Advances in Cognitive Systems 4 (2016) 151–168 Submitted 8/2015; published 6/2016

Functional Model Simulation for Evaluating Design Concepts

Bryan Wiltgen BRYAN.WILTGEN@GATECH.EDU
Ashok K. Goel GOEL@GATECH.EDU
Design & Intelligence Laboratory, School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia 30332 USA

Abstract

Evaluation is a key task in design, and a major goal in research on computational design is to develop techniques for evaluating design concepts throughout the design process, starting as early as possible. Conceptual design in engineering is abstracted as a function-to-structure mapping and engages the use of functional models of design candidates. This suggests functional model simulation as a method for early evaluation of these alternatives. We describe a computational technique that evaluates such candidates in the conceptual phase through simulation of hierarchically organized Structure-Behavior-Function models. We demonstrate the capabilities of our technique for evaluation in biologically inspired system design that uses biological analogues to address design problems.

1. Introduction

Design is a fundamentally iterative process of generation, evaluation and redesign (Chandrasekaran, 1990; Dym & Brown, 2012; French, 1985). This is because design problems often address large and complex systems, designers are sometimes encouraged to be creative, initial design concepts often fail, and the cost of failure for actual designs can be huge. Indeed, evaluation, failure, and iteration are so prevalent in practice that "fail early, fail often" has emerged as a mantra in many a design community. Early and frequent evaluation of ideas can help expose the structure and the constraints of the design problem space, focus the designer's attention to more productive lines of search and exploration, and help reframe and reformulate the problem.

Computational design thus aims to develop techniques for evaluating designs throughout the design process as one of its major goals, and it seeks to do so as early in the process as possible. Indeed, computational design research has built many methods for evaluating design concepts, ranging from design critiquing to geometric modeling to numerical simulation to virtual and physical prototyping. However, most of these evaluation methods are useful only relatively late in the process, after conceptual design has been completed. The issue thus becomes how to evaluate system designs in the conceptual phase itself.

Conceptual design is typically abstracted as a function-to-structure mapping and therefore engages the use of functional models of design concepts (Hubka & Eder, 1988; Pahl et al., 2007). Thus, simulation of functional models offers one strategy for evaluating design concepts but has not yet received much attention in the literature. From a cognitive systems perspective, the

Figure 1. The bullet shaped nose of the Shinkansen train inspired in part by the kingfisher's beak. (Adapted from The Biomimicry Institute's Ask Nature, www.asknature.org.)

question now becomes what kinds of knowledge, and what forms of knowledge representation and organization, may support functional model simulation of design concepts? We posit that functional model simulation requires knowledge of several kinds, including the functions of the design, the structure of the design, and the causal behaviors that compose the functions of the design's components into the function of the design as a whole. In particular, we hypothesize that Structure-Behavior-Function models (SBF for short; Goel, 2013; Goel, Rubager, & Vattam, 2009) capture these kinds of knowledge, thus enabling functional model simulation to evaluate design concepts. In this paper, we present a computational technique (called SBFCalc) that evaluates such concepts through simulation of hierarchically-organized SBF models. We demonstrate the capabilities of our technique in the context of using biological analogues to address system design problems.

2. Biologically Inspired Design

To contextualize our research problem, let us consider the redesign of the Japanese Shinkansen trains in the 1990s described by McKeag (2012) and analyzed by Hoeller (2013). The problem entailed redesign of high-speed Shinkansen trains, which succeeded in part through analogy to biological systems. The goal was to alter the Shinkansen 300 train to achieve faster speeds. The Japanese railway engineers' initial redesign achieved this goal, but the new version produced too much noise at the higher speeds because of ground vibrations, aerodynamic noise, and sonic booms when it entered tunnels. The designers then used biological analogies to further redesign the train. To reduce sonic booms, they took inspiration from the beak of the kingfisher bird, which helped them design a new nose for the train; Figure 1 illustrates this biological analogy. To reduce noise from turbulence, designers took inspiration from the fimbriae on owl wings and added a small vortex generator to each pantograph on the train; Figure 2 depicts this biological analogy. Suppose that the Japanese railway engineers had created functional models of these design concepts (as, say, in Hubka & Eder, 1988; Pahl et al., 2007) to check if the new design would achieve the functions desired, if it would result in undesired behaviors, or if it had other errors. How might designers verify their proposed conceptual solutions? This example illustrates our research problem.

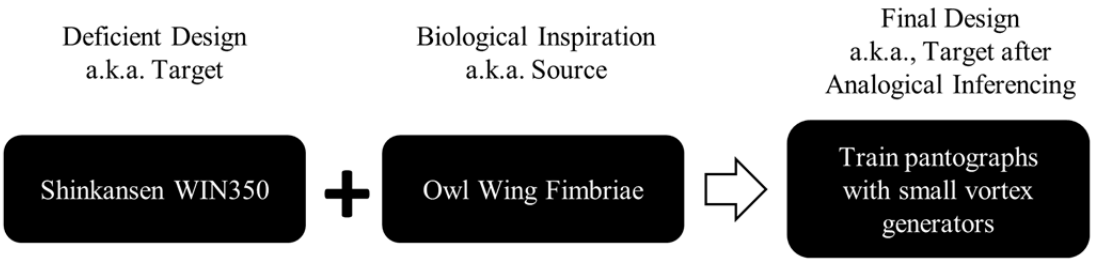

Figure 2. The analogy to owl wing fimbriae aided the redesign of the train pantographs.

The redesign of the Japanese Shinkansen trains in the 1990s is a well-known example of biologically inspired design (Baumeister et al., 2012; Benyus, 1997; French, 1994; Vincent & Mann, 2002). The conceptual phase here entails analogies in which the target problems come from design domains and the source analogues come from biology. Historically, this paradigm has been a source of design creativity and innovation. Its recent transformation into a design movement has been driven in large part by the need for environmentally sustainable designs. Goel, McAdams and Stone (2014) provide a compilation of recent progress on computational theories, techniques, and tools for biologically inspired design.

Analogical reasoning is also a common method of conceptual design in general (Goel, 1997). Although we developed our computational technique for evaluating design concepts through functional model simulation in the context of biologically inspired design, the technique is potentially applicable to general analogical design. Suppose that a designer uses an analogy to address a given problem in systems design, and that, after proposing a conceptual design, she wants to verify it. Our computational technique can help the designer verify the proposed conceptual design by simulating her functional model, comparing its simulation results to that model, and presenting its evaluation for inspection by the designer.

