A Unifying Theoretical Foundation (or perhaps better: Framework) for Software Engineering

Dewayne E Perry

Center for Advanced Research in Software Engineering (ARiSE)
perry@mail.utexas.edu
Separation of Concerns

- An important separation of concerns - distinguish between
  - Theories about software engineers
    - As people (individual or in teams), as designers, as creators, as programmers, as architects, as engineers, etc
    - How people and teams interact, cooperate to create and evolve software systems
    - Cognition is located here
  - Theories about software engineering
    - The actual crafting and engineering of software systems
    - The structure of the artifacts
    - How to create and evolve them
    - Techniques and structures to manage complexity is here
  - Theories about software project management
    - Managing software engineers and software engineering
    - How to best organize and assign people given resources
    - Managing project resources, roles, etc
Separation of Concerns

★ Theories about the relationship between the theories of software engineers and software engineering
   ➢ Eg, various cognitive issues for SEs are related to various principles and structures used in SEing
★ Theories about the relationships between theories of project management, software engineers, and software engineering
   ➢ Eg, SPM is concerned about the utility and effectiveness of SEs and the progress, quality and cost of SEing
   ➢ Eg, PM metrics and productivity of SEs
   ➢ Eg, SE roles and responsibilities wrt SEing artifacts

★ I am primarily interested in Theories about Software Engineering
★ But ultimately will want to compose?/integrate? theories of SE, SPM, SEing, SE-SEing, and SPM-SE=SEing
Introduction

- I have a general theory about software engineering – it is made up of two basic endeavors:
  - Design
    - Of the problem
    - Of the solution
    - And includes a mundane manufacture
  - Evaluation
    - Of our problem, its solution, and the solution’s utility
    - Of our evaluations themselves

- I also have a theory about Theories and Models

- See:
  - “A Unifying Theoretical Foundation for Software Engineering”
    users.ece.utexas.edu/~perry/work/papers/DP-sede11.pdf
Theories and Models

- Terms often used in a variety of ways
  - Informally, interchangeably

- Want to use in a very specific way
  - A theory is a description (an abstract entity)
  - Reified, represented, satisfied, etc by a model (a concrete entity)

- Derived from Turski and Maibaum [TM87]
  - “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”

- I want a theory to be broader than a specification and less formal – for software engineering think of
  - The requirements (and the domain) as a theory, and
  - The software system as a model of that theory
    - And there can be many models that satisfy that theory
Source of Theories

- **Scientific Theory**
  - Based on observations about the world
    - Some observations sets very old but still used/useful
  - Changed on the basis of
    - New interpretations of observations
    - New observations

- **Legal Theory**
  - Based on decisions about the world (e.g., acceptable behavior)
  - Changed on the basis of
    - New interpretations of decisions
    - New decisions

- **Normative Theory**
  - Based on a system of philosophical tenets about what is good and bad
  - Changed on the basis of
    - New inferences from those tenets
    - New interpretations of the tenets
    - New tenets
Context for Software Engineering

- Physical world provides hard constraints on theories
- Behavioral world provides probabilistic constraints
- Normative world provides qualitative constraints
- Technological world provides selectable constraints
- Intellectual context provides malleable constraints
Theories D & E

I begin with two simple theories:

- A theory about design - D
- A theory about empirical evaluation - E

And a theory about how to model theories

My theory of software engineering:

- Software engineering is composed of two things
  - A design D
  - And an evaluation of that design E:D
  - SE = < D, E:D >

My theory of software engineering research:

- A design of software engineering D:SE
- And an evaluation of that design E:(D:SE)
- RSE = < D:SE, E:(D:SE) >
  = < D:<D, E:D>, E:<D:D, D:(E:D)> >
  = < <D:D, D:(E:D)>, <E:(D:D), E:(D:(E:D))> >
  = < D:D, D:(E:D), E:(D:D), E:(D:(E:D)) >
Theory D (simplified: no iteration)
Model of D (simplified – no iteration)

- **W**  
The world - more specifically, the relevant part of the world - ie, the problem space

- **T**  
The theory initiated by observation and abstraction about the problem

- **M**  
A model that satisfies the theory

- **W\rightarrow T**  
Generate a theory: observe and abstract from the world (W) to create a theory (T)

