A Framework for Exploring Unifying Theories of Empirical Software Engineering

Dewayne E Perry
Center for Advanced Research in Software
Engineering (ARiSE)
The University of Texas at Austin

perry@mail.utexas.edu

Abstract

I believe that one of the reasons for the lack of rigor in software engineering compared to physical and behavioral sciences is that we have not given enough attention to the theories that underpin our work, both as software engineers and as software engineering researchers. To provide a step forward towards greater rigor, I have created a framework with which to explore theories of software engineering and software engineering research. This framework provides a simple theory modeling language and calculus to describe informally described theories and to generate the results of composing modeled theories.

To illustrate and evaluate this framework, I present my general theory about software engineering and then propose two simple theories, D and E as the basis for laying out a unified theoretical foundation for software engineering and software engineering research. Software Engineering consists of two logical parts: design and empirical evaluation (both terms used in their broadest senses). I propose theory D to as the theoretical basis for the design part, and theory E as the theoretical basis for empirical evaluation. These two theories are then composed in various ways to lay out a space (a taxonomy, or ontology if you will) for software engineering and software engineering research. Finally, I claim that software engineering and software engineering research (both fully integrated with empirical evaluations) are models (in the logical sense) for the atomic and composed theories. To further evaluate the framework, I provide examples of modeling (implicit) theories found in several software engineering (theory) papers.

The results of this research are 1) a scientific elegance in creating larger more complex theories out of simpler theories, 2) an elegant way of explaining the complexity of software engineering and software engineering research, and 3) a theory modeling language and calculus for composing the resulting theoretical models.

1. Introduction

The motivation for this research is twofold: 1) to establish a unifying foundation for software engineering, and 2) to establish the same rigorous empirical foundations for software engineering that we find in natural and behavioral sciences. As engineering pursuits go, software engineering is still a very young discipline. As both researchers and practitioners, we have not given sufficient attention to the theories that underpin our discipline, leaving those theories implicit rather than clearly delineated. Further, the empirical support for many of our (implicit) theories and decisions about research and practice are often anecdotal rather than rigorously and empirically evaluated. While our understanding of many critical issues in empirical studies has improved, we still have a significant amount of work to reach the level of understanding to be found in behavioral and physical sciences. There are many useful empirical techniques and structures we can borrow and adapt from them but there are still aspects to be more fully developed that are unique to software engineering.

In pursuit of these goals I have created a framework for modeling informally described theories and a calculus [15] for composing these modeled theories to create more complex theories from simpler ones. This framework provides a scientific elegance in creating more complex theories out of simpler ones, and an elegant way of explaining the complexity of software engineering and software engineering research. I claim that composition is as fundamental in the theoretical pursuits of software engineering as it is in the practical pursuits of constructing software systems.

In the context of a general theory of software engineering (GTSE), I am interested in the technical part of software engineering: the design, construction, evaluation, and deployment of software systems. As such, I separate out this part of software engineering from such aspects as project management, and business economics and strategic planning. I realize that software engineers play roles in the context of both project management and strategic planning, but I am concerned with their role building and evolving software systems. So with respect to a GTSE, I focus on what I consider to be the technical side of a GTSE. With respect to how this technical focus fits into a GTSE, Don Batory and I have proposed a theory for the compositional and structural organization of a GTSE that is described separately in [17].

My basic theory is that Software Engineering consists of two logical parts: design and empirical evaluation (both terms used in their broadest senses). Thus, I propose two simple theories, *D* and *E* as the basis for laying out a unified theoretical foundation for software engineering and software engineering research [16]. I propose theory D as the theoretical basis for the design part, and theory E as the theoretical basis for the empirical evaluation part. I then define models for D and E, using a model calculus, and subsequently compose theories D and E in various ways and use the composed models to explore various ways of thinking about the underlying foundations for software engineering and software engineering research.

Theories D and E abstract away details that undoubtedly differ significantly in actual research and practice, but, as in practice, abstraction is one of our most important intellectual tools for managing complexity. As will be illustrated in what follows, even with the abstract and simple theories D and E, the composition of theories results in significant increases in complexity, just as it does in the composition of

software components into software systems. Brooks in his *No Silver Bullet* paper [5], states: "*I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation*. . . .. If this is true, building software will always be hard. There is inherently no silver bullet." Further, he states that "Software entities are more complex for their size than perhaps any other human construct . . . ." I claim that a complete general theory of software engineering is at least equally hard, and at least as equally complex. My envisioned use of the framework and the model calculus is to delineate and explicate the elements of our design theories via models, and clarify the implications of those design models with respect to their evaluations via the resulting taxonomies. The resulting explicitness of both will help us to understand more fully and reason more carefully about what is often hidden in their current implicitness.

In what follows in this introduction, I discuss the following basic issues: experimental science; natural, behavioral, and artificial sciences; theories and models in the sense I will be using them in this research; and a further discussion about the sources, structure, and use of theories. Section 2 provides a formal definition of models and presents my model calculus. Section 3 illustrates my approach with simple versions of models for theories D and E and Section 4 illustrates the various basic compositions of E and D and what they are used for. In Section 5, I present more complex versions of D and E, D' and E', and the implications for compositions of these more complex theories in the composition E':E' – the evaluations of evaluations. To evaluate the utility of my approach, I apply my framework to three rather diverse theories and model them in Section 6 and summarize the paper in Section 7.

1.1 Experimental Science

Let us first take a basic look at science, even though one might argue that it is not necessary since everyone understands it thoroughly. My reason for doing this is to set the stage for the theories and models relevant for empirical software engineering.

Science is basically an iterative process consisting of the following steps:

- Observations and abstractions are used to create a theory T.
- We test theory T against reality W with an empirical evaluation E using one or more instruments I.
- We then reconcile theory with reality.
- When predictions don't agree with reality, we change the theory.

Gooding et al. [7] argue for the critical importance of the instruments we use in experimental work. They are the lens through which we observe the world. To paraphrase Wittgenstein [22], *the limits of my instruments are the limits of my world*. They enhance, limit, and color our view of the world. In natural sciences, instruments are often physical creations; in behavioral sciences they are often intellectual creations. Humans are common instruments in both. Instruments may be active or passive. They may be theory-laden or transparent and neutral. They may be reliable and standardized or not. In any case, they are a critical part of the empirical apparatus and as such will play a critical part in any scientific endeavor.

1.2 Natural, Behavioral & Artificial Sciences

In remedying our lack of rigor, it is critical to understand how the sciences of the artificial (that is, the sciences of design[1]) [19] differ from, and are similar to, behavioral and natural sciences. Obviously, we must have theories that are testable just as they do. The differences come in the context of testability and the constraints faced. The sciences of the artificial have some aspects in common with natural and behavioral sciences: testing is done in both physical and behavioral contexts. However, testing is also done in intellectual and technological worlds as well. For the physical and behavioral contexts we have the same hard and probabilistic constraints. For the technological context, we have selectable constraints – i.e., we have constraints we can select among, perhaps arbitrarily. For the intellectual context, we have malleable constraints – i.e., we have constraints that we can change, also arbitrarily.

There are interesting differences between natural and behavioral sciences that are relevant to design disciplines. The general goals of natural sciences are to understand natural phenomena and create a theoretical basis for *prediction*. Further, natural sciences provide a basis for invention and engineering. The general goals of behavioral sciences are to understand human and societal phenomena and provide a theoretical basis for prediction *and interventions*. The inventions and interventions are important because of the need to change the world which is one of the fundamental goals of software engineering: build systems of practical value in the world [10], not merely to observe it (though, of course, we do need to observe our software systems and make predictions about them as well).

1.3 Theories and Models

The terms "theory" and "model" are used and misused in a variety of ways, often informally and interchangeably. I want to use them in a very specific way: a theory (a more or less abstract entity) is reified, represented, satisfied, etc., by a model (a concrete entity).

---

[1] For those interested in delving more broadly into the issues of design, I highly recommend Simon's book [19] referenced here. Other useful references are: Suh's *The Axiomatic Theory of Design*, Oxford University Press1990; Gero's various papers on his function-behavior-structure framework; Cross's *Designerly Ways of Knowing*, Birkhäuser Architecture,2007; and Petre and van der Hoek, *Software Designers in Action: A Human-Centric Look at Design Work*, CRC Press, (forthcoming). And of course, there have been a variety of design methods proposed specifically for software engineering.

This view of theories and models is derived in part from Turski and Maibaum [20] where they state *"A specification is rather like a natural science theory of the application domain, but seen as a theory of the corresponding program it enjoys an unmatched status: it is truly a postulative theory, the program is nothing more than an exact embodiment of the specification"*. Note, however, that I want a theory to be broader than a specification and, more than likely, less formal.

We often use models as a representation of a theory. In natural sciences, the model is often a set of mathematical formulas. In logic, a model is an interpretation of a theory and has certain logical properties. Here, again, I want to broaden the notion of a model to be a representation (indeed, a reification) of the theory. The model is of paramount importance in design disciplines as it is the visible manifestation of a theory. Of fundamental importance is the fact that a theory can have an arbitrary number of models.

1.4 More About Theories

My claim is that the key to a unifying, rigorous, and systematic foundation for software engineering, software engineering research, and empirical studies in software engineering and software engineering research, is to be found in a focus on theory.

So what is it that I consider to be important in theories: 1) the source of the theories; 2) the structure of the theories; and 3) the use of the theories.

