Software Engineering Strengths and Weaknesses in Systems Engineers

Dr. Paul Shebalin, Director
The Wayne E. Meyer Institute, Naval Postgraduate School
pshebalin@nps.edu, 831-656-1047

“Applied and Basic Research in Systems Analysis, Modeling and Engineering”
Topics

• SE at the Naval Postgraduate School
• SE4003 Systems Software Engineering
• The Need for Systems Engineers to Serve as Software Engineers
• Software Engineering Capability Evaluation
• Observations
• Recommendations
Mission of the Naval Postgraduate School

• NPS provides high-quality, relevant and unique advanced education and research programs that increase the combat effectiveness of the Naval Services, other Armed Forces of the U.S. and our partners, to enhance our national security.
NPS Summary

• A Department of the Navy graduate school founded in 1909 located in Monterey, California
• Four schools and 65 Curricula in engineering, science, business, public policy, operations research, information sciences, international studies, national security studies and homeland security.
• Four research institutes and 20 centers
• Faculty: 248 Tenure-Track, 429 Non-TT
• Graduate Students: 1500 Resident, 750 Non-Resident
  – 44% Navy, 12% USMC, 23% other US Services, 14% International, 7% Civilian
## NPS Systems Engineering Programs

<table>
<thead>
<tr>
<th>Program Type</th>
<th>Course Code</th>
<th>Duration</th>
<th>Total Courses</th>
<th>Project Type</th>
<th>Cohorts/Year</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE Certificate</td>
<td>282</td>
<td>DL and resident</td>
<td>4 quarters</td>
<td>4 courses</td>
<td>2 cohorts/year</td>
<td>23 students</td>
</tr>
<tr>
<td>MSSE</td>
<td>308</td>
<td>Resident</td>
<td>8 quarters (with embedded refresher)</td>
<td>32 courses</td>
<td>Project</td>
<td>34 students</td>
</tr>
<tr>
<td>MSSE</td>
<td>311</td>
<td>DL</td>
<td>8 quarters</td>
<td>16 courses</td>
<td>Project</td>
<td>314 students</td>
</tr>
<tr>
<td>MSSE</td>
<td>580</td>
<td>Resident</td>
<td>8 quarters (with refresher)</td>
<td>36 courses</td>
<td>Thesis</td>
<td>314 students</td>
</tr>
<tr>
<td>MSSE</td>
<td>721</td>
<td>DL</td>
<td>8 quarters</td>
<td>16 courses</td>
<td>Thesis</td>
<td>43 onboard</td>
</tr>
<tr>
<td>MSSEM (PD21)</td>
<td>721</td>
<td>Resident</td>
<td>8 quarters</td>
<td>16 courses</td>
<td>Thesis</td>
<td>41 students</td>
</tr>
</tbody>
</table>

*AS of 9/9/10*
### MSSE Core Courses

- Resident and non-resident programs share a common nine course core curriculum
- Informed by INCOSE and DOD reference curricula
- DAU Equivalencies
- Burnt orange courses compose the certificate
- Degree requirements met by core, 4 course track, and 3 course project
- P-codes can impose additional requirements

<table>
<thead>
<tr>
<th>Course Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fundamentals of Systems Engineering</td>
</tr>
<tr>
<td>Systems Suitability</td>
</tr>
<tr>
<td>Systems Assessment</td>
</tr>
<tr>
<td>Fundamentals of Engineering Project Management</td>
</tr>
<tr>
<td>Engineering Economics and Cost Estimation</td>
</tr>
<tr>
<td>Capability Engineering</td>
</tr>
<tr>
<td>System Architecture and Design</td>
</tr>
<tr>
<td>Systems Software Engineering</td>
</tr>
<tr>
<td>Systems Integration and Development</td>
</tr>
</tbody>
</table>
SE4003 Systems Software Engineering

• Course objective: *teach students the basic concepts of software engineering and methods for requirements, definition, design and testing of software.*

