ICS 280: Theories and Practices in Collaboration

Position Paper for Week 2

Topic: Distributed Cognition

Sushil K Bajracharya

sbajrach@ics.uci.edu

1 Introduction

This brief position paper discusses some of the case studies where the key concepts in *Distributed Cognition* has been used, especially in the context of collaborative work practices. In particular I discuss three empirical/ethnographic studies (published in separate papers) that I feel concur with the concepts laid out in [4] and [1] and have some elements pertaining to the notion of *Distributed Cognition*.

The first paper shows how an empiricial field study of software development practices in open source community revealed an entirely different approach to requirements engineering in such projects and how that demonstrates a complex web of socio-technical processes involved in open source projects. The second example summarizes the results of an empirical survey that discovered some important practices followed by the members of a collaborative software development team. Some of those practices were not formally identified as a part of the software development process and were pertaining to the activities and procedures developers took care of while they moved their work between *private* and *public* project workspace. The third paper summarizes how an ethnographic approach led to novel interface and system designs of a ubiquitous application for vineyard management.

2 Three examples of appliations of ethnographic study techniques (in Distributed cognition)

2.1 Collaborative Work Practices in Open Source Requirements Gathering [5]

[5] presents a comparative and descriptive view of the requirements engineering processes in open source software communities as a result of empirical studies in these communities. The study was ethnographic in nature with an assumption that no prior established accepted framework defining how software should be developed exists in such communities.

The author uses seven different empirical field study principles for his research, - i) The hermeneutic circle (analysis of the evolution of the whole process from its parts and interaction between the whole and the parts), ii) contextualization (need to identify the context that characterizes social and historical background of the process/research topic) , iii) revealing the interaction of the researcher and the subjects/artifacts (researcher as a participant observer), iv) abstraction and generalization (generalization and summary of research findings across many similar communities) , v) dialogical reasoning (comparing existing methodology with that found empirically through participant observation), vi) multiple interpretations (need of the realization that different participants see and experience things differently) and vii) suspicion to possible biases or systematic distortions (necessity to look for alternative explanations of the data from the empirical findings and to welcome unbiased and different views on the topic).

The research based on these principles revealed interesting differences between the requirements engineering practices in the open source software community and the standard/classic software engineering process.Based on these differences [5] suggests the need of characterizing a common foundation for the development of open source software requirements. He presents a notion of software informalisms to collectively describe the Web-based descriptions of all the functional and non-functional requirements for open source software systems. Eight such software informalisms are listed such as - usage of computer based communication tools for community communications, using linked web pages as scenarios of usage, How-To guides, external publications in the domain, various interlinked web sources etc.

The paper thus concludes that development of open source software system requirements is a community building process, a complex web of socio-technical processes. It is a collaborative effort based on computer mediated communication that forms a complex social structure in the cyberspace. The author states that since in open source software systems, the developers are generally the end-users of the systems (unlike in commercial/traditional systems), open source software can suffice with reliance on these informal and light weight or non rigorous software informalisms.

The author concludes by raising few other questions regarding this informal nature of requirements engineering process in open source software systems.

2.2 Work practices in collaborative software development [3]

This paper documents the result of an empirical survey that discovered some important practices followed by the members of a collaborative software development team. Some of those practices were not formally identified as a part of the software development process and were pertaining to the activities and procedures developers took care of while they moved their work between private and public project workspace.

In a development environment where multiple stakeholders work on same set of documents concurrently, different kind of configuration management (CM) tools are employed that impose policies about the common and controlled usage of project artifacts. One of the features of such CM tools is that they allow developers to check-out the artifacts they want to work on from the public repository that is accessible to all the participants and then the developers work on the checked-out artifacts in their own private workspace that is not accessible to others. After finishing their works, modified artifacts are then checked-in to the public repository again by the developers. This much coordination is not always enough to get the smooth updates and versioning of commonly used artifacts. Besides the tools at disposal, developers often follow a set of their own methods that include formal and informal techniques of coordination and communication to avoid conflicts and mismatches that might occur as they check-out, modify and check-in their artifacts.

The study done for the paper included a moderately sized software development team at NASA/Ames Research Center. The team already had setup a well defined software process for a group of developers and verification and validation team distributed through out a building. The team relied on different software tools in their process, two of them being a CM tool and bug tracking tool.

Conversations (interviews), documents and observations based on shadowing techniques were used to collect data from the team in form of notes and the data were analyzed using grounded theory techniques. The result of the survey led the authors come up with a set of practices (articulation work) that developers adopted in four situations: private work, in transition form private to public, public and in transition from public to private. Some of the common practices that were followed by the developers were *partial check-ins* and holding onto check-ins as their manual endeavours to reduce conflicts, communication mechanisms relying on e-mails to get advice from experts and recording the impact of the changes in the e-mail.

2.3 Ethnographic study for enhancing interface and system design [2]

This paper is comparatively shorter than the earlier two but it includes a reference to the cognitive ethnography method as put forward by [4]. It also applies the concept in a new area of ubiquitous computing which by its nature is distributed. The three key ideas that [2] presents arguing the need of incorporating cognitive ethnography as a system design issue are -

- Applications need to go beyond simple data gathering and presentation. It is essential to think about how data will make the difference to users
- While desinging interfaces it is necessary to consider the fact how users are accustomed to their current practices. New technologies should provide an oppurtunity to incremental change supporting and exposing users for migration rather than completely replacing the interfaces they have been using.
- As ubiquitous computing spreads out the system interface devices, subjecting themselves to be used by various users in various roles, it is important to take measures to not to overload users with unnecessary interactions with such devices/interfaces

3 Conclusion

Even though the three examples I presented do not exactly follow the model of Distributed Cognition as prsented in [4], they certainly have common elements with each other and also follow some of the important concepts behind the theory of distibuted cognition (such as Ethnographic studies, contextualization, participant observation, social organizations). The key issue, I think of importance is that the notion of Distibuted Cognition does resemble the work practices followed in collaborative settings. Where people work and interact with each other there are certain norms they work by and certain idiosyncracies they have among themselves. They also have the ability to use and create tools and artifacts according to their ease. Study of this behavior and the complex socio-technical interaction can certainly help to enhance the work practices but capturing the interaction in a standard model or theory that would completely demystify or map all such interactions in every possible way seems impossible to me. The challenge I see as a technolgy developer is to identify the elements and interactions that are of importance and that need to be monitored or studied that would, if appropraitely changed bring out some efficient synchrony in collaborative work practices.