- **T\rightarrow M**  
From theory (T) create a model (M)

- **M*W\rightarrow W**  
Inject the model (M) into the world (W) - Thereby changing the world
Theory E (basic)
Model of E (basic)

- **W** The world - more specifically, the relevant part of the world
- **T** The theory initiated by observation and abstraction from the world
- **H** An hypothesis to test the theory
- **R** An regimen to test the hypothesis
- **W→T** Generate a theory: observe and abstract from the world (W) to create a theory (T)
- **T→H** From theory (T) generate an hypothesis (H)
- **H→R** From hypothesis (H) generate an empirical evaluation to test it
- **R*W→T** Apply R to W and reconcile T with reality
Theories D & E

What do we get from my approach:
 Scientific elegance in creating larger more complex theories out of simpler theories
 Explain the complexity of software engineering and software engineering research in an elegant way.
 A theory modeling language and a calculus for composing models

Why my approach?
 Description and understanding of what we do
 Provide a basis for exploring various approaches and what they entail
 To delineate the landscape of SE and RSE – lay out a taxonomic space
 To emphasize the importance of explicating theories in both software engineering design and empirical evaluation
 To emphasize the importance and extent of the empirical part of Software Engineering
Rest of Talk

- Several reviewers commented:
  - “Interesting but not practical”

- Two examples as illustrations and explorations
  - Steve Adolph and Philippe Kruchten’s approach in “Generating A Useful Theory of Software Engineering”
    - Delineate Adolph/Kruchten theory of SE research - call it P
    - Create a theory of research evaluation - the composition E:P
  - Don Batory’s Feature-Oriented Programming
    - Delineate Don’s theory of design - call it F
    - Use empirical evaluation E to create theory E:F
    - Use D as Don’s theory of SE research
    - Create theory D:F that represents research about feature oriented programming
    - Use empirical evaluation E to create a theory of evaluation of D:F - that is E:(D:F)
Models and Applications

- What is a model – a tuple of a set of objects and a set of mappings
  - < {objects}, {mappings} >
  - Objects - eg, W, T, M, etc
    - Elements, components, entities, etc
  - Mappings - W → T, R*W → T, etc
    - Transformations, generations, derivations, processes, etc

- Can treat models as
  - Atomic (A) - ie, abstract away the internal structure
  - Open structured (O) - ie, expose the internal structure

- Application: X:Y
  - Basically restricts X to Y - ie, focuses X on Y
  - A:O - yields n models where n is the number of objects + the number of mappings
  - O:A - yields 1 model where the objects and mappings are restricted to A
Model Calculus - Operators

- Special symbols (in their 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 a restriction of the left model or element to the right 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 one 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.
Model Calculus - Mappings

- All possible mappings are possible
  - One to one mappings are indicated by $A \rightarrow B$.
  - Many to one mappings are indicated in several different ways. For example, $A \ast B \rightarrow C$, and $A+ \rightarrow B$.
  - One to many mappings are indicated by $A \rightarrow B^+$ and $A \rightarrow B \ast C$.
  - Many to many mappings are indicated by any combinations using "+" and "∗" together with "→"
Model Calculus - Rules

- The following are the distribution rules among expressions about various operators.
  - "\(\rightarrow\)" is both left and right distributive over models.
  - "\(\rightarrow\)" is left distributive over "\(+\)", "\(*\)”, and "\(\rightarrow\)".
- Examples of the first distribution rule are above.
  Examples of the second are as follows (where EM denotes a model or a model element):
    - \((O^1 \rightarrow O^2) : EM = O^1 : EM \rightarrow O^2 : EM\)
    - \((O^1 \times O^2 \rightarrow O^3) : EM = O^1 : EM \times O^2 : EM \rightarrow O^3 : EM\)
    - \((O^1+ \rightarrow O^2) : EM = (O^1 : EM)^+ \rightarrow O^2 : EM\)
- There is one rule about the operator "\(+\)" (which implies that "\(+\)" is left distributive) over "\(*\)" and "\(\rightarrow\)". For example,
  - \((A \rightarrow B)^+ = A^+ \rightarrow B^+\)
  - \((A \times B)^+ = A^+ \times B^+\)
Adolph/Kruchten Theory - P

