally, but not consistently, found to be associated with greater similarity of understanding than low ranking concepts. This supports a finding that the measure is valid, but imprecise. Although no specific misunderstandings were identified within the team, some team members found the discussion, guided by the measured differences, valuable for improving shared understanding generally.

Conclusions

The results support the use of the measure as a tool to investigate shared understanding so long as consideration is given to its limitations. It is premature for practitioners to use the measure to improve the design process. The results are based on only two industrial teams without a history of failures related to shared understanding. Future research should re-evaluate the measurement in different contexts. Guidance is given for additional research to refine the measurement.

MEASURING SHARED UNDERSTANDING IN SOFTWARE DESIGN TEAMS

By

Brandt Benedict Braunschweig.

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2016

ProQuest Number: 10118999

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest 10118999

Published by ProQuest LLC (2016). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

© Copyright by
Brandt Benedict Braunschweig
2016

Dedication

For Suzanne: You make everything possible.

Acknowledgements

I would like to thank everyone that helped me on this long journey. First, I would like to thank my advisor, Dr. Carolyn Seaman, for her endless advice and guidance. I'm grateful for the assistance of my committee members Dr. Anthony Norcio, Dr. Lina Zhou, Dr. Malcom Gethers, and Dr. J. Alberto Espinosa. Thank you to Dr. Wayne Lutters who helped me explore the human side of technical systems.

I'd like to thank Dr. Susan Mitchell and Flynn Wolf for being note takers, and more importantly for their conversation and encouragement. I'm grateful for Dr. Rosa Heckle's enthusiastic support.

Peter and Mary Lou Hill provided unwavering encouragement. Thank you for everything.

I especially want to thank the participants that made this research possible: the professional programmers that made time for me, their management who granted access, and the student teams that took on one more thing.

Part of this work was supported by a grant from the UMBC Graduate Student Association.

Table of Contents

Dedication ii
Acknowledgements iii
Table of Contents iv
List of Tables vii
List of Figures viii
Chapter 1: Introduction 1
1.1 Theoretical Development 7
1.1.1 Synthesizing a Cognitive View of Team Software Design 7
1.1.2 The Theory Underlying Team Mental Models 15
1.2 Research Questions 19
1.3 Guide to the Rest of This Document 21
Chapter 2: Related Research 23
2.1 Approaches to Measuring Team Mental Models 23
2.1.1 Common Characteristics 24
2.1.2 Elicitation Methods 28
2.2 Prior Research Measuring Shared Understanding in Software Engineering 34
2.2.1 A Prior Study in Software Usability 35
2.2.2 A Prior Study in Requirements Understanding 37
2.2.3 A Prior Study in Software Team Research 38
2.2.4 Other Notable Research Measuring Team Mental Models 40
Chapter 3: Development of the Measure 45
3.1 Guiding Criteria 45
3.2 Selection of a Starting Technique 47
3.3 Paired Comparisons and Pathfinder 48
3.3.1 Concept Selection 49
3.3.2 Paired Comparison Ratings 49
3.3.3 Pathfinder Network Scaling 50
3.3.4 Graph Comparison 52
3.4 The Shared Understanding Measure for Software Teams (SUM-ST) 58
3.4.1 Concept Selection 59
3.4.2 Elicitation of Relatedness Ratings 60
3.4.3 Concept Similarity Calculation 61
3.4.4 Result Interpretation 64
Chapter 4: Method 66
4.1 General Study Approach 66
4.2 Pilot Study 68
4.2.1 Study Purpose and Research Questions 68
4.2.2 Pilot Study Method 70
4.2.3 Pilot Study Results 83
4.2.4 Pilot Study Discussion 95
4.3 Main Study Method 101
4.3.1 Design 101
4.3.2 Implementation 115
Chapter 5: Results 128

5.1 Ratings Questionnaire Results 128
5.1.1 Relatedness Ratings 128
5.1.2 Measures of Shared Understanding 132
5.1.3 Changes Over Time 137
5.2 Group Interview Results 139
5.2.1 Visualizations 139
5.2.2 Similarity Measure and Shared Understanding 141
5.2.3 Relationship of Measures to Outcomes 157
5.2.4 Usefulness of SUM-ST for Design Process Improvement 160
Chapter 6: Discussion 165
6.1 Research Question 1 165
6.1.1 Issues of Precision Attributable to the Measurement Method 167
6.1.2 Issues of Precision Attributable to the Phenomenon 175
6.1.3 Relationship Between Measures and Complexity 177
6.2 Research Question 2 179
6.2.1 Role Related Differences in Understanding 180
6.2.2 Nature of Collaboration in Selected Concepts 180
6.2.3 Shared Understanding Vs. Leader's Understanding 181
6.3 Research Question 3 182
6.4 Proposed SUM-ST Changes 186
6.4.1 Changes to the SUM-ST Method to Improve Precision 187
6.4.2 Using SUM-ST to Improve the Design Process 189
6.5 Recommendations and Future Work 190
6.6 Implications for Practitioners 193
6.6.1 Manage Terminology 193
6.6.2 Fit processes to needs 194
6.6.3 Adapt to changes in expertise 195
6.7 Validity Analysis and Limitations 195
6.8 Conclusions 198
6.9 Contributions 201
6.10 Summary 203
Appendix A: Associations Between Experience, Communication, and Measures of Knowledge Structures 206
A.1 Method 207
A.2 Results 208
A.3 Discussion 209
Appendix B: Pilot Study Additional Material 212
B.1 Demographic Questionnaire 212
B.2 Interaction Guide 213
B.3 Post-Questionnaire Interview 219
B.4 Team 10 Interview Guide 220
Appendix C: Main Study Additional Material 222
C.1 Supplemental tables 222
C.2 Demographic and Interaction Questionnaire 227
C.3 Team 2 Interview Guide 229
Appendix D: Executive Summary 231

Bibliography 236

List of Tables

Table 2.1. Summary of Other Research 41
Table 3.1. Example Relatedness Ratings 50
Table 3.2. Edge Weight Matrix Corresponding to Table 3.1 51
Table 3.3. C Node Calculations for Figure 3.2 54
Table 3.4. Distance Matrix for PFNet A in Figure 3.2 57
Table 3.5. Distance Matrix for PFNet B in Figure 3.2 57
Table 3.6. C_v Node Calculations for Figure 3.2 62
Table 3.7. r_v Node Calculations for Figure 3.2 63
Table 3.8. Summary of Graph Similarity Calculations 64
Table 4.1. Pilot Study Participants 79
Table 4.2. Team 10 Concepts Ranked by $\overline{C_v}$ 82
Table 4.3. Group Interview Codes 83
Table 4.4. Pilot Study Ratings Frequency Distribution 87
Table 4.5. Relatedness Ratings for Team 13, Team Members 1 and 2 89
Table 4.6. Team Similarity Questionnaire 1 vs. Questionnaire 2 92
Table 4.7. Self-Similarity Questionnaire 1 vs. Questionnaire 2 92
Table 4.8. Team 1 Participants 117
Table 4.9. Team 2 Participants 119
Table 4.10. Main Study Code List 126
Table 5.1. Questionnaire Part I Response Frequencies 129
Table 5.2. Team 1's Concepts Ranked by Questionnaire 1 134
Table 5.3. Team 2 Concepts Ranked by Questionnaire 1 135
Table 5.4. Team 2, Developers Only, Concepts Ranked by Questionnaire 1 136
Table 5.5. Disconnected Concepts 137
Table 5.6. Team Similarity Questionnaire 1 vs. Questionnaire 2 137
Table 5.7. Self-Similarity Questionnaire 1 vs. Questionnaire 2 138
Table 5.8. Partial Ratings for Team 2, Concept C2.2 149
Table A.1. Pairwise Experience, Communication, and Similarity Measure C 210
Table A.2. Correlation of Similarity Measure C vs. Experience and Communication 211
Table C.1. Main Study, Team 1 Pairwise Experience 222
Table C.2. Main Study, Team 2 Pairwise Experience 223
Table C.3. Main Study, Team 1 Pairwise Communication 223
Table C.4. Main Study, Team 2 Pairwise Communication 224
Table C.5. Main Study, Team 2 Similarity Excluding Individuals 226

List of Figures

Figure 1.1. Knowledge integration during team design 2
Figure 1.2. Shared Understanding in Software Design 15
Figure 1.3. Parallel Decision Making Activities 18
Figure 2.1. Example Knowledge Structure 25
Figure 2.2. Stages of TMM Measurement 28
Figure 3.1. Full Graph and PFNet for the Data in Table 3.1 51
Figure 3.2. PFNets Differing in Edges About Node v_4 'Leaves' 53
Figure 4.1. Pilot Study Design Overview 72
Figure 4.2. Pilot Study Rating Question 75
Figure 4.3. PFNet for Team 10, Participant 2 81
Figure 4.4. Graph Similarity Comparison 90
Figure 4.5. Main Study Design Overview 104
Figure 4.6. Main Study Rating Question 107
Figure 4.7. Main Study Relationship Description Question 108
Figure 4.8. Result Visualization Tool, Selection Page 112
Figure 4.9. Result Visualization Tool, Ranked Concepts 112
Figure 4.10. Result Visualization Tool, Team Map View 113
Figure 4.11. Result Visualization Tool, Detail Map View 113
Figure 5.1. Detail View for Team 1, Q1, C1.10 and C1.15 140
Figure 5.2. Team Map for Team 2, Questionnaire 1 142
Figure 5.3. Detail for Team 2, Concept C2.2 149

Chapter 1: Introduction

At its essence, software development is a problem solving activity. The process begins with a statement of system requirements that defines the problem to be solved. The solution to this problem requires creating a software design that specifies the form and function of the solution, implementing the design in software, testing the software for compliance with the requirements, and finally, delivering the software to put it into use. After delivery, assuming all went well, the problem should be solved. After the initial delivery, a software maintenance phase occurs in which changes are made to address previously undetected defects and evolving aspects of the problem.

In order to design the software, developers must integrate knowledge about the requirements, the problem domain, and the technologies available, into one coherent solution. As software has grown more complex, so has the diversity of knowledge required. To meet these knowledge demands, it is common for software to be developed by a team of people, each with specialized knowledge and skill. The overall design is decomposed into modules, which are implemented separately, and then integrated to create the final solution (Figure 1.1.) Although this approach attacks the problem of knowledge diversity, breaking the design activity across many people makes it difficult to keep the design internally consistent. Subtle differences in understanding between developers about the design can lead to failures, such as an integration failure that occurs when two different developers each believe they have implemented the design faithfully and, yet, the program does not work as intended when the parts are integrated (Leung and White 1990).

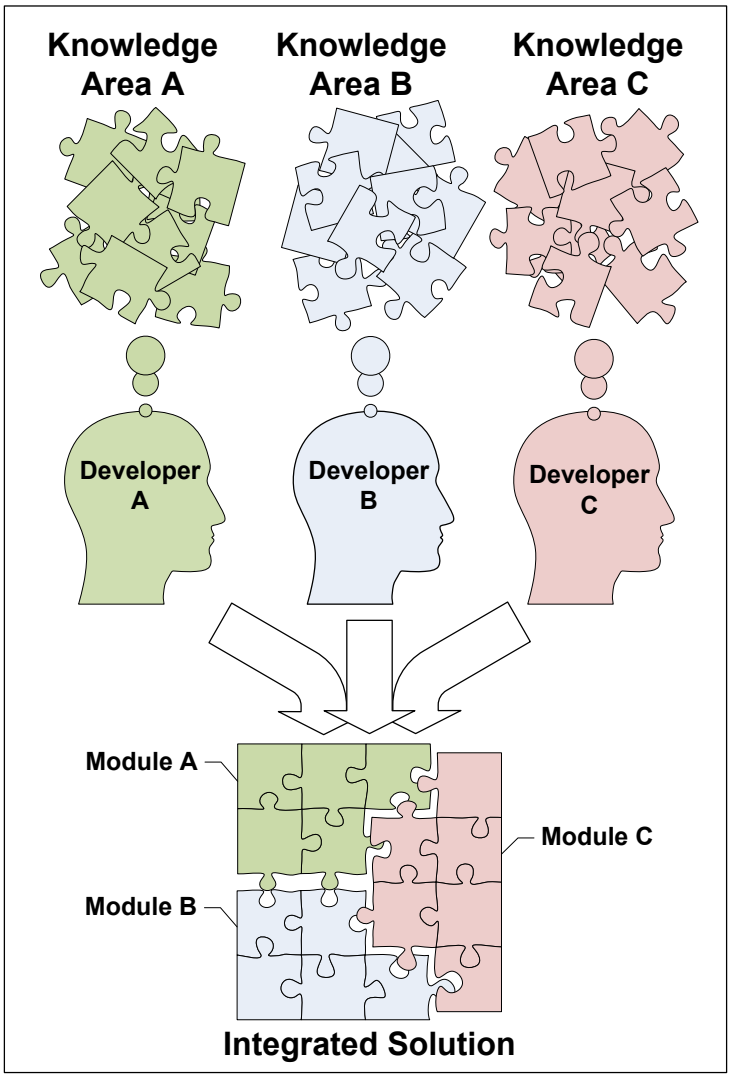


Figure 1.1. Knowledge integration during team design

To be successful, the developers must create and maintain a *shared understanding* of their design (Curtis, Walz, and Elam 1990; Aranda 2010). Shared understanding describes the ability of a team to apply shared knowledge to achieve a shared goal (Smart et al. 2009). Shared, in this sense, refers to something that is held in common, not something that is divided and apportioned to different members (Cooke et al. 2000). A key aspect of shared understanding is the ability of the team to coordinate their work with a minimum of overt planning and communication (Espinosa, Lerch,

and Kraut 2004). When a team has a shared understanding, members can work independently without creating conflicts between the working modules of the system.

Software design teams use many techniques to share knowledge and to check for misunderstandings. Writing and reading documentation (Whitehead 2007) and attending meetings (Curtis, Krasner, and Iscoe 1988) are the most obvious ways to share knowledge, but impromptu conversations (Curtis, Krasner, and Iscoe 1988), pair programming (Sharp and Robinson 2004), and reading source code (de Souza, Froehlich, and Dourish 2005) also share knowledge. Verification techniques, such as design reviews and integration testing, can identify misunderstandings within the team (Pfleeger 2001). However, even when these techniques are used, software development failures are common.

We understand little about how these activities effect their desired outcome. Although shared understanding has long been recognized as important in software design, it is poorly understood (Aranda 2010) and there is little research into how shared understanding develops and evolves during the software design process. One reason is that it is difficult to gather meaningful data about a team's understanding. An individual's understanding is intangible and cannot be measured directly. Additionally, people are unaware of much of what they understand or they find it hard to verbalize (Ryan and O'Connor 2009; Mohammed, Ferzandi, and Hamilton 2010). Therefore, traditional data gathering techniques, such as interviews or questionnaires, are limited in their ability to provide meaningful empirical data.

To investigate issues of shared understanding in software design, we need a technique that can measure shared understanding. Such a technique could be used by researchers to investigate how different software development practices affect the shared understanding of the team or how the development of shared understanding relates to team success. Therefore, the first goal of this research was to identify and validate a technique to measure shared understanding in the context of team software design.

If a technique for measuring shared understanding is specific about the source of the similarities and differences in how the team understands their design, practitioners may be able to act on the differences to improve the software design process. For example, an *interpretation error* occurs when one developer interprets a design specification differently from how the creator of the specification intended (Leung and White 1990). This, in effect, creates two incompatible versions of the specification, the one envisioned by the creator and the one envisioned by the developer. Ideally, such a misunderstanding would be caught during a design review but it is not possible for reviews to be exhaustive. A latent problem may go undetected until it is made explicit through a failure during test or use.

For practitioners, a technique that can reveal specific differences in understanding may complement other techniques, such as design reviews, by revealing differences in understanding that team members may not be aware of. This would allow the team to direct their development efforts to correct those differences earlier in the design process than they otherwise might, providing benefits to both cost and schedule.

Therefore, the second goal of this research was to evaluate the potential for a technique to measure shared understanding to improve the software design process.

How does one operationalize shared understanding? Recent research in Industrial and Organizational Psychology has been working to do just that. The construct of Team Mental Models (TMMs) offers a theoretical basis for shared understanding and many approaches for measuring TMMs have been proposed (Mohammed, Ferzandi, and Hamilton 2010). There are three main problems adapting this work to measure shared understanding in a software design team:

- TMMs were developed and validated with military action teams, such as aircraft flight crews. Because of the significant differences in context between action teams and software design teams, the measurement approach must be revalidated in the new context.

- The TMM literature offers many diverse approaches for measuring TMMs, not well-defined techniques. Each approach has its own advantages and limitations and it is not clear which approach is appropriate. The existing literature must be distilled to select one specific technique to test.

- TMM measurement approaches are generally not actionable by the team to improve shared understanding. Existing research has focused on establishing a statistically significant correlation between a measure of the TMM and a measure of the team's performance. This allows for comparisons between teams but doesn't explain how a team can make improvements. To identify problems that may lead to a failure, the measurement technique should elicit

knowledge the team is unaware of, identify specific areas of misunderstanding, and the results should be interpretable by the team.

This research addresses these three problems to propose a specific technique for measuring shared understanding in the context of team software design. The research approach taken was to first develop a theoretical understanding of *how* shared understanding supports effective team design. This was used as a guide to evaluate the TMM literature, select an appropriate starting point, and extend it as necessary. Then, this proposed technique was evaluated empirically with student and industrial software design teams. This evaluation was intended to be informative not only about the technique, but also about the role of shared understanding in software design.

This dissertation contributes to the body of software engineering research by:

- synthesizing a theoretical understanding of team software design based on existing literature.
- proposing one well-defined measurement technique, appropriate to the context of software design teams, and a description of its capabilities and limitations.
- proposing an interactive visualization tool that allows exploring data captured about team mental models.
- providing knowledge about how practitioners may use measures of shared understanding to improve the software design process, based on industrial trials.
- providing a qualitative research method to investigate software development methods and shared understanding.

- providing a description of the nature of shared understanding in a software design team and how it is similar and different from previous studies.

This introduction continues with the theoretical development in section 1.1. This begins with a synthesis of a cognitive view of shared understanding in the process of software design in section 1.1.1. This is followed by an introduction to the theory of team mental models in section 1.1.2, which also explains why they are a valid way to operationalize shared understanding in team design. This theoretical development leads to my research questions, presented in section 1.2. Section 1.3 is a guide to the rest of this document.

1.1 Theoretical Development

The goal of this section is to provide a theoretical basis for the research conducted. This section begins with a description of the process of team software design and an explanation of the role of shared understanding in improving software design. Next, I review the theory underlying Team Mental Models and show that, even though it was developed independently within the discipline of industrial and organizational psychology, it describes the same cognitive processes found in software design and is, therefore, appropriate for operationalizing shared understanding in a software design team.

1.1.1 Synthesizing a Cognitive View of Team Software Design

It is curious that, as important as the process of design is to software engineering, there is not a well-defined description of this process. The act of design is commonly defined in terms of the artifacts produced, such as system specifications (IEEE 1990;

Borque and Fairley 2014, 2–1), without addressing *how* they are produced. What follows is a synthesis of the available literature to explain how software developers produce a software design.

This is presented in two parts. The first part explains the cognitive processes of a single person engaged in the act of design. The second part extends this to consider the role of shared understanding when multiple developers are collaborating in a team.

1.1.1.1 A Cognitive View of the Design Process

The process of software design can best be described as a decision making process. As a process, it consists of multiple steps, each of which contributes to a final choice. For example, to meet a requirement to produce a sorted list, a developer may need to choose the sorting algorithm to use. They must first identify the available algorithms to choose from, evaluate their relative merits, and make a choice. The design specification is a record of the choice that was made. The rest of this section describes this process in more detail.

When designing, developers make decisions about the form and function of the software so that it meets the solution requirements and conforms to constraints imposed by the various technical components used (Taylor and van der Hoek 2007; Mitchell 1989; Herbsleb and Mockus 2003; Tang and Aleti 2010). Herbsleb and Mockus describe this process:

> One can then think of any particular software project as defined by the combinations of choices for those decisions that satisfy the requirements for that project. … Software engineering work proceeds by making choices for all of the decisions. As each is made, fewer decisions remain, until all of the decisions are made, resulting in a final product that may or may not satisfy the requirements. (Herbsleb and Mockus 2003)

Mitchell (1989) breaks down the decision making process in terms of traditional decision theory. Design decisions consist of the following steps:

- Create design alternatives
- Create criteria to evaluate the alternatives
- Choose between alternatives using criteria
- Check for consistency and conformance
- Cycle, covering more aspects and/or more detail.

The criteria used to evaluate the design alternatives are determined by the system requirements and the technical dependencies imposed by the prior design choices (Taylor and van der Hoek 2007).

The evaluation of design alternatives presents a chicken and egg problem. The evaluation requires comparing the system runtime behavior against the requirements but this isn't possible because the system doesn't exist yet. In order to evaluate alternatives, the developers use mental simulations to predict the runtime behavior for

each alternative (Adelson and Soloway 1986; LaToza, Venolia, and DeLine 2006; Curtis, Walz, and Elam 1990; Gasson 1998). The developers evaluate the predicted behavior of each alternative according to the design criteria to make their choice. Therefore, the quality of the design depends in large part on the ability of the developer to perform detailed and accurate mental simulations to predict the resulting system behavior and to compare these mental predictions against the criteria.

The ability to perform mental simulation is described by the psychological concept of a *mental model*. Although there are a variety of interpretations of mental models in the literature (Klimoski and Mohammed 1994), the functional view is that people construct a working model of the real world in their mind that allows them to make predictions about situations, either real or imagined (Cannon-Bowers, Salas, and Converse 1993). Mental models are defined as "the mechanisms whereby humans generate descriptions of system purpose and form, explanations of system functioning and observed system states, and predictions of future system states" (Rouse and Morris 1985).

Mental models, in turn, rely on the person's knowledge about the situation, including rules about how the situation may transform over time. More than a collection of facts, this knowledge is believed to be *structured*. This organized knowledge structure (also *schemata*) includes "objects, situations and events, and the *relationships between them*" [emphasis added] (Cannon-Bowers, Salas, and Converse 1993). Changes to either the knowledge or relationships may change the mental model and produce different predictions.

To summarize: Mental models are based on organized knowledge structures and are the mechanism people use for mental simulation and prediction. Developers use mental models to predict the runtime behavior of various design alternatives under consideration. They evaluate the predicted behavior according to the design criteria and choose the alternative that best satisfies the criteria.

More specifically, developer mental models consist of a model of the software being created supported by an extensive knowledge base that includes knowledge of the problem domain, the technologies being used, and similar problems solved previously (Storey 2006). Research has shown that developers invest a lot of time creating and maintaining their mental models (LaToza, Venolia, and DeLine 2006).

Difficulties with mental models can affect the design decision making process. Mental models may be incomplete or inconsistent, limiting their accuracy and predictive power (Guindon, Krasner, and Curtis 1987). Limits on human cognition prevent developers from keeping the entire model in focus at once (Adelson and Soloway 1986). Developers must frequently shift from one part of the model to another. It is, therefore, more accurate to think of the developers as having a collection of mental models rather than one large model. As the software design changes, so does the mental model. Developers tend to start with an abstract model and refine it as the design progresses (Adelson and Soloway 1986).

The process of software design is a complex decision making process that requires extensive mental simulation and evaluation in order to make the multitude of decisions required to define the form and function of the system. Mental simulation

and prediction is enabled by mental models of the problem and the solution. These mental models rely on detailed knowledge structures.

1.1.1.2 Shared Understanding and Team Design

An important aspect of the design decision making process described above is the interdependent nature of the design decisions. The decisions are interdependent because the choice of one alternative may constrain the available alternatives for another decision (Herbsleb and Mockus 2003; Parnas 1971). For example, the choice of data elements to present to the user on a screen constrains the choices for the underlying data structures to store that data and the data validation logic that can be applied. The design of a software system may contain many complex chains of interdependent decisions. Although they are commonly described as software dependencies between modules, they are really dependencies between the decisions made about the module design (Parnas 1971).

When more than one person is involved in the design activity, the design decisions are distributed among the team. In this case, dependencies between distributed decisions become dependencies between the developers making those decisions (Herbsleb and Mockus 2003). Therefore, team design is a *distributed decision making process*. Distributed decision making is different from group decision making. In group decision making, multiple people are contributing to a single choice. In distributed decision making, multiple people are making many decisions independently that support a common objective (Brehmer 1991). In group decision

making, everyone has the opportunity to understand the problem as a whole. In distributed decision making, each person understands only part of the problem.

To have an internally consistent design, different developers must be able to make choices that are compatible with the dependent decisions made by others in the team. The management of technical dependencies between parts of a software system is known as *technical coordination* (Espinosa, Lerch, and Kraut 2004; Curtis, Krasner, and Iscoe 1988). Traditional coordination theory defines coordination as managing dependencies between activities (Malone and Crowston 1994), however, many of the same principles apply to coordinating decisions. In a way, coordinating design decisions is indirectly coordinating future, planned activities. Coordinated design decisions produce software that is properly coordinated at runtime.

Coordination mechanisms may be either *explicit*, such as planning and communication, or *implicit* in which team members act in a coordinated way without explicit mechanisms (Espinosa, Lerch, and Kraut 2004). In the context of software design, shared understanding describes a mechanism for implicit coordination of distributed decisions within the team.

Easterbrook defines shared understanding as follows:

> Two or more people have a shared understanding of a situation if they have equivalent expectations about that situation. By 'expectations' we mean their explanations of the situation and predictions for how it might develop. We assume that such expectations are based on some form of mental model of the

> situation, although it is not our purpose to characterise such mental models further. (Easterbrook 1995)

Shared understanding enables implicit coordination through an ability of different team members to make similar predictions about the future. This allows different team members to either follow parallel decision making processes to arrive at similar choices directly, or it allows team members to consciously anticipate the choices of others and adapt their decisions to avoid anticipated conflicts. Therefore, a team that has a shared understanding of their design has the ability to coordinate distributed design decisions with a minimum of overt planning and communication.

Easterbrook (1995) describes how shared understanding transforms. *Coordination breakdowns* occur when one team member acts in a way that is unexpected to another, resulting in uncoordinated work. Coordination breakdowns reveal *conflicts*, which are differences in the understanding of the members of the group. Once the conflicts are exposed, the team can examine the differences in their understanding, repair the breakdown, and build additional shared understanding, a process called *harmonization*. A team with a greater shared understanding of their design is more likely to avoid conflicts in the design and thereby avoid the failures and rework required to detect and correct them.

In review, shared understanding describes the ability of the team to apply shared knowledge to achieve a shared goal. This ability is rooted in the decision making process of software design. A team has a shared understanding of their design when their individual mental models enable similar predictions of the outcome of different

design alternatives, allowing the team to implicitly choose alternatives that do not conflict with each other. This sequence is shown in Figure 1.2. This suggests that an approach to measuring shared understanding should be based in the team's mental models and the decision criteria linking dependent design decisions.

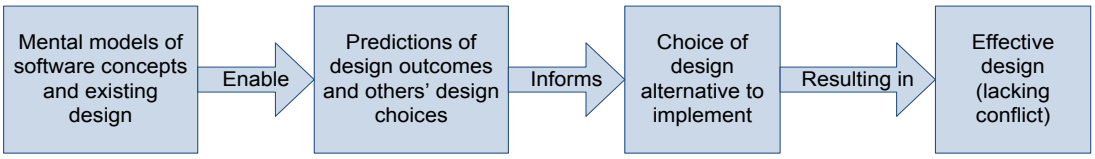


Figure 1.2. Shared Understanding in Software Design

1.1.2 The Theory Underlying Team Mental Models

This section reviews the theoretical development of the Team Mental Models construct. Although it was developed independently, the theory underlying TMMs describes a similar process of coordinated decision making described in the previous section.

The Team Mental Models construct developed as part of research into the distributed decision making process of expert teams (Cannon-Bowers, Salas, and Converse 1993; Kleinman and Serfaty 1989). These are *action teams* such as "cockpit crews, surgery teams, fire fighting teams, and military teams" (Cannon-Bowers, Salas, and Converse 1993). A team is a group of people working toward a shared goal. Specifically, a team is defined as:

> a group of two or more individuals who must interact cooperatively and adaptively in pursuit of shared valued objectives. Further, team members have clearly defined differentiated roles and responsibilities, hold task-relevant

> knowledge, and are interdependent (i.e., must rely on one another in order to accomplish goals.) (Cannon-Bowers, Salas, and Converse 1993)

Team Mental Models are an attempt "to explain the performance of expert teams who demonstrated the ability to seamlessly coordinate their actions without the need for overt communication" (DeChurch and Mesmer-Magnus 2010). The TMM literature hypothesizes that the ability to coordinate action without explicit communication is based in the ability of team members to properly anticipate the actions and needs of others in the team:

> When a novel event is encountered, teams that cannot strategize overtly must rely on existing expectations regarding the task and team in order to decide what action to take. The role of mental models in explaining team performance, then, stems from their ability to provide a set of organized expectations for performance from which accurate, timely predictions can be drawn. (Cannon-Bowers, Salas, and Converse 1993)

These predictions, in turn, are formed by a mental simulation that is based on "running a mental model" (Cannon-Bowers, Salas, and Converse 1993). It is assumed that, for team members to have compatible expectations, something is shared between their individual mental models. Therefore, "Team mental models are team members' shared, organized understanding and mental representation of knowledge or beliefs about key elements of the team's relevant environment" (Mohammed, Klimoski, and Rentsch 2000). The TMM literature emphasizes the role of knowledge structure in

mental models and that any measurement approach must measure not only knowledge, but the relationships that form its structure.

There are some variations in terminology in the TMM literature. The construct was originally termed *shared mental models*, but some researchers feel it is important to emphasize that they are descriptions of teams, as defined above, and not just any group of people (Klimoski and Mohammed 1994). Also, the terms *mental model* and *knowledge structures* are sometimes used interchangeably (Johnson and O'Connor 2008) even though the term mental model usually implies abilities (such as prediction) above and beyond the knowledge structures they depend on.

TMMs are sometimes classified according to how they are expected to achieve coordination (Cannon-Bowers, Salas, and Converse 1993). Models of *taskwork* allow predictions about the tasks required to accomplish work and the interdependencies between those tasks. Models of *teamwork* allow making predictions about team members to correctly anticipate their needs or actions. These are general classifications of what are assumed to be much more specific models that arise organically from the needs of the particular team and their situation (Mohammed, Ferzandi, and Hamilton 2010).

In summary, the TMM literature assumes that team members use their mental models of the task and the others in the team to make predictions about the outcomes of their actions. They use these predictions to choose the action that they should take. Teams whose members have similar mental models are more likely to correctly anticipate the needs and actions of others and avoid conflicts, leading to more effective teams. This

sequence is shown in the bottom of Figure 1.3 alongside the sequence from shared understanding in software design for comparison.

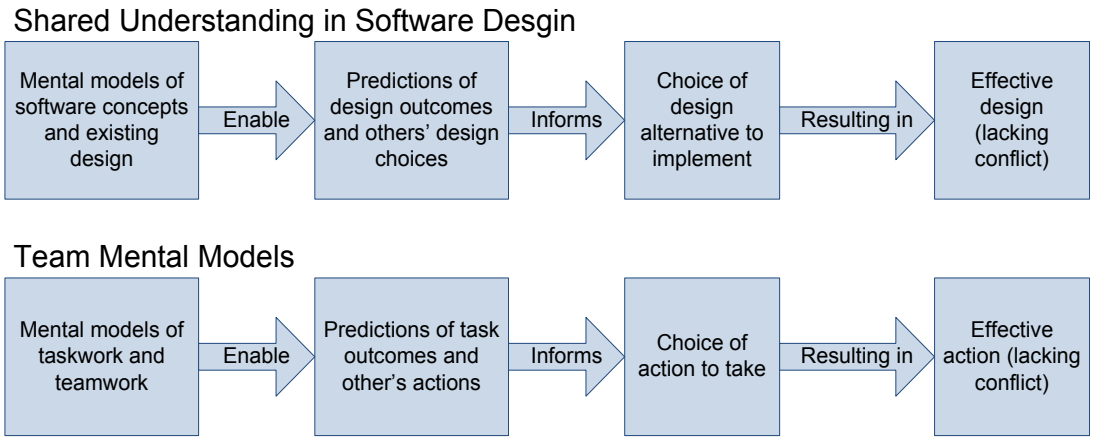


Figure 1.3. Parallel Decision Making Activities

The TMM construct, based on a model of expert team decision making, closely parallels the team decision making process in software design. In both cases, mental models enable the evaluation of decision alternatives and support choices that avoid conflict. In TMM, the choices involve individual actions in support of the team goal. In software design, the choices involve design decisions in support of the system requirements. Given these parallel theories of distributed decision making, it is reasonable to use measures developed for TMMs to operationalize shared understanding for software design teams.

Many different approaches have been developed to measure TMMs. These are described in detail in section 2.1. Although the approaches are quite different, they all assume that the greater the similarity of individual team member knowledge structures, the more likely the team members are to make similar predictions and

work without conflict. Therefore, these measurement approaches focus on ways to elicit knowledge structures and compare them. Numerous research studies have shown that TMM measures of similarity of individual knowledge structures are positively associated with measures of team performance (Cooke, Kiekel, and Helm 2001a; Edwards et al. 2006; Lim and Klein 2006; Mathieu, Goodwin, and Heffner 2000; Stout et al. 1999). In this study, I selected one of these measurement approaches, modified it to meet the research objectives, and then evaluated it empirically.

1.2 Research Questions

The objective of this research is to develop and evaluate a technique for measuring shared understanding in the context of software design, based on measurement techniques from the team mental models literature. I named this technique the Shared Understanding Measure for Software Teams (SUM-ST).

This research is explanatory in nature. Although I was interested in validating a numerical measurement, I was also interested in learning more about what the measure represents, the details of the phenomenon it is sensitive to, and knowing this, its fitness for a particular purpose: improving shared understanding. Therefore, the following research questions are phrased as "how" questions about the relationships of interest.

To validate SUM-ST, it was necessary to choose an independent measure for comparison as there are no widely accepted references. One approach would have been to use a measure of team performance, such as integration failures, rework, or

development time. However, this would introduce a number of potentially confounding variables, such as team processes, team communication, and organizational management, making it impossible to determine the association of shared understanding with the measured effect.

Therefore, I chose a direct comparison of the measurement results against the team member's own perceptions of their understanding. One concern about relying on team member perceptions is that team members may not be aware of some aspects of their understanding. This is addressed by first presenting the measurement results to the team members, making them aware, and allowing them to confirm or refute them. The validity of the measure, therefore, is rooted in the association between the measurement of their knowledge structures and their perceptions of their design. This leads to the first research question:

> RQ1: How are similarities and differences between the software development team members' knowledge structures, as measured by SUM-ST, associated with similarities and differences in how those team members understand their design?

Not all differences in knowledge structures are expected to be associated with design problems. Team members are expected to have differences in knowledge structures related to differences in specialized skill and experience. Even when differences are related to an area of collaboration, they may not be associated with interdependent design decisions. This leads to the following question:

RQ2: How are measured differences in software development team members' knowledge structures associated with identifiable difficulties in the software design process resulting in degradation of cost or quality?

If the team members are aware of the differences in their understanding, it is possible they could use this information to find and correct design problems earlier in the design process than they otherwise would. This leads to the last research question:

RQ3: How can measured similarities and differences in team member knowledge structures be used to improve the process of software design?

1.3 Guide to the Rest of This Document

The remainder of this document is organized as follows. Chapter 2 provides a structured way to compare TMM measurement approaches and then reviews and compares various approaches that appear in the literature. This forms the basis for selecting a starting point for a particular measurement technique in Chapter 3. Chapter 2 also reviews prior research measuring shared understanding in the field of software engineering. Chapter 3 contains a critical assessment of TMM measurement techniques and selects a starting point to use in the software design context. It then explains how I have adapted it and describes the resulting technique in detail. Chapter 4 presents the research method used in this study, including the full description of a pilot study used to refine the method and gather preliminary results. Chapter 5 presents the results of the study. Chapter 6 discusses the results of both the pilot and main studies and answers the research questions. The Appendices include supplemental material, including research instruments, additional detailed raw data,

and a quantitative analysis of the relationship between experience, communication, and measures of shared understanding from the main study.

Chapter 2: Related Research

This chapter presents relevant prior research. It begins by reviewing approaches to measuring team mental models. This is followed by a review of prior research involving measurement of shared understanding specifically in the field of software engineering.

2.1 Approaches to Measuring Team Mental Models

This section describes approaches to measuring team mental models from the literature. As described in section 1.1.2, the team mental model construct theorizes that, when team members hold portions of their knowledge structures in common, they are more likely to correctly anticipate each other's needs and actions, allowing them to act in a coordinated way with a minimum of communication. The measurement of a team mental model, therefore, is usually constructed as a measure of the homogeneity of the individual knowledge structures held by individual team members. Although TMM measurement approaches are maturing, there are no well-defined, generally accepted techniques. Rather, there are a number of variations on some common themes. Differences in approaches are attributable to different research applications, different aspects of team knowledge being investigated, or differences in the pre-existing research techniques they were adapted from.

Although measurement approaches are diverse, they share some common characteristics that allow for a comparative analysis (Mohammed and Dumville 2001; DeChurch and Mesmer-Magnus 2010; Langan-Fox, Code, and Langfield-Smith 2000). This review proceeds by describing these common characteristics in section

2.1.1 and then comparing some common measurement approaches according to these characteristics in section 2.1.2. This review provides a framework to understand prior research involving TMM measurement in section 2.2 and the rationale for selecting a starting point for a specific measurement technique in chapter 3.

2.1.1 Common Characteristics

The following characteristics may be used to describe most TMM measurement approaches:

- Knowledge Structures – Knowledge structures are composed of *concepts* and the *relationships* among them and form the theoretical basis of TMM measurements.

- Concept selection – Concept selection describes the process to select the concepts that define the content of the team mental model being measured.

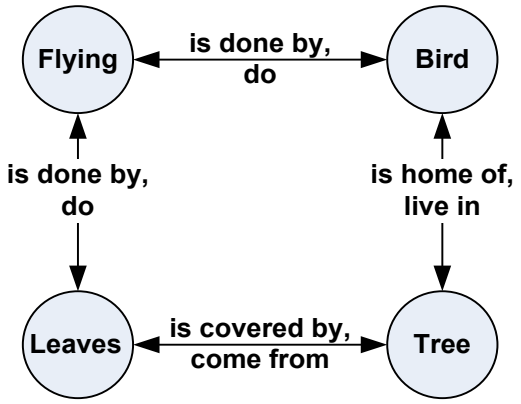
- Unit of Analysis – The unit of analysis describes the entity that the measurement is intended to represent, which may be the entire team, portions of the team, or specific concepts.

- Methods for Elicitation, Structure Representation, and Aggregation – The measurement process is conducted in three stages: *elicitation*, *structure representation*, and *aggregation*. Measurement approaches are largely defined by the methods used for each stage. (Terminology for *technique* and *method* are inconsistent in the literature. I refer to the activities for each stage as a method and a complete, well-defined process of measurement as a technique.)

Each of these characteristics is described in greater detail in the following subsections.

2.1.1.1 Knowledge Structures

Knowledge structures consist of *concepts,* which are the ideas known or believed to be true, and the *relationships* between those concepts. It is the relationships between concepts that determine how the knowledge is mentally organized and that give the knowledge its structure. Concepts can be anything that a person has knowledge of. For example, Cooke (1990) investigated the relationships among 20 naturalistic concepts, identified by single word labels, including the four concepts *Flying*, *Bird*, *Tree* and *Leaves*. The relationships she found among these concepts are shown visually in Figure 2.1. The concepts are displayed as nodes of a graph and the relationships are displayed as edges connecting the nodes. In this example, the relationships are all clearly labeled but it is possible to relate concepts without being able to verbalize the relationship.



(Representing data from Cooke (1990))

Figure 2.1. Example Knowledge Structure

2.1.1.2 Concept Selection

Although many kinds of knowledge may contribute to coordinated action, TMM research assumes that two types of mental models are particularly relevant for team performance: *taskwork* and *teamwork* (Mohammed, Ferzandi, and Hamilton 2010; Cannon-Bowers, Salas, and Converse 1993). Taskwork refers to work oriented concepts such as tasks, goals, performance requirements, and work standards. Teamwork refers to team member oriented concepts such as team members' knowledge, skills, habits, and the team's preferred methods of interacting. Taskwork and teamwork measures, independently, have been found to be predictive of team performance (Cooke, Kiekel, and Helm 2001b; Lim and Klein 2006). Therefore, the selection of concepts to investigate should be informed by an analysis of the relevant taskwork and teamwork being studied.

An important consideration for the process of selecting concepts to investigate is the potential for bias due to the researcher's involvement (Mohammed, Ferzandi, and Hamilton 2010). By identifying candidate concepts, identifying relationships of interest, defining comparison scales, and/or interacting with participants in the selection process, researchers can have considerable influence on the resulting content and structure. Any technique should acknowledge this risk and take steps to minimize researcher influence. Although this threat is frequently cited, no general approaches to address it were found in the literature review.