The biggest advantage of using and borrowing ideas from theories like Distributed Cognition, I see, in building collaborative environments is that it does not constrain the system designers only with their own perception of the system and assumption about the usability of the system, but it would rather enable to consider all the stakeholders along with the various environmental attributes and artifacts that can possibly be of importance for rich and efficient interplay of all the entities of the system.

References

[1] Mark S. Ackerman and Christine Halverson. Considering an organization's memory. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 39–48. ACM Press, 1998.

[2] Jenna Burrell, Tim Brooke, and Richard Beckwith. Extending ubiquitous computing to vineyards. In *CHI '03 extended abstracts on Human factors in computing systems*, pages 822–823. ACM Press, 2003.

[3] Cleidson R. B. de Souza, David Redmiles, and Paul Dourish. "breaking the code", moving between private and public work in collaborative software development. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 105–114. ACM Press, 2003.

[4] James Hollan, Edwin Hutchins, and David Kirsh. Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Trans. Comput.-Hum. Interact.*, 7(2):174–196, 2000.

[5] Walt Scacchi. Understanding the requirements for developing open source software systems. *IEEE Proceedings - Software*, 149(1):24–39, 2002.

ICS 280: Theories and Practices in Collaboration

Position Paper for Week 3

Topic: Sensemaking

Sushil K Bajracharya

sbajrach@ics.uci.edu

Abstract

The topic of discussion for this position paper is *Sensemaking*. The course material for the topic primarily is chapters 1 and 2 from [Wei95]. I dicuss my understanding of sensemaking as introduced in [Wei95] and as presented in a practical context in [BM02].

1 Sensemaking processes in a complex organizational scenario

In an attempt to find something more that would help me undersand how sensemaking applies in a real world scenario I came across [BM02]. which describes an ethnographic-based field work study of a front-end project and technology selection procedure in a complex organization. The results of the study was finding of the fact that how sensemaking activites are intervoven within the formal procedures being applied in selecting technologies and projects.

The research was conducted in a NASA research laboratory in southern California. The program that was under observation (named VAL) had the main mission to take new technologies that are deemed important to future science missions and perform validation on them. It was primarily a procedure to assess the quality of, and choose from a set of possible technologies and appropriate solutions that would contribute to reduce cost or to conduct experiments in various space science missions NASA conducts. After a technology/solution has been selected NASA would authorize funds/resources for its implementation. For conducting such a critical task like VAL with serious economic, technical and social implications a set of formal and offical procedures were already defined. The *procedure*s , as defined, were aimed to be automated, repeatable, definable, measurable and eventually optimizable. (The authors mention the similarity with software engineering processes and frameworks like CMM).

In order for a *Technology Announcement* (TA) to get finally adopted, it had to go through three review boards and a final selection from a set of *NASA administrators*. One of the reasons to define a formal procedure was to enable the handling of exceptions along with facilitating smooth information flow and precise documentation all along the procedure. The execution of the process was facilitated by highly competent *VAL technologists* who were responsible for *shepherding* the selection process. The NASA administrators desired quantifiable results in the process that they could fit in an algorithm to get precise results for selection. (One of the many reasons to do this being able to avoid any biasness during selection). However, even with all the expertise in quantitative metrics and measurements they had, quantifying the ratings and creating a selection algorithm could not be reasonably done. And, this is where the authors bring in the notion of sensemaking in the context and discuss how sensemaking is interwoven within the formal procedures and how difficulties existed to get the exact quantifiable measurement of the process and yet how all

the stakeholders were using sensemaking in resolving those difficulties.

Give below is the list of the authors' findings about sensemaking in the context of VAL.

Mulitple-stage sensemaking process The entire VAL selection procedure was a mixture of procedural and sensemaking processes. There existed subprocesses within well defined processes for which no known inputs, expected outputs or precise metrics of success were known. Under such conditions, time limits and budgets imposed constraints on the amount of sensemaking that could have been done. However the constant review of the process (like by, multiple review panels) created a *meta* process level sensemaking to occur over time that resulted in further evolution of the overall VAL program. In this sense, the entire process can be seen as a *structured sensemaking*.

Sensemaking in absense of consensus The heterogenity and diversity of technologies invloved in VAL created ambiguity where terms like *risk, quality* and *function* implied different meanings in different contexts even though there were precise techniques to quantify such terms. As a solution the VAL technologists created a *quad-chart* to capture the pertinent technical information on a single page that was further anayzed by review board members. They used the information to make judgements, utilizing their experience and discussed their judgements with each other to reach a consensus which again was a sensemaking process.

Sensemaking with invisible discussions for validation filters There were several *space validation filters* used to determine whether a proposed technology should be considered for space validation or not. These filters did not have precise quantitaive definitions mainly because of the complexity and the diversity of technologies those filters were applied to. The only solution to make decision in such filters was after going through a long discussion (among the experts in the board). These discussions were not outlined in the VAL procedural processes (thus *invisible*). These discussions often tended to be lasting long and borrowed information from the reports of VAL technologists. Overall the whole process of applying validation process required *judgement, learning, reliance on history and expertise, presentation, argumentation by and between multiple participants*, because of which the authors claim this filtering process as a typical sensemaking example.

Sensemaking using social networks It was noticed that the lab's luchrooms served as a space for collocation that facilitated some *unseen* sensemaking activities such as - *discovering, learning about, and soliciting new technologies*. These activities had some significant role in implementing the selection process. Alongwith being a place for casual discussion about the ongoing issues the lunchrooms were also a site for people to exploit the social network that existed in the organization.

Moreover, [BM02] suggests sensemaking is applicable and is present in any design implementation concerns that require *judicious use of learning, understanding and judging*. It argues an algorithmic solution process must be at least as good as a manual sensemaking process and in lack of a truly algorithmic solution, sensemaking process is a viable solution. In any case there must be a technical, economical and political consensus.

