Supporting the End Users' Views

David F. Redmiles
Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
+1 949 824 3823
redmiles@ics.uci.edu

ABSTRACT
End users of software have the right to systems that are both useful and usable, a property termed usability in the software and human-computer interaction communities. Unfortunately, it is not obvious what methods or techniques developers of software should adopt in order to achieve good usability in a product. There are a confounding number of questions. How can different points of view among end users be incorporated into a software development process? What does it mean to treat software developers as end users, namely of software tools? How do the limitations of software practice, such as minimizing time to release, affect what information can be collected and used to make usability decisions? This paper presents a variety of possibilities for supporting all the end users' views in a software development activity. Both tools and methods are suggested, roughly organized according to the different activities in software development. Moreover, end users are defined to be a variety of stakeholders in a software development project, including at the very least the end users of a product but also developers who are end users of software tools.

Categories and Subject Descriptors
D.2.2 [**Software**]: Design Tools and Techniques – *evolutionary prototyping, user interfaces.*

General Terms
Human Factors, Design, Management

Keywords
Software Engineering, Usability Engineering, Human-Computer Interaction, Knowledge-Based Systems, Cognitive Theory, Social Theory, Design, Design Environments, Event Monitoring, Organizational Memory, Activity Theory

1. INTRODUCTION
The creation of software systems is more challenging today than ever before. This statement seems paradoxical in light of decades of research into computer systems. However, these decades of research are intertwined with decades in which software technologies have been applied to new and increasingly complex problem situations. The increased complexity comes from many factors including a change in the nature of software from static to interactive, an expanded scope for systems including the shift from single to multiple end users, radically new problem domains such as office and other workplace settings, an increasingly sophisticated population such as a generation of end users who grew up with personal computers and computer games, and, finally, new deployment mechanisms such as the Internet. Simply stated, the creation of software is more challenging today because it is applied to increasingly complex problem situations.

As we realize from the trends mentioned above, the term complexity takes on a new meaning compared to its traditional usage in computer science. Previously, "complexity" referred to a measure of a system's computations from an algorithmic perspective. However, today, when factors such as human end users and diverse deployment environments are involved, a new kind of complexity emerges, one that is measured in terms of end users and workplace settings.

A duality emerges around this notion of complexity. First, in order to create software systems suitable for interactive applications in diverse settings and for diverse collections of end users, it is necessary to have a diverse software development team. People with knowledge of software engineering are required of course. However, people with knowledge of human-computer interaction, group interaction, organizational issues, and possibly knowledge-based user interfaces are required as well. Thus, the first part of the duality is the need for a diverse software development team to tackle the new breed of complex software. The second part of the duality is ironic. The kinds of problem domains that require a diverse team would never have been tackled were it not for the early work of a few inter-disciplinary pioneers. To be clear, the problems were there and thwarting software development projects. They were simply never clearly elucidated until new disciplines turned their interest to the kinds of artifacts software engineers were developing.

Recently, success cases of interdisciplinary design have begun to be documented. One recent book documents contributions of the field of human-computer interaction to design [48]. One particular case study documented how a cognitive analysis of a proposed re-design for a workstation would lead to less efficient completion of tasks by the end users compared to the existing workstation [3]. Another recent book documents the contributions of a variety of cognitive and social theories to the

AVI 2002, May 22-24, 2002, Trento, Italy.

design of systems [35]. For example, one study illustrates how a situated action perspective in field studies led to deep understanding of the requirements for building a document-based tool to support lawyers [54]. As a final example, an upcoming special issue of a journal documents how a social and psychological theory of activity contributes to design of software [38]. One study in that issue documents problems with process-centered software development environments with respect to the activity theory framework [4].

Taking an interdisciplinary view of the general problem of developing complex software systems leads to a variety of possibilities. Software engineering provides insight into the kinds of artifacts that must be engineered and ways of engineering (developing) them. Usability engineering provides insight into the kinds of data that can inform design and ways of collecting that data. Human-computer interaction provides insights into the cognitive needs of individuals. Knowledge-based systems (intelligent user interfaces) provides some unique techniques for implementation. Computer-supported cooperative work provides information about short and long-term inter-personal communication. Ethnographic methods and social and psychological theories, such as activity theory, provide an understanding of the workplace, a greater degree of realism.

The above list could continue. However, there are a few simple points. The increasing interaction among researchers from many disciplines is revealing the complexity of the new "software crisis" and new aspects to coping. Innovative combinations of many research disciplines are essential.

The remainder of this paper illustrates how all of the above-mentioned disciplines can weave together to create software tools and methods that support all of the end users, both end users of an intended product as well as end users of tools used to create products. These tools and methods were created out of a multi-disciplinary approach with an principal goal to make the development of complex, interactive systems a success. They fall under the rubric of a human-centered rather than an artifact-centered approach.

