

Interpreting Activity Theory as a Software Engineering Methodology

Cleidson R. B. de Souza

Universidade Federal do Pará, Brazil and University of California, Irvine, USA

cdesouza@ics.uci.edu

Abstract. Often the adoption of a theoretical framework is a matter of presentation or “packaging.” Different communities of practices generally differ in their use of terminology and practices. In this paper, I begin with a software engineering definition of “methodology” and examine ways activity theory may be interpreted according to this definition. My goal is to explore ways that activity theory may be made more accessible to software engineering researchers and practitioners.

Introduction

CSCW researches have sought for quite some time to understand how field work findings can be used to inform the design of computer-based systems. Several analytical frameworks have been used in this process, such as distributed cognition (Hutchins 1995), ethnomethodology (Garfinkel 1967), activity theory (AT), among others. These frameworks have informed the design of different computer-based applications (see for example Bodker (Bodker 1991) and Nardi and Redmiles (Nardi and Redmiles 2002) for examples of AT usage). However, none of these frameworks can be directly applied to the design of computer systems, since they were created for purposes other than software design. In other words, these frameworks do not provide ready-made solutions that can be directly applied to design efforts; they need to be adapted to be useful for designers.

Kaptelinin and colleagues (Kaptelinin, Nardi et al. 1999) created the activity checklist aiming to support the AT framework and that's the same goal of this workshop: "operationalize AT into a practicable research methodology as well as a practicable requirements analysis and design methodology in CSCW".

In this paper, I argue that key concepts from the software engineering community can be used to point possible research directions in operationalizing the AT framework. To be more specific, I use the concept of methodology, as defined in the software engineering community by Rumbaugh (Rumbaugh 1995). According to Rumbaugh, a methodology is composed of four different parts, and those parts can be related with some parts of the AT framework. On the other hand, other components of a methodology can not be associated with the AT framework, suggesting aspects to be explored to facilitate the design of computer-based systems. The experience reported in this based is based on a set of field studies performed in our research group in the last few years. The data collected on these field studies has been analyzed using the activity theory framework and are described in (Shukla, Nardi et al. 1998), (Collins, Shukla et al. 2002) and (de Souza and Redmiles 2003).

The rest of the paper is organized as follows. The next section describes the concept of software development methodologies. The following section briefly discusses how the components of this methodology could be used in CSCW focusing in the issues that need to be addressed. After that, I discuss how elements of the AT framework relate to the components of a methodology. This suggests other areas of AT research that need to be explored. Finally, conclusions and future work are described.

Methodologies in Software Engineering

Essential to the understanding of this paper is the concept of methodology as defined in the software engineering community. Indeed, Rumbaugh (Rumbaugh 1995) argues that a methodology should have the following set of components¹:

1. A set of *modeling concepts* that are used to capture information about a problem and its solution. For example object-oriented methodologies use concepts such as objects, classes, inheritance, methods and so on;
2. A set of *notations* and *views*, to represent the modeling concepts for human understanding and modification. Most of the time, notations are

¹ There is terminology problem here. Rumbaugh (1995) uses the term *method*. However, in his work it is clear that he is referring to a *methodology* instead of a method. For example, he refers to his own work, the OMT - Object Modeling Technique - (Rumbaugh, J., Blaha, M., Premerlani, W., et al., 1991)) as a *method*, while most software engineering literature considers OMT a *methodology*. According to Ghezzi, C., Jazayeri, M. and Mandrioli, D. (1991), *methods* are general guidelines that govern the execution of some activity; while a *methodology* packs together *methods* and techniques to promote a certain approach to solving a problem by pre-selecting the methods and techniques to be used. Thus, although Rumbaugh uses method and methodology as synonyms, they are not.

represented graphically. An example of a set of notations is the Unified Modeling Language (UML) (Rumbaugh, Jacobson et al. 1999), while the views are the several UML diagrams, such as activity, classes and so on;

3. A *development process* that is used to guide the development of the models. Models are constructed using the modeling concepts and represented using the notations and views. The process describes which models to construct and, also, how to construct them. In this case, a good example is the Unified Software Process (Jacobson, Booch et al. 1999).
4. And, finally, a collection of *hints* and *rules-of-thumb* that are used to guide, evaluate, or improve the development process, and therefore the construction of models.