2 Conclusion: Making Sense of Sensemaking

It is indeed frustrating when one cannot appreciate the value of a highly acclaimed scholarly work. That has been my experience in reading the first two chapters of [Wei95]. A part of the problem might be me lacking proper background in such area but nonthelesss I could not get a holistic view of the seven properties of

sensemaking from the examples. Especially when the picture I had about sensemaking from the *battered child syndrome* (BCR) got obscured and scattered among the many further examples the author presents. For instance I could not construe much about the notion of *identity* in sensemaking. For me the author was just refering to the identity of the sensemaker and the multiple roles one might have. But I do not see how that fits into BCR for instance - was he talking about the role of a parent as a caretaker and an assialant? The model for me again seems too general that I can fit anything relevant into it. For instance, I can force myself to describe all the scenarios of sensemaking as listed in the previous section from [BM02] based on the framework of seven properties.

A gentler framework of sensemaking as a cycle of *Enactment* → *Selection* → *Retention* → *Enactment* was more comfortable and [BM02] helped to see how sensemaking fits in practice.

As I see it right now, sensemaking in inherent, it is embedded in almost all productive actions we do, more as a *structured sensemaking* as in many situations as described in [BM02]. And, I am left with the question - is not sensemaking as described in [Wei95] simply a human nature of devising a solution to a problem ? If yes, I am yet to see an illustration how the sensemaking framework itself, as presented in first two chapters of [Wei95] is or can be used.

References

[BM02] Mark Bergman and Gloria Mark. Technology choice as a first step in design: the interplay of procedural and sensemaking processes. In *Proceedings of the conference on Designing interactive systems*, pages 224–234. ACM Press, 2002.

[Wei95] Karl E. Weick. *Sensemaking in organizations*. Thousand Oaks. CA: Sage, 1995.

ICS 280: Theories and Practices in Collaboration

Position Paper for Week 5

Topic: Intersubjectivity and Shared Meaning

Sushil K Bajracharya

sbajrach@ics.uci.edu

Abstract

In this position paper I discuss EVIDII (An Environment for Constructing Shared Understanding Through Visualizing Differences of Impressions) [MON99] as a tool related to intersubjectivity and shared meaning. First, I summarize the tool and its feature and second I'll discuss its feature within the context of intersubjectivity and shared meaning as implied by [Wei95] chapter 3, [AD] and [Wil88].

1 EVIDII as a tool for shared understanding

EVIDII [MON99] is described as an environment or a tool for cross-cultural communication between software designers and their clients. With a basic assumption that clients and designers belong to different *work cultures*, EVIDII is built as a platform to resolve problems related to social interaction among these two category of subjects from such different work cultures.

Following is the description of EVIDII from [MON99] that succintly explains the basic functionality it provides -

> EVIDII supports effective communication processes through creating shared understanding by visualizing differences among individual impressions of images and words... providing an environment to associate (visual) images (such as photographs and graphic images) with affective words (such as *refreshing* and *warm*). The system then visualizes relationships among the three sets of data persons, images, and words in a two- or three-dimensional space. By interacting with the visualization interfaces, both clients and designers gradually develop a shared understanding by asking questions such as *what does the client mean by using the word pretty*, *how does the client think of this particular image*, or *which designers find this image cool*. EVIDII provides a shared communicative environment, where the stakeholders can ground their communications for software design.

EVIDII is not a tool that provides a stable ontological mapping between two languages used in different cultures (clients and designers), rather it provides a computational environment that makes them aware of the existence of differences in their expressions for representing impressions. It builds upon the fact that *once they become aware of the differences, people are good at using the breakdown as an opportunity to develop further shared understanding*, this also closely reflects the sensemaking process as discussed in [Wei95].

1.1 Funtionalites provided by EVIDII

The EVIDII system deals with two sets of *objects* and a set of *people* allowing users to survey how *people* think about a set of *objects* or how *people* associate objects from one set of objects to another, for example, how people think of images.

[MON99] descirbes a set of visual images and a set of affective words as the two sets of objects. These features are supported by four components of EVIDII -

Word List Editor allows users to specify the set of words that are to be used. Users can add, modify or remove words from the set of words.

Image Object List Editor allows users to specify the set of visual images in the GIF or JPEG format.

User List Editor with which users can be registered. Each user is assigned an icon image, which is used when visualizing the relationship between the objects.

User Profile Editor with which each user can associate images with more than one word.

With these components users can enagage in three types of interactive functions -

Maps. Each map is a visualization of the set of objects in a two or three-dimensional space and provides a basis for how users can view the relationships among the sets of objects. A map can be either subjective (created using the direct manipulation) or objective (created using some computationally derived properties of the objects).

Perspectives. The perspective function allows users to change how they look at the data according to what the users want to know. For example, with an image perspective, the user can find the person who selected a particular image and the words that were associated with that image. Then with a *person* perspective, the user can examine the words that were associated with a particular image by a specific person. With this functionality, users can understand not only the relationships among the three sets of data but also characteristics of the sets of objects as a whole.

Viewers. Viewers are used to display a visualization of the relationship among the sets of objects on a particular map. Each viewer allows users to take a certain map and certain perspectives.

One of the possible scenarios for using EVIDII is where a client registers his/her interest and feelings about certain User Interface (UI) components to be used in his/her software and the designers then look at the client's interpretations of those components to reason whether such assumptions about UI components is coherent or not.

2 Analyzing EVIDII

The notion of shared meaning and intersubjectivity in our context is centered around the four levels withing sociology - i) individual ii) intersubjective ii) generic intersubjective and iii) extra subjective as descirbed in [Wei95] and [Wil88]. In particular, [Wei95] describes sensemaking in organization as an ongoing iterative process that switches between intersubjective and generic subjective levels. [AD] on the other hand proposes *equifinal meaning* as another way of looking at intersubjectivity and reason about actions taken by subjects, especially emphasizing i) Metaphor, ii)Logical Argument , iii) Affect Modulation and iv) Linguistic Indirection as the basic four communication mechanisms for the development of organizational meaning.