- “A Grounded Theory is a set of integrated conceptual hypotheses systematically generated to produce a theory”
- “Grounded Theory generates a substantive theory that explains participants’ behavior as a set of integrated hypotheses”
- “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 reaching out and negotiating a consensual perspective (which I refer to as observations – or for grounded theory, hypotheses)
Adolph/Kruchten Theory - P

- Incorporates D and E
- New elements for P
  - P: person (i.e., software engineer)
  - O: observations - negotiated perspectives - hypotheses
  - R: researcher (a special subset of P)
  - T_{se}: theory of software engineering

At an abstract level

- P+ * D \rightarrow O+
  - One or more people derive one or more observations about creating/evolving a design
- P+ * (E:D) \rightarrow O+
  - One or more people derive one or more observations about evaluating a design
- R+ * O+ \rightarrow T_{se} or alternatively R+ * O+ * T_{se} \rightarrow T_{se}
  - One or more researchers create or modify a theory of SE using the observations
Adolph/Kruchten - P

- Need to expand D
  - P+ * W \rightarrow O+
  - P+ * T \rightarrow O+
  - P+ * M \rightarrow O+
  - P+ * (W \rightarrow T) \rightarrow O+
  - P+ * (T \rightarrow M) \rightarrow O+
  - P+ * (M * W \rightarrow T) \rightarrow O+
  - R+ * O+ * T_{se} \rightarrow T_{se}
  - P+ * P \rightarrow O+
  - P+ * O \rightarrow O+

- Need to expand E:D
  - P+ * E:W \rightarrow O+
  - P+ * E:T \rightarrow O+
  - P+ * E:M \rightarrow O+
  - P+ * E:(W \rightarrow T) \rightarrow O+
  - P+ * E:(T \rightarrow M) \rightarrow O+
  - P+ * E:(M * W \rightarrow T) \rightarrow O+
  - R+ * O+ * T_{se} \rightarrow T_{se}
  - P+ * E:P \rightarrow O+
  - P+ * E:O \rightarrow O+
Adolph/Kruchten - P

* P * E:(T\(\rightarrow\)M) \(\rightarrow\) O+ - Model E from Atomic to Open Structured
  * P+ * W:(T\(\rightarrow\)M) \(\rightarrow\) O+
    - People’s observations about the world of creating a model from a theory
  * P+ * T:(T\(\rightarrow\)M) \(\rightarrow\) O+
    - People’s observations about a theory of creating a model from a theory
  * P+ * H:(T\(\rightarrow\)M) \(\rightarrow\) O+
    - People’s observations about an hypothesis about creating a model from a theory
  * P+ * R:(T\(\rightarrow\)M) \(\rightarrow\) O+
    - People’s observations about a regimen about creating a model from a theory
Adolph/Kruchten - P

★ P+ * ((W→T):(T→M)) → O+
   = P+ * W:((T→M) → T:(T→M)) → O+
   ▶ People's observations about deriving a theory of creating a model
      from a theory, from a world of creating models from theories

★ P+ * ((T→H):(T→M)) → O+
   = P+ * T:((T→M) → H:(T→M)) → O+
   ▶ People's observations about deriving an hypothesis about creating a
      model from a theory, from a theory of creating models from theories

★ P+ * ((H→R):(T→M)) → O+
   = P+ * H:((T→M) → R:(T→M)) → O+
   ▶ People's observations about deriving a regimen for evaluating the
      derivation of an model from a theory, from an hypothesis about
      creating models from theories

★ P+ * ((R*W→T):(T→M)) → O+
   = P+ * (R:(T→M) * W:(T→M) → T:(T→M)) → O+
   ▶ People's observations about reconciling the evaluation of a theory of
      creating a model from a theory, with the world of creating models
      from theories, possibly modifying that evaluated theory
Adolph/Kruchten – Model E:P

