Report on the Software Environments Technical Research Review

Jeffrey J. Blevins, E. James Whitehead, Jr., and Harry E. Yessayan

organized by: Leon J. Osterweil and Richard N. Taylor

Irvine Research Unit in Software
University of California, Irvine 92717-3425
{jblevins, ejw, yessayan}@ics.uci.edu

May 25-27, 1993

1 Introduction

In May, 1993, industry and academic representatives from IRUS member institutions met to conduct a unique technical research review on the topic of software environments. This group was charged with the goal of assessing the state of the art and state of the practice in software development environments with an eye towards formulating a research and technology transfer agenda for the deployment of effective software environments. The participants started with the premise that few environments are in actual use, and that there is little agreement on which principles should guide the development of a successful environment. Fundamental to this meeting was the reservoir of knowledge held by all participants, and the desire to see this knowledge consolidated and applied to take software environments from the state of the practice, namely infrequently used point solutions, to the desired goal of invaluable, effective, integrated tool sets.

This document describes the issues encountered during the technical research review, the observations of the participants, and some direction for the future.

2 History

The approach to building and using software environments has evolved from the early efforts of function-oriented environments, to an object-store approach, to the current approach emphasizing process-centered environments. By looking at a history of environments with respect to the evolution of the approach, we can begin to better understand where the current technology stands.

Tool-centered or function-oriented environments. The earliest work in software environments emerged from the world of programming-in-the-small. The focus was on "programming" rather than on software development. People built programming environments which essentially consisted of collections of tools, such as text editors, syntax-directed editors, and debuggers, to support programmers. These environments, such as Interlisp [TM81], Unix [RT78], CPS [TR81], and Mentor [DGLM85], are called tool-centered because of the near-exclusive focus on the tools provided to and used by the coders.

Object-store environments. As the development process grew more complex, much of the emphasis was shifted to managing the objects in the software environment and the relationship between those objects. Many different kinds of artifacts were recognized as part of the development activity: source code and object code files, requirements specifications, design documents, detailed design specifications, test plans, change request, etc. The number of artifacts in the environment increased along with the complexity of their structure. With the increased complexity, a functional orientation was inadequate. Systems such as Odin[CO90], DSEE[LRPC84], PCTE[BGMT88], and PMDB[Pen90] are examples of environments supporting an object-store approach.

Process-centered environments. The focus on objects in the environment proved to be not only useful but critical to the development process; however, it failed to address many of the problems that still existed in the software development process. Environment builders and researchers began focusing on the process itself by modeling, specifying, and executing the process using process formalisms and languages. Understanding and controlling the software development process has the potential for leading to successful management of its complexity and the quality of its products. Process-centered environments adopted a flexible, extensible, and programmable approach to the software development process. Most importantly, the roles of personnel were delineated and explicitly supported. Recent environment projects taking the process-centered approach include ISTAR [Dow87] , ESF [ESF89], IPSE 2.5 [Sno89], and Arcadia [Kad92].

3 Usage

After more than a decade of development, how well are CASE products being utilized by software developers? Well-informed sources report that only 20% of popular CASE product licenses are actually in use. It is believed those products are only being utilized in a very minimal manner, or for evaluation purposes. Furthermore, the only tools being used in CASE environments are ones that are pre-integrated by CASE vendors. Developers are not integrating the other tools they use into the environments. The only exception to this observation is document preparation systems, which a few developers are integrating with upper CASE design tools. For the most part, developers are treating CASE environments as they have treated existing CASE tools: as point solutions. This approach restricts the impact that the tools and environments can have on the development effort.

What is keeping software developers from using the existing CASE products?

- Are CASE products failing to address software developers' true concerns, and therefore providing no value to the development effort?
- Are the CASE products not meeting the right needs?
- Do the developers understand their needs and convey them to the CASE vendors?
- Are the developers waiting for the perfect CASE environment with all the answers?
- Are the developers waiting to see what becomes standard among their fellow developers?
- Is it all simply a matter of economics; do they not have enough resources to adopt the CASE environments?

In our attempts to answer these questions, we discovered a complex web of issues that contribute to the failure of CASE tools and environments to impact software development.

4 Issues

During the course of the research review, several major issues were identified and discussed as to why environments thus far have fallen short of expectations. These issues arose in varying and overlapping order, but are presented here separately. It is apparent that many of these issues are interrelated: certain aspects of some issues are relevant for other issues. However, in this section we focus on identifying and discussing the isssues independently.