3. Structure-Behavior-Function Modeling

An SBF model of a system contains three submodels. The function submodel specifies functions, each of which describe the intended or perceived purposes of the system. The behavior submodel specifies behaviors, each of which describes the internal causal mechanisms by which a function is achieved, and the structure submodel specifies the physical components, substances, and connections between the components that give rise to the behaviors. Here we describe only the parts of the function and behavior models relevant to functional model simulation.

A function model is composed of one or more functions, each of which specifies (a) a name that uniquely identifies it, (b) a "provides" condition that defines values of component and substance attributes in the world that must be true at the completion of the function, and (c) a pointer to a behavior that provides an implementation of that function.

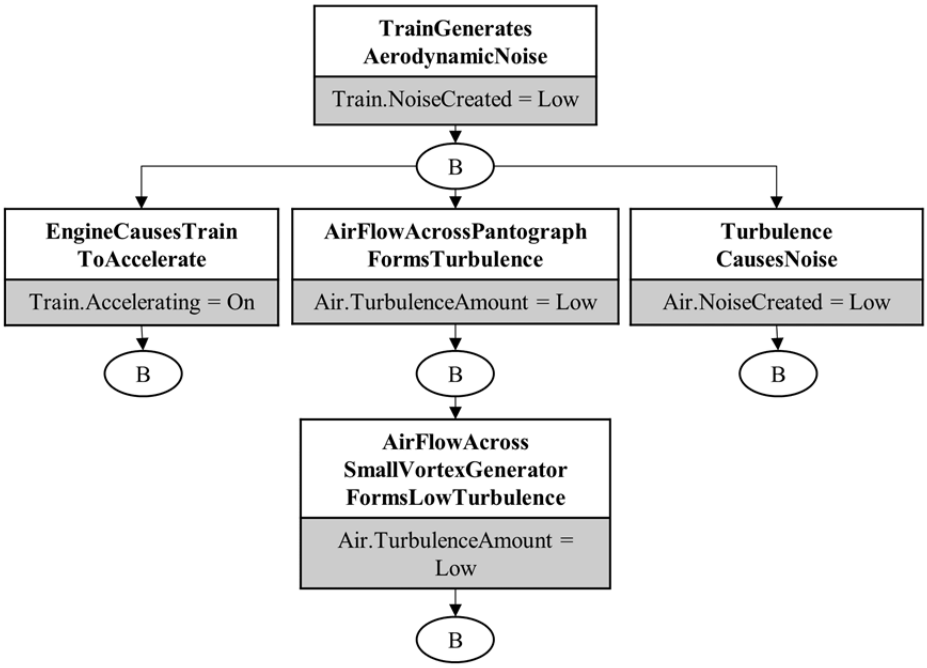

Figure 3. Decomposition of the function for reducing aerodynamic noise of the Shinkansen train. Each box represents a function and specifies its name (top half) and the "provides" condition (bottom half). The provides condition specifies the state of the world as a result of the function. The circled B represents a behavior. Each function indexes the behavior responsible for achieving it, and that behavior specifies its decomposition into subfunctions.

A behavior model consists of one or more behaviors composed of states and transitions. A behavioral state is specified as a set of component and substance attributes and their values. This may be a Start state (from whence the behavior begins), a Stop state (where the behavior ends), or an Intermediate one. A transition between two states describes a transformation from the Before state (where the transition begins) to the After state (where the transition ends). A transition is annotated with zero or more explanations, which specify why or how the Before state became the After state.

4. Illustrative Example: SBF Model of the Shinkansen Train

Figure 3 illustrates an SBF model of the conceptual design for the Shinkansen train with a small vortex generator attached to its pantograph (a mechanical linkage). As the train moves, air flows over the small vortex generator on the pantograph. This interaction creates low turbulence, which in turn reduces the noise made by the train to low noise.

The top half of each box in Figure 3 identifies the name of the function and the bottom half identifies its "provides" condition, which is the (possibly partial) state of the world that explicitly results from the function. For example, the function TrainGeneratesAerodnyamicNoise declares that the NoiseCreated by the Train will be Low when it is done, and the function Engine-

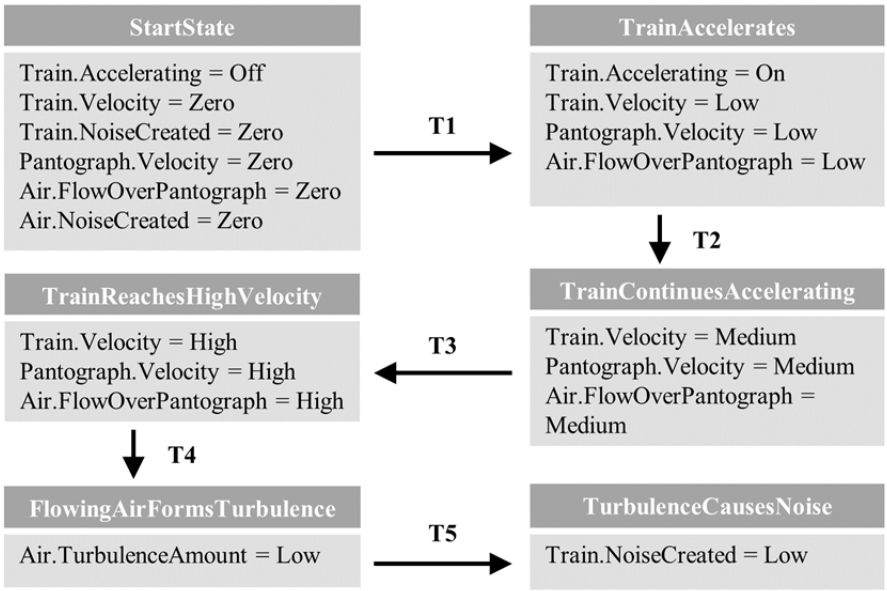

Figure 4. The behavior that implements the TrainGeneratesAerodynamicNoise function in the Shinkansen Train model. Boxes represent states and arrows represent transitions between states. Table 1 provides the explanations on the transitions.

CausesTrainToAccelerate declares that the Accelerating state of the Train will be On when it has completed. The B in a circle represents a behavior, and an arrow extending from a function to a circled B denotes a pointer to the behavior that achieves that function.