• Course framework:
  – 10-week quarter
  – Prerequisite: Computer Programming Course
  – Text by Pressman: *Software Engineering: A Practitioner’s Approach (7th Ed.*) (Chapters 1-10, 17-19)
  – Assigned readings and class presentations, exercises and discussions complement hands-on project experience.
  – Embedded System Software Project:
    • Team of 3-5 members
    • Software development for Lego NXT robot using NXC (Not eXactly C) or Java
    • Deliverable and non-deliverable software products
  – Basis for identifying SWE strengths and weaknesses
Hands-On Project Hardware

Robotic Autonomous Vehicle (RAV) Prototype

LEGO NXT Ultrasonic Sensor

LEGO NXT Brick

Two (2) LEGO NXT Servo Motors

LEGO NXT Light Sensor
Systems Engineers as Software Engineers?

• Why do systems engineers need software engineering knowledge, skills, and abilities?
  – To be better systems engineers?
  – To be better software engineers?

• Many systems engineers will be called to serve as software engineers on a software project – development, maintenance, V&V, T&E, …

• How software-engineering-capable are the systems engineers we’re graduating?

• How can we measure SWE capability?
Measuring SWE Capability

- Formal testing
- Peer evaluation
- Process quality
- Product quality
- Timeliness and appropriateness
- Repeatability and long-term performance
- Observation and evaluation
Observation and Evaluation Method

- Define evaluation criteria using
  - "Knowledge, Skills, and Abilities" (KSAs)
  - SWEBOK (IEEE) breakdown of SWE topics
- Consider the results of five offerings of SE4003 with 82 students and 27 project teams
- Subjectively, evaluate the student performance through the use of a Pugh Matrix
  - -1, 0, 1 representing Worse, Same, Better of average student performance compared with that expected of a competent software engineer
KSA’s

• **Skills**
  – Effectiveness in using tools and technology
  – E.g., effectiveness in using a specific software development environment (SDE)

• **Abilities**
  – The wherewithal to perform specific tasks
  – E.g., the ability to create a suitable Software Requirements Analysis Document

• **Knowledge**
  – Mental data representing information, facts, relationships, concepts, logical implications, etc.
  – E.g., object-oriented software engineering concepts
## SWEBOK Topics Breakdown

<table>
<thead>
<tr>
<th>Area</th>
<th>Subareas</th>
<th>Topics</th>
<th>Areas Evaluated</th>
<th>Subtopics</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. SW Requirements</td>
<td>7</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. Software Design</td>
<td>6</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. Software Construction</td>
<td>3</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. Software Testing</td>
<td>5</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. Software Maintenance</td>
<td>4</td>
<td>15</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>F. SW Configuration Mgt</td>
<td>6</td>
<td>17</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>G. SW Engineering Mgt</td>
<td>6</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H. SW Engineering Process</td>
<td>4</td>
<td>16</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>I. SWE Tools and Methods</td>
<td>2</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J. Software Quality</td>
<td>3</td>
<td>11</td>
<td></td>
<td>14</td>
</tr>
</tbody>
</table>
A. Software Requirements

1. Software requirements fundamentals
   a. Definition of software requirement
   b. Product and process requirements
   c. Functional and non-functional requirements
   d. Emergent properties
   e. Quantifiable requirements
   f. System requirements and software requirements

2. Requirements process
   a. Process models
   b. Process actors
   c. Process support and management
   d. Process quality and improvement

3. Requirements elicitation
   a. Requirements sources
   b. Elicitation techniques

4. Requirements analysis
   a. Requirements classification
   b. Conceptual modeling
   c. Architectural design and requirements allocation
   d. Requirements negotiation

5. Requirements specification
   a. System definition document
   b. System requirements specification
   c. Software requirements specification