Source of Theories. In terms of sources of theories relevant to software engineering, three different types of theories are important:

1. Scientific theory – Scientific theory is based on *observations of* the world. They change on the basis of new observations, or new interpretations of observations.
2. Legal theory – Legal theory is quite different: it is based on *decisions about* the world, and is changed on the basis of new decisions or new interpretations of decisions.
3. Normative theory – Normative theory is different yet and is based on a system of philosophical tenets about what is good and bad, and judgments are changed on the basis of new tenets, new inferences from tenets, or new interpretations of them.

Theories in design disciplines are a combination of all three of the above. They are based on observations, decisions, and judgments about the world. They change on the basis of new observations, decisions, and judgments or on the basis of new interpretations of those observations, decisions, and judgments.

Structure of Theories. Markus and Robey [13] distinguish two different theory structures:

1. Variance – In the case of variance, the theoretical structure is a set of laws about interactions or relationships. For example, given a variation in theory A, what other units can be linked to A such that they account for the variance in A.
2. Process – In the case of process, the theoretical structure is a temporal ordering of activities, steps, or events.

We find both kinds of theoretical structures in design discipline theories depending on what kind, and at what level, we are theorizing about design issues.

Use of Theories. The taxonomy of uses I describe here is derived from Gregor [9]. I distinguish five distinct uses of theories that may be used also in combinations:

1. Description – A theory is used to describe phenomena in terms of its constructs, properties, and relationships, and the boundaries within which those properties and relationships hold. Descriptions are intended to be complete.
2. Prescription – A theory is used to provide a set of constraints on its constructs, properties, and relationships, and the boundaries within which those properties and relationships hold. Prescriptions are intended to emphasize the crucial aspects of the theory.
3. Explanation – A theory is used to explain how, why and when things happen based on causality and methods of demonstration (that is, argumentation). The intent is to provide deeper understanding and insight into the subject phenomena.
4. Prediction – A theory is used to predict what will happen on the basis of necessary and sufficient conditions for the theorized phenomena. The phenomena *will not* happen if the necessary conditions are withheld; nor *will* they happen if the sufficient conditions are withheld.
5. Action – A theory provides principles, techniques, and methods for enabling the desired phenomena (for example, achieving a desired goal, or designing or constructing an artifact).

Depending on the context in software engineering, we make use of theory in all these different ways. Theories, of course, influence their models: the source of a theory will affect its model; the structure of a theory will influence the structure of its model; and, the use of a theory will also influence the structure of its model.

2. A Model Calculus

Since theories as used here are informal entities, their composition is also informal and the resulting integration is done informally.

My theory about models, however, has a more formal definition and a set of rules for the model operators. My theory about models is that we can only truly understand the implications of our theories and compositions of theories by reifying them into models that illustrate the true breadth, depth and complexity of those models and compositions.

2.1 Models

A model is a tuple consisting of two sets: a set of objects O (or elements), and a set of mappings M (or more informally, transformations, processes, derivations, etc.) from an object in one set of objects to another object in a (usually, but not necessarily, different) set of objects.

Model = < {O}, {M} >

Models can be arbitrarily considered to be either *atomic* or *open-structured*. If viewed as atomic, then their structure is abstract – that is, it is hidden and the model is considered as a whole, indivisible entity. If viewed as open-structured, all the individual objects and mappings are visible in the model.

2.2 Model Syntax

The following are the special symbols in the model calculus (in order of precedence):

"+"	a unary operator on objects that indicates 1 or more of the designated objects.
":"	a binary operator on models and model components that indicates composition of two such elements – i.e., the restriction of the left model element or model to the right model element or model.
"*"	a binary operator on objects that delineates an object in the Cartesian space of two objects. This can be thought of as functional application of the first object to the other yielding a specific object as its value.
"→"	a binary operator that maps one object onto another.
( )	parentheses may be used to clarify the use of these operators.

Generally I use one alphabetic letter to denote objects and models, though of course one may use descriptive names for clarity if needed. For clarity in distinguishing between models or model elements, I sometimes use lower case letters for one of the model's elements. Further, I sometimes use a dot notation, *Model.Element* (for example D.T), to represent a specific element of a model (element T of model D), typically to distinguish it from another, possibly identical, element in a different model. I may also use different font colors for distinguishing models and/or elements of models. I have tried to provide distinct letters for the different elements in the various theories but T (theory) is used in both the design and empirical models D and E. The generally denote different theories.

2.3 Mappings

All possible mappings are possible in this model calculus:

- One to one mappings are indicated by A → B.
- Many to one mappings are indicated in several different ways. For example, A * B → C, and A+ → B.
- One to many mappings are indicated by A → B+ and A → B * C.
- Many to many mappings are indicated by any combinations using "+" and "*" together with "→"

2.4 Calculus Rules

The following are the distribution rules among expressions about various operators.

- ":" is both left and right distributive over models.
- ":" is left distributive over "+", "*", and "→".

Examples of the first distribution rule can be seen in the next section. Examples of the second are as follows:

$(O^1 \rightarrow O^2):AM = O^1:AM \rightarrow O^2:AM$
$(O^1 * O^2 \rightarrow O^3):AM = O^1:AM * O^2:AM \rightarrow O^3:AM$
$(O^{1+} \rightarrow O^2):AM = (O^1:AM)+ \rightarrow O^2:A$

There is one rule about the operator "+" (which implies that "+" is left distributive) over "*" and "→". For example,

(A → B)+ = A+ → B+
(A * B)+ = A+ * B+

2.5 Model Composition

Models (and theories) can be composed to yield further models (and theories). As mentioned above, model compositions are indicated by **A : B**. Further, models can be arbitrarily considered to be *atomic* (that is, its structure is hidden) or *open-structured* (that is, the objects and mappings are visible).

For example, composing an open structured model (OSM) with an atomic model (AM) results in a single model via left distribution of the atomic model AM over the open-structure model OSM.

OSM : AM =
<{O},{M}> : AM =
<{O}:AM, {M}:AM> =
<{O^1:AM . . . O^n:AM}, {M^1:AM . . . M^n:AM}>

Note that in this case, the above left distribution rules of composition over mapping operators can then be applied.

On the other hand, composing an atomic model with an open structure model yields a number of models (equal to the number of objects and mappings) via right distribution of the atomic model AM over the open-structured model OSM.

AM : OSM =
AM : <{O},{M}> =
<AM:{O}, AM:{M}> =
<{AM:O^1, ... , AM:O^n}, {AM:M^1, … , AM:M^n}> =
AM:O^1, … , AM:O^n, AM:M^1, ... , AM:M^n

Each of these compositions is a model restricted to that particular object or mapping. Note that because of the left distributive rule of composition over mapping operators, nothing further can be done if AM remains atomic. However, if AM is treated instead as an open-structure model, then the various objects and mappings can be left distributed over that now open model.

3. The Two Basic Theories *D* and *E*

For purposes of illustration, I have deliberately provided extremely simple (and *very* abstract) examples of Theories D and E. In this way, we can explore the utility of the formal model description format and the utility and implications of the composition rules in the context of both open structured models and abstract models. I will later provide more realistic but still abstract theories and models for D and E.

3.1 Theory & Model *D*

Theory D is meant to capture the typical cycle of creating a theory that is then reified into a model where the model is then injected into the world – i.e., initial, non-iterative, non-evolutionary development of a model from a theory. The injection of the model into the world results in a changed world[2]. I summarize it as follows:

- We observe and abstract some specific part of the world and create a theory.
- From that theory we create a usable model to reify or represent that theory.
- When satisfied that the model adequately represents the theory, we inject the model into the world..

The model of D consists of three elements (objects) and three transformations (mappings, or, if you will, processes). The elements are as follows:

- **W** the world, but more specifically, the part of the world relevant to the theory
- **T** the theory initiated by observations and abstractions
- **M** a model that reifies, represents or satisfies the theory T

The transformations involving these elements of the model are as follows:

- **W → T** generate a theory: observe and abstract from the world W to create a theory T
- **T → M** from the theory T create a model M
- **M*W → W** inject model M into the world W changing the world W

I claim that this model represents the theory of D above.

I claim that the design part of software engineering, at a suitable level of abstraction, is a reification of model D, and hence is a model of theory D. For example, W in theory D contains what Jackson [10] calls the *problem space*. It is that part of the world that represents the problem that we want to address with our software system. We observe and abstract from this problem space to create a theory T (i.e., theory T in D) of the problem we want to solve. We refer to T as requirements. W also contains what Jackson calls the *solution space*. It is in this space that we find the elements that we put together to create the model M (the software system itself) that reifies and represents those requirements T.

W→T is the process of deriving the requirements from the chosen problem space by observing and abstracting what is considered to be critical and central to the problem to be solved. It is also the process of understanding the effects of a changing world on the requirements that exist as the basis for an existing system M. T→M is the process of creating and evolving the model/system from the

[2] Note, however, the world can also change for a variety of other reasons. The discussion of those reasons is beyond the scope of this paper.

theory/requirements. And, finally, M*W→W releases/injects the model/system into the world to be used in solving the intended problem, and, in doing so, often radically changes the world. This is often referred to as *technology transfer*.

An Example.

In my career as a practicing software engineer, I built a system to keep manage samples of college textbooks for Harcourt, Brace & Jovanovich (HBJ). The part of the world **W** that was important in this example was *the marketing of college textbooks*. From this problem space in W, we created a theory **T** of *how to manage the distribution of samples of these textbooks to college teachers*. The process of creating T, **W → T**, was *done in conjunction with the editors and marketers of HBJ* and focused on the *input, handling, tracking, and distributing sample textbook requests*. The result of this process was a written document describing T that was approved by HBJ. The process of transforming T into a model M, **T → M**, was done using *COBOL in a top-down, incremental manner*. Once we had a completed model M satisfying theory T, we *released M into use*, **M*W → W**, fundamentally changing how HBJ managed the distribution of their college textbook samples.