In summary, a methodology consists of a *development process* that guides the construction different *views* and *notations* that represent the *modeling concepts*. *Hints* and *rules-of-thumb* assist all those activities. In this case, a methodology would be particularly interesting in helping designers gathering requirements for the system and, specially, in making “designers to question the taken-for-granted assumptions embedded in the conventional problem-solution framework” (Anderson 1991). The rest of the development process (implementation, testing, and maintenance) is not important for this discussion, then, in the rest of the text term methodology will just a methodology for requirements elicitation, analysis, specification and validation.

Software Engineering Methodologies and CSCW Design

The definition of methodology presented earlier can be used by CSCW researchers to help in understanding the several issues to be addressed in order to cultivate a methodology that uses field studies to inform design. Indeed, some of the issues are described below, but they are not, in any sense, exhaustive. These issues are discussed according to the components of a methodology discussed in the previous section.

The lack of a set of *modeling concepts* creates a critical communication problem between designers and field workers (Jirotko and Wallen 2000). A possible solution is proposed by Hughes and colleagues (Hughes, O'Brien et al. 2000), that is the notion of viewpoints. Another approach is proposed by Erickson (Erickson 2000), who suggests workplace patterns as a *lingua franca* to facilitate the communication between all personnel involved in the development process.

Once these concepts are defined, the next question is how they are represented? Which *notations* and *views* are necessary? Again, the concept of viewpoints (Hughes, O'Brien et al. 2000) could be used as an answer to this question. A

notation to represent the modeling concepts is very useful allowing communication among all the personnel involved in the development.

Which *development process* should be used and in which level of abstraction should it be described? In this case, lessons from the software engineering community can be used during the design and implementation phases. The problem is located on how to define the requirements engineering phase.

Finally, what are the *hints* and *rules-of-thumb* that can be used during the development process proposed by the methodology? In this case, an interesting approach to follow is the idea of workplace patterns. Not as a *lingua franca* for design as suggested by Erickson (Erickson 2000), but as a repository of good practices to be used during the requirements engineering activities. Indeed, patterns are representations that condense the experience and knowledge of the good practices in a discipline (Gamma, Helm et al. 1995). They express a common solution to a recurring problem in this discipline. Workplace patterns could be developed to help the design phase in creating, preserving and enhancing features identified during the field studies. Those patterns would add up with other patterns for software development, such as object-oriented design (Gamma, Helm et al. 1995), programming language (Grand 1998), and so on.

Activity Theory as a Methodology?

In this section, I discuss in more details, based on my own experience using AT, how the parts of the AT framework can relate to the components of a methodology and help to answer some of the issues raised in the previous section. Note, that I am not necessarily proposing an AT-based methodology for systems design, I am only using the idea of software engineering methodologies to suggest possible research areas in activity theory so that it can be operationalized and adopted by others. Indeed, others (see (Mwanza 2001) and (Korpela, Soriyan et al. 2000)) have proposed AT-based methodologies. However, note that according to the concept of methodology adopted by the software engineering community, Mwanza's work (Mwanza 2001) describes mainly the development process of a methodology and introduces an additional notation, which is not clearly explained. In other words, she does not present all the components of a methodology. On the other hand, Korpela *et al.* (Korpela, Soriyan et al. 2000) only *implicitly* describes the development process of a methodology and does not discuss any other additional components.

Modeling Concepts and Notations

During the usage of AT in CSCW research is very common to build diagrammatic models such as the ones proposed by Engeström (Engeström 1999). For example,

de Souza and Redmiles (de Souza and Redmiles 2003) present one of these diagrams describing a software development team:

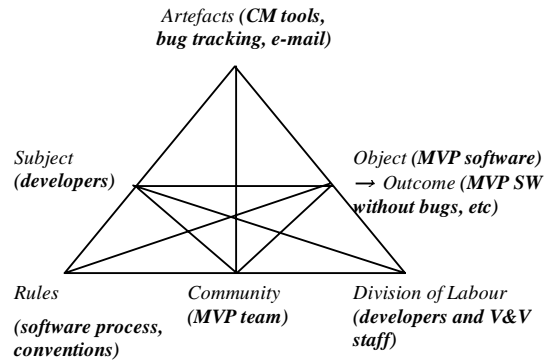


Figure 1. The MVP activity.