Now, when I look back at EVIDII with this framework in mind I find that EVIDII is mainly a tool that supports intersubjective communication. It might also be seen as a tool allowing the users to express their individual feelings or understanding but in general it facilitates communication in case of conflicts between individual interpretation and facilitates establishing *equifinal meaning*. I don't see a direct implication of the generic intersubjective and extra subjective levels in EVIDII. However, *cultural* differences is highlighted as the motivating factor behind EVIDII. But, it doesnot seem to directly connote the *subjectless batch of culture* where *pure meanings* replaces the *generic self*. Nevertheless the differences in intersubjective interpretations clients and designers have can be thought as influenced by what kind of *pure meanings* they have been exposed to for most of their life. Looking at the four basic mechanisms for communication as proposed by [AD], I see *affect modulation* being directly visible in the tool, *metaphors* might fit in but the tool seems to be supporting interpretations of *symbolic lexical choices* made by stakeholders coming from different cultural background. *Logical Argument* might result in to exlplain the subltleties of different subjective interpretations of objects, like images and association of dfferent words that go with images, to rule out ambiguities and conflicts that may arise. I donot see a placeholder for *linguistic indirection* (that seems to me a political way of presenting seemingly conflict arousing information) in EVIDII. Rather it helps to resolve impreciseness and conflicts that arise using such a method for communication.

References

[AD] Michel G. Bougon Anne Donnellon, Barbara Gray. Communication, meaning and organized ation. *Administrative Science Quarterly*, 31(1):43–45, March.

[MON99] Yasuhiro Yamamoto1 Masao Ohira1 and Kumiyo Nakakoji1. Evidii: An environment for constructing shared understanding through visualizing differences of impressions. *Proceedings of the International Symposium on Future Software Technology (ISFST-99), Nanjing, China*, pages 113–118, October 1999.

[Wei95] Karl E. Weick. *Sensemaking in organizations*. Thousand Oaks. CA: Sage, 1995.

[Wil88] Norbert Wiley. The micro-macro problem in social theory. *Sociologial Theory*, 6:254–261, Fall 1988.

ICS 280: Theories and Practices in Collaboration

Position Paper for Week 6
Topic: Collaboration Failures

Sushil K Bajracharya
sbajrach@ics.uci.edu

Abstract

In this position paper I discuss Therac-25 accidents, a series of accidents that took lives and caused other injuries due to the massive overdoses by Therac-25, a medical electron acelerator used to treat cancer/tumour patients. With the context of *seeking oppurtunities for sensemaking* [Wei95] and disasters caused as a consequence of failing in doing so [Wei93], I will try to look into the traces of sensemaking that was going on as the events contributing to Therac-25 accidents proliferate.

1 Summary of Therac-25 Accidents

Therac-25 accidents are among the mostly cited disasters due to software in safety-critical systems. Therac-25 was a computerized radiation therapy machine that had few enhancements upon its predecessors, Therac-6 and Therac-20. Among the biggest differences it had from its earlier variants were; (i) It was a dual mode linear accelerator that could deliver either photons at 25 MeV or electrons at various energy levels. (ii) Software functionality were limited in the earlier variants while in Therac-25, it was controlled by a PDP-11 computer. The software had more responsibility for maintaining safety as AECL (Atomic Energy of Canada Limited)[1] took advantage of the computer's abilities to control and monitor the hardware in order to avoid duplication of all the existing hardware safety mechanisms and interlocks. Some of the software for earlier machines was reused for Therac-25. The safety analysis of Therac-25 *excluded* the software with following assumptions -

- programming errors have been reduced by extensive testing on a hardware simulator
- program software donot degrade due to wear, reproduction or fatigue
- computer execution errors are due to faulty hardware components

1.1 Accident History

I have included the entire time-line of accident events and actions taken against them as a table (taken from [LT93]) in Section 3. Following the events in the time line we can see a pattern, where the user of Therac-25, operators in hospitals discover new problems and they make an attempt to bring this up to the manufacturer. In earlier stages, the manufacturers are overconfident with their assumptions and discover a quick fix to the problems, overlooking the possibility of faults in their own assumptions. These discrepancies bring other entities into action such as the FDA (Food and Drug Administration), initially FDA too mark the problem as low priority (Class II recall) but as serious accidents occur, like death of a patient, then FDA asks for a CAP (Corrective Action Plan) for Therac-25 from AECL. As new flavours of accidents are discovered leading to more disastrous results the CAP reaches its fith and final verssion in July 1987, 2 years after the orignal accident with Therac-25 was spotted. In 1988, November third, final safety analysis report is issued.

[1] Manufacturer of Therac-25 and co-manufacturer of its earlier variants

2 Seeking traces of 'Sensemaking' in Therac-25 accidents

As I see it the whole cycle of spotting new accidents and taking new immediate actions were sensemaking as being done by AECL on its part. Being biased from its own assumption those intermittent sensemaking were faulty, later on there occured sensemaking-at-large, where all the stakeholders of the system contributed from their part, that led to more appropriate safety analysis reports for Therac-25.

The nature and context of accidents in Therac-25 is different from those of Mann Gulch disaster. The latter was a short period of frenzy where the problem seem to be the breakdown of existing organizational strcuture and loss of shared meanings among the role players. The Threac-25 accidents, however lasted for several years with the organizational roles explicitly visible. At least four different stakeholders were into action - (i) Patients (ii) Users (Hospitals) (iii) Manufacturer (AECL) and (v) Government (FDA). Patients who were the victims of the disaster did not have a direct roleplaying except of few lawsuits that followed (that were externally settled). Users of the Therac-25 were the ones to reveal the discrepancies (*extracted cues*) that triggered *enactment* from both the manufacturer and government. The manufacturer in this case seem to be *biased by their hindsight* and seem to lack the *Attitute of Wisdom*. Also, the roles all the stakeholders had to play change the course of actions in the timeline for Therac-25 accidents. Initially, the users were merely reporting the accidents and bringing them to the manufacturer's attention. Later they formed their *user groups* to discuss the problems they have been facing that later on helped them to play an active role in contributing in the ammendments made by the manufacter in the fifth revision of CAP.

It seems Therac-25 accidents and the actions that followed can be seen as an example of *prolonged sensemaking* where the earlier ongoing sensemaking was incomplete due to inadequacy of looking into the complete picture of what was going on. And, later the rich social interaction of all the stakeholders in different role led to a more productive sensemaking. This phenomenon is so elegantly put in [LT93] as follows -

> Most accidents involving complex technology are caused by a combination of organizational, managerial, technical and sometime, socialogical or political factors. Preventing accidents requires paying attention to all the root causes not just the precipitating event in a particular circumstance.