3.2 Theory & Model *E*

As I did with D, I here propose a theory about a theory and model for E – a theory about empirical evaluation. For purposes of explanation and illustration I use a very simple theory for E. A more elaborate theory for is described below to illustrate more elaborate empirical evaluations. It is sufficient at this point to indicate that empirical evaluations can range from very informal (as indicated by this formulation of E) to formal and controlled experiments (as will be indicated by a more complete model of E).

Not surprisingly, the theory E is essentially a simplification of basic empirical science discussed above.

- Given a world W, observe and abstract to create a theory
- Given a theory T, generate an hypothesis H to test some part of the theory
- From the hypothesis H, generate an evaluation regimen R.
- Apply the regimen R to the world W and revise theory T if necessary.

Note that this is a very basic theory, but it still is sufficiently rich to cover the entire range of studies from exploratory through to rigorously explanatory studies. Of course, theory T may be vague and ill-formed (as it would be for exploratory work) or well-formed and mature (as it should be when doing explanatory work). Similarly the hypothesis may be generic and open-ended or focused and specific. Evaluations E may be human and opportunistic (for exploratory work) or specifically and well-designed. Further, the theory of *E* supports both theory generation (in the case of exploratory work) and focused evaluation of existing theory.

The basic elements in the model and their interrelationships are: world **W**, theory **T**, hypothesis **H**, and evaluation regimen **R**.

The following transformations represent the processes of conducting an empirical study.

- **W → T** generate a theory T by observing and abstracting from the world W
- **T → H** derive an hypothesis H from theory T
- **H → R** create an appropriate evaluation regimen R based on H
- **R * W → T** reconcile theory and reality – i.e., on the basis of the evaluation and the current theory T, revise T.

An Example.

In [11], we compared and evaluated two methods of deriving an architecture from a formal goal-oriented requirements specification of a power plant. The world **W** in this case was that of *methods of deriving models* (in this case, architectures) *from formal theory specifications* (requirements). Theory **T** is about *methods and the various relationships between formal specifications and their models*. In this case, part of the theory was that, given a formal specification, the methods would produce very similar, if not identical, architectures. The process of creating T from W, **W → T**, depends on a deep understanding of the domain *of formal specifications and reasoning about the formal or logical relationship between T and M*. In [C], we did not explicitly delineate T – something I claim we should be doing to make it clear what our underlying theory is in our empirical evaluations. Theories of evaluations are as critical as theories for our designs. Our (null) hypothesis **H** is the *models M (architectures) will be identical* – that is, there will be no differences in the architectures derived by the two different methods. The regimen **R** is as follows: create an architecture from the goal-oriented requirements specification using method A, then do the same derivation using method B, and compare the resulting architectures. The creating of R from H, **H → R**, was pretty straightforward in that R is supposed to test H and the best way to do that would be to create two example architectures and compare them. The regimen resulted in disproving the null hypothesis. In reconciling theory and reality, **R * W → T**, we would have to change our theory T about methods and their similar or identical effects.

4. Composing Theories

Thus we begin with two simple theories. It is my claim that these two simple theories are sufficient, with the model calculus rules, to generate a complete model of software engineering and software engineering research.

4.1 Evaluating the Design – *E:D* – i.e., E composed with D

It is here in the evaluation of the design part that we find the other half of the software engineering enterprise. It is here we determine the adequacy and utility of our theories and models, the efficacy of our processes in deriving these theories and models.

To evaluate theory D, we might compose an atomic model of E with an open structured model of D giving us the following 6 models – that is, a model for evaluating each element of D:

- Three models about evaluating the elements of D:
  - evaluation of the world of D **E:W**;
  - evaluation of the theory of D **E:T**;
  - evaluation of the model of D **E:M;**
- Three models about evaluating the processes of D:
  - creating a theory T from the world W **E:(W → T)**;
  - creating a model M from theory T **E:(T → M);**
  - evolving the world as a result of injecting model M into it **E:(M → W).**

Among the kinds of questions the evaluations might address are the following: the adequacy of D.T representing some part of W; the adequacy of D.M representing D.T; the utility of D.M in the world D.W; the effectiveness of such transformations as creating D.M from D.T, or of creating D.T from D.W.

Alternatively, one might want to evaluate D as a whole by composing an open structured model of E with an atomic D yielding a single model E:D that consists of

- The elements are
  - the world W of evaluations of D **W:D;**
  - the theory T of evaluations of D **T:D;**
  - the hypotheses H about evaluations of D **H:D**;
  - the regimens R for evaluations of D **R:D;**
- The processes of
  - creating a theory T:D from the world W:D **(W → T):D** = **W:D → T:D;**
  - deriving an hypothesis H:D from the theory T:D **(T → H):D** = **T:D → H:D;**
  - deriving a regimen R:D from the hypothesis H:D **(H → R):D** = **H:D → R:D**
  - applying regimen R:D to world W:D, evolving theory T:D **(R * W → T):D** = **R:D * W:D → T:D**

The fully expanded taxonomy of both E:D is seen in the matrices below:

E:D	**E:W**	**E:T**	**E:M**
W:D	**W:W**	**W:T**	**W:M**
T:D	**T:W**	**T:T**	**T:M**
H:D	**H:W**	**H:T**	**H:M**
R:D	**R:W**	**R:T**	**R:M**
(W → T):D	**W:W→T:W**	**W:T→T:T**	**W:M→T:M**
(T → H):D	**T:W→H:W**	**T:T→H:T**	**T:M→H:M**
(H → R):D	**H:W→R:W**	**H:T→R:T**	**H:M→R:M**
(R * W → T):D	**R:W*W:W→T:W**	**R:T*W:T→T:T**	**R:M*W:M→T:M**

E:(W → T)	**E:(T → M)**	**E:(M → W)**
W:(W→T)	**W:(T→M)**	**W:(M→W)**
T:(W→T)	**T:(T→M)**	**T:(M→W)**
H:(W→T)	**H:(T→M)**	**H:(M→W)**
R:(W→T)	**R:(T→M)**	**R:(M→W)**
W:(W→T) →T:(W→T)	**W:(T→M) →T:(T→M)**	**W:(M→W) →T: (M→W)**
T:(W→T) →H:(W→T)	**T:(T→M) →H:(T→M)**	**T:(M→W) →H:(M→W)**

H:(W→T) →R:(W→T)	**H:(T→M) →R:(T→M)**	**H:(M→W) →R:(M→W)**
R:(W→T)*W:(W→T) →T: (W→T)	**R:(T→M)*W:(T→M) →T:(T→M)**	**R:(M→W)*W:(M→W) →T:(M→W)**

In composing two models, we get n * m elements and processes – in this case, 6 * 8 = 48. Thus given these two models of design and evaluation, a fully rigorous evaluation of E:D would have 48 distinct elements and mappings for a full and rigorous evaluation.

Obviously, the more complex the models, the larger and more complex the taxonomy. For example the more complex alternative models present below will result in a much larger taxonomic space. Similarly if three models are composed together, the number of elements and processes would be n * m * o. The utility of the taxonomy here is that it delineates all the elements to be covered in systematically evaluating a design.

An Example.

In the example discussed above, the empirical study [11] could also be described as evaluations of the transformation of a theory (the goal-oriented requirements specification) to a model (in this case, an architecture), **E:(T → M)**, for each method**.** . Method A [12] began at a detailed level by constructing a dataflow architecture, applying architecture styles, and using patterns to achieve non-functional specifications, but did not have a specific criteria about when to stop applying styles and patterns. Method B [3,4] began at the most general level, goals, and required the user of the method to create an initial partitioning of the goals into the highest level components according to some architectural criteria (for example, non-functional characteristics, etc.). Once the initial structure was in place, the goals for each component were used to derive the sub-components to satisfy the various sub-goals. The architectural derivation was completed when all goals were completely covered. The two methods resulted in very different architectures: in method A, it was very easy to start but difficult to determine when to stop, and resulted in a detailed network-style architectural description that mirrored the structure of the specification; in method B, it was difficult to know how to begin but simple to determine when it was done, and resulted in a high level, hierarchical architectural prescription that did not mirror the specifications to the extent of the other.

4.2 Designing Design – Theory *D:D*

Theory D:D (the composition of D with itself) is meant to capture the typical cycle of creating a theory of D (i.e., a theory of producing a design product) that is then reified into a model of D and the model is injected into the world of D. I summarize it as follows:

- We observe and abstract some specific part of the world and create a theory of
  - What the world of D is like
  - What form a theory in D should take
  - What form a model in D should take
  - What form the processes of creating the theory and its model of D should take
  - How the resulting model of D should be injected into the world
- From that theory we create a usable model to reify or represent that theory of
  - What the world of D is like
  - What form the theory in D should take
  - What form the model in D should take
  - What form the processes of creating the theory and model D should take
  - How the resulting model of D should be injected into the world..
- When satisfied that the model adequately represents the theory we inject the model into the world.

The composition of model D (as an open structured model) with itself (as an atomic model) results in a new model with the following elements:

- **W:D** the world of D
- **T:D** the theory of D
- **M:D** the model of D

The transformations involving these elements of the model generate the following:

- **W:D → T:D** generate a theory: observe and abstract from the world of D to create a theory D
- **T:D → M:D** from the theory of D create a model of D
- **M:D * W:D → W:D** inject model of D into the world of D thereby changing it (which depends on both the model and the world before the injection of the model into it).

I claim that this model represents the theory D:D.