2.1.1.3 Unit of Analysis

It is possible to aggregate the gathered data in different ways to describe different things. The most common units of analysis are teams, individuals, and concepts.

A measure may describe the overall shared understanding of all of the members of a team. In this case, measures of different teams may be compared (a between-teams comparison.) For example, if team A has a higher team similarity than team B, team A would be expected to outperform team B. This type of comparison has been the focus of TMM research.

A measure may describe the degree to which the understanding of one subgroup or individual is shared with another subgroup in the same team (a within-teams comparison of team members.) For example, the measure of a team member may be compared to an average of the other members. A large difference may suggest this individual will have difficulty working with his or her teammates.

Finally, a measure may describe the degree to which the understanding of one concept is shared within one team. This allows comparing the measures of different concepts (a within-teams comparison of concepts.) For example, if concept A has a higher measure than concept B, team members would be expected to make more similar predictions about concept A than concept B.

2.1.1.4 Methods of Elicitation, Structure Representation, and Aggregation

Conducting a TMM measurement is a multi-stage process. First, it is necessary to elicit knowledge content from individual team members. Second, if the elicitation

method does not naturally provide structure, it is necessary to analyze the data collected to identify its structure. This produces a representation of knowledge structure for each individual. Finally, it is necessary to aggregate these knowledge structures to compute a measurement. This process (elicitation, structure representation, aggregation) is shown in Figure 2.2. The methods for each stage are interdependent. The method of elicitation dictates the possible methods of structure representation which, in turn, dictate the possible methods of aggregation.

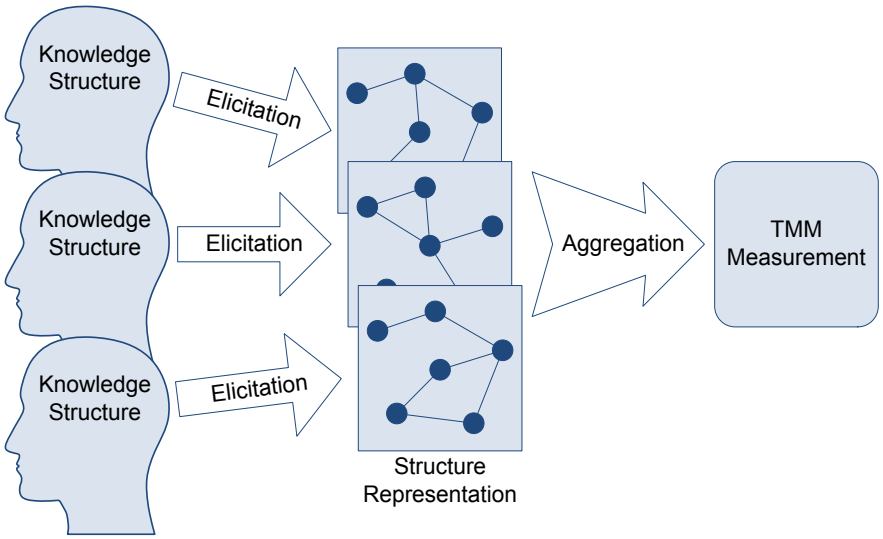


Figure 2.2. Stages of TMM Measurement

Common elicitation methods include *questionnaires*, *concept mapping*, *paired comparisons*, and *card sorts*. These are discussed next.

2.1.2 Elicitation Methods

The following sections describe 4 elicitation methods from the TMM literature that were considered as a starting point for measuring shared understanding in software design teams. Each subsection briefly describes the method, how it treats structure,

how individual structure representations may be aggregated, and some of the relative advantages and disadvantages of each.

2.1.2.1 Questionnaires

The use of questionnaires to measure TMMs is straightforward. Researchers create statements about teamwork or taskwork that are rated by the team members on a Likert-type scale indicating their agreement, usually from “strongly disagree” to “strongly agree”. For example, Espinosa and colleagues included this statement as part of the elicitation of taskwork: “I was never certain whether the code written for this [modification request] would need further rework” (Espinosa et al. 2002).

For questionnaires, aggregation is a measure of agreement, such as inter-rater reliability, or consistency, such as intraclass correlation (DeChurch and Mesmer-Magnus 2010; Cooke et al. 2000).

Questionnaires are easy to administer and are supported by established analysis methods. Although all elicitation methods are subject to researcher influence, the free-form nature of questionnaires makes them potentially more sensitive to researcher influence (through question selection and phrasing) than other methods.

There is disagreement in the literature about whether questionnaires are a valid method for measuring team mental models because they are not structure based. Mohammed and colleagues state “Because a distinguishing feature of TMMs is the measurement of both content and structure, […] questionnaires/scenarios are

therefore regarded as elicitation tools but not TMM measurement techniques in their own right" (Mohammed, Ferzandi, and Hamilton 2010).

Another concern with questionnaires is that a number of published studies do not ask about knowledge content directly. Instead, they ask team members for their perceptions of their work or their perceptions of others in their team. For example, one study, reviewed in more detail later, asked participants to rate their agreement with the statement "Team members had a common understanding of the application domain that the system was to support" (He, Butler, and King 2007, 280). This is essentially asking team members whether they have experienced breakdowns and whether they attribute them to differences in understanding. This approach does not reveal conflicts that have not already resulted in breakdowns and relies on participants to properly attribute the cause of those breakdowns.

2.1.2.2 Concept Mapping

In concept mapping, participants organize concepts as box and line drawings where boxes represent concepts and lines represent relationships between concepts. Concepts may be selected by the participants or the researchers. Although it is possible to allow each participant to provide their own concepts, this is rarely done because of the difficulty of comparing the resulting maps (Mohammed, Klimoski, and Rentsch 2000). Therefore, concepts are often pre-selected by the researcher. An advantage of concept mapping is that it captures structure directly. Aggregation is performed by comparing the similarity of boxes and lines between maps. Although this is often done with a custom scoring system (DeChurch and Mesmer-Magnus

2010), Johnson and O'Connor (2008) have proposed a standard method, the Analysis-Constructed Shared Mental Model. This is a qualitative, interpretive process of identifying explicit and implicit links, higher order clusters of concepts, and the degree of similarity between individual maps.

Due to their natural structure, concept maps are considered to have high face validity (Mohammed, Ferzandi, and Hamilton 2010). However, they require that participants be trained in the method, can be time consuming to complete, and depend upon the participants' proficiency in mapping. Their two dimensional nature may constrain the representation of complex relationships (DeChurch and Mesmer-Magnus 2010).

2.1.2.3 Paired Comparison Ratings

In this elicitation method, concepts are preselected by the researcher and then each team member rates the relatedness of all possible pairs of concepts on a Likert-type scale from "Least Related" to "Most Related". Relationships are usually considered bidirectional giving $n(n-1)\ /\ 2$ ratings for *n* concepts. Ratings can be compared directly for consistency using the Quadratic Assignment Procedure (Mohammed, Klimoski, and Rentsch 2000; Mathieu, Goodwin, and Heffner 2000). More commonly they are analyzed for structure using Pathfinder or Multidimensional Scaling (MDS) and these resulting structures are compared to provide a measure of similarity.

The Pathfinder algorithm transforms the set of paired comparison ratings from one individual into a network in which the nodes represent concepts and the links between nodes represent the salient relationships between concepts (Mohammed, Klimoski,

and Rentsch 2000). In the Pathfinder network, the primary information is the presence or absence of a link (Langan-Fox, Code, and Langfield-Smith 2000). Aggregation is performed by mathematical comparison of structure similarities between networks. Pathfinder is described in detail in section 3.3. Although Pathfinder identifies links, it does not describe them. Additional methods are available to interpret the resulting links (Cooke 1990).

Unlike Pathfinder, which is based on relationships, MDS attempts to identify criteria that individuals use to discriminate concepts and is intended to give insight into how people make judgments (Langan-Fox, Code, and Langfield-Smith 2000). MDS organizes the set of paired comparison ratings from an individual into a multidimensional space defined by the researcher such that concepts rated as similar are closer to each other and concepts rated as different are more distant. The axes then represent discriminating criteria. Measures of similarity may be calculated with matrix fitting (Mohammed, Klimoski, and Rentsch 2000). MDS does not interpret the discriminating criteria by itself, although team members are often able to provide interpretation when viewing contrasting clusters of concepts.

Paired comparisons are easy to administer. The most common method is by computer-based questionnaire but they can also be completed in paper-based response grids. Because the number of responses increases with the square of the number of concepts, it becomes fatiguing for a large number of concepts (Mohammed, Klimoski, and Rentsch 2000).

An advantage of structure representation by Pathfinder and MDS is that they can reveal aspects of the team's knowledge that the team members may not be explicitly aware of. Pathfinder reveals relationships and structure among concepts whereas MDS identifies clusters of concepts and discriminating criteria.

Elicitation by paired comparisons combined with structure representation by Pathfinder analysis have been found to be most predictive of team processes in a meta-analysis by DeChurch and Mesmer-Magnus (2010).

2.1.2.4 Card Sorting

In card sorting, concepts are preselected by the researchers and then grouped by team members into piles according to their perceived relationships (Langan-Fox, Code, and Langfield-Smith 2000). Team members then label each pile to identify the relationship the cards share. The sort can be quantified by forming a similarity matrix. For n concepts, this is an n by n matrix with rows and columns labeled by the concepts. The similarity matrix is populated by placing a 1 at the intersection of cards in the same pile, 0 otherwise. Similarity can be measured by correlation between matrices.

Card sorting is easy to administer and has good face validity (Langan-Fox, Code, and Langfield-Smith 2000) although it can become time consuming for large numbers of concepts.

2.2 Prior Research Measuring Shared Understanding in Software Engineering

I performed a literature search to find prior work identifying specific differences in understanding in the field of software engineering. I wanted to understand the techniques already tried and learn about their advantages and disadvantages. I only found three studies that were directly applicable, all of which are either based on the team mental models literature or use the same elicitation methods previously reviewed. However, none of these three examine software design. The following subsections describe these three studies in some detail. Each section summarizes the method, reviews its advantages and disadvantages, and discusses its applicability to measuring shared understanding in team design. An additional section describes other related work measuring team mental models in software teams.

In addition to the three studies described in the following subsections, the most comprehensive theoretical analysis of shared understanding in software engineering was developed by Aranda in his doctoral dissertation (Aranda 2010). Aranda examines the organizational and social environment for commercial software development and the difficulties of communication and coordination in these organizations. According to Aranda's theory, to coordinate effectively, teams must develop a shared understanding of their goals, plans, status, and context. The effectiveness of the team in developing this shared understanding is influenced by the way the team interacts and the organizational culture. Aranda's theory is primarily about organizational communication and does not address the issue of measuring shared understanding, although he does discuss this as part of the need to empirically

test his theory. In describing the difficulty of developing measures of shared understanding, Aranda refers to an unpublished manuscript by Easterbrook, saying:

> A second challenge is measuring, or even assessing, a concept such as "shared understanding." Easterbrook (1994) has pointed out the near-impossibility of assessing that two people have enough of a shared understanding: we can see in events of breakdown that their understanding has not been fully shared, but the absence of a breakdown does not imply a full shared understanding. (Aranda 2010, 149)

A team that is not experiencing breakdowns in their daily work activities may still have many latent conflicts. A valid technique for measuring shared understanding should, in some way, identify the underlying conflicts in understanding that have the potential to produce future coordination breakdowns. Finding a measurement technique to identify these latent conflicts is the goal of my research.

2.2.1 A Prior Study in Software Usability

Mancuso and colleagues (2011) used measurements of TMMs to evaluate the usability of a software program. Using TMM measurement techniques, they compared the task related knowledge structures of novice and expert users. They assumed that differences between novices and experts would indicate aspects of the user interface that may be misunderstood by novice users or that may require additional support to be used effectively. Six concepts were identified by a task analysis. They collected paired comparisons of these 6 concepts for 72 three-person teams of novice users and one three-person expert team. Ratings of relationships were

evaluated individually using one-sample *t*-tests. The results identified 4 relationships understood differently between novices and experts based on the *t* statistic and effect size. Further analysis of these four relationships identified two changes to the user interface that may improve program usability for novice users. This preliminary research did not report on the effectiveness of the proposed changes. They did not report on limitations of the technique or have an independent measure to establish validity.

Many aspects of this technique are relevant to the problem of measuring shared understanding in software design. The approach is based on team cognition and TMMs. The goal of the technique was to identify specific differences in understanding that the users were not able to verbalize, to interpret those differences, and to apply them to improve the software. This technique was different from software design understanding in that it was a between-teams comparison of novices and experts.

This study did not specifically report on the advantages and disadvantages of the technique, but it may be inferred that the paired comparisons were easy to administer and quantitatively analyze. The approach of using independent *t*-tests is not structure based and not supported by prior TMM research. The process for translating the relationships that were rated differently to specific user interface modifications depends on user interface specialists to interpret the relationships in the context of the user's tasks and hypothesize about possible related changes.

2.2.2 A Prior Study in Requirements Understanding

As part of his doctoral research, Kudikyala (2004) used analysis of TMMs to identify software requirements that are understood differently between the users specifying the requirements and the software developers implementing those requirements. In this study, concepts were requirements selected through an analysis of the systems requirement document. The primary elicitation method was a card sort. Index cards were labeled with the concepts and sorted individually by the users and the developers. A card could be duplicated if the participant felt it belonged in multiple piles. In subsequent trials, computer-based categorization replaced physical card sorts. Results of the card sort were used to create a similarity matrix for each individual. An aggregate similarity matrix for each group (users and developers) was created by adding all of the group's individual similarity matrices. Each group similarity matrix was converted to a network by the Pathfinder algorithm. These networks were quantitatively compared by correlation. Finally, requirements were ranked by their correlation coefficients. Requirements that were understood differently were identified by comparing the individual correlation coefficients to the overall correlation between the two groups.

Validation was performed by group interview. Users were interviewed about their satisfaction with the implementation of the requirements and developers were interviewed about their perceived understanding of the requirements. The analysis showed that requirements with low correlation coefficients were described as poorly understood by developers and had low satisfaction by the users. Requirements with

high correlation coefficients were perceived to be understood by the developers and had high satisfaction with the users (Kudikyala 2004).

The study reports on some of the advantages and disadvantages of the technique. Although the initial card sort was difficult to administer, the computer-based categorization was much easier. The quantitative analysis was easy to automate and the results were able to be interpreted by the team. Visual inspection of the network structures was also helpful for interpretation. The author noted that the technique produces only an indicator that a requirement may be misunderstood and that this must be confirmed using an alternate technique, such as user/developer discussions.

This technique is relevant to the current problem because it is based on TMMs, identifies specific requirements that are understood differently, and can identify differences in understanding that the team members are not aware of. The results are specific and interpretable by the team members. The application is different because it is used to evaluate differences between teams rather than within a team, but the overall approach is very similar to the type of measurement technique I am seeking.

2.2.3 A Prior Study in Software Team Research

A third technique, although intended as a research tool only, is also relevant. Scozzi and colleagues (2008) investigated shared understanding among open source software developers working in virtual teams. They used a specific version of concept maps where each link represents a directed cause/effect relationship. This type of map is known as a causal cognitive map. They investigated aspects of team interaction such as team processes, social norms, and organizational norms. Raw data for the concepts

and causal links were elicited by semi-structured interviews. These were then coded for relevant concepts and causal relationships. This analysis was used to create the causal cognitive maps. Maps were compared using mixed quantitative and qualitative methods. Using this technique, the researchers were able to identify specific conceptual topics (such as project goals) that were shared and those that were not. They found associations between concepts, team member roles, and team member tenure. This research did not have an independent measure by which to validate the technique.

This technique is relevant to shared understanding in design teams in that it compares individual knowledge structures, can identify concepts and relationships the team is not aware of, and identifies specific concepts that are understood similarly or differently. The results of this technique are not interpretable by the team and, although the team performs software design, the researchers were primarily investigating social and organizational aspects of team software development.

Evaluation of the technique was an explicit goal of this research. Among the advantages reported are the ability to visualize relationships and identify knowledge the team is unaware of (Scozzi et al. 2008).

As disadvantages, they noted two issues previously mentioned: researcher influence on results and the difficulty of comparing concepts selected by different participants (Scozzi et al. 2008). In particular, they note the influence that the researcher's knowledge scheme has on how questions are worded and that this affects the team's answers.

The researchers stated that they intended to refine and validate the technique in future work but no published results were found.

2.2.4 Other Notable Research Measuring Team Mental Models

The intent of the previous three studies was to develop a general purpose measurement technique to reveal concept-level differences in understanding. No other similar research within the domain of software engineering was found. However, a handful of other studies have used TMM measurement techniques to investigate factors influencing the development of shared understanding in software development or the association between shared understanding and some other measure, such as software quality. These studies, summarized in Table 2.1, adapt techniques used to measure TMMs and use them to investigate the association between some TMM measure and other variables. The focus of the following review is on the techniques used for measurement and how these studies can inform the development of a measure of shared understanding for software team design.

Four studies investigate the factors influencing the development of shared understanding or the changes in shared understanding over time. Levesque, Wilson, and Wholey (2001) surveyed student teams to investigate the association between shared understanding, team interaction, and role specialization. He, Butler, and King (2007) used surveys to examine pre-existing team characteristics and team interaction as factors affecting the development of task knowledge structures in undergraduate student software development teams. These two studies compared repeated team measures over time. Hsu and colleagues (2011) surveyed industrial software

Table 2.1. Summary of Other Research

Author (year)	Topic	Participants	Method	Design
Levesque, Wilson, and Wholey (2001)	Changes in shared understanding over time.	Undergraduate students.	Questionnaire, inter-rater reliability (r_{WG}).	Repeated measures (team.)
He, Butler, and King (2007)	Impact of communication on the development of task TMM.	Undergraduate students.	Questionnaire, multi-level models of change.	Repeated measures (team.)
Hsu et al. (2011)	Impact of team building on development of TMM.	Industrial teams.	Questionnaire, Partial least squares regression	Between teams.
Bibi (2010)	Effect of team organization on shared understanding of requirements.	Global distributed student teams.	Concept maps, qualitative, quantitative.	Repeated measures (team) and between teams.
Espinosa et al. (2002)	Association between TMM and coordination.	Industrial teams.	Questionnaire, inter-rater reliability (r_{WG}).	Between teams.
Balijepally, Nerur, Mahapatra (2012)	Novice/expert differences and software quality.	Undergraduate and graduate students.	Paired comparisons, QAP.	Between individuals.
Cooke, Schvaneveldt (1988)	Novice/expert differences and understanding of programming concepts.	Undergraduate, graduate, and faculty.	Paired comparisons, Pathfinder, path correlation.	Between groups.

development teams to investigate the impact of team building activities on the development of shared understanding and its association with the team's information utilization. The fourth study, by Bibi (2010), used a controlled experiment with student teams to investigate the impact of team organization and communication on the development of shared understanding of requirements. This study used the Analysis-Constructed Shared Mental Model concept mapping technique (Johnson and O'Connor 2008).

One study investigated the association between team mental model similarity and performance. Espinosa and colleagues (2002) used questionnaires to investigate the association of task and team mental models with coordination during maintenance activities.

The final two studies in Table 2.1 do not investigate team collaboration. Instead, they investigate differences between novice and expert software developers. Balijepally, Nerur, Mahapatra (2012) used paired comparisons and QAP to measure differences in mental models between novice and expert developers and the association between these differences and software quality. The last study, by Cooke and Schvaneveldt (1988), actually predates the development of the team mental models construct but is notable because one of the authors, Roger Schvaneveldt, is one of the developers of the Pathfinder algorithm and it was a goal of this study to validate Pathfinder as a valid method for identifying knowledge structures. Undergraduate students, graduate students, and faculty in psychology and computer science were grouped into 4 groups according to programming experience. These groups were analyzed for within group and between group correlations. The groups studied were not teams and, although

individual scores were averaged to form group level scores, the goal was not to form a team-level aggregate measure. Instead, the goal was "to reduce noise caused by spurious judgements" (Cooke and Schvaneveldt 1988). The relatedness ratings were analyzed with Pathfinder and compared by path length correlations. They found strong correlations within groups, and significant differences between groups, supporting the idea that novices and experts have different knowledge structures and Pathfinder as a valid method for measuring those differences.

Although none of the methods used in the above studies are directly applicable to measuring shared understanding in software design teams, it is instructive to compare the various methods. Questionnaires are popular, probably because they are easy to administer to a large number of participants. However, none of these questionnaires elicited knowledge concepts. Instead, they asked for participant perceptions. The analysis techniques used in these questionnaires found general statistical associations and did not describe individuals or teams.

In contrast to the questionnaires, the mixed methods concept mapping technique used by Bibi (2010) was complex and time consuming. Although this approach included concept-level analysis, it focused on between-groups measures.

The study by Balijepally (2012) is relevant because the concepts investigated were derived from the software design, such as class names and method names. In this sense, this study was a measure of the knowledge structures related to software design. By comparing measures from novices and experts, they validated that novices

and experts understand the same software design differently. However, they did not investigate issues of team collaboration.

This chapter reviewed general approaches to measure team mental models, described some specific methods and their advantages and disadvantages, and reviewed some specific implementations of those methods in software engineering. I review these advantages and disadvantages more formally in the next chapter, but some preliminary conclusions can be drawn:

- Questionnaires, although easy to administer, are difficult to construct and are less likely to be valid than other methods.
- Concept mapping, while valid, can be cumbersome to implement.
- Paired-comparisons and card sorts are efficient to administer, avoid some bias issues, and are considered valid.
- Methods of structure representation that are easily visualized, such as Pathfinder, aid in result interpretation.

Chapter 3: Development of the Measure

This chapter describes the Shared Understanding Measure for Software Teams including the rationale for its development and the details of the measure. The original goals, developed in Chapter 1, were to develop a technique to measure shared understanding in the context of a software design team that would elicit differences in understanding that the team is unaware of, identify specific areas of misunderstanding, and make the results interpretable by the team. What is presented here is the end result of an iterative process of literature review, refining the criteria, refining the technique, and filling gaps between the criteria and the expected performance of the measurement technique developed. Section 3.1 summarizes the guiding criteria. Section 3.2 describes the process of applying these criteria to the existing TMM literature to identify a starting point. Section 3.3 describes the details of this starting point without changes. Section 3.4 then describes the full SUM-ST with changes to extend it and adapt it to meet the original goals.

This chapter presents the original technique that was evaluated empirically in the pilot and main studies. There were refinements made after the pilot study (described in section 4.2.4.4) and additional changes proposed after the main study (described in section 6.4) that are not incorporated into this chapter.

3.1 Guiding Criteria

Because the prior measurement techniques described in the TMM literature are so diverse, I followed a structured analysis to select the most appropriate starting point. This analysis began with a definition of desirable properties.

The first consideration was the unit of analysis (section 2.1.1.3.) It is possible to form a measurement of the whole team (for between-teams comparisons), the team members (for within-teams comparisons of team members), or the concepts (for within-teams comparisons of concepts.) Because each software development project is unique, I saw no way to make a meaningful between-teams comparison. Within-teams comparisons of team members or concepts may each have value, although a comparison of concepts is more specific and more likely to be actionable to improve the software design.

For research purposes, a team-level measurement may help track changes in shared understanding over time. Measures could be taken over short periods of time, such as before and after a team meeting, or over longer periods of time, such as after each iteration of development (for a team following an iterative process model.)

For the team to use this technique to improve their design process, the measurement should identify, or at least suggest, specific design elements that may contribute to future breakdowns. This means that, in addition to a team-level measure for repeated measures comparisons, the measurement should also produce concept-level measurements that can identify individual concepts that may be misunderstood. Additionally, the results of the measurement should be interpretable by the software development team (and not only a researcher trained in measurement interpretation.)

To complement existing software artifact verification techniques, the measurement should be sensitive to differences in understanding the team may not be aware of.

Because practitioners always work under time pressure, the elicitation technique should place as little burden on the software developers as possible.

Finally, there should be empirical evidence that the measurement technique is a valid measure of shared understanding, such as a positive correlation with a measure of team performance.

3.2 Selection of a Starting Technique

Of the four general elicitation approaches reviewed earlier (questionnaires, concept mapping, paired-comparisons, card sorts), which is a good candidate?

Questionnaires are generally not a good choice because they are not sensitive to knowledge structures people are unaware of, they are not structure based, and they are particularly sensitive to researcher influence.

Concept mapping has advantages in terms of validity and capturing structure. But it requires that the developers be trained in the technique, which is burdensome. Also, the visual similarity between concept maps and software development diagramming techniques (such as UML class diagrams) may lead to confusion. For example, when asked to draw a concept map, the developer may be tempted to draw a visually related, but conceptually different, diagram. Therefore, concept mapping is not a good fit.

Card sorts and paired comparisons are both appropriate elicitation techniques. They require little developer training and can be easily administered by computer. However, the analysis techniques available for paired comparisons are more robust

than for card sorts. In particular, Pathfinder analysis produces network diagrams that can be visually interpreted as well as being quantitatively compared. Also, paired comparisons, combined with Pathfinder analysis, have been found to be most predictive of team processes (DeChurch and Mesmer-Magnus 2010).

One significant deficiency of paired comparisons/Pathfinder is that this approach does not have any standard method of producing concept-level measurements. However, in some implementations, the team-level measurements are derived from internal concept-level calculations, suggesting a path to extending this approach.

Another limitation of paired comparisons is that the technique becomes fatiguing for a large number of concepts. Applications of paired comparisons typically investigate 15 to 30 concepts (Mohammed, Klimoski, and Rentsch 2000). Although this is sufficient to develop valid cognitive structures, this is a comparatively small number of concepts compared to the total number of possible design concepts in a software system. Any use of paired comparisons will require additional guidance on how to select appropriate concepts for analysis.

Based on the above analysis, I selected paired comparisons with Pathfinder analysis as a starting point for measuring shared understanding in software team design. The next section describes these in detail.

3.3 Paired Comparisons and Pathfinder

This section describes the measurement of TMMs using paired comparisons and Pathfinder. In order to make a measurement with this approach, it is first necessary to

select the concepts that will be investigated. Then, team members are asked to rate the relatedness of each pair of concepts. These ratings are analyzed with Pathfinder to develop a network structure for each person. Finally, these network structures are compared to yield a similarity measurement. The following sections describe these steps in detail.

3.3.1 Concept Selection

Concepts may be obtained from text analysis of related documentation, interviewing subject matter experts, or through a task analysis of the team's activities (Mohammed, Klimoski, and Rentsch 2000). Concepts are commonly one or two word descriptive terms (such as "character data", "operator") (Cooke and Schvaneveldt 1988; Schvaneveldt et al. 1985) or short phrases (such as "change trajectory of ship", "Tactical UAV Coordinator") (Edwards et al. 2006; Shope et al. 2004). Consideration should be given to the fact that different terms may have different meanings to different team members and may be interpreted differently in different contexts (Mohammed, Klimoski, and Rentsch 2000).

3.3.2 Paired Comparison Ratings

Participants rate the relatedness of each pair of comparisons by computer or on paper grids. The relationship is usually considered to be bidirectional, which requires $n(n-1)/2$ comparisons for n concepts, although Mohammed and colleagues (2000) report that there has been some work that collects only a subset of all possible pairs. Participants can rate approximately 10 pairs of concepts per minute (Mohammed, Klimoski, and Rentsch 2000; Langan-Fox, Code, and Langfield-Smith 2000). At this

rate, a set of 15 concepts (requiring 105 comparisons) would take approximately 11 minutes and a set of 30 concepts (requiring 435 comparisons) would take approximately 44 minutes. Responses are rated on 5, 7, or 9 point Likert-type rating scales. Participants should be instructed to make "quick, intuitive judgments" (Mohammed, Klimoski, and Rentsch 2000) of the similarities between the pair of concepts.

The result of the paired comparison ratings is a triangular matrix of dimension equal to the number of concepts. This matrix contains the relatedness value for each pair of concepts. An example, inspired by Cooke (1990), is shown in Table 3.1.

Table 3.1. Example Relatedness Ratings

	Flying	Bird	Tree	Leaves
Flying	-			
Bird	7	-		
Tree	1	5	-	
Leaves	2	3	6	-

3.3.3 Pathfinder Network Scaling

The Pathfinder algorithm converts a matrix of distance values to a mathematical graph called a Pathfinder network or *PFNet* (Dearholt and Schvaneveldt 1990). (The terms *graph* and *network* are equivalent in this discussion.) In this graph, each node represents a concept and each edge represents a relationship between concepts. Pathfinder builds structure from the strongest relationships that connect the concepts. To create the PFNet, Pathfinder starts with a fully connected graph where each edge is given a weight derived from the relatedness rating for the pair of concepts that the edge connects. Pathfinder expects that edge weights represent distances between

concepts and that more similar concepts have smaller edge weights. Therefore, the relatedness matrix is converted to an edge weight matrix by subtracting each relatedness value from the maximum possible. The edge weight matrix corresponding to Table 3.1 is created by subtracting each value from 7 and is shown in Table 3.2.

Table 3.2. Edge Weight Matrix Corresponding to Table 3.1

	Flying	Bird	Tree	Leaves
Flying	-			
Bird	0	-		
Tree	6	2	-	
Leaves	5	4	1	-

The left-hand graph in Figure 3.1 is the fully connected graph corresponding to the edge weight matrix in Table 3.2. The right-hand graph in Figure 3.1 is the corresponding PFNet as computed by Pathfinder.

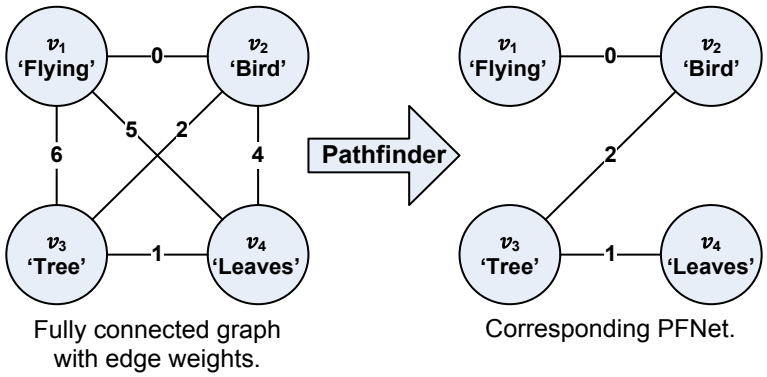


Figure 3.1. Full Graph and PFNet for the Data in Table 3.1

Pathfinder works by computing the shortest path(s) between all pairs of nodes and eliminating edges that are not on any shortest path. Pathfinder supports two parameters: r and q. The parameter r is based on the Minkowski distance and determines how the path length is computed from the edge weights along the path.

The parameter q limits the paths considered to those containing q or fewer edges. Smaller values for r and q generally result in PFNets with more edges. It is common to use values of $r = \infty$, meaning that the length of a path is calculated as the maximum edge weight along the path (and not the sum, as may be expected). Similarly, q is commonly set to the maximum value, the number of concepts minus one (n-1), meaning that all possible paths are considered. These choices require only ordinal assumptions about ratings (Cooke, Kiekel, and Helm 2001b; Cooke and Schvaneveldt 1988; Schvaneveldt, Durso, and Dearholt 1989). Using these settings for r and q in Figure 3.1, the path (v_1,v_2,v_3,v_4) has edge weights (0, 2, 1) and length 2. This is the shortest path between v_1 and v_4. The edges $\{v_1,v_3\}$ and $\{v_2,v_4\}$ are not on any shortest path and are removed. The computed PFNet in Figure 3.1 represents the knowledge structure for the ratings given in Table 3.1.

It is possible for there to be more than one path between two nodes that have the same shortest distance. In this case, Pathfinder preserves them all.

The Pathfinder algorithm described by Dearholt and Schvaneveldt (1990) has computational complexity of $O(n^4)$. This is not generally a problem for the thirty or so concepts investigated in this application. (Quirin and colleagues (2008) have published an optimized algorithm for the conditions of $r = \infty$ and $q = (n\text{-}1)$.)

3.3.4 Graph Comparison

With Pathfinder, aggregation is a measure of similarity between the PFNets for members of the team. As discussed by Goldsmith and Davenport (1990), there are multiple ways to interpret structural similarity between two graphs, each giving a

different quantitative calculation. It is assumed that the two graphs being compared contain the same nodes, therefore, the measure of similarity is a measure of differences in the arrangement of edges between the two graphs. Figure 3.2 shows two PFNets that are used as an example in the following discussion.

Goldsmith and colleagues (1991) define two general approaches to comparing graphs: *neighborhood* comparisons and *distance* comparisons. A third approach, by Schvaneveldt (2013), is based on a direct comparison of the corresponding edges in the two graphs. Goldsmith and Davenport (1990) state that different comparisons measure different properties and the correct measure depends on the particular application.

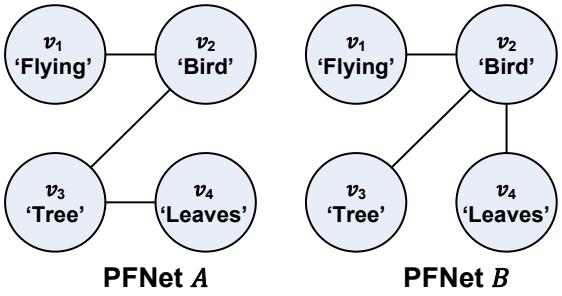


Figure 3.2. PFNets Differing in Edges About Node v_4 'Leaves'

3.3.4.1 Neighborhood Comparison (C)

C (for closeness) is a neighborhood comparison approach (Goldsmith, Johnson, and Acton 1991). The neighborhood of a node consists of all nodes directly connected to that node by an edge. The similarity of two graphs with respect to a single node is measured as the number of neighbors in common divided by the total number of neighbors. The C measure is the average similarity over all nodes.

Defined mathematically, following Goldsmith and Davenport (1990, 83), a graph $G = (V, E)$ is a finite set V of n nodes and a set E of edges such that $E \subseteq V \times V$. The graph distance between the nodes v and v' in graph G, denoted $\delta_G(v, v')$ is the minimum number of edges in any path between v and v' (provided such a path exists.) The neighborhood of v in graph G, denoted by G_v, is the set of nodes $v' \subseteq V$ such that $\delta_G(v, v') = 1$. Given two undirected graphs A and B with common node set V:

$$C(A, B) = \frac{1}{n} \sum_{v \in V} \frac{|A_v \cap B_v|}{|A_v \cup B_v|} \tag{3.1}$$

In the case where the dominator is zero (because neither node has any neighbors), that summand is defined to be 1. C has a range of 0 (least similar) to 1 (identical graphs.)

To compare the PFNets in Figure 3.2 using C, first compare each corresponding node as shown in Table 3.3.

Table 3.3. C Node Calculations for Figure 3.2

Node	Label	Neighbors in Common	All Neighbors	Node Similarity
v_1	Flying	$\{v_2\}$	$\{v_2\}$	1
v_2	Bird	$\{v_1, v_3\}$	$\{v_1, v_3, v_4\}$	$^2/_3$
v_3	Tree	$\{v_2\}$	$\{v_2, v_4\}$	$^1/_2$
v_4	Leaves	$\{\}$	$\{v_2, v_3\}$	0

The value of C, calculated by equation (3.1), is the average node similarity, which is $\frac{13}{24} \cong 0.5417$.

For teams larger than two, the team similarity is calculated as the average similarity of all possible pairs of team members (Langan-Fox, Code, and Langfield-Smith 2000). Let $G_1, G_2, G_3 \dots G_n$ be the PFNets for n team members, $n \geq 2$. The average team similarity as measured by C, denoted $\overline{C}$ is defined as follows:

$$\overline{C}(G_1 \dots G_n) = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=(i+1)}^{n} C(G_i, G_j) \tag{3.2}$$

3.3.4.2 Distance Comparison (*r*)

r (for Pearson's correlation coefficient) is a distance comparison approach (Goldsmith, Johnson, and Acton 1991). This is calculated by first calculating the graph distance between all pairs of nodes in each PFNet and then calculating the correlation between corresponding paths. The distance between each pair of nodes in one graph is most conveniently represented as a triangular matrix, not including the diagonal, where each node is represented by a row and a column, and each entry is the distance between the corresponding nodes. The value of *r* is the Pearson correlation coefficient of the corresponding entries in the distance matrices for the two graphs.

Given G,V,E and δ_G as defined above, let v_i denote the numbered nodes of V such that $V = \{v_1, v_2, v_3 \dots v_n\}$. The distance matrix of distances between all pairs of nodes in graph *G* is defined as:

$$D_G = \begin{bmatrix} - & \delta_G(v_1,v_2) & \delta_G(v_1,v_3) & \delta_G(v_1,v_4) & \cdots & \delta_G(v_1,v_{n-1}) & \delta_G(v_1,v_n) \\ & - & \delta_G(v_2,v_3) & \delta_G(v_2,v_4) & \cdots & \delta_G(v_2,v_{n-1}) & \delta_G(v_2,v_n) \\ & & - & \delta_G(v_3,v_4) & \cdots & \delta_G(v_3,v_{n-1}) & \delta_G(v_3,v_n) \\ & & & - & \ddots & \delta_G(v_4,v_{n-1}) & \delta_G(v_4,v_n) \\ & & & & - & \vdots & \vdots \\ & & & & & - & \delta_G(v_{n-1},v_n) \\ & & & & & & - \end{bmatrix}$$

The value of *r* is computed as the Pearson correlation coefficient of the distance matrices for the two graphs:

$$r(A,B) = \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\left(D_{A,i,j} - \overline{D_A}\right)\left(D_{B,i,j} - \overline{D_B}\right)}{\sqrt{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\left(D_{A,i,j} - \overline{D_A}\right)^2}\sqrt{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\left(D_{B,i,j} - \overline{D_B}\right)^2}} \quad (3.3)$$

where $D_{A,i,j}$ and $D_{B,i,j}$ are the elements of the distance matrices for graphs *A* and *B*, respectively, and $\overline{D_A}$ and $\overline{D_B}$ are the mean of all the elements of the distance matrices for graphs *A* and *B*, respectively. *r* has a range of -1 (least similar) to 1 (identical graphs.)

r will be undefined if either sum of squares in the denominator is zero. This occurs if all of the distances in either graph are the same. This issue is not addressed by Goldsmith and Davenport. Kudikyala (2004 eq. 3.1) defined the measure to be zero in this case, however, this case can occur with identical graphs and graphs that are quite different, so an undefined result is more correct.

To compare the PFNets in Figure 3.2 using *r*, first calculate the distance matrix for each graph. Distance is calculated as the minimum number of edges connecting the two nodes. (This can be found with the Floyd-Warshall algorithm.) The distances for the PFNets in Figure 3.2 are shown in Tables 3.4 and 3.5.

Table 3.4. Distance Matrix for PFNet A in Figure 3.2

	v_1	v_2	v_3	v_4
v_1	-	1	2	3
v_2		-	1	2
v_3			-	1
v_4				-

Table 3.5. Distance Matrix for PFNet B in Figure 3.2

	v_1	v_2	v_3	v_4
v_1	-	1	2	2
v_2		-	1	1
v_3			-	2
v_4				-

The value of *r*, calculated by equation (3.3), is the correlation between corresponding entries in the distance matrices for PFNet *A* and PFNet *B*, which is $\frac{1}{\sqrt{5}} \cong 0.4472$.

The similarity of teams larger than two is calculated as the average similarity of all possible pairs of team members. Following equation (3.2), the average team similarity as measured by *r*, denoted $\bar{r}$, is defined as follows:

$$\bar{r}(G_1 \dots G_n) = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=(i+1)}^{n} r(G_i, G_j) \tag{3.4}$$

3.3.4.3 Edge Comparison (*NETSIM*)

Another comparison approach, *NETSIM* (for Network Similarity) (Langan-Fox, Code, and Langfield-Smith 2000), is used by the KNOT software for Pathfinder network analysis (Interlink 2014). An advantage of *NETSIM* over the previous two calculations is that the probability distribution is known, allowing statistical significance testing (Schvaneveldt 2013).

NETSIM is an edge comparison method that takes the ratio of edges in common to the total number of edges (Schvaneveldt 2013; Langan-Fox, Code, and Langfield-Smith 2000). Given two undirected graphs $A = (V, E_A)$ and $B = (V, E_B)$ as defined above:

$$NETSIM(A,B) = \frac{|E_A \cap E_B|}{|E_A \cup E_B|} \tag{3.5}$$

This has a range of 0 (least similar) to 1 (identical graphs.) *NETSIM* is undefined if neither graph has any edges.

Comparing the PFNets in Figure 3.2 using *NETSIM* requires finding edges in common and in either PFNet. The edges in common are $\{v_1,v_2\}$ and $\{v_2,v_3\}$. The edges in either PNet are $\{v_1,v_2\},\{v_2,v_3\},\{v_2,v_4\},\{v_3,v_4\}$. The number of edges in common is 2 and the number of edges in either PFNet is 4. The value of *NETSIM*, calculated by equation (3.5), is the ratio of these, which is 0.5.