4.1 Economics

Economic viability seemed to be a universally recognized inhibitor to the adoption of software environment technology. Many participants felt that current technology has not been adopted extensively because potential users of these systems, and especially the management deciding on the purchase of such systems, are not convinced that the benefits will outweigh the costs. The bottom line for many potential users is a motivation of profit for the company. Some of the systems currently available support leading edge technology but are not necessarily designed with economic considerations in mind. Industrial users of CASE and software environments want something that will reduce their costs and make their operations more efficient. Many advanced CASE systems cost upwards of $10,000 per copy, making them prohibitively expensive. Companies are not willing to spend such money on a system that might only provide marginal improvement in efficiency, or worse, might slow things down because of a high learning curve or complexity of use. Moreover, people related a fear of possibly getting stuck with an albatross: spending lots of money and resources to get a system in place which might quickly become outdated or fail to provide adequate support. Participants generally agreed that CASE and software environments would be successful only if economic considerations were taken into account. There is a perception that the adoption of some systems actually makes things more complex and costly.

Suggestions for recitifying this situation included reducing the cost of software environment technology in terms of acquisition costs, training costs, customization effort, and maintenance costs. Also, ensuring that any perceived benefits could be easily realized, making clear the potential benefits, and ensuring the systems live up to their claimed benefits would make enviroment technology more palatable. One often overlooked consideration is the amortization of tool costs over the life of several projects. Accounting departments more often than not fail to take such an approach, perhaps from lack of understanding that tools and environments can be used beyond the life of the project for which they are purchased. Failure to take such considerations into account makes the cost of tools and environments appear much higher than they really are.

4.2 Convincing Management

Several participants felt that one of the major hurdles in adopting software environment technology is successfully convincing management of the potential benefits. Convincing management should involve communicating the needs of the people doing the development; sometimes management doesn't fully understand the development process or understand why a certain technology would be of benefit. Approaches to solving this problem could include developing channels of communication between users and management and between vendors and management, educating managers about the benefits, and providing managers with cost/benefit analyses. One of the problems is that benefits are often difficult to quantify. A participant commented, with the group generally agreeing, "there is a total disconnect between technologists and the people who own the checkbooks."

4.3 Identifying Relevance

While the participants were generally excited by the software environments technology issuing from the research community, they all shared a concern as to its relevance. Given their experience that most CASE tools end up as shelfware, and that most academics consider commercial CASE technology to be behind the state of the art, it is natural to wonder about the pertinence of a more advanced version of that which is not currently being used. One insightful question by a participant summed this up: "Why should we care about software environment research prototypes?"

The ready answer to this question is the promise that software environments have for improving the *quality* of software produced with the environment, with some *productivity* increases expected as a side-effect as error rates decrease. But this begs a follow up question: what specifically is needed to achieve this software environment utopia? What tools and methodologies and frameworks are required? On this issue there was no consensus, excepting the overwhelming feeling that existing software environment frameworks, such as NIST/ECMA and Eureka/Core, are not directly relevant, offering a "second or third order effect that doesn't directly affect real products." There was also general agreement that more advanced tools supporting a defined software process are needed.

All participants agreed academia should provide more robust, more easily transferable technologies that are solutions to non-toy problems. Participants uniformly look to academia to provide intellectual leadership in the effort to create more advanced tools, and herein lies the heart of the disconnect between industry and academia. While industry has been waiting for the next great environment to spring full grown out of a research laboratory, academia has been resource-constrained and unable to produce such a commercial product. Academia desires instead to have industry assume this burden. This disconnect leads to the feeling that academic research is irrelevant. Clearly, judicious technology transfer is needed.

4.4 Evaluation of Current Technology

A common experience among participants was the purchase of expensive CASE products that did not meet expectations. Frustrating to many was the perception that most products were rejected or not used for similar reasons by many companies. If only a common clearinghouse of CASE information were available, many expensive mistakes could have avoided duplication. Thus there is great interest in a "Consumer Reports" style evaluation of existing CASE products. While the exact details of funding, and who would perform such evaluations were not determined, there was agreement that a research university offered the ideal setting for performing such an evaluation.

This raises the question of why current review articles do not meet the needs of CASE evaluators. In fact, one participant was involved in writing such an article, and found it very difficult to get information from vendors. As well, evaluating the various products and writing the article itself was very time consuming compared to the offered remuneration. Neither of these are conducive to the production of high-quality CASE product evaluations.

An experience of many was that the sales literature for CASE products often obscured or did not address many pertinent technical topics. Often, a methodology used in evaluating the purchase of a CASE product involved solicitation of sales literature followed by a visit to the CASE vendor to see the product in use. One sentiment was a desire to share the experiences from these vendor visits, due to their cost. A common desire was to see academia take a leadership role in coordinating this collection and dissemination of shared information. This was viewed as natural due to academia's neutral position between competing companies.

