Lifecycle Environments:

A Retrospective View of the Contributions of Leon J. Osterweil

Lori A. Clarke

Department of Computer Science, University of Massachusetts, Amherst, MA USA 01003

clarke@cs.umass.edu

Abstract Throughout his career, Leon Osterweil has made significant contributions that have impacted the research and state-of-the-practice on development environments. Initially his focus was on programming environments, mostly addressing issues needed to support his work in program analysis. Later his focus expanded to software lifecycle issues, such as flexible component interaction models, efficient system regeneration, and the use of process definitions as the major coordination mechanism to orchestrate the interactions among collections of tools, hardware devices, and human agents. His current research continues to address environment issues, but now the emphasis is on supporting continuous process improvement by providing process languages, execution, simulation, and an assortment of analysis tools for evaluating the effectiveness, safety, and vulnerabilities of processes for a range of domains, from healthcare, to digital government, to scientific workflow.

1. From Program Analysis to Programming Environments

The early work on program analysis undertaken by Osterweil and Fosdick [21,37], Ryder [47], Balzer [5] and others initiated a new research direction, where analysis was used not only to assist with compilation but also to help find problems in the code. The early analysis work was primarily concerned with supporting FORTRAN, and Osterweil and Fosdick were working with the Numerical Analysis Group (NAG) in Oxford, England, which was trying to develop efficient and accurate numerical software packages [35] in close collaboration with Argonne National Labs, a major user of such libraries. As discussed in Part I of this book, program analysis has continued to grow as an important research area, as software systems have grown in size and complexity while also becoming a driving force of much of our societal infrastructure.

Early work in program analysis, however, soon confronted inherent and difficult problems in implementing these approaches, first as isolated tools and even more so as collections of tools. These problems laid the foundation for a thread of work, undertaken by Osterweil and others, addressing the need for *environments* of tools that support the full lifecycle of software activities. The early work on Software Development Environments (SDEs), such as Interlisp [60] and Mesa [54], were extremely innovative but were focused on a single, central programming language. The Gandalf Project [23] and the Cornell Program Synthesizer Project [44,59] built upon and generalized these approaches by creating meta-programming environments that could be instantiated for different programming languages. For the most part, these environments were tightly integrated around a central repository and one focused goal, the development and execution of a program. The Toolpack project [33] took a somewhat broader view of environments and recognized that collections of tools would be needed to support the various software engineering activities. It argued that these tools should not be monolithic, but instead they should be decomposed into tool fragments that could be called upon in different ways to achieve support for the many varied activities associated with software development. In many regards, this was one of the first arguments for component-based software engineering, made at a time before the infrastructure was available to easily define the components or flexibly glue them together.

In addition to recognizing the importance of component-based development, the Toolpack project was grappling with how to deal with software evolution. At that time, the FORTRAN systems being developed were considered large and recompilation and reanalysis were expensive. If one piece of the system changed, then did all of the tools in the environment have to be reapplied? Building on the success of Make [20], which automatically assembles executables from various source files for Unix, Clemm and Osterweil developed Odin [14]. Odin would also automatically assemble the executables from various source files, but it would first analyze what had changed in the system and then, based on those results, determine which tool fragments needed to be reapplied and automatically initiate their execution. Moreover, Odin tried to determine when it should eagerly recompute and save intermediate results versus lazily delay and only recompute when a current version was needed.

2. Integrated Software Development Environments

Toolpack was one of the first attempts to recognize that software development was a complicated set of processes and would need to be supported by a collection of tools that interacted with each other. The US Department of Defense (DoD) was just starting to recognize the importance of software systems to their mission, and Defense Advanced Research Projects Agency (DARPA), which previously had primarily focused on networking and artificial intelligence, initiated research programs to support the development of large, complex systems. The DARPA-funded Arcadia Project was novel in that from the get-go it involved collaborations among researchers from different institutions. The academic ties to Osterweil were quite strong however. The major academic departments were the University of California, Irvine, the University of Colorado, Boulder, and the University of Massachusetts, Amherst, along with TRW and Incremental Systems, Inc. It is notable that the university efforts were all led by former students of Osterweil or their descendants.