The similarity of teams larger than two is calculated as the average similarity of all possible pairs of team members. Following equation (3.2), the average team similarity as measured by *NETSIM*, denoted $\overline{NETSIM}$, is defined as follows:

$$\overline{NETSIM}(G_1 \dots G_n) = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=(i+1)}^{n} NETSIM(G_i, G_j) \tag{3.6}$$

3.4 The Shared Understanding Measure for Software Teams (SUM-ST)

A number of gaps need to be filled before the measurement technique described in the previous section can be used:

- The method for concept selection must be defined.
- The method for gathering relatedness ratings must be defined in greater detail.

- A technique to calculate concept similarity must be defined.

These issues are addressed in the following sections.

3.4.1 Concept Selection

The approach to selecting the concepts must accommodate the nature of the team, the project they are working on, and the goals of the measurement. For a team with a document centric process, concepts could be selected from design elements identified through document analysis. For less document centric teams, source code analysis or team interviews may be used. Besides design elements, other concepts could be selected if their shared understanding is expected to support coordinated decisions within the team.

Concepts must be selected and labeled. To help reduce the risk of bias in this process, the following guidelines should be followed:

- Concepts should be explicit terms or short phrases used by the team in specifications, source code, or interviews. They should not be vague ideas like design principles.

- Concepts should participate in some form of collaboration (i.e. held in common by two or more team members.)

- Concepts should be "important" in some way that the team will have a working knowledge of, such as class responsibilities or user interface components.

- Concepts should be defined within the context of the project and not be pre-existing ideas that are incorporated into the project.

- The phrasing should not be so short as to be ambiguous, or so long that it includes its own definition. Participants should not be trying to learn or infer from the concept labels themselves.

- If concepts must be rephrased, they should follow the team's terminology as closely as possible to minimize researcher reinterpretation.

Be aware that different team members may use different terms for the same concept and that the same term can represent different concepts.

3.4.2 Elicitation of Relatedness Ratings

A web-based questionnaire is particularly convenient for gathering relatedness ratings. Questions are formed from all possible pairs of the selected concepts. Each question presents two concepts and asks participants to rate the degree to which they are related on a 7-point Likert scale where 0 indicates Unrelated and 6 indicates Strongly Related. The order of questions and concepts is randomized.

The longer participants consider a relationship, the more likely they may be to contrive a relationship that is not representative of their internal knowledge structure. Therefore, participants should be instructed to give their first impression rather than analyzing the pair.

Knowledge structures are known to be context dependent (Mohammed, Ferzandi, and Hamilton 2010) but there is little guidance in the literature as to how to establish the

proper context in which to elicit knowledge content. The approach taken is to have participants participate in an activity that involves reflection about the concepts that will be asked about before beginning the rating process.

3.4.3 Concept Similarity Calculation

The graph comparison calculations in section 3.3.4 compare entire PFNets and give no information about the similarity of the individual concepts that make up the PFNets. Because I am interested in identifying specific concepts that may be understood differently by different team members, a method of calculating concept similarity must be devised. Each of the three calculations, C, r, and *NETSIM*, may be altered to allow comparing individual concepts. This is done by restricting the calculation to the nodes that represent the concept of interest. These concept-level variants are presented below.

3.4.3.1 Concept-Level Comparison by Neighbors (C_v)

To calculate a measure of the shared understanding of a single concept within the team by comparing neighborhoods, it is possible to limit the C calculation to the nodes representing that concept.

Let A and B be graphs as defined in section 3.3.4 with common node set V, $v \in V$:

$$C_v(A, B, v) = \frac{|A_v \cap B_v|}{|A_v \cup B_v|} \tag{3.7}$$

In the case where the dominator is zero (because neither node has any neighbors), C_v is defined to be 1. C_v has a range of 0 (least similar) to 1 (identical neighbors.)

Following equation (3.2), the average team similarity for more than two people, as measured by C_v, denoted $\overline{C_v}$, is defined as:

$$\overline{C_v}(G_1 \dots G_n, v) = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} C_v(G_i, G_j, v) \tag{3.8}$$

where n > 2. This definition is equivalent to the approach used by Kudikyala (2004 eq. 3.3). This comparison has a range of 0 (least similar) to 1 (identical). The C_v values for Figure 3.2 are shown in Table 3.6.

Table 3.6. C_v Node Calculations for Figure 3.2

Node	Label	C_v
v_1	Flying	1
v_2	Bird	$^2/_3$
v_3	Tree	$^1/_2$
v_4	Leaves	0

3.4.3.2 Concept-Level Comparison by Distance (r_v)

To calculate a measure of the shared understanding of a single concept within the team by comparing distances, it is possible to limit the correlation in *r* to paths that start or end with the node representing that concept. While the distance matrix for *r* is two dimensional (see section 3.3.4.2), the distances required for calculating r_v can be represented as a one dimensional vector where each entry contains the distance between the node v and every other node in the graph.

Given a graph $G = (V, E)$ as defined above, the distance vector of distances between the node v and all other nodes in graph *G* is defined as:

$$D_{G,v} = (\delta_G(v, v_1), \delta_G(v, v_2), \delta_G(v, v_3), \dots\ \delta_G(v, v_n)\ )$$

$D_{G,v}$ does not include the distance between the node v and itself (always zero) and, therefore, contains n-1 elements. The concept specific comparison r_v is defined as:

$$r_v(A, B, v) = \frac{\sum_{i=1}^{n-1}\left(D_{A,v,i} - \overline{D_{A,v}}\right)\left(D_{B,v,i} - \overline{D_{B,v}}\right)}{\sqrt{\sum_{i=1}^{n-1}\left(D_{A,v,i} - \overline{D_{A,v}}\right)^2}\sqrt{\sum_{i=1}^{n-1}\left(D_{B,v,i} - \overline{D_{B,v}}\right)^2}} \tag{3.9}$$

where $D_{A,v,i}$ and $D_{B,v,i}$ are the elements of the distance vectors for node v in graphs A and B respectively, $\overline{D_{A,v}}$ and $\overline{D_{B,v}}$ are the mean of all the elements of the distance vectors for node v in graphs A and B respectively, and n is the number of nodes in V. This is equivalent to the method used by Kudikyala (2004, 43). r_v has a range of -1 (least similar) to 1 (identical graphs.)

Following equation (3.2), the average team similarity for more than two people, as measured by r_v, denoted $\overline{r_v}$, is defined as follows:

$$\overline{r_v}(G_1 \dots G_n, v) = \frac{2}{n(n-1)}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} r_v(G_i, G_j, v) \tag{3.10}$$

The r_v values for Figure 3.2 are shown in Table 3.7.

Table 3.7. r_v Node Calculations for Figure 3.2

Node	Label	$D_{A,v}$	$D_{B,v}$	r_v
v_1	Flying	(1,2,3)	(1,2,2)	$\sqrt{3}/2 \cong 0.8660$
v_2	Bird	(1,1,2)	(1,1,1)	Undefined
v_3	Tree	(2,1,1)	(2,1,2)	$1/2$
v_4	Leaves	(3,2,1)	(2,1,2)	0

3.4.3.3 Concept-Level Comparison by Edges

It is natural to consider a concept specific comparison based on the *NETSIM* edge comparison calculation. By limiting the edges to those that contain a particular node v, this calculation becomes identical to C_v. Therefore, there is no concept similarity measure based on *NETSIM*.

3.4.3.4 Summary of Graph Similarity Calculations

The similarity calculations described previously are summarized in Table 3.8.

Table 3.8. Summary of Graph Similarity Calculations

Similarity Calculation	Description	Range
$\overline{C}(G_1 \dots G_n)$	Neighborhood-based comparison of all concepts in the PFNets for *n* participants. See 3.3.4.1.	0 to 1
$\overline{C_v}(G_1 \dots G_n, v)$	Neighborhood-based comparison of the single concept v in the PFNets for *n* participants. See 3.4.3.1.	0 to 1
$\overline{r}(G_1 \dots G_n)$	Distance-based comparison of all concepts in the PFNets for *n* participants. See 3.3.4.2.	-1 to 1
$\overline{r_v}(G_1 \dots G_n, v)$	Distance-based comparison of the single concept v in the PFNets for *n* participants. See 3.4.3.2.	-1 to 1
$\overline{NETSIM}(G_1 \dots G_n)$	Edge-based comparison of all concepts in the PFNets for *n* participants. See 3.3.4.3.	0 to 1

3.4.4 Result Interpretation

The graph similarity calculations in Table 3.8 provide single numeric values calculated from the selected team members and concepts. How should these be interpreted?

Prior research has shown that the higher the similarity measurement, the greater the shared understanding of the team (DeChurch and Mesmer-Magnus 2010). This allows

making relative comparisons between teams or repeated measures of the same team. As mentioned previously, because no two software development projects are alike, it is not expected that between-teams comparisons are useful in this context. However, repeated measures of the same team over time may be valuable for tracking progress or measuring the effect of team effort to develop shared understanding.

Concept-level measures are expected to allow comparisons between concepts understood by the team and to allow comparisons between repeated measures of the same concept over time. Because the number of data points contributing to the concept-level calculations are limited, these results should guide additional techniques, such as design reviews, and not used to make independent conclusions.

In addition to the relative measures just discussed, it is natural to consider making an interpretation of the degree of shared understanding by comparing the numerical measure to a fixed threshold. For example, Kudikyala (2004) suggested the following scale (for measures in the range of 0 to 1): measures less than 0.4 were assumed to indicate little shared understanding, measures between 0.4 and 0.7 (inclusive) were assumed to indicate moderate shared understanding, and measures above 0.7 were assumed to indicate strong shared understanding. However, there is little agreement among studies that have reported scores (Stout et al. 1999; Langan-Fox, Code, and Langfield-Smith 2000), suggesting that it is premature to make conclusions based fixed thresholds.

Chapter 4: Method

To answer the research questions introduced in section 1.2, I conducted two research studies: a pilot study with student teams in the fall of 2013 and a main study with industrial teams in the spring of 2015. Because the primary purpose of the pilot study was to inform the method for the main study, the entirety of the pilot study is included in this chapter. This chapter begins with the general approach to the study design, describes the particulars of the pilot study, and then concludes with the method for the main study.

4.1 General Study Approach

The research questions developed in section 1.2 take a layered approach to investigate the validity of measurements of shared understanding and their usefulness in the software design process. RQ1 investigates the association between the SUM-ST measures and the team's shared understanding. RQ2 extends this to ask about the association between these measures and difficulties the team experiences with the software design process. Finally, RQ3 asks about the potential of the technique to improve the software design process. As this research is explanatory in nature and investigates complex human factors, a rich, qualitative approach was chosen.

The answer to RQ1 requires an independent measure of shared understanding as a point of comparison. As no standard measure exists, these studies use group interviews with the teams to explore their members' understanding of the concepts and whether their understanding was similar or dissimilar. The point of comparison,

therefore, is explicit statements by team members about their shared understanding. This approach is similar to the verification performed by Kudikyala (2004).

The approach taken to answer RQ2 was twofold. First, the ratings questionnaire was administered at two different times during the software design process. This presented the opportunity to associate changes in the measurement with difficulties experienced by the team during this period, such as a breakdown in understanding that required a design revision. The second approach was to ask the team for their opinion about the importance of differences in understanding that were identified during the group interview.

To answer RQ3, industrial software team members were asked for their assessment of the technique after having the opportunity to apply the technique to their current projects.

In order to better understand the answers in context, and to explore possible alternate explanations of the results, the group interviews were supported by a demographic questionnaire, questions about team interaction and communication, and in the main study, observations of team interaction.

The methods used in this research differ from methods used to validate TMM measurements in the Industrial and Organizational Psychology literature because of the differences in the research questions asked and the different context being studied. Validations of TMM measurements are typically operationalized as statistical associations between the TMM measurement and some measure of team performance, such as the time to complete a mission (Stout et al. 1999; Cooke,

Kiekel, and Helm 2001a). Such studies rely on having a statistically significant number of teams performing similar functions, using similar methods, and being evaluated against common criteria. In contrast, software design teams have unique goals and methods, which prevents similar study designs. Additionally, this research goes beyond statistical validation to ask how the teams would apply the measurements for process improvement, a question not addressed in the TMM validation studies.

Due to the novelty of the measurement technique and research method, a pilot study was conducted to gain experience prior to the main study. The following section describes the pilot study and the lessons learned that informed the main study method.

4.2 Pilot Study

I conducted a pilot study in the fall of 2013 with undergraduate computer science student teams working on a semester-long project. The goals of the pilot study were to gain experience applying the SUM-ST technique, practice applying the research method, and gather preliminary results to answer the research questions. The pilot study has been published previously (Braunschweig and Seaman 2014).

4.2.1 Study Purpose and Research Questions

The pilot study had a number of objectives. The first was to gain experience applying the SUM-ST technique and to understand how developers would feel about participating.

The second objective was to develop the necessary software tools to implement the technique, such as the ratings questionnaire, Pathfinder algorithm, PFNet similarity calculation, and tools to interpret the results, such as PFNet visualization.

The third objective was to identify potential usability issues with the ratings questionnaire, such as the phrasing of the instructions, the time required, and any difficulties participants had taking it, such as fatigue.

The fourth objective was to gain experience using the research method and to identify any problems with or limitations of the method.

The final objective was to resolve the open question of which concept similarity comparison calculation is most appropriate. Three different team-level comparisons were described in section 3.3.4 ($\overline{C}$, $\overline{r}$, and $\overline{NETSIM}$), and two different concept-level comparisons were developed in section 3.4.3 ($\overline{C_v}$ and $\overline{r_v}$). The pilot study compared differences in these calculations. Additionally, to better understand these structure-based comparison techniques, I contrasted them with a simple correlation of the raw ratings values. Intuitively, if two participants rate relationships similarly, their ratings would have a strong positive correlation. For this purpose (following the development in section 3.3.4), *Cor* is defined as the Pearson correlation of the paired relatedness ratings for two team members, $\overline{Cor}$ as the average *Cor* over all possible pairs of team members, and $\overline{Cor_v}$ as the Pearson correlation of just those rating pairs that include concept represented by v.

The pilot study addressed the first two research questions (originally stated in section 1.2):

> RQ1: How are similarities and differences between the software development team members' knowledge structures, as measured by SUM-ST, associated with similarities and differences in how those team members understand their design?
>
> RQ2: How are measured differences in software development team members' knowledge structures associated with identifiable difficulties in the software design process resulting in degradation of cost or quality?

RQ3, which asks about using measurements of knowledge structures to improve the software design process, was not included in the pilot study because student teams do not have the necessary experience or mature process required to be informative.

The following research question comparing graph similarity measures was added:

> RQ4: Given individual relatedness ratings captured from a team of developers about a common design, how do the graph similarity measures $\overline{C}$, $\overline{r}$, $\overline{NETSIM}$, $\overline{C_v}$, $\overline{r_v}$, and $\overline{Cor_v}$ compare?

4.2.2 Pilot Study Method

The pilot study method is presented in two parts. The first part, the study design, describes the study as originally planned and is presented separately in order to be

more easily reused by other researchers. The second part, the implementation, describes the particulars of how the study was actually carried out.

4.2.2.1 Study Design

This section describes the pilot study design, which follows the general approach described in section 4.1.

The study design consisted of a variety of data gathering activities that are summarized in Figure 4.1. Basic demographic information was collected by questionnaire. Design concepts were selected by document analysis of each team's design documents. Relatedness ratings were gathered by questionnaire under researcher observation. In addition to the paired comparisons, this questionnaire asked participants to provide descriptions of the relationships in the resulting PFNet. An exit interview was used to ask participants about their experience taking the questionnaire. Participants took the ratings questionnaire twice, once after their project design was complete and again when their software was working. Finally, each team was interviewed as a group to investigate shared understanding within the team and find associations with the measurements from the ratings questionnaires. The significant portions of this design are described in detail below.

4.2.2.1.1 Participants

Participants were undergraduate computer science students working on a semester-long project consisting of requirements analysis, system design, implementation, and user acceptance testing. Students had to produce requirements and design documentation and working software.

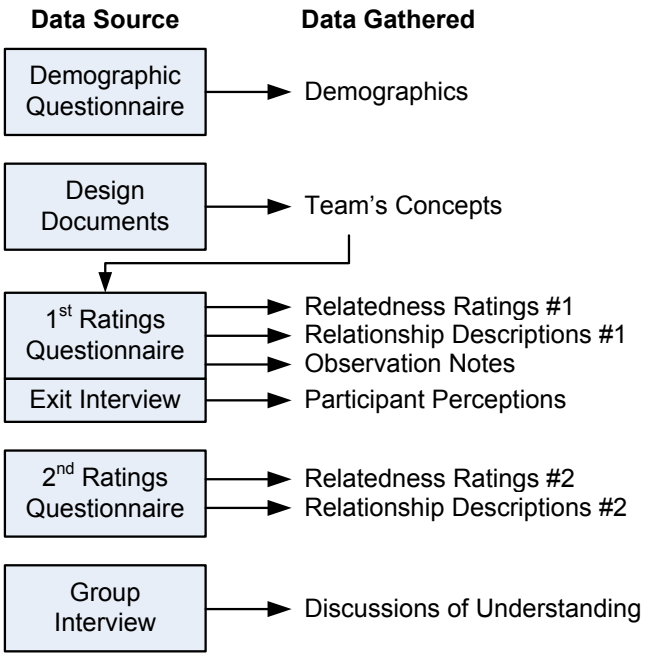


Figure 4.1. Pilot Study Design Overview

Students were offered $10 for participation in each of the rating questionnaires and the group interview ($30 total.) Students received no course credit for participating and each student decided individually whether to enroll. (Incentives based on group participation were prohibited by the UMBC Institutional Review Board.)

4.2.2.1.2 Demographic Questionnaire

Shortly after enrolling, participants were asked to take a demographic questionnaire that asked their age, gender, a description of their prior software engineering experience, whether they had worked together previously, and how their team was formed (instructor assignment, volunteer, or recruited.) A sample questionnaire is included in Appendix A, section B.1.

4.2.2.1.3 Concept Selection

Although the focus of SUM-ST is shared understanding among design concepts, two additional categories of concepts were included for exploratory purposes. Because of the importance of the relationship between design elements and requirements, requirements were also used as concepts. To explore knowledge differentiation within the team, team members' names were also used as concepts.

Because subject fatigue is a concern for a large number of concepts (see section 3.3.2), I targeted a 30 minute ratings questionnaire. Prior research suggests that participants can rate approximately 10 pairs of concepts per minute so this works out to a maximum of 25 concepts (300 comparisons.) The 25 concepts were allocated as follows: 5 for requirements, 1 for each team member name (5 or 6), and the remaining for design elements (14 or 15.)

Concepts representing design elements were selected according to the guidelines in section 3.4.1 and were the names of classes, methods, database tables, etc. To help reduce bias in the concept selection process, the final concept selection and phrasing was decided by consensus of three researchers familiar with programming concepts.

4.2.2.1.4 Ratings Questionnaire

A two-part web-based questionnaire was used to gather data about concept relationships. Part I of the questionnaire asked participants to provide pairwise relatedness ratings as described in section 3.4.2. Because Pathfinder does not assist in interpreting the nature of the relationship (only the strength), part II of the questionnaire asked participants to describe the resulting relationships. The intent was

to compare descriptions of the same relationship to see if participants had the same interpretation. Each team received a unique questionnaire developed from the concepts selected for the team. Part I of the questionnaire was identical for all members of the same team but part II was dynamic with each team member asked about relationships in their PFNet calculated from their answers to part I.

To establish context and orient participants to their projects, the questionnaire began with a question about their project and their role:

> Before starting, please think for a minute about the work your group has done on your project so far: identifying a customer, developing requirements, creating the system design, and now implementing the design.
>
> Please briefly describe the parts of the project that you worked on or are currently working on.

Part I presented all possible pairs of the 25 concepts, 20 per page. The question order and the order within the pair were randomized. Teams were given the following instruction:

> In part 1 you will judge the relatedness on a scale of 0 (Unrelated) to 6 (Strongly Related.) If you think the items are not related at all, choose 0. If the items have an obvious, strong relationship, choose 6. If the items seem related but the relationship isn't so clear, choose an appropriate level between 0 and 6. There are no right or wrong answers and you are not expected to take much time answering. For each answer, please give your first impression and then move to the next

question. There is no time limit. If you get tired, please take a short break and then come back to complete the rest of the questions.

Thinking of your group project, how strongly are the following words or phrases related to each other?

Part II of the questionnaire asked participants to give a short free-form text description of the relationship represented by each edge in the PFNet calculated from the answers to part I. Again, concept pairs were randomized. Because the number of edges in the PFNet is variable depending on the number of ties in path lengths, the number of questions in Part II was limited to 50 (selected at random) to guard against an overly long questionnaire.

I used web-based survey software LimeSurvey (Schmitz and LimeSurvey Project Team 2012) to gather relatedness ratings. I developed software in PHP (The PHP Group 2014), MySQL (Oracle Corporation 2016), and C to gather relationship descriptions, calculate the PFNet, and automate the conversion of a list of concepts to a working survey. A rating question is shown in Figure 4.2.

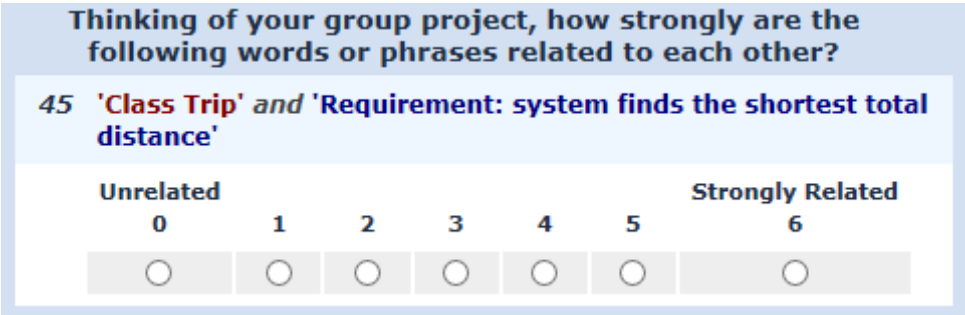


Figure 4.2. Pilot Study Rating Question

To investigate how shared understanding may change over time, and to associate changes in shared understanding with project design difficulties, participants were asked to complete the ratings questionnaire twice. They took it for the first time when the first draft of the system design document was complete and again shortly after the project was complete.

Participants completed the first ratings questionnaire in a lab under researcher observation following an interaction script (see Appendix A, section B.2.) Up to 4 participants were scheduled at the same time in the same room; however, to prevent discussion about concepts, members of the same team completed the questionnaire at different times.

After participants completed the questionnaire they were interviewed in private about the clarity of instructions, difficulty, fatigue, and any other problems (see the interview guide in Appendix A, section B.3.)

The second questionnaire was completed by participants on their own, unobserved.

4.2.2.1.5 Group Interviews

Group interviews were the primary method of data collection to answer the research questions (see section 4.2.1). The group interviews followed a semi-structured interview guide to drive at issues of shared understanding of the software design from two different perspectives. The first half of the guide consisted of general questions about the team's design process and became more focused on examples of where new design ideas were shared successfully or resulted in breakdowns. This portion of the interview guide was standardized across teams.

The second half of the interview guide was customized for each team based on an analysis of the team's rating questionnaire results. To form these questions, the concepts for the team were ranked by the two concept-level similarity calculations $\overline{C_v}$ and $\overline{r_v}$ (described in section 3.4.3) and by $\overline{Cor_v}$ (described in section 4.2.1). Approximately 3 each of the highest and lowest ranking concepts were selected to ask the team about. For these concepts, the team members' PFNets were analyzed to identify relationships in common and relationships that differed between pairs of team members. Questions were constructed by describing these similarities and differences, asking team members to discuss them, and asking them to speculate on the potential impact they may have had on the development of their project. A sample interview guide is in Appendix A, section B.4.

Group interviews were audio recorded, transcribed, and coded to find themes addressing the research questions.

4.2.2.2 Implementation

This section explains how the study design was implemented. It is presented in three parts, the study setting, the data collection activities, and the analysis procedures used.

4.2.2.2.1 Study Setting

I recruited participants from three sections of a 3rd year undergraduate computer science course on software design and development at UMBC during the fall semester (August through December) of 2013. The sections had a total of 89 students organized by the instructors into 17 teams of 5 or 6 students each. Of these, 39

students signed consent forms, of those 26 completed the demographic questionnaire, 20 of those completed the first ratings questionnaire, 14 of those participated in the group interview, and, finally, 11 completed the second ratings questionnaire. The analysis and results are limited to the 14 that participated in all the steps up to and including the group interview. There was not complete participation from any group.

Shortly after enrolling, participants were sent an e-mail invitation to take a web-based demographic questionnaire. The participants and demographics are shown in Table 4.1. Of the 14 participants that completed the study, 9 were male and 5 were female. The age range was 19 to 24. All participants had prior programming experience as part of course work. Two participants in team 9 and all participants in team 13 had professional programming experience. None of the participants had prior experience working together.

4.2.2.2.2 Data Collection

Before creating the questionnaires, concepts had to be selected from the teams' requirements and design documents. I asked teams to provide copies of their documents approximately 8 weeks into the class, shortly after the due date for the first draft. These documents varied greatly in style and detail. This was partially due to different teams following different process models (waterfall vs. iterative) and partially due to the inexperience of the student teams. Concepts were selected from text, tables, or diagrams. Because interaction with the teams was limited, it was impractical to determine which design elements were used by multiple team members prior to selection. A list of one team's concepts is shown in Table 4.2 on page 82. I

Table 4.1. Pilot Study Participants

Team	Team Size	Participant	Age	Gender	Experience
6	6	P6.2	20	Male	Hobby
		P6.3	19	Female	Hobby
		P6.5	23	Male	
9	6	P9.1	21	Female	Professional
		P9.2	22	Male	Professional
		P9.3	24	Male	
10	6	P10.2	20	Male	
		P10.4	22	Female	
13	5	P13.1	21	Male	Professional, Hobby
		P13.2	24	Female	Professional, Hobby
		P13.3	20	Female	Professional
		P13.4	19	Male	Professional, Hobby
14	5	P14.1	20	Male	
		P14.2	19	Male	

used custom software developed in MySQL to convert lists of concepts for each team into a fully functioning LimeSurvey questionnaire.

Participants made appointments to take the first ratings questionnaire in a usability lab over a one-week period. Upon arrival, I welcomed participants, gave instructions, and started each questionnaire, remaining in the lab to answer questions and observe any problems. There was no interaction between participants as they answered the questionnaire. I interviewed participants about their experience in private immediately after they completed part II.

I sent participants an e-mail invitation to take the second ratings questionnaire after their final projects were due. Most were completed before the scheduled group interview.

The group interviews were scheduled after the final projects were due. Group interviews were conducted by myself and most interviews had a second researcher as scribe. All interviews were audio recorded.

4.2.2.2.3 Analysis

Participant perceptions and usability of the elicitation technique were based on observations of participants taking the ratings questionnaire and the exit interviews after the questionnaire was complete. Exit interview notes were coded using an open coding approach starting with codes for accuracy and difficulty derived from the research questions. Coding was performed with MAXQDA (VERBI GmbH 2013).

I developed software in R (R Core Team 2013) to calculate and visualize PFNets and to calculate concept similarity measures. The visualization layout was based on the iGraph method layout.fruchterman.reingold() (igraph Authors 2014). An example of a PFNet is given in Figure 4.3.

A preliminary review of the PFNets revealed that 6 out of the 14 participants had trouble providing relatedness ratings for relationships involving team members. In these cases, participants rated some or all relationships involving team members equally. These equal ratings resulted in large numbers of edges in the PFNets. Many of these same participants had reported this difficulty in the exit interview. Based on this, all concepts based on team member names were eliminated from further analysis. This reduced the number of concepts from 25 to either 19 or 20, depending on the team size. Concepts for team 10 are shown in Table 4.2. This table is ranked by $\overline{C_v}$ with the values for $\overline{r_v}$ and $\overline{Cor_v}$ for comparison.

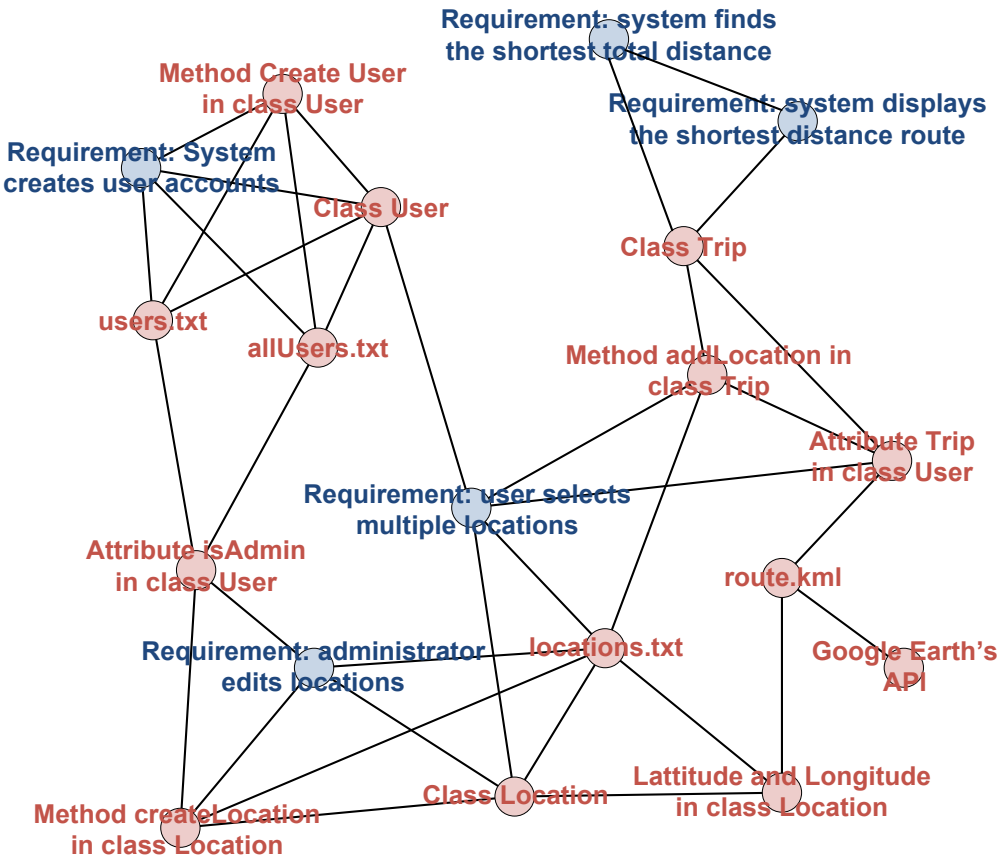


Figure 4.3. PFNet for Team 10, Participant 2

I constructed the interview guide for each team by ranking the remaining concepts according to the 3 similarity measures and selecting the highest 3 and lowest 3 concepts for discussion, as time allowed.

Group interviews were transcribed and coded using an open coding approach with predefined codes based on the research questions, but with codes added in vivo to represent emergent concepts, using MAXQDA (VERBI GmbH 2013). The code list is shown in Table 4.3. Initial codes identified the rationale for relatedness ratings and descriptions of similarities or differences in how team members understood the

Table 4.2. Team 10 Concepts Ranked by $\overline{C_v}$

$\overline{C_v}$	$\overline{r_v}$	$\overline{Cor_v}$	Concept
1.00	0.81	0.96	Requirement: system creates user accounts
0.80	0.67	0.81	Method createLocation in class Location
0.75	0.82	0.77	users.txt
0.67	0.85	0.77	locations.txt
0.60	0.53	0.82	Method addLocation in class Trip
0.60	0.43	0.56	Attribute Trip in class User
0.50	0.55	0.75	Requirement: system displays the shortest distance
0.50	0.83	0.69	Requirement: administrator edits locations
0.50	0.69	0.49	Class Location
0.50	0.76	0.65	Google Earth's API
0.43	0.37	0.62	Requirement: user selects multiple locations
0.40	0.79	0.89	Method createUser in class User
0.38	0.64	0.75	Class User
0.33	0.74	0.76	Requirement: system finds the shortest total distance
0.29	0.66	0.69	route.kml
0.29	0.77	0.78	Attribute isAdmin in class User
0.25	0.72	0.84	allUsers.txt
0.20	0.72	0.80	Attributes Lattitude and Longitude in class Location
0.14	0.24	0.77	Class Trip

concept. I also coded for descriptions of the possible impact those differences may have had.

The relationship descriptions (from part II of the questionnaire) were analyzed to see if team members shared the same interpretation of concept relationships. These descriptions were coded independently by three researchers. The same relationship only had multiple descriptions when the same edge occurred in the PFNet for more than one member of the team and when the description for that edge was not skipped due to excessive numbers of edges (> 50.) The descriptions were coded as *similar*, *different*, *incomplete*, or *mixed*. However, the free-form nature of the responses made

Table 4.3. Group Interview Codes

Code	Definition
Reasons for ratings	Rationale for relatedness ratings.
Personal	Reason for an individual rating.
Similarity between ratings	Why two people rated relationship similarly.
Difference between ratings	Why two people rated relationship differently.
Relationship Types*	Classifies how participants described a relationship.
Unknown relationship*	Participant cannot classify.
Implementation dependency*	Described as a dependency between program modules.
Satisfies requirement*	Relationship allows the program to satisfy a requirement.
User activity*	Described as a sequence of user activities.
Program activity*	Described as a sequence of program activities.
Impact	Impact of differences on outcomes.
Some effect on project outcomes	Differences had an impact.
No effect on project outcomes	Differences did not have an impact.
Not shared*	Because concept was not shared.
Natural maturation*	Because difference was discovered before it became a problem.

*code was added in vivo

interpretation difficult and the analysis had a low inter-rater reliability. Because this was inconclusive, results are not reported.

4.2.3 Pilot Study Results

The objectives of the study were to identify problems with the study method, identify potential usability issues with the technique that may affect results, select an appropriate graph similarity calculation method, and gain preliminary data to answer the research questions (listed in section 4.2.1.) This section presents relevant results.

The following discussion section discusses these results to address the research questions.

These results begin with an analysis of issues that may affect the accuracy of the relatedness ratings in section 4.2.3.1. This is followed by a summary of those ratings in section 4.2.3.2. Section 4.2.3.3 reviews the graph similarity calculation methods to select a method for use for subsequent results. Section 4.2.3.4 examines issues of stability and potential changes in shared understanding by comparing the two questionnaires taken by each team. Section 4.2.3.5 compares the concept similarity measures with participant's statements of shared understanding in the group interviews. Finally, section 4.2.3.6 compares these results with design difficulties experienced by the team.

4.2.3.1 Ratings Questionnaire Experience and Usability Issues

For SUM-ST to be a valid measure of shared understanding, the elicitation method must accurately capture how participants actually feel about the relatedness of each pair of concepts. For example, if team members become fatigued, or if user interface design problems lead team members to select the wrong response, the ratings captured may not be accurate. Although there was no independent measure to determine how participants "felt", it was possible to gain some insight into this issue through observation, the exit interviews, and the relationship descriptions from part II of the questionnaire.

During the first ratings questionnaire, I was available to answer questions and to observe participants. There were no significant questions asked or problems observed.

The time to complete the relatedness ratings ranged from 27 to 68 minutes with a median of 40 minutes. This corresponds to a rate of 4.4 to 11.1 ratings per minute with a median of 7.5 ratings per minute. Part II, describing relationships, was not timed but no participant exceeded the 90 minute (total) appointment.

In the exit interview, participants universally described the questionnaire as easy, yet long and repetitive. Some described it as fatiguing but others did not. A few said that they had difficulty remembering what all of the concepts meant.

Although participants did not report difficulty with the instructions, they did not always follow them, particularly in cases where they could not easily conceive of a relationship. Instead of marking the pair as unrelated, they reported trying to make up a relationship or choosing an answer without justification. One participant said “if I had no idea what something was, I just made the relationship like a 2 or something." This suggests that participants distinguish between relationships for which they have a basis of comparison (and could be marked “unrelated”), and relationships that have no basis for comparison, for which they had no correct action to take.

Some participants reported that they had trouble remembering what was in their documentation. One participant explained that the team’s terminology had drifted from what they used in their document so they were not as familiar with it any longer. One person gave a more general response: "It is hard to remember everything."

A number of participants reported that the process of rating became easier over time. One participant said that he created a “relationship model” that he mentally referred to later in the questionnaire. Others reported that they learned about their project by

considering relationships they had not considered before, suggesting that their knowledge structure was changing as they answered questions.

There were few self-reported errors. One participant said he became more “care free” and made mistakes as the questionnaire progressed. There were a small number of cases where relationships were marked with a strong relatedness rating in Part I but were described as “not related” in Part II, suggesting the rating was made in error.

In summary, the questionnaire was easy but tedious. Some pairs were difficult to evaluate because of changes in terminology or a lack of a basis for comparison. A few answers were inconsistent indicating an error. On the whole, though, these problems were minimal.

4.2.3.2 Relatedness Ratings

Although the emphasis of this research is on the comparison of networks produced by Pathfinder from the relatedness ratings, one of the assumptions of the SUM-ST approach is that, by tracing the network differences to their source, those results may be made actionable. Therefore, this section reviews the relatedness ratings themselves.

Table 4.4 shows the frequency distribution of the relatedness ratings given in part I of the ratings questionnaire (excluding concepts representing people (see section 4.2.2.2.3.)) On the whole, the greatest number of pairs are rated ‘Unrelated’ followed by ‘Strongly Related’. Similar summaries of ratings are rarely reported in the literature but one study was found for comparison. That study was a novice/expert

Table 4.4. Pilot Study Ratings Frequency Distribution

		Questionnaire 1 Rating							Questionnaire 2 Rating						
		0	**1**	**2**	**3**	**4**	**5**	**6**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Team 6[a]	P6.2	35	33	17	8	24	27	27	19	31	20	11	26	28	36
	P6.3	6	19	47	32	31	19	17	*not taken*						
	P6.5	11	36	7	22	44	27	24	30	34	13	26	23	24	21
Team 9[a]	P9.1	21	29	40	36	24	10	11	6	6	30	37	57	29	6
	P9.2	13	19	24	12	33	35	35	*not taken*						
	P9.3	38	9	23	29	33	20	19	7	17	21	22	50	42	12
Team 10[a]	P10.2	61	19	12	14	15	20	30	86	8	7	12	13	11	34
	P10.4	20	44	13	17	17	24	36	15	51	10	23	21	16	35
Team 13[b]	P13.1	81	21	16	19	9	17	27	103	32	19	9	12	9	6
	P13.2	92	4	9	21	6	4	54	1	0	4	25	44	21	95
	P13.3	51	13	13	37	31	12	33	10	33	22	48	26	6	45
	P13.4	71	13	12	35	27	10	22	74	14	10	47	15	3	27
Team 14[b]	P14.1	74	31	18	14	16	13	24	*not taken*						
	P14.2	57	40	32	17	14	6	24	*not taken*						
	Total	**659**	**354**	**303**	**329**	**336**	**252**	**387**	**379**	**250**	**176**	**276**	**299**	**197**	**321**
		25%	**13%**	**11%**	**12%**	**13%**	**10%**	**15%**	**20%**	**13%**	**9%**	**15%**	**16%**	**11%**	**18%**
	Rank	**1**	**3**	**6**	**5**	**4**	**7**	**2**	**1**	**5**	**7**	**4**	**3**	**6**	**2**

[a] Out of 171 questions total **Key:** 0 - 6 'Unrelated' to 'Strongly Related'

[b] Out of 190 questions total

comparison of students learning research techniques (Goldsmith and Johnson 1990). These results contrast with those in that they found neither "most related" or "least related" to be most frequently used, in fact, 'least related' was the least common response. Ratings are expected to be domain and context dependent (Gammack 1990) and, therefore, these differences in findings are not surprising, but understanding the reasons for such differences may be valuable for future research.

4.2.3.3 Comparison of Graph Similarity Calculation Methods

RQ4 asks to compare the $\overline{C}$, $\overline{r}$, $\overline{NETSIM}$, $\overline{C_v}$, and $\overline{r_v}$ graph similarity calculation methods (described in sections 3.3.4 and 3.4.3.) The $\overline{Cor_v}$ similarity measure (correlation of the raw relatedness ratings) was used as a point of comparison. The analysis is based on an inspection of the PFNets for the various team members and a review of the corresponding team similarity scores and concepts ranked by the concept-level similarity measures. Cases were identified where the similarity calculations gave widely varying measurements and those were investigated to identify the source of the differences.

Based on this analysis, $\overline{Cor_v}$ gives an indicator of the degree to which two participants agree. However, because most pairs of concepts are unrelated, these unrelated pairs tend to dominate the correlation, resulting in a measure of the degree to which the members agree that concepts are *unrelated* rather than the way in which they are related. For example, consider concept #3 for team 13, participants 1 and 2. Their relatedness ratings are shown in Table 4.5. Of 19 concepts, 9 are rated as unrelated (0) for both team members. The overall correlation is 0.6670. Ignoring the 9 concepts

marked as unrelated gives a much lower correlation of 0.2737. Pathfinder avoids this problem by identifying the salient relationships within the relatedness data and eliminating the others from the comparison.

