Truth, Science and Software Engineering: A View from the Trenches

W. Michael McCracken

College of Computing

and

EduTech Institute

Georgia Institute of Technology

A talk given at the Leaders in Software Engineering Education Luncheon
Conference on Software Engineering Education and Training
March 23, 1999

Software

- **Too broad a term to describe anything**
- **Jackson - “The ability to build a machine by simply describing it”**
- **A system that interacts within a context and with humans**
- **Winograd - “A medium for creating virtualities - A world in which the user of the software perceives, acts, and responds to experiences”**

Constructed Artifacts and Intangible Artifacts

- **Architects and Engineers Construct Physical Artifacts for Use**
  - **They are constrained by nature which creates numerous opportunities as well as frustrations**
    - » **Design costs are secondary to construction costs**
    - » **Standard parts, reusable components, trade practices occur easily**
- **Software Engineers Construct Stuff**
  - **They are constrained by nothing but their imagination**
    - » **Design costs dominate**
    - » **Reinventing the wheel is the standard practice**

A Simple View of Software

- **Engineering Software - Software for machines**
  - **The augmentation of human physical ability by means of devices controlled by computation**
    - » **A fuel control system on an automobile**
    - » **A power distribution system**
    - » **A product distribution management system**
- **Human Software - Software for people**
  - **The augmentation of human intellect by means of computation**
    - » **A word processor**
    - » **A decision support system**
    - » **A piece of educational software**

Implications of the Simple Model

- **Engineering Software can be understood, modeled and analyzed using standard engineering methods**
  - **A flight control system, whether hydraulic or fly by wire, implements the same function.**
- **Human Software cannot be understood, modeled or analyzed using standard engineering methods**
  - **Those darn users just don't do what they are supposed to do**

Design of Human Software (Interactive Systems)

- **Is focused on the creation of usable computer artifacts**
- **Is the creation of solutions to ill-structured problems**
- **Is dominated by issues of problem understanding**

Where are we?

- **Software is not harder or easier, just different**
  - **Intangible artifacts**
  - **Context of use is with humans**
  - **Relatively young**
  - **Has no constraints**

Problem

- **The current approaches of teaching engineering (science)/computer science are in conflict with the learning of software engineering**

Truth and Science

- **The teaching and learning of science**
  - **Science is the accumulation of factual knowledge about the universe we live in**
  - **Science learning is about how to assimilate the facts of science, and how to create those facts**

Engineering Science

- **Early engineering was experimental and heuristic**
- **Engineering science is the application of scientific principles to the design of artifacts**
- **A body of factual knowledge guides engineering design**
- **Methods of teaching engineering science follow the model of science education**

Computer Science

- **By name - An attempt to attach science to the generation of computational artifacts**
- **By discovery - The accumulation of factual knowledge about computing (proofs of correctness, complexity theory, turing machines, unified models of cognition, etc)**
- **By reaction - To deflect criticism at a young field**
- **The teaching of computer science follows the method of science education**

A Sidetrip into Education

- **Most university education continues the theme of the acquisition of factual knowledge (learn science not by how it is discovered, but by studying the results of the discoveries, I.e, factual knowledge)**
  - **Individual doses of deconstructed facts**
  - **Students are expected to integrate the facts on their own**

Models of Designing

- **A means of describing the activities of designers**
  - **By describing the activities of experts and novices we may be able to understand how to teach design.**
- **There are many studies of design in architecture, engineering, and a few in software development we can take advantage of**

Models of Software Designing

- **Jeffries , Turner, Polson, and Atwood**
  - **Schema based**
  - **Novice versus Expert Designing**
- **Adelson and Soloway**
  - **Goals and Operators of Design**
- **Guindon and Curtis**
  - **Schema based**
  - **Design Breakdowns**