Early SDEs, now called Integrated Development Environments (IDEs), focused on how to have a collection of tools work together to support the software development process. Based on the success of the programming environments for LISP and Mesa, interest developed for providing an environment for Ada, an emerging language at the time. Early Ada documents [6] outlined an agenda that went beyond just programming language support and included support for the full software lifecycle. Most of these efforts, however, assumed that these environments would be tightly integrated, in that there would be a single repository and a single user interface. Thus, any new tools would have to be developed with this common architectural view, which would no doubt limit extensibility. The Arcadia Project [26,57,58] broke from this view and argued for alternative integration models. These included using loose interaction models, object management, tool composition, and process models.

In their work on interaction models, Maybee, Heimbigner and Osterweil developed the Q system [28]. Q, like the Field system being developed about the same time by Reiss [42,43], supported loose interaction among distributed components and provided much more flexibility than commonly used RPC or message passing models. With respect to data interoperability, Q built upon the Module Interconnect Language [39,40] and IDL [27,52] work that was going on at that time to support data interoperability across languages. It is interesting to note that Q was the first open source and publicly available implementation of the CORBA 2.0 standard [32]. Subsequently, CORBA and other middleware systems built upon and extended many of the ideas that initially appeared in Q. A similar approach to loose interaction was also incorporated into the Chiron user interface system [66] that is now the standard architectural model for user interfaces.

The work on object management was an attempt to circumvent the restricted relational data base view of objects that assumed that there would be, at least conceptually, a single repository and associated data schema. APPL/A [53], PGraphite [56,62], Triton [24] and then Pleiades [55], Arcadia object management prototypes, included capabilities that allowed abstract data types to be defined, manipulated, and made persistent. The Arcadia object management work, as well as other efforts in this area (e.g., [3,31]), led to interesting interactions between the database community and the software engineering community and was the precursor of work on object-oriented data bases (e.g., [1,2]) and the impetus to incorporate persistence into programming languages [22].

Another contribution that arose from the Arcadia project was the importance of providing clean interfaces to various language-independent, intermediate results that arose from front-end compilation and analyzes. For example, language-independent interfaces to commonly used objects such as abstract-syntax trees, control flow graphs, dependency graphs, etc., facilitated the application of further analyses, one of the focal areas of the Arcadia Project [13,45,46]. This tool composition approach that was advocated in the Arcadia project has been subsequently widely adopted in environments such as Eclipse [18] and Visual Studio [61].

The research on interaction models, object management, and tool composition was, in some regards, focused on software architectural models, an important thread of much of the Arcadia project. This emphasis, directly or indirectly, led to some of the earliest work on software architecture, such as the PIC model for describing access control among components [63], the C2 interaction model [29], architectural classification work [30], and one of the earliest papers to introduce the concept of software architecture and associated concerns [38]. All of the Arcadia researchers engaged in long, and often heated, arguments about these topics and all benefited from these exchanges. The Taylor paper in Chapter 9 further elaborates on many of these issues and the ensuing research that built upon these early insights.

3. Process-driven Environments

One of the major insights that arose from the Arcadia project was Osterweil's realization that Make, Odin, and all of the existing scripting notations were inadequate to capture the complex interactions that were needed to describe how agents—that is, software components, hardware components, or human users—were to interrelate and interact in an IDE. Osterweil postulated that nothing short of a programming language would suffice in his seminal paper [34], "Software Processes are Software Too" (reprinted in Chapter 17). In this paper, Osterweil argued that it was necessary to accurately represent all the desired interactions among agents required by all of the development phases (e.g., requirements, design, etc.) in order to support the careful planning required to develop a software system. Moreover, he argued that the many analysis tools in the Arcadia environment, as well as the infrastructure components, such as the middleware and object management components, had to be orchestrated by process definitions defined in a rich, process language with well-defined semantics so that it, too, could be the subject of analysis.

This work was the harbinger of a rich body of work on software processes, described in more detail in Part III. Osterweil, however, soon viewed this work on process definition as going far beyond software development, which he now viewed as just one domain of interest, albeit an important one. He saw processes everywhere and soon came to realize that having an articulate process language provided an important basis for developing an environment to support systematic process improvement in many different domains.