Table 4.5. Relatedness Ratings for Team 13, Team Members 1 and 2

	Rating between concept 3 and concept																			
Person	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P13.1	0	0	-	0	0	2	1	3	2	6	0	4	3	0	0	0	0	0	0	5
P13.2	0	0	-	0	0	6	0	4	0	6	0	6	6	6	0	0	0	0	6	6

This analysis shows that $\overline{r}$ is the most sensitive to the overall shape of the network. For example, consider the graphs in Figure 4.4. The difference between graph *A* and graph *B* is the addition of edge $\{V_2, V_4\}$. This makes only a local change in neighborhoods but global changes in distances. This results in a much lower similarity measure by $\overline{r}$ than by $\overline{C}$ or $\overline{NETSIM}$. Based on analysis of these data, $\overline{r}$ is overly sensitive to single edge differences between graphs. Additionally, $\overline{r}$ is undefined when all of the distances between nodes are the same, which can happen when there are many identical relatedness ratings. Based on these properties, the conclusion is that $\overline{C}$ and $\overline{NETSIM}$ are more appropriate graph similarity measures than $\overline{r}$. A prior comparative study reached a similar conclusion (Goldsmith and Johnson 1990).

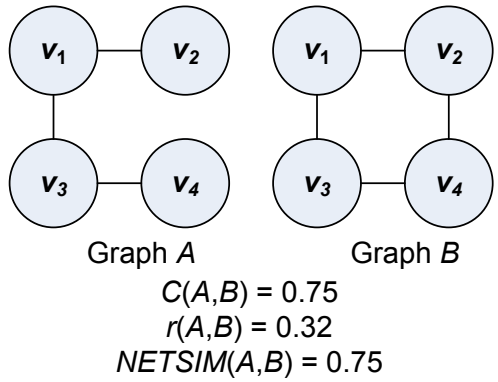


Figure 4.4. Graph Similarity Comparison

The measures $\overline{C}$ and $\overline{NETSIM}$ are very similar, producing a difference of no greater than 0.04 in the data set collected for the pilot study. The primary advantage of $\overline{NETSIM}$ is the ability to perform statistical significance testing, however, that is not an advantage in this application. Because of the structural similarities between C, $\bar{C}$, C_v, and $\overline{C_v}$, it was concluded that the family of measures based on C, rather than *NETSIM*, were most compatible. Therefore, the remaining analysis was based on these measures.

The analysis of the PFNets identified a potential issue where small changes in the relatedness ratings can produce large differences in the PFNets produced by Pathfinder. This issue occurs when there are many ties in the relatedness ratings. If these ties result in equal path lengths, Pathfinder may preserve a large number of edges. However, a small change in the weight of one edge may break these ties, producing one shortest path and eliminating the others. Under these conditions, a small difference in one rating response that may not be considered significant by the team member may yield a large difference in the structure of the PFNet and,

therefore, any graph similarity comparison based on it. This is an important limitation of Pathfinder analysis.

4.2.3.4 Changes Over Time

Changes in relatedness ratings over time are measured in two different ways. The first is to compare team similarity calculated from the first questionnaire to that calculated from the second questionnaire. Changes in this measure may be associated with changes in shared understanding by the team in the interval between taking the questionnaires. The second way is to measure the similarity of PFNets over time for each individual. Although not a measure of shared understanding, this gives an indication of the stability of the measure and may provide insight to the changes in team understanding.

Not all participants took the 2nd questionnaire. Teams 6 and 9 lost one participant each and team 14 lost both participants, leaving only 4 teams.

Table 4.6 shows the within-team similarity, as measured by $\overline{C}$, for each questionnaire, and their difference. Measures for teams 6, 10, and 13 were relatively stable but team 9 showed an increase from the first questionnaire to the second. These results are inconsistent with results reported by Curtis and Walz (1990) and Levesque and colleagues (2001) who reported that, after the initial design phase, understanding diverges in software teams. One possible explanation for the results from team 9 is that the design document could have been primarily the product of one member of the team and that other members gained understanding as the project was implemented, but this was not verified with the team.

Table 4.7 shows the comparison between each participant's individual PFNets from each questionnaire, as measured by C. The range is 0.19 to 0.64 with a median of 0.42. The members of team 9 had the lowest self-similarity over the period which is consistent with the greatest change in team level similarity discussed above.

Table 4.6. Team Similarity Questionnaire 1 vs. Questionnaire 2

Team	Q1 similarity ($\overline{C}$)	Q2 similarity ($\overline{C}$)	Difference
6	0.39	0.34	-0.05
9	0.15	0.28	0.13
10	0.48	0.52	0.04
13	0.31	0.30	-0.01

Table 4.7. Self-Similarity Questionnaire 1 vs. Questionnaire 2

Group	Participant	C
6	P6.2	0.35
	P6.5	0.49
9	P9.1	0.22
	P9.3	0.19
10	P10.2	0.52
	P10.4	0.49
13	P13.1	0.34
	P13.2	0.50
	P13.3	0.64
	P13.4	0.35

4.2.3.5 Concept Similarity Measure and Shared Understanding

When teams were asked about concepts with low similarity measures, discussion usually revealed a difference in understanding of the concept within the team. Participants were able to provide a rationale for the relationships in their PFNets and, by comparison with other team members, identify differences in understanding. For example, one team discussed a difference in understanding of the startup operation of their program. Their program was an implementation of the traveling salesman

problem. Based on their PFNets, one participant had a relationship between program initialization and route calculation that another participant did not. In discussion, the first participant said that they expected distances between cities would be calculated once at initialization. This had not occurred to the other participant.

Concepts with low similarity scores could best be characterized as having a general lack of understanding rather than conflicting understanding. Sometimes this could be attributed to immature ideas at the time the questionnaire was taken. In other cases, one person appeared to have a mature understanding but the others did not, associated with specialization and role responsibilities in the team.

There were two cases of low similarity scores that were not confirmed as differences in understanding during the interviews. In these cases, the participants suggested it was a false difference because two terms used in their documentation represented the same concept but were presented as separate concepts in the questionnaire. Although the structure in the PFNets was different around these concepts, the team members did not think these differences were important.

It was harder to investigate cases where concepts had high similarity measures than cases of low similarity measures. Without a conflict identified to discuss, there was not much to talk about. When pressed to explain a case where two participants had identical relationships, one team member said "I guess it was just a mutual understanding." followed by another saying "Common sense. Like now. It just makes sense." In another case, the team’s highest similarity score was for a requirement for the project. The participants suggested the reason that they understood this similarly

was that it was a core requirement saying “Everyone knew it’s going to be what’s at the centerpiece of our program because it had to calculate the route to meet the requirements of the project.”

In one case, one team suggested the reason a concept representing a Java class had a high similarity score was that it was small with little functionality and that the simplicity contributed to the high similarity measure.

4.2.3.6 Association Between Measured Differences and Design Difficulties

In the group interviews, when teams described differences in understanding about a concept they were asked to judge whether this could have had an impact on the development of their project and its outcomes. This was a difficult question for the teams to answer. Teams, in general, had more difficulty understanding and using new technologies, such as databases and web design technologies, than they did their own designs. When discussing difficulties in the project, it was difficult for teams to distinguish between concepts in the questionnaire and other issues.

In two cases, participants explained that differences in understanding did not have an impact because the differences were in portions of code that were not shared between two people. This is consistent with expectations for modular design and the limitation of this study of not being able to specifically target collaborative concepts during concept selection.

In two other cases, participants explained that the differences in understanding were related to problems in the design but explained this was due to an immature design

that was corrected naturally as the project developed. For one of these cases, one participant suggested that the team could not have understood the problem prior to attempting to code the invalid design.

Although not described as problems by the teams, there were three cases of low similarity concepts that were associated with changes in the project. For example, one team's design had a class that saved details of a traveling salesman's route to disk. This class had the lowest similarity rating of concepts in their team. As program development progressed, the team realized that another feature of the program, sending routes by e-mail, satisfied the requirement and saving to disk was not needed, so the class with low similarity rating was removed.

4.2.4 Pilot Study Discussion

This section reviews the results of the pilot study to answer the research questions and to address the objectives of the study.

4.2.4.1 Answers to the Research Questions

The pilot study evaluated the SUM-ST technique for measuring concept similarity within a software design team, the possible association of these measures with differences in understanding in the team, and the possible implications of those differences in the development of the software design for the project. The pilot study addressed the following research questions (RQ3 was not addressed in the pilot):

RQ1: How are similarities and differences between the software development team members' knowledge structures, as measured by SUM-ST, associated with similarities and differences in how those team members understand their design?

RQ2: How are measured differences in software development team members' knowledge structures associated with identifiable difficulties in the software design process resulting in degradation of cost or quality?

RQ4: Given individual relatedness ratings captured from a team of developers about a common design, how do the graph similarity measures $\overline{C}$, $\overline{r}$, $\overline{NETSIM}$, $\overline{C_v}$, $\overline{r_v}$, and $\overline{Cor_v}$ compare?

The study results found that concepts with a high similarity measure generally have a greater shared understanding in the team than concepts with a low similarity measure (RQ1), supporting this technique as a valid measure of shared understanding. Teams described concepts with low similarity scores in different ways and discussion of these concepts revealed differences in the knowledge of these concepts within the team. In many cases, teams identified differences that they were unaware of. When asked about concepts with high similarity, teams had similar knowledge of the concept and were unable to articulate significant differences. However, results were not perfect and counterexamples exist. These results are consistent with previous research using paired-comparisons/Pathfinder for measuring shared understanding (Kudikyala 2004).

The association between the differences in understanding and process difficulties within the team (RQ2) is less clear. Technical coordination problems in the teams

were dominated by external components, rather than the shared design constructed by the team. Many differences in understanding did not involve collaboration between team members and, therefore, did not contribute to coordination problems.

Some differences in understanding were associated with changes in the project design. From the team's point of view, this lack of shared understanding was described as "immature design" that was corrected "naturally" as the team continued their development. Due to the small scope of the student projects, it is not possible to differentiate between an immature design being refined by an efficient design process or a sequence of faults and corrective rework. This issue was re-examined in the main study with industrial development teams.

The analysis of the graph similarity measures (RQ4) found that the correlation of the relatedness ratings ($\overline{Cor_v}$) was dominated by unrelated relationships and, therefore, is not a measure of shared understanding. The $\overline{r}$ and $\overline{r_v}$ measures are undefined in important cases and are sensitive to single edge differences that may be more attributable to variations in how team members rate a relationship than their underlying knowledge structure. The $\overline{NETSIM}$ measure mostly duplicates the $\overline{C}$ measure but offers no additional benefits in this application. The neighborhood comparison methods $\overline{C}$ and $\overline{C_v}$ are defined in all cases and had the best representation of the differences in the ratings of the similarity measures analyzed. This is consistent with Goldsmith and Johnson (1990) and Kudikyala (2004).

4.2.4.2 Discussion of the Study Objectives

The objectives of the study (detailed in section 4.2.1) were to gain experience using the SUM-ST technique with developers, to develop the necessary software tools, to evaluate the developer's experience using it, and to identify potential problems with the research method.

Although I had researched and planned the pilot study extensively, I was concerned that, due to the newness of research method and SUM-ST technique, some significant problem might occur. The most significant problem was the difficulty that some participants had providing relatedness ratings for comparisons involving their team members. Otherwise, the study went according to plan and the software tools that were developed performed as expected.

The ratings questionnaires took longer than projected at a median of 40 minutes instead of the planned 30 minutes (corresponding to a median rate of 7.5 ratings per minute instead of 10.) The participants reported that the ratings questionnaires were easy to take, but were long and tedious. To keep the questionnaire within the original 30 minutes, the number of concepts should be reduced from 25 to 22.

Some valid differences in understanding did not have an impact on project outcomes because the team members did not collaborate about those concepts. This was due, at least in part, to the limited interaction with the team that prevented targeting concept selection in this way. This emphasizes the need to focus the limited number of concepts that can be measured at once to specific areas of collaboration where a difference in understanding is expected to lead to coordination problems. Concept

selection may be guided by team communication patterns, logical dependencies, or defined interfaces.

While few errors were found in relatedness ratings, the impact of an error could significantly affect the structure of the PFNet, and therefore the similarity measures. One approach to reduce these errors would be to calculate the PFNet once all of the ratings are provided and then ask participants to re-rate those relationships represented in the PFNet. The value of this additional step would need to be balanced against the additional burden on the participants for additional ratings.

The results included cases where participants were unable to judge the relatedness of a pair of concepts and did not have an appropriate selection, leading to invalid ratings. This suggested the need for a way to skip the rating of a pair of concepts. The two reasons encountered in the pilot study were concept labels that were unfamiliar and comparisons that made no sense. Additional answers should be added to the questionnaire to address these cases and a catch-all for other cases that may not have been identified. Additional research was conducted as part of the main study to address the appropriate edge weight for these cases.

The guidelines for concept selection are helpful but need to be refined. Concept labels that are synonymous and very simple concepts can result in similarity measures that do not reflect the team's understanding. Also, selecting concepts based on a document review may select terms that are better understood by the document's author than their teammates and introduce bias in the measurement. Documents may also be

inappropriate if they are out of date or for teams who do not have a document centric process. Interview techniques may be more appropriate in these cases.

4.2.4.3 Limitations

The following limitations are recognized in this study. With respect to construct validity, student teams and projects are a poor proxy for industrial teams due to differences in experience, project scope, and other factors. By using document analysis for concept selection, the concepts and terms may have been overly influenced by the document's author. Although steps were taken to standardize concept selection, it is still a subjective process and the concepts selected may not have been the most illuminating.

With respect to internal validity, although relatedness ratings are a well-established approach, there are limited data to verify that the ratings captured the participant's intent and some inconsistencies were found indicating some ratings were in error. Participants had difficulty rating concepts based on team member names. Although these were discarded, this could have affected other ratings. This study did not have complete participation of any team and there may be selection/attrition bias associated with those that chose to participate. This study used only one indicator of shared understanding, group interviews, with no other indicators as a point of comparison. Interviews may be limited due to participant difficulty in articulating issues of understanding or the researcher interpreting the results.

This was a pilot study with a small number of subjects and do not necessarily generalize to other software design teams.

4.2.4.4 Lessons Learned

The results of the pilot study suggested the following changes to the SUM-ST technique and the research method:

- The neighborhood-based comparison measures $\overline{C}$ and $\overline{C_v}$ were found to be the most appropriate.
- The maximum number of concepts should be 22 instead of 25.
- Participants should have the option to not rate a pair of concepts.
- Concept labels should not be synonymous and should not represent very simple concepts.
- The relationship descriptions (from part II of the questionnaire) should be restructured to improve analysis.

4.3 Main Study Method

As with the pilot study, the main study method is presented in two parts. The first part, the study design, describes the study as originally planned while the second part, the implementation, describes the details of how the study was conducted.

4.3.1 Design

The goal of the main study was to further explore the SUM-ST technique and gather data to answer the research questions introduced in section 1.2, reproduced here for convenience:

RQ1: How are similarities and differences between the software development team members' knowledge structures, as measured by SUM-ST, associated with similarities and differences in how those team members understand their design?

RQ2: How are measured differences in software development team members' knowledge structures associated with identifiable difficulties in the software design process resulting in degradation of cost or quality?

RQ3: How can measured similarities and differences in team member knowledge structures be used to improve the process of software design?

The main study followed the same template as the pilot study but, based on the lessons learned, had the following changes to the SUM-ST technique and the research method.

Changes to the implementation of the SUM-ST technique were:

- Concepts were selected by team leaders and validated by another team member.
- The ratings questionnaire included an option to not rate individual concept pairs.
- PFNet comparison was performed solely with the neighborhood comparison measures $\overline{C}$ and $\overline{C_v}$.

Changes to the research method were:

- Participants were members of industrial software development teams with all developers participating.

- To better understand the team context and to investigate other factors that may influence shared understanding, the demographic and interaction questionnaires asked each participant about the amount of experience they had working with the others in their team and the amount of communication they shared.

- All concepts represent design elements (not requirements or people.)

- Participants were not observed while taking the questionnaires and there were no exit interviews.

- Part II of the ratings questionnaire (that prompts for relationship descriptions) was semi-structured instead of free-form.

- Opportunistic observations were added to give insight into the team working processes and to (possibly) observe breakdowns in the design process.

- Group interviews were supported by new interactive software to visualize the PFNet comparisons.

- At the conclusion of the study, the findings were member checked by asking participants via e-mail whether or not they agreed with them.

The main study data gathering activities are summarized in Figure 4.5.

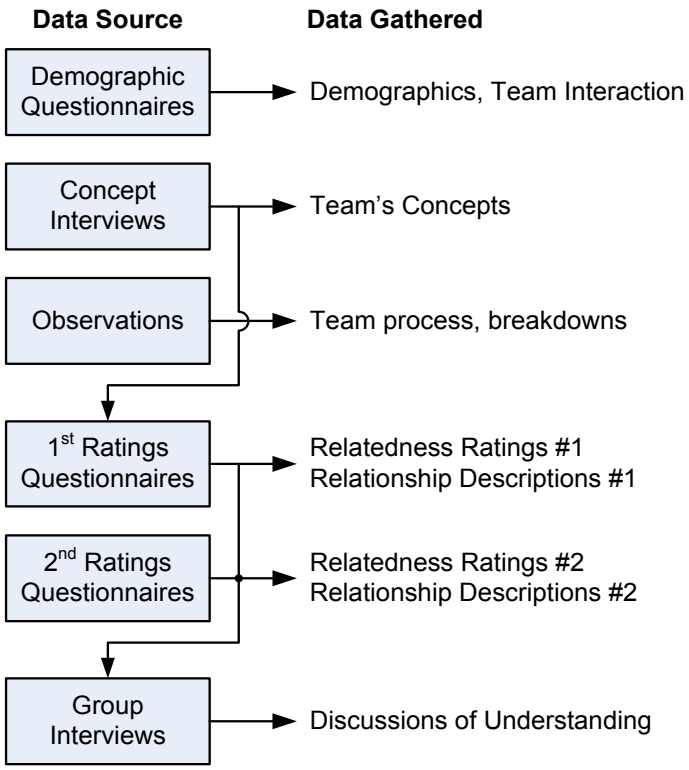


Figure 4.5. Main Study Design Overview

4.3.1.1 Participants

To be eligible, participants had to be members of an industrial software development team that was developing new software features that required some form of collaborative design activity. Teams were required to have at least 5 and no more than 10 members.

4.3.1.2 Demographic and Interaction Questionnaire

To understand more about the working habits of the team, factors that may contribute to the development of shared understanding, and possible alternative explanations of the results, the participants were asked to take a questionnaire that asked about

demographics and team interaction. This was a web-based questionnaire that asked their age, gender and years of software development experience. It also asked about the amount of experience, including previous employment, with each other member of the team and the time spent each week communicating with each other member of the team in various modes (e.g. group meeting, text chat, etc.) A sample questionnaire is included in section C.2 of the Appendix.

4.3.1.3 Concept Selection

Between 20 and 22 concepts representing design elements from the team's current work were selected by the team leader using the guidelines in section 3.4.1. To ensure concepts were clear and unambiguous, I interviewed a second member of the team to validate the concepts. This person defined each term and verified it against the guidelines. These interviews were audio recorded.

4.3.1.4 Observation

Team meetings were observed to provide further context to interpret results and to give the opportunity to observe breakdowns of shared understanding within the team. Observations were opportunistic and guided by the following sensitizing concepts (Bowen 2008):

- Poor outcomes such as design changes, rework, or test failures.
- Conflicts or lack of understanding such as discussion of different points of view or explanations of software design.

- Design concepts from the team's list.

4.3.1.5 Concept Rating Questionnaire

The concept rating questionnaire was similar to that used for the pilot study (see section 4.2.2.1.4) with some changes based on the lessons learned.

Feedback from the pilot study suggested that participants conceptualized unrelated concepts differently from related concepts. That is, 'unrelated' is not a relationship strength, but a different category of answer. Accordingly, the option for 'Not at all' related was separated from the options indicating a relationship strength. The number of options for strength was reduced from 6 to 5. An additional option of 'No answer' was added that, when selected, allowed the user to give a reason for not providing an answer. The updated question format is shown in Figure 4.6.

Participants were given the following instruction:

> In part 1 you will judge the relatedness on a scale of 0 (not at all related) to 5 (strongly related.) If you don't recognize a term or the question doesn't make any sense, you may choose "No Answer" and provide a reason. There are no right or wrong answers and you are not expected to take much time to answer. For each question, give your first impression and then move to the next question. There is no time limit. Thinking of your project, how strongly are the following words or phrases related to each other?

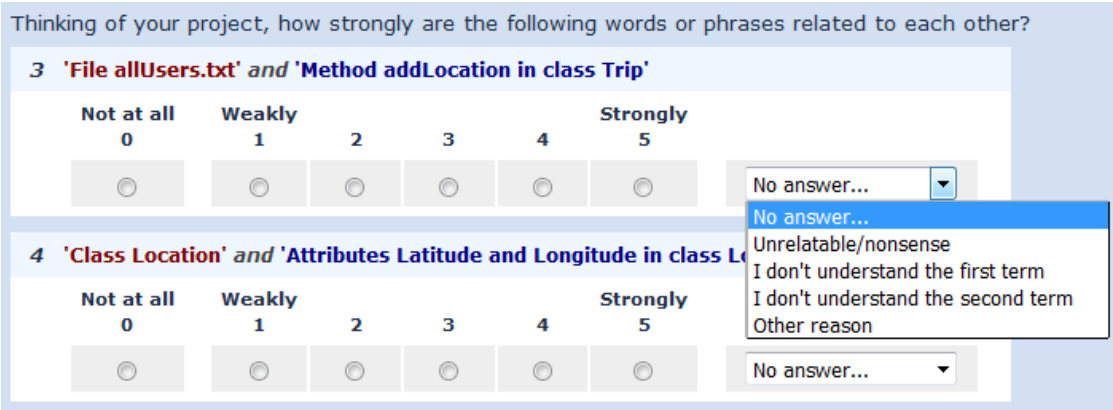


Figure 4.6. Main Study Rating Question

To help make the relationship descriptions easier to analyze, the questions for part II of the questionnaire were changed from free-form to semi-structured. Based on work from Cooke and Schvaneveldt (1988, 421), participants were asked to fill in a linking phrase to make a valid sentence relating the two concepts. The order of the concepts in the pair was user selectable. The updated question structure is shown in Figure 4.7.

The instructions for part II were:

> Part 2 asks you about just a few of the pairs that you rated in part 1. For this part, fill in the blank to make a sentence that links the two words or phrases. For example, given the words "Trees" and "Leaves" you might answer "Trees **are covered by** Leaves." Examples of linking words are: "does", "contains", "is a type of", "describes", "controls", "is part of", "is made of", "requires", "uses", etc. You may use any words that make a sentence that makes sense to you. If there is more than one possible answer, choose the first answer that occurs to you. There are no right or wrong answers. You can change the order of the words by clicking the button.

4	'Class Trip'		'Class Location'
5	'Attribute Trip in class User'		'File locations.txt'
6	'File users.txt'		'Google Earth's API'

Figure 4.7. Main Study Relationship Description Question

I wrote new custom software in PHP (The PHP Group 2014) to convert a list of concepts to a functioning LimeSurvey (Schmitz and LimeSurvey Project Team 2012) questionnaire. I customized the LimeSurvey question format to support the drop-down style "No Answer" field using JavaScript. I wrote Part II of the questionnaire in PHP, JavaScript, and C.

4.3.1.6 Computation of Not Related and No Answer Responses

As defined, the Pathfinder algorithm requires that relatedness ratings use a scale where all pairs are considered to be related, even if only minimally. In extreme cases, it is possible for edges representing minimally related pairs to be retained in the PFNet. The changes to the ratings questionnaire to support a separate "Not Related" and various "No Answer" responses required changes to the algorithm to support missing relationships.

Without modification (as defined by Dearholt and Schvaneveldt (1990, 5)) the weights used to define the PFNet are finite and the resulting PFNet will be connected (meaning there is some path between all node pairs.) Most prior studies in the

literature used the anchors of “least related” and “most related” for the relatedness rating. In the Pathfinder computation, “least related” is mapped to the largest possible edge weight and, because path distances are the maximum edge weight along the path, paths including these edges have the largest possible distance. Pathfinder will drop these edges from the network if there are any other paths with a smaller distance. However, if *all* possible paths contain a “least related” edge, all of the path distances will be equal and the “least related” edge will not be dropped. For example, if every rating involving a particular concept is “least related”, all paths involving this concept will be equal, and none of the edges will be dropped. This results in the concept being linked to every other concept in the PFNet. Although perhaps unexpected, this is consistent with the anchor interpretation.

To prevent pairs rated as “not related at all” or “No answer” from being connected in the PFNet they cannot simply be treated as a large weight. However, rather than redefine the Pathfinder algorithm directly, these cases were handled in a post-processing step. These responses were assigned a special large weight, the PFNet was computed as usual, and then any remaining edges containing this weight were dropped. Because this can only occur when there are no valid relatedness ratings connecting some pair of nodes, this divides the graph into two or more sub-graphs. The case where a sub-graph contains a single node is a special case for the concept similarity measure C_v. The similarity between two disconnected nodes is 1 and between a disconnected node and a node that is not disconnected is 0 (see equation (3.7).)

4.3.1.7 Result Visualization Tool

Some basic PFNet visualizations were developed in R for the pilot study to assist with analyzing the results and were used to manually identify similarities and differences between participant's PFNets to investigate during the group interviews. This was adequate for viewing the PFNets but cumbersome for comparing them, even for the relatively small number of participants in the pilot study. Given the increase in team size for the main study, a new approach was required.

Of particular concern for the main study was that different sub-teams may collaborate about different concepts and it wasn't possible to know this structure in advance of the group interview. In the worst case, a team of 10 people has 1,013 possible subgroups to compare (calculated as the power set (1,024) minus the single person subsets (10) minus the empty set (1)). Because of the large number of possible comparisons, I wanted an interactive tool that allowed exploring the people and relationships of interest and that could be interpreted by the team members themselves. I designed and coded such a tool and used it for my own analysis and to support the group interviews. This tool calculates ranked concepts, measures of shared understanding, and gives visualizations of the relationships between concepts shared by team members. The tool was developed in PHP (The PHP Group 2014), JavaScript, and D3.js (Bostock 2015).

The tool allows exploring the members and concepts of a single team. It is organized as two web pages. The first page allows selecting the concepts to be included and the questionnaire (first or second) to analyze (Figure 4.8.) After selection, the PFNets are

calculated and a results page is displayed. The results page allows individual team members to be selected for comparison, compares their PFNets, and displays three different views of the results.

The first view, the *Ranked Concepts* (Figure 4.9), identifies the concepts with high and low similarity. This view shows the overall team similarity $\overline{C}$ and a table of concepts ranked by $\overline{C_v}$. The table allows drill-down on individual concepts to show the linked concepts, the members that have this edge, and those that do not. The drill-down table is sorted according to the number of people that share the edge.

The second view, the *Team Map* (Figure 4.10), identifies edges in common to different numbers of team members. This view shows the union of the PFNets for the team with the edges color coded according to the number of people that share that edge. Edges in common to a larger number of team members may be more salient than those in common to a smaller number of team members. Edges shared by about half the team members may represent disagreement. Because this view can be busy, it supports filtering based on edge count. The layout is based on d3.layout.force() and nodes are labeled and colored according to $\overline{C_v}$.

The third view, the *Detail Map* (Figure 4.11), gives a graphic view of a selected concept and is closely related to the drilldown table in the ranked concept view. It shows all concepts linked to the selected concept grouped by the team members that share the edge. Links are sorted based on the number of team members sharing the edge(s).

Analyzing group 23 - Demo.

Choose design elements:

- ☑ External Processor
- ☑ Protocol Element
- ☑ Index Protocol Element
- ☑ File Transfer Failure
- ☑ Transfer App
- ☑ Interface App
- ☑ Primary Initialization
- ☑ Backup Initialization
- ☑ Recording Unit
- ☑ Network Processor
- ☑ Operating System
- ☑ File Transfer App
- ☑ End of File
- ☑ File Replication
- ☑ System Reset
- ☑ External Commands
- ☑ File System Software
- ☑ Directory Index
- ☑ Disk Subsystem
- ☑ CPU Load

Choose questionnaire:

- ◉ Questionnaire 1.
- ○ Questionnaire 2.

Results...

Figure 4.8. Result Visualization Tool, Selection Page

Analyzing group 23 - Demo, Questionnaire 1.

Choose members: ☑ David ☑ Alice ☑ Bob ☑ Christie ☑ Erika

Average team similarity: 31%

Ranked design element similarity:

%	Design Element	- Related Element	Having	Not Having
48%	Network Processor	Protocol Element	David, Alice, Bob, Christie, Erika	
		Index Protocol Element	David, Alice, Bob, Christie, Erika	
		File Transfer App	David, Alice, Bob, Christie, Erika	
		Interface App	Bob, Christie	David, Alice, Erika
		Primary Initialization	David, Christie	Alice, Bob, Erika
		Backup Initialization	David, Alice	Bob, Christie, Erika
		External Commands	Alice, Christie	David, Bob, Erika
		Directory Index	Christie, Erika	David, Alice, Bob
		Recording Unit	Christie	David, Alice, Bob, Erika
		Operating System	Christie	David, Alice, Bob, Erika
		End of File	Christie	David, Alice, Bob, Erika
48%	File Replication	+		
47%	External Processor	+		
43%	Disk Subsystem	+		
41%	File System Software	+		
41%	Index Protocol Element	+		
38%	Protocol Element	+		
35%	Recording Unit	+		
35%	File Transfer App	+		
34%	File Transfer Failure	+		
32%	End of File	+		
31%	Interface App	+		
28%	External Commands	+		
27%	Transfer App	+		
22%	Directory Index	+		
19%	Operating System	+		
17%	Primary Initialization	+		
16%	Backup Initialization	+		
14%	System Reset	+		
7%	CPU Load	+		

Figure 4.9. Result Visualization Tool, Ranked Concepts

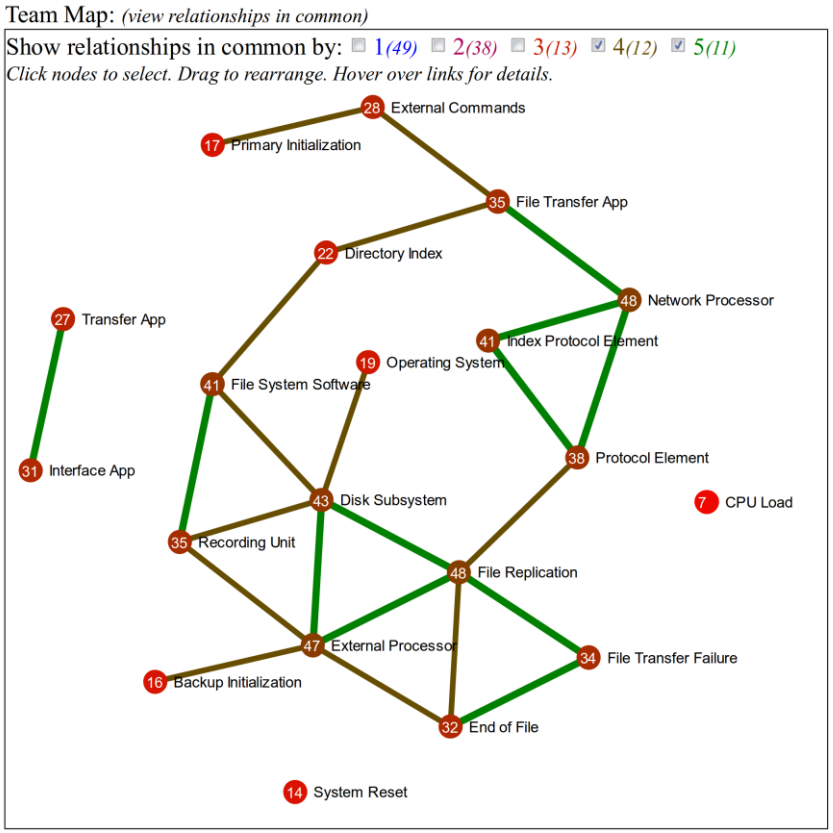


Figure 4.10. Result Visualization Tool, Team Map View

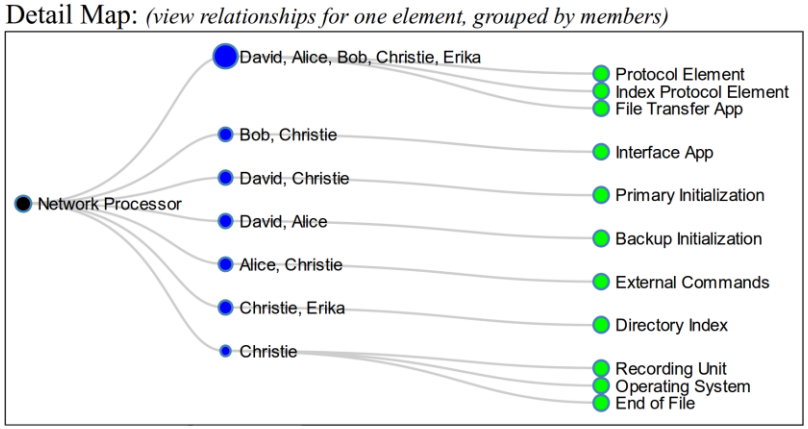


Figure 4.11. Result Visualization Tool, Detail Map View

4.3.1.8 Member Checking

After analysis was complete, I summarized my findings in a bulleted list in an e-mail and asked each of the participants to tell me whether they agreed with them or not and to provide any additional comments that they wished. This served as a check to verify that I correctly interpreted their comments during the group interview and served as an independent evaluation of the findings.

4.3.1.9 Group Interviews

The group interviews were semi-structured and customized for each team based on team's ratings questionnaire results. Each team was asked about approximately 3 each of the highest and lowest ranking concepts according to the $\overline{C_v}$ measure for the entire team, as time allowed. The team was also asked about concepts that changed significantly in rank between the two questionnaires. The detail view of the result visualization tool was used to identify team members sharing specific relationships among their PFNets. Questions were constructed by asking how these members understood these relationships and asking other members if they agreed or how they differed. Participants were also asked about difficulties they experienced in their project and any relationship between difficulties and the ranked concepts. At the end of the interview, teams were asked to provide their opinions on the value of using the SUM-ST method as a regular part of their work.

Group interviews were recorded, transcribed, and coded to find themes addressing the research questions.

4.3.2 Implementation

This section explains how the study was conducted. This consists of the study settings, the details of the data collection activities, and the analysis conducted.

4.3.2.1 Study Setting

I recruited industrial software development teams through the network of my personal contacts and the contacts of others in my research group. The only incentive offered was the chance to improve shared understanding within the team. Two industrial teams from two different organizations agreed to participate. A third team initially agreed but withdrew after a loss of internal funding. All research activities took place at the teams' places of employment between March and June of 2015.

4.3.2.1.1 Team 1

Team 1 developed spacecraft control software for a subcontractor to NASA. This was one of many software teams on the project, which includes other types of teams, such as various hardware teams, mission science teams, etc. This particular team had formed about a year prior to the start of the study but most team members had worked together on previous missions. The median of the pairwise experience reported on the demographic and interaction questionnaire was 3-4 years. Prior to the study period the team focused on creating the software development and test environment on the custom hardware and feature development had been in progress for about three months.

The team followed a formal waterfall-style process. The high-level software design specifications had been finalized and the team was working to implement them. The

developers had considerable autonomy for the low-level design and implementation choices. Developers had “ownership” over their assigned modules and the software architecture intentionally minimizes dependence on lower-level shared libraries. Team members collaborated as needed and there were few standing meetings. All new code was reviewed by other team members.

Communication in team 1 was primarily in unscheduled, face-to-face meetings. This was consistent with observations that showed group meetings were avoided and individuals were expected to take initiative and work together as needed.

The team was motivated to participate by the opportunity to discover faults in their design. They believed they had a strong process, communicated well, and did not suffer from problems with shared understanding. However, they were very concerned about any potential problem that could lead to a failure during spaceflight. The team leader was motivated to participate by the potential of a new way to discover problems and he approached his management for approval of his team’s participation. Although management gave permission, team members agreed to participate individually and did so on personal time (their lunch hour.)

The team leader recruited 4 of the other 6 developers to participate. All 5 completed the study. All participants were full-time employees, co-located, dedicated to this project, and had prior experience developing similar software. The participants are summarized in Table 4.8.

Table 4.8. Team 1 Participants

Participant	Role	Age	Gender	Experience (years)
P1.1	Leader/Developer	45-54	Male	15+
P1.2	Developer	35-44	Male	8-15
P1.3	Developer	*	*	4-7
P1.4	Developer	55-64	Male	15+
P1.5	Developer	55-64	Male	15+

* This participant declined to answer these questions on the demographic questionnaire but presented as a male 25-34 years old.

4.3.2.1.2 Team 2

Team 2 developed complex scheduling software for a large governmental organization. They were a temporary team formed from a larger pool of software development staff that developed and maintained a variety of related administrative support software. The scheduling software had existed for about 12 years and was undergoing a significant maintenance effort to update both the features and the underlying technology. The team was formed about one year prior to the study period.

The team followed a strict Scrum methodology with a two-week iteration cycle. (Scrum is an agile software development methodology that emphasizes self-organizing teams and develops work in short cycles called *sprints*.) Agile development was relatively new to the organization and this was the first agile project for many in the team. The team began each sprint with a planning meeting to select the features to be implemented during the sprint. At this meeting, the team members prioritized the unimplemented features, chose the features to implement, estimated the effort, scoped the work to fit within the two-week period, and made a commitment as a team to deliver the work on time. The sprint concluded with a

demonstration to representatives of the users and a retrospective meeting. The purpose of the retrospective meeting was to discuss process-related problems experienced during the sprint and to make necessary changes for the next sprint. The team also had a daily stand-up meeting (a short meeting to make everyone aware of each team member's current status.)

Communication in team 2 was primarily scheduled group meetings and scheduled face-to-face meetings. This was consistent with their Scrum process.

The team's management was motivated to participate by the opportunity to better understand the effects of a major reorganization effort that was still in progress. The reorganization restructured teams and introduced agile development practices. Management was concerned about the impact this had on the knowledge distribution within the teams and the potential for poor shared understanding as a result. They selected this team to participate because it was established in the new working style, not because they believed it was experiencing problems. Team members were not obligated to participate. Activities took place during normal working hours.

Of the 10 team members invited, one (a subject matter expert) declined to participate. Two others (a subject matter expert and a business analyst) withdrew after completing the demographic and interaction questionnaire. The participants were a mixture of employees and dedicated contractors and divided time between this project and other software maintenance activities. The participants consisted of 1 team leader/developer, 3 developers, 2 testers, and 1 Scrum master. (In the Scrum method, the Scrum master is a facilitator for the software development process.) The

developers and Scrum master worked at the primary location and the two testers worked at a remote location. They communicate by teleconferencing, desktop sharing, and some remote video. Prior experience with the software under development ranged from approximately 2 years to 12 years. Some members had worked together for years but others were new to the project. The median of the pairwise experience reported on the demographic and interaction questionnaire was 6-11 months. Compared to team 1, team 2 had much less experience working together. The team 2 participants are summarized in Table 4.9.

Table 4.9. Team 2 Participants

Participant	Role	Age	Gender	Experience (years)
P2.1	Leader/Developer	45-54	Male	15+
P2.2	Developer	25-34	Male	8-15
P2.3	Developer	35-44	Male	8-15
P2.4	Developer	25-34	Male	8-15
P2.5	Scrum Master	25-34	Male	4-7
P2.6	Tester	35-44	Female	8-15
P2.7	Tester	25-34	Male	NA

4.3.2.2 Data Collection

After obtaining approval from each organization to participate, I held a kick-off meeting with each team to explain the research activities. Informed consent was obtained from each participant and they were sent an e-mail invitation to take the demographic and interaction questionnaire online.

I asked each team leader for the opportunity to observe team meetings that had the potential to include discussion of design issues. Team #1 invited me to observe some regular weekly status meetings. I observed three status meetings for a total of three

hours. Team #2 invited me to observe bi-weekly sprint retrospective and sprint planning meetings. The purpose of the retrospective meeting is to discuss process-related problems experienced in the previous sprint and make necessary changes. The purpose of the sprint planning meeting is to select features to be implemented in the next sprint. I observed 3 retrospective and 3 sprint planning meetings for a total of 10 hours.

The concepts for each team were selected by the team's leader. For each team, I met with the leader, explained the concept selection process (described in section 3.3.1) and gave them example concepts from the pilot study. I asked the leader not to discuss the concepts with their teammates (outside of normal work) because those discussions could resolve differences in understanding prior to measurement. Each leader was asked to identify 5 concepts initially. I reviewed these for compliance with the guidelines, gave feedback, and then asked the leader to complete a list of between 20 and 22 concepts.

The leader for team 1 was interested in the possibility of the SUM-ST technique to identify weaknesses in shared understanding that may lead to a critical failure on the spacecraft. Guided by a concern for how exceptional conditions are handled when multiple modules are involved, he chose to investigate concepts related to failures in data transfers between modules. A side-effect of this approach was that the concepts selected were high-level modules and not detailed design elements as were used in the pilot study. I suggested using more specific design elements (such as the fields of a message being passed between modules) but the leader preferred to start with a more holistic view. Although more abstract than expected, the concepts met the

selection guidelines. After careful consideration, my advisor and I decided to accept these concepts rather than risk biasing the study results through additional researcher influence. The preliminary list contained 21 concepts.