The function TrainGeneratesAerodynamicNoise is the top-level function. Its behavior describes situations in which the train is going from standstill to high velocity and producing low noise. EngineCausesTrainToAccelerate is a subfunction of this top-level function, and its behavior describes the result of the train's throttle being turned on, which in turn causes the train to accelerate and increase its velocity. AirFlowAcrossPantographFormsTurbulence is another subfunction at the same level. Its behavior describes air flowing towards the pantograph and small vortex generator, which causes low turbulence. AirFlowAcrossSmallVortexGeneratorFormsLowTurbulence is a subfunction of AirFlowAcrossPantographFormsTurbulence, and its behavior describes how a low amount of turbulence results from the small vortex generator producing a small (represented by the Low value) vortex size. TurbulenceCausesNoise is the final subfunction of the top-level function, and its behavior describes how low noise is created given low turbulence.

Figure 4 depicts the behavior that implements the TrainGeneratesAerodynamicNoise function. In this figure, boxes represent states with the name of the state in the top and the condition in the bottom. For example, this figure says that the TurbulenceAmount of Air is Low in the state FlowingAirFormsTurbulence. StartState is the start state for this behavior, and TurbulenceCausesNoise is the stop state. Arrows represent transitions. For simplicity, we have not included the explanatory annotations on the transitions in the figure. Table 1 lists the causal explanations that annotate these transitions.

5. Functional Model Simulation for Design Concept Verification

As mentioned in the introduction, our hypothesis is that an SBF model captures the kinds of knowledge needed for verifying a design concept. In particular, as Section 4 illustrates, the SBF model captures five kinds of knowledge needed for verification by functional model simulation: (1) functions of the design that must be verified, (2) causal behaviors intended to accomplish the functions, (3) explanations for each state transition in a causal behavior, (4) the design structure, where the components annotate state transitions in the behaviors, and (5) a recursive function-behavior-function decomposition. Figure 3 presents an illustration. We now describe our approach to using these knowledge contents for design concept verification.

We implemented our computational technique, SBFCalc, as a Java program that takes as input an SBF model of a candidate design concept. SBFCalc simulates the model, replacing the values of attributes in the specification of the behavioral states with attributes and values it infers through simulation. It then evaluates the derived behaviors with respect to the desired functions and the specified behaviors.

5.1 Evaluating the Behavior Model

When evaluating a model, SBFCalc must verify the behaviors of the model because they are how the functions are achieved, and thus errors in behaviors may reflect misconceptions or modeling mistakes about how the system works. Our computational technique takes a two-step process to evaluate each behavior. First, it simulates the behavior: given the attributes' values in the start

Table 1. Causal explanations for the transitions in Figure 4.

Transition Identifier	Explanations for that Transition
T1	• Function: EngineCausesTrainToAccelerate • Equation E1 "qual: Pantograph.Velocity is directly proportional to the qualitative expression Train.Velocity:After - Train.Velocity:Before" • Equation E2 "qual: Air.FlowOverPantograph is directly proportional to the qualitative expression Pantograph.Velocity:After - Pantograph.Velocity:Before"
T2	• Equation: E1 "qual: Train.Velocity is directly proportional to the qualitative expression Train.Accelerating:After " • Equation E2 "qual: Pantograph.Velocity is directly proportional to the qualitative expression Train.Velocity:After - Train.Velocity:Before" • Equation E3 "qual: Air.FlowOverPantograph is directly proportional to the qualitative expression Pantograph.Velocity:After - Pantograph.Velocity:Before"
T3	The same explanations as for T2
T4	• Function AirFlowAcrossPantographFormsTurbulence
T5	• Function TurbulenceCausesNoise • Equation E1 "qual: Train.NoiseCreated is directly proportional to the qualitative expression Air.NoiseCreated:After - Air.NoiseCreated:Before"

state, it infers the attributes and values for the subsequent states. Second, it compares the inferred states with the originally specified states: a difference between an inferred state and the equivalent specified state signals a potential problem.

To simulate a behavior, SBFCalc begins at the start state of the behavior, traverses the outgoing transition, and infers the attributes and values of the subsequent states based the explanatory annotations on the transitions. SBFCalc recursively repeats this process for each subsequent state until it runs out of transitions to traverse. The annotations on the transitions have several types (Goel, Rugaber, & Vattam, 2009). Below we describe how SBFCalc reasons about two kinds of annotations: functions (as indicated in Figure 3) and equations. We also describe implicit value forwarding, a technique used to infer the value of an attribute in the absence of any annotations that affect it.

5.1.1 Reasoning about Quantitative Equations

Recall that a transition in a behavior describes the transformation of one state (the Before state) to another state (the After state), and a transition is annotated with zero or more explanations. Each explanation clarifies why or how the system moves from the Before state to the After state. Equation explanations can be either quantitative or qualitative, which are denoted by the prefixes "quant:" and "qual:", respectively.

A quantitative equation says that an attribute's value in the After state will be equal to a mathematical expression in which variables denote component or substance attributes that resolve to numerical values. The syntax of a quantitative equation is:

```
quant: <Attribute> = <Expression>
```

Here, "quant:" signifies that this is a quantitative equation, <Attribute> is an attribute of a component or substance to which we are assigning a value (e.g., Box.Weight, where Box is a component and Weight is one of its attributes), and <Expression> refers to a mathematical expression that may contain attributes as variables. Each attribute in <Expression> has an additional :Before or :After tag, indicating if the value should be taken from the attribute's value in the Before state or the After state, respectively. For example, an <Expression> could be `Box.NumberOfOranges:After * Orange.Weight:Before`.

