Outline

- Introduction
- Past and Present
- Future
- References
Introduction

- Software: intangible, malleable, fragile
- Decade-wise evolution of SE
- Timeless principles (+)
- Aging practices (-)
- Hegelian view

Thesis → Antithesis

Synthesis
Past and Present
Boehm’s Definition

Software Engineering is

“the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them.” [1976]

“the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation.” [1979]
Boehm’s Hypothesis

“Software people do not like to see software engineering done unsuccessfully, and try to make things better.”
1950s

- Thesis: SE was like HE
- Engineering notebooks kept
- Practiced hardware precepts e.g. “measure twice, cut once”
1960s

- Software is malleable
- S/W & H/W maintenance are different
- Antithesis: code-and-fix
- Demand > Supply
- Hacker culture, cowboy programmers
- CS departments & S/W houses emerge
1970s

- Antithesis: structured approach
- Waterfall processes
- Formal methods
- Quantitative approaches
  - Complexity metrics
  - Estimation models
- Reliability
- Cost and Schedule
1980s

- Synthesis: productivity and scalability are essential
- Process compliance via maturity models: S/W-CMM & ISO-9001
- OOD
- Software tools: CM, CASE
- Software processes (are S/W too)
- No silver bullet: Accidental vs. Essential tasks
- Payoff from reuse: OO methods (UML)
1990s

- Antithesis: shrinking cycle time is important
- Concurrent processes e.g. spiral model
- Reuse: COTS and open source
  - (top-down) requirements-to-capabilities
  - (bottom-up) capabilities-to-requirements
- Emphasis on user needs
  - IKIWISI
  - Emergent vs. prespecifiable requirements
  - Golden vs. Platinum rule
2000s

- Partial synthesis: agility & value
- Speedup process
- Agile development e.g. XP
- Value-based Software Engineering (VBSE)
- S/W criticality and dependability
  - Vendor warranties
  - Accountability
- COTS, open source, and legacy S/W
  - Problems: usability, dependability, interoperability, localizability
- MDD
  - Domain models
  - Enterprise Architectures
- Interacting S/W and Systems Engineering
  - Integrate application-domain and software-domain models
  - CMMI
Timeless Principals and Ageing Practices

+ Do not neglect the sciences
+ Look before you leap
- Avoid using a rigorous sequential process

+ Think outside the box
+ Respect software’s differences
- Avoid cowboy programming

+ Eliminate errors early
+ Determine the system’s purpose
- Avoid top-down development and reductionism

+ There are many roads to increased productivity
+ What is good for products is good for process
- Be skeptical about silver bullets

+ Time is money
+ Make software useful to people
- Be quick, but do not hurry

+ If change is rapid, adaptability trumps repeatability
+ Consider and satisfy all of the SCS’s value propositions
- Avoid falling in love with your slogans

6/15/2008  LUMS
21st Century

- Software century
- Software
  - Necessary capabilities
  - Quality of life
  - Competitive differentiation
- 5 challenges for Software Engineers
1. Rapid Change

- Change
  - “the” only constant
  - Rapid
  - Unpredictable
  - Generally, fruitful but disliked

- Avoid
  - THWADI
  - Contract structures: fixed-price, build-to-specification

- Separate obsolete practices from enduring principles
2. Uncertainty and Emergence

- Rapid change
  - Predictable e.g. Moore’s Law
  - Unpredictable e.g. new technologies (e.g. WWW, GPS), desired UI characteristics (IKIWÍSI)

- Avoid sequential waterfall-like model
- Need agility and adaptability
- Use BITAR
  - Narrows Cone of Uncertainty
  - Examples: prototypes, COTS evaluation
Cone of Uncertainty

Phases and Milestones

- Feasibility
- Plans/Rqts.
- Design
- Develop and Test
- Operational Concept
- Life Cycle Objectives
- Life Cycle Architecture
- Initial Operating Capability

Relative Size Range

- 4x
- 2x
- x
- 0.5x
- 0.25x
3. Dependability

- Need both agility and dependability
- Definition of dependability
  - Subjective: MTBF or system uptime (NASA)
  - Universal: DAVAS
- Identify SCS and their value propositions
- Ensure Win-Win situations
  - Requires understanding of domain knowledge
- Example: Model clashes in MasterNet
### MasterNet: Model Clashes

<table>
<thead>
<tr>
<th><strong>Users</strong></th>
<th><strong>Acquirers</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Many features</td>
<td>Mission cost/effectiveness</td>
</tr>
<tr>
<td>Changeable requirements</td>
<td>Limited development budget, schedule</td>
</tr>
<tr>
<td>Applications compatibility</td>
<td>Government standards compliance</td>
</tr>
<tr>
<td>High levels of service</td>
<td>Political correctness</td>
</tr>
<tr>
<td>Voice in acquisition</td>
<td>Development visibility and control</td>
</tr>
<tr>
<td>Flexible contract</td>
<td>Rigorous contact</td>
</tr>
<tr>
<td>Early availability</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Maintainers</strong></th>
<th><strong>Developers</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of transition</td>
<td>Flexible contract</td>
</tr>
<tr>
<td>Ease of maintenance</td>
<td>Ease of meeting budget and schedule</td>
</tr>
<tr>
<td>Applications compatibility</td>
<td>Stable requirements</td>
</tr>
<tr>
<td>Voice in acquisition</td>
<td>Freedom of choice: process</td>
</tr>
<tr>
<td></td>
<td>Freedom of choice: team</td>
</tr>
<tr>
<td></td>
<td>Freedom of choice: COTS/reuse</td>
</tr>
</tbody>
</table>

**Abbreviations:**
- PC: Process
- PD: Product
- PP: Property
- S: Success
4. Diversity

- Cross-cultural globalization
  - Importance of cultural context
- Avoid value-neutral OSUFA
  - Detrimental to SISOS
    - scalability + agility + dependability
5. Interdependence

- TANIA
  - SISOS are ecosystems
    - Internal and external ecologies
  - Applies to processes, methods, and tools
  - Concurrent engineering of software, hardware, and human factors

- Necessary considerations
  - Interoperability
  - Network security
Conclusions

- Future software
  - Sensing, communication, decision support
  - Address personal, local, and global issues
- Software done well
  - Golden Age
- Software done poorly
  - Exacerbate tensions and mistrust
  - Obfuscate mutual understanding
- Software not a silver bullet
  - Conway’s Law
- Key: Incrementally improve software development
- 21st Century Software Engineer – rewarding and challenging profession
References