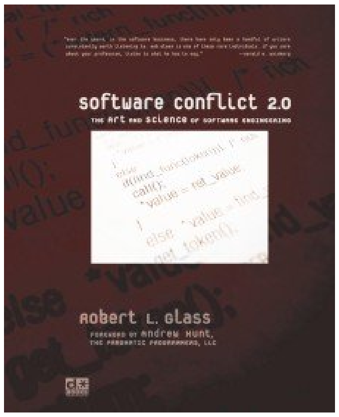
Book Excerpt



Software Conflict 2.0:

The Art and Science of Software Engineering

by Robert L. Glass

ISBN 0977213307; US $29.99 / UK £22.99; 308 pages

The Cognitive View: A Different Look at Software Design

The question I want to deal with here is "What is software design?"

That may strike you as an odd question to ask. After all, people have been designing software for 30+ years now. And don't we have all the design methodologies and design languages common sense says we need? So why, now, ask "What is software design?"

Well the point I want to make is that all this time, when we've talked about design, we've been playing around on the periphery of the subject. Methodologies aren't design; they're frameworks for organizing our design efforts. Languages aren't design; they're representations for writing down the design once we get it. Design is something that happens inside the head, inside the brain, and it happens at a speed faster than lightning. It's *that* concept of design that I want to pursue here.

I've done my share of software design over the years, and I've taught the subject a few times, but I want to confess something to you. I've never felt that I really understood what design was, and whether what I was doing was the right approach to design.

In recent years, I've found I am not alone. While I was teaching software engineering in the graduate program at Seattle University, some of us avoided teaching design because we felt we didn't know what is really was, and therefore we didn't know how to teach it. Then I ran into some people during my year at the Software Engineering Institute at

Carnegie Mellon who not only didn't feel they knew how to teach design, but in fact felt that no one knew how to teach it! At that point, I felt less alone, but I was still no closer to understanding what design really was.

Empirical Research Findings

All that is beginning to change. Other computer people who also felt that design is often an elusive subject have been conducting research into what it is. And they're beginning to come up with answers.

To understand what follows, you may have to loosen your grip on a few ideas that have become tradition in the short history of software. Quit thinking of external representations as what design is all about, and focus on mental process. Inside the mind lies the secret of design.

I still remember the occasion when that became clear to me. I was attending a class in the Ada language at the Air Force Academy. It was the last day of the week-long class, and the instructors had divided us into teams and given us a problem to solve. They had just finished laying out the requirements for us when one of my teammates, a bright young man from Boeing, came up with a design solution. Here I was, still trying to grasp the problem statement, and he had whipped through a design solution in his mind! It was then that I realized that design happened lightning fast inside the mind, and incidentally that some people's lightning struck much faster than others!

But what really happened inside that bright young man's mind? Answers to that question, at least the general question behind it, are beginning to emerge.

Before I talk about what those answers are, let me explain a little about where they're coming from. There's a new thrust emerging in software-related research, and it's from that thrust that the answers are finally coming. Researchers such as Bill Curtis at the Microelectronics and Computing Consortium (MCC) and Elliot Soloway of the University of Michigan (formerly of Yale) have long been interested in the idea that the study of software was at least as much about studying programmers as it was about studying programs. They've called this field of research "empirical studies of programmers," and the latest focus of this research area is on software design.

Confronted with the same question that I posed at the beginning of this article ("What is software design?"), these researchers laid out a plan for finding out. What they needed to do was to capture the thought processes of designers doing actual design work without intruding on, and therefore perturbing, the process. That's easier said than done.

But with the methods of a field called “protocal analysis” they have managed to do that. They have sat quietly with designers at work and prodded them to think aloud, recording what they say. They have audio taped design sessions. They have video taped group design processes. They have poured over the results of the recording process. And they have begun to formulate a theoretical description of what designers do.

The first set of things they learned wasn’t all that illuminating. They found that design involved:

- understanding the problem
- decomposing the problem into goals and objects
- selecting and composing plans to solve the problem
- implementing the plans
- reflecting on the product and the process

All that was not very helpful. With a little tweaking of the words, in fact, that sequence is not much more than what we call the software life cycle, a thing we’ve known about and wrestled with for years.

It wasn’t until they delved more deeply into “selecting and composing plans” from the preceding list that they finally struck pay dirt. Lurking inside that generic category was a simple set of steps that is the essence of design.

The Essence of Design

What were those steps?

The designers, mentally and at lightning speed, were doing the following things:

1. They constructed a mental model of a proposed solution to the problem.
2. They mentally executed the model—in essence, running a simulation on the model—to see if it solved the problem.
3. When they found that it didn’t (usually because it was too simple), they played the inadequate model back against those parts of the problem to see where it failed, and enhanced the model in those areas.
4. They repeated steps 1-3 until they had a model that appeared to solve the problem.

