Formally Specifying Engineering Design Rationale

Keith Williamson, Michael Healy
The Boeing Company, Seattle, Washington
Keith.Williamson@boeing.com

Abstract

This paper briefly describes our initial experiences in applied research of formal approaches to the generation and maintenance of software systems supporting structural engineering tasks. We describe the business context giving rise to this activity, and give an example of the type of engineering problem we have focused on. We briefly describe our approach to software generation and maintenance, and point out the challenges that we appear to face in transferring this technology into actual practice.

1. Introduction

With the advent of intelligent computer aided design systems, companies such as Boeing have embarked on an era in which core competitive engineering knowledge and design rationale is being encoded in software systems. The promise of this technology is that this knowledge can be leveraged across many different designs, product families, and even different uses (e.g., manufacturing process planning or diagnostic systems). This should lead to higher quality software and lower costs for software development and maintenance. Software construction by the assembly of reusable library components has been suggested as a way to achieve these goals. However, this promise has been hard to achieve.

There are at least two types of problems with this approach to knowledge reuse. First, software components often have assumptions or constraints on their use that are not clearly or explicitly stated. Second, even when these assumptions are clearly and explicitly stated, the assumptions that were applied when the software was originally written may turn out to be different than the assumptions that apply when someone else tries to reuse that component at some point in the future.

A fundamental problem in this paradigm of reuse is that what we are trying to reuse is software - the end artifact in a long and complicated process that goes from requirement specifications to implementations, via a process of design. The first problem above arises from the lack of traceability of requirements and specifications down to the software. The second problem arises from the inability to tailor the software to slightly different requirements. Knowledge sharing and reuse can not easily and uniformly occur at the software level.

2. An Example

We began our work by examining a candidate library component. The details of this example, and what we did with it, are described in [1]. The software component solves a structural engineering layout problem of how to space lightening holes in a panel. The component was originally part of an application that designs lay-up mandrels, which are tools that are used in the manufacturing process for composite skin panels. At first glance, the software component appears to solve the following one dimensional panel layout task (this was pulled from a comment in the header of this component).

Given a length of a panel, a minimal separation distance between holes in the panel, a minimal separation distance between the end holes and the ends of the panel, and a minimum and maximum width for holes, determine the number of holes, and their width, that can be placed in a panel. This software component solves a specific design task that is part of a broader design task. Prior to the invocation of this function, a structural engineer has determined a minimum spacing necessary to assure structural integrity of the panel.

Upon closer inspection of the software, one realizes that the software actually minimizes the number of holes subject to the constraints specified by the input parameter values. The original set of constraints defines a space of feasible solutions. Given a set of parameter values for the inputs, there may be more than one solution to picking the number of holes and their width so that the constraints are satisfied. So, the software documentation is incomplete.
However, going beyond this, one is inclined to ask “why did the programmer choose to minimize the number of holes?” Is there an implicit cost function defined over the feasible solutions to the original set of constraints? If so, what is it? Presumably, this is all part of the engineering design rationale that went into coming up with the (not fully stated) specification for the software component in the first place. If we were to use this component to design a panel that was to fly on an airplane, the panel would be structurally sound, but not necessarily of optimal cost (e.g., not making the best trade-off between manufacturing cost and overall weight of the panel).

3. Our Approach

Rather than put this incompletely documented software component into a reuse library, we seek to explicate the engineering design rationale and tie it directly to the software. For this purpose, we have used the Specware™ system [2,3] to first document the engineering design rationale leading to the software component specification, and then generate the software that provably implements the specification (which in turn requires documenting software design rationale).

Specware™ allows specifications to be composed in a very modular fashion (using the colimit construction from category theory). In our example, we were able to generate a specification for basic structural parts by taking a colimit of a diagram which related specifications for basic physical properties, material properties, and geometry. A specification for stiffened panels was derived by taking the colimit of another diagram, this time relating basic structural parts to panel parts and manufactured parts. This specification was then imported into another which added manufacturing properties that are specific to stiffened panels (it is here we finally state the (originally implicit) cost function). From this specification, and another describing basic optimization problems, we are able to formally state the panel layout problem. This specification was then refined into Lisp code.

If we place requirement specifications (at their various levels of abstraction, as described in the previous paragraph) and software derivations in a repository, we can reuse them to derive similar engineering software. For example, suppose we now wish to design stiffened panels for some part of the airplane itself. By plugging in specifications of the new panel manufacturing costs, and specifications of the trade-off between those costs and the panel weight, we should be able to generate software to design these new panels. In this way, we reuse knowledge at the appropriate level of abstraction, and not solely at the software level.

4. Challenges

There are various challenges in the use of this technology. We are currently working to apply this technology to larger engineering examples, and are thus checking the scalability of the basic technology. We suspect that current theorem proving technology may limit the size of problems we can handle. Another issue that needs to be addressed is the usability of Specware™. Currently, an understanding of category theory is necessary to use the tool. This is probably not acceptable if we are to place this tool in the hands of engineers. Another capability is needed in Specware™ - the ability to semi-automatically manage and propagate changes in the specifications leading to software components.

Perhaps the biggest challenge is in the substantial up-front costs associated with formally stating the engineering design rationale underlying software systems. It is much easier to just write some engineering software system. However, if we are willing to look at the long term, and amortize the up-front costs over a five, ten, or fifteen year time period, then those costs will more than make up for themselves. Software maintenance is extremely expensive. Underlying the challenges before us are the many, difficult, non-technical, management issues that must be addressed. For example, how can we change our employee reward mechanisms to reward people for performing more work at earlier design stages, with the aim of an even greater pay off in the future? This is a crucial issue. Another issue is the following: are we willing to allow more flexibility in our schedules to anticipate future use of these endeavours? What all these issues boil down to is a longer term view of our business enterprise.

5. Bibliography

