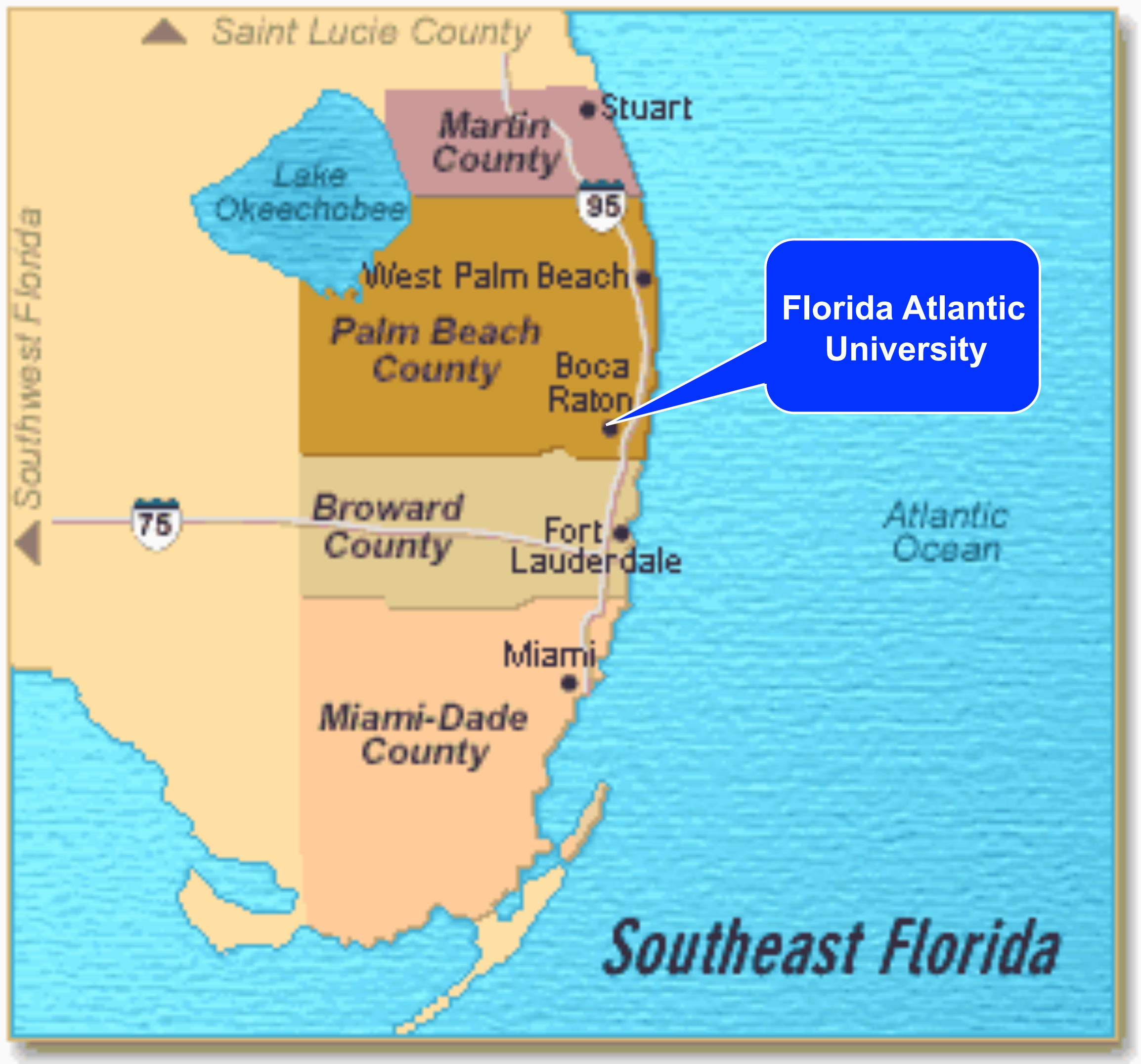
Capturing the Essence of Software Engineering

