SEMAT

Cleaning up the Confusion, Jargon and Fads of Software Engineering

Chuck Connell, chc-3.com

Introduction

- Software design/development consultant in Bedford, MA
- Lotus/IBM previous to consulting
- Instructor of software engineering at BU
- Author of recent book *Beautiful Software*
  - Topics related to this talk and other issues
  - Two giveaways

Outline

- A problem in software engineering
- What SEMAT is and how they are trying to solve the problem
- An example from my work, in the spirit of SEMAT
- Analysis: SEMAT successes and warnings

A quote....

- "Software engineering is gravely hampered today by immature practices. Specific problems include:
  - The prevalence of fads more typical of fashion industry than of an engineering discipline.
  - The lack of a sound, widely accepted theoretical basis.
  - The huge number of methods and method variants, with differences little understood and artificially magnified.
  - The lack of credible experimental evaluation and validation.
  - The split between industry practice and academic research. "
- Do you agree? Feedback from roundtable?

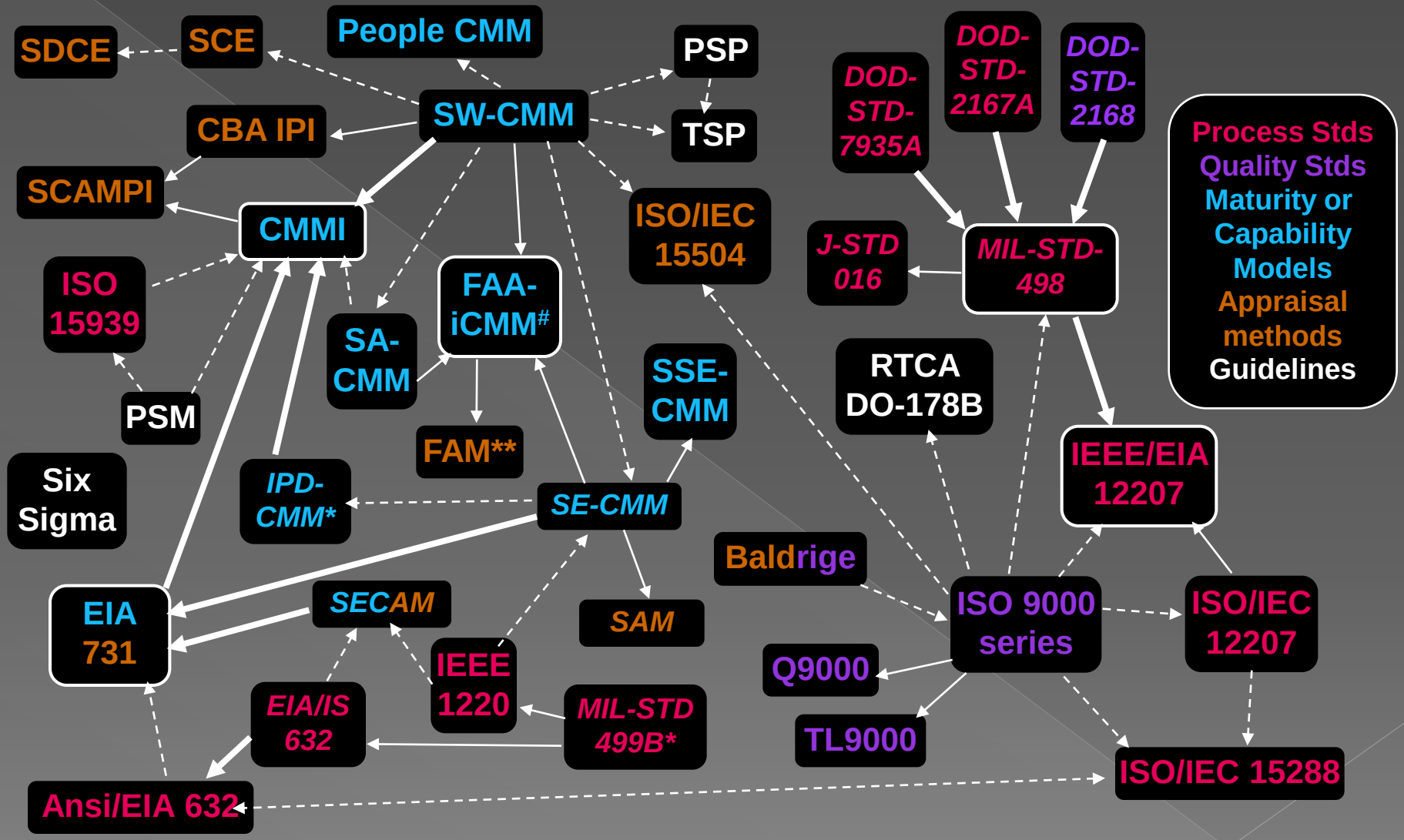
I agree…

- Fads
  - Structured A/D (1980), CMM (1990), object oriented A/D (1995), open source (2000), agile (2005), more…
- Freebie: who wrote seminal book on SD?
- Method overload
  - CMM vs. ISO 9126 RAD vs. agile
  - Key similarities, differences?
- Lack of theory
  - Why does refactoring work?