Although the nature of Therac-25 and Mann Gulch being so different the effect they brought in were surprisingly same. Both led to new ammendments and possiblities to look into to devise solutions for problems that were otherwise overlooked. Indeed, we learn from our mistakes, but it is also unfortunate that some mistakes put our lives on stake. A deeper analysis of this fact that needs a higher ethical and philosophical thinking along with others, indeed is of upmost concern.

References

[LT93] N. G. Leveson and C. S. Turner. An investigation of the therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[Wei93] Karl E. Weick. The collapse of sensemaking in organizations: The mann gulch disaster. *Administrative Science Quarterly*, 38(4):628–652, December 1993.

[Wei95] Karl E. Weick. *Sensemaking in organizations*. Thousand Oaks. CA: Sage, 1995.

3 Appendix: Major Event Time-line of Therac-25 Accidents [LT93]

Major event time line

1985

JUN – **3rd:** Marietta, Georgia, overdose.
Later in the month, Tim Still calls AECL and asks if overdose by Therac-25 is possible.

JUL – **26th:** Hamilton, Ontario, Canada, overdose; AECL notified and determines microswitch failure was the cause.

SEP – AECL makes changes to microswitch and notifies users of increased safety.
Independent consultant (for Hamilton Clinic) recommends potentiometer on turntable.

OCT – Georgia patient files suit against AECL and hospital.

NOV – **8th:** Letter from CRPB to AECL asking for additional hardware interlocks and software changes.

DEC – Yakima, Washington, clinic overdose.

1986

JAN – Attorney for Hamilton clinic requests that potentiometer be installed on turntable.
31st: Letter to AECL from Yakima reporting overdose possibility.

FEB – **24th:** Letter from AECL to Yakima saying overdose was impossible and no other incidents had occurred.

MAR – **21st:** Tyler, Texas, overdose. AECL notified; claims overdose impossible and no other accidents had occurred previously. AECL suggests hospital might have an electrical problem.

APR – **7th:** Tyler machine put back in service after no electrical problem could be found.
11th: Second Tyler overdose. AECL again notified. Software problem found.
15th: AECL files accident report with FDA.

MAY – **2nd:** FDA declares Therac-25 defective. Asks for CAP and proper renotification of Therac-25 users.

JUN – **13th:** First version of CAP sent to FDA.

JUL – **23rd:** FDA responds and asks for more information.

AUG – First user group meeting.

SEP – **26th:** AECL sends FDA additional information.

OCT – **30th:** FDA requests more information.

NOV – **12th:** AECL submits revision of CAP.

DEC – Therac-20 users notified of a software bug.
11th: FDA requests further changes to CAP.
22nd: AECL submits second revision of CAP.

1987

JAN – **17th:** Second overdose at Yakima.
26th: AECL sends FDA its revised test plan.

FEB – Hamilton clinic investigates first accident and concludes there was an overdose.
3rd: AECL announces changes to Therac-25.
10th: FDA sends notice of adverse findings to AECL declaring Therac-25 defective under US law and asking AECL to notify customers that it should not be used for routine therapy. Health Protection Branch of Canada does the same thing. This lasts until August 1987.

MAR – Second user group meeting.
5th: AECL sends third revision of CAP to FDA.

APR – **9th:** FDA responds to CAP and asks for additional information.

MAY – **1st:** AECL sends fourth revision of CAP to FDA.
26th: FDA approves CAP subject to final testing and safety analysis.

JUN – **5th:** AECL sends final test plan and draft safety analysis to FDA.

JUL – Third user group meeting.
21st: Fifth (and final) revision of CAP sent to FDA.

1988

JAN – **29th:** Interim safety analysis report issued.

NOV – **3rd:** Final safety analysis report issued.

Figure 1: Major Event Time-line of Therac-25 Accidents [LT93]

ICS 280: Theories and Practices in Collaboration

Position Paper for Week 7
Topic: Communication and Collaboration

Sushil K Bajracharya
sbajrach@ics.uci.edu

Abstract

An attempt to indentify some genre of communication action in Free/Open Source (F/OSS) Community is made in this position paper.

Genre and Genre Repertoire in F/OSS Community

[OY94] defines a *genre* of organizational communication as

> a distinctive type of communication action, characterized by a socially recognized communicative *purpose* and common aspects of *form* ..Such communicative actions are typified responses to recurrent situations within organizations

With this definition in mind I present some genres of communication action that constitutes the genre repertoire in F/OSS Community. This classification is based on an empirical study of requirements engineering processes in open source software communities described in much detail in [Sca02].

Based on the differences between the practice in commercial software development communities and the F/OSS community, the author in [Sca02] suggests the need of characterizing a common foundation for the development of open source software requirements. He presents a notion of *software informalisms* to collectively describe the Web-based descriptions of all the functional and non-functional requirements for open source software systems as discovered in various stages of software life cycle in F/OSS. Eight such software informalisms are listed and those eight software informalism serve particular purposes for communication in the F/OSS community, most of them can be distinctly identified by their specific forms and every one of them has specific purpose that leads to some discourse among the paricipants of the F/OSS community. Thus I believe each of these eight software informalisms can be identified either as genres of communication actions or a repertoire-subset [1] of the F/OSS genre repertoire for communication and as a whole these eight informalisms constitute the entire repertoire for F/OSS community. I present my classification of these informalisms as a genre or a repertoire subset in Table 1. I further classify them as whether they can be identified with the distinct form they have or the purpose they serve. Sometimes the participants also make sense of the communication genre/artifact depending on the context so I add context to form and structure used to identify a particular genre or a repertoire -subset.

Given below is brief description of each of these informalisms taken from [Sca02]-

[1] By repertoire-subset I mean they further can constitute several genre inside them but they are just the subset of the genre repertoire not the entire repertoire.

Informalisms	genre/repertoire-subset	Identified by
computer based communication tools	repertoire-subset	form, purpose, context
How-to guides	genre	form, purpose
Software Bug Reports/Issue Tracking	genre	form, purpose
External publications	repertoire-subset	form, purpose
Traditional Software system Documentation	repertoire-subset	purpose, form
Software Extension Mechanism/Architectures	genre	form
Scenarios of usage//linked web pages	repertoire-subset	purpose, context
open source web site and source webs	repertoire-subset	purpose, context

Table 1: Classifying software informalisms to genre/repertoire-subset

Computer based communication tools for community communications. The communication systems appearing in the form of: (a) messages placed in a Webbased bboard discussion forums; (b) email list servers; (c) network news groups; (d)Internet-based chat (instant messaging) fall under this.