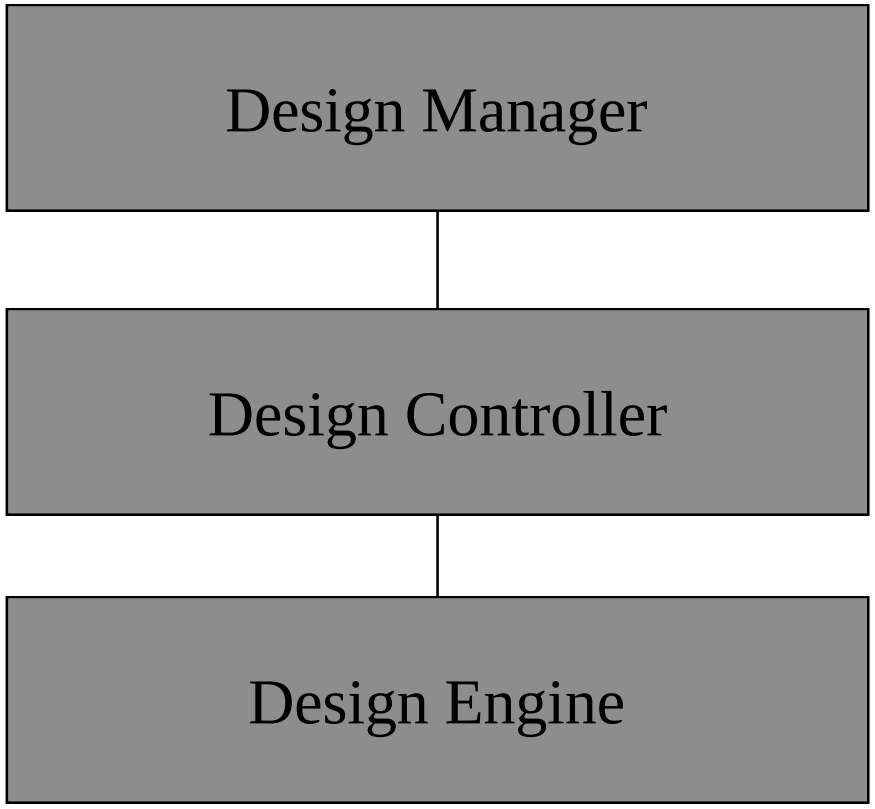
What these models tell us

- **Software designers are similar to other designers**
  - **Generation of alternatives**
  - **Evaluation of alternatives against constraints**
  - **Iteratively refine problems and solutions**
  - **Move between multiple layers of abstraction**
  - **Use cases from prior experience**
  - **Have an overall high level control to manage design process**

Deficiencies of the models

- **Don’t explicitly concern themselves with learning to design**
- **Minimally integrate the tools of design (e.g., ac circuit analysis, or algorithm analysis) with design processes**

Another Model of Software Design

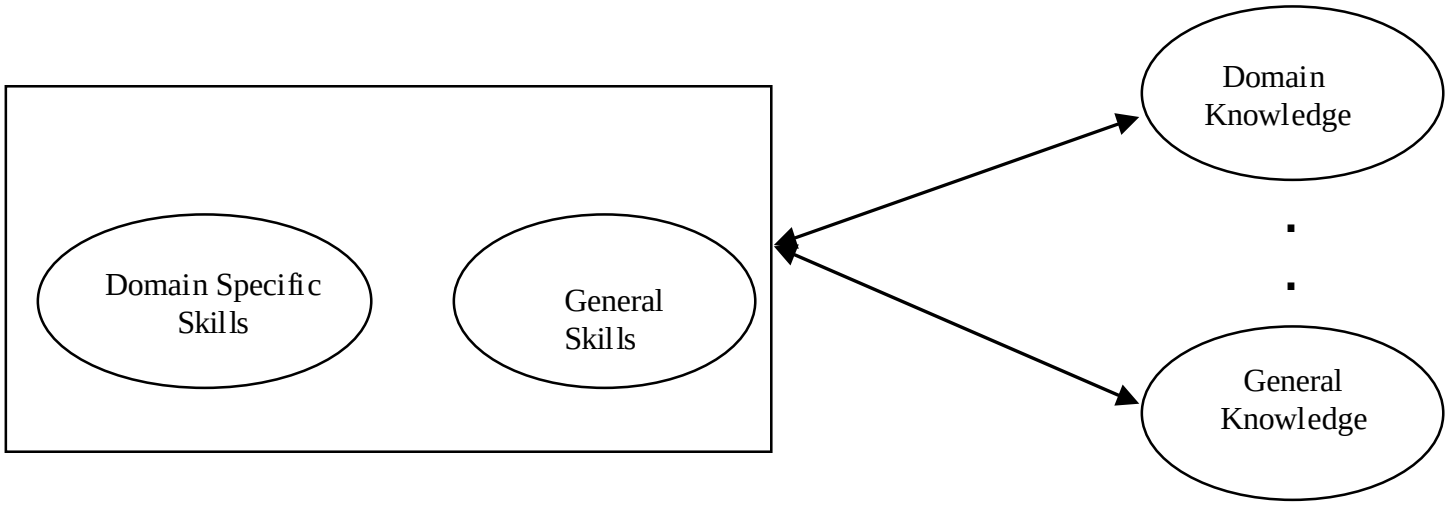


Domain Knowledge
General Design Guidance
Question Asking

Domain Independent Design Invariants
Iteration, generation of alternatives, use of artificial symbol systems, problem formulation, etc.

