20
System Engineering Lite

18-540 Distributed Embedded Systems
Philip Koopman
November 27, 2000

Required Reading: Brooks, Mythical Man-Month (condensed version)
Ganssle, Habits of Defective Developers

Recommended: Kopetz, Chapter 13
Assignments

- Reading for next lecture covers ethics
  - Other topics to be surveyed as well

- Homework #9 due Wednesday 11/29

- Project #6 due Friday 12/1
Where Are We Now?

◆ Where we’ve been:
  • Lots of lectures on how to build systems

◆ Where we’re going today:
  • Perspective on putting all the pieces together
    – Methodology
    – Management ideas

◆ Where we’re going next:
  • Ethics, survey of advanced topics
  • Bluetooth, embedded toaster ovens, and all that
"If you don't know where you're going, any road will do."
"If you don't know where you are, a map won't help."
(proverbs quoted in Watts Humphrey, Managing the Software Process)

- **System engineering is:**
  - A set of technical approaches to building systems
  - A set of management approaches to control the technical processes

- **Software engineering isn’t exactly system engineering**
  - BUT, software engineering deals with complexity
  - System engineering deals with complexity too

- **CMM: Capability Maturity Model**
  - A way to understand how rigorous the software/system capability is in a development organization
Long Life Cycles

- Picture is typical of a military life cycle
  - Consumer products are similar, but more compressed.

- Some embedded products last a long time!
  - Emphasizes doing a good job up front
  - But, unfortunately the concept exploration phase/architecture phase is more art than science

- So, for this lecture, assume we know how to create good architectures/specifications and proceed from there

<table>
<thead>
<tr>
<th>Phase</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>User need determination</td>
<td></td>
</tr>
<tr>
<td>Concept exploration phase</td>
<td>1-2 years</td>
</tr>
<tr>
<td>Define system requirements</td>
<td></td>
</tr>
<tr>
<td>Establish design concepts</td>
<td></td>
</tr>
<tr>
<td>Demonstration/validation phase</td>
<td>2-3 years</td>
</tr>
<tr>
<td>Allocate requirements</td>
<td></td>
</tr>
<tr>
<td>Establish baseline design</td>
<td></td>
</tr>
<tr>
<td>Full-scale development phase</td>
<td>2-4 years</td>
</tr>
<tr>
<td>Detailed design</td>
<td></td>
</tr>
<tr>
<td>Build test articles</td>
<td></td>
</tr>
<tr>
<td>Qualification tests</td>
<td></td>
</tr>
<tr>
<td>Production/deployment phase</td>
<td>2-10 years</td>
</tr>
<tr>
<td>Product units plus spares</td>
<td></td>
</tr>
<tr>
<td>Train and deploy</td>
<td></td>
</tr>
<tr>
<td>Operational phase</td>
<td>20-30 years</td>
</tr>
<tr>
<td>Operate, maintain, and support</td>
<td></td>
</tr>
<tr>
<td>Product improvement phase</td>
<td>[Sailor90]</td>
</tr>
<tr>
<td>Maintain configuration status</td>
<td></td>
</tr>
<tr>
<td>Build mod kits</td>
<td></td>
</tr>
<tr>
<td>Retirement phase</td>
<td></td>
</tr>
</tbody>
</table>
Generic System Life Cycle

- Based on CMU EDRC model
  - Note that most of product life cycle is not about design
  - BUT, design is what is (predominantly) taught in universities

- System engineering in the real world takes into account:
  - Production
  - Deployment
  - Upgrades
  - Retirement/disposal
When Is The Money Spent On A Product?

- Most costs are determined early in the project
  - “all the really bad mistakes are made on the first day”

Figure 1: System life-cycle cost expenditure versus commitment

(after Defense Systems Management College)

[Sailor90]
What’s The Cost Of A Finding & Fixing A Defect?