-- A Reflection on SEMAT Vision Statement

Shihong Huang
Department of Computer Science & Engineering
Florida Atlantic University

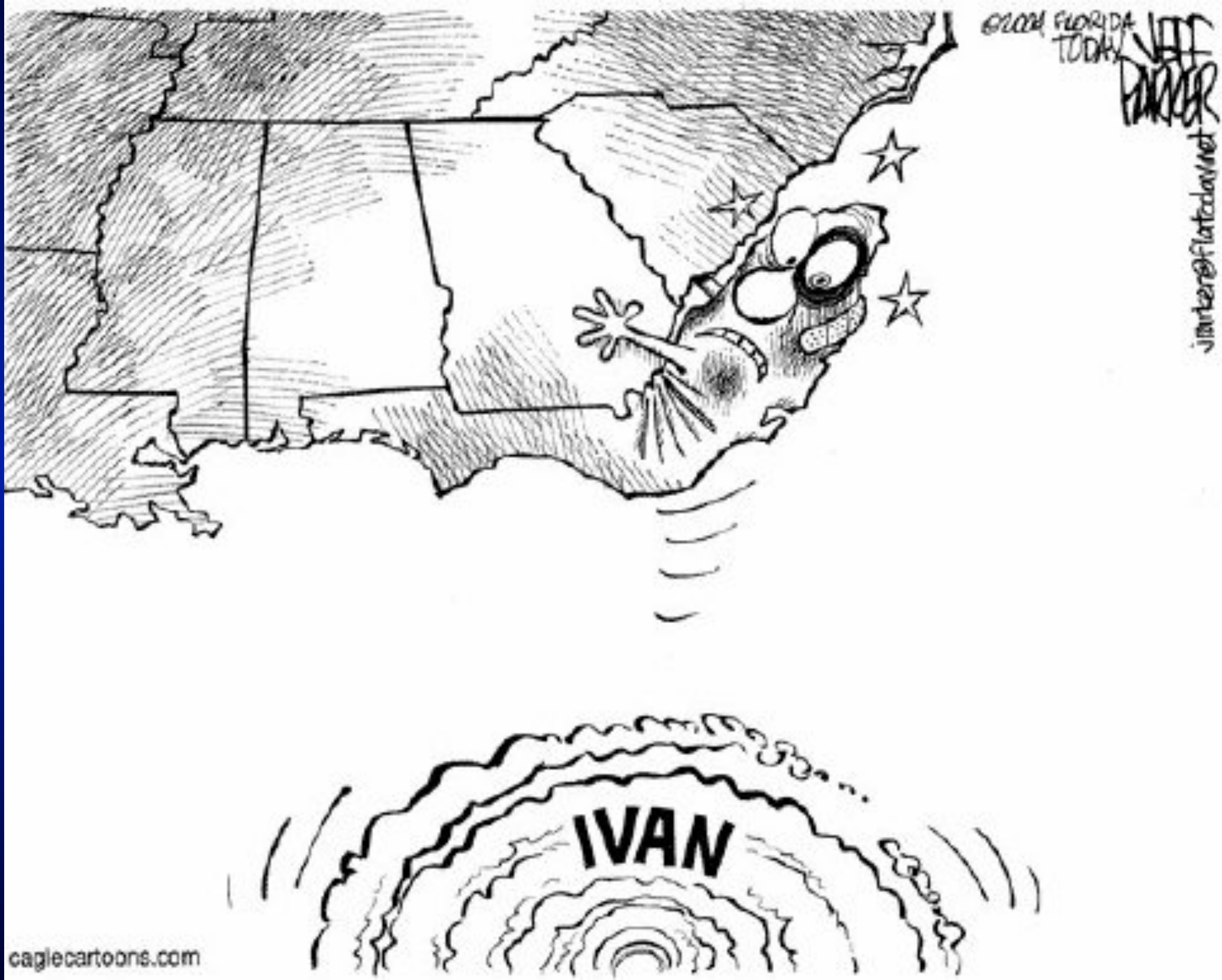
1st SEMAT Workshop
March 17 - 19, 2010 Zurich

