

Real Design Feedback: A Necessity for Real Design Progress

Mats P.E. Heimdahl and Eric Van Wyk
University of Minnesota

Design is the process followed to construct an artifact that is fit for purpose. A *design* is an artifact capturing some essential qualities of a product to be constructed—in this context a software intensive system. The key element of the design processes as well as design artifacts is that they help solve the problem at hand—if design processes and artifacts help an analyst achieve his/her design goals, it is a good process or artifact.¹ Thus, *the goal of the research community* is to provide design theories, methods, and tools that *help practicing designers solve practical design problems*.

Based on years of experience working with engineers in various critical systems domains (air traffic control, flight guidance, munition fuzing, cardiac phasing, etc.) we offer the following conjectures that will guide the remainder of the discussions and our recommendations for future research directions.

Conjecture 1 *No design process or design artifact will be universally accepted, nor universally applicable; even closely related domains, such as avionics and medical technology, have justifiably different and entrenched views of which design artifacts are useful and worthy of design effort.*

Conjecture 2 *Solutions to today’s and tomorrow’s design problems will rely on tool support to increase productivity, reduce cost, improve reliability, or whatever the design goals may be.*

Conjecture 3 *To make progress in design research, practicing designers must evaluate proposed solutions on practical design problems; if proposed theories, methods, and tools do not solve real problems, they are of little or no value.*

Since design is about solving real problems in industrial settings, the only way to evaluate proposed research solutions is to put them into the hands of practicing designers so that they can apply them on realistic problems and determine what actually works in their domain. To achieve this, two general problems must be addressed. First, close collaboration between industries involved the design of software intensive systems and the research community must be established to facilitate technology evaluation and technology transfer. Second, the research community must be able to provide their industry partners with well supported tools so that they dare to adopt them in their design activities. To illustrate our point, we offer the following experiences from our own work.

We have worked in several projects developing formal specification languages and accompanying static and dynamic analysis tools. The work has been largely successful and some technology transfer is taking place [4], but, largely, the industrial adoption of promising tools and techniques has been small. Typical obstacles to successful collaboration are industrial comments such as “*We like what you are doing, but you do not have internal events like Esterel [1]*”, “*We like SCR [3], can you work with that?*”, and “*We need analysis support for SCADE’ [2]*”. Naturally, these languages are all semantically equivalent and potentially useful *language independent* modeling, analysis, and testing techniques are not evaluated for all the wrong reasons, and promising collaborations do not materialize over the perceived “academic” nature of the supporting tools. These requests could be accommodated, we could modify our tools and techniques to address the concerns of each industrial partner, but we would spend all our time making superficial tools changes without making any real progress on the real design problems.

In the programming language community many interesting concepts and language extensions do not get exposed to developers where they can be evaluated and adopted. To make an extension to Java, such as open-classes or aspects, researchers typically build a monolithic compiler extended with a new feature. There is no reuse of a compiler infrastructure that would allow one to specify the extension as a module that can be imported into a developer’s existing, and trusted, compiler.

¹We here use analyst in its broadest sense to include all software professionals involved in a design process.

This limits the idea's exposure. In recognizing this problem, work on product family architectures and generative programming aims to provide tools that domain experts can extend with domain specific constructs. Robison [6] further argues that compilers be built in an extensible manner that allow users to specify domain specific optimizations and import them into their compiler. Work on extensible compilers has been done where expressive constructs, such as AspectJ-like aspects, can be imported by the programmer into the compiler as a modular language extension [7]. These notions of modularity and extensibility apply to tools beyond the formal specification tools and compilers we used as examples here.

Any such tools infrastructure, be it for programming, specification, design, or work-flow languages, that allows for the type of reuse we seek should allow tools to be specified and constructed in a *highly modular manner* since different components would need to be composed in order to create specific tool artifacts. The components of an infrastructure must also be *extensible* so that tools can be easily extended with features specific to a particular target domain. Components must also be *amenable to analysis* so that one can statically determine component's compatibility. Finally, these components must work at a fine level of granularity in order to provide precise control over the tools being developed so that they can meet the demands of the target domain. Eclipse [5] is an example of a powerful and well-used tools infrastructure, but the level of granularity is, in our opinion, too coarse to address challenges we have faced in our collaborations with industry.

What can the NSF do to help? Firstly, we assert that the research community must abandon its reliance on tools developed largely by individual research groups. Instead, we must increasingly work towards modular, extensible, and robust tools that can make the transition to industrial use and allow research breakthroughs to be quickly adopted and evaluated by industrial users, research peers, and students. Such an effort would necessitate research programs into *how* to build such flexible environments for various design domains and design activities. Furthermore, the various communities related to design must show *real leadership* so that researcher can organize around tools infrastructure efforts that will provide the platforms for design research over the next 10 years. Secondly, programs encouraging (or requiring) industry/academia collaboration must be expanded (or established in areas where they do not exist). We cannot, unfortunately, offer any solution on how to structure such programs. Possible programs may involve faculty leave programs to join development organizations, industry sabbaticals in academia, as well as more traditional project based collaborations. Any program must, however, encourage ownership and commitment by the industrial partner so that new techniques can be put though realistic design challenges and be used by practicing analysts to provide *real design feedback* on our work so that we can make *real progress* in our design research.

References

- [1] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.
- [2] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.
- [3] K. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering*, 6(1):2–13, January 1980.
- [4] S. P. Miller, A. C. Tribble, and M. P. Heimdahl. Proving the shalls. In *Proceedings of the 12th International Formal Methods Europe Symposium-FME'03*, Pisa, Italy, September 2003.
- [5] I. Object Technology International. Eclipse platform technical overview, Feb. 2003. Available at www.eclipse.org.
- [6] A. D. Robison. Impact of economics on compiler optimization. In *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, pages 1–10. ACM Press, 2001.
- [7] E. Van Wyk. Aspects as modular language extensions. In *Proc. of Language Descriptions, Tools and Applications (LDTA)*, volume 82.3. Elsevier Science, 2003.