4.5 Intended Audience

One old and unresolved issue in environments is who should be the intended audience of a software development environment. Put a different way, should an environment help those educated in computer science become better, or should the environment try to make software development accessible to people with lower skill levels? Or should software environments be geared towards gurus or those who are naturally attracted to experimenting with a new technology? While no clear consensus developed on this issue, the most prevalent view was that environments should help to make the best programmers better. This reflects the general view that most environment technology is perceived as being oriented for the experts, requiring a great deal of commitment to surpass a high learning curve and requiring a high level of sophistication to use successfully. However, the best computer technologies are always enabling, and an expected result of environments research would be greater accessibility to a wide audience of programmers.

Software environments also have as their audience the management of software projects. Unfortunately, the common experience is that most management needs to become more educated in software environments. One participant commented, speaking primarily of management, "there are hundreds of people at my company who need a CASE briefing." As with any new, sophisticated software artifact, training and education are essential parts of acceptance and integration of CASE tools into daily practise.

4.6 Process/Project Level Issues

CASE environments strive to facilitate tool integration to enable them to have value above the sum of their component parts. Very few CASE vendors have convinced software developers that this can be done. The majority of software developers view their process as a series of tasks, with different tools required for each task. This simplified view of their process ignores the value of the relationship between the different tasks that make up their process, and therefore makes it difficult for them to see the value of having relationships between the tools they use. Additionally, by only viewing the development process as a series of tasks, developers are ignoring the critical problems that only exist at the overall, organizational level.

This incomplete understanding of organizational and project process contributes to the lack of success of CASE environment integration. Many companies are only now beginning to achieve a macro-level understanding of their development process. CASE environments are integrated on a finer level of detail. For the developers to understand the value of tool integration, they might need to specify their process at this finer level. Once they understand their organizational integration, they might have a better chance of assessing CASE tool integration. One possible win for software developers will come when they have defined their processes and then found CASE vendors with low cost solutions that integrate into their processes.

4.7 Communication

Having high quality means of communication is vital to project success. Two specific channels of communication are particularly important in software development. The first critical communication link exists between project managers and the rest of the development team. The second critical link exists between individuals, or subteams, working on a project. The importance of such communications has been ignored by CASE vendors and needs to be realized and capitalized upon to help coordinate a project's successful completion.

These valuable communication channels consist of both formal and informal means of conveying project information. Formal means, such as electronic mail, video conferencing, and meeting minutes, are fairly easy to maintain. It is the informal means, such as casual conversations by the water cooler, over lunch, or in the elevator, that are harder to maintain. For these informal means to persist, someone must exert extra effort to document information resulting from them. It may be necessary to develop methods that will enable the burden of this task to be removed, or at least lightened, from human dependence.

Integrating means and methods of communication into CASE environments requires understanding how they contribute or relate to the different activities and artifacts of the software development process. Much like the process/project management component of the CASE environment, the communication component will not be targeted at a specific task within the project, but at the entire project, or perhaps even beyond a single project. Complete utilization of an environment that provided these capabilities would require the software developers to abandon their practice of utilizing tools at only specific points of a project and begin to understand the totality of their development interactions.

5 Conclusion

In spite of recent advances in software development technology, industrial practitioners have been slow to adopt and successfully utilize CASE and software environments. Several specific problem areas were identified and discussed during the technical research review and have been presented in this document. We learned from this research review that better interaction between academic researchers and industrial practitioners is critical to successful deployment of new technology. Participants felt that this technical research review was valuable in opening an informal dialogue between academia and industry and in facilitating the sharing of ideas and problems among industry practitioners concerned with CASE and software environment technology.

Lessons learned from this research review suggest establishing regular meetings to provide an organized forum for the CASE community to discuss problems and share ideas. Such meetings are a valuable way for researchers in academia to learn about the needs of those people in industry who stand the most to gain from what comes out of the research community. In addition, such meetings are a valuable means for people in industry to interact with one another.

A Specific Needs and Requests

While a detailed wish list of features desired in future CASE tools could easily fill many volumes, some specific CASE tool requests discussed by participants are given below.

A.1 General Problem Tracking

One tool in great demand by participants was a system trouble report tracking program, or abstracted, a general problem tracking tool that could track problems in all phases of development. While participants reported instances of these tools being developed in-house, many voiced a desire for a feature-rich commercially available general problem tracking tool. One desired feature of these tools is tight integration with the rest of the software environment. Further investigation into this type of tool, perhaps with process support, would be beneficial.