The composition of model D (as an atomic model) with itself (as an open structured model) yields nine models: **D:W**, **D:T**, **D:M, D:(W → T)**, **D:(T → M), D:(T → T), D:(M → M), D:(M → T)**, **D:(M x W → W)** – i.e., the design of each of the elements in the design model D, and the various transformations that take place in the design model D.

To illustrate the richness of compositional results, consider D:T where we now view D as an open structured model. We first get the objects **W:T**, **T:T** and **M:T** (the world of T, the theory of T, and the model of T respectively). We also get the following transformations: **W:T → T:T)**, **T:T → M:T, T:T → T:T, M:T → M:T, MT → T:T**, and **M:T * W:T → W:T,** exactly analogous to the transformations of D. The same follows for each of the remaining composed models above.

Analogous to my claim that the design aspect of *software engineering* is a reification of the model of D, I also claim that the design aspect of *software engineering research* is a reification of the model of D:D, and it is here that things get really interesting.

Discussion of Examples.

T:W, M:W – world of software development. The world of software systems is a varied and multi-faceted world. It is a world of problems and solutions [10]. It is a world that changes for a variety of reasons. It is a world where some problems are not solvable at all by automation as well as a world where some problems are just too hard to solve at all [6]. For the problems that are solvable, there are those that are solvable by what Vincenti [21] calls normal design and those that are solvable only by radical design. We may or may not be successful in solving problems that require radical design, but when we are successful we almost always need several iterations before we achieve that success.

It is a world of rapid technological change where software-intensive systems are increasingly invading our lives, where computation is constantly getting faster and cheaper, and where electronic storage is getting larger, faster and cheaper as well. It is a world where the bases for design decisions are constantly changing, where the tradeoffs we previously made must be re-examined in the light of the current state of the world.

T:T, M:T – theories/models of requirements. Frustratingly, there is little theory that is explicit in D:D.T:T or D:D.T:M; it is by-and-large implicit. Or, more specifically it is often stated normatively rather than descriptively (as one would find in natural sciences, for example). In one way, this is not surprising as our theories in D are largely normative: the system ought to do …; it ought to respond within …; it must provide …. Indeed, this normative approach is a feature of the sciences of the artificial [19]. And, of course, it is seen all too easily in every new *salvation du jour*.

However, as my goal in this paper is to lay a foundation for empirical software engineering, I claim that to make progress towards the kind of rigor we find in natural and behavioral sciences, that for this level of discourse we need to be more descriptive – that is, we need to be more explicit about our theories in such a way as to be easily testable. Note also that theories are critical in our empirical evaluations and there too we have not been as explicit as we need to be to become a rigorous discipline.

T:M, M:M – theories/models of software systems. Common theories in D:D about the form that a model D.M (or parts of the model) should take include structured programming, object oriented programming, aspect-oriented programming, etc. Looking at this in a different way, there are the theories about creating systems bottom up, or top down, or about structuring them for future change, or about organizing them hierarchically, or as networks of cooperating processes, or to reflect the shape of the problem. There are those who theorize that the components in software systems should be orthogonal and each component do one thing well, while others such as Jackson indicate we should be mindful of the fact that the world where we find our problem space has been implemented with the full exploitation of the Shanley Principle [10] of efficient design where each element serves multiple purposes.

There are a variety of theories in D:D about how we do the transformation from requirements to the system D:(T→M). The more or less standard ones include waterfall development, Boehm's spiral development, refinement, etc. A more radical departure from these standard approaches is that of Extreme Programming. An interesting variation of refinement can be found in Batory's algebraic compositional approach [2] that we consider below.

4.3 Evaluating the Theory *D:D – E:D:D*

To evaluate the design theory of D:D, we compose an atomic model of E with an open structured model D and atomic D in D:D giving us the following:

- the evaluation of
  - **E:(W:D)** the world of D
  - **E:(T:D)** the theory of D
  - **E:(M:D)** the model of D
- the evaluation of the processes of
  - **E:(W:D → T:D)** creating a theory T:D from the world W:D
  - **E:(T:D → M:D)** creating a model M:D from theory T:D
  - **E:(M:D → W:D)** injecting model M:D into the world W:D

As D:D was significantly more complex than D, so E:D:D is significantly more complex than E:D. Despite this increased complexity, the aims are still the same as with E:D. It's just that there are many more elements and processes to evaluate. The space is much larger – in this case 8 * 6 * 6 = 288 objects and mappings. But of course that is to be expected when we are concerned with theories and models about theories and models as we are in software engineering research.

However, just as the software engineering of software systems is composed of design and evaluation, so too is research about the design and evaluation of software systems composed of both design and evaluation. For example, evaluating

- the theory T in T:D might represent evaluating theory completeness, theory representativeness of the user's needs or problems, or theory adequacy (how good is the theory for this domain);
- the world W in W:D might represent evaluating the user's needs and desires about D, or the using of D, or the problem domain of D;
- the model M in M:D might represent evaluation the adequacy of model M of D – that is, how well does M:D represent T:D (black box testing) – or how good is M:D relative to the intent of the modeler (white box testing) – or the usefulness of M:D in W:D;
- the transformation W:D → T:D might represent evaluating the quality of the theory of D formation process;
- the transformation T:D → M:D might represent evaluating the quality of the model formation process; and
- the transformation M:D → W:D might represent evaluating the quality of the model deployment process.

Alternatively, consider E:D:D where D:D is atomic and E is open structured. E:(D:D) creates a new model of empirical evaluation focused specifically on D:D – i.e., evaluating D:D as a whole. That is, there are theories about D:D (derived from the world W:(D:D)) from which we create hypotheses, and from those hypotheses we create regimens (i.e., evaluations) – and we have the processes for doing just that with respect to D:D.

The elements of this model are as follows:

- **W:(D:D)** the world W of designing designs D:D
- **T:(D:D)** the theory T of designing designs D:D
- **H:(D:D)** the hypothesis about designing designs D:D
- **R:(D:D)** The regimen for evaluating the designing of designs D:D

The processes of evaluating the designing of designs D:D (left propagating D:D)

- **(W → T):(D:D)** **= W:(D:D) → T:(D:D)**
- **(T → H):(D:D)** **= T:(D:D) → H:(D:D)**
- **(H → R):(D:D)** **= H:(D:D) → R:(D:D)**
- **(R * W → T):(D:D)** **= R:(D:D) * W:(D:D) → T :(D:D)**

If we then take this model and consider the first D, the design of software engineering research, and make it open structure, leaving the second D atomic, we get an even more complex set of models. I take as an example the evaluation E:(T:D):

- **W:(T:D)** the world of theories about D
- **T:(T:D)** a theory of the theories about D
- **H:(T:D)** an hypothesis derived from the theory about theories about D
- **R:(T:D)** a regimen to test the hypothesis about the theory about theories about D
- **W:(T:D)→T:(T:D)** deriving a theory about the theories about D from the world of theories about D
- **T:(T:D)→H:(T:D)** deriving an hypothesis about theories about D from the above theory
- **H:(T:D)→R:(T:D)** deriving a regimen to test the hypothesis about theories about D
- **R:(T:D)*W:(T:D)→T:(T:D)** applying the regimen in the world of theories about D, evolving the theory if needed

Clearly, a full and rigorous evaluation of software engineering research designs is a large and complex process. However, just as we seldom do research on the entirety of the domain of software engineering, so do we seldom need to do a full and complete evaluation of software engineering research design of the entire domain of software engineering. We do, however, need to do a full and rigorous evaluation of that part of the domain of software engineering on which we do conduct research.

Discussion of Examples.

In the context of E:(D:D), there are several interesting examples. One is an instantiation of E:(T:D → M:D) . There is a common sub-theory of T:D that van Lamsweerde [12] and Brandozzi and Perry [3, 4] agree on: namely, that an architecture description can be generated effectively from a formal goal-oriented requirements specification – that is, a structural model can be generated from a goal-based theory. Not surprisingly, the two models are significantly different, as van Lamsweerde's is based on using the structure of the theory to generate the structure of the model, where Perry's is based on the goals described in the theory to generate the structure of the model. The evaluation of E:(T:D → M:D) is provided in Jani et al. [11] in which the strengths and weakness of each generation process and model are evaluated and described using a common goal-based theory.

Another example is found in Shao et al. [18] where there is a theory about software engineering, T:D, specifically a theory about models, T:M, such that changes to a model made under certain conditions are at increased risk of inserting faults with those changes. A model was created for this theory, M:D, that detects faults under the conditions described in the T:D. The specific evaluation E:(M:D) was performed in which it was determined that the model M was effective in detecting the predicted faults under the described theoretical conditions. In addition the evaluation determined what kinds of faults it would not detect.

4.4 Evaluating Evaluations – *E:E, E:E:D, and E:E:D:D*

In addition to evaluating our designs, we need also to evaluate our evaluations as well as evaluate our evaluations of designs, especially relative to issues of construct, internal, conclusion, and external validity. To evaluate the empirical theory E, we compose an atomic model of E with an open structured model of E giving us the following eight models:

- the evaluation of the elements of E:E
  - **E:W** evaluation of the world of W of E
  - **E:T** evaluation of the theory of T of E
  - **E:H** evaluation of the hypothesis H of E
  - **E:R** evaluation of the evaluation regimen R of E
- the evaluation of the processes of
  - **E:(W → T)** evaluation of creating a theory T from the world W
  - **E:(T → H)** evaluation of creating a hypothesis H from theory T
  - **E:(H → R)** evaluation of creating an evaluation from hypothesis H
  - **E:(R * W → T)** evaluation of evolving theory T as a result of the evaluation E

The issues that need to be considered here are those concerning the adequacy of the evaluations and the effectiveness of the evaluation processes. Among the critical issues are those such as the appropriateness of the world W of possible Es, the completeness and consistency of the theory as well as its appropriateness and adequacy, the relevance of the hypothesis to the theory, the relevance of the empirical regimen to the hypothesis and theory, and the standard problems of construct, internal and external validity. Further there is the need to evaluate the quality of the theory derivation, hypothesis derivation, regimen generation, and the application and reconciliation processes.