4. Environments for Continuous Process Improvements

Osterweil was strongly influenced by the work of Deming [16] and Shewhart [48] on the study of process improvement. He realized that the process language that he had developed, Little-JIL [7], could be used to capture complex processes in a number of domains. Building upon his earlier work on SDEs and his view that processes are software too, he argued that process definitions need to be as carefully developed and analyzed as any other software system. This led to work with Clarke and Avrunin on developing a Process Improvement Environment [4] that included a visual editor for the language, plus a set of analysis capabilities. Analysis techniques that were originally developed to capture requirements of software systems [15,51] and to verify these requirements [17] were enhanced to address the complexities of process definitions. In some cases, the strong control-oriented view of process definitions made them even more amenable to this type of analysis than more data-oriented software systems.

To evaluate the hypothesis that a process improvement environment could benefit a wide variety of domains, case studies were undertaken in the areas of healthcare [9,10,25], on-line dispute resolution [11], elections [49], scientific workflow [19,36] and other areas. Each domain illustrated the benefits of this approach and provided insights about possible enhancements to the process improvement environment itself. For example, the work on healthcare resulted in the development of hazard analysis techniques to detect vulnerabilities, such as single points of failure [8], and discrete event simulation capabilities [41] to evaluate the comparative effectiveness of alternative processes or resource assignments. Some of the issues that have arisen during this work are reminiscent of early research threads that emerged in the early SDE work. For example, maintaining coherence between process performers and executing processes extends the early work on GUI design and event based notification with more extensive mediation mechanisms [50] that now need to be extended even further to support on-line process-guidance. Modeling complex processes, such as emergency room patient flow, often requires not only object management, but also resource management so that contested items can be effectively allocated and utilized. This has led to research on defining and allocating very diverse types of resources, such as those found in challenging real-world domains [64,65].

Because the Little-JIL language was specifically designed to support the flexibility that human agents like to retain, this process improvement approach is seen as particularly applicable to *human-intensive systems*, that is systems where human decisions and participation are an integral part of a complex process [12]. Such human-intensive systems arise in a range of domains from healthcare, to emergency response, to command and control, and will probably continue to grow in importance as devices, software systems, and human ingenuity are brought together to solve complex problems. Cugola et al. describe their recent work on applying process programming to the human-intensive domain of service-oriented computing in Chapter 10. Osterweil's current work is focusing on environments for modeling, evaluating, and executing such systems, going beyond the application of static analysis techniques to detect errors and vulnerabilities to also include online process monitoring and guidance as well as process improvements based on post-execution assessment and probabilistic analysis.