A.2 Tool Support for Reuse and Reengineering

Another holy grail of software engineering discussed was that of a tool to support software reuse and reengineering. While no insights into how this might be accomplished were forthcoming, all were confident they could use such a tool if developed. Clearly this is an area where academia could direct some research and assume a leadership role.

A.3 Project Planning Simulation

Participants agreed the capability to analyze and make projections based on a defined software development process is important. For example, a process simulation tool would allow managers to estimate the amount of time and resources need for a particualr process. It might also identify process bottlenecks or other potential problems. The tool should allow the manager to perform ‘what-if’ scenarios on their project. The tool could be further enhanced by integration with metrics and evaluation systems.

A.4 Real-time/Embedded Debugging

Finally, one area noted as deficient by participants was the state of the art in real-time debugging technology. A wish list of features desired from a real-time debugger was compiled, as follows:

- Debug multiple processes simultaneously
- Handle timing information
- Reason about different hardware platforms (MIPS, Sparc, other)
- Handle cache / main memory interactions
- Handle networking
- Handle backplanes and busses
- Handle coprocessors
- Model custom hardware

Lamentably, participants noted that no commercial product implemented even a substantial subset of these features.

B Participants

The following individuals participated in the Software Environments Technical Research Review on May 25-27, 1993.

Paul Ashton, Beckman Instruments
Jeffrey Blevins, U.C. Irvine
Terrance Domae, Northrop Corporation
Cameron Florus, Rockwell International
Kari Forester, U.C. Irvine
Denny Frayne, McDonnell Douglas Corporation
Mark Gerhardt, TRW
Richard McWhorter, McDonnell Douglas Corporation
Larry Miller, The Aerospace Corporation
Leon J. Osterweil, U.C. Irvine
Christine Shu, TRW
Rick Sidwell, Rockwell International
John Strong, Northrop Corporation
Richard Taylor, U.C. Irvine
Jock Rader, Hughes Aircraft Company
James Whitehead, U.C. Irvine
Thomas Winfield, Hughes Aircraft Company
Harry Yessayan, U.C. Irvine
Patrick Young, U.C. Irvine

References

[BGMT88] Gerard Boudier, Ferdinando Gallo, Regis Minot, and Ian Thomas. An overview of pcte and pcte+. *SIGSOFT Software Engineering Notes*, 13(5), November 1988.

[CO90] Geoffrey M. Clemm and Leon J. Osterweil. A mechanism for environment integration. *ACM Transactions on Programming Languages and Systems*, 12(1):1–25, January 1990.

[DGLM85] V. Donzeau-Gouge, B. Lang, and B. Me'le'se. A tool for Ada program manipulations: Mentor-Ada. In John G. P. Barnes and Gerald A. Fisher, Jr., editors, *Ada in Use: Proceedings of the Ada International Conference*, pages 297–308, Paris, May 1985. Association for Computing Machinery and Ada-Europe, Cambridge University Press. Appeared as *Ada Letters V*(2), September/October 1985.

[Dow87] Mark Dowson. Integated project support with ISTAR. *IEEE Software*, 4(6):6–15, November 1987.

[ESF89] Eureka Software Factory Consortium ESF. Technical reference guide, June 1989.

[Kad92] R. Kadia. Issues encountered in building a flexible software development environment: Lessons learned from the Arcadia project. In *Proceedings of ACM SIGSOFT '92: Fifth Symposium on Software Development Environments*, Tyson's Corner, Virginia, December 1992.

[LRPC84] David B. Leblang and Jr. Robert P. Chase. Computer-aided software engineering in a distributed workstation environment. *ACM SIGPLAN Notices*, 19(5):104–112, May

1984. (Proceedings of the First ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments).

[Pen90] Maria H. Penedo. Acquiring experiences with executable process models. In Dewayne E. Perry, editor, *Proceedings of the 5th International Software Process Workshop*, pages 112–115, 1990.

[RT78] D. M. Ritchie and K. Thompson. The UNIX time-sharing system. *The Bell System Technical Journal*, 57(6):1905–30, July-August 1978.

[Sno89] R. A. Snowdon. An introduction to the ipse 2.5 project. In Fred Long, editor, *Software Engineering Environments: Proceedings of the International Workshop on Environments*, volume 467 of *Lecture Notes on Computer Science*, pages 13–24. Springer-Verlag, 1989.

[TM81] W. Teitelman and L. Masinter. The Interlisp programming environment. *IEEE Computer*, 14(4):25–33, April 1981.

[TR81] Tim Teitelbaum and Thomas Reps. The Cornell Program Synthesizer: A syntax directed programming environment. *Communications of the ACM*, 24(9):563–573, September 1981.