In addition to evaluating our different forms of empirical evaluations themselves (E:E), we also need to evaluate our specific evaluations of design in its various incarnations (E:E:D and E:E:D:D).

4.5 Designing Evaluations – *D:E* and *D:E:E*

To design the empirical evaluation theory E, we compose an atomic model of D with an open structured model of E giving us the following:

- the design
  - **D:W** & **D:(E:W)** of the world W of E and its evaluation
  - **D:T** & **D:(E:T)** of the theory T of E and its evaluation
  - **D:H** & **D:(E:H)** of the hypothesis H of E and its evaluation
  - **D:R** & **D:(E:R)** of the regimen of E and its evaluation
- the design of the processes, and their evaluations, of
  - **D:(W → T)** & **D:(E:(W → T))** creating a theory T from world W
  - **D:(T → H)** & **D:(E:(T → H))** creating a hypothesis H from theory T
  - **D:(H → R)** & **D:(E:(H → R))** creating an evaluation regimen from hypothesis H
  - **D:(R x W → T)** & **D:(E:(R x W → T))** evolving theory T as a result of the evaluation R –

The design of empirical evaluations and the design of evaluating empirical evaluations (D:E and D:E:E) is analogous to D:D: it is part of the software engineering research enterprise.

5. Alternative and More Complex Theories for D and E

In the discussions and examples above of the theories D and E and their various compositions, I deliberately used extremely simple versions of the two base theories. In the case of the theory of design D, I omitted that fact that design is both iterative and evolutionary. Similarly, in the case of E, I deliberately simplified the theory of evaluation to make it easier to illustrate the issues of composition. In this section I expand both D and E to be more realistic, though in the case of D, still at a rather high level of abstraction. Evaluation theory E represents a class experimental or quasi-experimental structure.

5.1 An Iterative and Evolutionary Theory of Design – *D'*

The model of D' consists of three elements (objects) and six transformations (mappings, or processes). The elements are as follows:

- **W** the world, but more specifically, the part of the world relevant to the theory
- **T** the theory initiated by observations and abstractions
- **M** **a** model that reifies, represents or satisfies the theory T

The mappings involving these elements of the model are as follows:

- **W → T** generate a theory: observe and abstract from the world W to create a theory T
- **T → M** from the theory T create/evolve a model M
- **T → T** evolve theory T until satisfied
- **M → M** evolve the model M until satisfied
- **M → T** change the theory T to better conform to model M
- **M * W → W** inject model M into the world W thereby changing it (which depends on both the model and the world before the injection of the model into it).

The process of iteration includes **T → T** (refining and changing the theory until it is satisfactory), **M → M** (refining and changing the model until it is satisfactory), and **M → T** (providing feedback from the model M to evolve the theory T to bring it more in line with what is possible in the model M). The injection changes the world, **M * W → W,** induces a cycle of evolution for both the theory and its model. Obviously, other changes in the world induce the same cycle.

5.2 Refining the Model

The model in D is a monolithic entity where in reality we consider the model to be comprised of three components: architecture, design, and code. We can extend the iterative theory and model D' into **R** by replacing the various uses of M in D' with elements A, D, and C (architecture, design, and code) and introduce a new element S (system) that is the result of "building" the code into a system.

- Replace **M** with **A, D**, and **C**
- Add a new object **S** and a mapping **C → S**
- **T → T** is replaced with **T → A, T → D,** and **T → C**
- **M → T** is replaced with **A → T, D → T,** and **C → T**
- **M →** M is replaced with
  - **A → D, D → C**
  - **A → A, D → D, C → C**
  - **C → A, D → A, C → D**

The new design theory and model R has 6 objects and 21 mappings. Obviously much more complex and the compositions we have considered as basic are even more so. Small wonder that Brooks [2] considers complexity to be the most significant of the essential characteristics. Using the simple model E, the evaluation of R, E:R, has 8 * 27 = 216 elements and evaluations of E:R. E:E:R has 1728 elements. E':R (the more complete evaluation theory in the next subsection) explodes even more.

5.3 A More Complete Evaluation Theory – *E'*

What is missing in E is the centrality of instruments as a critically important [8] element in empirical science. They are the lens through which we observe the world. To paraphrase Wittgenstein [22], *the limits of my instruments are the limits of my world*. They enhance, limit, and color our view of the world. In natural sciences, instruments are often physical creations; in behavioral sciences they are often intellectual creations. Humans are common instruments in both. Instruments may be active or passive. They may be theory-laden or transparent and neutral. They may be reliable and standardized or not. In any case, they are a critical part of the empirical apparatus and as such will play a critical part in any empirical evaluation endeavor.

Theory E' then is described as follows:

- Given a *theory*, generate an *hypothesis* to test some part of the theory
- From the hypothesis, generate a *regimen* (that is, a treatment) and one or more *instruments*.
- The regimen is a treatment mechanism to generate desired data in the designated *context*.
- The *instruments* are used to provide *observations* about the generated *data* resulting from the manipulations of the independent variables.
- On the basis of the instrumented *observations*, generate a set of result *analyses* to use to reconcile the observations with the theory, revising the theory if needed.

Iteration is inherent in E' as it is in both E and D. We iterate revising T and generating new hypotheses, regimens and instruments to refine theory T. Theory E' then provides us with a more realistic view of empirical ventures.

Of course, with the richer theory E', we need a richer model to represent E'. The model of E' has 8 objects and 7 mappings. The objects are as follows:

- **T** theory
- **H** hypothesis
- **R** regimen (treatment) manipulates independent variables in context C
- **C** context (where the independent and dependent variables are found) in which R is applied and observed by I
- **D** data resulting from application of the regimen
- **I** instrument(s)
- **O** observations of the data D seen through instrument(s) I

- **A** analyses based on the observations

The mappings are as follows:

- **T→H** deriving a hypothesis H from theory T
- **H→I** create appropriate instruments/mechanisms for observing the dependent variables
- **H→R** create appropriate regimen with manipulations of independent variables
- **R*C→D** performing the experiment: apply regimen to context yielding resulting data
- **I*D→O** use instrument I to provide observations of data D
- **O→A** generate analyses from the observations O – i.e., drawing conclusions from the experiment
- **A*T→T'** reconcile theory and reality (that is, the results of the analysis): evaluate T possibly revising it into T'

An Example.

It is usually the case that D.T is a different theory from E.T. In the retrospective study [18], the two are basically the same. This is because the Semantic Conflict Analyzer (SCA) is an instrument. The theories E.T and D.T are basically that software engineers need a reasonable amount of time to digest changes and to understand their implications. As a result of this, changes made concurrently, or at least within a reasonably short amount of time, are more likely to introduce faults. SCA was designed and implemented to efficiently detect these interfering changes that result from semantic conflicts. The null hypothesis H is that *time intervals among changes have nothing to do with the likelihood of introducing faults*. The regimen is to use SCA as the instrument I to analyze three populations of changes where the time intervals between changes are 1) a month or more, 2) a week or more but less than a month, and 3) less than a week. We also factored in whether the changes represented bug fixes, improvements, or new features. The context C was change and version control databases in a large real-time project. Because the regimen is the application of SCA to these various programs the data D and observations O are the same: a set of semantic conflicts. The analysis A compared the set of semantic conflicts with the known sets of faults for the programs and determined which were relevant conflict and which were false positives (conflicts where there were not faults – semantic interference often results from changing code and is then intentional) and which were were any false negatives (faults where there were no conflicts). The null hypothesis was disproved and the study provided support for both theories that time is an important factor in the likelihood of faults. The results enabled us to extend the theory with the following: there were few unintentional conflicts in fixing bugs and making improvements, but semantic conflicts were very likely when introducing new features.

5.4 Evaluating the Alternative Evaluation Theory E' – *E':E'*

With the more complex theory and model of E' comes a larger number of models as a results of the composition E':E' – 16 for E':E' instead of 9 for E:E. The taxonomic space is significantly larger as well – 225 elements instead of 64.almost a factor of 4 larger. The usefulness of the more complete theory and model is the exposure of what in many cases are critically important issues.

1) *Using an Atomic E' To Evaluate an Open-Structured e'.* In this and the following sections we use E' and e' to distinguish the two uses of E'. In this and the next section, we also illustrate the usefulness in the atomic versus open-structured. In this case we consider E' to be atomic, and used as a whole to evaluate the individual objects and mappings of e'.