Now, since those four steps are the key to the answer to our original question, let’s spend a little more time on them before we move on. What we see here is a mental process, a very rapid process, an iterative process, a process in fact of fast trial and error. The mind forms a solution to the problem, knowing that it will be inadequate because the mind is not yet able to fully grasp all the facets of the problem. That problem solution, the mind knows, must be in the form of a model, because it is going to be necessary to try sample input against the model, run a quick simulation (inside the mind) on the model using the sample input, and get sample outputs (still inside the mind) from that simulation.

The essence of design, then, is rapid modeling and simulation. And a key factor in design is the ability to propose solutions and allow them to fail!

It is interesting to mention in passing that these same researchers have explored the problems of people who are not very good at design. Those people tend to build representations of a design rather than models; they are then unable to perform simulation runs; and the result is they invent and are stuck with inadequate design solutions. One of *my* favorite thoughts connected with these findings is that failure is an essential part of successful design! The end of every one of the previous iterations is a failed model, one shown to be inadequate to solve the problem; and that in turn suggests that an integral part of success is the ability to fail and to recover from it. I find that thought has intriguing implications from the point of view of teaching design, or any other subject for that matter. Where do we teach failure and how to recover from it?!

Several famous software designers have articulated some of these same ideas about design. In *Programmers at Work*, published by Microsoft Press from interviews conducted by Susan Lammers, they commented on the design process as they saw it, saying things like:

“The first step in programming is imagining. Just making it crystal clear in my mind what is going to happen. In this initial stage, I use pencil and paper. I just doodle . . . because the real picture is in my mind.” (Charles Simonyi, creator of Multiplan)

“At some point, the [design] gets explosive and I have everything inside my brain at one time. . . . All sorts of things go on in my brain that I can’t put on paper because I’m always changing them.” (Gary Kildall, creator of CP/M)

“You have to simulate in your mind how the program’s going to work. . . . When you’re creating something . . . and you have that model in your mind, it’s a lonely thing.” (Bill Gates, chief executive of Microsoft)

It is fascinating that the findings of the researchers in many ways match these informal quotations from well-known designers.

Some Other Findings

There is one other key finding about design that enters into this. Other researchers, such as Willemien Visser of *Institut National de Recherche en Informatique et en Automatique*, France, have found that "designers rarely start from scratch." That is, they use an existing model from a prior solution of a similar problem, as that first cut at the model that begins the simulation process. In fact, in retrospect, I think *that's* how the young man from Boeing ran such mental circles around me at the Ada class. He'd solved that class of problem before, and he had a tentative solution already lurking in his mind! (Isn't it amazing how well we can rationalize our personal inadequacies?!)

Now so far, this has all been about individuals doing design. But as we know, design in the 1980s has evolved into a team process. Problems have gotten too big for individual designers, and interdisciplinary problems cause a need for multiple and diverse skills among designers.

Those same researchers have been looking into team design. And they've found that, in many ways, it's a shared form of individual design:

- The teams create a shared mental model.
- Team members, sometimes in their own minds and sometimes in the group, run simulations against the shared model.
- The teams evaluate the simulations and prepare the next level of the model.

But in some ways these efforts are unlike individual design:

- Conflict is an inevitable part of the design process. It must be managed rather than avoided.
- Communication techniques become a vital part of the design process.
- Issues sometimes "fall through the crack" because no individual claims responsibility for them.

These teams of designers are typically 3 to 6 people. Sometimes, for enormously complex tasks, even that is not enough people. Then, design becomes an organizational problem. Typical organizational design evolves into a hierarchy of design teams, each with its own assigned problem subdomain, with an additional special team of chief architects whose job it is to hold the whole design effort together.

But design in team and organizational settings creates its own problems. Design can easily evolve into committee work, with all the disadvantages that implies (remember that the “camel is a race horse designed by a committee”). Fred Brooks, of *The Mythical Man-Month* and the more recent Silver Bullet paper, has pointed out that the best products we see, the ones that we agree have conceptual integrity (like Pascal and Unix), were designed by individuals. There *are* successful team designs, like Ada and COBOL and the IBM mainframe operating systems, but they are generally looked down on as being successful but clumsy, as Brooks points out.

Where Do We Go From Here?

So we have some new understandings. There is a well-defined process that goes on inside the mind during design. It begins with an existing or simplified new model, runs a simulation on that model, and continues iteratively until the design solution meshes with the problem to be solved. If the problem is big enough or complex enough, then teams or even organizations may do design. They use many of the same techniques as individuals but also use group process, a clumsy but sometimes necessary way to do business.

What do we do with these new understandings?

It seems to me that there are three categories to deal with that: the implications on the teaching of design, the implications on the doing of design, and the implications on the managing of design.