A Scary Picture

- Credit to Sarah Sheard in *Evolution of the Frameworks Quagmire*.
- 2001/2003, but still relevant

The Frameworks Quagmire



So What is SEMAT?

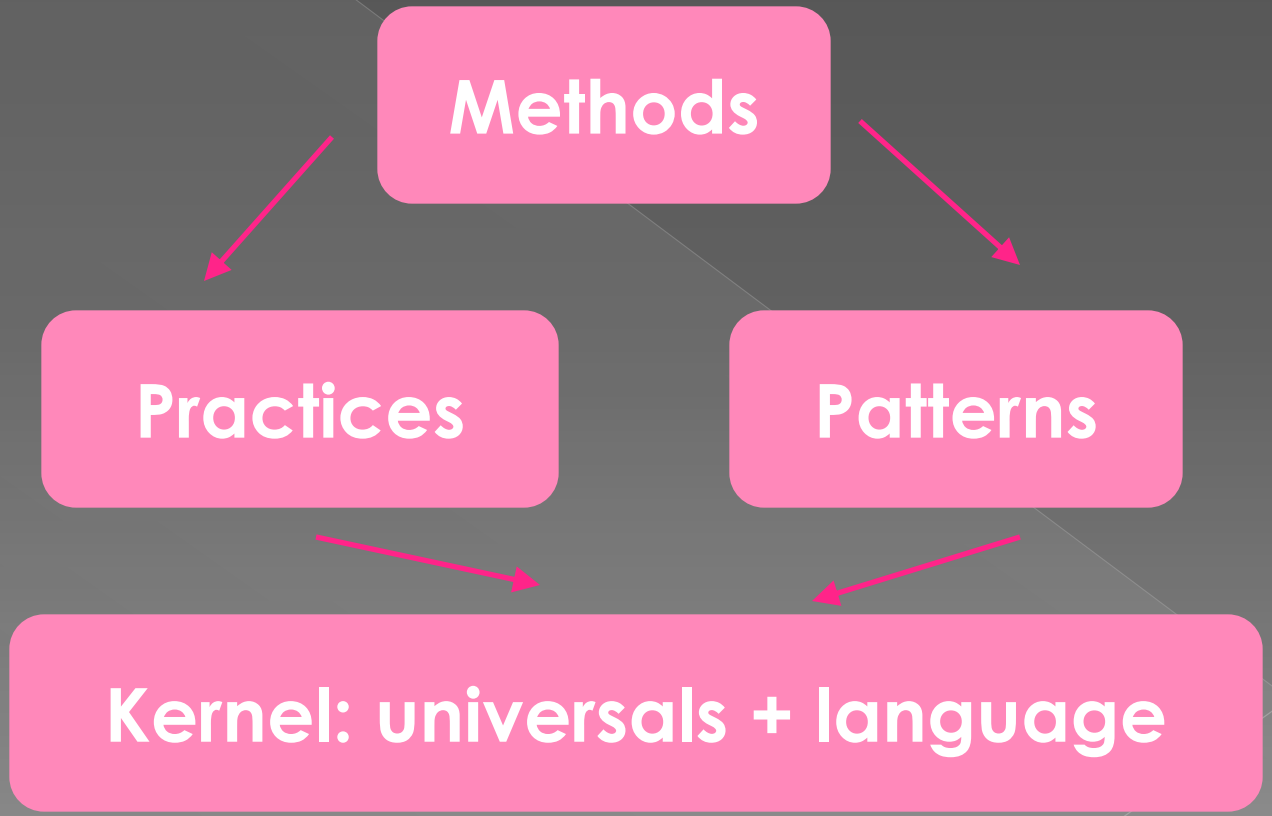
- Software Engineering Method and Theory
- Organization dedicated to fixing these problems
- Started in 2009 with 3 articles in DDJ by Ivar Jacobson, Bertrand Meyer and Richard Soley
- Now 30 famous people + 15 major institution “signatories”, and 1600 “supporters”

SEMAT Mission

- "We support a process to re-found software engineering based on a solid theory, proven principles and best practices that:
  - Include a kernel of widely-agreed elements, extensible for specific uses
  - Addresses both technology and people issues
  - Are supported by industry, academia, researchers and users
  - Support extension in the face of changing requirements and technology"

An Improved Picture

All of SWE expressible from…



This Would Be a Big Deal!

- Describe all current SWE methods with a common language and concepts
- Know that ABC method is a superset of XYZ
- Know that ISO-1234 just a restatement of CMM-AB
- Easily describe new process for new situation
- Like discovery of DNA and the A-T-C-G language of genetics!