Note: cost includes both money and lost time to market!
Waterfall Model

- Traditional/“first” process model
  - Works well with understood systems (e.g., yet another payroll program)
  - Requirements error can make things unravel as you go back up the daisy chain
V Model

- Waterfall model, but also takes into consideration subsystems
Spiral Model

- **Risk-based model**
  - Iterated prototyping at *increasing levels of detail*
  - Usually a better fit for systems that are novel or not well understood

- **Other related concepts**
  - Iterated working prototype with increased feature content
  - Putting biggest risk items early in schedule (risk-driven scheduling)
Figure 1: Spiral Model of the Software Process

[Boehm87]
<table>
<thead>
<tr>
<th>Risk Item</th>
<th>Risk Management Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Personnel shortfalls</td>
<td>Staffing with top talent, job matching; teambuilding; morale building, cross-training; pre-scheduling key people.</td>
</tr>
<tr>
<td>2. Unrealistic schedules and budgets</td>
<td>Detailed, multi-source cost and schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing.</td>
</tr>
<tr>
<td>3. Developing the wrong software functions</td>
<td>Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users’ manuals.</td>
</tr>
<tr>
<td>4. Developing the wrong user interface</td>
<td>Task analysis; prototyping; scenarios.</td>
</tr>
<tr>
<td>5. Gold plating</td>
<td>Requirements scrubbing; prototyping; cost-benefit analysis; design to cost.</td>
</tr>
<tr>
<td>6. Continuing stream of requirement changes</td>
<td>High change threshold, information hiding; incremental development (defer changes to later increments).</td>
</tr>
<tr>
<td>7. Shortfalls in externally-furnished components</td>
<td>Benchmarking; inspections; reference checking; compatibility analysis.</td>
</tr>
<tr>
<td>8. Shortfalls in externally-performed tasks</td>
<td>Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding.</td>
</tr>
<tr>
<td>9. Real-time performance shortfalls</td>
<td>Simulation; benchmarking; modeling; prototyping; instrumentation; tuning.</td>
</tr>
<tr>
<td>10. Straining computer science capabilities</td>
<td>Technical analysis; cost-benefit analysis; prototyping; reference checking.</td>
</tr>
</tbody>
</table>

[Boehm87]
The “Software Crisis” (now in its third decade)

- Hardware complexity is growing exponentially
  - But, programmer productivity is, at best, growing linearly
  - What to do?
- Move to object-oriented programming
  - More than doubles productivity
- US DoD creates the Software Engineering Institute (SEI) at CMU
  - SEI creates the SEI-CMM
- Use COTS software
  - (e.g., Oracle)
- Make everyone a programmer
  - (e.g., Excel)

Source: Software Requirements: objects, functions, states; Davis, 1993.
Object Oriented vs. Structured Approaches

◆ Structured approaches
  • No “goto”
  • Put elements in order of temporal sequentiality (control flow centric)
  • Notation: flowchart and others
  • Example characteristic: INIT_ALL routing
  • Typical software productivity 4 lines per programmer per day
    – For industrial-strength software process, including all factors

◆ Object Oriented approaches
  • Group elements by data types; encapsulate details & control access via methods
  • Notation: UML and precursors
  • Example characteristic: data initialization code associated with class hierarchy
  • Typical software productivity 10 lines per programmer per day
CMM – Capability Maturity Model

- **Five levels of increasing process maturity**
  - Extensive checklist of activities/processes for each level
  - Primarily designed for large-scale software activities
    - Must be tailored to be reasonable for small embedded system projects
  - Growing into a family of models: software/systems/people/…

- [CMM Level 0: “What’s the CMM?”]
CMM Level 1: Initial

- **CMM Level 1: Initial**
  The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.
  - *Process is informal and unpredictable*
  - Intuitively: You have little idea what’s going on with software process

![Diagram](chart.png)
CMM Level 2: Repeatable

- Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
  - *Project management system is in place; performance is repeatable*
  - Intuitively: You know mean productivity
CMM Level 3 - Defined