- **E':t** evaluate the theory of e'
- **E':h** evaluate the hypothesis of e'
- **E':I** evaluate the instrument of e'
- **E':r** evaluate the regimen of e'
- **E':c** evaluate the context of e'
- **E':d** evaluate the data of e'
- **E':o** evaluate the observations of e'
- **E':a** evaluate the analyses used in e'
- **E':(t→h)** evaluate the derivation of the hypothesis of e' from the theory of e'
- **E':(h→i)** evaluate the determination of the instrument for e' from the hypothesis of e'
- **E':(h→r)** evaluate the design of the regimen derived from the hypothesis for e'
- **E':(r*c→d)** evaluate generation of data by means of the application of the regimen for evaluating e' to its context
- **E':(i*d→o)** evaluate the generation of observations from the data using the instruments of e'
- **E':(o→a)** evaluate the derivations of the analysis from the observations for e'
- **E':(a*t→t')** evaluate the reconciliation of the analysis and the theory and possibly revising the theory for e'

The interpretation of elements of the model E':e' may be as follows. E':t may evaluate theory completeness and consistency, or theory appropriateness and adequacy. E':h may evaluate the appropriateness of the hypothesis h relative to the theory t, or evaluate the construct validity of hypothesis h. E':r may consider the appropriateness of regimen r relative to hypothesis h and theory t, or consider the construct and/or internal validity of regimen r. E':c may evaluate the appropriateness of context c relative to theory t and hypothesis h. E':(t→h)

may evaluate the quality of the hypothesis generation process, or evaluate construct validity. E':(h→r) may address the problems of construct and internal validity.

2) *Using an Open-Structured E' To Evaluate Atomic e'.* In this case, we consider E' to be open-structured and e' to be atomic, thus evaluating e' as a whole in terms of the individual elements of E'

The set of objects then in the composition of E':e' are as follows.

- **T:e'** a theory about evaluating e'
- **H:e'** an hypothesis about evaluating e'
- **R:e'** a regimen for evaluating e'
- **C:e'** a context for evaluating e'
- **D:e'** data for e' resulting from the application of the regimen for evaluating e'
- **I:e'** instruments for evaluating e'
- **O:e'** observations of the data seen through the instruments for evaluating e'
- **A:e'** analyses based on the observations in evaluating e'

The set of mappings in the composition of E':e' are as follows:

- **(T→H):e' = T:e' → H:e'**

  Deriving an hypothesis for evaluating e' from the theory about evaluating e'

- **(H→I):e' = H:e' → I:e'**

  Determining an instrument to be used in the evaluation of e' from the hypothesis for evaluating e'

- **(H→R):e' = H:e' → R:e'**

  Determining a regimen for evaluating e' from the hypothesis for evaluating e'

- **(R*C→D):e' = R:e' * C:e' → D:e'**

  Performing the evaluation of e' in the context for evaluating e' yielding the evaluation data for e'

- **(I*D→O):e' = I:e' * D:e' → O:e'**

  Applying the instrument for evaluating e' to the evaluation data for e', yielding evaluation observations about e'

- **(O→A):e' = O:e' → A:e'**

  Deriving an analysis of the evaluation of e from the evaluation observations

- **(A*T→T'):e' = A:e' * T:e' → T':e'**

  Reconciling the evaluation of e' with the theory about evaluating e' and possibly revising that theory.

3) Expanding the Details – Both Models Open-Structured. To understand fully the implication of what is hidden by keeping one of the models atomic (which does provide a useful abstraction in reducing the inherent complexity of model composition), we consider an example from each of the preceding compositions to illustrate that complexity.

Let us consider a simple example from using E' to evaluate an object of e', E':t. We have a set of objects in this model as follows:

- **T:t** a theory T about the theory t of e'
- **H:t** an hypotheses H about the theory t of e'
- **I:t** an instrument I used for theory t of e'
- **R:t** a regimen R used for the theory t of e'
- **C:t** a context C appropriate for theory t of e'
- **D:t** data for the theory t of e'
- **O:t** observations for the theory t of e'
- **A:t** analysis relevant to the theory t of e'

We then use these objects in the mappings of E' for the evaluation of t.

- **(T→H):t = T:t → H:t**

  Generating an hypothesis H about t from the theory T about t

- **(H→I):t = H:t → I:t**

  Deriving an instrument I for t from the hypothesis H about t

- **(H→R):t = H:t → R:t**

Deriving a regimen R about t from the hypothesis H about t

- **(R*C→D):t = R:t * C:t → D:t**

  Applying the regimen for t to the context of t to generate the data about t

- **(I*D→O):t = I:t * D:t → O:t**

  Using the instruments for t to yield observations about t from the data of t

- **(O→A):t = O:t → A:t**

  Deriving analyses about t from the observations on t

- **(A*T→T'):t = A:t * T:t → T':t**

  Reconciling the analyses about t with the theory about t, possibly modifying the theory about t

In the second example where we applied individual objects and mappings to e' as a whole (that is, as atomic), to fully understand the implications of that we must open the structure of e'. Consider the example, (T→H):e' = T:e' → H:e'.

- **(T→H):t = T:t → H:t**

Generating an hypothesis H about *theory t* from the theory T about *theory t*. The rest of the compositions of objects are analogous

- **(T→H):h = T:h → H:h**
- **(T→H):I = T:i → H:i**
- **(T→H):r = T:r → H:r**
- **(T→H):c = T:c → H:c**
- **(T→H):d = T:d → H:d**
- **(T→H):o = T:o → H:o**
- **(T→H):a = T:a → H:a**
- **(T→H):(t→h) = T:(t→h) → H:(t→h)**

Generating an hypothesis H about *the process of generating an hypothesis h from the theory t* from the theory T about *generating the hypothesis h from t.* The rest of the compositions of mappings of this form are analogous.

- **(T→H):(h→i) = T:(h→i) → H:(h→i)**
- **(T→H):(h→r) = T:(h→r) → H:(h→r)**
- **(T→H):(r*c→d) = T:(r*c→d) → H:(r*c→d)**

Generating an hypothesis H about *the process of generating data by the application of the regimen r to the context c* from the theory T about *the process of generating data by the application of the regimen r to the context c.* The remaining mappings of this form are also analogous to this one.

- **(T→H):(i*d→o) = T:(i*d→o) → H:(i*d→o)**
- **(T→H):(o→a) = T:(o→a) → H:(o→a)**
- **(T→H):(a*t→t') = T:(a*t→t') → H:(a*t→t')**

These two examples delineating the full composition of several instances of the composition E':e' provide a good illustration of the size of the full taxonomic matrix resulting from such a composition.

Discussion.

Theory and model E' provide a more detailed and fuller approach to empirical evaluation. The difference between E and E' is primarily that of detail with E' providing an emphasis on the importance of instruments, observations, and analyses. Both E and E' are intended to cover the entire range of empirical studies from exploratory through to rigorous experimental studies. The theory T may be vague and ill formed when doing exploratory work, or well-formed and mature when doing explanatory work (typically using a null hypothesis experiment). The hypothesis H may be generic and open-ended when doing exploratory work, or focused and specific when doing explanatory work. The instruments I and regimens R (i.e., treatments) may be human and opportunistic when doing exploratory work, or specifically and well-designed in both cases when doing explanatory work. Reconciliation of theory and reality may result in theory generation when doing exploratory work, or adding support for the theory or revising it when doing explanatory work.

With respect to the issues of validity, the various elements of E':e' are as follows:

- Construct validity – (the most critical issue for design decisions) is addressed by at least **E':h, E':i,** and **E':(h→i)**
- Internal validity – is addressed by at least **E':h, E':i, E':r, E':(h→r), E':(r*c→d),** and **E':(i*d→o)**
- Statistical validity – is addressed by at least **E':a, E':(o→a),** and **E':(a*t→t')**
- External validity – is addressed by at least **E':c**

An interesting question is whether the E' should be used to evaluate itself or whether some other theory and model, perhaps simpler, should be used instead – for example E. As seen above, the taxonomic space is large using E' to evaluate itself. This is not necessarily a bad thing as it does have the utility of illustrating just how large and complex the problem is of evaluating our evaluations. The simpler theory E hides a large amount of that complexity. Further exploration of this issue is needed to understand the utility, strengths and weaknesses of alternative approaches.

6. Applying the Framework to Other Theories

So far the validation of this framework and approach to a theory for software engineering and software engineering research has been basically an existence proof with some illustrative examples or discussions. – that is, showing that it works for what I want to accomplish in exploring my theory of software engineering and software engineering research as a set of compositions of two simple theories. In this section I apply the framework to a variety of other theories about various aspects of software engineering to illustrate its validity and utility with respect to these theories.

6.1 Multiple Viewpoints Requirements Engineering based Software Engineering

An example of such a more detailed theory and model is that of Nuseibeh, Kramer and Finkelstein's multiple viewpoints [14] approach in which there are different stakeholders with respect to the problem to be solved; these stakeholders have different views on what is important in the theory (i.e., requirements that need to be captured; and eventually any and all apparent and real conflicts need to be resolved to provide a consistent theory (i.e., a consistent set of requirements).

1) Theory V (Viewpoints)

The added details to theory V are primarily in the generation of a theory that is then the basis for creating the corresponding model.

- Multiple stakeholders observe and abstract some specific part of the world and create a part of a theory that is relevant to them.
- These stakeholders' partial theories must be merged into a consistent single theory.
- From that theory we create a usable model to reify or represent that theory.
- We iteratively adjust both the theory (and sometimes the stakeholders' partial theories) and the model as our understanding of the theory and its model evolves, both iteratively and interactively.
- When satisfied that the model adequately represents the theory, we inject the model into the world.
- Injecting the model into the world changes the world.
- The changes brought about by these changes as well as other changes often lead to adjustments and extensions to the stakeholders partial theories and the merged theory.
- Changes to the theory in turn lead to further changes in the model and the world.

So, theory iteration becomes more complex and in turn makes the iteration between the theory and model more complex as well.