Florida Atlantic University on the Map



Florida Atlantic University

Florida: A Different Map



SEMAT Mission

- Refound software engineering based on a solid theory, proven principles and best practices
- Address some of the prevalent problems
  - Prevalence of fad
  - Lack of a sound, widely accepted theories
  - Large number of methods and variants
  - Need of credible empirical evaluation and validation
  - Gap between industry and academia

SEMAT Goals

- Defining the basic definition of software engineering
- Providing a strong mathematical basis
- Identifying the truly *universal* elements
- Defining a kernel language that describes the “method elements” -- practices, patterns, and methods
- Providing assessment techniques evaluating software practice and theories

Definition and Universal

- The goal of the *Universal*
  - Identifying the universal elements of software engineering to be integrated into "*kernel*"
  - In the meantime, "keep the kernel concrete, focused and small"
- Universals and Definitions are mutually tightly coupled
  - Definition defines the scope of the Universals
  - Universals codify Definition
- Basic understanding of what "software engineering" is
- What the uniqueness of software engineering

What is “Software Engineering”

- We leave this question to Track 1 to answer
- Software Engineering = “Software” + “Engineering”
- “The application of engineering methods and discipline to the field of software”
- Although some question its sufficiency or precision [A. Cockburn]
- Software engineering is indeed an “Engineering” discipline, it should be treated the “engineering way”

Difference between Science and Engineering

- Science seeks to understand what is, whereas
- Engineering seeks to create what never was

--- [Henry Petroski 2010]

- It is not appropriate to describe engineering as mere applied science
- Some extra-scientific components to engineering:
  - Creative nature
  - Situated culture particularity to a specific application domain

Difference between Science and Engineering

- When defining “software engineering” and the “Universals”
- It is essential to keep in mind the similarities and differences between science and engineering
- Science
  - Deals with the universal laws
  - Context and time independent and true everywhere
- In engineering
  - Analysis follows synthesis and observation
- Engineering
  - Situated culture
  - Needs to have constant learning, refinement and adaptation to meet the environmental requirements

Difference between Science and Engineering

In engineering

Analysis follows synthesis and observation

Not the other way around

Uniqueness of Software

- While software engineering follows the engineering fundamentals

Some unique features of

software engineering and software products

vs.

General engineering and engineering products

Engineering model vs. Software Model

- Full specification
- Design
- Manufacture
- Test
- Install
- maintain

- Incomplete specification
- First three stages are often blurred
- Final product is intangible
- Doesn’t wear out

The Malleable Nature of Software

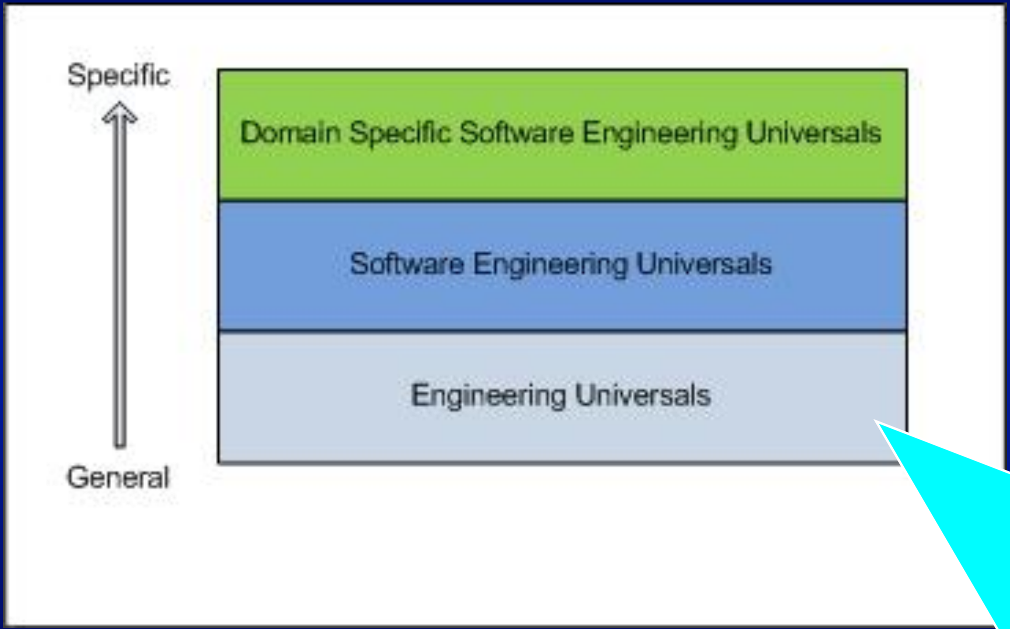
- Evolution is more important in software than in other engineering disciplines
- Software engineering rarely involves "green field" development
- Software needs to be constantly maintained and evolved to meet new business requirements
- The cost incurred in evolution usually exceed the development cost by a factor of 3 or 4