To evaluate the creation/evolution of P

★ $E : (P^+ \ast W \rightarrow O^+)$
★ $E : (P^+ \ast T \rightarrow O^+)$
★ $E : (P^+ \ast M \rightarrow O^+)$
★ $E : (P^+ \ast (W \rightarrow T) \rightarrow O^+)$
★ $E : (P^+ \ast (T \rightarrow M) \rightarrow O^+)$
★ $E : (P^+ \ast (M \ast W \rightarrow T) \rightarrow O^+)$
★ $E : (P^+ \ast E : (W \rightarrow O^+))$
★ $E : (P^+ \ast E : (T \rightarrow O^+))$
★ $E : (P^+ \ast E : (M \rightarrow O^+))$
★ $E : (P^+ \ast E : (W \rightarrow T) \rightarrow O^+)$
★ $E : (P^+ \ast E : (T \rightarrow M) \rightarrow O^+)$
★ $E : (P^+ \ast E : (M \ast W \rightarrow T) \rightarrow O^+)$
★ $E : (R^+ \ast O^+ \ast T_{se} \rightarrow T_{se})$
Adolph/Kruchten - Model E:P

- \( E: (P+ * (T \rightarrow M) \rightarrow O+) \)
  - \( W: (P+ * (T \rightarrow M) \rightarrow O+) \)
    - A world of people's observations about deriving a model from a theory
  - \( T: (P+ * (T \rightarrow M) \rightarrow O+) \)
    - A theory about people's observations about deriving a model from a theory
  - \( H: (P+ * (T \rightarrow M) \rightarrow O+) \)
    - An hypothesis about people's observations about deriving a model from a theory
  - \( R: (P+ * (T \rightarrow M) \rightarrow O+) \)
    - A regimen for evaluating people's observations about deriving a model from a theory
  - \( (W \rightarrow T): (P+ * (T \rightarrow M) \rightarrow O+) = W: (P+ * (T \rightarrow M) \rightarrow O+) \rightarrow T: (P+ * (T \rightarrow M) \rightarrow O+) \)
    - 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
Adolph/Kruchten - Model E:P


- 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


- 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


- 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
Batory Theory of Design F

“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”

★ “the constants and functions of a domain model — which is an algebra — can be implemented with many different technologies”

★ “equational representations of programs are very powerful”

★ “Design rules capture semantic constraints that govern legal compositions”
Batory Theory of Design F - Model

- **Elements in F (simplified - ie no iteration)**
  - **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 * F+ * R+ → M** derive a model from the features via the algebra
  - **M * W → W** inject the model into the world
E:F - Evaluating Design Theory F

- Evaluating F - E: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 * F+ * R+ → 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
Theory of Research D:F and D:(E:F)

- D:F
  - D:W
  - D:T
  - D:F
  - D:A
  - D:R
  - D:M
  - D:(W \rightarrow T)
  - D:(T \rightarrow F+)
  - D:(A \ast F+ \ast R+ \rightarrow M)
  - D:(M \ast W \rightarrow W)

- D:(E:F)
  - D:(E:W)
  - D:(E:T)
  - D:(E:F)
  - D:(E:A)
  - D:(E:R)
  - D:(E:M)
  - D:(E:(W \rightarrow T))
  - D:(E:(T \rightarrow F+))
  - D:(E:(A \ast F+ \ast R+ \rightarrow M))
  - D:(E:(M \ast W \rightarrow W))
Theory of Research D:F and D:(E:F)

- **D:(T → F+)**
  - ★ **W:(T→F+)***
    - World of processes where features are derived from a theory
  - ★ **T:(T→F+)***
    - Theory of a process of deriving features from a theory
  - ★ **M:(T→F+)***
    - Model of a process of deriving features from a theory
- ★ **(W→T):(T→F+) = W:(T → F+) → T:(T → F+)***
  - A process of creating a theory of deriving features from a world of deriving features from a theory
- ★ **(T→M):(T→F+) = T:(T → F+) → M:(T → F+)***
  - A process of deriving a model of deriving features from a theory from a theory of deriving features from theories
  - Injecting a model of deriving features from a theory into the world of deriving features from theories
Evaluating Batory's F-O Research

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

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

- Small, simple theories D and E form the basis for laying out a very rich space and an underlying theoretical foundation for SE, SE research, and other design disciplines
  - Compose D and E into more complex theories to extend and illuminate the space for design disciplines
- Useful properties
  - Regularity among the various theories
  - Levels of abstraction (stratification) within the composed theories providing
    - Intuitive high level abstractions
    - Explicit low level detailed abstractions
- Used approach to model two very different approaches to theories of software engineering: P and F - and the utility of applying D and E to both of them.