A list of team 1's concepts is shown in Table 5.2 on page 134. The text descriptions are limited, and some have been slightly edited, to preserve the anonymity of the team.

Although the leader for team 2 did not mention a specific selection strategy, the concepts selected were also high-level modules. The team's software design follows a Service Oriented Architecture pattern and is constructed from a large number of small services. Most of the concepts chosen were specific service names. Again, the concepts met the predefined guidelines and were accepted. The preliminary list contained 27 concepts, 5 more than requested.

A list of team 2's concepts is shown in Table 5.3 on page 134. Again, the text descriptions are limited and edited to preserve the anonymity of the team.

The next step was to validate the concepts with another member of the team. Each leader nominated another member of their team for this purpose. I interviewed this person individually about the concepts in their team's list. I presented each concept and asked the validator to define the term and verify it against the guidelines. I asked the validator not to discuss the concepts with teammates outside of normal work. Afterward, I presented the results of the validation to the team leader and asked them to make appropriate edits to the list. The validation interviews were audio recorded.

For team 1, a few concepts were unclear or had ambiguous phrasing and were reworded. Two concepts were dropped. One was dropped because the validator felt it was no longer relevant to the current design. (This, by itself, suggests a lack of shared understanding within the team about this concept.) The other was dropped because it was identified to be a design goal rather than an element of the design. One of the dropped concepts was replaced with a new concept that was not validated. The final list contained 20 concepts.

For team 2, all of the concepts were accepted as part of the validation. Because the initial list was too long, I asked the validator to suggest concepts to keep based on importance to the project and significance for collaboration. He selected 7 concepts to drop that he thought were less relevant to work the team planned to do in the next 2-8 weeks. I presented this list of 7 concepts (but not the rationale) to the leader to consider for final selection. He agreed with most, but not all, of this list, and made his own selection to cut the list to 21 concepts.

The first ratings questionnaire (Q1) for each team was scheduled as soon as their concept list was finalized. Participants were scheduled to take the questionnaire at their place of work while I was on-site to assist with any problems. Participants took the questionnaire at the same time without discussion.

Most members of team 1 took the first questionnaire in their offices but some took it on laptops in the conference room in which I was waiting. Although no issues were reported while I was on site, participant P1.4 sent me an e-mail afterward explaining he realized halfway through that he had probably misinterpreted concept C1.11 ‘FS

Flight System Software'. This issue was significant to team 1's results and is discussed in detail in section 5.2.2.

Team 2 reported no issues taking the first questionnaire.

The second ratings questionnaire (Q2) was scheduled approximately two weeks later and was conducted similarly to the first questionnaire except that I was not present. No issues were reported.

I created an interview guide as described in section 4.3.1.8. I used the result visualization tool to compute the list of concepts ranked by $\overline{C_v}$ for each questionnaire and selected high ranking concepts, low ranking concepts, and concepts that changed rank significantly between the two questionnaires. The interview guide for team 2 is shown in Appendix B, section C.3. Each group interview was scheduled between 2 and 3 weeks after the team completed their second ratings questionnaire and was conducted at the team's place of employment.

For team 1, the interview was held in a conference room and the result visualization tool was projected on a screen. All team members participated. For team 2, the interview was held in the Scrum meeting room and the result visualization tool was projected on the wall. Participant P2.6 (remote) joined by telephone and desktop sharing but participant P2.7 (also remote) did not join the interview.

As I conducted the group interview, I used the result visualization tool to display relationships relevant to the current question. Discussion was guided primarily by the detail map view. I showed the detail map for a selected concept and asked

participants to explain why they thought the concepts were related or not and explored any differences in statements among the team members. When differences in understanding were identified, I asked team members whether those differences would have an impact on the project in terms of quality, cost, or schedule. As the interview progressed and the team members gained an understanding of how to interpret the visualizations, I let them request results that interested them. This was valuable because they could interpret the results in the context of work assignments and collaborations that I was not aware of. At the end of the interview, the teams were asked to comment on whether the activity would be valuable as a software design technique. Each interview lasted approximately 75 minutes. The interviews were audio recorded and transcribed. The transcripts for team 1 and team 2 contained approximately 17,000 and 15,000 words respectively.

I held an additional group interview with team 2 to more carefully explore any associations between the team's results and the development activities for the next sprint. I first observed the team's sprint planning meeting and noted any discussion that included the team's concepts. At the conclusion of the meeting, I interviewed the group and asked how the discussion to develop the plan related to the ranked results of concepts from the ratings questionnaires.

In January of 2016, after all analysis was complete, I asked each participant to review a summary of the study's conclusions. Responses were received from 4 of the 12 participants: P1.1, P1.4, P2.1, and P2.5.

4.3.2.3 Analysis

I transcribed and coded the group interviews according to a pre-defined code list developed from the research questions while also open coding for negative cases and emergent themes using MAXQDA (VERBI GmbH 2013). The code list is shown in Table 4.10.

Part II of the questionnaire asked participants to provide descriptions of the relationships represented by the edges in their PFNets. The objective was to find out if different participants interpret relationships in the same way. Questions were semi-structured asking participants to enter a linking phrase to make a sentence relating the two concepts. A review of the data found that this structured approach helped make the answers easier to interpret, but there was still too much variability to allow coding with high reliability. The types of relationships between technical elements of a software design are diverse and can be described in different levels of abstraction and using project specific jargon, making researcher interpretation difficult. To help explain this, some examples are described below.

Table 4.10. Main Study Code List

Code	Definition
Explanation of rating	One participant's explanation, justification, or rationale for a relationship rating.
Unrelated	Described as unrelated.
Related	Described as related.
Terminology*	The rating was dependent on the term used.
Unknown*	The relationship was unknown prior to discussion.
Explanation of comparison	How one person understands a relationship compared to how another person understands a relationship.
Similar understanding	Agreement with other's understanding.
Different understanding	Disagreement with other's understanding.
Lack of understanding	No prior understanding.
Validity of similarity score*	How the team's shared understanding compares to the similarity score.
Similarity valid*	A high similarity and shared understanding.
Similarity not valid*	A high similarity but different understanding.
Difference valid*	A low similarity and different understanding.
Difference not valid*	A low similarity but shared understanding.
Context dependent*	Rating depends on contextual interpretation.
Impact of difference	Description of the impact of a difference in understanding on the software development process.
No impact	Difference would not result in a problem.
Has impact	Difference would result in a problem.
SUM-ST Utility	The value of SUM-ST as a software development practice.
Useful	Would provide value.
Not useful	Would not provide value.
Difficulties	Difficulties experienced by the team in the software development process.
Integration problem	A problem integrating work of two or more people.
Rework	A problem required additional work in an area thought to be complete.
Quality	A problem that degraded the system quality.
Cost	A problem that added additional cost.
Discussion of terminology*	Discussion of terminology and shared understanding.
Design process*	Discussion of the design process.
Recency*	Discussion of how recently work had been done and how this relates to understanding.
Discussion of ranking*	Discussion of concept ranking.
Role-based*	Discussion how understanding depends on role.
Triggered Learning*	A team member learns something while discussing concept differences.

*Added while coding

This first example shows responses that were easily interpretable as similar descriptions of the same relationship:

Participant	Description (C1.7 vs. C1.8)
P1.1	'Command Loads on Backup Spare' **is the same as** 'Command Loads on Hot Spare'
P1.2	'Command Loads on Backup Spare' **are similar to** 'Command Loads on Hot Spare'
P1.3	'Command Loads on Hot Spare' **are a similar process to** 'Command Loads on Backup Spare'
P1.4	'Command Loads on Backup Spare' **are basically the same as** 'Command Loads on Hot Spare'

The following example was difficult to interpret because, without further understanding of the concepts, it was not possible to know if the descriptions were conceptually similar or not:

Participant	Description (C2.9 vs. C2.10)
P2.1	'Person Conflict Service' **queries** 'Person Service'
P2.2	'Person Conflict Service' **is based off data from the** 'Person Service'
P2.3	'Person Conflict Service' **uses** 'Person Service'
P2.4	'Person Conflict Service' **uses** 'Person Service'
P2.5	'Person Service' **broad details of each name pair in** 'Person Conflict Service'

Using a semi-structured approach did not resolve the problems with inter-rater reliability and so no results are reported about the participant's interpretations of the concept relationships.

Chapter 5: Results

The results are organized under two main sections. The first section summarizes responses to the ratings questionnaires, and the measures of shared understanding calculated from them, without comparing them to other data. The second section presents the results from the group interviews, including comparisons to the measures of shared understanding, and contains the primary data to answer the research questions.

5.1 Ratings Questionnaire Results

This section presents the data from the ratings questionnaires without comparing it to other data. This begins with a summary of the ratings themselves, especially the 'No answer' responses. This is followed by the measures of shared understanding calculated from the ratings and a comparison of the results from the first and second questionnaires.

5.1.1 Relatedness Ratings

I reviewed the raw relatedness ratings looking for potential problems, especially with respect to the new response categories of 'Not at all related' and 'No answer' (NA). This review identified some interesting results that I clarified through checking with some of the participants, as described below.

Table 5.1 shows the frequency of responses to the relatedness rating portion of the questionnaires. Team 1 had 20 concepts and 190 paired comparisons. Team 2 had 21 concepts and 210 paired comparisons. The most frequent response was 'Not at all

Table 5.1. Questionnaire Part I Response Frequencies

		Questionnaire 1 Rating										Questionnaire 2 Rating									
		X	**1**	**2**	**3**	**4**	**5**	**NA1**	**NA2**	**NA3**	**NA4**	**X**	**1**	**2**	**3**	**4**	**5**	**NA1**	**NA2**	**NA3**	**NA4**
Team 1[a]	**P1.1**	54	19	17	14	26	60	0	0	0	0	30	30	25	20	30	55	0	0	0	0
	P1.2	62	25	12	23	24	26	0	8	10	0	76	24	6	14	13	21	0	18	18	0
	P1.3	22	11	9	15	26	106	1	0	0	0	17	12	26	29	33	73	0	0	0	0
	P1.4	33	62	13	21	36	24	1	0	0	0	69	34	6	12	31	38	0	0	0	0
	P1.5	64	58	3	18	0	46	0	0	0	1	65	30	3	32	0	60	0	0	0	0
Team 2[b]	**P2.1**	92	9	27	46	11	25	0	0	0	0	87	32	12	45	9	25	0	0	0	0
	P2.2	51	58	49	32	14	5	0	0	0	1	88	68	30	14	7	3	0	0	0	0
	P2.3	142	16	17	15	9	11	0	0	0	0	124	31	44	0	1	1	0	6	3	0
	P2.4	79	8	11	21	26	13	3	20	25	4	98	19	25	38	19	7	0	2	2	0
	P2.5	29	49	52	44	30	6	0	0	0	0	84	23	44	31	28	0	0	0	0	0
	P2.6	0	15	44	38	43	23	0	0	0	47	0	0	3	32	6	2	0	0	0	167
	P2.7	0	0	18	131	55	6	0	0	0	0	0	0	1	111	97	1	0	0	0	0
	Total	**628**	**330**	**272**	**418**	**300**	**351**	**5**	**28**	**35**	**53**	**738**	**303**	**225**	**378**	**274**	**286**	**0**	**26**	**23**	**167**
		26%	**14%**	**11%**	**17%**	**12%**	**15%**	**0%**	**1%**	**1%**	**2%**	**30%**	**13%**	**9%**	**16%**	**11%**	**12%**	**0%**	**1%**	**1%**	**7%**
	Rank	**1**	**4**	**6**	**2**	**5**	**3**	**10**	**9**	**8**	**7**	**1**	**3**	**6**	**2**	**5**	**4**	**10**	**8**	**9**	**7**

[a] Out of 190 questions total

[b] Out of 210 questions total

Key:

X	Not at all related
1-5	Weakly to strongly related
NA1	Unrelatable/nonsense
NA2	I don't understand the first term
NA3	I don't understand the second term
NA4	Other reason

related' and the most frequent relatedness value was '3', the middle of the scale.

The responses of the two testers on team 2, P2.6 and P2.7, stand out as different from the others. They strongly favored the middle of the scale and were the only ones not to rate anything as unrelated ('X') on either questionnaire. Participant P2.7 used the ratings '3' and '4' almost exclusively on the second questionnaire. This raised the question of whether this person took the exercise seriously.

I asked participant P2.7 about his responses by e-mail prior to the group interview. He explained that he viewed everything as related because, in his words, if one function doesn't work, the other won't work, and that he does not need to know which functions are being called to perform his tests. Based on this response, there is evidence that he did attempt to answer faithfully but lacked sufficient understanding of the non-visible parts of the system to discriminate among them.

The other interesting results relate to the 'No Answer' (NA) responses. I expected that the concept validation step would prevent use of a NA response, however, they represented over 5% of the responses, primarily by participants P1.2, P2.4, and P2.6.

Participant P2.6 responded 'No Answer – Other reason' frequently. On the first questionnaire, the answers were clustered around two of the concepts but were inconsistent. On the second questionnaire, she consistently answered 'No answer - Other reason' for 15 of the 21 concepts resulting in 167 responses. I asked her about her responses by e-mail prior to the group interview. Her explanation echoed that of participant P2.7, the other tester in her team. She said that, in her role as a tester, she was familiar with the terms but didn't know enough about the concepts to judge

whether or not they were related. For the first questionnaire, she did some research to try to judge pairs but, on the second questionnaire, she followed the instructions to answer quickly and chose 'No answer - Other reason' as her best response. Therefore, the answers to the first questionnaire are, in some cases, the result of a different psychological process and not a description of her knowledge structures. The answers to the second questionnaire should be more accurate, if incomplete.

Participant P1.2 answered 'I don't understand the first/second term' for all comparisons involving concept C1.20 'Processing loading' on the first questionnaire and both C1.20 and C1.11 'FS Flight System Software' on the second questionnaire. During the group interview, this participant said that they thought these terms were ambiguous or wrong. This aligns with the report by participant P1.4 about concept C1.11 being ambiguous after taking the first questionnaire. These concepts are central to this team's discussion in the group interview (discussed in section 5.1.2.)

Participant P2.4 answered 'I don't understand the first/second term' for the majority of comparisons involving 4 concepts on the first questionnaire but not on the second questionnaire. When asked about this, the participant said that he was originally unfamiliar with the 4 concepts but learned about them during his assigned work in the interval between taking the first and second questionnaires.

The 'Unrelateable/nonsense' response was used only 5 times and only on the first questionnaire. In team 1, this was used by two different participants for the same concept, but not the same comparison. The other three instances were from one participant in team 2 and were mixed among other NA responses for the same

concept, possibly indicating the participant wasn't sure which the most appropriate response was.

These cases reveal that validation by only one other member of the team is not sufficient to assure the terms are clear to everyone in the team and that the NA response can detect problems with terms and understanding. In the case of P2.4, this also happened to detect a case of learning and, possibly, improved shared understanding within the team.

5.1.2 Measures of Shared Understanding

This section describes the results of applying Pathfinder and the neighborhood comparison measurement $\overline{C_v}$ to the relatedness ratings.

First, there is a question about which team members to include in the analysis. The review of the ratings frequencies found that the two testers from team 2 (P2.6 and P2.7) lacked a meaningful understanding of many of the concepts in the team's list. It could be argued that, as testers, these team members do not collaborate about the design of the software and should be dropped from further analysis. However, from observation of the team's meetings it is clear that participant P2.6 holds a great deal of domain knowledge and understands the complex interaction of the software features better than many of the developers. (Tester P2.7 has less experience and did not participate much in meetings so I do not know his contributions.) Under Scrum, the entire team participates in sprint planning and this process depends very much on the team's shared understanding. Therefore, the testers do collaborate about the design and should be included. We expect, though, that these known differences in

understanding related to role should stand out in measures of knowledge structure similarity that include these team members.

However, including the testers in team 2 makes it more difficult to compare data and results with team 1. Therefore, the numerical results for team 2 are presented twice, once for the full team and again with participants P2.6 and P2.7 dropped, referred to as "developers only". (The Scrum master is included as a developer based on his demonstrated knowledge of programming concepts at the team meetings.)

The concept similarity $\overline{C_v}$ and team similarity $\overline{C}$ were calculated for each questionnaire for each team and are shown in tables 5.2, 5.3, and 5.4. These tables formed the basis for concepts to investigate in the group interviews. (The selected concepts are indicated in the table.)

For team 1, $\overline{C_v}$ ranged from 0.09 to 0.54. The average team similarity $\overline{C}$ was 0.33 and 0.32 for questionnaires 1 and 2 respectively.

For team 2, $\overline{C_v}$ ranged from 0.04 to 0.33 (0.05 to 0.52 for developers only.) The average team similarity $\overline{C}$ was 0.17 and 0.19 for questionnaires 1 and 2 respectively (0.21 and 0.25 for developers only). The measure of shared understanding for the full team 2 including the testers is lower than for just the developers. This is expected based on the differences in the testers' ratings as discussed in the previous section.

Table 5.2. Team 1's Concepts Ranked by Questionnaire 1

Concept Label	Text[a]	Q1 Rank	$\overline{C_v}$	Q2 Rank	$\overline{C_v}$	Difference Rank	$\overline{C_v}$
C1.10[b]	Ground software uplink	1	0.54	5	0.41	4	-0.14
C1.1[b]	Single Board Computer (SBC) App	2	0.50	1	0.49	-1	-0.01
C1.19	*	3	0.44	3	0.43	0	-0.01
C1.3	CFDP Metadata Protocol Data Unit	4	0.43	4	0.41	0	-0.02
C1.14	*	5	0.42	7	0.40	2	-0.02
C1.9	*	6	0.40	12	0.32	6	-0.09
C1.12[c]	*	7	0.37	15	0.26	8	-0.11
C1.2[c]	*	8	0.36	2	0.46	-6	0.10
C1.5	*	9	0.36	11	0.33	2	-0.03
C1.18	*	10	0.35	10	0.33	0	-0.02
C1.17	Record (REC) App	11	0.34	6	0.40	-5	0.06
C1.4	*	12	0.32	13	0.31	1	-0.01
C1.6[c]	Spacecraft Interface (SI) App	13	0.31	19	0.16	6	-0.15
C1.13	*	14	0.30	8	0.37	-6	0.06
C1.11	FS Flight System Software	15	0.26	14	0.29	-1	0.04
C1.8	Command Loads on Backup Spare	16	0.24	16	0.21	0	-0.03
C1.7	Command Loads on Hot Spare	17	0.18	17	0.18	0	0.00
C1.16[b]	*	18	0.17	20	0.11	2	-0.06
C1.20	Processing loading	19	0.15	18	0.17	-1	0.02
C1.15[bc]	Prime software watchdog reset	20	0.09	9	0.36	-11	0.27
	Team $\overline{C}$		0.33		0.32		

[a] Some concepts have been edited or masked completely to preserve the anonymity of the team
[b] Selected for group interview based on high/low rank
[c] Selected for group interview based on change in rank

Table 5.3. Team 2 Concepts Ranked by Questionnaire 1

Concept Label	Text[a]	Q1 Rank	$\overline{C_v}$	Q2 Rank	$\overline{C_v}$	Difference Rank	$\overline{C_v}$
C2.14[bc]	Person Cards Service	1	0.33	20	0.13	19	-0.19
C2.20[b]	Meetings UI	2	0.30	5	0.23	3	-0.07
C2.13[c]	*	3	0.25	15	0.16	12	-0.09
C2.17	*	4	0.24	6	0.19	2	-0.05
C2.11	*	5	0.22	10	0.18	5	-0.04
C2.15	*	6	0.21	4	0.25	-2	0.04
C2.9	Person Service	7	0.20	13	0.16	6	-0.04
C2.1	Assignment Setup Service	8	0.20	3	0.28	-5	0.09
C2.10	Person Conflict Service	9	0.19	2	0.31	-7	0.13
C2.19	*	10	0.17	9	0.18	-1	0.01
C2.8	Office Setup Service	11	0.15	7	0.19	-4	0.03
C2.12[c]	Meetings Service	12	0.15	1	0.32	-11	0.17
C2.3	Core Case Service	13	0.14	11	0.16	-2	0.02
C2.16	*	14	0.13	12	0.16	-2	0.03
C2.7	*	15	0.12	16	0.15	1	0.04
C2.18	*	16	0.11	18	0.15	2	0.04
C2.21	Edit Auto Assign	17	0.10	19	0.15	2	0.05
C2.6[c]	*	18	0.09	8	0.18	-10	0.09
C2.5	*	19	0.08	14	0.16	-5	0.07
C2.2[b]	*	20	0.07	21	0.10	1	0.03
C2.4[b]	Case Person Service	21	0.04	17	0.15	-4	0.11
	Team $\overline{C}$		0.17		0.19		

[a] Some concepts have been edited or masked completely to preserve the anonymity of the team
[b] Selected for group interview based on high/low rank
[c] Selected for group interview based on change in rank

Table 5.4. Team 2, Developers Only, Concepts Ranked by Questionnaire 1

Concept Label	Text[a]	Q1 Rank	$\overline{C_v}$	Q2 Rank	$\overline{C_v}$	Difference Rank	$\overline{C_v}$
C2.20	Meetings UI	1	0.41	4	0.32	3	-0.09
C2.14	Person Cards Service	2	0.37	15	0.19	13	-0.19
C2.3	Core Case Service	3	0.30	6	0.30	3	0.00
C2.10	Person Conflict Service	4	0.30	1	0.52	-3	0.22
C2.15	*	5	0.28	5	0.32	0	0.04
C2.9	Person Service	6	0.28	10	0.25	4	-0.03
C2.11	*	7	0.26	19	0.16	12	-0.10
C2.13	*	8	0.25	20	0.16	12	-0.09
C2.7	*	9	0.25	8	0.27	-1	0.02
C2.18	*	10	0.21	21	0.12	11	-0.09
C2.12	Meetings Service	11	0.21	2	0.45	-9	0.24
C2.19	*	12	0.21	17	0.17	5	-0.03
C2.1	Assignment Setup Service	13	0.20	3	0.38	-10	0.18
C2.17	*	14	0.20	14	0.19	0	-0.01
C2.2	*	15	0.14	18	0.17	3	0.03
C2.8	Office Setup Service	16	0.14	13	0.19	-3	0.05
C2.5	*	17	0.13	7	0.30	-10	0.17
C2.16	*	18	0.11	12	0.19	-6	0.08
C2.21	Edit Auto Assign	19	0.10	11	0.21	-8	0.12
C2.6	*	20	0.08	16	0.18	-4	0.10
C2.4	Case Person Service	21	0.05	9	0.25	-12	0.20
	Team $\overline{C}$		0.21		0.25		

[a] Some concepts have been edited or masked completely to preserve the anonymity of the team

Although the range of $\overline{C_v}$ was similar between questionnaire 1 and questionnaire 2, some individual concepts shifted significantly in rank. For team 1, C1.15 'Prime software watchdog reset' rose in rank from 20 to 9. For team 2, C2.14 'Person Cards Service' dropped in rank from 1 to 20. Concepts that shifted in rank were also selected for the group interview and are discussed in section 5.2.2.1.

There were instances where every relationship involving a concept was marked as either 'Not at all related' or 'No answer', resulting in these concepts being disconnected from the rest of the PFNet (see section 4.3.1.6.) Most of these cases are the same NA issues discussed in the previous section.

Table 5.5. Disconnected Concepts

Participant	Questionnaire	Concepts
P1.2	Q1	C1.20
	Q2	C1.11, C1.15, C1.20
P2.4	Q1	C2.5, C2.7, C2.21
	Q2	C2.21
P2.6	Q2	C2.3, C2.5, C2.9

5.1.3 Changes Over Time

The ratings questionnaire was given twice to allow investigation of how shared understanding changes over time and to be able to associate any changes in measurements with difficulties experienced by the team in the interval between questionnaires. This section reports two different comparisons. The first comparison is the change in team-level similarity between the first and second questionnaires. The second comparison is the similarity between the two PFNets for each individual.

Table 5.6 shows the team level similarity $\overline{C}$ for each questionnaire. According to this measure, team 1 had little change in shared understanding between the two questionnaires but team 2 had a slight increase and this was more pronounced among the subgroup of developers. This suggested investigating whether this increase could be attributed to participant P2.4 because he had reported learning about concepts between the two questionnaires. To test this, I approximated each individual's

Table 5.6. Team Similarity Questionnaire 1 vs. Questionnaire 2

Team	Q1 similarity ($\overline{C}$)	Q2 similarity ($\overline{C}$)	Difference
1	0.33	0.32	-0.01
2	0.17	0.19	0.02
2 (Dev. only)	0.21	0.25	0.04

contribution to the overall change in team similarity by recalculating $\overline{C}$ with each individual excluded (see Appendix B, Table C.5). When participant P2.4 was excluded, there was no change in team similarity whereas all subgroups containing P2.4 showed an increase in similarity. This supports the notion that the team's increase in $\overline{C}$ is due largely to changes in participant P2.4's knowledge structures and is consistent with his statement that his understanding had changed to be more in agreement with his teammates at the time of the second questionnaire.

Table 5.7 shows the similarity between each member's own PFNets from the two questionnaires as calculated by the similarity measure C. The range across all teams was 0.20 to 0.57. The median was 0.55 for team 1 and 0.23 for team 2.

Table 5.7. Self-Similarity Questionnaire 1 vs. Questionnaire 2

Team	Participant	C
1	P1.1	0.55
	P1.2	0.57
	P1.3	0.55
	P1.4	0.43
	P1.5	0.42
2	P2.1	0.55
	P2.2	0.35
	P2.3	0.22
	P2.4	0.20
	P2.5	0.43
	P2.6	0.20
	P2.7	0.23

The highest values, a little above mid-scale, were lower than I expected. Each team's concepts represented core, high-level modules that did not experience design revision during the period of study. The leaders (P1.1 and P2.1), in particular, played a significant role in the original design of the concepts and they selected these concepts

according to their perceived importance to the success of the team. Possible explanations for these “low values” are that my interpretation of the values is simply incorrect and that mid-scale represents significant agreement, that the measures are influenced by unrelated factors that give low values even when there is agreement, or that the team members understanding of the concepts is not as stable as I had expected.

The lowest values, for participants P2.4 and P2.6, may be related to known changes between questionnaires. Participant P2.4 reported learning about concepts between questionnaires, discussed previously. Participant P2.6 changed her answer strategy for concepts she did not understand between questionnaires marking many more NA on questionnaire 2.

5.2 Group Interview Results

This section presents the group interview results. This section begins with some example visualizations from the data. The next three sections present the primary results to answer the research questions. Of these, the first presents how similarities and differences in the ratings are associated with team member’s understanding (RQ1.) The next section presents the importance of the identified differences in shared understanding (RQ2). The final section presents the teams’ feedback on the usefulness of SUM-ST to improve the software design process (RQ3.)

5.2.1 Visualizations

The result visualization tool played a significant role in the group interviews. This section begins with some example visualizations to help explain how they were used.

The primary view used was the detail map. Figure 5.1 shows the detail view for team 1's highest and lowest ranking concepts from questionnaire 1. This shows that C1.10 'Ground software uplink' has three edges shared by all 5 team members and this contributes significantly to the relatively large $\overline{C_v}$. In contrast, C1.15 'Prime software watchdog reset' has smaller groups sharing fewer edges and more edges unique to individuals resulting in a low $\overline{C_v}$. Note that the highest ranking concept, C1.10, has 5 edges unique to P1.3 and the lowest ranking concept, C1.15, has one subgroup

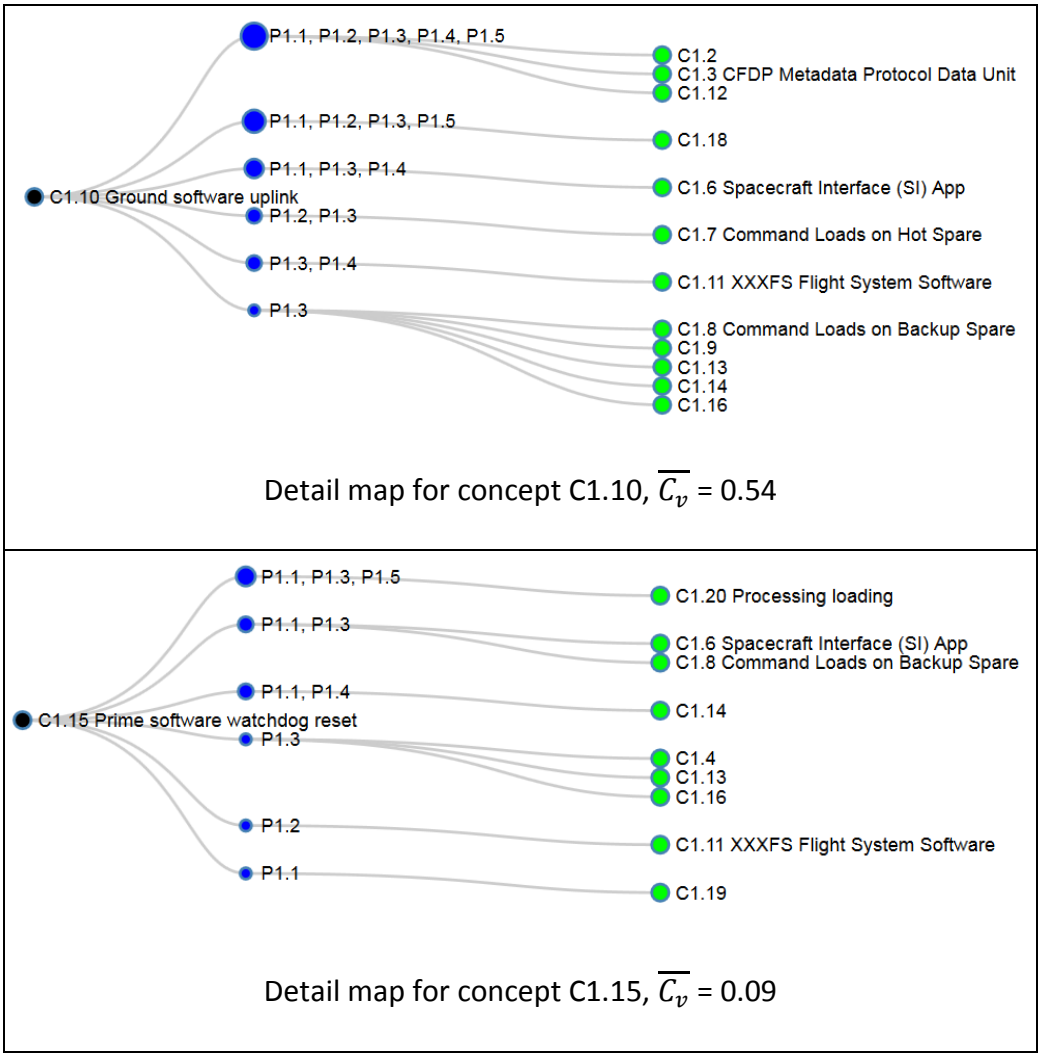


Figure 5.1. Detail View for Team 1, Q1, C1.10 and C1.15

containing over half of the members (for concept C1.20). Therefore, a high $\overline{C_v}$ does not indicate a general agreement among all members nor does a low $\overline{C_v}$ does imply a lack of similarity among all members. It only indicates the proportion of edges shared between pairs of members.

In the group interview, questions about C1.15 followed the largest group's relationship to C1.20 'Processing loading'. Participants P1.1, P1.3, and P1.5 were asked how they thought the two concepts were related and whether they agreed with each other. The others, participants P1.2 and P1.4, were asked if they agreed with them or not.

In practice, the team map was not used much. The concepts to investigate were already identified in the interview guide and discussing them took most of the available time. Perhaps the most valuable insight from the team map was the ability to identify concepts that are strongly connected vs. those that are weakly connected. Figure 5.2 shows the edges in common to 4 or more members of team 2. This shows that concepts C2.11, C2.12 'Meetings Service', and C2.20 'Meetings UI' are closely related to each other, but not the other concepts, which was confirmed in the group interview.

5.2.2 Similarity Measure and Shared Understanding

The group interviews were used to investigate the relationship between the concept ranking and how the team members understood the concepts. In most cases, a discussion of differences in the PFNets between participants revealed differences in how participants thought about those concepts. Usually, the difference was attributed

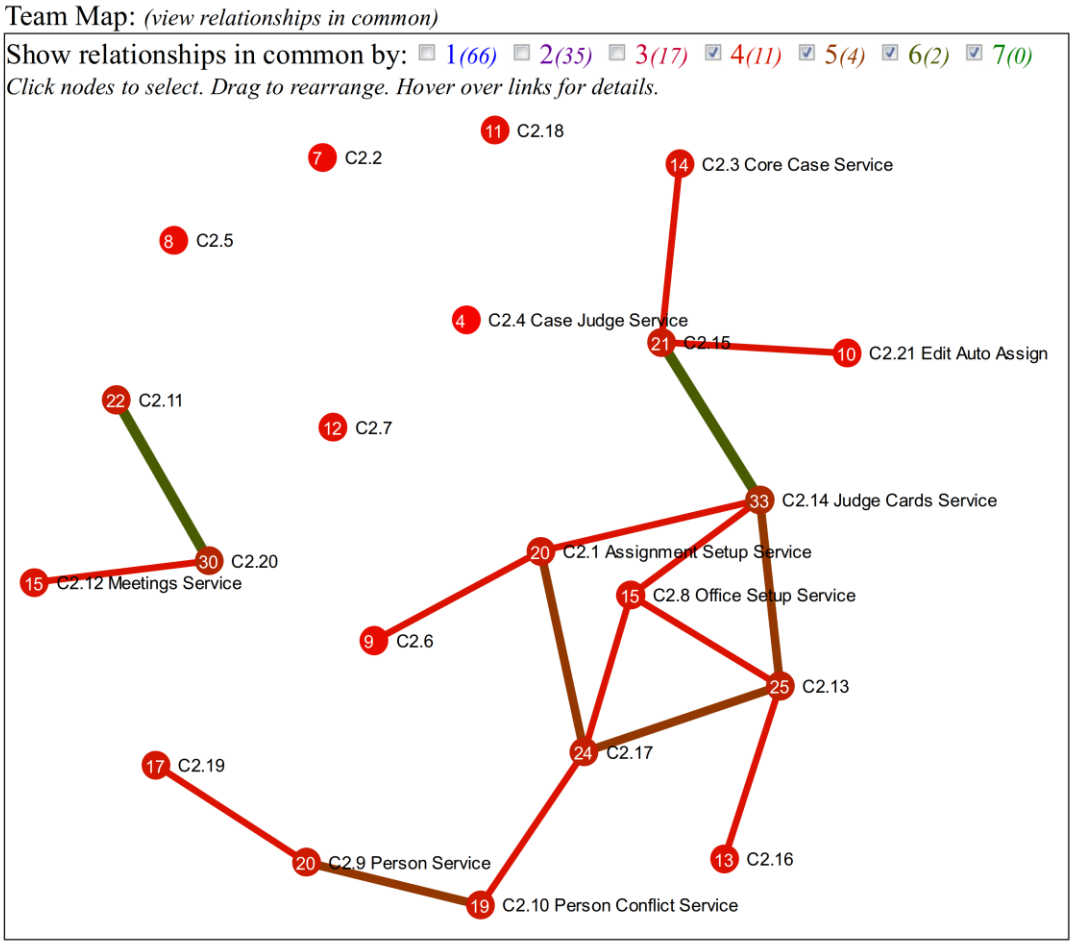


Figure 5.2. Team Map for Team 2, Questionnaire 1

to a lack of understanding rather than a conflict. Although more difficult to evaluate, similarities between PFNets were associated with agreement among participants' statements.

The exploration of similarities and differences during the group interviews was more complex in the main study than in the pilot study due to the larger number of team members participating. In the pilot study, most group interviews had 2 to 3 participants and the comparisons tended to be between individuals. In the main study, the interviews had 5 or 6 participants and the comparisons tended to be between

subgroups. As discussed in the previous section, even low ranking concepts contained subgroups holding edges in common, so it was possible to find similarities and differences among all concepts, independent of rank.

Team 1's discussion focused on three concepts that ranked low. Concepts C1.11 'FS Flight System Software', C1.20 'Processing loading', and C1.15 'Prime software watchdog reset' ranked 15, 19, and 20 out of 20, respectively. The team's discussion focused on the terms chosen for the concepts more than the concepts they represented, although this still reveals a difference in how the team understood the concepts.

Although terms were chosen by the leader, P1.1, and validated by participant P1.3, the edits made after validation were not re-validated. Concept C1.15 'Prime software watchdog reset' was not changed but C1.11 'FS Flight System Software' was originally a more generic term encompassing two modules and changed to be specific to one of those modules. Concept C1.20 'Processing loading' was added to replace a concept and was not, itself, validated. The view that these were bad terms was summarized by the following exchange:

> P1.2: Well, for the bottom three are ones that I thought were bad terms. Flash File System is not really Flash File System.
>
> P1.4: Yeah, yeah, that's the term that I had a problem with.
>
> P1.2: Software watchdog reset is ambiguous and processing loading is ambiguous or is, I don't know, silly. So, those three, I didn't like.

Discussion of the problems with C1.15 'Prime software watchdog reset' revealed that the leader was thinking of an older version of the design when he proposed the

concept. The concept was related to a system reset function that, when revised, was broken out into two separate functions. The leader was aware of the revision, but still thought in terms of the original design. In discussions, others in the team were unaware of the difference in intended meaning:

> P1.1: [P1.4]'s example was good about the prime software watchdog reset because that's really … we now have these multiple watchdogs, so the terminology I used there was misleading which leads me to … realize that when I talk about … the two different watchdogs, I have to make sure … that people understand which one I'm talking about.
>
> P1.4: And I didn't even realize that.
>
> P1.1: I wasn't even … didn't cross my mind either. I'm thinking SVC reset, what other one is there? And then [P1.5], of course, the other one.

Discussion of C1.11 'FS Flight System Software' revealed an error in the term which had mixed two similar acronyms based on '*Flight* System Software' and '*Flash* System Software'. Although not an official name for any module, team members were able to infer the meaning but without certainty. For example, the developer of that module (P1.4) interpreted it correctly but reported that it occurred to him halfway through the first questionnaire that he might have interpreted it incorrectly. I thought that the difficulty to detect the error might have reflected a difference in, or lack of, understanding within the team. When I asked the team about this, they all agreed that this did not represent a difference in understanding but was "just a simple mistake" preparing the concept list.

The concept C1.20 'Processing loading' was not a module name but a system resource: the amount of CPU load being used. An excessive CPU load could cause a

critical failure. The leader had mentally organized modules according to his perceived risk that they could cause such a failure and his relatedness ratings represented the risk of the module. “Failure”, however, was not part of the term. With this unstated, the other team members did not have enough context to relate the concept in the same way and this was reflected in the differences measured.

However, even though the other members of the team were aware of the risk and may have responded differently if the concept had been, perhaps ‘risk of failure due to processor loading’, they do not seem to have thought about the risk in the same way. The leader, P1.1, said:

> And in the case of processor loading, I was a little surprised too because my interpretation of processor loading is that you look at the applications you expect to have high processor loading, so that’s why I had SI high, because I know that’s a concern for us, and I’m not really concerned about, say, SBC having high processor loading. So, it was interesting that some people took that to mean different things. Because my interpretation of processor loading is that we should be worried about the ones that really are going to cause a problem, not that everybody should be concerned about processor loading of course, but that there’s very specific ones in our application that are of a concern of ours which is GNC, SI, and Record.

The leader spoke in terms of how to manage this risk across multiple modules. This is consistent with his responsibilities as technical leader. His unique point of view may be the reason that risk was implied in the term for him but not for others.

Cases of similarity in measurements are harder to investigate because, without a specific difference, it is harder to find a topic to discuss. There were statements of agreement about concepts that ranked higher, or it was difficult to find differences, and this was taken as support for shared understanding of these terms.

Team 2's discussion revealed differences in experience and roles within the team. For example, participants P2.1 and P2.5 had many years of experience in the business area and with this specific software application while others, such as P2.2 and P2.4, were relatively new to the software.

Discussion of the lowest ranking concept on the first questionnaire, C2.4 'Case Person Service', revealed a specific difference between how P2.2 and P2.3 thought about a relationship. Participant P2.2 thought in terms of data elements underlying a complete business record. P2.3 thought in terms of the functional relationship between the services that managed those records:

> P2.2: …to have a, this is just a vague concept of mine, a complete case you have to have both [records] I guess was the way I looked at it.
>
> P2.1: That's fair.
>
> P2.3: For me, because of the term service, I look at it as a service to service relationship, not in terms of the case. Yes, they are related because they both deal with a case, but I tend to be on the lighter side of the relationship which is, they are related, but not very strong.

Here, the participants are explicit that they associate the concepts differently.