This communication infrastructure serve as the *place* where all the work pertaining to requirements is done. Messages written and read through these systems, together with references or links to other messages or software webs, help the participants make sense of them. This subset can mediate genres like *memo* and *dialogues*.

HowTo guides Categorized into *formal* and *informal* HowToś. These documents capture and condense *HowTo* perform some behavior , operation, or function with a system, serve as a semi-structured narrative that assert or imply end-user requirements. *Formal* HowToś descriptions include explicit declarations of their purpose as a HowTo and may be identified as a system tutorial. Informal HowToś may appear as a selection, composition, or recomposition of any of the proceeding. These informal HowTo guides may be labeled as a *FAQ*; that is, as a list of frequently asked questions about how a system operates, how to use it, where to find itŠs development status, who developed what, known bugs and workarounds, etc.

Software bug reporting and issue tracking This genre addresses the one of the most obvious and frequent types of discourse that appears with open software systems - discussion about operational problems with the current version of the system implementation. Bugs and other issues (missing functionality, incorrect calculation, incorrect rendering of application domain constructs, etc). Communication media facilitating this genre are email, bug report bboards, threaded email discussion lists and related bug/issue tracking mechanisms like like Bugzilla.

External publications in the domain. This includes external publications that describe open software available for consumption by the public or by community members such as - technical articles, books (less common), professional articles in trade publications (like the *Linux Journal* or *Game Developer* and academic articles that are refereed and appear in conference proceedings or scholarly journals. Each of these types of publications can be seens as a genre of communicative action.

Traditional software system documentation This constitutes more traditional-like artifacts like online manuals and instruction sets.

Software extension mechanism and architectures Documentation of the APIs and source code fall into this.

Scenarios of usage as Linked web pages as scenarios of usage Artifacts created and linked together in a hypermedia like structure by the community members, like screenshots, guided tours, or navigational click-through sequences with supplementary narrative descriptions of how the system operates, or how it appears to a user when used constitute this repertoire-subset.

Open software web sites and source webs All of the open source projects use the world Wide Web as a global communication platform. The *contents* and *links* that constitues the elements of the various interlinked web sources (Web-sites) related to an open source project serve as a repertiore-at-large. This subset almost constitutes all of the repertoire and is larger than the linked-web pages for explaining scenarios of usage as described earlier.

Development of open source software system requirements is a community building process, a complex web of socio-technical processes. It is a collaborative effort based on computer mediated communication that forms a complex social structure in the cyberspace. In open source software systems, the developers are generally the end-users of the systems (unlike in commercial/traditional systems), thus [Sca02] argues open source software can suffice with reliance on these informal and light weight or non rigorous software informalisms. I feel that these *software-informalism* can be identified as the genre-repertoire constituting the genres of communication actions in F/OSS communities, almost all of them manisfested in some electronic form in the Internet and the World Wide Web.

References

[OY94] Wanda J. Orlikowski and JoAnne Yates. Genre rrepertoire: The structuring of communicative practices in organizations. *Administrative Science Quarterly*, 39(4):541–574, December 1994.

[Sca02] Walt Scacchi. Understanding the requirements for developing open source software systems. *IEEE Proceedings - Software*, 149(1):24–39, 2002.

ICS 280: Theories and Practices in Collaboration

Position Paper for Week 8

Topic: New Collaboration Forms I: Radical Collocation,Extreme Collaboration

Sushil K Bajracharya

sbajrach@ics.uci.edu

Abstract

I discuss Extreme Programming [chr03] as a new form of collaboration in software development and summarize the arguments for applying extreme programming principles in large- open-distributed teams (like Free/Open Source projects) as discussed in various sources [MKL01], [KL01] and finally see what are the issues that need to be taken care of before considering applying extreme programming to large distributed projects [Cro01].

Extreme Programming: Extreme Collaboration in Software Development

There are several characteristics of eXtreme programming (XP) that makes it a very likely , if not exact, exemplar of Extreme Collaboration [Mar02]. Besides relying on normal electronic information tracking tools like e-mail, project tracking tools, spreadsheet or webpage, XP advocates the use of physical artifacts and facilities like - (i) Story Cards - for customers to put down 'What should be done' , (ii) Task Cards - for developers to put down 'How should it be done' and (iii) The Bullpen (War Room). Such practice carry in itself the essence of Distributed Cognition [**?**] and the practice of radical collocation for communication [TCKO00]. Similarly, the role structure in XP puts the *Customer* in constant communication with the *Developers* and additionaly suuplementary roles like of the *Coach* (mentor) and *Tracker* are defined. XP's philoshopy that a software project can be managed in terms of four variables *time, scope, resources* and *quality* (any one of these cannot be overemphasized without considering the effect on another) is also very similar to *timeboxing* as discussed in [TCKO00].

The core of XP lies in its 12 practices, (4 coding, 4 developer and 4 Business) given below -

1. Code and Design Simply
2. Refactor Mercilessly
3. Develop Coding Standards
4. Develop a Common Vocabulary (Metaphor)
5. Adopt Test Driven Development
6. Practice Pair Programming
7. Adopt Collective Code Ownership
8. Integrate Continually
9. Add a customer to the team
10. Play the planning game
11. Release Regularly
12. Work at a substantial Pace

Almost all of these practices require constant *communication* and *rapid feedback*. Among above 12, four practices rely on physical collocation and face-face-face communication, that can be replaced to some degree using electronic commincation media as listed in Table 1. Table 1 is a generalization of alternatives sought out for communication media in extending XP to DXP (Distributed eXtreme Programming) in a globally distributed project that is fully explained in [MKL01]. The findings of the experimental setup are summarized below -

1. No communication medium can completely replace or provide the same experience as face-to-face communication in a physically collocated environment. However in conditions that limit such physical collocation, combination of synchronous (video-conferencing) and asynchronous (e-mail) mode of electronic communication is effective.