2. A Roadmap to Human-Centered Software Development

Software engineering has favored process views of software development and the software lifecycle for many years. Figure 1 shows an adapted lifecycle. The figure only loosely represents a process or lifecycle. More precisely, it represents a set of activities with loose connectivity. The activities all support a human-centered software development perspective. The activities include observation, design, use, and review. These activities support one another although the figure illustrates only some of the more typical connections and reasons for the connections. The figure resembles other iterative design methods (e.g., [21]), scenario-based design (e.g., [8] [41] [47]), and some approaches to design by software reuse (e.g., [17]). It is also inspired by early work on user-centered system design (e.g., [39]), cognitive studies of end users (e.g., [7]), and participatory design (e.g., [23]). The common elements are that the views of all stakeholders in the software development project are considered and that a software product evolves through data-driven feedback.

When possible, software developers should observe actual end users of proposed software products and their organizational context. They may need to interview various stakeholders, people affected by the proposed software. Observations lead to design of a software product based, presumably, on more complete requirements. The product may be deployed with software agents to monitor its use. Automated and manual usability information may be stored for review. Re-design and re-development of a software product leads to better fit with the end users and their workplace setting.

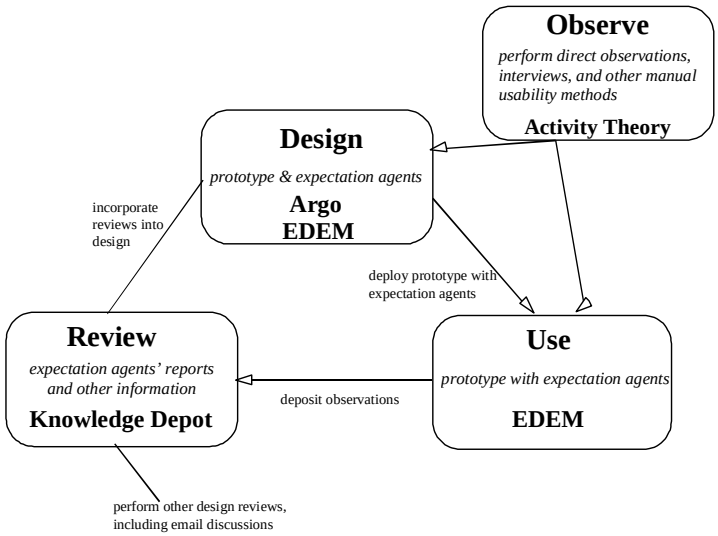


Figure 1: Activities in Human-Centered Software Development

The activity represented by this figure is human-centered in two ways. First, the emphasis on prototypes and evaluation of usage maintains a focus on the requirements of end users and other stakeholders in the project. Second, there is an underlying assumption that the software developers need cognitive support for managing the complexity of usability information and incorporating that data and other information into design. Thus, the applications being developed are engineered to meet the human cognitive needs of the end users and the software tools used to develop those applications are engineered to the human cognitive needs of the software developers.

The figure serves as a roadmap to the remainder of this paper and, to a degree, human-centered software development in general. The activities are labeled with names of methods and research tools that were used to explore the different activities and, where appropriate, automation. Observations about end users and other stakeholders, as well as information about the organization in which they will use software, may be analyzed according to activities (activity theory). A software design environment (Argo) provides software developers with knowledge about what constitutes a good design and supports the design of prototype applications. A software substrate (EDEM) provides support for collecting data about actual system usage. Data about the actual usage as well as other information related

to the design of a system can be recorded and elaborated upon by a group memory tool (Knowledge Depot).

3. Observing End Users and their Workplace

There is a great deal of work in the requirements engineering and software engineering communities on better articulating requirements. Some of the approaches emphasize multiple stakeholders and negotiation (e.g., [5] and [20]), others emphasize a formal analysis basis (e.g., [29]) or modeling basis (e.g., [40]). However, the communities of ethnographic researchers (e.g., [37]), researchers in computer-supported cooperative work (e.g., [1]), and researchers in cognitive psychology (e.g., [28]) have taken a more general approach. They have begun to look at how people interact with artifacts in their environment including the organizational issues.

Ethnography and software development come together when a project requires an understanding of the organizational and workplace context in which the proposed software will be deployed. Too often projects fail because these contexts were not considered early enough in a project. Ethnography encompasses a large number of methods for making observations "in the field." In general, it is a practice of interviewing and observing people on their jobs and in their workplaces. One approach to applying ethnography to the practice of design is activity theory [37].

