Differences in Cognitive Skills Required for Systems Engineering Versus Software Engineering

Tom McDermott, Dr. Dennis Folds
Georgia Institute of Technology
October 31, 2013
• Georgia Tech Systems Engineering Education Program

• Several hundred students widely diversified across government, aerospace, and commercial industry

• Generally all with >5 years experience in engineering fields, 5-30 year breadth of experience levels and roles

• Observation: when dealing with open-ended problems and complex architectures, software trained students are more adept at grasping SE concepts

• Premise: experience and training in SE fields does not emphasize abstract thought, software does better

• Is there a model we can promote, based on role?
General history of SE disciplines

Control Theory & Cybernetics (Weiner)
Operations Research (Morse)
“Systems Engineering” (Bell Labs, Gilman)
The Computer

Systems Management (Ackoff)
Organizational Systems (Checkland)
“Software Engineering” (Bell Labs, Gilman)
Software Languages

System Dynamics (Forrester)
Soft Systems (Checkland)
SE Discipline
Structured Programming

Theory of Constraints (Goldratt)
Complex Systems (Simon)
Life Cycle Engineering
Object Oriented design

Systems Sciences
System Engineering
Software Engineering

Concrete and Abstract Thinking

- Distinction between concrete and abstract thought predates Psychology as a science
- Many definitions of “abstraction” in the research literature
- Representative definition: the “process of identifying a set of invariant central characteristics of a thing” (Burgoon, Henderson, & Markman, 2013)
- Generally thought of as a useful process
  - Marker of cognitive development in children
  - Some disorders manifest in incorrect abstractions
• Generalization of acquired knowledge to novel situations (e.g., wayfinding, categorization)
• Anticipation / prediction of future conditions or events
• Basis for creativity and innovation
• But possible errors due to inappropriate stereotypes, extrapolations, etc.
A Continuum, not a Dichotomy

BLOOM’S REVISED TAXONOMY

Creating
Generating new ideas, products, or ways of viewing things
Designing, constructing, planning, producing, inventing.

Evaluating
Justifying a decision or course of action
Checking, hypothesising, critiquing, experimenting, judging

Analysing
Breaking information into parts to explore understandings and relationships
Comparing, organising, deconstructing, interrogating, finding

Applying
Using information in another familiar situation
Implementing, carrying out, using, executing

Understanding
Explaining ideas or concepts
Interpreting, summarising, paraphrasing, classifying, explaining

Remembering
Recalling information
Recognising, listing, describing, retrieving, naming, finding
Cynefin Dynamics

Divergent & Convergent Thinking

The Software Revolution

- Impossible or impractical machines now feasible
- Design (or Requirements) can be changed without retooling or remanufacturing
- Design is separated from physical realization
  - Design becomes abstract concept
  - Process is separated from other disciplines

\[
\text{General Purpose Machine} + \text{Software} = \text{Special Purpose Machine}
\]

- Heuristics:
  - “Software is never finished”
  - “Software never costs less”
Software’s Role in a System

• **SCALE** - Generation, storage, manipulation and interpretation of large volumes of information

• **SIMPLIFICATION** - Human interfaces that abstract away the underlying hardware scale

• **POLICY** - Control of complex, non-linear systems

• **AUTOMATION** - of operator provided functions

• **AUTONOMY** - Adaption of the system to the behavior of the environment and users

• **ADAPTATION** - Customized user capability and experience
“The essence of a software entity is a construct of interlocking constructs: data sets, relationships among data items, algorithms, and invocations of functions. This essence is abstract, in that the conceptual construct is the same under many representations. It is nonetheless highly precise and richly detailed.” (Fred Brooks)

- The essence of software, 4 differentiating properties:
  1. Complexity (# of states, lack of repetition)
  2. Conformity (to other man made constructs)
  3. Changeability (emergence, infinite life)
  4. Invisibility (intangible, unvisualizable)

A central tenet of classical systems engineering is that all systems can be viewed in hierarchies:

- A system is composed of subsystems that are composed of smaller units
- One person's component is another's system

Object-oriented software construction observes two tenets:

- Hierarchy – via hierarchical types or modules
- Abstraction – via abstract types or classes
The hardware and software design of such systems sits in the realm of best practices.

The system of systems design will require deep understanding of:

- Complexity - swarm behaviors
- Conformity - man-unmanned teaming
- Changeability - emergence
- Invisibility – behaviors observable only in usage
• Software Engineering Body of Knowledge (SWEBOK V3):
  • Computing Foundations
    • Problem solving techniques
    • Algorithms and complexity
    • Abstraction
    • Data structure and representation
  • Software Design and Construction
    • Software structure and architecture
    • Software construction
  • Software Programming
    • Abstraction
    • Information hiding
    • Object-oriented programming

“Great designs come from great designers. Software construction is a creative process.” (Fred Brooks)
• System Engineering Body of Knowledge (SEBOK V1.1):
  • Systems Thinking
    • Problem solving techniques
    • Patterns and complexity
  • System Modeling
    • System modeling concepts =>
  • System Architecture
    • Logical and physical architecture =>
      • System of systems (complexity)
      • Conops & Scenarios (conformity)
      • Business models (changeability)
      • Views & viewpoints (invisibility)

• Computing foundations
• Software design & construction
• Software programming

• Architecture construction: an architecture represents a set of abstracted designs of the system
Specific recommendations:

- Systems thinking
  - Case studies, applied throughout the curriculum
  - Capstone projects
- Systems modeling or Software systems
  - Hierarchy and abstraction
  - Object-oriented design
- Programming languages
  - SysML, UML
- Systems architecture
  - Fundamentals
  - Complexity & Systems-of-systems
  - Business and enterprise
  - Evaluation methods