# Myths and Mythconceptions in Software Design

## Mary Shaw

CMU/ISR, draft of 1/10/2017

“Design”, as used in the software domain, is too often bound up with developing code to meet specifications. Today’s problems are richer and more embedded in complex contexts that the conventional wisdom can support. Here I identify some of the problematic, but persistent, myths and mythconceptions about software design and suggest more realistic aspirations.

|  |  |  |
| --- | --- | --- |
| **The Myth** | **What we should be addressing** | **Reading** |
| Myths about the nature of our problems | | |
| Well-defined specification (RFP) | Wicked problems – sociotechnical problems or ultra large-scale systems | Horst W.J. Rittel and Melvin M Webber. Dilemmas in a general theory of planning Policy Sciences, vol 4 no 2, pp. 155-169, June 1973 |
|  | Software development processes often assume well-defined problems. However, in many settings we often can’t get consensus on either the problem definition or the evaluation criteria | |
| Correctness, preferably via proof | Acceptability in context |  |
|  | Software engineering doctrine emphasizes correctness. But in practice we have always tolerated incorrectness – even major products are released with known bugs. Indeed, we often don’t really know exactly what practical software does at all the corner cases, but we have some confidence that its behavior is probably within some envelope of acceptability | |
| Myths about the nature of design | | |
| Design as problem solving | Design as problem setting | Willemien Visser. The Cognitive Artifiacts of Designing. Parts II,III,IV. Lawrence Elbaum, 2006 |
|  | Simon studied design as problem solving, which assumes that the problem was well posed. Schoen argues that problem setting is equally important – that design is a dialog among the designer and the materials in which the problem is refined as the affordances of the materials interact with it | |
| Design as writing UML | Deciding what to write in UML | Michael Jackson: Problem Frames Addison-Wesley, 2001 |
|  | Writing high-level descriptions in a notation such as UML is often thought of as design. Yes, at some level, but a more significant kind of design takes place long before anything as specific as UML enters the picture. This includes working through the tradeoffs among constraints, exploring alternative approaches, translating the requirements into an architectural vision, and ??? | |
| **The Myth** | **What we should be addressing** | **Reading** |
| Myths about the solutions or implementations | | |
| Find any solution | Explore design space breadth first | Design Spaces |
|  | Some software development processes assume that the solution will be developed in a particular framework. Others tend to go depth-first, seeking the first apparently acceptable solution. A better path to satisfactory, cost-effective solutions is to thoughtfully consider the opportunities in the domain, specifically by systematically exploring the design space that is generated by considering the design decisions to be made and the alternatives for each decision | |
| Increments, but correct ones | Progressively better approximations |  |
|  | We must come to grips with software for which we don’t have structural explanations of its behavior. This started with rule-based systems in the 70?s, which received a chilly reception in the software community because they started with rules that covered common cases and expanded the rules set as the problem was better understood. (It turns out that you need structure and precedence; an undifferentiated bucket of rules doesn’t work.) Systems based on machine learning seem to be a modern incarnation of this – you train the system, see whether it behaves acceptably, then retrain. But note that there’s a big difference between automatic generation of possible actions for review and automation that does not review the results before executing them | |
| Code | Designer intent, architecture, component orchestration, invariants,… |  |
|  | We are still shackled by the notion that the code is the definitive representation of a solution. This discards all of the designer’s intent and intended system structure. It is true that current practice allows the code to drift from the intent. The correct response is not to neglect “documentation” because it becomes obsolete, but rather to drive the build from the higher-level documents, so they can’t diverge without breaking the build. | |
| The general theme here is learning to cope with increasing uncertainty – in our understanding of the problem, of our ability to determine acceptability of a “solution”, and in the very processes that create the product. Some of the uncertainty arises from problem complexity, especially when multiple diverse stakeholders are involved. Other kinds of uncertainty arise from solution strategies that do not lend themselves to exact analysis and explanation, such as machine learning. In addition, our long-standing faith that we can actually reason about correctness continues to be thwarted by the complexity of that analysis.  So, has all of our effort to introduce precise reasoning, careful analysis, engineering discipline, and other systematic processes been for naught?  No, because for science, | | |
| Finding truth | Seeking truth |  |
|  | Many, especially the public, think of science as delivering truth; in fact, it’s the best way we know to seek truth, delivering the best knowledge we can along the way. In the same way. Science does not actually promise perfection today, just a commitment to strive, evaluate, and be clear about the limitations of the knowledge | |
| Thus we should recognize the conventional wisdom as the best we can do at the moment – but we should welcome attempts to go beyond those constraints, not reject those attempts as violating established norms.  The question now is what it means to do engineering in this world.  Finally, currently accepted software engineering research paradigms don’t suit design | | |
| Myths about the solutions or implementations | | |
| research validation via proof or empirical study | ??? |  |
|  | I’m stuck on this one, but we need to think about both what we accept as evidence and how we convince the profession that it’s good enough | |

Oh, and we need tangible, visual stuff to inspire logos