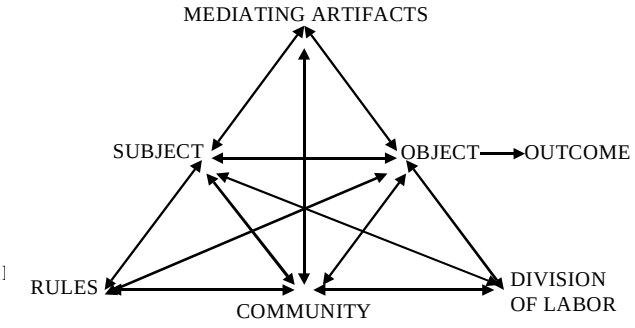


Figure 2: Engeström's Activity System Model [14]

Figure 2 is a classic illustration of the components of activity theory as set down by Engeström [14]. Subjects are people within a community that work with objects to obtain an outcome. Rules within the community determine the behavior of subjects and their access and their interaction with objects. The division of labor also determines at a higher level who performs what actions in a community. Mediating artifacts (sometimes called mediating means, instruments and by some, tools) help subjects manipulate objects. They have a history within a community such as a traditional set of tools.

In one study of a software support organization, the activity was customer support, the subjects were customer support engineers, and the common object was a problem resolution document. These documents had specific rules for completing them (see [9] for details). The larger community consisted of the authors of the problem resolution documents, consumers including managers and other software engineers, and customers who under certain conditions had access to the documents. Job descriptions and customer contracts determined the assignment of roles within this community. In this application of activity theory, the field study consisted of interviews with 32 people. The interviews provided information about the use of software tools to support the customer support activity and in particular information pertinent to the redesign of software tools for supporting this activity. As the information was analyzed with the activity theory model, it became apparent that there were contradictions in the organizational goals. For instance, the rules and division of labor were designed for customer support engineers to produce problem resolution forms. But, another set of rules and division of labor required the same individuals to provide prompt response and additional care to certain customers. Identifying the conflict enabled managers to realize more clearly how the software system was failing and could not solve the problem of getting all problem resolution reports fully documented and filed on time. For other results, the reader is referred to [9] and in general the special issue of the journal in which it appears [38].

Designing software for real use places a burden on the designers to understand not just their end users, but the workplace and organizational context in which software will be deployed. Ethnographic methods including activity theory can support software developers in their search for these points of views corresponding to the many stakeholders.

4. Supporting Design with Knowledge-Based Users Interfaces

In medium and large-scale software development projects, good design is critical to efficiency of the system but more importantly to correctness and future maintenance. However, a quandary in software engineering, as Brooks states it, has been that only "great designers create great designs" and there are a small minority of great designers in the world [6]. Hence for many years, some software researchers have sought to augment typical designers with tools that could supply information they needed to create good designs [13]. Some of the earliest work researched plan-based representations of software developers' knowledge (e.g. [52]) and applying representations of plans when software developers needed it (e.g., [44] and [15]). Some of these environments provided quite elaborate support for different aspects of a person's working style providing a catalog of examples to refer to, access to explanations, the ability to add new "knowledge," and critics ([18] and [16]).

On-going work in this general area led to a re-examination of the cognitive basis for software design and resulted in the Argo design environment illustrated in Figure 3 (see [45] and [46] for details on Argo). This interface presents several features tied to specific results from the cognitive studies of design. From one perspective, designers use Argo/UML much like they would use other object-oriented design tools: they place class, state, and use case icons in diagrams and draw relationships between them (upper right of Figure 3). However, while designers work, design critics analyze the design and provide helpful advice. The "to do"

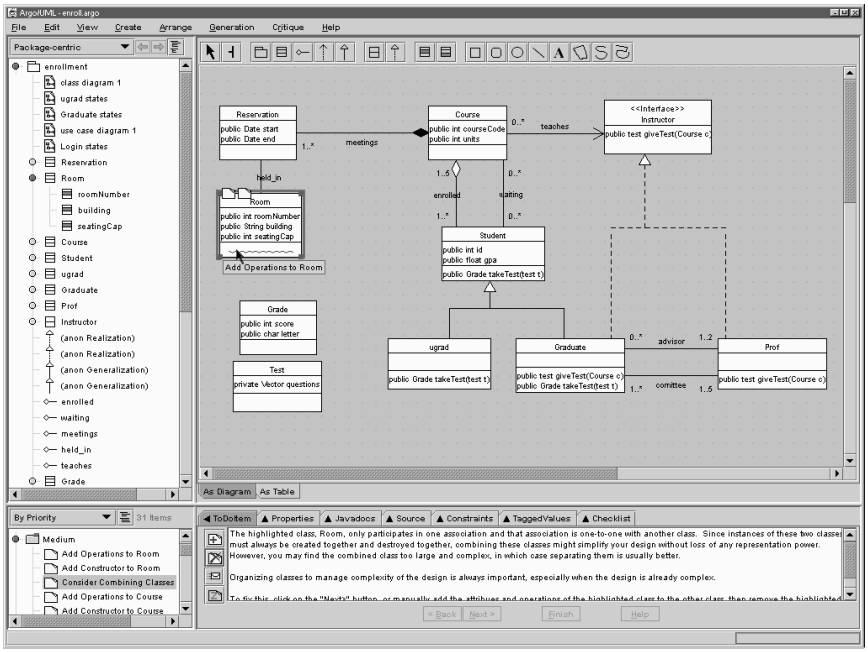


Figure 3: The Argo/UML Domain Oriented Design Environment

list (lower left) presents and organizes advice about pending design changes.