5. References

1. Andrews T, Harris C Combining Language and Database Advances in an Object-Oriented Development Environment. In: Object-Oriented Programming Systems Languages and Applications, Orlando, FL, October 1987. pp 430-440
2. Atkinson M, Bancilhon F, DeWitt D, Dittch K, Maier D, Zdonik S The Object-Oriented Database System Manifesto. In: First International Conference on Deductive and Object-Oriented Databases, 1989. pp 166-178
3. Atkinson MP, Bailey PJ, Chisholm KJ, Cockshott WP, Morrison R (1983) An Approach to Persistent Programming. The Computer Journal 26 (4):360-365
4. Avrunin GS, Clarke LA, Osterweil LJ, Christov SC, Chen B, Henneman EA, Henneman PL, L. C, Mertens W Experience Modeling and Analyzing Medical Processes: UMass/Baystate Medical Safety Project Overview. In: First International Health Informatics Symposium, Arlington, VA, November 11-12 2010. ACM, pp 316-325
5. Balzer RM Exdams -- Extendable Debugging and Monitoring System. In: 1919 Spring Joint Computer Conference, 1969. AFIPS Press, pp 567-580
6. Buxton J (1980) Requirements for Ada Programming Support Environments. Department of Defense,
7. Cass AG, Lerner BS, McCall EK, Osterweil LJ, Sutton Jr. SM, Wise A Little-JIL/Juliette: A Process Definition Language and Interpreter, ,demonstration paper. In: 22nd International Conference on Software Engineering, Limerick, Ireland, June 4-11 2000. pp 754-758
8. Chen B, Avrunin GS, Clarke LA, Osterweil LJ Automatic Fault Tree Derivation from Little-JIL Process Definitions. In: Software Process Workshop and Process Simulation Workshop, Shanghai, China, January 2006. Springer-Verlag, pp 150-158
9. Chen B, Avrunin GS, Henneman EA, Clarke LA, Osterweil LJ, Henneman PL Analyzing Medical Processes. In: Thirtieth International Conference on Software Engineering, Leipzig, Germany, May 2008. pp 623-632
10. Clarke LA, Avrunin GS, Osterweil LJ Using Software Engineering Technology to Improve the Quality of Medical Processes, Invited Keynote. In: Thirtieth International Conference on Software Engineering, Leipzig, Germany, ACM 2008. pp 889-898
11. Clarke LA, Gaitenby A, Gyllstrom D, Katsh E, Marzilli M, Osterweil LJ, Sondheimer NK, Wing L, Wise A, Rainey D A Process-Driven Tool to Support Online Dispute Resolution. In: 2006 International Conference on Digital Government Research, San Diego, CA, 2006. ACM, pp 356-357
12. Clarke LA, Osterweil LJ, Avrunin GS Supporting Human-Intensive Systems. In: FSE/SDP Workshop on the Future of Software Engineering Research, Santa Fe, NM, November 7-8 2010. ACM, pp 87-92
13. Clarke LA, Richardson DJ, Zeil SJ TEAM: A Support Environment for Testing Evaluation and Analysis. In: SIGSOFT '88: Third Symposium on Software Development Environments, 1988. ACM, pp 153-162

14. Clemm GM, Osterweil LJ (1990) A Mechanism for Environment Integration. ACM Transactions on Programming Languages and Systems 12 (1):1-25
15. Cobleigh RL, Avrunin GS, Clarke LA User Guidance for Creating Precise and Accessible Property Specifications. In: 14th SIGSOFT International Symposium on Foundations of Software Engineering, Portland, OR, November 2006. ACM, pp 208-218
16. Deming WE (1982) Out of the Crisis. MIT Press, Cambridge
17. Dwyer MB, Clarke LA, Cobleigh JM, Naumovich G (2004) Flow Analysis for Verifying Properties of Concurrent Software Systems. ACM Transactions on Software Engineering and Methodology 13 (4):359-430
18. Eclipse-an Open Development Platform. (2007). http://www.eclipse.org/.
19. Ellison AM, Osterweil LJ, Hadley JL, Wise A, Boose E, Clarke LA, Foster D, Hanson A, Jensen D, Kuzeja P, Riseman E, Schultz H (2006) Analytic Webs Support the Synthesis of Ecological Data Sets. Ecology 87 (6):1345-1358
20. Feldman SI (1979) MAKE- A Program for Maintaining Computer Programs. Software - Practice and Experience 9 (4):255-265
21. Fosdick LD, Osterweil LJ (1976) Data Flow Analysis in Software Reliability. ACM Computing Surveys 8 (3):305-330
22. Gosling J, Joy B, Steele GL (1996) The Java Language Specification. Addison-Wesley, Boston
23. Habermann AN, Notkin D (1986) Gandalf: Software Development Environments. IEEE Transactions on Software Engineering SE-12 (12):1117-1127
24. Heimbigner D Experiences with an Object-Manager for A Process-Centered Environment. In: Eighteenth International Conference on Very Large Data Bases, Vancouver, British Columbia, Canada, August 24-27 1992. pp 585-595
25. Henneman EA, Avrunin GS, Clarke LA, Osterweil LJ, Andrzejewski CJ, Merrigan K, Cobleigh R, Frederick K, Katz-Basset E, Henneman PL (2007) Increasing Patient Safety and Efficiency in Transfusion Therapy Using Formal Process Definitions. Transfusion Medicine Reviews 21 (1):49-57
26. Kadia R Issues Encountered in Building a Flexible Software Development Environment: Lessons from the Arcadia Project. In: Fifth SIGSOFT Symposium on Software Development Environment, Tyson's Corner, VA, December 1992. ACM pp 169-180
27. Lamb DA (1987) IDL: Sharing Intermediate Representations. ACM Transactions on Programming Languages and Systems 9 (3):297-318
28. Maybee MJ, Heimbigner DM, Osterweil LJ Multilanguage Interoperability in Distributed Systems. In: 18th International Conference on Software Engineering, Berlin, Germany, March 1996. pp 451-463
29. Medvidovic N, Oreizy P, Robbins JE, Taylor RN Using Object-Oriented Typing to Support Architectural Design in the C2 Style. In: Garlan D (ed) SIGSOFT '96 Fourth Symposium on the Foundations of Software Engineering, San Francisco, CA, October 1996. ACM, pp 24–32
30. Medvidovic N, Taylor RN (2000) A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering
31. Morrison R, Dearle A, Bailey PJ, Brown AL, Atkinson MP The Persistent Store as an Enabling Technology for Project Support Environments. In: Eighth International Conference on Software Engineering, 1985. IEEE, pp 166-172
32. OMG (1995) CORBA 2.0/Interoperability, vol OMG TC Document 95.3.xx. Revised 1.8 edn. Object Management Group, Framingham, MA
33. Osterweil LJ (1983) Toolpack -- An Experimental Software Development Environment Research Project. IEEE Transactions on Software Engineering SE-9 (6):673-685
34. Osterweil LJ Software Processes are Software, Too. In: Ninth International Conference on Software Engineering, Monterey, CA, March 30-April 2 1987. IEEE Computer Society Press, pp 2-13
35. Osterweil LJ Improving the Quality of Software Quality Determination Processes. In: Boisvert R (ed) The Quality of Numerical Software: IFIP TC2/WG2.5 Working Conference on