XP Practices	DXP [1] Solution
Pair Programming	Videoconferencing, application sharing and personal familiarity
Planning Game	Application/Desktop sharing
On-Site Customer	*Virtual* on-site customer via videoconferencing and application sharing
Continuos Integration	Remote access to integration machines

Table 1: DXP Solutions to XP Practices that require physical collocation [MKL01]

2. Configuration Management (CM) Tools help to a great extent in maintaining code integrity and facilitates parallel development but *making mutually exclusive changes helps reduce merge conflicts*. Leveraging communication facilities beyond just using CM tools, like using a simple *e-mail token to serialize change access* while working on common code simplifies the task. This phenomenon is also very well described in [dRD03] where team members seek out solutions other than those provided by CM tools to resolve conflicts.

3. Other than these two major findings, there were several problems while practicing XP in a distributed environment, such as -

   - limitation of videoconferencing tool to handle more than two concurrent users
   - problem in capturing story cards in a text file (a solution of using WikiWikiWeb is proposed)
   - videoconferencing not allowing sharing applications
   - simple text editor not enough for brainstorming
   - bandwidth-network-power problems (for instance the participant in India had low bandwidth connection, from behind Firewall and frequent power outages)
   - differences in locale settings, for different keyboard layouts caused problems in electronic-conferences.

Although the solutions outlined in Table 1 cannot provide the same communication experience as in physically collocated environment, for some categories of projects that already has a loose coupling between team members such technologies suffice or even traditional electronic communication mechanisms like Message-boards, E-mail List servers, Web and Instant Messaging are enough to create a complex web of socio-technical interaction among the participants, of course these kinds of projects have special characteristics that make them lenient towards adoption of such communication tools [Sca02]. Arguments given in [KL01] in favour of the fact that XP can be practiced in large open source source communities more or less exploit this fact. As already stated, among the 12 XP practices, four are the most demanding of physical collocation. However, the argument given for the fact that they can be somwhat compromised in distributed-open projects goes as follows -

1. The Role of customers are not explicitly visible in such open projects. Most of the times, developers are customers sp issues can be resolved by participating in communication leveraging electronic communication media that furthermore, provides mechanism for persistence (storage, retrieval and search facility) of on going communication

2. Pair Programming is an issue that seems to have no better alternative in such projects. Solutions like remote pair programming using collaborative editors or even practicing serial pair programming rather than parallel pair programming is being considered as alternatives. (The argument for this is, four eyes sees better than two, but do you need to see at the same time? However the fact that pair programming is not just a matter of looking at the code at the same time makes this argument weaker in some aspect.)

3. Moderator based software builds and reliance in modern CM tools are taken as the mechanism for continuos integration.

4. High standard coding guidelines with a proper gatekeeper process (norms that all participants follow) and well-mainained user groups with fast responses for good planning. (Again the usage of specific genre of communication actions like BugZilla for bug-reports, well managed howt-to guides and design documents all available publicly helps to a great extent in planning)

Besides these difficult to practice principles other pratices are easier to adopt in open-distributed projects, for instance the notion of metaphor is very well captured by using *Design-Patterns* as a medium of communication as well as foundation for development.

Despite of all the good words in favour of XP it is arguably one of the most controversial software development paradigms. The bottomline is either XP is or is not for a project. XP has been criticized due to its extreme buttom up approach and its failure to lead projects to success where the strategy should be the opposite [Bin04]. I conclude by reiterating the main points presented in [Cro01] that put forwards five points to reconsider to make XP scale to large projects.

Loosely Coupled Teams. This becomes a necessity when the team would end up being more than a handful of participants, say more than 20.

Team Co-ordination Layer. Additional layer, outside the XP, to support team interaction. This replaces metaphor, pair programming, collective code ownership and on-site customer practices with the following -

Up-Front Architectural Lite. An incomplete but do-able architectural description of the project that ensures that the design knowledge permeates the team

Liaison. Members of various teams joining forces to develop the architectural-lite.

Team-Wise Planning Game (TPG). Allowing each sub-team to practice XP and facilitating planning across sub-teams. This includes concepts like *proxy customer*, where a sub-team on behalf of a customer can participate in TPG.

Finally, I would say no arguments for and against practices like XP would be completely valid because there are people and projects that can or cannot practice Extreme Collaboration, its not for everyone and every project. But, when applied to a suitable one, it is unbeatable. Furthermore, the quest should be to look into the organization in the chaos in such radical collocations, the balance between human and electronic communication network that gives surprisingly good results in collaborative practices.

References

[Bin04] Andrew Binstock. Not so extreme programming. *SD(Software Development) Times*, May 1 2004.

[chr03] chromatic. *Extreme Programming - Pocket Guide*. O'reilly And Associates, first edition, July 2003.

[Cro01] R. Crocker. The 5 reasons xp canŠt scale and what to do about them. *XP2001, Italy*, May 21-23 2001.

[dRD03] Cleidson R. B. de Souza, David Redmiles, and Paul Dourish. "breaking the code", moving between private and public work in collaborative software development. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 105–114. ACM Press, 2003.

[KL01] Michael Kircher and David L. Levine. The xp of tao: extreme programming of large, open-source frameworks. pages 463–485, 2001.

[Mar02] Gloria Mark. Extreme collaboration. *Commun. ACM*, 45(6):89–93, 2002.

[MKL01] A. Corsaro M. Kircher, P. Jain and D. Levine. Distributed extreme programming. *XP2001, Italy*, May 21-23 2001.

[Sca02] Walt Scacchi. Understanding the requirements for developing open source software systems. *IEEE Proceedings - Software*, 149(1):24–39, 2002.

[TCKO00] Stephanie Teasley, Lisa Covi, M. S. Krishnan, and Judith S. Olson. How does radical collocation help a team succeed? In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 339–346. ACM Press, 2000.

ICS 280: Theories and Practices in Collaboration

Position Paper for Week 10

Topic: Distibuted Software Development

Sushil K Bajracharya

sbajrach@ics.uci.edu

Conflicts and Collaboration in Free/Open Source Software Development

Conventional software design principles advocate problem decomposition by breaking down the entire problem to well managable modules that are flexible and amenable to future changes (evolution). Consequently, *Recomposition* [Gri98] - putting those modules together, serves as an articulation work that constitutes the coordination and negotiation necessary to get the work done. Empirical studies show global dispersion and distance among participants introduce delay in such collaborative efforts [HMFG00] . This has been the central theme of our discussion for Distributed Software Development.

