Speeding-Up
Software Engineering's
Escape from Its
Pre-paradigmatic Stage

Iaakov Exman

laakov <@> jce.ac.il

May 2013

Copyright by Iaakov Exman (c)
Status

... of **Software Engineering** theory

schwarzwald tårta

One recognizes instances of the chocolate polygon class, but overall the cake is not reusable...
Goal

Neurons

Na+, K+  Diffusion equations

Biology

Physics

Formal theories in well-separated layers.
1- From Kuhn to Escape Velocity
2- Four \textit{NOT} Theorems
3- Crisp Summary
4- \textbf{Separation of Concerns}
5- Preview of a \textit{True Paradigm}…
From Pre-Paradigm to a true Paradigm

Away from competing alternatives

Open ended → significant problems to be solved
Wrong, mistaken, undesirable notions… leave behind, what is **NOT** Software Engineering.
Four **NOT** Theorems

1- **NOT** Static code **but** Running Software
2- **NOT** Machines **but** Abstract Systems
3- **NOT** Algorithms **but** Protocols
4- **NOT** Logic **but** more Expressive Math
**Theorem 1**

*Not* Static code but Running Software

Changing the software’s **structure** should change the software’s **behavior**, therefore **static code** is *not* software.
By Counter-Example

Software obfuscation $\rightarrow$ changes code structure without changing behavior, making code understanding difficult.
Running software is actual software.

allows:

• Measurements
• Understanding (debugging, agility, …)

The main purpose for running software is to gain understanding of a problem.
Machines are limited by their resources, while software systems are given by concepts; therefore Software systems are not defined by machines.
NOT T-2

NOT Machines but Abstract Systems

Proof Outline

Turing machine
= prototype machine;
limiting resources tape
  table – of instructions

UML = prototype software system; by requirements

Resources → may be insufficient/exact/superfluous for requirements
thus purely incidental to software system.

May 2013
Copyright by Iaakov Exman (c)
NOT Machines but Abstract Systems

Implications

No reductionist principle from software systems to computing machine resources.

Particular Software systems cannot be deduced from Universal machine specifications.
Algorithms have a **halting state**, while software protocols do not; therefore the **halting problem** is **not** an important Software systems’ problem.
**NOT T-3**

**NOT Algorithms but Protocols**

**Proof Outline**

By Software *non-halting counter-examples:*

- **large software systems:** e.g. airline reservation
- **simulations** of non-halting systems: elevators

- Software *protocols* have message sequences but no halting state.

**run forever** → servers listen indefinitely for client requests
There are important problems for theories of algorithms, e.g. the halting problem, which are not important problems for the theory of software and vice-versa.

One should actively search and solve important software engineering problems.
Logic has numerous operators acting on simple structures, while software has complex structures; therefore logic is not an expressive enough language to describe software.
Choose a measure of structure complexity.
Apply it to representative logical structures, and software structures.

The ratio of the logical to software structure complexity measures is an expressiveness index.

If the ratio is much less than 1, the theorem is proved.
Not T-4

Not Logic but more Expressive Math

Implications

Mathematical logic is not specifically suitable for reasoning about software, no more than for other engineering disciplines.

Attempts to use logic for software structures added and modified Logic:

- Z language: ad hoc schema, very restricted;
- Temporal logic: applied to model checking.
Crisp Summary

**NOT theorems** rephrased:
1. Static code is not software;
2. Software systems are not defined by machines;
3. The halting problem is not an important software systems’ problem.
4. Logic is not the main language for software.

Essentially: software engineering has concerns differing from computer science.
No claims that computer science as a discipline is not important for software engineering. We advocate a separation of concerns; i.e. important algorithmic problems should remain within computer science, and vice-versa.

Neuron signals have inherent diffusion. But diffusion equations are not laws of biology. They remain laws of the underlying physics.
Separation of Concerns

Software Systems

Turing Machines

Latest Fashion

Copyright by Iaakov Exman (c) 2013
Preview of a True Paradigm

**SOFTWARE:**

**Coupling**

is just

Linear Dependence

**Linear Software Models**

YouTube VIDEO Lecture ➔ [http://www.youtube.com/watch?v=EJfzArH8-Is](http://www.youtube.com/watch?v=EJfzArH8-Is)
Speeding-Up
Software Engineering's
Escape from Its
Pre-paradigmatic Stage

Iaakov Exman

iaakov <@> jce.ac.il

@laakovExman