the Quality of Numerical Software Assessment and Enhancement, Oxford, UK, July 8-12, 1996 1997. Chapman & Hall London, pp 90-106

36. Osterweil LJ, Clarke LA, Ellison AM, Boose ER, Podorozhny R, Wise A (2010) Clear and Precise Specification of Scientific Processes. IEEE Transactions on Automation Science and Engineering 7 (1):189-195
37. Osterweil LJ, Fosdick LD (1976) DAVE -- A Validation Error Detection and Documentation System for Fortran Programs. Software Practice and Experience 6 (4):473-486
38. Perry DE, Wolf AL (1992) Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes 17 (4):40–52
39. Purtilo J Polylith: An Environment to Support Management of Tool Interfaces. In: SIGPLAN '85 Symposium on Language Issues in Programming Environments, Seattle, WA, July 1985. ACM pp 12-18
40. Purtilo JM (1994) The POLYLITH Software Bus. ACM Transactions on Programming Languages and Systems 16 (1):151–174
41. Raunak MS, Osterweil LJ, Wise A, Clarke LA, Henneman PL Simulating Patient Flow through an Emergency Department Using Process-Driven Discrete Event Simulation. In: 31st International Conference on Software Engineering Workshop on Software Engineering in Health Care, Vancouver, Canada, May 2009. pp 73-83
42. Reiss SP (1985) PECAN: Program Development Systems that Support Multiple Views. IEEE Transactions on Software Engineering SE-11 (3):276-285
43. Reiss SP (1990) Connecting Tools Using Message Passing in the FIELD Environment. IEEE Software 7 (4):57-67
44. Reps TW, Teitelbaum T The Synthesizer Generator. In: SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, April 1984. ACM pp 42-48
45. Richardson DJ, Aha SL, Osterweil LJ Integrating Testing Techniques Through Process Programming. In: SIGSOFT Third Symposium on Testing, Analysis, and Verification Key West, FL, December 13-15 1989. ACM, pp 219-228
46. Richardson DJ, O'Malley TO, Moore CT, Aha SL Developing and Integrating ProDAG in the Arcadia Environment. In: Fifth SIGSOFT Symposium on Software Development Environments, Tyson's Corner, VA, December 1992. ACM, pp 109-119
47. Ryder BG (1974) The PFORT Verifier. Software - Practice and Experience 4:359-378
48. Shewhart WA (1931) Economic Control of Quality of Manufactured Product. D. Van Nostrand Co.,
49. Simidchieva B, Engle SJ, Clifford M, Jones AC, Peisert S, Bishop M, Clarke LA, Osterweil LJ Modeling and Analyzing Faults to Improve Election Process Robustness. In: 2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections, Washington, DC, August 9-10 2010.
50. Sliski TJ, Billmers MP, Clarke LA, Osterweil LJ An Architecture for Flexible, Evolvable Process-Driven User Guidance Environments. In: Joint Eighth European Software Engineering Conference and Ninth ACM SIGSOFT Symposium on the Foundations of Software Engineering, Vienna, Austria, September 2001. ACM Press, pp 33-43
51. Smith RL, Avrunin GS, Clarke LA, Osterweil LJ PROPEL: An Approach Supporting Property Elucidation. In: 24th International Conference on Software Engineering, Orlando, FL, May 2002. pp 11-21
52. Snodgrass RT (1989) The Interface Description Language: Definition and Use. Computer Science Press, Rockville, MD
53. Sutton Jr. SM, Heimbigner D, Osterweil LJ (1995) APPL/A: A Language for Software-Process Programming. ACM Transactions on Software Engineering and Methodology 4 (3):221-286
54. Sweet RE The Mesa Programming Environment. In: SIGSOFT/SIGPLAN Symposium on Language Issues in Programming Environments, June 1985. ACM pp 216-229