2) Model V

For the most part, the theory of D is carried over into V. Similar to the refinement of D' discussed above, V is a refinement on the structure of deriving the theory T that is changed and provided in more detail

- **W** the world, but more specifically, the part of the world relevant to the theory
- **S** the stakeholders who create partial theories
- **P** partial theories
- **T** the theory resulting from merging partial theories
- **M** a model that reifies, represents or satisfies the theory T

The mappings involving these elements of the model are as follows:

- **S+ * W → P+** stakeholders generate partial theories: observe and abstract from the W to create a partial theories P+
- **P+ → T** the partial theories are merged into a consistent theory
- **T → P+** the partial are adjusted due to consistency issues in theory T
- **T → T** evolve theory T until satisfied
- **T → M** from the theory T create/evolve a model M
- **M → M** evolve the model M until satisfied
- **M → T** change the theory T to better conform to model M
- **M * W → W** inject model M into the world W thereby changing it (which depends on both the model and the world before the injection of the model into it).

A possible alternative to T → P+ to capture the process of theory consistency issues resulting from either inconsistencies in the partial theories P+ or changes in the theory T to better to conform to the model M is to use instead a much more complex mapping: T → **(**S+ ***** W → P+). This mapping would represent the process of the theory T generating a new set of partial theories by the various stakeholders. In T → P+, the need for the stakeholders to modify their partial theories is implicit rather than explicit.

As always, then, how models reify their theories is a creative process often with various alternatives of perhaps differing utility and clarity. And, there is usually a tradeoff between simplicity and expressiveness.

3) Compositions with V

Analogous to D:D, the obvious thing to do is to use V in the same way to create a theory and model about creating a theory and model. The details of the composition of V:V I leave as an exercise to the reader to help expand their understanding of the model calculus in the filling out of the taxonomic matrix.

The interesting question here is whether V is appropriate since a theory is created from partial theories created by various stakeholders. There are two cases to consider: where there is one researcher, and where there is more than one researcher. V actually can be used to represent both of these cases as the "+" operation is defined to mean "one of more" stakeholders. As we often work in teams, even as researchers, V is quite appropriate for the task. Not only is appropriate, it is probably more advantageous: V:V more clearly represents collaborative research than D:D.

The question, however, of whether this is the best solution, or whether a different, or simpler, theory and model is more useful or appropriate. This, as with the same question about evaluating evaluations, needs further study.

6.2 People-Centered Software Engineering

Adolph and Kruchten, in their GTSE workshop paper [1], argue that people are central, even dominant, in the software engineering enterprise: "the main concern of people involved in the process of software development is getting the job done and that different points of view and expectations create impediments – a perspective mismatch". When a perspective mismatch is discovered, people converge their mismatched perspectives by negotiating a consensual perspective. These consensual perspectives are viewed as forming a set of "integrated hypotheses" that are used to produce a grounded theory [7] to explain the behavior of various participants.

In what follows, I use the term "observations" (for the sake of simplicity) for Adolph's and Kruchten's "consensual perspectives" or grounded theory's "hypotheses" in the discussion of their approach. I claim their theory P includes E and extends D and E:D. At an abstract level, theory P adds four new elements and three extended theory mappings (based on D and E) for P:

- **P** person, typically a software engineer
- **O** observations (negotiated perspectives, hypothesis)
- **R** researcher (a special subset of P)
- **T_{se}** theory of software engineering
- **P+ * D → O+** one or more people derive one or more observations about creating/evolving design
- **P+ * (E:D) → O+** one or more people derive one or more observations about evaluating a design
- **R+ * O+ * T_{se} → T_{se}** one or more researchers create or modify a theory of SE using the derived observations

To make these abstractions more concrete, I expand D to make it clearer what P entails:

- **P+ * W → O+**
- **P+ * T → O+**
- **P+ * M → O+**
- **P+ * (W→T) → O+**
- **P+ * (T→M) → O+**
- **P+ * (M*W→T) → O+**
- **R+ * O+ * T_{se} → T_{se}**

- **P+ * E:W → O+**
- **P+ * E:T → O+**
- **P+ * E:M → O+**
- **P+ * E:(W→T) → O+**
- **P+ * E:(T→M) → O+**
- **P+ * E:(M*W→T) → O+**

I add to further mappings to represent people providing observations about people and observations as part of creating/evolving T_{se}.

- **P+ * P → O+**
- **P+ * O → O+**

Further, E needs to be expanded as well to indicate the full range of observations that need to be provided to fully ground the resulting theory of software engineering – I leave that as an exercise for the reader. At this point we have a full model of P.

I claim that P itself now needs to be evaluated, just as the theory of design D needed to be evaluated as represented by E:D. The following is the list of elements in evaluating P, E:P.

- **E:(P+ * W → O+)**
- **E:(P+ * T → O+)**
- **E:(P+ * M → O+)**
- **E:(P+ * P → O+)**
- **E:(P+ * O → O+)**
- **E:(P+ * (W→T) → O+)**
- **E:(P+ * (T→M) → O+)**
- **E:(P+ * (M*W→T) → O+)**

- **E:(P+ * E:W → O+)**
- **E:(P+ * E:T → O+)**
- **E:(P+ * E:M → O+)**
- **E:(P+ * E:(W→T) → O+)**
- **E:(P+ * E:(T→M) → O+)**
- **E:(P+ * E:(M*W→T) → O+)**

- **E:(R+ * O+ * T_{se} → T_{se})**

To illustrate what E:P implies, let us take a look at the one of these mappings and expand E to an open structure. One of the more interesting of the elements is **E:(P+ * (T→M) → O+)** – that is, evaluating the generation of peoples' observations about the process of producing a model from a theory. We have a set of objects as follows (using a colored font to represent the open-structured E).

- A world of people's observations about deriving a model from a theory
  - **W:(P+ * (T → M) → O+)**
- A theory about evaluating people's observations about deriving a model from a theory
  - **T:(P+ * (T → M) → O+)**
- An hypothesis to test about people's observation about deriving a model from a theory
  - **H:(P+ * (T → M) → O+)**
- A regimen for evaluating people's observations about deriving a model from a theory
  - **R:(P+ * (T → M) → O+)**

The evaluations of the mappings appear to be, and, indeed are, complicated.

- Deriving a theory about peoples observations about deriving a model from a theory from the world of peoples observations about deriving a model from a theory
  - **(W→T) : (P+ * (T → M) → O+)** = **W:(P+ * (T → M) → O+) → T:(P+ * (T → M) → O+)**
- Deriving an hypothesis about peoples observations about deriving a model from a theory from a theory of peoples observations about deriving a model from a theory
  - **(H→R) : (P+ * (T → M) → O+)** = **H:(P+ * (T → M) → O+) → R:(P+ * (T → M) → O+)**
- Reconciling the results of a regimen evaluating peoples observations about deriving a model from a theory, with the world of peoples observations about deriving a model from a theory, possibly modifying the evaluated theory
  - **(R*W→T) : (P+ * (T → M) → O+)**
    = R:(P+ * (T → M) → O+) * W:(P+ * (T → M) → O+)) → T:(P+ * (T → M) → O+)

The rest of the evaluations follow the same pattern. The composition delineates the full taxonomy of what it means to evaluate the theory P via the composition of its model with the model of E. The notion of an evaluation represented with E is very broad where that of E' is much more narrow and the composition E':P much more complex.

6.3 Algebraic Refinement

Having taken a very traditional approach to design and evaluation in D and E, it is worthwhile to consider a very non-traditional approach such as that of Batory's Feature Oriented Programming [2].

> *"Feature Oriented Programming (FOP) is a design methodology and tools for program synthesis. The goal is to specify a target program in terms of the features that it offers, and to synthesize an efficient program that meets these specifications"*

Batory has created an algebra used to describe how features are composed together to create a system. Design rules define what comprises a legal composition by delineating the semantic constraints that need to be met for a proper composition. Compositions are expressed as algebraic equations that are then used, with the design rules and the supporting tools, to synthesize the system composition from the individual feature components. Evolution of the system is then a matter of synthesizing a new system by evolving the algebraic equations to incorporate new features or to eliminate no longer desired features. The algebraic approach and the design rules are innovative but the lynchpin are the underlying tools that enable the use of the equations and rules to synthesize a system from the set of features.

As I have done earlier, I present a simplified version of his approach (leaving out iteration – that can be done by the reader as an intellectual exercise). The elements in Batory's design theory F, which extend and modify D, are as follows:

- **W** world
- **T** theory
- **F** feature
- **A** algebra
- **R** design rule
- **M** model
- **W → T** derive a theory from the world
- **T → F+** derive features from the theory
- **(A * R+) * F+ → M** derive a model from the features via the algebra and design rules
- **M * W → W** inject the model into the world

To get the evaluation half of software engineering, we compose E with F:

- **E:W** evaluate the relevant world
- **E:T** evaluate the theory
- **E:F** evaluate the features

- **E**:A evaluate the algebra
- **E:R** evaluate the design rules
- **E:M** evaluate the model
- **E:(W → T)** evaluate the process of deriving a theory from the world
- **E:(T → F+)** evaluate the process of deriving features from the theory
- **E:((A * R+) * F+ → M)** evaluate the creation of a model from applying the algebra and design rules to the features
- **E:(M * W → W)** evaluate injecting the model into the world

Again I leave it as an exercise for the reader to delineate the full implications of this composition and its taxonomy. By this time, it should be clear as to how to do this.

Having laid out what Batory's theory F entails as used by a software engineer to design and evaluate feature oriented systems, I turn to what a theory of research might be in creating theory F by composing D and F – composing a traditional theory of theory and model generation with a non-traditional theory. I leave it to Batory to explore what F:F might look like and what the meta-models of the features, the algebra, and the design rules might look like, how you would generate them, and evaluate them. And of course, we also need E:D:F for evaluating the designs D:F, D:E:F for the designs of E:F, and E:D:E:F for evaluating those designs of E:F.