To solve an <Expression>, all attributes within the expression must resolve to numerical values. SBFCalc checks to see if it has a value for all the attributes within an <Expression>. If there are any After attributes within the expression for which there is no value, it may need to solve another equation in the transition before it can resolve the After attribute in that expression. For example, consider a hypothetical situation in which there are two equation explanations on the same transition:

```
quant: Box.Weight = Box.NumberOfOranges:After * Orange.Weight:Before
quant: Box.NumberOfOranges = Box.NumberOfOranges:Before + 1
```

To solve the <Expression> in the first equation, SBFCalc must know the value for Box.NumberOfOranges:After, which requires solving the second equation. It tackles such situations in two ways. First, it initially reasons about function explanations and implicit value

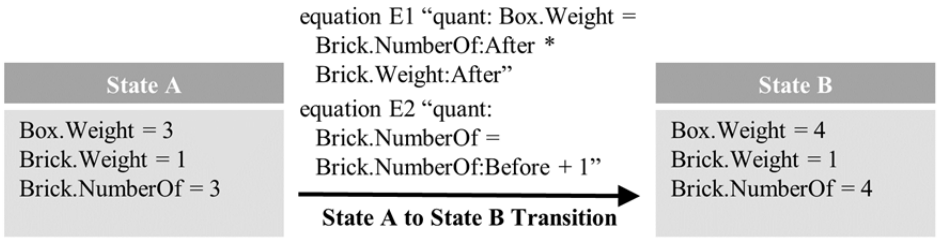

Figure 5. Hypothetical example of reasoning using quantitative equations.

forwarding so that it knows as many After values as possible before reasoning about equations. Second, it takes an iterative approach by solving at most one equation at a time, starting with an equation with no unresolved attributes. If no equations are solvable, SBFCalc fails the task and exits. An <Expression> might also be unsolvable because it contains Before attributes for which SBFCalc does not know a value. This is also covered by the iterative approach because all attributes in an <Expression> must be resolvable for it to be solvable.

Figure 5 depicts a hypothetical example of reasoning with quantitative equation. This example also uses implicit value forwarding. The Before and After states describe the change in weight of a box due to an increase in the number of bricks in the box. The transition between these two states is annotated with two quantitative equations. The first equation, E1, describes how to calculate the weight of the box. The second equation, E2, describes how an additional brick is being added from the Before state to the After state.

5.1.2 Reasoning about Qualitative Equation Explanations

A qualitative equation specifies whether an attribute's value, defined as a quantity in a predefined quantity space, in the After state is either directly or inversely proportional to a qualitative or quantitative expression. SBFCalc uses two predefined quantity spaces, one with quantities Zero, Low, Medium, High, and Maximum and the other with quantities Off and On. A qualitative expression is one in which all attributes resolve to values in the two quantity spaces. The syntax of a qualitative equation is:

```
qual: <Attribute> is (directly | inversely) proportional to the
      (quantitative | qualitative) expression <Expression>
```

Here, "qual:" signifies a qualitative equation, and <Attribute> means the same as in quantitative equations, except that its value now is a quantity in a quantity space rather than a numerical value. To solve a qualitative expression, SBFCalc first replaces all the attributes with their respective values, using the same procedure for deciding whether the <Expression> is solvable as with quantitative expressions. Next, it replaces the qualitative values with their numerical equivalents and then solves the <Expression> as if it were a quantitative expression. SBFCalc then inspects the result to see if it is either less than, equal to, or greater than zero. If the specified relationship is directly proportional, the value of <Attribute> will increase if the result of <Expression> was

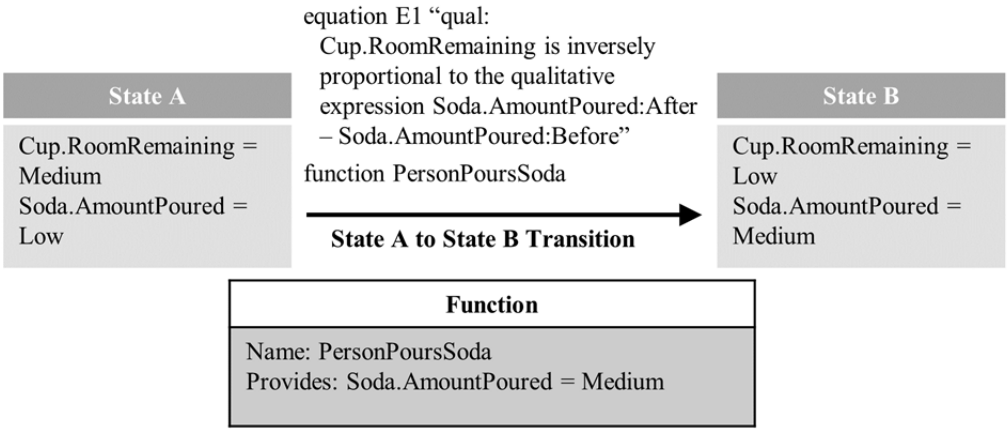

Figure 6. Hypothetical example for reasoning with a qualitative equation and a function explanation.

greater than zero, stay the same if the result was equal to zero, or decrease if the result was less than zero. If the relationship is inversely proportional, the increase and decrease conditions are reversed. However, the change in <Attribute> is limited in that an attribute's value can never increase beyond the maximum quantity in the quantity space, nor can it ever decrease below the minimum quantity in the quantity space.

Figure 6 depicts a hypothetical example using both a qualitative equation and a function explanation. The Before and After state pair of this example describes how the room remaining in a cup decreases as the amount of soda poured into the cup increases. The model annotates the transition between the two states with both a qualitative equation explanation and a function explanation. Reasoning about the function explanation determines that the After value of Soda.AmountPoured is Medium. The qualitative equation E1 specifies that the RoomRemaining attribute of Cup is inversely proportional to the change in value of Soda.AmountPoured. Thus, if AmountPoured increases between the Before and After states, then RoomRemaining will decrease and vice versa, with the limitation that they cannot increase or decrease beyond the bounds of their quantity spaces.

5.1.3 Reasoning about Function Explanations

Figure 3 illustrates the centrality of functional decomposition to SBF modeling. The functional decomposition allows a partitioning of the large and complex verification problem into a series of smaller and simpler verification problems. Figure 3 also illustrates that in SBF modeling, behaviors mediate functional decomposition: a behavior specifies how a function is decomposed into subfunctions, or conversely, how the functions of components are composed into the functions of systems as a whole.

A function explanation on a transition in a behavior indicates that a subfunction is responsible for the change in some or all of the attributes' values from the Before state to the After state. To address a function explanation, SBFCalc runs a simulation of the behavior linked to the function

in the explanation. The system then sets the attributes and values based on that simulation's output.

Figure 6 depicts a hypothetical example of reasoning about a function explanation. When SBFCalc encounters this example, it simulates the behavior pointed to by the function PersonPoursSoda and uses the simulation's output to infer the After state attributes and values. In this case, the behavior pointed to by the function PersonPoursSoda has the same output as its "provides" condition, so the system infers that the AmountPoured attribute of Soda in the After state is equal to Medium.