Intuitively, feedback and checklists are useful devices in our everyday lives. Previous work in applying rule-based technologies to design analysis illustrated how knowledge-based technology could be useful (see e.g., [36], [15], and [19]). However, the goal of finding the best way to provide these kinds of devices and the goal of discovering what other devices could support software designers led to a re-examination of the cognitive theories that existed about design in various domains.

In the domain of architectural design, Donald Schön had proposed the theory of reflection-in action, that designers do not conceive of designs fully formed but instead postulate one solution, evaluate it, reflect on its failures, and revise it, repeating the reflective activity when there is another breakdown or failure [49] [50]. Although superficially, this theory resembles a simple notion of feedback, the details address issues of when and how designers are best served by feedback. Specifically, designers should receive feedback while they are in the context of design, while they are making decisions about the design. Traditionally, software analysis tools provide feedback only after a design is submitted for review. In this case, feedback comes separately from design time. Moreover, most analysis tools require a complete design in order to operate. Again, Schön's theory provides a distinction in that designers reflect on partial designs. The notion of critics used in Argo, and other critiquing systems support feedback to designers in the context in which they are making design decisions and based on partial designs.

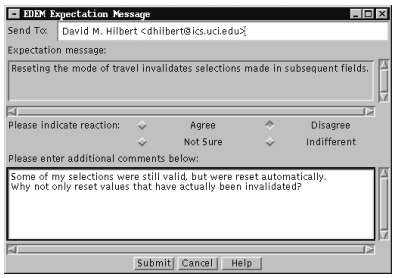
Another subtle example of applying cognitive theory to develop features for the Argo system derived from studying opportunistic design [24] [55]. This theory explains that although designers plan and describe their work in an ordered, hierarchical fashion, in actuality, they choose successive tasks based on the criteria of cognitive cost. Simply stated, designers do not follow even their own plans in order, but choose steps that are cognitively the least cost among alternatives. The cognitive cost of a task depends on the background knowledge of designers, accessibility of pertinent information, and complexity of the task. Thus, although it is customary to think of solutions to design problems in terms of a hierarchical plan since hierarchical decomposition is a common strategy to cope with complex design situations, in practice, designers have been observed to perform tasks in an opportunistic order. Design environments can allow the benefits of both an opportunistic and a prescribed design process. They should allow, and where possible augment, human designers' abilities to choose the next design task to be performed. They can also provide information to lower the cost of following the prescribed process and avoid context switches that deviate from it. Without examining this theory, it might have been more compatible with other software research to provide a prescriptive process representation of the tasks to be completed as opposed to the less restrictive checklist.

Descriptions of how theories from comprehension and problem solving (e.g., [31]) and design visualization (e.g., [22]) may be found elsewhere [45] [46]. In general, the approach espoused here is that cognitive theories of design and problem solving can be applied to software systems to support the human cognitive aspects of end users, including designers as end users of software tools.

5. Supporting Observations About Software Use

Even with good field methods, the end users' needs for software fluctuate during the lifetime of a project [10]. Part of the fluctuation occurs because not all stakeholders communicate in the same terms or mean the same concepts by the same terms. A process of mutual education among all stakeholders increases the

(a) Expectation agents monitor system usage.



(c) Users may provide optional feedback.

(b) Agents may post messages.

Figure 4: Expectation-Driven Event Monitoring

chances that software will fit with its delivery environment [23]. Furthermore, once software is deployed in an environment, the environment, including the end users, adapt [51]. Thus, end users may have different needs for software after it has been in place for a time. Finally knowledge about requirements is inherently tacit [53] [42]. Human end users may be able to perform their work, but be unable verbally to recount "how" during a standard interview. Prototyping and scenario-based design is increasingly popular for addressing this problem [8] [41] [47]. Certainly on-going field studies can collect data about how systems are used and evolve. However, these are costly to perform in terms of time and placing observers on-site.

The Internet provides one avenue for collecting observations about the use of software in the field at minimal cost. Specifically, data about applications' use can be extracted from events reflecting end users' interaction with software. Extraction of usability data based on events is newly possible because of the distributed nature of most applications and current programming platforms for implementing interactive software. Moreover, the Internet allows packaged data to be delivered to those seeking to analyze it. There are a variety of methods and tools that provide the ability to extract usability data from events (see [26]).