In teaching design, it is no longer enough to teach one or two or three methodologies and representations. Those older topics must be taught within a framework that includes the new concept of design as a mental process.

In doing design, it is helpful for the designer to understand that the heart of design is not what he or she thought is was supposed to be, and that the clumsy iterative process of trial and error that the designer is probably pursuing is actually the way it is supposed to be. That may give designers the confidence to pursue these appropriate design approaches without guilt. (Could we dare call this a “guilt-free” approach to design? No, that would be too much psychobabble!)

In managing design, managers can focus on communication facilitation and conflict resolution as their contribution to design. The empirical studies researchers further suggest that management of design should be the management of the key issues that arise during the design process.

In pursuit of these goals of better design education, practice, and management, several tool concepts are proposed by the researchers. We do not know how to build all of these tools yet, but if we did it would go a long way towards assisting the now-better-understood design process.

1. Modeling and simulation packages to support the mental process.
2. Idea archive and retrieve packages to prevent thoughts from falling through the cracks.
3. Strategic assumption surfacers that keep track of key requirements and pop them up when a candidate design is about to violate one.
4. Issue-based conflict resolution support.
5. Unresolved issue recording/tracking.
6. Mediated discussion support.
7. Group idea collection and coordination.

It may be premature, however, to talk about tools to support this process. We are just beginning to understand what design is really all about, after thinking we knew what it was about for over 30 years. Perhaps just that should be enough to absorb us for awhile. Figuring out what to do with that knowledge may be an issue for a later time.

We probably ought to have a handle to attach to this new understanding. The researchers call it “cognitive processes in design.” I like that. It says the things that I believe should be said about design being a mental process.

This article, then, has been a look at the cognitive aspects of software design.

###

A Retrospective Afterword

Wow! A lot of things have changed since I wrote that essay some 15 years ago. It refers to the pioneering design studies of Bill Curtis and Elliot Soloway as "recent" (they happened in the late 1980s!). It presents the thoughts of such sterling technologists as Charles Simonyi, Gary Kildall, and Bill Gates. (But who has ever heard of *them* recently? Well, I've heard of this guy Gates, but I can't remember the context!) It discusses such "modern" programming tools as Pascal and Unix.

But the really funny thing about this essay is this: The technical essence of what I said there, I believe, remains valid today. First of all, there has been no follow-on research that would obsolete what Curtis and Soloway learned about design in practice way back then. In fact, there is every reason to believe that their empirical studies of programmers at work captured some vitally important and still-true facts about how real programmers do real design. I present these same findings in classes and lectures all the time to practitioner audiences, and not one person has said "hold it, those ideas are totally (or even partially) out of date." If there are classic findings and classic publications in the computing field, this report on the work of Curtis and Soloway belongs in that category (he modestly said!)

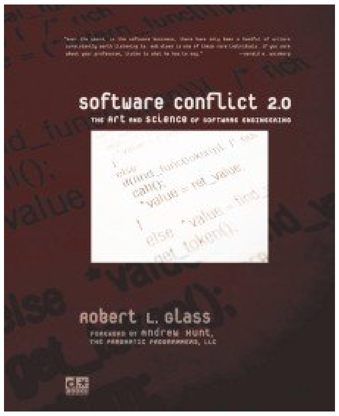
You might be interested in the bottom line to the Curtis/Soloway research. The original intent of their studies was to result in a toolset that designers would be able to use to improve the way they did business. I hinted at that, in fact, in my essay. But that bottom line was to be dashed. Because the nature of design, as they discovered, is so intensely cognitive, happening inside the mind at mind speed, the researchers could conceive of no useful tools to help in that process! Eventually, their efforts were disbanded and they moved on to what I prefer to think of as browner pastures!

—Robert L. Glass, 2005

About the Author

Robert L. Glass held his first job in computing in 1954. Author of over 25 books, he is one of the true pioneers of the software field. He is the editor and publisher of *The Software Practitioner*, and also writes regular columns for *Communications of the ACM* and *IEEE Software*. In 1995 he was awarded an honorary Ph.D. from Linkoping University of Sweden, and in 1999 he was named a Fellow of the ACM professional society. His unique viewpoint and timeless writings have for decades offered insights to practitioners, managers, professors, entrepreneurs, researchers, and students alike.

About the Book


software conflict 2.0
The Art and Science of Software Engineering
Robert L. Glass
Foreword by Andrew Hunt, The Pragmatic Programmers, LLC

Title	*Software Conflict 2.0: The Art and Science of Software Engineering*
Author	Robert L. Glass
Foreword	Andrew Hunt, Pragmatic Programmers, LLC First edition Foreword by Donald J. Reifer, Reifer Consultants Inc.
ISBN	0977213307
Pages	308
Price	$29.95 U.S. / UK£22.99

(continued)