Example: Refactoring

- But what could SEMAT look like in practice?
- A possible example, from my own work…
- Goal is universal sw design principles
  - Could serve as foundation for all analysis/design methods
- Freebie: what is source code refactoring?
  - Hint: two key aspects

Refactoring Example

```
temp = 2 * (_height + _width); 'perimeter
System.out.println (temp);
temp = _height * _width;    'area
System.out.println (temp);
```

- What is the problem(s) with this code?
- Solution: Split Temporary Variable

```
perimeter = 2 * (_height + _width);
System.out.println (perimeter);
area = _height * _width;
System.out.println (area);
```

So What?

- Programmers have been tweaking code since 1950.
- Disciplined, correct refactoring has at least three benefits.
  - Successive, small changes can produce BIG improvements.
  - Lightens the load on design phase.
  - More realistic design phase.
  - (Latter two support agile methods.)

But Refactoring is Ad-Hoc

- Unanswered questions…
  1. When should you refactor?
     - When is source code "not good" so it needs improvement?
  2. Which refactoring method to use?
     - At least 70, several for each case.
     - Some contradict each other.
  3. Why is the change an improvement?
     - No explanation for what is happening.

Current Answers No Good

- Summarizing thousands of pages of research…
  1. A section of source code should be refactored when it "smells bad."
  2. We should apply the refactoring that helps with this smell.
  3. No one knows.
- We need a *theory* of refactoring.
  - What is refactoring?
  - Why does some code smells bad?
  - Why does refactoring make code better?

My Proposal

- 7 universal principles of good sw design
  - **Cooperation**. Work well with its surrounding environment.
  - **Appropriate form**. Form follow function.
  - **Minimality**. As small as it can be.
  - **Singularity**. Contain one instance of each component.
  - **Locality**. Place related items together.
  - **Visibility**. Built-in clarity plus comments.
  - **Simplicity**. Solve its problems in the simplest manner possible.

A Theory of Refactoring

- This theory answers the open questions
  1. You should refactor when one/more of the 7 tenets are broken.
  2. Use the transformation that most easily reestablishes good design where it is currently broken.
  3. Refactoring works by bringing software more in line with 7 principles.

What the Theory Give Us

- There are not 70 transformations, there are only 7!
  - The 7 can be combined in various ways.
  - By Occam's Razor, this is *much* better.
- In the spirit of chemistry and physics.
  - Substances → elements → particles.
- Predicts new transformations.
  - The best way to test any theory.

As With Any Theory…

- Needs evidence and arguments.
- Could be improved over time.
- But is within the spirit of SEMAT by offering an overall theory of software design .

SEMAT Successes

- There is a clear problem.
  - Solution would obviously be useful
- Many important people are behind the effort
  - Lots of "working together"
- Have had 3 int'l conferences, each with report
- Broken into 6 tracks
  - Requirements
  - Universals
  - Assessment
  - Theory
  - Kernel language
  - Definitions
  - Architecture (spike)

SEMAT Successes

- Goal is to improve *practice* not just create abstract results
- Foundation (kernel + language) acceptance transferred to OMG in June 2011
  - So SEMAT is not voting on its own work
  - SEMAT is now one org that can propose solutions for problem it defined
- 20 people working on kernel since March 2010.
  - 8 universals proposed: opportunity, stakeholder community, requirement, software system, work, team, method, practice
- Want to remove split between process nazies and programmers
  - Good!

SEMAT Problems

- Lots of discussion, few results
  - 1p problem statement → 20p vision → 44p RFP
- RFP actually a step backwards
  - It is a "request for" a result, not a proposal
- Could become more jargon on top of existing jargon

… a "method" must be *enactable,* while a "practice" in isolation will in general not be. In the context of this RFP, the enactment of a method can be defined as the carrying out of that method in the context of a specific project effort. Within this context, the practices within the method may be considered use cases for the work that must be carried out to achieve the project objectives, with each practice providing a specific aspect of the overall method.

SEMAT Problems

- Need to watch for agile bias
  - Agile is FOTM, something else in 2020
  - SEMAT must define earlier flavors and next
- Need to watch for cult of personality
  - Trumpet names/# of famous signatories
  - But science is about evidence, prediction, internal consistency; doesn't matter who says it
  - History of science shows famous people are wrong about the next breakthrough

SEMAT Problems

- This has all been tried before!
  - System Process Engineering Meta-Model (SPEM)
  - ISO/IEC 24744
  - Eclipse Process Framework
  - Software Engineering Body of Knowledge (SWEBOK)
  - Unified Method Framework
  - More….
- The RFP says each is inadequate
  - Worth a raised eyebrow though

Questions / Comments

- Questions?
- Comments?

For more information…

- SEMAT home page
  - semat.org
- SEMAT vision statement
  - www.semat.org/pub/Main/WebHome/SEMAT-vision.pdf
- SEMAT blog
  - sematblog.wordpress.com/
- My home page
  - chc-3.com
- My book, including these issues
  - www.amazon.com/dp/1456438786/