More often, differences in understanding were due to a lack of understanding. For example, when asked about a difference in how participants P2.4 and P2.5 thought

about concept C2.21 ‘Edit Auto Assign’, participant P2.4 explained that he was familiar with the code, but not with the business function it implemented and, therefore, was unable to make the same relationships with other related concepts. Participant P2.4 said:

> I guess, because I don’t know anything about Auto Assign, maybe [P2.5], since he knows a lot about this stuff … I think he is able to relate it, because if I see the Edit Auto Assign at the core service, I think in terms of code, not from the business point [of view].

In other cases, members had forgotten about details of features or modules they had not worked with recently. In fact, the technical leader who selected the concepts, P2.1, admitted that he couldn’t remember everything and took time to research relationships he had forgotten while he was taking the questionnaire, ignoring the instructions to answer quickly. The team relies heavily on names for mental cues to remember details, but this is unreliable, as P2.1 said:

> I mean must admit the second time I answered this questionnaire I cheated and looked up some things … I have some remembrance, but yeah, I think we’re consistently finding that these [C2.11 and C2.12 ‘Meetings Service’] are not correctly named.

He is suggesting that he has knowledge about services and their relationships, but it is organized by service responsibilities, not names. If the names are not sufficiently descriptive, or inconsistent, he isn’t able to link the name to the correct mental

construct of the service. The entire team is in agreement that some service names, including C2.11, C2.13, C2.16 and C2.17, are confusing.

Although there was generally support for measured similarities and differences representing actual differences in understanding, this was not universally true. One issue raised by team 1 that seemed particularly important was the wide interpretation possible due to the lack of a specific context to interpret relationships. In one case, participant P1.1 pointed out two somewhat opposite interpretations of a relationship:

> Just how you interpret that statement has a lot to do with it too. So, I could say "Are you looking at things that are going to cause a … reset?" or "Are you going to look at things that are affected if there was a reset?"

Again, for team 2 there was at least one negative case where a difference in measurement was not supported by the discussion. Figure 5.3 shows the detail view for concept C2.2 limited to participants P2.1 and P2.2. Although their PFNets contained no edges in common, the participants agreed that they understood the relationships for C2.2 in the same way. In fact, they rated them substantially the same. Table 5.8 shows the relevant ratings for concept C2.2. Concepts C2.3, C2.4, and C2.9 were rated identically. The reason the PFNets differ is because participant P2.1 had other, shorter paths connecting C2.2 to these nodes. Remember that path length is calculated as the maximum weight along the path (see section 3.3.3.) For participant P2.1, the path (C2.2, C2.5, C2.3) has weights (1, 1) and path length 1. The direct path (C2.2, C2.3) has weight (2) and a length of 2 and is not part of participant P2.1's PFNet. Similarly, other, shorter paths exist for C2.4 and C2.9. The theory

behind Pathfinder suggests that these alternate paths indicate a difference in understanding between participants P2.1 and P2.2 about concept C2.2. This was not confirmed in this case.

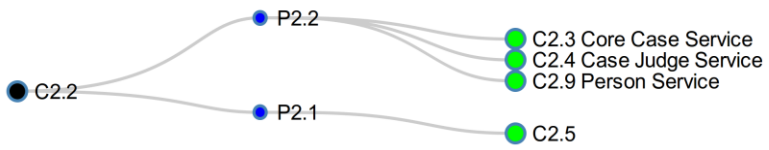


Figure 5.3. Detail for Team 2, Concept C2.2

Table 5.8. Partial Ratings for Team 2, Concept C2.2

		Rating (weight)	
Pair		P2.1	P2.2
C2.2	C2.3	4 (2)	4 (2)*
C2.2	C2.4	4 (2)	4 (2)*
C2.2	C2.5	5 (1)*	2 (4)
C2.2	C2.9	3 (3)	3 (3)*
C2.3	C2.5	5 (1)*	2 (4)

* This edge is in the participant's PFNet

On the whole, concepts that ranked higher had greater shared understanding in the team and concepts that ranked lower had less shared understanding, most often attributed to a lack of understanding by some team members. Differences among PFNets were associated with differences in understanding in the team. When there were similarities in PFNets, it was hard to find meaningful differences in understanding. Although this was generally true, it was not universally true. Counter examples exist for both similarities and differences.

5.2.2.1 Changes in Rank

I asked each team about concepts that had a large change in rank to see if these changes were associated with changes in understanding in the team.

For team 1, concept C1.15 'Prime software watchdog reset' increased from rank 20 to rank 9. The difficulties the team had with this concept are discussed in the previous section. One possible explanation for the increase in rank is that team members inferred, or discussed, the intent of this term after the first questionnaire, although this was not confirmed with the team. Concept C1.6 'Spacecraft Interface (SI) App' dropped from rank 13 to rank 19. No particular reason for a change was identified in discussion with the team, although they did suggest the overall low ranking could be attributed to the independence of the module and that not everyone needed to understand it.

In team 2, concept C2.14 'Person Cards Service' dropped from rank 1 to rank 20. The biggest source of difference were changes in the relatedness ratings from participants P2.3 and P2.4. Participant P2.3 started rating some comparisons involving concept C2.13 as 'No Answer – I don't understand the term' in the second questionnaire. This was one of the concepts with a confusing name and this participant changed his strategy for providing ratings on the second questionnaire. Two other concepts (C2.1 'Assignment Setup Service' and C2.15) changed in rating from Strongly Related ('5') to Not at all related ('X'). This was part of a general shift in this participant's ratings between questionnaires. There was also an effect from participant P2.6 choosing to answer many more ratings as NA in questionnaire 2. Therefore, the change in rank for

this concept reflected a shift in how the participants answered the questionnaire rather than a change in understanding in the team.

Concept C2.12 ‘Meetings Service’ increased from rank 12 to rank 1. In this case, the biggest source of difference were changes in the ratings from participant P2.4 that brought their PFNet into agreement with others. These were not related to the concepts that this participant learned about between questionnaires. No specific reason for these changes was found, suggesting that either some ratings were made in error, or the ratings are unstable.

I also asked the teams to review the ranked concepts to provide their interpretation of what the rank represented. Team 1’s answer was not definite but suggested that higher ranking concepts had been under recent development and lower ranking concepts were related to future work or were concepts with terminology problems.

Team 2’s initial answer was similar. They said that the ranking reflected either how central the concepts were to the design or how recently the team had been working with them. (Centrality and recency of work were associated with higher ranking.) But they noticed that some concepts did not fit this pattern. In particular, C2.20 ‘Meetings UI’ was ranked #2 on the first questionnaire. The team described this as neither central or related to recent work. Further discussion identified that this concept was simple and was related to few other concepts, as described in this exchange:

Researcher: But, is it perhaps simple, obvious, and intuitive?

P2.2: Yes. That was my thought. It's simple and we finished it a while ago and … clear delineation between that and the rest of the stuff.

P2.3: And there's not a lot of relationship between Meetings and the other stuff that we've been working on.

P2.1: Right, Right.

P2.2: Those are very simple.

P2.1: Periphery stuff.

This can be seen visually in Figure 5.2. Concepts C2.11and C2.20 'Meetings UI' are more closely related to each other than other concepts. The suggestion was that, because the concepts were simple, the relationships were more obvious and this supported greater agreement.

Overall there were more NA responses for questionnaire 2 than questionnaire 1. At least one participant (P2.6) said she became more confident about responding with NA as she gained experience with the exercise. Participant P2.5 mentioned that they changed their strategy for making comparisons as they worked their way through the first questionnaire.

5.2.2.2 Terminology

Open coding of the group interviews identified terminology as a common theme in both team's interviews. Terminology was used to explain differences in similarity measures, differences in understanding, and as a source of internal communication problems.

As discussed in the previous section, team 1 had 2 concepts for which the low rank was attributed to the terms selected for concepts. Differences involving C1.11 'FS Flight System Software' were attributed to a mistake among acronyms. Differences involving C1.20 'Processing loading' were found to be related to an implied meaning in the term for the concept that was not shared by the others in the team.

Differences involving C1.15 'Prime software watchdog reset' were found to be based on a difference in understanding of the concept between the team leader and others in the team. There are clearly two mental concepts at play, the original design with one type of reset, and the current design with two. The original design is more abstract and encompasses the redesign. Stated by itself, 'Prime software watchdog reset' is ambiguous, and the leader thinks in terms of the old design while other team members may think of either of the two newer types of reset. The team does not view this as a problem in understanding, only of terminology. In this exchange, the leader (P1.1) first confirms the difference in thinking, but proposes word choice as the solution:

> P1.1: [Participant P1.2] understood the system level better than what I was thinking which is that when I was thinking of watchdog, I still think of them as grouped together, as the same thing, but in fact there are two watchdogs, so …
>
> P1.4: Yeah.
>
> P1.1: I think we have to be more explicit when we talk about the watchdogs … make sure that people understand which watchdog we're talking about.

In order to speak more explicitly, it seems necessary that the leader first think more explicitly, but the team did not interpret the problem in this way.

In team 2, the discussion of terminology focused more on the naming of services and how the names are important cues to the service's function. This was summarized by this discussion:

> P2.1: I think in this case we understand how the underlying parts play together but we're not sure what those parts are named.
>
> [later…]
>
> P2.2: It's the same thing with the judge settings and judge setup. We know that there is a general page or configuration for all judges and we know there's a general and then there's a more specific one. But, we get caught up on the … on which one is which because settings and setup are …
>
> P2.1: synonyms.
>
> P2.2: Yeah. Yeah. There's just … they're very generic and they don't really give you that additional pieces of knowledge that we need.

The team members believe they understand the underlying concepts but the names of the technical components are not sufficiently descriptive to reliably recall the component, especially if they have not worked with that component recently. This could indicate a problem with concept selection (and validation.) However, the fact that this is a more significant problem with concepts that have not been used recently suggests that their understanding actually has diminished and the lack of recall is just a symptom of degraded understanding.

5.2.2.3 Learning

During the group interview, there were a number of cases where one team member asked the others a question about the design of the software or the leader explained

some portion of a concept to others. There were two triggers that preceded these cases. The first was the exposure of a lack of understanding of a concept by at least one member of the team. This was triggered either by the team member who lacked understanding asking a simple question or stating they were not familiar with something. The other trigger was someone considering the relationship between two concepts that they had not considered before.

There were few examples of this from team 1. Participants P1.3 and P1.4 had different questions about details of how modules would interact, such as how messages were encoded when being passed between modules. These were simple questions that were answered directly by the team leader.

There was more discussion of this type in team 2. In one case, the leader described a legacy module to newer members of the team and, in another, there was a short design discussion about how modules should interact according to the architectural goals. In the following exchange, prompted by a difference in team members' PFNets, the leader (P2.1) asked the team to clarify a relationship:

> P2.1: Almost all the [user interfaces] don't call the configuration directly, I don't think?
>
> P2.4: It does.
>
> P2.3: They do.
>
> P2.1: Like, can you give me an example?
>
> P2.3: For setting up county zip codes … the general settings there are a lot of configuration items like … the telephone number, what's your …

P2.1: Right, that's direct.

P2.3: …docket date. What … what is your …

P2.1: … is a good example of one case, it makes sense.

A different kind of learning was reported by some team members. These team members reported gaining insight into the software design as a result of taking the first questionnaire and being forced to consider relationships that they had either never considered previously or, perhaps, had considered but since forgotten. Participant P2.5 said he "realized that some closely coupled portions of the code shouldn't actually be coupled at all." These were relationships inherited from the previous monolithic system. Focusing only on the relationships may have allowed him to gain a more abstract view of the design. After taking the first questionnaire, the leader of team 1 identified 5 specific ideas about potential problems that he wanted to follow up on as a result. The leader of team 2 said he was surprised that some of the rating questions were difficult to answer and that, as the architect who designed those relationships, he expected to be the "answer key". As a result, he intended to research those relationships in the current design.

These comments are evidence of an observer effect whereby the elicitation technique (paired-comparisons) can induce changes in the phenomenon it is intended to measure. In this case, the practitioners thought these new insights were good for their work but, from a research and measurement point of view, it is an important limitation of the elicitation technique.

5.2.3 Relationship of Measures to Outcomes

I had hoped to be able to link measures of shared understanding with difficulties that the team experienced during development but this was mostly unsuccessful. Team #1 has a long development cycle, on the order of months, and there was little testing or integration activity during the research period. Therefore, there were no development difficulties identified that could be related to the SUM-ST results.

I conducted an additional interview with team 2 after a sprint planning session to search for associations between work planned during the sprint and the measured concepts in the team's list. It was difficult for the team to find associations between the abstract concepts in the list and the much more specific development work being done. However, two of the concepts were directly related to work during the sprint. The team reported that, although there were no specific issues in development during the sprint that could be associated with differences in shared understanding as measured by SUM-ST, they believed there was a correlation between the concept rankings and the amount of discussion they needed to have prior to starting work. This is discussed more fully in section 5.2.4 as part of design process.

The other approach to investigating the importance of differences in understanding found was to simply ask teams for their assessment during the group interviews. When asked, each team stated that some of the issues identified could have had an impact on software development outcomes, but that the impact was either not significant or would have been mitigated by other team processes.

Team 1's discussion focused on three problematic concepts: C1.11 'FS Flight System Software', C1.15 'Prime software watchdog reset', and C1.20 'Processing loading'. The issues with these concepts are described in section 5.2.2.

The problem with C1.11 'FS Flight System Software' was attributed to an error in selecting the term for the concept. The team agreed that it was a "simple mistake" and not important to the development of the project. The issues with C1.15 and C1.20 went further.

Although the team viewed the issue with C1.15 'Prime software watchdog reset' primarily as a terminology issue, they said a misunderstanding about terminology could lead to confusion within the team, such as attributing a reported problem to the wrong module. They did not speculate on the magnitude of the potential impact.

For concept C1.20 'Processing loading', the leader viewed the concept in terms of the risk of failure presented by the other modules and effort required to mitigate this risk. He was surprised that the others in the team did not share this view and was concerned whether this risk was being properly managed, as shown in this dialog between participants P1.4 and P1.1:

> P1.4: And you would expect the three apps that you're concerned about with respect to processor loading, as the … software lead, you have a concern for three areas of processor loading, and then … we do all this stuff and it turns out that the three people that are doing those three apps have the same concern of processor loading.
>
> P1.1: Yeah.
>
> P1.4: Or, they don't.

P1.1: Um hum.

P1.4: Then that might be your clue that processor loading as a concern is not being taken seriously by these three people.

P1.1: Yeah

During member checking, the leader, P1.1 stated that he followed up on some of the measured differences uncovered during the group discussion, although he did not identify which.

Team 2 felt that most of the differences in understanding found would not have an impact on software development outcomes. For example, some of the differences found were between how the less experienced members of the project *expected* something to be designed compared to how the more experienced members *knew* it to be designed. These were described as design problems inherited from the legacy system and were quality issues in the sense that, if they were redesigned, they would be easier to understand and reduce the code size, reducing the potential maintenance effort. However, these modules were stable, functional, easy enough to learn, and not currently experiencing problems, and so these potential problems were not considered important.

Many differences in understanding were attributed to service names that were confusing. Although the team agreed this caused a problem in team communication, the felt that it was easily resolved by looking things up, or using alternate descriptions, and they did not judge the potential impact to be serious.

5.2.4 Usefulness of SUM-ST for Design Process Improvement

During the group interview, I asked the teams about the potential value of incorporating the SUM-ST technique as a regular activity in the software design process. The leader of team 1 thought that it was potentially useful, but was more interested in using it to investigate differences in understanding between his team and other teams, such as the independent test team, than within his own team. Team 2 thought that the technique did not inform them of significant new issues and believed that their existing processes provided substantially the same benefit.

Both teams thought that communication and shared understanding within their teams was strong and they were not evaluating SUM-ST as a potential solution to an existing problem. But, each team did express an interest in process improvement. Team 1, in particular, was interested in ways to prevent a critical failure during spaceflight.

In the following exchange from team 1, participant P1.3 discusses using the SUM-ST results to guide communication between developers, using the results presented during the group interview as an example. Participant P1.1 agrees it could be helpful, but is concerned about the trade-off in time invested to research measured differences versus the actual benefit gained:

> P1.3: I could see when you come to implementing … or in a design review or … especially when you implement this or that functionality, pull [the SUM-ST results] up and see who's mapping it to what, especially if, you know, I'm working on SBC and I see [P1.4] didn't think it's related to the metadata PDUs, I'm going to

talk to [P1.4] and I'm going to say why am … I not getting these PDUs from you or …

P1.1: I think it would prompt questions that you wouldn't normally ask, I think that's the key… But then you also run into a lot of these sorts of questions where you end up saying why did I do this and then it turns out, you just did it because you kind of assumed that it was related so was that extra discussion useful?

On the whole, team 1 thought the results were interesting and informative, but not specifically actionable. The leader and the other team members had different opinions about the specificity of the measured differences. The rest of the team believed that, without more clearly defined results, it was difficult to justify making a change to their existing processes. The leader, P1.1, was more favorable and believed that identifying differences in understanding for discussion was valuable even if it those differences were not specifically actionable.

Team 1 was interested in using the results in two new ways that I had not suggested. The first was as a tool to investigate shared understanding between their team and other teams working on the same mission; the second way was as a decision support tool for the leader of the team.

Team 1 was more concerned about shared understanding between their team and other teams than within their team. As an example, they described a recent review of their design conducted by an independent team. Team 1's concern was that it was not possible to know if the review was effective. If the review didn't identify any issues, it could be that there were none, or it could be that the independent team never developed a sufficient understanding to perform an effective review. Thus their first suggested novel way to use SUM-ST was to do a between-teams comparison, using a

variation of SUM-ST, as an objective measure of the capacity of the independent team to perform the review.

The second suggestion by team 1 involves the use of architecture as a coordination tool in software development (Ovaska, Rossi, and Marttiin 2003; Avritzer et al. 2010). In their architecture, a minimum of components are shared between modules. Team members are assigned to modules and are responsible for understanding these module(s) in detail. The goal of module independence, of course, is to minimize the degree of understanding required of other modules. This form of coordination has limits because it is very hard to contain some design decisions to individual modules and it is very hard to maintain awareness of the remaining logical dependencies (Cataldo, Herbsleb, and Carley 2008). To address these coordination difficulties, the leader actively manages the team's information sharing according to his understanding of the architecture and each developer's assignments (a form of explicit vertical coordination (Parolia et al. 2007)). This is difficult in practice because it isn't possible to know exactly what understanding and information needs to be shared within the team. The SUM-ST results provide some visibility to the relationship between team members, technical components, and understanding that could be used to improve management. Describing this, participant P1.1 said:

> … it's just interesting. Some of the things that I look at are just interesting just to wonder about, why they linked it, and then I think leads me to think, well, maybe I need to make sure they are in on certain meetings that they weren't in on before. That they're really off in left field because they think that something is involved in something that completely isn't.

It was clear from many of participant P1.1's comments in the discussion that his concern is as much about whether his team members understand the *right* things, according to his view, as whether they understand each other. During member checking, participant P1.1 said that he "learned a lot about how the group understands various aspects of the code and design" and that he took action to make sure that members of his team were more informed about some of the measured differences discussed during the group interview.

Team 2 said that, while the results were valid, they did not reveal much information that the team was not already aware of. In their team, the biggest differences in understanding were related to areas of the design lacking understanding and concepts with confusing names. Discussing the lack of understanding of one of their low ranking concepts, C2.21 'Edit Auto Assign', participant P2.1 said of the results "So … this didn't reveal that as a mystery to us, we knew that was a mystery already." Because it did not reveal significant new information, they did not believe it had an advantage over their current processes.

The Scrum master, participant P2.5, said that he thought that the agile software development practice of planning poker provided similar information about how the team understood a planned change to the software. The team uses planning poker (Mahnič and Hovelja 2012) during their sprint planning meeting to estimate the effort required to implement the next feature from the list of unimplemented features. In planning poker, each team member makes a private estimate and then everyone reveals their estimate simultaneously (usually by turning over a card). High and low estimates are discussed to identify differences and the process repeats until consensus

is reached. According to participant P2.5, differences in estimation are usually based on differences in the understanding of the complexity of the change, and, therefore, these discussions reveal differences in shared understanding. Although similar, he thought the results of SUM-ST were more specific saying "…it sounds like [SUM-ST] might … give a different insight into some of the same questions that we are trying to answer with [planning poker]. …this might illuminate some … more specific things for technical discussion more so than the general conversations we have had." The rest of the team did not believe any difference would be enough of an advantage to change their current process.

In summary, team 1 viewed SUM-ST more favorably than team 2, but neither team saw enough potential benefit to recommend using it for the original stated goal of identifying specific misunderstandings about the design. The leader of team 1 believed using SUM-ST was valuable to help him understand and manage his team and he was also interested in investigating shared understanding across teams. He approached his management to extend the research for this purpose but his request wasn't approved.

Chapter 6: Discussion

This chapter discusses the results of the main study, and where appropriate, the pilot study. The discussion focuses on methodologic issues and the role of shared understanding in software design. Sections 6.1, 6.2, and 6.3 answer the research questions. Section 6.4 reviews the lessons learned proposes improvements to the SUM-ST method. Section 6.5 makes recommendations for future work. Section 6.6 contains a validity analysis of the research findings and a discussion of the study limitations. Section 6.9 summarizes the contributions of this research. Finally, section 6.10 concludes with a summary of this entire research study.

6.1 Research Question 1

The goal of this research is to develop a measure of shared understanding that can be used to investigate and improve shared understanding in team software design. I used the team mental models construct to operationalize shared understanding and developed the SUM-ST measure to use for software teams. To understand the validity of SUM-ST as a measure of shared understanding, research question 1 asks:

> RQ1: How are similarities and differences between the software development team members' knowledge structures, as measured by SUM-ST, associated with similarities and differences in how those team members understand their design?

The simple, high-level answer to this question is that concepts that rank higher according to $\overline{C_v}$ appear to have greater shared understanding among the team than

concepts that rank lower. Higher ranking concepts tend to be associated with more statements of agreement among team members about how they understand the concept. Concepts that rank lower tend to be associated with more statements of disagreement among the team about the concept. Most often, this disagreement reveals a lack of understanding on at least one person's part rather than well-developed but conflicting points of view. During member checking, all four participants that responded agreed with this conclusion. These data support a finding of a moderately strong positive association between the SUM-ST measure of knowledge structures and similarities in how team members understand their design.

However, as expected for any complex human phenomenon, the interesting part of the answer is in the details, particularly those that do not fit neatly into the above description. While there appears to be a valid overall association, there is significant variation between the team members' stated understanding and measures of individual concepts, meaning that the measure is not very precise. In the extreme, some low ranking concepts were found to be understood similarly and some high ranking concepts were found to be understood differently. This lack of precision may be acceptable for some applications, such as tracking changes in team-level shared understanding over time, but is a problem for other applications, such as the stated goal of identifying specific differences in understanding to make them actionable.

The lack of precision may be due to random factors that cannot be controlled (noise) or to extraneous factors that, once identified, could be excluded by changes to the method. The method-based factors that were identified are discussed in section 6.1.1.

Examining the data for variations in the measure of shared understanding uncovered an important limitation of the validation method while emphasizing the possibility of short-term changes in shared understanding. The approach of comparing measures of shared understanding with statements made by the team members assumes that shared understanding changes little over the interval between the two. There are indications in the collected data that this assumption is false. That is, shared understanding, and individual knowledge structures, may experience important changes over short periods of time, as short as the 30 minutes to take the questionnaire. This presents a paradoxical view of the precision problem. It is possible that the measure is precise but our concept of shared understanding doesn't account for these rapid changes. This view is discussed in section 6.1.2.

Finally, participants in both the pilot and main studies observed that simple concepts tend to rank higher than complex concepts. The role of complexity as a possible confounding variable is discussed in section 6.1.3.

6.1.1 Issues of Precision Attributable to the Measurement Method

The SUM-ST method is complex and unwanted factors may be introduced during concept selection or in the phases of measurement: elicitation, structure representation, or aggregation. Some possible factors were suggested by the data collected, either as explicit statements by the participants or during data analysis. The following sections discuss these factors.

6.1.1.1 Concept Selection

The main study showed that the results depend on the concepts selected for elicitation and their phrasing. The low ranking concepts for team 1 were dominated by phrasing issues. Some of these were introduced by the un-validated edits, but the discussion of terminology shows a more significant issue: it is difficult to separate issues of shared understanding from the issues of terminology used to communicate that understanding. Many of the concepts for team 2 were confusing despite being current working terms used by the team.

One of the guidelines for concept selection is "The phrasing should not be so short as to be ambiguous or so long that it includes its own definition." This guideline assumes that there is a specific concept in each developer's mind that can be triggered by a sufficiently detailed label without providing new knowledge that could affect the measure. The results, from both the pilot as well as the main study, suggest that developer's concepts are not so neatly labelled. Even when the concepts are specific identifiers taken directly from the team's current design, they may not be descriptive enough to elicit the desired knowledge. During member checking, all 4 participants that responded agreed with this interpretation. Although there are indications that differences in interpretation due to phrasing may be linked to differences in understanding, the objective of the SUM-ST technique is to identify issues with understanding, not communication. The results suggest that it is better to err on the side of more descriptive phrases for concepts. This change is detailed with the other proposed changes in section 6.4.1.

Additionally, the issue of linking concept terms to their conceptual meaning is comingled with the issue of context, which is discussed in section 6.1.1.2.

The concepts selected for team 2 were not understood by all members of the team, especially the testers. This resulted in unpredictable answers that affected the results. Participants were instructed to select 'No Answer' if they "don't recognize a term". The testers recognized the terms but did not understand some well enough to provide a meaningful rating. It is tempting to fault the instructions, but I view this as a problem of concept selection and poor communication between myself and the team leader about appropriate concepts to investigate. Certainly the testers, and probably the team, were already aware they lacked shared understanding about these concepts so there is nothing to be gained by the exercise. Concept selection should be more carefully targeted to concepts that the team believes all understand. This is discussed as part of the proposed changes in section 6.4.1.

6.1.1.2 Knowledge Elicitation

Pairwise comparison forces team members to consider comparisons that the team members may have never considered previously. In some cases, participants have difficulty making a judgement and this can produce unpredictable results. In the pilot study, the majority of student teams had no basis for rating relationships between people and requirements or between people and design elements leading to arbitrary answers that were discarded. The addition of the 'No Answer' option in the main study helped, but some comparisons were still problematic. When faced with a novel comparison, some participants spent considerable time thinking about the answer and

some went so far as to consult the software or documentation in order to provide an answer. They did this knowing it was against the instructions. When asked about this, participants emphasized the desire to be "accurate" in their responses. Participant P2.5 said "… the team (myself included) had a leaning towards perfectionism rather than immediacy." This suggests the participants are trying to achieve *closure* for the current task and not providing a rating is "skipping" the task. It may be possible to modify the elicitation method to give options that would allow closure in these cases. Options are discussed as part of the proposed changes in section 6.4.1.

An obvious source of variation is the rating process itself. There was evidence for three different factors affecting the precision of the ratings: changes in the strategy that people used to judge relatedness, the context in which they evaluate the relationship, and random variation (noise) because people are not good at quantifying relationship strength. These are discussed next.

6.1.1.2.1 Strategy

The strategy that some participants used to make comparisons shifted during the ratings questionnaire, leading to inconsistent relatedness ratings. Participants are not used to making relatedness ratings and participants said they needed to develop a strategy to make comparisons. In the pilot study, one participant reported creating a mental "relationship model" that he referred to as the questionnaire progressed. In the main study, participant P2.5 said that he changed his basis for providing ratings during the questionnaire. Poor correlation of repeated ratings has been reported previously (Goldsmith and Johnson 1990). It is possible that variations in ratings may diminish over time as people stabilize their strategies to provide ratings.

An alternative view is that participants should not be consciously aware of a strategy at all and that these reports indicate that, instead of making quick, intuitive judgments, they are making conscious, analytical judgements. That may be true in some cases, but based on observation and discussion with participants, I do not think this is generally true. Some of the participants that made these reports were also the fastest at rating and some participants indicated that adopting a strategy aids in quick judgements.

Clearly, the ratings process itself is poorly understood and requires additional research.

6.1.1.2.2 Context

The context in which participants evaluate relatedness ratings is a well-known factor affecting measures of knowledge structures (Mohammed, Ferzandi, and Hamilton 2010). Although the ratings questionnaire began with a question intended to orient participants to their projects, this may not have been specific enough to establish sufficient context in which to evaluate the ratings. Comparisons of software engineering concepts can be made along many dimensions and across various levels of abstraction. The context in which the participants made their evaluations may have been affected by various external factors such as the task they were performing prior to taking the ratings questionnaire. For example, if they were debugging a feature related to some of the concepts, they may be predisposed to evaluate relationships according to that feature.

One approach to reducing variability due to differences in contextual thinking is to provide much more specific context. For example, the questionnaire could be scenario-based. The leader in team 1 was specifically interested in probing failures related to data transfer. To establish a specific context, the relatedness question could be "In a scenario where data transmitted to the spacecraft is not stored properly, how strongly is X related to Y?" Or, the questionnaire could be preceded by an activity, such as a design review of the concepts, to establish a common context.

However, any changes to the method intended to focus the result on a preconceived context should be considered with caution. Limiting the context of inquiry may have the effect of excluding the most important differences in understanding. Knowledge structures are unique to each individual and context may be as well. Gammack (1990, 224) explores the idea that context is constructed inductively from the concepts, their relationships, and the purpose for holding the knowledge. The most important differences may be in relationships that cut across context resulting in different interpretations. Instead of seeking to eliminate measured differences attributable to context, it is possible the goal should be to elicit each team member's notion of context and identify differences in how team members perceive context.

6.1.1.2.3 Noise

Finally, some proportion of the ratings is likely to be noise because people are not good at quantifying relationship strength. In one previous study, Goldsmith and Johnson (1990) reported a correlation of only 0.60 in repeated ratings. The relationship between noise in individual ratings and the final similarity measure is

complex and not well understood (Gammack 1990). It could be that this noise 'averages out'. But in some situations, it appears to be amplified, discussed next.

6.1.1.3 Structure Representation

The Pathfinder algorithm is used to translate paired relatedness ratings into a network structure. Some properties of this algorithm contribute to unwanted variations in the final measure. As observed in both the pilot and main studies, the inclusion of a specific edge in the PFNet can become critically dependent on small relative differences in ratings. A change in a rating that may be considered irrelevant by the rater could have a dramatic effect on structure of the resulting PFNet and the similarity calculations derived from it (also noted by Gammack (1990)). When there are many similar ratings, noise in an individual rating may be amplified by Pathfinder to produce a noisy structure. Ties are particularly problematic and large numbers of ties may indicate an unstable PFNet.

Possible evidence of this noise may be found in the uniformly low measures for team similarity (tables 4.6 and 5.6) and self-similarity (tables 4.7 and 5.7). The highest values never exceed about half of the scale. The maximum team similarity $\overline{C}$ was 0.52 for the pilot study and 0.33 for the main study. These are consistent with prior reports in the literature: 0.35 (Stout et al. 1999), 0.60 (for NETSIM) (Langan-Fox, Code, and Langfield-Smith 2000), 0.50 (Goldsmith and Johnson 1990). I expected that the self-similarity measures for the main study would be high, at least for the team leaders, because the concepts were core architectural components that the leaders were very familiar with and they did not change during the period between the first and second

questionnaires. However, the self-similarity for each leader was 0.55. The highest measure overall was 0.65 for the self-similarity of participant P13.3 in the pilot study. Why is it that the range of $\overline{C}$ appears to be restricted?

One possible explanation is that noise in the structure representation produced by Pathfinder limits the upper range of the similarity measures to about half of the scale. That is, a significant portion of the measured difference between two or more team members is this noise. In this case, the precision could be improved by changing the Pathfinder algorithm and graph similarity measure to be less sensitive to the critical transition points. These data are not conclusive and this matter deserves further research.

The goal of modifying the algorithm would be to have a small change in a rating produce a proportionately small change in the similarity measure, avoiding the critical transitions. The critical transitions are related to the original objective of Pathfinder to discard information, to identify salient relationships while discarding the others. From an algorithmic design point of view, discarding information prior to rank determination introduces unwanted artifacts into the final measure.

To retain more information, Pathfinder could instead produce a fully connected graph where each edge has a weight equal to the minimum path length between each pair of nodes. The network comparison would then compare these edge weights instead of sets of edges. Although not a trivial exercise, the development of a refined algorithm could be beneficial to the diverse applications of Pathfinder in other fields.

6.1.2 Issues of Precision Attributable to the Phenomenon

Variation in the results can be in part attributable to properties of the measurement method, as described in the previous section, but another set of explanations for the variability have to do with the nature of shared understanding itself. This second set of explanations is the subject of this section.

While analyzing the collected data, I realized that one of the assumptions that I had been making was that shared understanding was stable over short periods of time. This assumption comes from mistakenly equating shared understanding with shared knowledge. Shared knowledge refers to the knowledge that is held in common among the members of the team. Shared understanding refers to the ability to *apply* this shared knowledge to achieve shared goals (Smart et al. 2009; Bittner and Leimeister 2013). Shared understanding incorporates knowledge recall, decision making processes, team interaction, environmental influences, and other factors that may affect the ability to coordinate. Because of these additional process related factors, shared knowledge may be stable but shared understanding may not be.

I had also been assuming that knowledge changes primarily through communication such as discussions with teammates, reading specifications, or reading source code. There is some evidence, however, that knowledge structures can change as a result of internal reflection about things that are already known. Participants in both the pilot and the main studies reported "learning" about their concepts as a result of taking the ratings questionnaire. There is no reason to expect that any new knowledge was imparted from rating pairs of terms the team members were already familiar with. I

interpret these reports of learning as changes in the structure of the knowledge already held. Perhaps, by considering so many relationships in rapid succession, some relationships became more evident and the knowledge structure was reorganized to accommodate these new relationships. In particular, when the Scrum master for team 2 (P2.5) said he "realized that some closely coupled portions of the code shouldn't actually be coupled at all." he is clearly referring to a change in how he understands the relationships between the concepts. His realization occurred during the 30 minutes spent taking the first questionnaire, emphasizing the possibility of short-term changes in knowledge structure.

Turning to the TMM literature, I found that short-term changes in knowledge structures have been identified as an important, but open, topic. Mohammed et al. stated "It is clear that the inconsistent results reported by the few studies that have measured TMMs longitudinally have not clarified our understanding of TMM stability. Furthermore, existing models have said relatively little concerning the speed with which TMMs develop or their rate of change once they are formed" (Mohammed, Ferzandi, and Hamilton 2010, 902). Cooke et al. point out that team mental models are expected to be more dynamic than individual knowledge structures, saying "… in the team domain, changes can be expected to occur much more rapidly than at the individual level, because any individual change can potentially change the overall team knowledge" (Cooke et al. 2000, 157).

The method used in this study was not designed to be sensitive to such short-term changes. Therefore, it cannot distinguish between noise and actual high frequency changes in shared understanding. It is possible that some of the unexpected variation

observed in the measurements (and correspondingly lower similarity) can be attributed to actual changes in the phenomenon. This is a topic that requires further research.

It is possible that these short-term changes in knowledge structures are of greater significance in the domain of software design than in other domains, such as action teams. The number of concepts involved may be greater and the relationships among them may be more complex than other domains. More importantly, when measurement occurs as part of the design process, some portion of the design is intentionally being changed. As developers think about their design, elaborate, and refine portions of the design, their understanding of the design changes. Portions of the design that are considered stable may become understood in a new way. These changes may occur without communication, explicit decisions, or awareness of the team members. Understanding how conceptual design evolves over time would be a valuable topic to explore with a validated measure of shared understanding.

6.1.3 Relationship Between Measures and Complexity

The results of both the pilot and main studies suggest that there is a relationship between the complexity of the concept and the SUM-ST ranking. Simple concepts are more likely to rank high and complex concepts are more likely to rank low. This isn't surprising. It is more difficult to share understanding about a complex concept than a simple one. However, an alternate explanation is that it is more difficult to measure a complex concept than a simple one and the measure may reflect the concept's complexity more than its understanding. A simple concept can be interpreted in only

a few ways and is more likely to be rated similarly by two people. Conversely, a complex concept can be interpreted in many ways and supports a greater number of different, but valid, ratings. In this case, the measured similarity may be more sensitive to the potential for varied interpretation than actual differences in understanding. Other statements by the participants do not support this explanation because participants were usually able to articulate specific differences in low ranking concepts. Still, this is a relationship worth exploring in additional research. It may be valuable, for example, to control for complexity when selecting concepts.

In summary, the evidence supports SUM-ST as a valid measure of shared understanding but as an imprecise measure of the specific similarities and differences in how the team understands their design. Some factors were identified that may contribute to the lack of precision including:

- the selection of concepts that do not represent collaborative tasks,
- the phrasing of some concept labels was not detailed enough to identify the proper concept,
- participants did not always follow instructions to answer quickly without thinking,
- lack of context may have led to multiple, valid interpretations of concepts
- properties of the Pathfinder algorithm can amplify noise that may be inherent in how people assign numeric scores to relationship strength.

Additionally, there is evidence that shared understanding may vary even if the shared knowledge it is based on is stable, but the study method was not able to relate this to the quality of the measurement.

6.2 Research Question 2

Although any differences in knowledge structures within the team are potentially interesting, not all are expected to lead to poor software engineering outcomes. To affect shared understanding, the differences must affect how the team coordinates to achieve team goals. Research question 2 asks:

> RQ2: How are measured differences in software development team members' knowledge structures associated with identifiable difficulties in the software design process resulting in degradation of cost or quality?

The data to answer this question are limited. No specific difficulties (coordination breakdowns) were found to link to measured differences. This was attributed to the abstract concepts selected by the team leaders and the processes used by the teams. When asked to speculate about the relationship between the identified differences in knowledge structures and project difficulties, the participants' opinions were that they were not likely to lead to poor outcomes. The participants did not take action to make any specific design changes as a result of the group discussion. Statements made by the participants give some insights as to why the results were not actionable: many differences were related to knowledge differences between roles, differences were not associated with collaborations and were unimportant for coordination, and the leaders

seemed to have a different view of what makes a difference in understanding important. These topics are discussed in the following sections.

6.2.1 Role Related Differences in Understanding

Many of the differences in knowledge structures that were found were associated with differences in team member roles. The most extreme example was that of the testers from team 2 who had large measureable differences in a large number of concepts. The testers confirmed that most of these differences were associated with their role in the team. The developers in each team also reported differences attributed to role. These role-related differences in understanding were already known to the teams and were quickly ignored in favor of other, unexpected differences. The team members did not expect role-related differences in understanding to be associated with any poor outcomes.

6.2.2 Nature of Collaboration in Selected Concepts

I was initially unaware of the relationship between the concepts selected by the leaders and the roles of the team members. As the discussions progressed, it became clear that each leader's concept selection strategy was not particularly sensitive to the active collaborations within the team. This, in effect, cast a wide net and likely captured more uninteresting differences in understanding than would have been if the leaders had selected more targeted concepts.

This emphasizes the need for the design concepts to participate in active collaborations by the team members at the time of measurement. It may even be

advisable for the concepts to be equally important to everyone taking the questionnaire, to the extent possible. This is discussed further in section 6.4.1.

An alternative approach, suggested by members of team 1, would be to redirect the effort from finding implementation specific differences in understanding within one team to finding gaps in understanding between collaborating teams. Kudikyala (2004) found some success using a similar process to identify misunderstandings in software requirements between stakeholders and developers. These are intuitive applications, but, because these teams do not interact as closely as members of one team, it isn't clear that the roles of shared understanding and implicit coordination are the same. These other applications should be supported by research to identify the coordination needs prior to trying a potential solution.

6.2.3 Shared Understanding Vs. Leader's Understanding

There was a difference between my view of shared understanding and the teams' views of shared understanding. This was evident at the beginning of the concept selection process where I had difficulty having each team leader produce the types of concepts that I originally wanted them to investigate. Each leader selected concepts that reflected the software architecture that they were instrumental in developing and that they had communicated to their team. Each leader's goal was to check how well the team shared their architectural vision. This is similar to a novice/expert comparison and is not the same as shared understanding. Additional research should investigate which is more important in software design: greater shared understanding

within the team or greater similarity between the team members and the technical leader.

6.3 Research Question 3

If team members can discover differences in their understanding, they may be able to act on those differences to improve the software design process. Research question 3 asks:

> RQ3: How can measured similarities and differences in team member knowledge structures be used to improve the process of software design?

RQ3 is exploratory and the answer provided here comes from the opinions of the participants based on their experience and expertise. During the group interview, after completing the review of similarities and differences of the team's knowledge structures, I asked the participants to comment on the potential for such a review to be incorporated into a regular part of their design process and any benefit they thought this would provide. Naturally, the answers were based on the team's current software development projects and processes.

Both teams agreed that shared understanding of their software design is important for their project success and it was clear from descriptions of their processes that they spend a lot of effort to maintain a shared understanding. Team 1's strategy to manage shared understanding involves making a deep investment in each individual to thoroughly understand modules assigned to them and to minimize the need for understanding across modules. Understanding across modules is managed by the

team leader. Team 2 takes a much more distributed, cross-functional approach and they invest heavily in team communication to support this. They have many group meetings, share knowledge and responsibility through pair programming, and emphasize group decision making. Each team expressed confidence in their current processes to manage shared understanding within their team. Neither team reported the kinds of breakdowns that would be symptomatic of problems with shared understanding, such as software integration issues.