D:F	**D:(E:F)**	**E:(D:F)**	**E:(D:(E:F))**
D:W	**D:(E:W)**	**E:(D:W)**	**E:(D:(E:W))**
D:T	**D:(E:T)**	**E:(D:T)**	**E:(D:(E:T))**
D:F	**D:(E:F)**	**E:(D:F)**	**E:(D:(E:F))**
D:A	**D:(E:A)**	**E:(D:A)**	**E:(D:(E:A))**
D:R	**D:(E:R)**	**E:(D:R)**	**E:(D:(E:R))**
D:M	**D:(E:M)**	**E:(D:M)**	**E:(D:(E:M))**
D:(W → T)	**D:(E:(W → T))**	**E:(D:(W → T))**	**E:(D:(E:(W → T)))**
D:(T → F+)	**D:(E:(T → F+))**	**E:(D:(T → F+))**	**E:(D:(E:(T → F+)))**
D:((A * R+) * F+ → M)	**D:(E:((A * R+) * F+→ M))**	**E:(D:((A * R+) * F+→ M))**	**E:(D:(E:((A * R+) * F+→ M)))**
D:(M * W → W)	**D:(E:(M * W → W))**	**E:(D:(M * W → W))**	**E:(D:(E:(M * W → W)))**

To illustrate, as we did the previous section, the implications of these composition, we use the mapping **D:(T → F+)** from the composition **D:F** as our example and open the structure of D (the structure of F is already open).

- The world of processes where features are derived from a theory
  - **W:(T→F+)**
- A theory of a process of deriving features from a theory
  - **T:(T→F+)**
- A model of a process of deriving features from a theory
  - **M:(T→F+)**
- A process of creating a theory of deriving features from a world of deriving features from a theory
  - **(W→T):(T→F+)** = **W:(T → F+) → T:(T → F+)**
- A process of deriving a model of deriving features from a theory from a theory of deriving features from theories
  - **(T→M):(T→F+)** = **T:(T → F+) → M:(T → F+)**
- Injecting a model of deriving features from a theory into the world of deriving features from theories
  - **(M*W→W) :(T→F+)** = **M:(T→F+) * W:(T→F+) → W:(T→F+)**

Based on the number of elements in F, D, and E (10, 6, and 8 respectively) the taxonomic spaces for these four compositions are as follows:

- D:F 60 elements
- D:E F 480 elements
- E:D:F 480 elements
- E:D:E:F 3840 elements

Obviously things get much more complicated if we instead used D' and E' in the compositions with their respective element counts of 9 and 15. The main point is that the taxonomic space grows significantly the more complex the theories and their models. The likelihood that one would cover, or perhaps even need to cover, all of these elements is small, especially in the richer compositions. However, the taxonomies do provide a space where once can locate precisely where one's research focus lies and which of the taxonomic elements one's research and evaluation cover.

7. Conclusions

The motivation for my research is to create a unifying theoretical basis for software engineering and software engineering research. In particular I want to make two important points: 1) the usefulness of my view of theories and models, and 2) the important distinction between design and evaluation, both of which are critical parts of software engineering and software engineering research.

In making this distinction I especially want to emphasize the centrality and criticality of theory in both software engineering and software engineering research. Too often, little is explicit in the underlying theoretical basis of our work in software engineering and software engineering research. We need to make our underpinning theories more explicit and central in our work.

Similarly, I want to emphasize the centrality and criticality of empirical evaluations in both enterprises as well. Too little attention is paid to the evaluation part of software engineering and software engineering research with the primary, and sometimes sole, emphasis put on design (obviously critically important in its own right). We need a more explicit, systematic, and deeper approach to empirical evaluations in both software engineering and software engineering research.

I propose theory D as the theoretical basis for the design part of software engineering, and theory E as the theoretical basis for the empirical evaluation part (which is the composition E:D). These two theories and the composed theories then lay out a rich space (a taxonomy, or ontology if you will) for all of software engineering and software engineering research.

From these two theories I have created a set of various composed theories that focus on various aspects of design and evaluation. The first composition is that of E:D in which we actually realize the empirical evaluation part of software engineering. The composition of D with itself, D:D, gives us the design portion of software engineering research, while E:D:D provides us with the empirical evaluation of the design of our research. The empirical evaluations themselves need to be empirically evaluated and E composed with itself, E:E, provides that. Of course, there is then the design of the empirical evaluations that we represent with the compositions D:E and D:E:E.

My goal has been to illustrate the very rich space that we can define using small and relatively simple theories – in this case making D' and E' slightly richer than D and E. The beauty of this approach is that the composition of these theories and their models extend our understanding and illuminate the taxonomic space for the resulting theories and their models.

Moreover, this approach is even more general than that. I also claim that such design disciplines as project management, instrument creation and evolution, empirical studies themselves are also models of these atomic and composed theories – that is, they are design disciplines and hence models for the theories of design disciplines that I have presented and alluded to.

Finally, there are a number of useful properties in my approach: 1) a theory modeling language and calculus for composing the resulting theoretical models; 2) the utility of the two levels of abstraction (atomic and open-structured) that I use to provide an intuitive high level abstraction, and the explicit low level details; 3) regularity among the various theories; 4) a scientific elegance in creating larger more complex theories out of simpler theories; and 5) an elegant way of explaining the complexity of software systems, software engineering and software engineering research.

8. Acknowledgements

This research was sponsored in part by NSF CISE SRS Grant CCF-0820251. My thanks also to Don Batory for a willingness to listen and comment.

9. References

[1] Steve Adolph and Philippe Kruchten, "Generating A Useful Theory of Software Engineering"

[2] D. Batory, J.N. Sarvela, and A. Rauschmayer. "Scaling Step-Wise Refinement", IEEE Trans. on Software Engineering, June 2004

[3] Manuel Brandozzi and Dewayne E Perry. "Transforming Goal Oriented Requirements Specifications into Architectural Prescriptions." Workshop From Software Requirements to Architectures (STRAW1), International Conference on Software Engineering 2001, Toronto, May 2001, 54-61.

[4] Manuel Brandozzi and Dewayne E Perry. "From Goal-Oriented Requirements to Architectural Prescriptions: The Preskiptor Process." International Workshop From Software Requirements to Architectures (STRAW2), International Conference on Software Engineering 2003, Portland OR, May 2003, pp 107-113.

[5] Frederick P. Brooks, Jr., "No Silver Bullet: Essence and Accidents of Software Engineering," reprinted in *The Mythical Man Month, Silver Anniversary Edition*, Addison-Wesley, 1995.

[6] L. Fortnow, Steve Homer. "A Short History of Computational Complexity". In D. van Dalen, J. Dawson, and A. Kanamori, editors, *The History of Mathematical Logic*. North-Holland, 2002.

[7] B. Glaser and A. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago, Illinois: Aldine, 1967.

[8] David Gooding, Trevor Pinch, and Simon Schaffer, Editors. *The Uses of Experiment: Studies in the Natural Sciences.* Cambridge: Cambridge University Press, 1989.

[9] Shirley Gregor. "The Nature of Theory in Information Systems", MIS Quarterly 30 (2006), pp 611-642

[10] Michael Jackson. "The World and the Machine", 17th International Conference on Software Engineering, 1995, Seattle WA, 283-292.

[11] Divya Jani, Damien Vanderveken and Dewayne E Perry. "Deriving Architectural Specifications from KAOS Specifications: A Research Case Study", European Workshop on Software Architecture 2005, Pisa Italy, June 2005.

[12] van Lamsweerde, A. "From system goals to software architecture". In Bernardo, M., Inverardi, P., eds.: Formal Methods for Software Architectures. Volume 2804 of Lecture Notes in Computer Science. Springer-Verlag (2003) 25-43.

[13] M. L. Markus and D Robey. "Information Technology and Organizational Change: Causal Structure in theory and Research", *Management Science* 34:5 (1988), pp 583-598.

[14] Bashar Nuseibeh, Jeff Kramer, Anthony Finkelstein. “Expressing the Relationships Between Multiple Views in Requirements Specification”, The International Conference on Software Engineering 1993, Austin TX, May 1993.

[15] Dewayne E Perry, “A Theoretical Foundation for Software Engineering: A Model Calculus", 2nd Semat Workshop on a General Theory of Software Engineering (GTSE 2013), 35th International Conference on Software Engineering (ICSE 2013), San Francisco, May 2013.

[16] Dewayne E Perry. "A Unifying Theoretical Foundation for Software Engineering", 20th International Conference on Software Engineering and Data Engineering (SEDE), June 2011.

[17] Dewayne E Perry and Don Batory, “On the Structure of General Theories of Software Engineering,” ARiSE Technical Report xxx, October 2013. http://users.ece.utexas.edu/~perry/work/papers/gtse-structure.pdf

[18] D. Shao, S. Khurshid, and D. Perry. “Evaluation of Semantic Interference Detection in Parallel Changes: an Exploratory Experiment”. 23rd International Conference on Software Maintenance (ICSM 2007). Paris, France. Oct 2007.

[19] Herbert A. Simon. *The Sciences of the Artificial*. Cambridge: MIT Press, 1969.

[20] Turski, Wladyslaw M. and Maibaum, Thomas S. E. *The Specification of Computer Programs*. Reading, Mass: Addison-Wesley, 1987.

[21] Walter G. Vincenti. *What Engineers Know and How They Know It*. Baltimore: The Johns Hopkins University Press, 1990.

[22] Wittgenstein, Ludwig. *Tractatus Logico-Philosophicus*, ed. A. J. Ayer, London: Routledge & Kegan Paul, 1961. Section 5.6.