An Observation of *Universals*

- Given the malleable nature of software, a good collection of Universals should
  - Include general engineering universals that capture the core practices of engineering disciplines
  - Unique features of software
- From general to specific
- Approach should be continuum and continuum should be respected
- Not everything must be *universal* or that everything must be situation specific

A Hierarchical Structure of Universals

- Layer 1: the “engineering” aspect


Specific
Domain Specific Software Engineering Universals
Software Engineering Universals
Engineering Universals
General

Best practices of engineering discipline applicable to software:

Project:

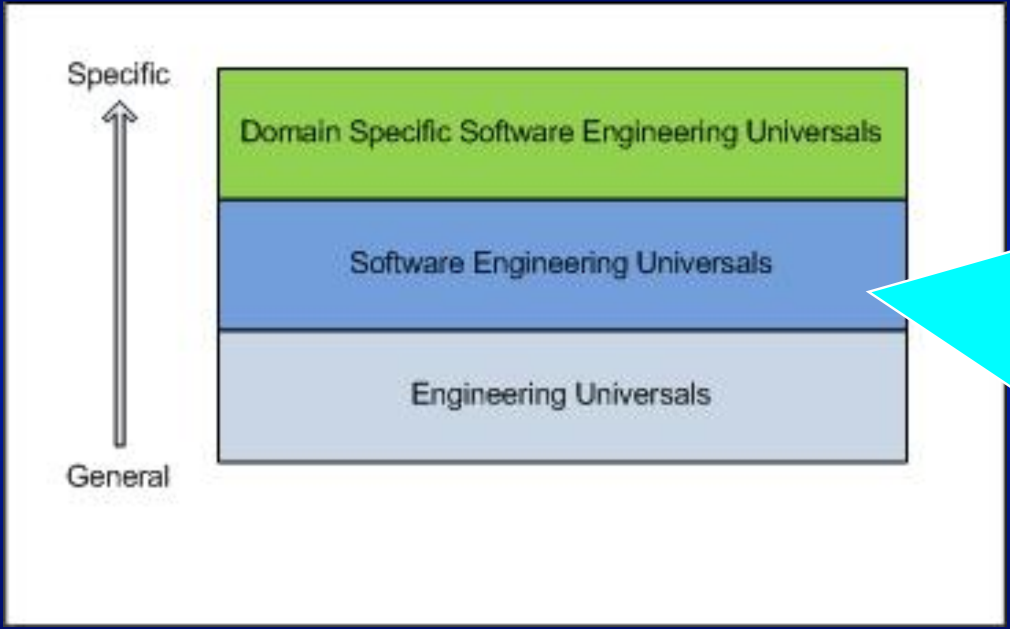
- **Transformation**
- **Flow**
- **Value generation**