6. Requirements validation
   a. Requirements reviews
   b. Prototyping
   c. Model validation
   d. Acceptance tests

7. Practical considerations
   a. Iterative nature of requirements process
   b. Change management
   c. Requirements attributes
   d. Requirements tracing
   e. Measuring Requirements
B. Software Design

1. Software design fundamentals
   a. General design concepts
   b. Context of software design
   c. Software design process
   d. Enabling techniques

2. Key issues in software design
   a. Concurrency
   b. Control and handling of events
   c. Distribution of components
   d. Error and exception handling and fault tolerance
   e. Interaction and presentation
   f. Data persistence

3. Software structure and architecture
   a. Architectural structures and viewpoints
   b. Architectural styles (macroarchitectural patterns)
   c. Design patterns (microarchitectural patterns)
   d. Families of programs and frameworks

4. Software design quality analysis and evaluation
   a. Quality attributes
   b. Quality analysis and evaluation techniques
   c. Measures

5. Software design notations
   a. Structural descriptions (static)
   b. Behavioral descriptions (dynamic)

6. Software design strategies and methods
   a. General strategies
   b. Function-oriented (structured) design
   c. Object-oriented design
   d. Data-structure centered design
   e. Component-based design (CBD)
   f. Other methods
C. Software Construction

1. Software construction fundamentals
   a. Minimizing complexity
   b. Anticipating change
   c. Construction for verification
   d. Standards in construction

2. Managing construction
   a. Construction methods
   b. Construction planning
   c. Construction measurement

3. Practical considerations
   a. Construction design
   b. Construction languages
   c. Coding
   d. Construction testing
   e. Construction quality
   f. Integration
D. Software Testing

1. Software testing fundamentals
   a. Testing-related terminology
   b. Key issues
   c. Relationships of testing to other activities

2. Test levels
   a. The target of the tests
   b. Objectives of testing

3. Test techniques
   a. Based on tester’s intuition and experience
   b. Specification-based
   c. Code-based
   d. Fault-based
   e. Usage-based
   f. Based on nature of application
   g. Selecting and combining techniques

4. Test-related measures
   a. Evaluation of the program under test
   b. Evaluation of the tests performed

5. Test process
   a. Management concerns
   b. Test activities
When assigned a software engineering role:

What level of software engineering knowledge would the systems engineer have with regard to a SWEBOK topic?

What level of software engineering ability would the systems engineer have with regard to a SWEBOK topic?
A. Software Requirements: Knowledge

1. Software requirements fundamentals
   a. Definition of software requirement
   b. Product and process requirements
   c. Functional and non-functional requirements
   d. Emergent properties
   e. Quantifiable requirements
   f. System requirements and software requirements

2. Requirements process
   a. Process models
   b. Process actors
   c. Process support and management
   d. Process quality and improvement

3. Requirements elicitation
   a. Requirements sources
   b. Elicitation techniques

4. Requirements analysis
   a. Requirements classification
   b. Conceptual modeling
   c. Architectural design and requirements allocation
   d. Requirements negotiation

5. Requirements specification
   a. System definition document
   b. System requirements specification
   c. Software requirements specification

6. Requirements validation
   a. Requirements reviews
   b. Prototyping
   c. Model validation
   d. Acceptance tests

7. Practical considerations
   a. Iterative nature of requirements process
   b. Change management
   c. Requirements attributes
   d. Requirements tracing
   e. Measuring Requirements

<table>
<thead>
<tr>
<th>Strength</th>
<th>Weakness</th>
</tr>
</thead>
</table>

October 2010 v4
A. Software Requirements: Ability

1. Software requirements fundamentals
   a. Definition of software requirement
   b. Product and process requirements
   c. Functional and non-functional requirements
   d. Emergent properties
   e. Quantifiable requirements
   f. System requirements and software requirements

2. Requirements process
   a. Process models
   b. Process actors
   c. Process support and management
   d. Process quality and improvement