An important aspect of thinking about design problems in terms of *recomposition* is that it allows us to look at the social and as well as technical aspects of conflicts and negotiations that take place during the articulation work. In communities like Free/Open Source Software (F/OSS) teams where the contributors are inherently distributed and the major motivation for collaboration is because of the common belief in values and culture in the virtual organization, we get to see a different picture of conflicts, debates and negotiations taking place. Especially how computer mediated communication (CMC) (almost the only one form of communication medium that exists in such communities) provide the platform for initiating discussions and well as resolving conflicts fostering collaboration is an intersting phenomena to study.

[ES03] presents two examples of such articulation among participants of GNUe, an open source project targeting development of a freely available ERP (Enterprise Resource Planning) solution. [ES] extends this example to refute and support some of the earlier assertions made about conflicts and negotiation in collborative work practices. A brief summary of these facts is presented further in this document.

Both of the debates took place in the GNUe IRC channel. All the discussions going on in the IRC channel are stored persistently and anyone can read previod message logs and start a new discussion later in the IRC channel itself or mailing lists. But, both of the conflicts discussed in [ES03] took place entirely in the GNUe IRC channel, the archive of the discussions were studied by the authors to come to their assertions about use of CMC and nature of the articulation going on. The first debate was initiated by an outsider to GNUe indicating his concern about *use of a non-free graphics tool* to produce a graphic on the GNUe website. This discussion lasted for 1 day with total 7 contributors, 9 regular and 8 infrequent contributors. After the outsider points out the use of non-free graphics tool, one of the core contributor defends by saying that he does not object using the tool as it does not put any restictions on GNUe from being a free software product. He takes this discussion to the contributor using the non-free software. Later, the outsider starts discussion with the original user of the non-free software and makes him aware of some alternatives available. The user of the non-free software immediately installs and tries out the alternatives suggested, and goes on further discussing with the outsider. As more people join in the discussion this leads to further discussion on using appropriate graphics file formats and tools. The issue finally gets resolved with the user of the non-free graphics tool accepting to produce the graphics later using a free software.

The second debate is also of similar nature where a frequent contributor put forwards his complain about needing to install a non-free software to edit the documentation for the project. In this case the debate lasted 3 days with total 24 contributors, 9 regular and 15 infrequent contributors. The choice for the non-free software for preparing documentation was the teams inability to get a free alternative document preparation tool ready immediately. The team resorted to using a free software that had a component that was not free. This debate again brought together many members, even provocative expressions of beliefs and arguments, apologies etc. Finally the settlement was made when the instigator "backed off" on peer pressure as lot of the contributors commented the discussion was futile and also upon reaching a conclusion that a completely free version of the tool will be used in future when available

and, the documents that will be distributed for time being would not require the users to install non free software. Settlement in this case was quite different from the earlier one. It took more time (3 days) and also the conflict in this case delayed the work of some contributors. The instigator, an insider in this case also backed down due to peer pressure.*There appeared to be less resistance to change regarding free versus non-free tools when the change involved an impact to one personŠs work and was not of a procedural nature.*

Given all these facts the following issues are reflected from the conflicts and negotiation techniques in above examples -

- Immediate acceptance of outside contributors
- Belief in free software, value in community and value in cooperative work
- Involvement in detailed philoshopical discussions not taken as interruption to the work but as a reinforcement of values and beliefs

Drawing conclusion from these results [ES] presents an analysis of few assertions made about conflicts and coordination in cooperative work as follows -

Anonymity and physical separation does not always contribute to conflict. This is asserting the fact that although conflicts occur in daily work practices, strong cultural ties diminish the effects of anonymity and distance separation. In addition, the frequent contributors are not really anonymous as their credentials are available on public web sites. Authors agree with findings by Lea and Spears [LS91] by saying -

> They suggest that the social context influences the occurrence of de-individuation. If de-individuation occurs with immersion in a group, then this enhances the salience of the group and strengthens its norms. However, if the group identity is not already established, then de-individuation only serves to strengthen oneŠs sense of individuality, and weakens group norms.

It is not always true that conflicts are unlikely to be resolved if participants argue from entrenched positions. This assertion refutes an earlier claim made by Easterbrook et al [SMEC93],[Pac]. It highlights the fact that the recommendation for handling *competitive conflict* by separating people from their positions to help support creativity in resolving the conflict is not always possible or desirable especially in a virtual community. In such communities the values in community and cooperative work motivates people to try and resolve differences without a third party/manager.

Articulating conflicts helps in its resolution With this assertion the authors in [ES] agree with previous studies like that in [SMEC93] and [Pac]. They agree *differentiation* - a group process of identifying and understanding the dimensions of a conflict by making the conflict explicit, identifying issues involved, and recognition of individual views by other members of the group play an important role in conflict resolution as illustrated by the two examples given above.

Finally, I will conclude this document by saying that there is certainly much more to look at when we deal with analyzing conflicts and negotiations among contributors participating in collaborative efforts. Strong beliefs in values and culture of organization and strong commitment to contribution as seem to exist so persistently among members of FOSS teams are crucial factors to their success despite of several impediments they face like geographic separation and differences in time zones.

References

[ES] Margaret S. Elliott and Walt Scacchi. Free software: A case study of software development in virtual organizational culture. *ISR Technical Report UCI-ISR-03-6*.

[ES03] Margaret S. Elliott and Walt Scacchi. Free software developers as an occupational community: resolving conflicts and fostering collaboration. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 21–30. ACM Press, 2003.

[Gri98] Rebecca E. Grinter. Recomposition: putting it all back together again. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 393–402. ACM Press, 1998.

[HMFG00] James D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. Distance, dependencies, and delay in a global collaboration. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 319–328. ACM Press, 2000.

[LS91] M. Lea and R Spears. Computer-mediated communication, de-individuation and group decision-making. *International Journal of Man-Machine Studies*, 34:282–301, 1991.

[Pac] R. C. Pace. Personalized and depersonalized conflict in small group discussions: An examination of differentiation. *Small Group Research, 21(1), 79-96*.

[SMEC93] J. S. Goodlet M. Plowman M. Sharples S. M. Easterbrook, E. E. Beck and C. C.Wood. A survey of empirical studies of conflict, cscw: Cooperation or conflict? *(pp. 1–68). London: Springer-Verlag*, 1993.