

Some Factors in a Science of Design

Dewayne E. Perry
ECE & UT ARISE
The University of Texas at Austin
perry@ece.utexas.edu

The engineering (ie, the designing and building) of software systems depends on three factors [1]: theory, experience and process. These factors suggest several interesting directions we might take in making progress in a science of design.

Depending on the circumstances, one of these factors may dominate the architecting and designing of a system. It may be the domain theory itself that dictates the architectural structure of the system, or at least provides its basic shape; certainly it is likely to dictate much of the algorithmic and data structures. At other times, it is experience that dominates: echoing Brooks [2], we certainly know better how to have designed a system after we've built it. From experience comes our knowledge of successes and failures, and with luck, our design judgment. At other times it may well be our processes (our methods, techniques, tools, environment) that shape our architecture and design.

Theory and Codification. One of the early hopes of software architecture was for the codification of well understood domains, the provision of architectural templates and architectural styles for specific domains, so that the designing of these systems could be standardized, and thus simplified. While there is some glimmer of this hope in the work on product line architectures, there has been little practical realization of it. We as researchers have spent our time on the easy aspects of creating architectural description languages, not on the hard part of designing and validating architectural templates or styles.

Experience and Evaluation. With luck, our experiences bring us both knowledge and judgment about designs and the act of designing. Sound judgments about design are fundamental to the engineering of useful software systems. Equally so for evaluating alternative decisions. Alas, we find ourselves in the position of art critics in evaluating designs instead of engineers with sound and reliable evaluative measures we have empirically derived and validated. We have (apparently) useful constructs (abstractions) with no reliable or valid observable measures. The solution to this problem is foundational to the progress in a science of design.

Process and Techniques. Design methods continue to be little more than a set of document templates and helpful heuristics. Except for having new sets of emperor's new clothes, we have not progressed much beyond where we were in the 70s. Fortunately, theory and experience have provided us with instances of domain specific techniques and design rules that we can use effectively in these domains. We should apply one of our primary design rules to how we think about design: it is only by narrowing our focus that we gain greater leverage in solving our problems. Only after we have sets of sharply focused, domain specific methods and techniques do we have a hope of generalizing across the entire domain of software engineering.

Summary. I fear we are still where we were in 1989 when Alex Wolf and I wrote our initial paper on software architecture [3]: *we have trained carpenters and contractors, but no architects.* While I agree with Brooks [2] that great architects and designers are born, we need to be able to train competent architects that will be able to do the bulk of the design work, freeing the great ones for the really critical systems.

[1] Dewayne E. Perry, *Dimension of Software Evolution*, Keynote, International Conference on Software Maintenance, 1994.

[2]Frederick P. Brooks, Jr. **The Mythical Man-Month**, 1995.

[3] Dewayne E. Perry and Alexander L. Wolf, *Software Architecture*, August 1989. Published as *Foundations for the study of Software Architecture* ACM SIGSOFT Software Engineering Notes, 17:4 (October 1992).

Biography.

I have spent approximately half of my career as a practicing software engineer designing and building systems that ranged from very small to extremely large and over a wide variety of different domains. From 1973 through 1983 I had my own consulting firm, Pegasus Systems, providing primarily system architecture and design services. Almost all my development experience was at the leading edge both in terms of technology but also in terms automating new domains.

Therefore, from my practical experience I bring a substantial experience in software architecture and design in a wide variety of domains both in terms of new systems and in terms of evolving established systems into new uses and features.

After having gotten my PhD in 1978 in the midst of my practical phase, I then began to straddle both camps: research and practice. Research began with working with Nico Haberman, CMU, and led to a visiting, part-time faculty position at CMU working with Nico both individually as well as with the Gandalf Project. My research with Nico focused on design issues in version management systems and functional programming languages. There were other research projects with Fort Monmouth as well.

My full-time research career began with joining Bell Labs in late 1983. There my research my research was generally done in conjunction with software development labs. The broad areas of my research were empirical studies, light semantics techniques and tools, software development environments (SDEs), software architecture and software processes. Of course my research in the practical use of formal specifications (Inscape), SDEs, various process formalisms, visualization and analysis techniques and tools all involved issues of design. However, design was not the primary focus.

Of direct importance to this workshop are my research in empirical studies and software and software architecture. The software architecture research grew out of a joint project with an architecture recovery group. They were working bottom up trying to find out what was there and reconstruct the architecture from the components. Alex Wolf (now at U Colorado, Boulder) and I were trying to determine what a software architecture ought to be. This led to our seminal paper *Foundations for the Study of Software Architecture* [3]. Subsequent joint projects with developments labs included research about the design of generic, or product line, architectures for dynamically reconfigurable systems (obviously in a specific practical domain).

Also of direct relevance is my empirical work in terms of observational, relational and experimental studies of software development processes, how developers spend their time, how developers work in cooperation separated both geographically and temporally, parallel changes etc. One important aspect of this work has been that where we have used the empirical work as the basis for the design of tools, techniques and processes. This research provides the practical basis for considering the important issue of empirically measuring and evaluating system architectures and designs.

I came to UT Austin at the beginning of 2000 and my research focus has been on the design of an analysis system to detect semantic interference in parallel changes and, more important directly, on the transformation of goal directed requirements into architectural prescriptions (an NSF funded project). Also of importance is my graduate course on empirical software engineering where we cover the critical issues in the various kinds of empirical studies and the specific problems in software engineering, one of which is the issue of construct validity (directly relevant to the problems of evaluating designs and design alternatives).