One particular method for collecting usability information remotely is EDEM—Expectation-Driven Event Monitoring (see Figure 4) [25] [27]. EDEM allows software developers to articulate expectations of end users' behavior using agents. These expectation agents are downloaded when an application is initiated. They monitor events as a user interacts with the applications (Figure 4a). Expectation agents may take a variety of actions as they collect information. Specifically, they may collect information unobtrusively and report it back to a developer or deposit it into a data store. They may warn the end user of an application that he or she has done something unexpected (Figure 4b). In this case, the end user has the option of exploring the rationale behind the objection and responding (Figure 4c). When necessary, privacy can be maintained by using a proxy server to make the source of the data anonymous.

Thus, expectation agents are based on the concept that many usability problems occur because of developers making false assumptions about the end users or their use situation. The mismatch between assumptions and actual behavior may occur for any number of the reasons articulated above. In general, false assumptions reflect some kind of problem in communication. What EDEM supports is increased communication between developers and end users, both implicit through observation and explicit through optional comments. It supports this communication during the context of the breakdown. It supports directed observation about the use of software. It supports a direct link between developers and end users. In short it has some of the advantages of direct observation but at reduced cost. There are many advantages and disadvantages to this kind of automated collection of usability data. The reader is referred elsewhere for detailed discussion of these issues [26] [27].

In sum, observations in the field before design and good feedback on the system's perspective of design are only part of the needs in software design. Software requirements evolve after deployment. Software and its specification must co-evolve [17]. However, the method for gathering information after deployment must be consistent with the practice of software developers and limitations on resources.

6. Awareness and Review of Project Information using a Group Knowledge Depot

Reviews are a common part of any software development project. They are essential to reviewing fluctuating and misunderstood requirements, as noted above. They are also key to understanding the state of a software project: who is working on what problems, whether the project is close to completion, etc. Much of this information is on-line today because of the increased geographic distribution among software teams. However, even if a software

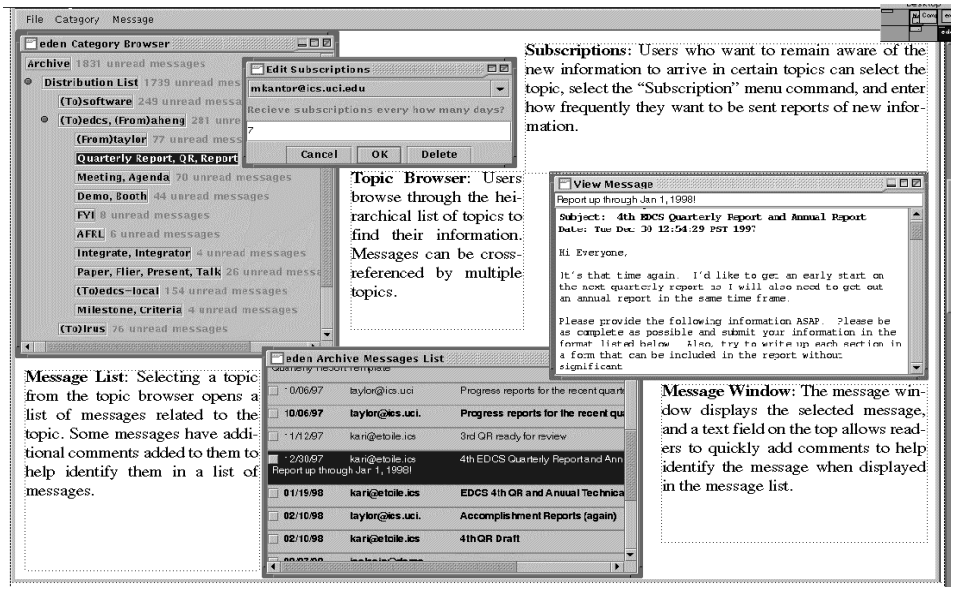


Figure 5: Annotated, End Users' View of the Knowledge Depot

development project is taking place in-house, many stakeholders are separated from one another either by organizational division or sufficiently removed that communication does not take place frequently enough to maintain an awareness of project activities.

To maintain greater mutual awareness of the activities among members of a group, project, or community, researchers have created a variety of tools such as software Portholes that allow end users to view camera views (e.g., [11] [12]) and other gauges of physical activity (e.g., [32] [33]) in their colleagues' offices. Others have experimented with awareness based on sound and audio (e.g., [2]).

Another approach is to monitor email and classify it according to categories. Figure 5 illustrates one such system called the Knowledge Depot [43] [30]. Versions of the Knowledge Depot have been experimented with by different researchers for several years (see [30] [34] [43]). In this approach, emails about a topic are collected and classified in a topic hierarchy. Objects are classified (and are retrieveable) according to every topic that applies.

This email-based approach has the advantage for software development that greater information content can be provided than other kinds of awareness. Stakeholders in a project can subscribe to categories of information. When email (or a project document) is associated with a category, a subscriber may receive a notification. Notifications can vary in frequency and detail. For instance, summaries of category activity with only message headers might be sent once a week. This frequency and detail may be sufficient for a manager involved in a project. A programmer working on one feature of some software might want to be aware as soon as more information is available about that feature. Information collected automatically by usability monitoring can be inserted directly into appropriate categories.