A prerequisite of process improvement is the practitioners' ability to interpret the results of the measures. These teams demonstrated the ability to interpret the results with minimal training. Although their first instinct was to focus on the absolute similarity scores, after walking through some examples and discussion, team members demonstrated the ability to interpret the relationships displayed, to compare them with their expectations according to how team members collaborated on the concepts, and to identify similarities and differences in measures that did not agree with their expectations. During member checking, all participants that responded agreed that the visualizations were easy to interpret. Each team requested continued access to the results after the group interview was completed so they could explore the data further on their own. (This could not be granted due to participant anonymity commitments.)

RQ3 proposes to improve shared understanding of the team's design. To understand each team's answer, it is helpful to first understand how each team views the differences in understanding they currently experience. Team 1 commented that their differences in understanding are related to new areas of the design that are

undergoing elaboration and refinement. These differences are a natural part of the design process and tend to work themselves out in the normal course of development. Therefore, these differences are not a problem and are not an opportunity for improvement. (This view agrees with statements made by some of the students in the pilot study.) The differences identified by the measurement of the team's knowledge structures were viewed similarly. That is, they were interesting, but not a cause for concern because such differences are already well managed.

Team 2 described differences in understanding as based in differences in team member's prior experience with the software under maintenance. Again, differences in understanding were accepted and they believed their process was effective in identifying relevant differences in understanding prior to starting work on changes.

Each team's answer to RQ3 was cast as a cost/benefit analysis. They were concerned about any additional time spent on a new activity and this needed to be weighed against the expected benefit. The answers to RQ3 were mixed, depending on the expected benefit. The original goal of using SUM-ST for design process improvement was to bring awareness to specific latent breakdowns that would manifest themselves as faults during implementation. Because none of the identified differences in understanding was this specific, SUM-ST was not judged useful for this goal. However, the leader of team 1 supported using SUM-ST in a more general way to expose alternate viewpoints within the team and to gauge comprehension of key concepts important for coordination.

Team 1 made two suggestions for using measures of shared understanding for improving the software design process. The first suggestion was to look for differences in understanding between members of different teams rather than within one team. The thinking is that, because there is considerably less communication between teams than there is within a team, the risk for undiscovered differences is much greater and this represents a much more valuable area to investigate. This suggests the technique may be more advantageous to apply to teams of teams designing large systems rather than within one closely-collaborating team.

The other suggestion from team 1 was not to identify areas of misunderstanding within the team, but to use the measures of knowledge structures as a kind of decision support tool for the team leader. Previous research has shown that the system architect is a focal point of communication in early development phases when there is a high degree of coordination between teams (Avritzer et al. 2010). Team 1 is at a critical stage of transition from the initial system design to the initial integration of the implementation phase. The architecture of the system has recently been passed from the architects to the team members for implementation. Such a tool would provide a novel form of feedback from the implementers to the architects about this process. Some research has been performed in the educational assessment literature on the use of Pathfinder for this kind of feedback (Trumpower and Sarwar 2010). It would be interesting to pursue the use of such tools in software architecture.

One finding of this study is that participants in both the pilot and main studies commented that they learned about their project as a result of taking the ratings questionnaire. Considering all pairwise comparisons of the concepts forced the

participants to consider novel comparisons that produced new insights to the software design. This process, without subsequent knowledge structure measurement, may have value in support of effective system design reviews. Pairwise comparisons have been recommended as part of software release planning to identify interdependencies between requirements (Carlshamre et al. 2001). It may be possible to similarly integrate pairwise comparison into a form of structured design review.

6.4 Proposed SUM-ST Changes

The results show SUM-ST is valid, but with limited precision. The answer to RQ1 identified some method related issues affecting precision. The answers to RQ2 and RQ3 identified some issues applying SUM-ST to improve shared understanding. This section summarizes these issues and discusses proposed changes to the method to help address them. Future researchers may consider this as starting point should they wish to refine or apply this method in the future.

This section contains lessons learned and describes proposed changes to repeat the main study for the same study goals. This section does not address the more general issue of the merits of using SUM-ST for these or other goals. This is discussed in the Conclusions in section 6.8.

There is empirical evidence for each of the problems addressed by this section, but not for the proposed solution. That is, these changes are speculative and must be evaluated by further research.

These proposed changes are organized into two groups. The first group are changes expected to improve the precision of the measurement. The second group address the goal of applying SUM-ST to improve shared understanding.

6.4.1 Changes to the SUM-ST Method to Improve Precision

The following changes to the SUM-ST method are expected to improve the precision of the measure of shared understanding.

The results of the pilot and main studies were too often not relevant because not all of the participating team members collaborated about the selected concepts. I suggest selecting concepts and people relevant to a specific topic of collaboration. Therefore:

- Begin by identifying a specific topic of collaboration and the team members that need to coordinate in that collaboration. The concepts should be selected from this specific topic and only these collaborating team members should participate in the measure.

In cases where many ratings are similar, Pathfinder may be poor at discriminating salient relationships, resulting in a noisy measure. This may be indicated by a "large" number of ties in the PFNet. Therefore:

- The number of ties in each person's PFNet should be reported. Large numbers of ties may indicate difficulty with the questionnaire or unstable results that should be used with caution. Unfortunately, currently there is not enough data to provide guidance on the percentage of ties to be considered "large". This must be determined by further research.

The paired comparison method of elicitation of knowledge structures requires that participants answer quickly without engaging in problem solving. However, when participants can't easily think of a basis for comparison, they frequently ignore the instruction to answer quickly. A possible solution to this problem is to give a specific response for this condition such as "I'm not sure right now" or "I'd have to think about it." Additionally, outside of research into the method itself, the various 'No Answer' responses may be collapsed into one. Therefore, I propose the following rating scale:

- The ratings questionnaire should have the following responses: "Not at all related", relatedness levels 1 – 5, and two 'No Answer' responses: "I'd have to think about it", and "I don't know".

Some participants reported the need to develop a strategy for making comparisons while taking the ratings questionnaire. This suggests that the measurement may be more stable if the participants take a practice questionnaire (for different concepts) to become familiar with the ratings process. It should be understood that any activity intended to help participants develop a strategy to make comparisons is intentionally affecting the phenomenon under study and may trade accuracy for precision.

- Participants should take a practice ratings questionnaire prior to the measurement ratings questionnaire.

6.4.2 Using SUM-ST to Improve the Design Process

A goal of this research is for practitioners to be able to use measures of knowledge structures to identify and correct misunderstandings in the design. Prior to the development and trial of the measure it was not possible to provide any detailed guidance on how practitioners should use SUM-ST other than "make a measurement and discuss the differences". Based on the results of the study, however, it is possible to begin to build a process around the measurement to support the goal of making the measure useful for design process improvement. This section suggests steps for this process.

Concept selection by one team member (the team leader or the author of a document) may over-represent one point of view in the results. This may be avoided with equal participation in concept selection. Therefore:

- Each team member proposes an equal number of concepts, their share of the total, which should be no greater than 22 (for a 30-minute questionnaire.) Concepts should be named according to the guidelines already defined, but they may be changed in the following step.

The results of this study found that issues of terminology are difficult to separate from issues of shared understanding and that software design concepts do not always have short, clear labels. A team discussion of the differences in meaning of terminology may improve shared understanding while establishing clear meaning for concept labels used in the ratings questionnaire. Therefore:

- The team members meet to negotiate the final list of concepts prior to the ratings questionnaire. They must identify and eliminate duplicates and agree on terminology. This meeting should be moderated and confusing terms or differences in interpretation should be tracked and included in the group discussion.

The detailed view of the result visualization tool was developed to dynamically explore a large number of possible relationships. An alternate view may be appropriate for more targeted concept selection involving 4 or less team members:

- Detailed results for 4 or fewer people may be viewed as Venn diagrams. There would be a closed curve representing each person and the relationships would be located according to the shared edges in their PFNets.

Differences in terminology revealed in concept negotiation are expected to be important to building shared understanding during group discussion. Therefore:

- Team discussion of differences in understanding should be guided by differences identified as part of terminology clarification as well as ranked differences of concepts identified by measurements of shared understanding.

6.5 Recommendations and Future Work

In the software engineering literature, the term *shared understanding* is frequently used as a retrospective evaluation of the quality of coordination. (For example, see Ovaska et al. (2003, para. 4.3.3).) A team that has had few coordination breakdowns is described as having had a shared understanding. Conversely, a team that

experienced many coordination breakdowns must have lacked a shared understanding. In this research, I've tried to shift the perspective from a retrospective evaluation of outcomes to a concurrent evaluation of the mechanisms that produce those outcomes. The discussion of the research questions identified a number of topics for further research, summarized here.

- When taking the ratings questionnaire, participants did not follow the instructions to answer quickly in certain circumstances. The TMM literature has emphasized the need to answer quickly to elicit knowledge structures rather than other relationships. Further research could investigate the difference in ratings quality between answering quickly or slowly and the effect of alternate questionnaire formats and possible responses to improve ratings quality.

- Because there is evidence that the ratings process may induce changes in knowledge structures in software developers, it is important to know more about the stability of knowledge structures and shared understanding and the effect the ratings may have. Do people develop a consistent strategy for ratings? How does the stability of ratings relate to the stability of shared understanding?

- What is the relationship between the perceived complexity of the concepts being measured and the measure of shared understanding? Do simple concepts always rank higher than complex concepts when ranked according to their measured shared understanding?

- The team leaders were more interested in the accuracy of shared understanding than similarity. In the context of software design, which is more important, greater team similarity or greater similarity with the lead designer?

- Because shared understanding may be more difficult to create and maintain between collaborating teams of a large project than within those teams, coordination between teams may present a valuable application to improve shared understanding. How can measures of shared understanding be used to improve the coordination of collaborating teams in a large project?

- Comments by participants on the pairwise ratings process suggests that it can make developers aware of possible design problems they were not aware of previously. Further research could evaluate the efficacy of pairwise comparisons to facilitate design reviews.

- This research was not able to associate specific coordination breakdowns with specific differences in knowledge structures. One approach to investigate this that could be more successful would be to investigate breakdowns retrospectively when they occur. For example, an industrial team could be monitored for breakdowns such as integration failures or design rework. Ideally, it would be possible to take measurements before and after the breakdown is resolved. For example, if the breakdown is detected by a separate test and verification team, an appropriate measure could be constructed and taken before the design team reacts to the error report. This

approach would require a team willing to give significant access to their processes.

- More in-depth research into the role of shared understanding in team software design should be guided by a careful analysis of the collaboration and communication practices in the team. Recent work in socio-technical congruence (Cataldo and Herbsleb 2013) has offered methods to quantitatively evaluate these relationships and may offer a compatible approach to understand these relationships. Measures of shared understanding may be guided congruence measures.

6.6 Implications for Practitioners

The analysis of the industrial teams' software development processes and the associated discussions of shared understanding within the teams provide insights that may be of value to practitioners. Three topics are presented below.

6.6.1 Manage Terminology

The differences in terminology used by different team members were viewed primarily as a communication problem instead of an indication of a deeper difference in understanding. That is, team members tended to assume that their underlying mental concepts were the same and that differences in terminology only represented inconsistent labels used for these identical concepts. This assumption is likely to be false. The design is a complex mental construction of the team and its understanding is shared only in varying degrees. When a difference in the working terminology between team members is discovered, it should first be assumed to reveal an

important difference in how team members understand the concepts that the terms represent. Teams should use this discovery as an opportunity to improve shared understanding by having a group discussion of these concepts. These discussions should start with the functional relationships among concepts to resolve differences in understanding and then conclude with a common definition of terms.

6.6.2 Fit processes to needs

The two teams used very different processes to manage shared understanding and each appeared to be successful. It would be a mistake, however, to assume that the processes are interchangeable. These teams were successful because they achieved a good fit between their coordination needs and their processes.

Team 1 had significant prior experience in the problem domain, in the prior solutions, and in working with each other. This team favored ad-hoc communication between collaborating individuals and shunned scheduled meetings. This would be a very risky working style for less experienced teams.

In contrast, members of team 2 had varying levels of experience, including some inexperienced with the problem domain and with each other, and all had recently switched to agile development practices. They used scheduled group meetings and pair development practices extensively.

Clearly, most teams adapt their processes to their coordination needs. But, they are more likely to achieve a good fit if they take a structured approach to this problem. Implicit coordination is much more efficient than explicit coordination, but requires a significant initial investment to develop the requisite shared understanding. Shared

understanding itself can be divided into task knowledge and team knowledge. Task knowledge can be divided into problem domain, solution domain, and technologies. Team knowledge can be divided into knowledge of team processes and knowledge of individual team members. By considering the distribution of shared knowledge within the team, in each of these areas, teams can make informed decisions about where to invest in shared understanding and where to invest in explicit coordination mechanisms.

6.6.3 Adapt to changes in expertise

In each of the industrial teams that participated, the leader was responsible for the original technical vision that was being developed by their team. It was also the case in each team that, as development progressed, the expertise for some details shifted from the leaders to the developers. The leaders were either unaware of recent changes or had been aware but had forgotten about them. Because part of the leader's role in each team was to manage technical coordination across modules, this shift in expertise is a potentially serious threat to design consistency. Leaders must shift from being knowledge repositories able to resolve conflicts independently to being facilitators to assist the developers to resolve conflicts for themselves.

6.7 Validity Analysis and Limitations

The results of this research are subject to a number of threats to validity and there are limits that should be recognized. This section describes these threats and limitations.

The validity of the SUM-ST measure of shared understanding was based on a single indicator, the statements made during the group interviews, without another indicator

for comparison. If the group interviews did not produce reliable results (for various reasons discussed below), there are no other supporting data to help establish the accuracy or validity of the measure.

The group interviews were coded by a single coder. The statements of understanding made by the team members are subjective and must be interpreted by the other team members and by the researcher. Misunderstandings between team members may give unpredictable results. Misunderstandings by the researcher may, unintentionally, be falsely interpreted in favor of validity.

The group discussions were, no doubt, influenced by the power dynamics within the teams and this could have influenced the veracity of their statements. For example, team members may have been motivated by team inclusion to voice agreement when they did not agree or to make statements they believed the researcher desired, rather than what they believed was true. Although steps were taken to mitigate this (e.g. by asking participants to provide explicit examples and to react to each other), it is impossible to know to what extent participants may have expressed themselves accurately and completely. If distortions in participants' statements were correlated with the measures of knowledge structures, this would provide an alternate explanation for the apparent validity of SUM-ST and reduce support for the results.

The concepts that were investigated were selected by the team leaders. Investigation of different concepts may lead to different results. Additionally, the team leader has a different point of view of these concepts than the other team members. It is possible,

for example, that validity only holds under the condition where the technical leader selects key architectural elements as concepts.

There were some documented quality issues with the relatedness ratings used as part of the measurement. The relatedness ratings are supposed to represent quick, intuitive assessments but sometimes participants evaluated some pairs analytically. There were a few instances where the rating for a pair and the relationship description for the same pair disagreed. Based on prior research, I assume that errors in ratings reduce the quality of the measure of shared understanding, but I have no evidence to discriminate between intended and actual ratings or the actual effect this may have had.

This research is limited by the number and nature of the teams that chose to participate. The 2 industrial and 5 student teams that participated are not diverse in terms of process, quality, or problem domain and cannot represent any larger class of software developers. The small number of teams meant that I was not able to accumulate enough comparative data to identify trends across teams and to discriminate reliable relationships from anomalous ones. The teams appeared to be competent and did not report experiencing difficulties with breakdowns in design or implementation. Virtual teams constrained by communication, global teams with temporal, language, or cultural differences, or commercial teams under extreme time pressure, may be more likely to experience benefits to their design process or otherwise produce different results regarding process improvement.

Access to the teams was limited by the limited time granted by the participant's employers. This prevented a careful analysis of the team roles and collaborations that may have enabled more specific and actionable results. This also limited the depth of inquiry and range of issues that could be explored in the group interview.

6.8 Conclusions

This research was motivated by the difficulty that software design teams face to integrate their knowledge to produce a complete design that is internally consistent. Teams invest heavily in developing and maintaining a shared understanding of the design, but still struggle with breakdowns, such as errors of interpretation that lead to integration failures. These errors can be particularly difficult to identify because team members themselves are not aware of the differences in understanding prior to experiencing the breakdown. This makes the study of shared understanding important for researchers and we should pursue this research with the goal of providing solutions to practitioners. Therefore, the goals of this research were to provide a way for researchers to measure shared understanding and to evaluate the potential for such a measure to be used by practitioners improve the process of team software design.

This research was guided by viewing team design as a distributed decision making process and technical coordination as the management of dependencies between these distributed decisions. Shared understanding of design describes the ability to use shared knowledge to implicitly coordinate these distributed design decisions.

The time spent with the industrial teams confirmed that shared understanding of design is a real problem for software developers. The two teams were each very much aware of issues of shared understanding and each invested heavily in team processes to develop and maintain a shared understanding. Although they used very different strategies in architecture, team structure, and communication, their goals were the same.

The results show that it is possible to measure shared understanding. Although this is true on average, SUM-ST is imprecise at identifying specific differences in understanding. Additionally, SUM-ST was able to identify some differences in understanding that the teams were not already aware of. The current evidence does not support SUM-ST being used to improve team design. The most significant issue is that the measure does not discriminate between differences in understanding that are likely to lead to poor outcomes, and the myriad of other "normal" differences that are not expected to lead to poor outcomes. There is some support for the use of SUM-ST in a more general way to facilitate design discussions about different viewpoints without identifying specific misunderstandings.

This research shows that it is hard to develop a measure for a phenomenon that is not already well understood. The results indicate that shared understanding is more than similarity of knowledge structures. Operationalizing shared understanding as measures of knowledge structures may be a good first-order approximation but leaves a lot of variance unaccounted for. Improvements in measuring shared understanding will require a better theoretical description of how shared knowledge is actively used to achieve coordination. With additional research, we may be able to decompose the

concept of shared understanding into constituent parts. Measures of knowledge structures may describe the *potential* for achieving coordination while other cognitive measures may describe the *process* that transforms this potential into actual results, such as memory recall and the cognitive processes involved in decision making.

This suggests that research should continue along two tracks. The first track is to move forward and apply SUM-ST to investigate shared understanding, to describe it more fully with the tools available. For example, this research suggests that shared understanding is dynamic, even fleeting, and may differ in important ways from shared understanding in action teams. Research to pursue this may start a new line of research in software engineering that is distinct from that in team mental models. These results are preliminary so research that applies SUM-ST must proceed cautiously with awareness of the nature of the measure and its limitations (as documented in this study.)

The second track is to use the results of this research to refine SUM-ST to improve the precision. By understanding the phenomenon better, we can tailor the measurement method to avoid capturing unwanted factors, making it more specific and relevant. This is the process I used to propose the improvements in section 6.4.1. These proposals must be validated.

By pursuing these two tracks in parallel, our understanding of shared understanding and techniques for empirical research of shared understanding will co-evolve.

There is also a third research track that would pursue ways to improve the software design process based on research into shared understanding. Although these teams

were not favorable to using SUM-ST to improve their design process, it is premature to abandon this approach. These teams did not report having the kinds of differences in understanding that were sought, and there were some difficulties implementing the process, such as the abstract nature of the concepts selected, that may have affected the results. Research should continue with other teams, especially those that have a history of coordination breakdowns related to design.

These results also show that applying measures of shared understanding to improve design are not as simple as taking a questionnaire and discussing the results. The improvements that I have suggested, such as guiding concept selection, targeting collaborations, and including issues of terminology, begin to build a process around the measurement. This process to improve software design may include the suggestions that I have made for using SUM-ST or may go in a different direction. For example, because terminology was such a prominent issue for these teams, a process to negotiate and clarify terms may be valuable by itself. Similarly, if pairwise comparisons of design concepts can lead to new insights to the design, this may have its own application. This research track would investigate these and other novel approaches to improving team software design.

6.9 Contributions

This study makes a number of contributions to the body of empirical software engineering research as detailed below:

- The Shared Understanding Measure for Software Teams is a valid measure of the similarity of team knowledge structures and provides a new empirical data

gathering tool to investigate the effects of software development practices. This study includes a great deal of knowledge about its capabilities and limitations.

- The result visualization tool developed to explore the similarities and differences between team member's PFNets is valuable for research that includes interpretation of Pathfinder results, rather than just statistical summaries.

- Although it is premature for practitioners to use measures of knowledge structures to improve the software design process, this study produced knowledge about the needs and current limitations that can guide future work to support this goal.

- The synthesis of a cognitive view of software team design described in section 1.1.1 provides a more comprehensive view of the activity of team design than previously available and directly relates software dependencies between modules to the design decision making process. This may form the basis of new testable hypotheses for future work.

- This study developed a qualitative research method to investigate the complex nature of knowledge structures and their application, not possible with statistical methods, that may form the basis of future research.

- Finally, this study contributed to understanding the nature of shared understanding in a new domain, that of software engineering design.

6.10 Summary

This research has explored the role of shared understanding in a software design team. Shared understanding describes the ability of a team to apply shared knowledge to achieve shared goals. Shared understanding supports implicit coordination whereby team members are able to anticipate the actions and needs of others and act in a coordinated way with a minimum of communication.

The first goal of this research was to develop a measure of shared understanding that may be used by researchers to investigate the role of shared understanding in the performance of software design teams. The second goal of this research was to evaluate the potential for this measure of shared understanding to be used by practitioners to improve the software design process. To be actionable, the measure needed to elicit knowledge the team was not aware of, identify specific areas of misunderstanding, and present results in a way that were interpretable by the design team.

This work was guided by a synthesis of a cognitive view of team software design. In this view, team design is a distributed decision making process and shared understanding enables the implicit coordination of dependencies between distributed decisions. A potential solution to the measurement problem was drawn from the Team Mental Models (TMM) literature, which operationalizes shared understanding as similarity between the individual knowledge structures held by the team members.

The Shared Understanding Measure for Software Teams (SUM-ST) was developed from a review of existing measurement approaches in the TMM literature.

Applying SUM-ST to measure shared understanding in a particular team comprises several steps. First, design elements are selected as concepts, which form the basis for eliciting individual knowledge structures. Individual team member knowledge structures are then elicited through a ratings questionnaire that asks team members to judge the relatedness of all possible pairs of concepts. The pairwise ratings are transformed into networks by the Pathfinder algorithm and compared for similarity using a concept specific variant of the C graph similarity measure. Ranked concept similarity measures are used to guide team discussion to identify potential differences in understanding within the team.

A pilot study was conducted with student teams to evaluate the study method and gather preliminary results. A main study was conducted with two industrial teams. The SUM-ST method was applied and the results were validated against the results of a group interview meant to elicit statements of similarity of understanding by the team members. The results found that the SUM-ST measure was a valid, but imprecise measure of shared understanding. The results were analyzed to identify possible factors contributing to the imprecision and improvements to the measurement method were proposed. Although the SUM-ST measure was valid, it did not identify enough actionable misunderstandings in the teams to be used as a basis for process improvement in software design.

The research results were limited by the small participation rate, only 2 industrial teams. Future research should evaluate the effectiveness of the proposed changes to the measurement method and evaluate the potential of the approach to improve the software design process in teams that have a documented history of coordination

breakdowns related to software design. Analysis of the data collected identified a number of research topics for additional research.

Appendix A: Associations Between Experience, Communication, and Measures of Knowledge Structures

This appendix describes an additional analysis I performed to examine the relationship between the amount of experience people have working together, the amount of weekly communication, and the measured similarity of their knowledge structures.

Prior research has theorized about the role of communication and interaction in the development and evolution of shared knowledge structures (Rentsch and Klimoski 2001; Mathieu, Goodwin, and Heffner 2000; Levesque, Wilson, and Wholey 2001). The theory of team mental models assumes that knowledge is shared through both communication and shared team experience and that this knowledge sharing leads to more similar knowledge structures over time. However, the results of empirical studies do not consistently support this. For example, Mathieu et al. (2000) hypothesized that work experience and similarity of knowledge structures would both increase over time. Instead, they found that similarity of knowledge structures was constant and, therefore, not associated with experience. Studying student software development teams, Levesque et al. (2001) found that team member's mental models diverged over time. Instead of questioning the fundamentals of the theory, researchers assume that the unexpected results are due to complex process factors, such as team working style, or research method related issues, such as the length of the period of study.

After collecting my data, it occurred to me that I could perform a similar, if limited, analysis of my data. For the participating teams, I expected that past working experience contributed to a shared understanding of the design concepts and, therefore, I expected a positive correlation between pairwise working experience and measured similarity of knowledge structures. The relationship for communication is less clear. One could argue that shared understanding enables implicit communication and, therefore, there should be a negative correlation between pairwise communication and measured similarity of knowledge structures. However, these teams are still actively developing their design. Communication may be associated with new development, which may or may not be related to past work that has been shared.

The method is described in section A.1, the results are described in section A.2, and the significance of the results is discussed in section A.3.

A.1 Method

The demographic and interaction questionnaire asked each team member to report the amount of previous working experience, including other employment, with each other member of their team. Similarly, it asked them to report the amount of weekly communication with each other member of their team. These pairwise experience and communication values were compared with the measured similarity C for each of the ratings questionnaires for matching pairs. This gives 2 comparisons for each questionnaire: experience vs. similarity and communication vs. similarity. The measure of association used was Kendall's tau-b (τ_b) (Agresti 2010, 188; Hollander,

Chicken, and Wolfe 2014, 393). Tau can be interpreted as the probability of observing agreement among rank order pairs and has a range of -1 (perfect inversion) to 1 (perfect agreement). Sample estimates of tau were computed in R with the rankor() function in module pvrank (Amerise, Marozzi, and Tarsitano 2015). Reported p-values are for a one-sided test (greater) and, under the condition of ties, the conservative value was used (*Cpv*). The p-value represents the probability of finding data at least as extreme as the sample data, under the condition of independence of the data sets.

Participants reported their experience working with others as ordered categorical data. There are two responses for each pair, one from each participant. An estimate of the working experience, in months, was calculated for each pair as follows. If there was a response from both participants, the experience was taken as the mean of the minimum and maximum range for both responses. If there was only one response (because one person skipped the question), the experience was taken as the mean of the minimum and maximum range for the single response. An estimate of the number of hours per week each pair communicated was calculated in a similar fashion. Estimates for each communication mode were summed to obtain an estimate of the total communication hours per week.

A.2 Results

The estimates of pairwise experience, pairwise communication, and the pairwise similarity measure C for each of the two questionnaires are shown in Table A.1. Sample estimates of Kendall's tau-b and p-values are shown in Table A.2. The results

are limited by the small sample size (10 and 21 pairs) (Bonett and Wright 2000). Team 2 shows a positive correlation between C and experience ($p < 0.05$). The correlations for team 1 are similar but not significant, most likely due to insufficient power. Although the full team 2 shows a significant positive correlation between communication and C for Q2, this is not true for the other comparisons. This is suggestive of a positive correlation between pairwise experience and C but does not support a positive correlation between pairwise weekly communication and C.

A.3 Discussion

These results are limited by the data collection method and small sample sizes but there is evidence of a moderate, positive correlation between SUM-ST measures of knowledge structures and prior working experience. However, there was no evidence that weekly communication is associated with shared understanding.

The results are in agreement with expectations for these teams. However, it should be clear that this does not imply the mechanism by which the similarity in knowledge structures has formed. It is possible that prior working experience is an indicator of direct interaction and knowledge sharing about the design concepts. But, it is also possible that it is a proxy for technical expertise with the concepts that developed through other means (independent design work, software maintenance, etc.)

These results are not strong enough to make any general conclusions but they do support further investigation of the relationship between experience and shared understanding.

Table A.1. Pairwise Experience, Communication, and Similarity Measure C

Team 1		Pair	Q1 C	Q2 C	Exp. (mo.)	Comm. (hrs./week)
	P1.1	P1.2	0.35	0.30	48.00	10.25
	P1.1	P1.3	0.38	0.33	48.00	3.75
	P1.1	P1.4	0.35	0.48	60.00	18.50
	P1.1	P1.5	0.48	0.42	60.00	4.50
	P1.2	P1.3	0.24	0.18	33.00	4.25
	P1.2	P1.4	0.27	0.34	33.00	20.25
	P1.2	P1.5	0.33	0.20	60.00	3.25
	P1.3	P1.4	0.31	0.28	13.50	15.00
	P1.3	P1.5	0.32	0.32	18.00	1.00
	P1.4	P1.5	0.24	0.36	48.00	4.25

Team 2		Pair	Q1 C	Q2 C	Exp. (mo.)	Comm. (hrs./week)
	P2.1	P2.2	18.41	28.97	18.00	9.50
	P2.1	P2.3	42.90	22.53	48.00	3.25
	P2.1	P2.4	9.71	30.62	18.00	11.50
	P2.1	P2.5	33.99	30.59	48.00	5.75
	P2.1	P2.6	15.65	16.51	34.50	2.50
	P2.1	P2.7	8.45	10.78	9.00	2.25
	P2.2	P2.3	13.01	20.45	9.00	3.00
	P2.2	P2.4	13.24	19.02	4.50	10.50
	P2.2	P2.5	28.71	39.46	18.00	8.00
	P2.2	P2.6	11.67	8.53	9.00	3.00
	P2.2	P2.7	12.44	10.58	9.00	3.50
	P2.3	P2.4	15.80	26.33	18.00	3.00
	P2.3	P2.5	24.31	17.99	11.25	3.75
	P2.3	P2.6	13.93	10.79	9.00	3.00
	P2.3	P2.7	7.91	18.38	9.00	2.75
	P2.4	P2.5	13.14	16.52	9.00	10.25
	P2.4	P2.6	11.54	10.50	6.75	3.00
	P2.4	P2.7	8.85	10.85	9.00	2.75
	P2.5	P2.6	17.57	13.64	39.00	7.00
	P2.5	P2.7	13.41	15.15	9.00	4.00
	P2.6	P2.7	15.25	16.93	48.00	10.25

Table A.2. Correlation of Similarity Measure C vs. Experience and Communication

Team	Pairs		τ_b Expr. (p-value)	τ_b Comm. (p-value)
1	10	Q1 C	0.4000 (0.0779)	0.0000 (0.5691)
		Q2 C	0.3556 (0.1082)	0.2667 (0.1904)
2	21	Q1 C	0.4524 (0.0020)*	0.2381 (0.0702)
		Q2 C	0.3381 (0.0180)*	0.3333 (0.0180)*
2 (Dev. only)	10	Q1 C	0.4667 (0.0363)*	-0.3111 (0.8918)
		Q2 C	0.4222 (0.0542)	0.0889 (0.4309)

* p-value ≤ 0.05

Appendix B: Pilot Study Additional Material

B.1 Demographic Questionnaire

CMSC 345 Research Project - Preliminary questions

0% 100%

Personal History

Thank you for helping us with this research project. To start, we would like you to tell us a bit about yourself. This information will help us to understand the answers to the questionnaires and the group interviews. Like all information you provide, this informaiton will be kept confidential.

What is your age?

Only numbers may be entered in this field.

What is your gender?

○ Female ○ Male ◉ No answer

Please describe your prior software engineering experience. Check any that apply

- ☐ I've written programs for classes (high school, college, other.)
- ☐ I've been paid to write programs.
- ☐ I've written programs as a personal hobby.
- ☐ I've contributed to open source software projects.
- ☐ Other:

Do you have prior experience developing software with anyone in your group? If so, please explain who you worked with and the kind of work done (class project, job, etc.)

How did you become a member of this group? Check any that apply

- ☐ The course instructor assigned me to this group.
- ☐ I asked to be a member of this group.
- ☐ Someone else asked me to be a member of this group.
- ☐ Other:

Resume later | Next | Exit and clear survey

B.2 Interaction Guide

Latent Design Errors Pilot Study
First Design Questionnaire Interaction Script

10/26/2013

Overview

This document outlines the details of the how the first design questionnaire will be conducted. See the Research Study Design for additional background on this questionnaire. The participants will take this questionnaire in the Usability Lab, room 443, in the ITE building on the UMBC campus. It is a computer based questionnaire requiring each participant to have access to a computer, monitor, mouse, and keyboard. The questionnaire itself is expected to take from 45 to 60 minutes. Additional time for late arrivals, instructions, questions, and post-questionnaire interview mean the entire procedure could take up to approximately 90 minutes.

Objectives

As a pilot, this study is as much about the novel data collection method as the data collected. This questionnaire is being conducted in a monitored laboratory setting to allow evaluation of participant's experience via direct observation, to identify difficulties with the protocol as evidenced by participant questions and behaviors, and to collect participant opinions via a short post-questionnaire interview.

The questionnaire consists of a large number of comparison ratings. Although simple, the repetitive nature of these questions is taxing to participants and "subject fatigue" has been reported as a limiting factor in prior research (Mohammed, Klimoski, and Rentsch 2000). Therefore, we wish to minimize the overall stress on participants and be able to monitor for signs of fatigue (behaviors such as taking breaks, making explicit comments about fatigue or difficulty).

Other feedback of interest includes the clarity of instructions and any reasons other than fatigue that may be difficult for participants.

Of course, we would also like to collect valid data for subsequent analysis to support the objectives of the study which include finding relationships between knowledge structure and software development performance and evaluating techniques for comparing PFNets among team participants.

General

Up to 4 participants will take the questionnaire at the same time. This limit is primarily due to the limits of the lone researcher to be able to observe and support the participants. It also accommodates the space and computer resources available.

Outline of the procedure:

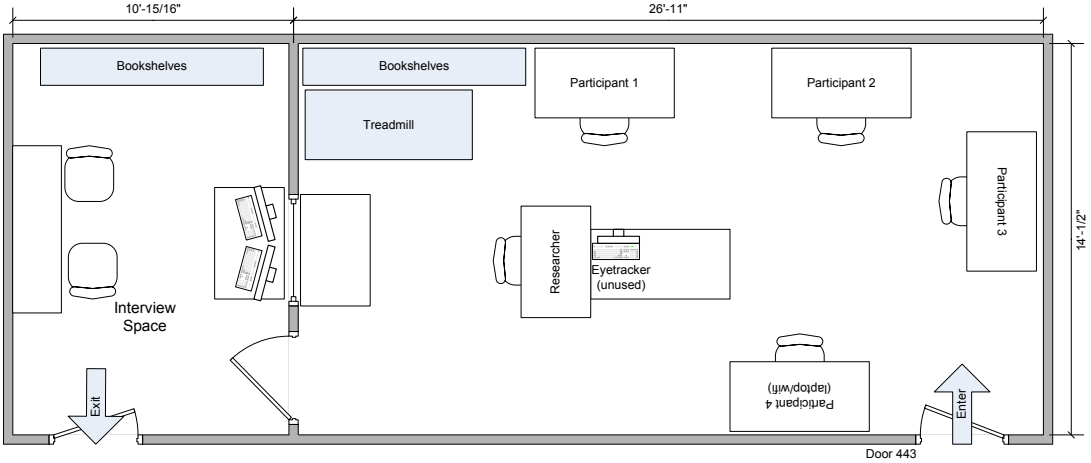
- Environmental setup
- For each 90 minute session:
  - The researcher will prepare a list of scheduled attendees and their questionnaire tokens.

    - Receive and welcome scheduled participants
    - Identify missed appointments
    - Provide instructions as a group
    - Assist each participant in starting the questionnaire
    - Participants take the questionnaire while the researcher observes and responds to questions
    - As each participant completes the questionnaire, the researcher administers the post-questionnaire interview
    - Participants are awarded their incentive
    - Participants exit
    - The lab is reset for the next session
- After the last session, the original lab setup is restored

Procedures

Setup

Room layout:

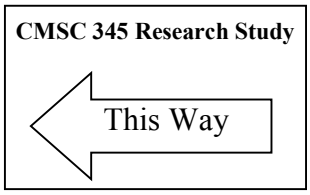


Equipment

- 6 tables (already present in the lab)
- 5 chairs (currently there is an excess that may have to be moved)
- (4) computers (tower, monitor, keyboard, mouse)
- The Eyetracker will not be used. According to lab policy it will not be disconnected.

Other setup

- Hang signs in the hall leading to room 443 from the stairs and the elevator. Sample Sign:



- There will be a reminder sign on the closed entrance door asking them to exit the other way.
- There will be a sign on the door to the control room asking them to wait until the previous interview is complete.
- Money to pay incentives will be obtained ($10 bills) and kept in an envelope held by the researcher.

Preparation

Before each session, the researcher will prepare by identifying the scheduled participants, writing their tokens on a card, checking that there is a signed consent form, and that they have completed the personal history questionnaire. If they haven't completed the personal history questionnaire, the researcher will have this token as well.

Arrival

The researcher will wait just outside door 443 to greet and receive participants. The researcher will invite them in and to have a seat. Seating is open. The researcher will track attendance.
"Hi, I'm <Name>. Thanks for coming. Please come in and have a seat over here. We'll get started as soon as everyone has arrived."
If someone hasn't completed the personal history questionnaire, the researcher will instruct them to complete this while waiting for others to arrive, or, before taking the design questionnaire.
"I see you haven't had a chance to complete the preliminary questionnaire. Let me get you started on that now." The researcher will open the browser at the URL for the preliminary questionnaire and give the participant the token to take their copy.

Instructions

When everyone has arrived or at 5 minutes past the scheduled start, the researcher will get everyone's attention and provide instructions for the questionnaire. This is just a review of the written instructions provided in the questionnaire itself.
"Ok, it is time to get started. Thank you for taking time to assist with this project. I'll try to get you out of here in about an hour. This questionnaire is based on your work for your CMSC 345 class project and the requirements and design that you developed. This is a computer based questionnaire that has two parts. The first part asks you to judge how closely related a pair of words or phrases are. These words or phrases are taken from your project documentation. Since you are from different

groups, each of you has a different set of questions. You'll rate the pairs on a scale of 0 to 6 where 0 means they are not related and 6 means that they are strongly related. There are not right or wrong answers. We are not judging you on whether you are correct or not. You are not expected to take much time answering. Think about the phrases as part of the work you have done for your project, and then see if you think they are related, and mark your first impression. There are a lot of questions so if you get tired, take a short break.
The second part of the questionnaire asks you about just a few of the pairs that you rated in part 1. In part 2, instead of rating them, you are asked to describe the relationship. How are they related? If you think about one of the phrases, why does it make you think about the other one?
After you are done, I'd like to meet with you individually in the back room there to just ask you a few questions about how everything went. If the door is closed, please wait a few minutes until I finish with the person ahead of you.
I'll hang out here in the lab while you are taking the questionnaire so I can help if there are any questions or problems.
You each have a token that you need to take the questionnaire which I've written on a piece of paper that I'll give you in just a second.
Are there any questions before we get started?"

Starting the questionnaire

Each computer should already be logged in (may use the researcher's UMBC account) with a browser window open, full screen, and at the URL for the start page. The participant is prompted to enter their token and is taken to the questionnaire for their group.

Questions, Assistance during the questionnaire

All questions and assistance during the questionnaire will be documented to at least capture the nature of the question and the kind of assistance given.
It is important that participants give their own impressions of the questions asked within the context of their project. Assistance can clarify instructions or solve technical problems, such as getting the browser to the correct URL. The researcher must not give any opinion on the participant's project or answers to the questions in the questionnaire.
Possible questions and responses:

- "Can I check Facebook/email while I wait?" Sorry, but the computers are reserved for the research study.
- "What is my token?" *Look up the participant's token and give it to them.*
- "Why use a token?" We avoid using personal identifiers to keep answers confidential.
- "What do you mean by "closely related"? How do I know if they are closely related or not?" The way in which they are related is entirely up to you, but any kind of relationship works. If you were to make groups out of a bunch of these phrases, how likely would you be to group these two things together? If you think about them long enough, you can come up with some kind of

relationship. That isn't what we are looking for. We are interested in your first impression.

- "The questionnaire is asking me about XXX but I don't know what that is." If you don't know what one of the things is then mark it as unrelated.
- "I'm asked how strongly XXX and YYY are related, but there is more than one way they may be related. Which do I choose?" Choose the first that occurred to you.
- "I'm asked how XXX and YYY are related, but I don't think they are related." Then just say that you don't think they are related.
- "I'm asked how XXX and YYY are related. I can see that they are related, but I don't know how to describe it." Why does one make you think of the other? Can you have one without the other? What about the other way around? If you changed one, would the other change? Why?
- "I'm asked how XXX and YYY are related, but there are many ways they may be related." Then just give the one you think is strongest, or the first that occurred to you.
- "How much longer do I have to do this?" You are on page XXX of YYY so you are about ZZ% done. Do you mind if I watch you answer the next few questions? [This may be an indication that they are having difficulty with the questionnaire. Watch a few responses to see if they are over-thinking the answers or having some other difficulty. Be sure to note this exchange.]
- "Can I go to the bathroom/get a drink?" Sure. But please don't use your cell phone while you are taking a break and come back as quickly as you can. [This kind of a break can affect their contextual thinking. You may remind them to think about their project when they resume. Be sure to note this event.]
- "I don't see the point to this. Why are you asking these kinds of questions?" "Let's talk about that more after you're done."