Management

- **Planning**
- **Execution**
- **Controlling**

A Hierarchical Structure of Universals

- Layer 2: the “software” aspect

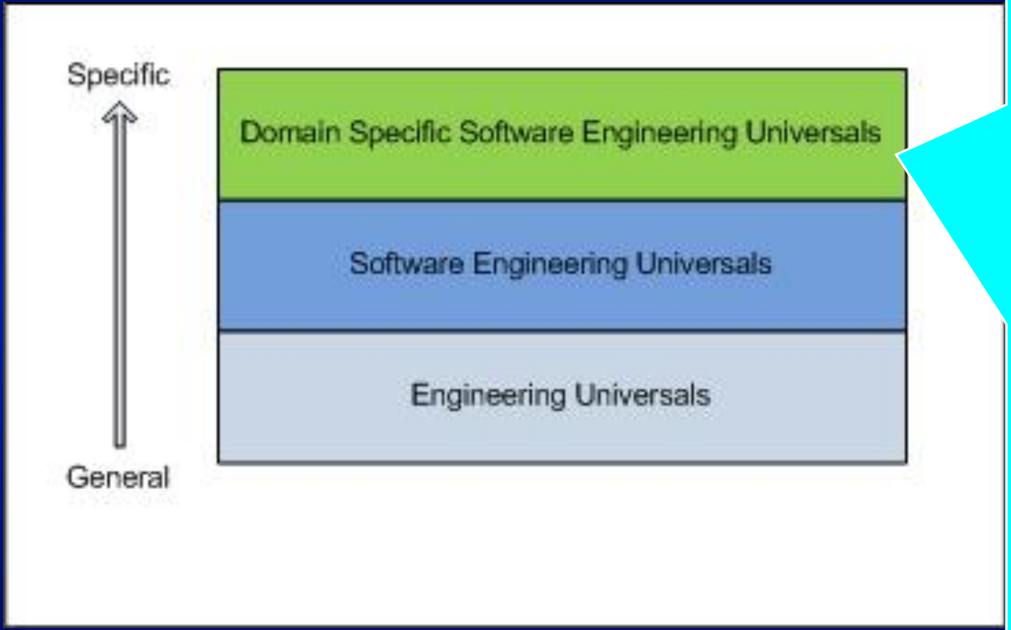

Specific
Domain Specific Software Engineering Universals
Software Engineering Universals
Engineering Universals
General

Unique practices to software:

- **Extensibility**
- **Interoperability**
- **Evolveability**
- **Reusability**
- **Maintainability**

A Hierarchical Structure of Universals

- Layer3: “variability” -- situated culture


Specific
General
Domain Specific Software Engineering Universals
Software Engineering Universals
Engineering Universals

Reflect and address the knowledge of different more situated application domain

- **Real-time systems**
- **Self-adaptive systems**
- **Self-management systems**
- **Web systems**
- **… more**

Software Engineering: A University Perspective

- Poorly perceived: “anyone can teach it”
- Scarcely founded (e.g., Federal and States)
- Challenging Quality publications

Prevalence of fads -- “acronym soup”

Moving Targets

References

- Software Engineering Method and Theory (SEMAT) online at www.semat.org
- SEMAT Vision Statement online at http://www.semat.org/pub/Main/WebHome/SEMAT-vision.pdf
- A. Cockburn. The end of software engineering and the start of economic gaming. http://alistair.cockburn.us/The+end+of+software+engineering+and+the+start+of+economic-cooperative+gaming).
- I. Jacobson, P. W. Ng, I. Spence "Enough of Processes - Lets do Practices" *Journal of Object Technology*, Vol. 6, No. 6, July -August 2007
- Henry Petroski The Essential Engineer: Why Science Alone Will Not Solve Our Global Problems, Knopf, February 23, 2010
- L. Koskela, Lauri and G. Howell. The underlying theory of project management is obsolete.
- http://www.leanconstruction.org/pdf/ObsoleteTheory.pdf
- H. Müller, J. Jahnke, D. Smith, M-A. Storey, S. Tilley, and K. Wong. "Reverse engineering a roadmap." International Conference on Software Engineering Proceedings of the Conference on The Future of Software Engineering (ICSE 2000: Limerick, Ireland). pp. 47-60.
- I. Jacobson, M. Griss, and P. Jonsson. Software Reuse: Architecture, Process and Organization for Business Success. Addison Wesley Professional June 1, 1997.