Blueprints

Leslie Lamport
Microsoft Research

25 September 2010

Abstract

Verification techniques have been used extensively for checking code. This is important and successful work, and it will be the main and perhaps the only topic of this workshop. However, I believe that in many cases, the possibility of producing good software vanishes the moment coding starts. Below is the opening section of a hyperbook I'm writing that I hope explains why. The little of the hyperbook that has been written is available at

`http://research.microsoft.com/en-us/um/people/lamport/tla/hyperbook.html`

A number of years ago, my wife and I had our house remodeled. When the architect came over to meet us and see the house, I expected that we would discuss the placement of walls and the location of appliances and cabinets. Instead, he questioned us about our lifestyle: Where did we spend our time? Did we cook a lot? How often did we have people over? He then sketched the layout of our house and left to think about what we had told him.

Later on in the design process, we discussed one design that would require removing an existing beam. I asked him if this would be structurally sound. He replied that he was quite sure that the remaining beams were adequate, but we didn't have to trust his intuition. When he went back to his office, he would use the plans to calculate the stresses and check that it was safe.

We went through several designs. The architect would draw the plans, and my wife and I would go over them—looking for problems and thinking of possible improvements. When we were satisfied, he drew a set of blueprints and made final cost estimates, and we hired a contractor.

During construction, some small problems were discovered and minor changes were made to the design. The contractor took longer than expected,

but overall construction was largely uneventful. We have been very happy with the results.

Not long ago, I attended a workshop on concurrent programming. There was a break-out session in which some ten of us discussed how to write correct programs. Several ideas were presented. They all involved either tools for finding errors in the code or better programming languages.

As people spoke, I compared what they were saying to my experience remodeling our house. In my mind, I translated their proposals for eliminating errors in programs into the analogous ones for houses. They wanted to avoid mistakes in building houses by providing new construction tools or better building materials—bigger hammers and stronger nails, or better concrete. It never occurred to them that to build better houses, you should start by drawing plans.

It is obviously much easier and cheaper to correct problems in houses or in programs when designing them, before construction begins. But this group of system designers and computer scientists were acting as if they had never heard of blueprints.

Programming is very different from building houses. Programs are much more complicated than houses. To me, programs seem to differ from one another much more than houses do. (Perhaps architects have the opposite view.) Houses use a small number of well-understood design elements such as doors, windows, floors, and roofs. We have yet to develop corresponding design elements for programs. These differences make careful design and planning even more important for programming than for building houses. Writing a complex program without a blueprint is much harder and more error prone than building a house without one.

Programs also differ from houses by being non-physical objects. This makes them appear to be easier to modify than houses. Why bother drawing blueprints if it's so easy to fix things? Programs are not easier to modify than houses. It's much faster and cheaper to move an interior wall of a house than to modify an internal interface of a complicated program. It's a lot easier to fix a blueprint than to fix either a house or a program.

Blueprints for programs are called specifications. This hyperbook is about writing and checking specifications. People have given many reasons why these specifications are a waste of time: you can't generate code from them; you can never be sure that a program implements its specification; programs need to be changed after they are written, and there is no way to ensure that their specifications are changed accordingly; there are things like "ease of use" that can't be specified. When you hear such an argument, translate it to the corresponding statement about blueprints and houses.

You will find that the argument is both true and irrelevant.