5.1.4 Reasoning with Implicit Value Forwarding

In addition to reasoning about equation and function explanations, SBFCalc uses a technique that we call *implicit value forwarding*. In a given Before and After state pair, the system may not always be able to infer the After state values for all the attributes in the Before state. When this is the case, it assumes that the values of those attributes remain the same. Thus, it will set the After state's value for that attribute to be the same as the Before state's value.

Figure 7 depicts a hypothetical example without (the top half of the figure) and with (the bottom half of the figure) implicit value forwarding. Without this mechanism, the value for the WaterFlowing attribute of Hose is missing in State B—the After state—because SBFCalc could not infer it from any explanation on the transition. With implicit value forwarding, the system still cannot infer the value from any explanation, but it sets Hose.WaterFlow = On for State B, forwarding it from State A, the Before state.

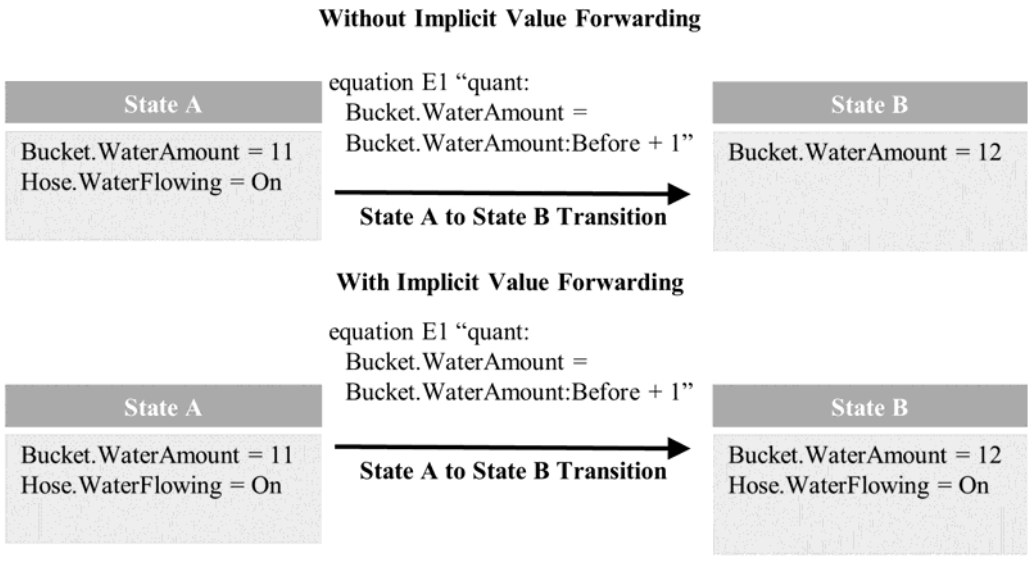

Figure 7. Hypothetical example *without* (top half) and *with* (bottom half) implicit value forwarding.

5.1.5 Tying Things Together: Inferring an After State

The previous sections looked at how SBFCalc reasons about/with quantitative and qualitative equations, function explanations, and implicit value forwarding to determine values in the After state. We will briefly describe how, given a Before state, an After state, and a set of explanations, the system combines these various techniques to infer the values for the After state.

SBFCalc builds a map that connects Before and After state attributes to their values in those states. This map is only applied to the After state at the end of reasoning about the Before-After state pair. The system first stores the Before attribute and values pairs in the map, then reasons over any function explanations on the transition, storing the After state attribute and value pairs that it infers. Next, it uses implicit value forwarding to store After state attributes and values that (a) have not already been set by the function explanation reasoning and (b) will not be set by the equation explanation reasoning, as determined by inspecting the equation annotations on the transition. After this, it processes the qualitative and quantitative equations (if any exist) and stores the results in the map. Finally, it uses the map to set the After state attributes and values.

5.2 Evaluating the Function Model

The function model is comprised of one or more functions. Verification of this model ensures that the proposed conceptual design actually delivers the functions desired of it. To this end, our computational technique determines the extent to which the behavior responsible for a function actually achieves it. After it has completed its behavior simulations and evaluations, SBFCalc evaluates all the functions. Conceptually, the "provides" condition of a function specifies the state of the world that must be true at the execution of the function, and the behavior it points to should implement the function. Therefore, the Stop state of a behavior should reflect a world state that is consistent with the "provides" condition. For a given function, SBFCalc compares the attribute and value pairs from this condition with the attribute and value pairs that result from the simulated behavior, looking for contradictions and thereby determining whether the behavior will achieve the desired function.

6. Evaluation of Model Simulation for Design Verification

We have evaluated SBFCalc on a small set of verification cases in biologically inspired design. In this section, we first describe of the case of the Shinkansen Train in some detail, and then briefly summarize two other cases on which we have tested the computational technique.

6.1 The Case of the Shinkanesen Train

Although we ran SBFCalc against the entire Shinkansen Train model (shown in Figures 3 through 5), for brevity we will only report here the verification results related to the top-level TrainGeneratesAerodynamicNoise function (Figure 3) and its associated behavior. Figure 8 depicts the results of function verification. Attribute and value pairs with a + prefix were in the behavior output but not in the function's "provides" condition. Only one attribute and value pair, Train.NoiseCreated = Low, is missing this prefix, as it was in both the behavior's output and the function's "provides" condition. As can be seen, SBFCalc deems the output of the inferred behavior to be completely compatible with the target function. Note the many extra attribute-

Compatible Properties and Values	Incompatible Properties and Values
Train.NoiseCreated = Low +Train.Accelerating = On +Pantograph.Velocity = High +Air.FlowOverPantograph = High +Air.TurbulenceAmount = Low +Pantograph.Size = High +SmallVortexGenerator.VortexSizeProduced = Low +Air.NoiseCreated = Low +Air.FlowDirection = Towards Small Vortex Generator +Train.EngineThrottle = On +Train.Velocity = High	[none]

Figure 8. Results of evaluating the TrainGeneratesAerodynamicNoise function of the Shinkansen Train model.

value pairs produced by the behavior, which suggests a more complex world state than examining the function's "provides" condition alone.

We found that the attribute-value pairs that appear in both the simulated and original behavioral states (see Figure 4) were identical. However, we also found that the simulated behavioral states had many other attribute-value pairs. Figure 9 illustrates these additional pairs for each behavioral state of Figure 4. We have prefixed these additional attribute-value pairs with a +.