Engeström's terminology and diagrammatic notation have used similarly as in the concepts of a software engineering methodology by different authors (Korpela, Soriyan et al. 2000) (Mwanza 2001) (de Souza and Redmiles 2003). The terminology used describe the initial set of *modeling concepts* proposed by Rumbaugh (Rumbaugh 1995), while the diagram was used as a *notation* or *view* to describe the phenomena observed. Furthermore, there seems to be some overlap with object-oriented analysis (Rumbaugh, Blaha et al. 1991), which is largely used by the software engineering community. Thus, although there is still a great deal of craft involved in becoming acquainted with and applying activity theory, other members of the research group and I have experienced many positives in our analyses in different work settings and anticipate the methodology becoming more refined and documented. On the other hand, embedding Engeström's terminology and notations into a methodology will imply in additional learning by software engineers, which might be very problematic. In this case, the work of Korpela *et al.* (Korpela, Soriyan et al. 2000) trying to reduce the gap between use-case diagrams and activity theory diagrams is very important.

Development Process

Iterative refinement of the model described in Figure 1 appeared to be an open-ended process. The stopping point occurred when all phenomena identified in the data collection was accountable using the model. This seems to suggest that the application of AT still needs to define a clear-cut set of steps to be followed during its application (a process). Similarly to software development processes, this process might be specified in low or high-level details, meaning that it can

prescribe in more or less details the steps that need to be performed. Of course, I do not mean to say that there is no guidance in how to apply activity theory, but I argue that more details need to be provided, in attempts similar to the AT checklist (Kaptelinin, Nardi et al. 1999) and (Mwanza 2001). By providing such guidance, AT might be made more accessible, especially, for novice practitioners and software engineers.

On the other hand, my experience in using the AT framework and experiences from my colleagues have shown many positives to activity theory. It is open ended, which, although a challenge, allows some flexibility for the introduction of new ideas and refinements. This raises the question of how to provide guidance in the usage of the AT framework without reducing its flexibility. Furthermore, the AT framework is noninvasive, using open-ended interviews or even more informal observations of work. It readily yields to iterative refinement. When more detail is needed in a model, additional activities may be named and analyzed.

Hints and Rules of Thumb

In software engineering methodologies, a collection of hints and rules-of-thumb are used to guide, evaluate, or improve the development process, and therefore the construction of models. Similarly, the AT research community needs to be able to document and reuse the experience of using the AT framework. A common way of doing that is documenting patterns (Gamma, Helm et al. 1995). Patterns are a way of recording knowledge used to solve recurring problems. When these solutions are documented, they can be made available to others that can reuse the knowledge and experience embedded in these solutions. A potential problem, in this case, it to be able to represent the intrinsic characteristics of the setting where the solution was applied. In other words, the challenge is to identify and represent the context where the application of the AT was successful so that one does not try to reuse solutions in inappropriate contexts.

Mwanza (Mwanza 2001) claims to present three rules-of-thumb to facilitate the decomposition of the situation's activity system into smaller sub-activities. However, these rules-of-thumb need to be applied at a particular moment in her methodology, which suggests that they are more likely to be part of the development process than rules-of-thumb.

Conclusions

In this paper I discuss how the software engineering definition of "methodology" can be used to interpret the activity theory framework. A "methodology" is composed of four different components: a set of concepts, a notation to represent these concepts, a process to guide the construction of the models, and finally, a set

of rules of thumb to help the several steps performed. I discussed how the AT framework has been used in such way that provides two of these components, namely the modeling concepts and the notation. However, a development process that guides the usage of the concepts and the construction of the models is not clearly provided, although attempts have been made. Furthermore, it is not easy to identify the rules-of-thumb that provide hints in how to use the AT framework. I believe that these aspects are important and need to be properly explored by researchers so that activity theory may be made more accessible to software engineering researchers and practitioners.