Thus, the Knowledge Depot illustrates one way that information from many sources pertinent to many stakeholders may be collected in one location and provide awareness of that information to those who need it. It provides organization of information through a category hierarchy; supports category operations for interaction including maintenance, interaction, query, and subscription; and maintains a persistent data store. It supports the needs of many end users. One can imagine that selective end users involved in a software development project would be given access to at least subscriptions to topics relating to evolving software requirements.

Although it is a goal to support the needs of all stakeholders in a software development project, the reality is that different stakeholders (including intended end users) have different needs for information and require different perspectives. The Knowledge Depot illustrates one way that the information needs of multiple stakeholders may be supported and in turn support the need for information review in an on-going software development project.

7. Conclusion

Although it is a stated goal of many software development projects that the end users' views should be represented and incorporated into the design, the field of knowledge about how to best achieve this goal is distributed across many disciplines. Moreover the best way to incorporate some of the methods and tools from other disciplines into software development is still an open area of research.

In the preceding sections, descriptions of some methods and tools were presented. All of these have been applied to the problem of developing software systems in a way that captures as many points of view as possible and yet make the captured information available to software developers in a way that is compatible with their practice.

The descriptions above indicate that the goal of software development for today's world is more complex than ever. It involves many stakeholders representing many points of view. It involves engineering for interactive and collaborative applications that must function in a cohesive fashion with the workplace environments in which they are deployed. The definition of "end users" can even be confusing. It can refer to the people who will interact with a software application that is deployed, but it can also refer to those who are otherwise affected by the deployment, such as colleagues of the end users, their managers, and customers who might be served by those applying the software. Moreover, software developers are also end users, namely of software tools. They have cognitive needs that if met can lead to better designs.

Many researchers have pioneered the way for this present understanding of software. To the degree possible in this conference paper, they have been acknowledged above. The work of theirs highlighted above will provide the reader with more inspiration about the diversity needed in developing software that fits end users' many views. Much more work remains. It is hoped that this present paper indicates some of the possibilities for augmenting software developers' abilities to be aware of and to incorporate the views of all of the end users, by employing software tools and methods based on current cognitive and social theories. Moreover, it is the intention that on-going work in this direction maintain a focus on tools and methods that are accessible and pragmatic.

8. ACKNOWLEDGMENTS

This paper has resulted from several years of effort by the author and his former and present graduate students: David M. Hilbert (EDEM), Jason E. Robbins (Argo), Michael Kantor (Knowledge Depot), Shilpa V. Shukla (Activity Theory), Cleidson R. B. De Souza (critics and event notification), and Santoshi D. Basaveswara (event notification). Parts of this effort were 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 and by the National Science Foundation (NSF) under grants 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.

9. REFERENCES

[1] Ackerman, M., Halverson, C. *Considering an Organization's Memory*, The 1998 ACM Conference on Computer Supported Cooperative Work (Seattle, WA), ACM Press, New York, NY, November 14-18 1998, pp.39-48.

[2] Ackerman, M., Starr, B., Hindus, D., Mainwaring, S. *Hanging on the Wire: a field study of an audio-only media space*, ACM Transactions on Computer-Human Interface, Vol. 4, No. 1, 1997, pp. 39-66.

[3] Atwood, M., Gray, W., John, B. *Project Ernestine: Analytic and Empirical Methods Applied to a Real-World CHI Problem*, in Rudisill, M., Lewis, C., Polson, P., McKay, T., Eds. Human-Computer Interface Design: Success Stories, Emerging Methods, and Real-World Context, Morgan Kaufmann Publishers, Inc., San Francisco, 1996, pp. 101-121.

[4] Barthelmess, P., Anderson, K. *A View of Software Development Environments Based on Activity Theory*, Computer-supported Cooperative Work, Special Issue on Activity Theory and the Practice of Design, forthcoming, 2002.

[5] Boehm, B., Bose, P. *A collaborative spiral software process model based on Theory W*, Third International Conference on the Software Process, (Reston, VA), IEEE Computer Society Press, Los Alamitos, CA, October 10-11, 1994, pp.59-68.

[6] Brooks, F. *No Silver Bullet: essence and accidents of software engineering*, Computer, Vol. 20, No. 4, April 1987, pp. 10-19.

[7] Card, S., Newell, A., Moran, T. *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, N.J., 1983.

[8] Carroll, J. *Scenario-based Design: envisioning work and technology in system development*, Wiley, New York, NY, 1995.

[9] Collins, P., Shukla, S., Redmiles, D. *Activity Theory and System Design: A View from the Trenches*, Computer-supported Cooperative Work, Special Issue on Activity Theory and the Practice of Design, forthcoming, 2002.