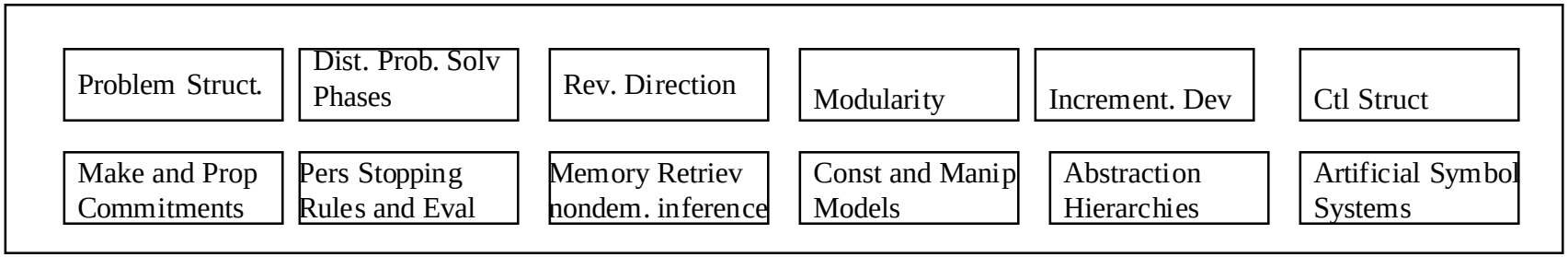
Domain dependent and independent skills, methods, etc. (e.g., math, science, analysis, synthesis)

Design Engine



The Design Engine solves well structured problems
It is developed through typical engineering or computer science education.
It is the tools of designing (math, analysis, physics, etc.)

Design Controller



The Design Controller forms well structured problems from ill-structured problems for the Design Engine to solve
It is developed by actual design activity. Not from solving "toy" problems.

Design Manager

Question Asker

The Design Manager opportunistically controls design activities

Should I consider the economic implications of this approach?
Why don't I explore this previous design that appears related to the current problem
Why don't I try this unrelated path to see if it exposes some element of the problem I am overlooking

The design manager develops primarily through experience, but can be developed by by appropriate problem selection at the university level.

Learning to Design

- **The Engineering Science View (seek the truth)**
  - **A body of factual knowledge guides engineering design. Learn those facts and you can design**
  - **This leads to a robust design engine**
- **The Computer Science View (seek what appears to be the truth)**
  - **Attempts to achieve an engineering science of computing**
  - **This leads to a robus design engine**

Teaching Design

- **University based models of design teaching**
  - **ID/Architecture use immersive techniques**
    - » **Design studios throughout the curriculum interspersed with factual knowledge learning**
    - » **This attempts to initiate development of the design manager, develops a robust design controller, and may develop a less than robust design engine**
  - **Engineering uses capstone models**
    - » **An integrative course at the end intended to tie everything together in a semester**
    - » **This typically adds little to the design controller, and fosters a belief in students that the design engine can solve all design problems**

Teaching Design (cont.)

- **Computer science doesn't know what it uses**
  - » **Most programs have some type of software engineering course that is fact based (survey) but there is little understanding of what design is (algorithm design is not design)**

- **Software engineering uses the engineering model (most existing MSSE programs)**
  - » **Feed them some stuff and wrap it up in a project course**

The Impact of Truth Seeking on Designing

- **A robust Design Engine**
  - **Algorithmic approach to all problems**
  - **Single point solution fixation**
  - **Design stuckness**
  - **Case indices are narrow**
- **Rudimentary or no Design Controller**
- **Design Manager is naive if it exists**

Our Research Tells Us

- **Isolated islands of knowledge are constructed in most design education programs**
- **Capstone courses don’t support construction of design controller and manager**
- **Design is an integrative activity, and should be taught as an integrative activity**
- **Design engines need to be constructed that are accommodating of change**

Conclusion

- **Design uses truth**
- **It doesn't**
  - **Create truth**
  - **Adhere to truth**
  - **Lend itself to mechanisms that prescribe, and measure truthful activities**