

Retrospective and iterative design

Michael D. Ernst
MIT Computer Science & Artificial Intelligence Lab
mernst@csail.mit.edu

This position paper addresses automation of design-related tasks, integration of design with other system development activities, and managing the inherent imprecision present in designs.

Traditional ahead-of-time design has not lived up to its promise. We do not understand design and designs well enough to successfully estimate costs; predict characteristics of the final system, including interactions with the environment and its users; and guide implementation and future design decisions. Furthermore, the state of the design art is not always used in practice. This indicates a pressing need for research to address these problems. We need better understanding of design and designs, and we need better understanding of the limitations of our knowledge.

In most domains, ahead-of-time design is not a practical approach. Lack of information makes it infeasible and undesirable to complete a design before embarking on construction. However, neither is it reasonable to terminate design activities once construction commences. Design must be integrated into all parts of system construction. Design should not be viewed as a separate activity. Design should not even be viewed as a special activity: it cannot take precedence over other tasks, even those like implementation and testing that traditionally follow it in the project's lifecycle. (The best designers already perform this integration mentally by visualizing how a design will be achieved and thus predict possible problems or features of the design.) Integrating design with other activities will have greater benefits both for design and for all other system-building activities.

Doing exhaustive design ahead of time may not be desirable, even if it were feasible, because of uncertainty — and because of the certainty of change. (The more successful the design and the more long-lived the resulting system, the more change there will be; thus, a successful design will eventually become less appropriate, less clear, and less true to its original conception.) A number of development methodologies recognize the need to integrate design into the development process; for example, the spiral development model calls for multiple distinct design phases, in rotation with other tasks. Extreme Programming (XP) forbids design that makes allowances for future features. XP works well in certain environments, such as for relatively small projects; it is a challenge to apply its lessons more broadly.

Many real systems have imperfect designs. Furthermore, this problem will be with us forever, regardless of our success at formulating a science of design. (Therefore, a science of design must account for systems that do not have a design!) Few designs perfectly model the constructed system; artifact understanding and design recovery are crucial in such circumstances. For legacy systems, the design may have been lost or partly lost, or be in an inappropriate formalism; for successful systems, more effort is devoted to maintenance than to design and construction, and legacy systems will always be with us. Newly constructed systems may have been constructed using a methodology that advocates lightweight or just-in-time design, or with poor or no design methodology. Or, the system developers may have determined that constructing or documenting a design was not a worthwhile activity; economic real options theory indicates that it is advantageous to defer activities (perhaps including design) with lower expected value. Alternatively, the implementation might have begun before design was complete, or might use pre-existing or separately constructed components that may not have their own designs or may not mesh perfectly with the overall system's design.

In order to address these problems, I propose retrospective and iterative design. Retrospective

design can also be viewed as design recovery or as artifact understanding. Iterative design can be viewed as performing design continuously through the lifecycle of a project, not as a separate activity or in separate phases. It is essential, for both of these tasks, to recognize, accept, and manage the fact that designs are inherently imperfect and approximate.

Retrospective design starts with an artifact and perhaps a partial design. It analyzes the artifact to determine a (potential) design for that artifact, and to reveal key properties of the artifact and of the design. This methodology may seem backward, because it reverses the usual order of system construction, but it is highly practical because: for the reasons described above, designs are often unavailable for real systems, and this will always be the case. Retrospective design can use static analysis, dynamic analysis, or a combination of both to capture system properties and thus to recover parts of the design. Examples of properties recovered may include which components communicate with one another (and how), design patterns appearing in the system, invariants over the system's data structures and computations, etc. People play a key role in the process of retrospective design as well as using its outputs. As one example, a human can compare his or her beliefs about a design to the properties of the artifact, and thus find errors in the beliefs, the design, or the artifact.

As a concrete instance of retrospective design, my research on dynamic detection of likely program invariants provides valuable information about the structure of a software system. It performs a dynamic analysis over a running software system and reports properties that were always true at run time. For instance, these might be that a particular parameter to a routine always had a specific type; that a list was always sorted or always contained a particular element; data structure invariants such as that linked list pointers are internally consistent; and the like. This research provides a start at design recovery and retrospective design; it would be worthwhile to extend it further to provide more design-level information.

Iterative design encompasses design after part of an artifact has already been completed; re-design; design in the presence of changing requirements; and adjusting a design in response to changes to an artifact. Retrospective design is an essential enabling technology for iterative design, because iterative design must take account of, and respect, existing components and their interactions. Iterative design goes further in comparing versions of requirements, designs, systems, and in being part of a continuing process rather than in providing the information needed to start a process. Successful iterative design makes the design a living rather than a static artifact, supports development activities, uses automatic consistency checking (within the design and between the design and the artifact), and explicitly omits (from the methodology and the design) the parts of design that are beyond the state of the art.

While some of my previous research has been in the area of recovering information from artifacts, my interest in the science of design is considerably broader. In particular, I believe the field must also examine the inherently inexact nature of design. No design is perfectly precise, for a number of reasons. First, the artifact or domain may not be perfectly modelable in the formalism chosen for the design. Second, a design is an approximation rather than a perfect model of the artifact or its domain. Third, few designs are internally consistent. Fourth, a design is a complex and subtle agglomeration of ideas; humans cannot expect to have perfect understanding of it. Thus, a science of design must address imprecision. We must avoid an all-or-nothing approach that insists designs can be complete or even that designs are accurate. We know that people are highly resilient to moderate amounts of imprecision (even in the domain of program semantics), and my research and that of others has shown that the same can be true of input to program analysis systems. Furthermore, we must not create merely a science of design; a science that does not encompass integrative activities would be barren. Instead, we must focus on all parts of system construction and how design interacts with (helps and is helped by) them.