Acknowledgments

The authors thank CAPES (grant BEX 1312/99-5) and NASA/Ames for financial support. This effort was also sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0599. Funding also was provided by the National Science Foundation under grant numbers CCR-0205724 and 9624846. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Laboratory, or the U.S. Government.

References

- Anderson, R. (1991). "Representation and Requirements: The Value of Ethnography in System Design." Human-Computer Interaction **9**: 151-182.
- Bodker, S. (1991). Through the Interface: A Human Activity Approach to User Interface Design. Hillsdale, NJ, Lawrence Erlbaum.
- Collins, P., Shukla, S. and Redmiles, D. (2002). "Activity Theory and System Design: A View from the Trenches." Computer Supported Cooperative Work (CSCW) - Special Issue on Activity Theory and the Practice of Design **11**(1-2): 55-80.
- de Souza, C. R. B. and Redmiles, D. F. (2003). Using Activity Theory to Understand Contradictions in Collaborative Software Development (submitted). Automated Software Engineering, Montreal, CA, IEEE Press.
- Engeström, Y. (1999). Activity Theory and Individual and Social Transformation. Perspectives on Activity Theory. Y. Engeström, R. Miettinen and R. L. Punamäki. Cambridge, UK, Cambridge University Press: 19-38.
- Erickson, T. (2000). interdisciplinary design: towards pattern languages for workplaces (chapter 13). Workplace Studies. P. Luff, J. Hindmarsh and C. Heath. Cambridge, UK, Cambridge University Press: 252-261.
- Gamma, E., Helm, R., Johnson, R., et al. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA, Addison-Wesley.

- Garfinkel, H. (1967). Studies in Ethnomethodology. Englewood Cliffs, New Jersey, Prentice-Hall.
- Ghezzi, C., Jazayeri, M. and Mandrioli, D. (1991). Fundamentals of Software Engineering, Prentice Hall.
- Grand, M. (1998). Patterns in Java: a catalog of reusable design patterns illustrated with UML, John Wiley & Sons. Inc.
- Hughes, J., O'Brien, J., Rodden, T., et al. (2000). Ethnography, communication and support for design (chapter 9). Workplace Studies. P. Luff, J. Hindmarsh and C. Heath. Cambridge, UK, Cambridge University Press: 187-214.
- Hutchins, E. (1995). Cognition in the Wild. Cambridge, MA, The MIT Press.
- Jacobson, I., Booch, G. and Rumbaugh, J. (1999). The Unified Software Development Process. Reading, MA, Addison Wesley Longman, Inc.
- Jirotko, M. and Wallen, L. (2000). Analyzing the workplace and user requirements: challenges for the development of methods for requirements engineering (chapter 12). Workplace Studies. P. Luff, J. Hindmarsh and C. Heath. Cambridge, UK, Cambridge University Press: 242-251.
- Kaptelinin, V., Nardi, B. and Macaulay, C. (1999). "The activity checklist: a tool for representing the "space" of context." Interactions 6(4): 27-39.
- Korpela, M., Soriyan, H. A. and Olufokunbi, K. C. (2000). "Activity Analysis as a Method for Information System Development." Scandinavian Journal of Information Systems(12): 191-210.
- Mwanza, D. (2001). Where theory meets practice: A case for an Activity Theory based methodology to guide computer system design. INTERACT'2001, Oxford, UK, IOS Press.
- Nardi, B. and Redmiles, D. (2002). "Introduction to the Special Issue on Activity Theory and the Practice of Design." Computer Supported Cooperative Work, The Journal of Collaborative Computing 11(1-2): 1-11.
- Rumbaugh, J. (1995). "What is a Method?" Journal of Object-Oriented Programming 8(6): 10-16,26.
- Rumbaugh, J., Blaha, M., Premerlani, W., et al. (1991). Object Oriented Modeling and Design, Addison-Wesley.
- Rumbaugh, J., Jacobson, I. and Booch, G. (1999). The Unified Modeling Language Reference Manual. Reading, MA, Addison Wesley Longman, Inc.
- Shukla, S., Nardi, B. and Redmiles, D. (1998). Hit squads & bug meisters: discovering new artifacts for the design of software supporting collaborative work. Conference on Human Factors and Computing Systems, Los Angeles, CA, USA, 363 - 364, ACM Press.