55. Tarr PL, Clarke LA PLEIADES: An Object Management System for Software Engineering Environments. In: SIGSOFT Symposium on Foundations of Software Engineering, Los Angeles, CA, December 1993. ACM, pp 56-70
56. Tarr PL, Wileden JC, Clarke LA Extending and Limiting PGraphite- style Persistence. In: Fourth International Workshop on Persistent Object Systems, Martha's Vineyard, MA, August 1990. pp 74-86
57. Taylor RN, Belz FC, Clarke LA, Osterweil LJ, Selby RW, Wileden JC, Wolf A, Young M Foundations for the Arcadia Environment Architecture. In: ACM SIGSOFT Software Engineering Symposium on Practical Software Development Environments, 1988. ACM, pp 1-13
58. Taylor RN, Clarke LA, Osterweil LJ, W. SR, Wileden JC, Wolf A, Young M Arcadia: A Software Development Environment Research Project. In: ACM/IEEE Symposium on Ada Tools and Environments, Miami, Florida, April 1986.
59. Teitelbaum T, Reps TR (1981) The Cornell Program Synthesizer: A Syntax Directed Programming Environment. Communications of the ACM 24 (9):563-573
60. Teitelman W, Masinter L (1981) The InterLisp Programming Environment. Computer 14 (4):25-33
61. Visual Studio. (2010). http://www.microsoft.com/visualstudio/en-us/.
62. Wileden JC, Wolf AL, Fisher CD, Tarr PL PGraphite: An Experiment in Persistent Typed Object Management. In: Third ACM SIGPLAN/SIGSOFT Symposium on Practical Software Development Environments, Boston, MA, November 1988. pp 130-142
63. Wolf AL, Clarke LA, Wileden JC (1989) The AdaPIC Toolset: Supporting Interface Control and Analysis Throughout the Software Development Process. IEEE Transactions on Software Engineering 15 (3):250-263
64. Xiao J, Osterweil LJ, Wang Q Dynamic Scheduling of Emergency Department Resources. In: First ACM International Health Informatics Symposium, Arlington, VA, November 11-12 2010. pp 590-599
65. Xiao J, Osterweil LJ, Wang Q, Li M Dynamic Resource Scheduling in Disruption-Prone Software Development Environments. In: Fundamental Approaches to Software Engineering, Paphos, Cyprus, March 2010.
66. Young M, Taylor RN, Troup DB Design Principles Behind Chiron: A UIMS for Software Environments. In: Tenth International Conference on Software Engineering, Singapore, April 1988. pp 367-376