As noted above, the behavioral states simulated by SBFCalc for this function's behavior agreed with all the state conditions in the original in that no attributes had different values. This is a positive outcome because it shows that the functional model of the train correctly specified its behaviors. In the results for another behavior, SBFCalc found a different value for one attribute, showing that it can find differences if they occur. Second, our computational technique identified many additional attribute-value pairs in the simulated behavior that likely came from implicit value forwarding and function explanation reasoning. Although these do reflect differences between the simulated and original behaviors, we are considering how to handle the new attribute-value pairs because they do not represent contradictions in the model, and they may have been deliberately left out by the modeler.

The successful verification of the conceptual design of the Shinkansen train provides evidence in support of our hypothesis that an SBF model captures the kinds of knowledge useful for verifying the design concept. Several aspects of SBF models for enabling functional model simulation and design concept verification are especially noteworthy: Knowledge of the structure of the design is distributed through annotations on the state transitions in the behaviors; this enables functional model simulation to take design structure into account. Each state transition in a behavior is annotated by explanations on the transition; this enables behavioral simulation. Function is broken down through a recursive function-behavior-function decomposition; this ensures that the behavior corresponding to any function in the hierarchy is small and simple and that it can be simulated easily and efficiently.

6.2 Additional Case Studies

In addition to the Shinkansen train design, we have tested our computational technique on two additional examples. For the sake of brevity, we provide here only short descriptions of them as evidence for the generality of our approach. In the first study, we intentionally allowed an SBFCalc-identified error to persist to illustrate its ability to catch incorrectness in models.

In the first case, we modeled a medical patch inspired by the spiny headed worm. We derived this from McKeag (2015), wherein he describes efforts to design biologically inspired attachment (or attachment-removal) mechanisms in the domain of invasive surgery. The patch incorporates conical tips on a needle array. When the patch is inserted, the tips swell, allowing the patch to adhere to the location. Our model of this device contains a superfunction (PatchHoldsOn) and a subfunction (TipsSwell), each with its own behavior. PatchHoldOn's behavior describes the patch being inserted and obtaining a 3.5 adhesion strength relative to staples. This behavior contains three states and two transitions. TipsSwell's behavior, which describes the tips swelling, contains two states and one transition.

When we ran SBFCalc on the SBF functional model of the medical patch, the simulated behaviors agreed with the behavioral models, except that both included additional state attribute-value pairs due to implicit value forwarding and, in the behavior for the superfunction, function

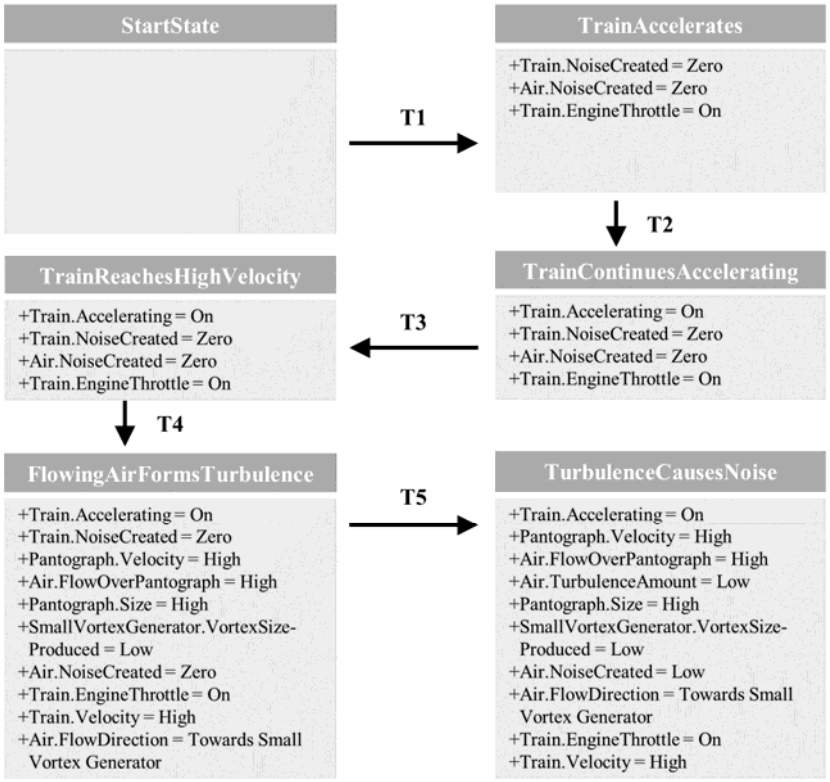

Figure 9. The results of evaluating the behavior that achieves the TrainGeneratesAerodynamicNoise function of the Shinkansen Train model. Boxes represent states and arrows represent transitions between states.

explanation reasoning. This is similar to the results we presented earlier. SBFCalc found the PatchHoldsOn function to be compatible with its behavior's results. However, SBFCalc found the TipsSwell function to be incompatible with its behavior, for that function's "provides" condition states that NeedleArray.TipSize should be Large when instead the behavior resulted in it being Medium[1]. This provides evidence that our approach to functional model simulation can both verify that a conceptual design is correct and can also catch errors in the design concept.

Until now, we have presented SBFCalc as a technology for verifying design concepts. In a second case study, we also modeled the source analogue for the small vortex generator that appeared in our train example. Unlike the earlier examples, this illustrates the usefulness of SBFCalc for validating source analogues. The SBF model describes how the fimbriae (serrations) on the owl's primary wing feathers create micro-turbulences that let the owl fly quietly. This model, derived from McKeag (2012) and Hoeller (2013), contains a superfunction (OwlFliesSilently) and a subfunction (FimbriaeBreakDownAir), each with its own behavior. OwlFliesSilently's behavior contains three states and two transitions that describes the owl moving and creating low noise. FimbriaeBreakDownAir's behavior contains two states and one transition that describe how the air moving past the owl generates micro-turbulences.

When we ran SBFCalc on this model of an owl's flight, the simulated behaviors agreed completely with the specified ones, with the exception that, as with our prior examples, it identified additional attribute-value pairs through implicit value forwarding and, in the behavior for the superfunction, function explanation reasoning. It also found both functions to be compatible with their behaviors. These findings illustrate that SBFCalc can help verify source analogues in analogical design in addition to verifying candidate designs. The case studies further support our original hypothesis. In particular, they indicate that SBF models capture the kinds of knowledge needed to verify the correctness of conceptual designs, identify errors in design concepts, and verify source analogues in biologically inspired design.

