A Theory about the Structure of GTSEs

Dewayne E Perry
ENS 623
Perry@ece.utexas.edu
Separation of Concerns

- An important *separation of concerns* - distinguish between
  - Theories about *software engineers (SEs)*
    - 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 (SEing)*
    - 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 (SPM)*
    - 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
Background

Software Engineering

- A broad complex field
- Fundamental software engineering principles:
  - Modularity
  - Encapsulation
  - Abstraction
  - Separation of concerns
- These principles should apply to General Theories of Software Engineering (GTSEs) as well
  - GTSE will also be complex

SEMAT 2013 – one result of the discussion

- Just as our software systems are component-based, multi-level, we need to think about a multi-level GTSE

SEMAT 2014 – Wieringa’s paper

- Argues for a variety of “middle-range” theories rather than one grand theory
Our Theory

- Using the Perry/Wolf architecture metaphor
  - [http://users.ece.utexas.edu/~perry/work/papers/swa-sen.pdf](http://users.ece.utexas.edu/~perry/work/papers/swa-sen.pdf)

- Proposed architectural structure of a GTSE
  - **Component theories**
    - Major components – for example
      - Business Strategy and Economics
      - Software Project Management
      - Software Engineers
      - Software Engineering
  - **Connector theories**
    - Relationships and interdependencies among component theories – for example
      - Cognition – a critical element of a theory about software engineers
      - Structural complexity – a critical element in theory of components
      - A connector theory would delineate the relationship between the two
        - Eg, see Bill Curtis et al, “Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics.” *IEEE Transactions on Software Engineering*, 5 (2), 96-104. (1979)
An Example Overview of Components

- **Business Strategy and Tactics - Economics**
  - The business folks can address these issues - for example
  - Core competencies
  - Market windows
  - Perceived demand
  - Costs and profit

- **Project Management - possible components**
  - **Planning**
    - Effort Estimation
    - Resource Costs
    - Project Planning
    - Project Constraints
  - **Resource Allocation**
  - **Monitoring and Metrics**
An Example Overview of Components

Software Engineers – some component theories

★ As Individuals

✦ Basic skills
  ✔ Programmers as knowledge-based understanders
  ✔ Distributed cognition in software teams

✦ Training, education, and experience

✦ Judgment

★ As Members of Teams

✦ Team formation

✦ Team structure
An Example Overview of Components

- **Software Engineering** - some component theories
  - **Software Architecture**
    - Components – capturing computation
    - Connectors – capturing interactions and relationships
  - **UML Diagrams** – captures design level
    - Classes
    - Relationships
  - **Model Driven Engineering (MDE)**
    - Metamodels
    - Compositions
  - **Software Product Lines**
    - Features
    - Feature Interactions
  - **Software Design in general**
    - Satisfiability problems
Examples of Connector Theories

- A Relationship between software engineers and software engineering: *cognition, complexity, and software structure*
  - Software structure - extremely complex
  - Complexity: partly structural, partly cognitive (cf Curtis)
    - Primary issue: relationship and interdependency between
      - Cognitive load
      - Program structure
    - Curtis et al provide a connector theory
  - SE techniques to reduce or manage complexity
    - Structured programming
    - Modularity
    - Encapsulation
    - Abstraction
      - Parameterization
      - Information hiding
    - [OO captures these three in Classes]
Examples of Connector Theories

★ Techniques reduce cognitive load
  ➢ Simplify similar pieces of code
  ➢ Reduces amount of code where there is pervasive use of abstraction
  ➢ Simplifies interfaces
  ➢ Provides intuitive organization

❖ Project Planning and Software Engineer Estimates
  ★ Software Engineers where multiple hats
    ➢ A designer hat and an estimator hat among them
  ★ Project Planning requires estimates as to how much time is needed for a particular activity
  ★ There are two forms of time: race time & lapse time
    ➢ SEs tend to think in race time
    ➢ Project Planners tend to think in lapse time
    ➢ Our studies in 5ESS showed a factor of 2.5 difference there
Summary

- A full GTSE is analogous to a very large complex system
  - Needs to be decomposed into pieces
  - Modularity, encapsulation, and abstraction are needed
  - Multi-level architecture is an appropriate model

- Our theory about the architectural structure of a GTSE
  - Component theories to capture domain specific theories
  - Connector theories to capture inter-relationship theories
  - Hierarchical decompositions to refine complex component and connector theories - ie, recursively refine and explicate

- We have illustrated our theory with examples

- Our the simple elegance of this approach provides two basic elements to be used recursively to expand the full space of general theories of software engineering.