[10] Curtis, B., Krasner, H., Iscoe, N. *A Field Study of the Software Design Process for Large Systems*, Communications of the ACM, Vol. 31, No. 11, November 1988, pp. 1268-1287.

[11] Dourish, P., Bly, S. *Portholes: Supporting Awareness in a Distributed Work Group*, The ACM Conference on Human Factors in Computing Systems (CHI'92, Monterey, CA), May 3-7, 1992, 541-547.

[12] Dourish, P. and Bellotti, V. *Awareness and Coordination in Shared Workspaces*, The ACM Conference on Computer-Supported Cooperative Work (CSCW'92, Toronto, Ontario), November 1-4, 1992, pp. 107-114.

[13] Engelbart, D. *A Conceptual Framework for the Augmentation of Man's Intellect*, in Greif, I., Ed., Computer-Supported Cooperative Work: a book of readings, Morgan Kaufmann, San Mateo, CA, 1998, pp. 35-66, Ch. 2.

[14] Engeström, Y. Learning, *Working and Imagining: twelve studies in activity theory*, Orienta-Konsultit, Helsiniki, Finland, 1990.

[15] Fischer, G. *A Critic for LISP*, The Tenth International Joint Conference on Artificial Intelligence (Milan, Italy), Morgan Kaufmann Publishers, Los Altos, CA, August 1987, pp. 177-184.

[16] Fischer, G. *Domain-Oriented Design Environments,* Automated Software Engineering, 1994, pp. 177-203.

[17] Fischer, G., Henninger, S., Redmiles, D. *Cognitive Tools for Locating and Comprehending Software Objects for Reuse,* Thirteenth International Conference on Software Engineering (Austin, TX), IEEE Computer Society Press, ACM, IEEE, Los Alamitos, CA, May 1991, pp. 318-328.

[18] Fischer, G., Girgensohn, A., Nakakoji, K., Redmiles, D. *Supporting Software Designers with Integrated, Domain-Oriented Design Environments,* IEEE Transactions on Software Engineering, Special Issue on Knowledge Representation and Reasoning in Software Engineering, Vol. 18, No. 6, 1992, pp. 511-522.

[19] Fischer, G., Nakakoji, K., Ostwald, J., Stahl, G., Sumner, T. *Embedding Critics in Design Environments,* Knowledge Engineering Review, Vol. 8, No.4, December 1993, pp.285-307.

[20] Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M., *Viewpoints: a framework for integrating multiple perspectives in system development,* International Journal of Software Engineering and Knowledge Engineering, Vol. 2, No. 1, 1992, pp. 31-57.

[21] Gould, J., Boies, S., Lewis, C. *Designing for Usability: Key Principles and What Designers Think,* Communications of the ACM, Vol. 34, No. 1, 1991, pp. 75-85.

[22] Green, T., Petre, M. *Usability Analysis of Visual Programming Environments: a 'cognitive dimensions' framework,* Journal of Visual Languages and Computing, Vol. 7, 1996, pp. 131-174.

[23] Greenbaum, J., Kyng, M., Eds. *Design at Work: Cooperative Design of Computer Systems,* Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

[24] Guindon, R., Krasner, H., Curtis, B. *Breakdown and Processes During Early Activities of Software Design by Professionals,* in Olson, G., Sheppard, S., Soloway, E., Eds., Empirical Studies of Programmers: Second Workshop, Ablex Publishing Corporation, Lawrence Erlbaum Associates, Norwood, NJ, 1987, 65-82.

[25] Hilbert, D., Redmiles, D. *An Approach to Large-Scale Collection of Application Usage Data Over the Internet,* The Twentieth International Conference on Software Engineering (ICSE '98, Kyoto, Japan), IEEE Computer Society Press, April 19-25, 1998, pp. 136-145.

[26] Hilbert, D., Redmiles, D. *Extracting Usability Information from User Interface Events,* ACM Computing Surveys, Vol. 32, No. 4, December 2000, pp. 384-421.

[27] Hilbert, D., Redmiles, D. *Large-Scale Collection of Usage Data to Inform Design,* Eight IFIP TC 13 Conference on Human-Computer Interaction (INTERACT 2001, Tokyo, Japan), July 2001, pp. 569-576.

[28] Hutchins, E. *Cognition in the Wild,* MIT Press, Cambridge, MA, 1995.

[29] Jackson, D., Rinard, M. *The Future of Software Analysis,* in Finkelstein, A., Kramer, J., Eds., FOSE 00: The Future of Software Engineering (Limerick, Ireland), ACM Press, New York, NY, June 4-11, 2000.

[30] Kantor, M., Zimmermann, B., Redmiles, D. *From Group Memory to Project Awareness Through Use of the Knowledge Depot,* The 1997 California Software Symposium (Irvine, CA), UCI Irvine Research Unit in Software, Irvine, CA, November 7, 1997, pp. 19-26.