7. Related Research

This work builds on several lines of research in conceptual design, functional modeling, qualitative simulation, analogical design, and biologically inspired design. The process of engineering design consists of several phases, with problem formulation and conceptual design being the earliest phases (Dym & Brown, 2012; French, 1985; Hubka & Eder, 1988; Pahl et al., 2007). The task of conceptual design takes a desired function as the input; the goal is to generate a structure that delivers this function. Thus, the task is abstracted as a function-to-structure mapping. This is why languages for formulating design problems typically specify the desired functions, the operating environment, the performance criteria, and constraints on structure (Helms & Goel, 2014; MacLellan et al., 2013).

This abstraction has led to the development of several functional modeling schemes (Chandrasekaran, Goel, & Iwasaki, 1993; Gero & Kannengiesser, 2004; Kitamura et al., 2004; Rasmussen, 1985; Sembugamoorthy & Chandrasekaran, 1986; Umeda & Tomiyama, 1997). According to Simon (1996), a functional model of a design provides a functional decomposition of the design and a functional explanation of how the structure of the design delivers the desired functions. Functional models typically use behavior as an intermediate abstraction to explain how the structure achieves the functions. Our technique works on Structure-Behavior-Function (SBF)

[1] This is the error that we allowed to persist in order to demonstrate error detection.

models, in which a behavior is a causal process that composes the functions of sub-systems into the functions of the system as a whole (Goel, Rugaber, & Vattam, 2009).

Cognitive systems research on analogical reasoning has a long history (Falkenhainer, Forbus, & Gentner 1989; Hofstadter, 1995; Holyoak & Thagard 1996). Regarding computational approaches specifically for analogy evaluation, Falkenhainer (1987) evaluated an analogy by simulating a qualitative model of the inferred concept and comparing the results of the simulation with observations. Falkenhainer leverages observations of actual data for verification, while our work instead compares the results of simulation to a conceptual model. In addition, we use our computational technique to verify the source analogue.

Other researchers have also pursued verification of functional models using qualitative simulation. For example, Price (1998) uses "functional labels" to analyze the results of a simulation derived from a component model. Klenk et al.'s (2012) work combines qualitative simulation with Modelica models of designs, verifying functional requirements against simulation results drawn from topologies. D'Amelio et al. (2011) use qualitative reasoning to simulate Function-Behavior-State models (Umeda & Tomiyama, 1997) in order to detect anomalous states due to redesign or module combinations. Iwasaki et al. (1995) verify functions by simulating a component model with additional behavioral pieces and comparing resultant trajectories against a function description that includes behavioral descriptions.

Our computational technique differs from these methods in two significant ways. First, it preserves the hierarchical nature of SBF models by independently performing a simulation and verification pass for each function-behavior pairing in a model. Other models may have abstraction built in (e.g., Price's (1998) "[e]ncapsulate complex behavior within a component"), but their simulations and thus verifications produce results for the entire model in one chunk. Our technique enables individual verifications to stay focused on one function or one behavior at a time even if the functional decomposition is very large, which should make results easier to handle. Additionally, we believe our work is a step towards the "[m]ulti-level modelling" mentioned by Price et al. (2006) as necessary to achieve future targets for qualitative reasoning.

Second, our technique leverages the same causal process representation (state diagrams) for both reasoning and representing simulation results, whereas other work uses representations for reasoning (e.g., component models or model fragments) that differ from the state-based representations in their simulation results. Our approach allows a natural comparison between the state diagrams produced by the simulation and those made by the modeler. In addition, from the perspective of SBF modeling, this also means that our technique does not require modelers to learn a new representation it since it leverages something that is already part of SBF models, although modelers will need to learn our equation syntax.

Finally, a sister project in our laboratory on scientific modeling has developed an interactive technique for verifying conceptual models of ecological phenomena through simulation using off-the-shelf simulation platforms (Joyner, Goel, & Papin, 2014). The work we have described here differs in that it evaluates design concepts through simulation of associated functional models in the context of system design.

8. Conclusions

As one of its major goals, research in computational design seeks to develop techniques for evaluating design concepts early in the conceptual phase. Engineering typically abstracts this task

as finding a function-to-structure mapping and engages the use of functional models of the design concepts. Simulation, in particular functional model simulation, is a potential method for early evaluation of design concepts. The question then becomes what kinds of knowledge, and what forms of knowledge representation and organization, may support functional model simulation of such concepts?

We posited that the knowledge captured by Structure-Behavior-Function models would enable functional model simulation for design concept verification. In particular, we showed evidence that SBF models capture several kinds of knowledge, described in section 5, that are useful for verifying design concepts.

Three aspects of SBF models are especially important for enabling qualitative simulation and design verification: (1) Knowledge of the structure of the design on state transition annotations in behaviors enables simulation to account for structure; (2) Explanations that annotate state transitions in a behavior enable behavioral simulation; and (3) The recursive function-behavior-function decomposition ensures that each function's behavior will be small and simple, enabling easy and efficient simulation. Regarding (3), SBF models decompose and organize the simulations of the conceptual design into simulations of smaller, simpler subsystems, and organize and abstract the simulations of the subsystems into simulation of the system as a whole.

We described a computational technique called SBFCalc that evaluates design concepts through simulation of their SBF models. This technique represents the main contribution of this work and rests at the intersection of conceptual design, functional modeling, and qualitative simulation. We demonstrated its capabilities for verifying design concepts in the context of biologically inspired system design using functional model simulation. In particular, the Shinkansen train case study shows that our technique can verify the correctness of conceptual design, the medical patch study indicates that it can identify errors in design concepts, and the owl flight study demonstrates that it can verify source analogues in biologically inspired design.

That said, our technique does have room to grow. First, we believe that a designer could use the same technique not only to verify proposed conceptual designs and source analogues but also to verify the functional models of deficient designs. Second, our computational technique should support more types of causal explanations so that it can interpret and usefully evaluate models written in a larger subset of the SBF language. Finally, our technique should better handle ambiguity in qualitative equations, which due to their abstractness, may have ambiguous results. For example, given a simulation result that says a value (e.g., Low) should increase, it may be unclear whether it should stay the same or change to Medium or High. To account for this kind of ambiguity in simulation, SBFCalc should produce an envisionment with multiple trajectories, with each one representing a possible configuration of values through the state space. Our system could then verify the given behavior against these projected trajectories. Considering trajectories in verification builds conceptually on work by Iwasaki et al. (1995) and Klenk et al. (2012), who generated them via simulation for reasoning about this task.