Post-questionnaire Interview

After completing the questionnaire participants will be invited to the observation room to answer questions without disturbing those still taking the questionnaire and provide privacy for answers. Each participant will be asked (see the instrument at the end for exact questions):

- What, if anything, is confusing in the instructions or questions?
- How difficult was this? What made it difficult? Did this affect your answers?
- Is there anything else you would like to tell me about?

Payment and Exit

Each participant that completes the questionnaire (and has therefore also completed the preliminary questionnaire) will be paid $10. Necessary paperwork for GSA Grant reimbursement will be completed as each payment is made.
Participants will exit by door 442 to avoid disturbing those still working.
"Ok, thank you for that information. Here is your payment for today's activities. There are two more parts coming up. Around the time if your final deliverable, I'll send you a link to another copy of this same questionnaire to take online. We are curious to see how your answers may change by the time you have completed your project. You don't have to come to the lab to take this one; you can do it from home. Around the same time I'll ask your group to sign up for the group interview. I'll provide some food and drink and ask you a few questions about how your project went.
Thanks for coming in today. You can leave from here. Bye"

Reset

After each session the browser will be closed and reopened to the start page.

Follow up

The researcher will contact any missed appointments to request that they reschedule as soon as possible.

B.3 Post-Questionnaire Interview

Participant:

_____

What, if anything, is confusing in the instructions or questions?

_____

_____

_____

On a scale of 0 to 10 where 10 is the most difficult test you've ever taken, how difficult was this?

0 1 2 3 4 5 6 7 8 9 10

What made it difficult?

_____

_____

_____

Do you think this affected your answers? How?

_____

_____

_____

Is there anything else you would like to tell me about?

_____

_____

_____

B.4 Team 10 Interview Guide

Team 10
Participants P10.2, P10.4

Intro	I would like to talk about how the team collaborated on the software design for your project and how the team reached a common understanding of the parts of the design that they had to make work together. I'm interested in what worked well because everyone had the same idea, and what did not work well because different people had different ideas about the same thing.
Q1	Can you tell me a bit about how the original design was developed? Probes: • Who developed that part of the design? • Did this require agreement from others on the team? • How did you reach agreement? • What other part of the project was dependent on this?
Q2	Where there cases where something developed by one person didn't work as expected by someone else? Probes: • Can you describe the example? • Who created the work in this example? • What was different between what you expected and how it worked? • What did the team change to overcome this difference? • How did you reach agreement?
Q3	What about the parts of the design that did work as expected? Probes: ○ Who created the work in this example? ○ How did you know what to expect?
Q4	Now I would like to ask you about some specific parts of the design. By looking at your answers to the questionnaire, one item that you rated differently was the "Class Trip". First, can you describe these relationships? Second, how significant are these differences? Did these differences have an impact in your project? Common: • Method addLocation in class Trip (P10.2, P10.4) P10.2: • Class User (P10.2)

	• route.kml (P10.2) • Requirement: user selects multiple locations (P10.2) P10.4: • Requirement: system displays the shortest distance (P10.4) • Requirement system finds the shortest total distance (P10.4) • Attribute Trip in class User (P10.4)
Q5	What about "Requirement: system creates user accounts"? This was in common in all of your answers. It was related to: In common: ○ allUsers.txt (P10.2, P10.4) ○ Class User (P10.2, P10.4) ○ Method createUser in class User (P10.2, P10.4) ○ users.txt (P10.2, P10.4)
Q6	What about "Attributes Latitude and Longitude in class Location"? This was rated differently. In common: ○ Class Location (P10.2) P10.2: ○ Method createLocation in class Location (P10.2) ○ GoogleEarth's API (P10.2) P10.4: ○ locations.txt (P10.4) ○ route.kml (P10.4)
Q7	What about "Method createLocation in class Location"? This was rated similarly. In common: ○ Attribute isAdmin in class User (P10.2, P10.4) ○ Class Location (P10.2, P10.4) ○ locations.txt (P10.2, P10.4) ○ Requirement: administrator edits locations (P10.2, P10.4) P10.2: ○ Attributes Lattitude and Longitude in class Location (P10.2)
Q8	What about "allUsers.txt"? This was rated differently. In common: ○ Requirement: system creates user accounts (P10.2, P10.4) P10.4: ○ Method createUser in class User (P10.4) ○ Class User (P10.4) ○ Attribute isAdmin in class User (P10.4)

Appendix C: Main Study Additional Material

C.1 Supplemental tables

The full demographic and interaction questionnaire responses are reported in tables C.1, C.2, C.3, and C.4. Reviewing the experience and interaction data identified some missing data and some unusual responses. Participant P2.2 did not answer any pairwise experience or communication questions and participants P1.4, P1.5, and P2.4 skipped some questions. For all pairs, there was at least one response. In a few cases, there were significant differences between the two responses for the same pair. In the most extreme, P2.1 responded working with P2.6 for 6-11 months but P2.6 responded working with P2.1 for more than 5 years. Also unusual was that P1.4 responded communicating a total of 60-114 hours per week, more than twice as much as any other team member. Responding for each communication mode separately may have led this participant to overestimate their time spent communicating.

Table C.1. Main Study, Team 1 Pairwise Experience

	P1.1	P1.2	P1.3	P1.4	P1.5
P1.1		3-4 yr.	3-4 yr.	5+ yr.	5+ yr.
P1.2	3-4 yr.		1-2 yr.	3-4 yr.	5+ yr.
P1.3	3-4 yr.	3-4 yr.		6-11 mo.	1-2 yr.
P1.4	5+ yr.	1-2 yr.	1-2 yr.		3-4 yr.
P1.5	5+ yr.	NA	NA	NA	

Each row contains the responses from the labelled participant.
NA = No Answer

Table C.2. Main Study, Team 2 Pairwise Experience

	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P2.1		1-2 yr.	3-4 yr.	1-2 yr.	3-4 yr.	6-11 mo.	6-11 mo.
P2.2	NA		NA	NA	NA	NA	NA
P2.3	3-4 yr.	6-11 mo.		1-2 yr.	3-5 mo.	6-11 mo.	6-11 mo.
P2.4	NA	3-5 mo.	NA		NA	3-5 mo.	NA
P2.5	3-4 yr.	1-2 yr.	1-2 yr.	6-11 mo.		1-2 yr.	6-11 mo.
P2.6	5+ yr.	6-11 mo.	6-11 mo.	6-11 mo.	5+ yr.		3-4 yr.
P2.7	6-11 mo.	6-11 mo.	6-11 mo.	6-11 mo.	6-11 mo.	3-4 yr.	

Each row contains the responses from the labelled participant.
NA = No Answer

Table C.3. Main Study, Team 1 Pairwise Communication

Communication Method Participant	Communication Time (hours per week)				
Unscheduled Face-to-Face Meeting	P1.1	P1.2	P1.3	P1.4	P1.5
P1.1		5-9	<1	1-5	<1
P1.2	1-5		<1	1-5	<1
P1.3	<1	1-5		5-9	0
P1.4	10-14	15+	5-9		1-5
P1.5	<1	<1	<1	<1	
Scheduled Face-to-Face Meeting	P1.1	P1.2	P1.3	P1.4	P1.5
P1.1		1-5	1-5	1-5	1-5
P1.2	<1		<1	<1	<1
P1.3	0	0		0	0
P1.4	0	<1	1-5		0
P1.5	<1	<1	0	<1	
Scheduled Group Meeting	P1.1	P1.2	P1.3	P1.4	P1.5
P1.1		0	0	0	0
P1.2	1-5		1-5	1-5	1-5
P1.3	<1	<1		<1	<1
P1.4	1-5	1-5	5-9		1-5
P1.5	<1	<1	<1	<1	
Telephone	P1.1	P1.2	P1.3	P1.4	P1.5
P1.1		0	0	0	0
P1.2	0		0	0	0
P1.3	0	0		0	0
P1.4	0	0	0		0
P1.5	0	0	0	0	

Text Chat	P1.1	P1.2	P1.3	P1.4	P1.5
P1.1		0	0	0	0
P1.2	0		0	0	0
P1.3	0	0		0	0
P1.4	0	0	NA		0
P1.5	0	0	0	0	

Remote Video	P1.1	P1.2	P1.3	P1.4	P1.5
P1.1	NA	1-5	1-5	1-5	1-5
P1.2	<1	NA	0	<1	<1
P1.3	0	<1	NA	1-5	0
P1.4	10-14	10-14	<1	NA	<1
P1.5	<1	<1	<1	NA	NA

E-Mail	P1.1	P1.2	P1.3	P1.4	P1.5
P1.1	NA	0	0	0	0
P1.2	0	NA	0	0	0
P1.3	0	<1	NA	0	0
P1.4	0	0	<1	NA	0
P1.5	<1	0	0	0	NA

Each row contains the responses from the labelled participant.
NA = No Answer

Table C.4. Main Study, Team 2 Pairwise Communication

Communication Method Participant	Communication Time (hours per week)						
Unscheduled Face-to-Face Meeting	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P2.1		1-5	<1	<1	<1	0	0
P2.2	NA		NA	NA	NA	NA	NA
P2.3	<1	<1		<1	<1	0	0
P2.4	1-5	1-5	NA		0	0	0
P2.5	1-5	1-5	<1	<1		0	0
P2.6	0	0	0	0	0		1-5
P2.7	0	0	0	0	0	1-5	

Scheduled Face-to-Face Meeting	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P2.1		1-5	<1	<1	<1	0	0
P2.2	NA		NA	NA	NA	NA	NA
P2.3	<1	<1		<1	<1	0	0
P2.4	5-9	1-5	NA		5-9	0	0
P2.5	<1	<1	<1	<1		0	0
P2.6	0	0	0	0	0		1-5
P2.7	0	0	0	0	0	1-5	

Scheduled Group Meeting	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P2.1		1-5	1-5	1-5	1-5	<1	<1
P2.2	NA		NA	NA	NA	NA	NA
P2.3	<1	<1		<1	<1	<1	<1
P2.4	5-9	1-5	NA		5-9	NA	<1
P2.5	1-5	1-5	1-5	1-5		1-5	1-5
P2.6	0	0	0	0	0		1-5
P2.7	1-5	1-5	1-5	1-5	1-5	1-5	

Telephone	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P2.1		0	0	0	0	0	0
P2.2	NA		NA	NA	NA	NA	NA
P2.3	0	<1		<1	0	<1	<1
P2.4	0	<1	NA		0	<1	<1
P2.5	0	<1	<1	<1		1-5	<1
P2.6	0	0	0	0	<1		<1
P2.7	0	0	0	0	0	<1	

Text Chat	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P2.1		0	0	0	0	0	0
P2.2	NA		NA	NA	NA	NA	NA
P2.3	0	0		0	0	0	0
P2.4	NA	0	NA		NA	NA	NA
P2.5	0	0	0	0		0	0
P2.6	0	0	<1	<1	<1		0
P2.7	0	0	0	0	0	0	

Remote Video	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P2.1		<1	<1	<1	<1	<1	<1
P2.2	NA		NA	NA	NA	NA	NA
P2.3	<1	<1		<1	<1	<1	<1
P2.4	<1	<1	NA		<1	<1	<1
P2.5	<1	<1	<1	<1		1-5	<1
P2.6	<1	0	<1	<1	<1		<1
P2.7	<1	<1	<1	<1	<1	<1	

E-Mail	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
P2.1		0	0	0	0	<1	0
P2.2	NA		NA	NA	NA	NA	NA
P2.3	0	<1		<1	<1	<1	<1
P2.4	0	<1	NA		0	<1	<1
P2.5	0	<1	0	0		<1	<1
P2.6	1-5	1-5	1-5	1-5	1-5		<1
P2.7	0	0	0	0	0	0	

Each row contains the responses from the labelled participant.
NA = No Answer

Table C.5 shows the team level similarity, as measured by $\overline{C}$, for each 6-person sub-team for each of the two ratings questionnaires, and their difference.

Table C.5. Main Study, Team 2 Similarity Excluding Individuals

Team 2	Q1 similarity ($\overline{C}$)	Q2 similarity ($\overline{C}$)	Difference
Excluding P2.1	0.15	0.17	0.02
Excluding P2.2	0.17	0.18	0.01
Excluding P2.3	0.15	0.19	0.04
Excluding P2.4	0.19	0.19	0.00
Excluding P2.5	0.15	0.17	0.02
Excluding P2.6	0.18	0.21	0.03
Excluding P2.7	0.19	0.22	0.03

C.2 Demographic and Interaction Questionnaire

Research Project - Preliminary questions

0% 100%

About you

Thank you for helping us with this research project. To start, we would like you to tell us a bit about yourself. This information will help us to understand your answers to the questionnaire. Like all information you provide, this will be kept confidential.

What is your age?
Choose one of the following answers

- 18-24
- 25-34
- 35-44
- 45-54
- 55-64
- 65+
- No answer

What is your gender?

- Female
- Male
- No answer

How many years of experience do you have designing or writing software?
Choose one of the following answers

- Less than 1
- 1-3
- 4-7
- 8-15
- 15+
- No answer

Resume later | Next | Exit and clear survey

Research Project - Preliminary questions

0% 100%

About your team members

Please tell us about working with [Team Member 1]

How much experience do you have (including previous employment) working with [Team Member 1]?

Choose one of the following answers

- 0-2 months
- 1-2 years
- No answer
- 3-5 months
- 3-4 years
- 6-11 months
- More than 5 years

In a typical week, how much time do you spend interacting with [Team Member 1] in the following ways?

	None	Less than 60 min.	1 - 5 hours	5 - 9 hours	10 - 14 hours	15 hours or more	No answer
Unscheduled Face-to-Face Meeting							
Scheduled Face-to-Face Meeting							
Scheduled Group Geeting							
Telephone							
Text Chat							
Remote Video							
E-Mail							

Resume later | Next | Exit and clear survey

C.3 Team 2 Interview Guide

Team 2
Participants: P2.5, P2.4, P2.3, P2.1, P2.6, P2.7, P2.2

Intro	The purpose of this discussion is to evaluate the effectiveness of the questionnaires you took in identifying potential problems in the development of the project. This isn't a test of the people or the project. I would like to talk about how the team collaborated on the software design for your project and similarities and differences in how you answered the questionnaire.
Q1	By looking at your answers to the questionnaire, one item that you rated differently was "C2.2". First, can you describe the following relationships? Second, how significant are these differences? Third, did these differences have an impact in your project? (Relationships shown in result visualization tool.)
Q2	What about "Person Cards Service"? First, can you describe these relationships? Second, how significant are these differences? Third, did these differences have an impact in your project? Relationships: (Relationships shown in result visualization tool.)
Q3	What about "Case Person Service"? First, can you describe these relationships? Second, how significant are these differences? Third, did these differences have an impact in your project? Relationships: (Relationships shown in result visualization tool.)
Q4	What about "Meetings UI "? First, can you describe these relationships? Second, how significant are these differences? Third, did these differences have an impact in your project? (Relationships shown in result visualization tool.)
Q5	Ask about concepts that changed in rank between the 1st and 2nd questionnaires: • Person Cards Service • C2.13 • Meetings Service • C2.6 Is this a result of recent development?

Q6	Show the map. Ask about links that many people agree on (Meetings Service – Meetings UI), (C2.15 – C2.15). Why do they agree? What about the person that doesn't? (P2.7,P2.6)
Q7	Show the map. Ask about links that 3 or 4 people agree on. Why don't they agree? Do people's role have anything to do with whether or not they agree?
Q8	Would the outcome of the project been different if you had been aware of the results of the questionnaire that I just reviewed?
Q9	What is your opinion of incorporating such a questionnaire into the software development process?

Appendix D: Executive Summary

This research explored the role of shared understanding in a software design team. Shared understanding describes the ability of a team to apply shared knowledge to achieve shared goals. Shared understanding supports implicit coordination whereby team members are able to anticipate the actions and needs of others and act in a coordinated way with a minimum of communication.

The first goal of this research was to develop a measure of shared understanding that may be used by researchers to investigate the role of shared understanding in the performance of software design teams. The second goal of this research was to evaluate the potential for this measure of shared understanding to be used by practitioners to improve the software design process. To be actionable, the measure needed to elicit knowledge the team was not aware of, identify specific areas of misunderstanding, and present results in a way that were interpretable by the design team.

This work was guided by a synthesis of a cognitive view of team software design. In this view, team design is a distributed decision making process and shared understanding enables the implicit coordination of dependencies between distributed decisions. A potential solution to the measurement problem was drawn from the Team Mental Models (TMM) literature, which operationalizes shared understanding as similarity between the individual knowledge structures held by the team members.

The Shared Understanding Measure for Software Teams (SUM-ST) was developed from a review of existing measurement approaches in the TMM literature. Applying

SUM-ST to measure shared understanding in a particular team comprises several steps. First, design elements are selected as concepts, which form the basis for eliciting individual knowledge structures. Individual team member knowledge structures are then elicited through a ratings questionnaire that asks team members to judge the relatedness of all possible pairs of concepts. The pairwise ratings are transformed into networks by the Pathfinder algorithm and compared for similarity using a concept specific variant of the C graph similarity measure. Ranked concept similarity measures are used to guide team discussion to identify potential differences in understanding within the team.

A Result Visualization Tool was developed to allow the team members to interpret the measurement results. This interactive web-based tool displays the concepts, relationships between concepts (as represented by team members' Pathfinder networks), and the relationships in common among the team members. Three visualizations are provided: The Ranked Concepts view displays a table of concepts ranked greatest to least by their similarity score. The Team Map view displays a graphic layout of the Pathfinder networks for all team members and allows filtering the visible edges based on the number of team members sharing each relationship. The Detail Map view displays all relationships for one selected concept, grouped by the team members that share each relationship.

Three research questions take a layered approach to investigate the validity of the measure and its usefulness in the software design process. RQ1 investigates the association between the SUM-ST measures and the team's shared understanding. RQ2 extends this to ask about the association between these measures and difficulties

the team experiences with the software design process. Finally, RQ3 asks about the potential of the technique to improve the software design process.

The research is explanatory and used qualitative methods. The SUM-ST method was applied and the results were validated against the results of a group interview meant to elicit statements of similarity of understanding by the team members. This was supported by demographic and interaction questionnaires and observations of team meetings. A pilot study was conducted with five student teams to evaluate the study method and gather preliminary results. A main study was conducted with two industrial teams to answer the research questions.

With regard to RQ1, the results found support for SUM-ST as a valid measure of shared understanding in the teams. Higher ranking concepts were generally, but not consistently, associated with more statements of agreement among team members about how they understand the design compared to lower ranking concepts and were judged to require less discussion when planning new work. Differences in understanding that were associated with low ranking concepts were characterized as revealing a lack of understanding on at least one person's part rather than well-developed but conflicting points of view.

With regard to RQ2, participants judged that the differences in understanding that were identified would not have led to poor outcomes. Reasons included that some differences are a normal part of the design process or that existing team processes would have given similar results. However, discussion of low ranking concepts

identified issues of terminology used by the teams, revealed different points of view, and supported question and answer dialogs within the teams.

With regard to RQ3, the results indicate that it is premature to use SUM-ST to improve the design process. The participants found the results interesting, but felt that they were not specific enough to justify the effort to change their existing processes. However, the leader of one of the teams studied disagreed, saying that the feedback from the discussions helped him to manage his team better. He in fact asked his management for approval to extend the study, but his request was not approved.

Analysis of the results identified some factors that may affect the precision of the SUM-ST measurement. A number of potential improvements to the measure were proposed including a way to better target collaborative design concepts and a change to the questionnaire to encourage more accurate ratings. Proposed improvements for improving the design process include a new concept selection process and a process to identify and resolve terminology issues.

The research results were limited by the small participation rate, only 2 industrial teams. Additionally, validity was based on a single indicator, the statements during the group interview.

The results suggest that operationalizing shared understanding as the similarity of knowledge structures may be a good first-order approximation but some variance is unaccounted for. Improvements in measuring shared understanding will require a better theoretical description of how shared knowledge is used to achieve coordination. Measures of knowledge structures may describe the potential for

achieving coordination while other cognitive measures may describe the process that transforms this potential into actual results, such as memory recall and the cognitive processes involved in decision making.

Research should continue along three tracks. The first track is to move forward and apply SUM-ST to describe shared understanding in software design more fully with the tools available. Research that applies SUM-ST must proceed cautiously with awareness of the nature of the measure and its limitations.

The second track is to use the results of this research to refine SUM-ST to improve its precision. By understanding the phenomenon better, we can tailor the measurement method to avoid capturing unwanted factors, making it more specific and relevant. By pursuing these first two tracks in parallel, our understanding of shared understanding and techniques for empirical research of shared understanding will co-evolve.

The third track is to pursue ways to improve the software design process based on research into shared understanding. Research should continue with additional teams, especially those that have a history of difficulties in the design process. This research track would investigate proposed approaches to improving team software design.

Bibliography

Adelson, Beth, and Elliot Soloway. 1986. "A Model of Software Design." *International Journal of Intelligent Systems* 1 (3): 195–213.

Agresti, Alan. 2010. *Analysis of Ordinal Categorical Data*. 2nd ed. Hoboken, N.J.: John Wiley & Sons, Inc.

Amerise, I. L., M. Marozzi, and A. Tarsitano. 2015. *Pvrank: Rank Correlations* (version 1.0). https://cran.r-project.org/web/packages/pvrank/index.html.

Aranda, Jorge. 2010. "A Theory of Shared Understanding for Software Organizations." Doctoral Dissertation, University of Toronto.

Avritzer, Alberto, Daniel Paulish, Yuanfang Cai, and Kanwarpreet Sethi. 2010. "Coordination Implications of Software Architecture in a Global Software Development Project." *Journal of Systems and Software* 83 (10): 1881–95. doi:10.1016/j.jss.2010.05.070.

Balijepally, VenuGopal, Sridhar Nerur, and RadhaKanta Mahapatra. 2012. "Effect of Task Mental Models on Software Developer's Performance: An Experimental Investigation." In *2012 45th Hawaii International Conference on System Science (HICSS)*, 5442–51. doi:10.1109/HICSS.2012.9.

Bibi, Mamoona. 2010. "Shared Understanding of Requirements within GSD Using Knowledge Management Repository: A Field Experiment." MS Thesis, Islamabad, Pakistan: Riphah International University.

Bittner, Eva Alice Christiane, and Jan Marco Leimeister. 2013. "Why Shared Understanding Matters – Engineering a Collaboration Process for Shared Understanding to Improve Collaboration Effectiveness in Heterogeneous Teams." In *2013 46th Hawaii International Conference on System Sciences (HICSS)*, 106–14. doi:10.1109/HICSS.2013.608.

Bonett, Douglas G., and Thomas A. Wright. 2000. "Sample Size Requirements for Estimating Pearson, Kendall and Spearman Correlations." *Psychometrika* 65 (1): 23–28. doi:10.1007/BF02294183.

Borque, Pierre, and Richard E. Fairley, eds. 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 3.0*. v3.0 ed. Piscataway, NJ: IEEE Computer Society. http://www.swebok.org.

Bostock, Mike. 2015. "D3.js - Data-Driven Documents." Accessed October 16. http://d3js.org/.

Bowen, Glen. 2008. "Grounded Theory and Sensitizing Concepts." *International Journal of Qualitative Methods* 5 (3): 12–23.

Braunschweig, Brandt, and Carolyn Seaman. 2014. "Measuring Shared Understanding in Software Project Teams Using Pathfinder Networks." In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 41:1–41:10. ESEM '14. Torino, Italy: ACM. doi:10.1145/2652524.2652543.

Brehmer, Berndt. 1991. "Distributed Decision Making: Some Notes on the Literature." In *Distributed Decision Making: Cognitive Models for Cooperative Work*. West Sussex, England: John Wiley & Sons, Inc.

Cannon-Bowers, J. A, E. Salas, and S. Converse. 1993. "Shared Mental Models in Expert Team Decision-Making." In *Individual and Group Decision Making: Current Issues*, edited by N. John Castellan, 221–45.

Carlshamre, P., K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag. 2001. "An Industrial Survey of Requirements Interdependencies in Software Product Release Planning." In *Fifth IEEE International Symposium on Requirements Engineering, 2001. Proceedings*, 84–91. doi:10.1109/ISRE.2001.948547.

Cataldo, Marcelo, and James D. Herbsleb. 2013. "Coordination Breakdowns and Their Impact on Development Productivity and Software Failures." *IEEE Transactions on Software Engineering* 39 (3): 343–60. doi:10.1109/TSE.2012.32.

Cataldo, Marcelo, James D. Herbsleb, and Kathleen M. Carley. 2008. "Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development Productivity." In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2–11. ESEM '08. New York, NY, USA: ACM. doi:10.1145/1414004.1414008.

Cooke, Nancy J. 1990. "Using Pathfinder as a Knowledge Elicitation Tool: Link Interpretation." In *Pathfinder Associative Networks*, 227–39. Norwood, NJ, USA: Ablex Publishing Corp.

Cooke, Nancy J., Preston A. Kiekel, and Erin E. Helm. 2001a. "Measuring Team Knowledge During Skill Acquisition of a Complex Task." *International Journal of Cognitive Ergonomics* 5 (3): 297–315. doi:10.1207/S15327566IJCE0503_10.

———. 2001b. "Comparing and Validating Measures of Team Knowledge." *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 45 (4): 361–65. doi:10.1177/154193120104500420.

Cooke, Nancy J., Eduardo Salas, Janis A. Cannon-Bowers, and Renée J. Stout. 2000. "Measuring Team Knowledge." *Human Factors: The Journal of the Human Factors and Ergonomics Society* 42 (1): 151–73. doi:10.1518/001872000779656561.

Cooke, Nancy J., and Roger W. Schvaneveldt. 1988. "Effects of Computer Programming Experience on Network Representations of Abstract Programming Concepts." *Int. J. Man-Machine Studies* 29: 407–27.

Curtis, Bill, Herb Krasner, and Neil Iscoe. 1988. "A Field Study of the Software Design Process for Large Systems." *Commun. ACM* 31 (11): 1268–87. doi:10.1145/50087.50089.

Curtis, Bill, and Diane Walz. 1990. "The Psychology of Programming in the Large: Team and Organizational Behaviour." In *Psychology of Programming*, edited by Jean-Michel Hoc, 253–70. Computers and People. London: Academic.

Curtis, Bill, Diane Walz, and Joyce Elam. 1990. "Studying the Process of Software Design Teams." In *ISPW '90: Proceedings of the 5th International Software Process Workshop on Experience with Software Process Models*, 52–53. Los Alamitos, CA, USA: IEEE Computer Society Press.

Dearholt, Donald W., and Roger W. Schvaneveldt. 1990. "Properties of Pathfinder Networks." In *Pathfinder Associative Networks: Studies in Knowledge*

Organization, edited by Roger W Schvaneveldt, 1–30. Norwood N.J.: Ablex Publishing Corp.

DeChurch, Leslie A., and Jessica R. Mesmer-Magnus. 2010. "Measuring Shared Team Mental Models: A Meta-Analysis." *Group Dynamics* 14 (1): 1.

de Souza, Cleidson, Jon Froehlich, and Paul Dourish. 2005. "Seeking the Source: Software Source Code as a Social and Technical Artifact." In *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work*, 197–206. GROUP '05. New York, NY, USA: ACM. doi:10.1145/1099203.1099239.

Easterbrook, Steve. 1995. "Coordination Breakdowns: Why Groupware Is so Difficult to Design." In *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences, 1995*, 4:191–99. Hawaii: IEEE. doi:10.1109/HICSS.1995.375730.

Edwards, Bryan D., Eric Anthony Day, Winfred Arthur, and Suzanne T. Bell. 2006. "Relationships among Team Ability Composition, Team Mental Models, and Team Performance." *Journal of Applied Psychology* 91 (3): 727.

Espinosa, J. Alberto, Robert E. Kraut, Sandra A. Slaughter, Javier F. Lerch, James D. Herbsleb, and Audris Mockus. 2002. "Shared Mental Models, Familiarity, and Coordination: A Multi-Method Study of Distributed Software Teams." In *Intern. Conf. Information Systems*, 425–33.

Espinosa, J. Alberto, F. Javier Lerch, and Robert E. Kraut. 2004. "Explicit versus Implicit Coordination Mechanisms and Task Dependencies: One Size Does Not Fit All." In *Team Cognition: Understanding the Factors That Drive Process and Performance*, edited by Eduardo Salas and Stephen M. Fiore, 107–29. Washington, DC, US: American Psychological Association.

Gammack, John G. 1990. "Expert Conceptual Structure: The Stability of Pathfinder Representations." In *Pathfinder Associative Networks: Studies in Knowledge Organization*, by Roger W Schvaneveldt, 213–26. Ablex Series in Computational Sciences. Westport, CT, US: Ablex Publishing.

Gasson, Susan. 1998. "Framing Design: A Social Process View of Information System Development." In *Proceedings of the International Conference on Information Systems*, 224–36. ICIS '98. Atlanta, GA, USA: Association for Information Systems.

Goldsmith, Timothy E., and Daniel M. Davenport. 1990. "Assessing Structural Similarity of Graphs." In *Pathfinder Associative Networks: Studies in Knowledge Organization*, edited by Roger W Schvaneveldt, 75–87. Norwood N.J.: Ablex Publishing Corp.

Goldsmith, Timothy E., and Peder J. Johnson. 1990. "A Structural Assessment of Classroom Learning." In *Pathfinder Associative Networks: Studies in Knowledge Organization*, 241–54. Ablex Series in Computational Sciences. Westport, CT, US: Ablex Publishing.

Goldsmith, Timothy E., Peder J. Johnson, and William H. Acton. 1991. "Assessing Structural Knowledge." *Journal of Educational Psychology* 83 (1): 88–96. doi:10.1037/0022-0663.83.1.88.

Guindon, Raymonde, Herb Krasner, and Bill Curtis. 1987. "Breakdowns and Processes during the Early Activities of Software Design by Professionals." In

Empirical Studies of Programmers: Second Workshop, 65–82. Norwood, NJ, USA: Ablex Publishing Corp.

He, Jun, Brian Butler, and William King. 2007. "Team Cognition: Development and Evolution in Software Project Teams." *Journal of Management Information Systems* 24 (2): 261–92. doi:10.2753/MIS0742-1222240210.

Herbsleb, James D., and Audris Mockus. 2003. "Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering." In *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 138–137. ESEC/FSE-11. New York, NY, USA: ACM. doi:10.1145/940071.940091.

Hollander, Myles, Eric Chicken, and Douglas A Wolfe. 2014. *Nonparametric Statistical Methods*. Hoboken, New Jersey: John Wiley & Sons, Inc.

Hsu, Jack S.C., Jamie Y.T. Chang, Gary Klein, and James J. Jiang. 2011. "Exploring the Impact of Team Mental Models on Information Utilization and Project Performance in System Development." *International Journal of Project Management* 29 (1): 1–12. doi:10.1016/j.ijproman.2009.12.001.

IEEE. 1990. "IEEE Standard Glossary of Software Engineering Terminology." IEEE.

igraph Authors. 2014. "Igraph: Network Analysis and Visualization." January 14. https://cran.r-project.org/web/packages/igraph/index.html.

Interlink. 2014. "The Knowledge Network Organizing Tool (KNOT) Software." Accessed July 8. http://interlinkinc.net/KNOT.html.

Johnson, Tristan E., and Debra L. O'Connor. 2008. "Measuring Team Shared Understanding Using the Analysis-Constructed Shared Mental Model Methodology." *Performance Improvement Quarterly* 21 (3): 113–34. doi:10.1002/piq.20034.

Kleinman, David L., and Daniel Serfaty. 1989. "Team Performance Assessment in Distributed Decision Making." In *Proceedings of the Symposium on Interactive Networked Simulation for Training*, edited by J.P. Kincaid Gilson and B. Goldiez, 22–27. Orlando, Florida: Institute for Simulation and Training.

Klimoski, Richard, and Susan Mohammed. 1994. "Team Mental Model: Construct or Metaphor?" *Journal of Management* 20 (2): 403.

Kudikyala, Udai Kumar. 2004. "Reducing Misunderstanding of Software Requirements by Conceptualization of Mental Models Using Pathfinder Networks." Doctoral Dissertation, Mississippi, USA: Mississippi State University.

Langan-Fox, Janice, Sharon Code, and Kim Langfield-Smith. 2000. "Team Mental Models: Techniques, Methods, and Analytic Approaches." *Human Factors: The Journal of the Human Factors and Ergonomics Society* 42 (2): 242–71. doi:10.1518/001872000779656534.

LaToza, Thomas D., Gina Venolia, and Robert DeLine. 2006. "Maintaining Mental Models: A Study of Developer Work Habits." In *Proceedings of the 28th International Conference on Software Engineering*, 492–501.

Leung, Hareton K. N., and Lee White. 1990. "A Study of Integration Testing and Software Regression at the Integration Level." In *Proceedings of the*

Conference on Software Maintenance 1990, 290–301. San Diego, CA, USA. doi:10.1109/ICSM.1990.131377.

Levesque, Laurie L., Jeanne M. Wilson, and Douglas R. Wholey. 2001. "Cognitive Divergence and Shared Mental Models in Software Development Project Teams." *Journal of Organizational Behavior* 22 (2): 135–44.

Lim, Beng-Chong, and Katherine J Klein. 2006. "Team Mental Models and Team Performance: A Field Study of the Effects of Team Mental Model Similarity and Accuracy." *Journal of Organizational Behavior* 27 (4): 403–18. doi:10.1002/job.387.

Mahnič, Viljan, and Tomaž Hovelja. 2012. "On Using Planning Poker for Estimating User Stories." *Journal of Systems and Software*, Selected papers from the 2011 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA 2011), 85 (9): 2086–95. doi:10.1016/j.jss.2012.04.005.

Malone, Thomas W., and Kevin Crowston. 1994. "The Interdisciplinary Study of Coordination." *ACM Comput. Surv.* 26 (1): 87–119. doi:10.1145/174666.174668.

Mancuso, Vincent, Katherine Hamilton, Eric McMillan, Rachel Tesler, Susan Mohammed, and Michael McNeese. 2011. "What's on 'Their' Mind Evaluating Collaborative Systems Using Team Mental Models." *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 55 (1): 1284–88. doi:10.1177/1071181311551267.

Mathieu, J. E, G. F Goodwin, and T. S Heffner. 2000. "The Influence of Shared Mental Models on Team Process and Performance." *Journal of Applied Psychology* 85 (2): 273–83.

Mitchell, R. J. 1989. "It's All Design [software Development]." In *IEE Colloquium on Methods for Creative Design*, 3/1–3/3.

Mohammed, Susan, and Brad C. Dumville. 2001. "Team Mental Models in a Team Knowledge Framework: Expanding Theory and Measurement across Disciplinary Boundaries." *Journal of Organizational Behavior* 22 (2): 89–106. doi:10.1002/job.86.

Mohammed, Susan, Lori Ferzandi, and Katherine Hamilton. 2010. "Metaphor No More: A 15-Year Review of the Team Mental Model Construct." *Journal of Management* 36 (4): 876–910. doi:10.1177/0149206309356804.

Mohammed, Susan, Richard Klimoski, and Joan R. Rentsch. 2000. "The Measurement of Team Mental Models: We Have No Shared Schema." *Organizational Research Methods* 3 (2): 123–65. doi:10.1177/109442810032001.

Oracle Corporation. 2016. "MySQL :: The World's Most Popular Open Source Database." http://www.mysql.com/.

Ovaska, Päivi, Matti Rossi, and Pentti Marttiin. 2003. "Architecture as a Coordination Tool in Multi-Site Software Development." *Software Process: Improvement & Practice* 8 (4): 233–47. doi:10.1002/spip.186.

Parnas, David Lorge. 1971. "Information Distribution Aspects of Design Methodology." 1829. Pittsburgh, Pennsylvania: Computer Science Department, Carnegie Mellon University.

http://repository.cmu.edu/cgi/viewcontent.cgi?article=2828&context=compsci.

Parolia, Neeraj, Stephen Goodman, Yuzhu Li, and James J. Jiang. 2007. "Mediators between Coordination and IS Project Performance." *Information & Management* 44 (7): 635–45. doi:10.1016/j.im.2007.06.003.

Pfleeger, Shari Lawrence. 2001. *Software Engineering: Theory and Practice*. Upper Saddle River, N.J.: Prentice Hall.

Quirin, Arnaud, Oscar Cordón, Vicente P. Guerrero-Bote, Benjamín Vargas-Quesada, and Felix Moya-Anegón. 2008. "A Quick MST-Based Algorithm to Obtain Pathfinder Networks (∞, N − 1)." *Journal of the American Society for Information Science and Technology* 59 (12): 1912–24. doi:10.1002/asi.20904.

R Core Team. 2013. "R: A Language and Environment for Statistical Computing." *R Foundation for Statistical Computing*. http://www.r-project.org/.

Rentsch, Joan R., and Richard J. Klimoski. 2001. "Why Do 'great Minds' Think Alike?: Antecedents of Team Member Schema Agreement." *Journal of Organizational Behavior* 22 (2): 107–20. doi:10.1002/job.81.

Rouse, William B., and Nancy M. Morris. 1985. "On Looking into the Black Box: Prospects and Limits in the Search for Mental Models." 85-2. Atlanta, Georgia, USA: Center for Man-Machine Systems Research, School of Industrial and System Engineering, Georgia Institute of Technology, University System Georgia. http://www.dtic.mil/dtic/tr/fulltext/u2/a159080.pdf.

Ryan, Sharon, and Rory V. O'Connor. 2009. "Development of a Team Measure for Tacit Knowledge in Software Development Teams." *Journal of Systems and Software* 82 (2): 229–40. doi:10.1016/j.jss.2008.05.037.

Schmitz, Carsten, and LimeSurvey Project Team. 2012. "LimeSurvey: An Open Source Survey Tool." *LimeSurvey: An Open Source Survey Tool*. http://www.limesurvey.org/.

Schvaneveldt, Roger W. 2013. "What Is the Difference between Network Similarity and 'C' That I Have Read About?" *FAQ's*. Accessed November 29. http://www.interlinkinc.net/FAQ.html.

Schvaneveldt, Roger W., Francis T. Durso, and Donald W. Dearholt. 1989. "Network Structures in Proximity Data." *The Psychology of Learning and Motivation* 24: 249–84.

Schvaneveldt, Roger W., Francis T. Durso, Timothy E. Goldsmith, Timothy J. Breen, Nancy M. Cooke, Richard G. Tucker, and Joseph C. De Maio. 1985. "Measuring the Structure of Expertise." *Int. J. Man-Machine Studies* 23: 699–728.

Scozzi, Barbara, Kevin Crowston, U. Yeliz Eseryel, and Qing Li. 2008. "Shared Mental Models among Open Source Software Developers." In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, 306. doi:10.1109/HICSS.2008.391.

Sharp, Helen, and Hugh Robinson. 2004. "An Ethnographic Study of XP Practice." *Empirical Software Engineering* 9 (4): 353–75. doi:10.1023/B:EMSE.0000039884.79385.54.

Shope, Steven M., Janie A. DeJoode, Nancy J. Cooke, and Harry Pedersen. 2004. "Using Pathfinder to Generate Communication Networks in a Cognitive Task Analysis." *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 48 (3): 678–82. doi:10.1177/154193120404800386.

Smart, Paul R., David Mott, Katia Sycara, Dave Braines, Michael Strub, and Nigel R. Shadbolt. 2009. "Shared Understanding within Military Coalitions: A Definition and Review of Research Challenges." In *Knowledge Systems for Coalition Operations (KSCO'09)*. Southampton, UK.

Storey, Margaret-Anne. 2006. "Theories, Tools and Research Methods in Program Comprehension: Past, Present and Future." *Software Quality Journal* 14 (3): 187–208. doi:10.1007/s11219-006-9216-4.

Stout, Renée J., Janis A. Cannon-Bowers, Eduardo Salas, and Dana M. Milanovich. 1999. "Planning, Shared Mental Models, and Coordinated Performance: An Empirical Link Is Established." *Human Factors: The Journal of the Human Factors and Ergonomics Society* 41 (1): 61–71. doi:10.1518/001872099779577273.

Tang, Antony, and Aldeida Aleti. 2010. "What Makes Software Design Effective?" *Design Studies* 31 (6): 614–40. doi:10.1016/j.destud.2010.09.004.

Taylor, Richard N., and Andre van der Hoek. 2007. "Software Design and Architecture: The Once and Future Focus of Software Engineering." In *2007 Future of Software Engineering*, 226–43. IEEE Computer Society.

The PHP Group. 2014. "PHP: Hypertext Preprocessor." *PHP: Hypertext Preprocessor*. http://www.php.net/.

Trumpower, David L., and Gul Shahzad Sarwar. 2010. "Effectiveness of Structural Feedback Provided by Pathfinder Networks." *Journal of Educational Computing Research* 43 (1): 7–24. doi:10.2190/EC.43.1.b.

VERBI GmbH. 2013. "MAXQDA: Qualitative Data Analysis Software." http://www.maxqda.com/.

Whitehead, Jim. 2007. "Collaboration in Software Engineering: A Roadmap." In *2007 Future of Software Engineering*, 214–25. FOSE '07. Washington, DC, USA: IEEE Computer Society. doi:10.1109/FOSE.2007.4.