3. Requirements elicitation
   a. Requirements sources
   b. Elicitation techniques

4. Requirements analysis
   a. Requirements classification
   b. Conceptual modeling
   c. Architectural design and requirements allocation
   d. Requirements negotiation

5. Requirements specification
   a. System definition document
   b. System requirements specification
   c. Software requirements specification

6. Requirements validation
   a. Requirements reviews
   b. Prototyping
   c. Model validation
   d. Acceptance tests

7. Practical considerations
   a. Iterative nature of requirements process
   b. Change management
   c. Requirements attributes
   d. Requirements tracing
   e. Measuring Requirements

Strength
Weakness
B. Software Design: Knowledge

1. Software design fundamentals
   a. General design concepts
   b. Context of software design
   c. Software design process
   d. Enabling techniques

2. Key issues in software design
   a. Concurrency
   b. Control and handling of events
   c. Distribution of components
   d. Error and exception handling and fault tolerance
   e. Interaction and presentation
   f. Data persistence

3. Software structure and architecture
   a. Architectural structures and viewpoints
   b. Architectural styles (macroarchitectural patterns)
   c. Design patterns (microarchitectural patterns)
   d. Families of programs and frameworks

4. Software design quality analysis and evaluation
   a. Quality attributes
   b. Quality analysis and evaluation techniques
   c. Measures

5. Software design notations
   a. Structural descriptions (static)
   b. Behavioral descriptions (dynamic)

6. Software design strategies and methods
   a. General strategies
   b. Function-oriented (structured) design
   c. Object-oriented design
   d. Data-structure centered design
   e. Component-based design (CBD)
   f. Other methods
## Weakness

### B. Software Design: Ability

<table>
<thead>
<tr>
<th>1. Software design fundamentals</th>
<th>4. Software design quality analysis and evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. General design concepts</td>
<td>a. Quality attributes</td>
</tr>
<tr>
<td>b. Context of software design</td>
<td>b. Quality analysis and evaluation techniques</td>
</tr>
<tr>
<td>c. Software design process</td>
<td>c. Measures</td>
</tr>
<tr>
<td>d. Enabling techniques</td>
<td></td>
</tr>
</tbody>
</table>

### 2. Key issues in software design

| a. Concurrency               | b. Control and handling of events               |
| b. Control and handling of events | c. Distribution of components                  |
| c. Distribution of components | d. Error and exception handling and fault tolerance |
| e. Interaction and presentation |                                              |
| f. Data persistence           |                                              |

### 3. Software structure and architecture

<table>
<thead>
<tr>
<th>a. Architectural structures and viewpoints</th>
<th>b. Architectural styles (macroarchitectural patterns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c. Design patterns (microarchitectural patterns)</td>
<td></td>
</tr>
<tr>
<td>d. Families of programs and frameworks</td>
<td></td>
</tr>
</tbody>
</table>

### 5. Software design notations

<table>
<thead>
<tr>
<th>a. Structural descriptions (static)</th>
<th>b. Behavioral descriptions (dynamic)</th>
</tr>
</thead>
</table>

### 6. Software design strategies and methods

<table>
<thead>
<tr>
<th>a. General strategies</th>
<th>b. Function-oriented (structured) design</th>
</tr>
</thead>
<tbody>
<tr>
<td>c. Object-oriented design</td>
<td></td>
</tr>
<tr>
<td>d. Data-structure centered design</td>
<td></td>
</tr>
<tr>
<td>e. Component-based design (CBD)</td>
<td></td>
</tr>
<tr>
<td>f. Other methods</td>
<td></td>
</tr>
</tbody>
</table>
## C. Software Construction: Knowledge

1. Software construction fundamentals
   - a. Minimizing complexity
   - b. Anticipating change
   - c. Construction for verification
   - d. Standards in construction

2. Managing construction
   - a. Construction methods
   - b. Construction planning
   - c. Construction measurement

3. Practical considerations
   - a. Construction design
   - b. Construction languages
   - c. Coding
   - d. Construction testing
   - e. Construction quality
   - f. Integration