Acknowledgements

We thank Spencer Rugaber and Arvind Jagannathan for their contributions to the SBF editor. This paper is a significantly revised version of Wiltgen and Goel (2015), and has benefited from feedback from Pat Langley, Ken Forbus, and anonymous reviewers of the earlier paper.

References

Baumeister, D., Tocke, R., Dwyer, J., Ritter, S., & Benyus, J. (2012). *Biomimicry resource handbook*. Missoula, MT: Biomimicry 3.8.

Benyus, J. (1997). *Biomimicry: Innovation inspired by nature*. New York: William Morrow.

Boehm, B. W. (1984). Verifying and Validating Software Requirements and Design Specifications. *IEEE Software*, January, 75-88.

Chandrasekaran, B. (1990). Design problem solving: A task analysis. *AI Magazine, 11*, 59-71.

Chandrasekaran, B., Goel, A., & Iwasaki, Y. (1993). Functional representation as a basis for design rationale. *IEEE Computer*, *26*, 48–56.

D'Amelio, V., Chmarra, M. K., & Tomiyama, T. (2011). Early design interference detection based on qualitative physics. *Research in Engineering Design*, *22*, 223-243.

Dym, C., & Brown, D. (2012). *Engineering design: Representation and reasoning* (2nd ed.). New York: Cambridge University Press.

Falkenhainer, B. (1987). An examination of the third stage in the analogy process: Verification-based analogical learning. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 260–263). San Francisco: Morgan Kaufmann Publishers Inc.

Falkenhainer, B., Forbus, K., & Gentner, D. (1989). Structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, *41*, 1–63.

French, M. (1985). *Conceptual design for engineers* (2nd ed.). Berlin: Springer-Verlag.

French, M. (1994). *Invention and evolution: Design in nature and engineering* (2nd ed.). New York: Cambridge University Press.

Gero, J., & Kannengiesser, U. (2004). The situated function-behavior-structure framework. *Design Studies*, *25*, 373-391.

Goel, A. (1997). Design, analogy and creativity. *IEEE Intelligent Systems*, *12*, 62–70.

Goel, A. (2013). One thirty year long case study; Fifteen principles: Implications of an AI methodology for functional modeling. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *27*, 203-215.

Goel, A., McAdams, D., & Stone, R. (Eds.). (2014). *Biologically inspired design: Computational methods and tools*. London: Springer-Verlag.

Goel, A., Rugaber, S., & Vattam, S. (2009). Structure, behavior & function of complex systems: The structure-behavior-function modeling language. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *23*, 23–35.

Helms, M., & Goel, A. (2014). The four-box method: Problem formulation and analogy evaluation in biologically inspired design. *ASME Journal of Mechanical Design*, *136*, 1–12.

Hoeller, N. (2013). Tools: Structure-Behavior-Function and functional modeling. *Zygote Quarterly*, Spring, 150–167.

Hofstadter, D. (Ed.). (1995). *Fluid concepts & creative analogies: Computer models of the fundamental mechanisms of thought*. New York: Basic Books.

Holyoak, K., & Thagard, P. (1996). *Mental leaps: Analogy in creative thought*. Cambridge, MA: MIT Press.

Hubka, V., & Eder, E. (1988) *Theory of technical systems.* Berlin: Springer-Verlag.

Iwasaki, Y., Vescovi, M., Fikes, R., & Chandrasekaran, B. (1995). A causal functional representation language with behavior-based semantics. *Applied Artificial Intelligence, 9*, 5-31.

Joyner, D., Goel, A., & Papin, N. (2014). Intelligent generation of agent-based simulations from conceptual models. *Proceedings of the Eighteenth International Conference on Intelligent User Interfaces* (pp. 289-298). Haifa, Israel.

Kitamura, Y., Kashiwase, M., Fuse, M., & Mizoguchi, R. (2004). Deployment of an ontological framework for functional design knowledge. *Advanced Engineering Informatics*, *18*, 115–127.

Klenk, M., de Kleer, J., Bobrow, D., Yoon, S., Hanley, J., & Janssen, B. (2012). Guiding and verifying early design using qualitative simulation. *Proceedings of the ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2012*. Chicago, IL.

MacLellan, C., Langley, P., Shah, J., & Dinar, M. (2013). A conceptual aid for problem formulation in early conceptual design. *ASME Journal of Computing and Information Science in Engineering*, 13.

McKeag, T. (2012). Auspicious forms: Designing the Sanyo Shinkansen 500-Series bullet train. *Zygote Quarterly, 2*, 14-35.

McKeag, T. (2015). Case study: Sticky wicket: A search for an optimal adhesive for surgery. *Zygote Quarterly*, *12*, 18-41.

Pahl, G., Beitz, W., Feldhusen, J., & Grote, K. (2007). In K. Wallace & L. Blessing (Eds.), *Engineering design: A systematic approach* (3rd ed.). New York: Springer-Verlag.

Price, C. (1998). Function-directed electrical design analysis. *Artificial Intelligence in Engineering, 12*, 445-456.

Price, C., Trave´-Massuye`s, L., Milne, R., Ironi, L., Forbus, K., Bredeweg, B., Lee, M.H., Struss, P., Snooke, N., Lucas, P., Cavazza, M., & Coghill, G. (2006). Qualitative futures. *Knowledge Engineering Review, 21*, 317–334.

Rasmussen, J. (1985). The role of hierarchical knowledge representation in decision making and system management. *IEEE Transactions on Systems, Man, and Cybernetics*, *15*, 234–243.

Sembugamoorthy, V., & Chandrasekaran, B. (1986). Functional representation of devices and compilation of diagnostic problem-solving systems. In J. Kolodner & C. Riesbeck (Eds.), *Experience, Memory, and Learning*, 47–73. Mahwah, NJ: Erlbaum.

Simon, H. A. (1996). *Sciences of the artificial* (3rd ed.). Cambridge, MA: MIT Press.

Umeda, Y., & Tomiyama, T. (1997). Functional reasoning in design. *IEEE Intelligent Systems*, *12*, 42–48.

Vincent, J., & Mann, D. (2002). Systematic technology transfer from biology to engineering. *Philosophical Transactions of the Royal Society of London A, 360*, 159-173.

Wiltgen, B., & Goel, A. (2015). Evaluating design concepts through functional model simulation. *Proceedings of the Third Annual Conference on Advances in Cognitive Systems.* Atlanta, GA.