[31] Kintsch, W., Greeno, J.G. *Understanding and Solving Word Arithmetic Problems,* Psychological Review, Vol. 92, No. 1985, pp. 109-129.

[32] Lee, A., Girgensohn, A., Schlueter, K. *Sensing Activity in Video Images,* The ACM Conference on Human Factors in Computing Systems (CHI'97), Extended Abstracts (Atlanta, GA), March 1997, pp. 319-320.

[33] Lee, A., Girgensohn, A., Schlueter, K. *NYNEX Portholes: Initial User Reactions and Redesign Implications,* The International ACM SIGGROUP Conference on Supporting Group Work (Phoenix, AZ), November 1997, pp. 385-394.

[34] Lindstaedt, S., Schneider, K. *Bridging the Gap Between Face-to-Face Communication and Long-Term Collaboration,* The International ACM SIGGROUP Conference on Supporting Group Work (Phoenix, AZ), November 16-19, 1997, pp. 331-340.

[35] Luff, P., Hindmarsh, J. Heath, C. Eds. *Workplace Studies: recovering work practice and informing system design,* Cambridge University Press, Cambridge, UK, 2000.

[36] McDermott, J. R1: the formative years, AI Magazine, Vol. 2, No. 2, 1981, pp. 21-29.

[37] Nardi, B., Ed. *Context and Consciousness: activity theory and human-computer interaction,* MIT Press, Cambridge, MA, 1996.

[38] Nardi, B., Redmiles, D., Eds. *Journal of Computer-supported Cooperative Work, Special Issue on Activity Theory and the Practice of Design,* forthcoming, 2002.

[39] Norman, D., Draper, S. *User Centered System Design: new perspectives on human-computer interaction,* Lawrence Erlbaum Associates, Hillsdale, N.J., 1986.

[40] Nuseibeh, B., Easterbrook, S. *Requirements Engineering,* in Finkelstein, A., Kramer, J., Eds., FOSE 00: The Future of Software Engineering (Limerick, Ireland), ACM Press, New York, NY, June 4-11, 2000.

[41] Potts, C., Takahashi, K., Anton, *A. Inquiry-Based Requirements Analysis,* IEEE Software, Vol. 11, No. 2, March 1994, pp. 21-32.

[42] Polanyi, M. *Tacit Dimension,* Peter Smith Publishers, 1983.

[43] Redmiles, D. *Software Requirements for Supporting Collaboration through Categories,* Workshop on Classification Schemes in Cooperative Work, ACM Conference on Computer Supported Cooperative Work (CSCW 2000—Philadelphia, PA), December 2000, published online at http://bscw.gmd.de/bscw/bscw.cgi.

[44] Rich, C., Waters, R. *The Programmer's Apprentice,* ACM Press, New York, NY, 1990.

[45] Robbins, J., Hilbert, D., Redmiles, D. *Extending Design Environments to Software Architecture Design,* Automated Software Engineering, Vol. 5, No. 3, July 1998, pp. 261-290

[46] Robbins, J., Redmiles, D. *Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML,* Information and Software Technology, Vol. 42, No.2, January 2000, pp.79-89.

[47] Rosson, M., Carroll, J. *Usability Engineering: scenario-based development of human-computer interaction,* Morgan Kaufmann, San Fancisco, CA, 2002.

[48] Rudisill, M., Lewis, C., Polson, P., McKay, T., Eds. *Human-Computer Interface Design: success stories, emerging methods, and real-world context,* Morgan Kaufmann Publishers, Inc., San Francisco, 1996.

[49] Schön, D. *The Reflective Practitioner: how professionals think in action,* Basic Books, New York, NY, 1983.

[50] Schön, D. *Designing as Reflective Conversation with the Materials of a Design Situation,* Knowledge-Based Systems, Vol. 5, No. 1, 1992, pp. 3-14.

[51] Simon, H. *The Sciences of the Artificial,* The MIT Press, Cambridge, MA, 1981.

[52] Soloway, E., Ehrlich, K. *Empirical Studies of Programming Knowledge,* IEEE Transactions on Software Engineering, Vol. SE-10, No. 5, 1984, pp. 595-609.

[53] Suchman, L. *Plans and Situated Actions,* Cambridge University Press, Cambridge, UK, 1987

[54] Suchman, L. *Making a case: 'knowledge' and 'routine' work in document production,* in Luff, P., Hindmarsh, J. Heath, C. Eds. Workplace Studies: Recovering Work Practice and Informing System Design, Cambridge University Press, Cambridge, UK, 2000, pp. 29-45.

[55] Visser, W. *More or Less Following a Plan During Design: opportunistic deviations in specification,* International Journal of Man-Machine Studies, Vol. 33, No. 1990, pp. 247-278.