- The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

  - Software engineering and management processes are defined and integrated
  - Intuitively: You know standard deviation of productivity

![Graph showing probability distribution for development time and cost with estimate/target point](image-url)
CMM Level 4: Managed

- Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.
  - Quantitative Process Management and Software Quality Management
  - *Product and process are quantitatively controlled*
  - Intuitively: You can improve the standard deviation of productivity
CMM Level 5: Optimizing

- Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.
  - Process improvement is institutionalized
  - Intuitively: You can consistently improve mean productivity
CMM In Perspective

- High-quality products can be, and have been, developed by Level 1 organizations.
  - But, often they rely on personal heroics
  - It is hard to predict how the same organization will do on the next job

- CMM is complex, hindering use by small projects
  - 5 levels; 18 key areas; 52 goals; 316 key practices

- The CMM itself does not specify a particular process
  - The CMM does not mandate how the software process should be implemented; it describes what characteristics the software process should have.
  - Ultimately, an organization must recognize that continual improvement (and continual change) are necessary to survive.

- Good process enables, but does not ensure, good product
Why UML (Or Any Other Notation?)

- **UML isn’t novel; it’s just a common representation**
  - Gives a way to exchange ideas
  - Gives a standard way to document thoughts for later access

  - UML is NOT a design methodology …
    …there are many ways to use UML in a methodology and we’re just now figuring out the better ways to do this.

- **There is not (yet) any “best” process**
  - “Best” varies depending on size of team, complexity/novelty, and company culture
  - But, it’s better to have some process than no process
A Typical UML-Based Process

- People create architecture
- People + tools create design
- Tools (attempt to) synthesize implementation

- Weak points:
  - Where’s “distributed”?
  - Where’s “real time”?
  - Where’s the network?
  - Where’s hardware?
Process Taught In This Course

- **(Work in progress…)**
  - Real time aspects not rigorous
  - Still a lot of gaps in UML Design portion (*e.g.*, going from requirements to statecharts)

- **18-849A next semester:**
  - Fill in the gaps
  - Get real about real-time
  - Create a “real” elevator system using UML++ (fewer shortcuts)

![Diagram of the process](image)
Process Wisdom

- **Testing doesn’t find defects, it is merely a QA technique**
  - When a defect is found, it indicates a broken process
    - Figure out what is broken in the process and fix it
    - Look for other similar defects already created by that same process problem
  - 100% “coverage” and 100% “passes” on tests does not equate to perfection
    - The pesticide paradox means fairly good software always passes 100% of tests

- **Use standard techniques – build on others’ work**
  - “In physics they stand on each other’s shoulders; in computer science we stand on each other’s feet”
  - IEEE and others have a dizzying array of standards – don’t reinvent the wheel

- **Be driven by change and risk**
  - Create architectures that won’t be broken by expected changes
  - Run projects to maximize impact and reduce risk (use risk reduction techniques when they make sense)
<table>
<thead>
<tr>
<th>Our project was late, so we added more people. The problem got worse</th>
<th>We can’t get it right and still come in on schedule. Why can’t we do both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>When requirements changed, the schedule did not. Were we in trouble?</td>
<td>There is no more time, but the work is unfinished. Take more time from Test.</td>
</tr>
<tr>
<td>I gave estimates. They cut all of them in half. Next time I’ll pad them.</td>
<td>If a project fails, but we keep working on it, has it really failed?</td>
</tr>
</tbody>
</table>
Review

- **System engineering is:**
  - A set of technical approaches to building systems
  - A set of management approaches to control the technical processes

- **System cost is determined early in life cycle**
  - So, it’s important to be smart about designing it well!

- **CMM: Capability Maturity Model**
  - A way to understand how rigorous the software/system capability is in a development organization

- **UML & methodologies**
  - This course has taught an augmented, UML-based design methodology
  - There are many methodologies; use the one that makes sense for your situation