<table>
<thead>
<tr>
<th>Strength</th>
<th>Weakness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C. Software Construction: Ability

1. Software construction fundamentals
   a. Minimizing complexity
   b. Anticipating change
   c. Construction for verification
   d. Standards in construction

2. Managing construction
   a. Construction methods
   b. Construction planning
   c. Construction measurement

3. Practical considerations
   a. Construction design
   b. Construction languages
   c. Coding
   d. Construction testing
   e. Construction quality
   f. Integration

Strength
Weakness
# D. Software Testing: Knowledge

1. Software testing fundamentals
   - a. Testing-related terminology
   - b. Key issues
   - c. Relationships of testing to other activities

2. Test levels
   - a. The target of the tests
   - b. Objectives of testing

3. Test techniques
   - a. Based on tester’s intuition and experience
   - b. Specification-based
   - c. Code-based
   - d. Fault-based
   - e. Usage-based
   - f. Based on nature of application
   - g. Selecting and combining techniques

4. Test-related measures
   - a. Evaluation of the program under test
   - b. Evaluation of the tests performed

5. Test process
   - a. Management concerns
   - b. Test activities
D. Software Testing: Ability

1. Software testing fundamentals
   a. Testing-related terminology
   b. Key issues
   c. Relationships of testing to other activities

2. Test levels
   a. The target of the tests
   b. Objectives of testing

3. Test techniques
   a. Based on tester’s intuition and experience
   b. Specification-based
   c. Code-based
   d. Fault-based
   e. Usage-based
   f. Based on nature of application
   g. Selecting and combining techniques

4. Test-related measures
   a. Evaluation of the program under test
   b. Evaluation of the tests performed

5. Test process
   a. Management concerns
   b. Test activities
Evaluation Summary

• Strengths in all four evaluated SWEBOK areas

• Weaknesses in first two SWEBOK areas
  – A. Software Requirements
  – B. Software Design

• Knowledge may be satisfactory, but Abilities to perform are the real “proofs of the pudding”

• In general, the typical systems engineering student did well as a “competent software engineer”.
Classroom Observations (1 of 4)

1. Some discussion required to clarify the differences between functional and non-functional requirements (system and SW).

2. Initial definition of top-level use case framework was weak, but then detailed use case descriptions were well done.

3. Allocation of system requirements to software requirements was not easily done.

4. Concept of a “CSCI” required discussion and project experience to understand.
5. Actually scripting a CSCI requirement was surprisingly difficult.

6. Object-oriented class modeling was weak – some students got it quickly, most did not.

7. Structured analysis (Data Flow Diagrams) was not easy, but most students quickly learned.

8. Constructing state transition diagrams (STDs) was surprisingly difficult.
9. Experience with Requirements Traceability Matrices was mixed
   • General understanding and use was strong, but
   • “Atomically” capturing and listing software (CSCI) requirements was weak.

10. Following architecture and design patterns was strong, but creating new patterns was weak.

11. Object-oriented design and construction was bi-modal – some got it, some didn’t.
12. Behavior modeling (UML) was mixed
   • Strong: Activity and Swim Lane Diagrams
   • Weak: Control Flow, Sequence, and State Transition Diagrams.

13. Algorithm design (using PDL) was weak.
14. Construction – coding and debugging – was strong in general.
15. System integration and testing was strong.
16. Project management and team work was strong.
Conclusions and Recommendations

• NPS MSSE and SE4003 are, in general, good preparation for students to act as software engineers, but weaknesses may exist.

• Recommendations:
  – Prioritize the observed weaknesses and do a causality analysis.
  – For the high-priority (critical) weaknesses, determine which core courses (including SE4003) might need changes.
  – Evaluate impact of proposed changes and determine a balanced curriculum update package.
  – Brief to curriculum sponsor, as required, and implement approved changes.
  – Re-evaluate after several course offerings.