Schedule

- Final All-Sites: Friday June 9th, 7:00 PM
- Final report: short report
  - What was accomplished
  - What was effective (e.g., communications, specifications) and what was not
  - What you learned
  - What you would change
Overview

• “Houston, we have a problem.” – on the nature of the ongoing “software crisis”
• No silver bullet – roadblocks to crisis resolution
• A road less traveled – going around rather than through the roadblocks
• Déjà vu all over again - why we cannot understand the future if we will not understand the past
• The Art of War - strategy before tactics

“Houston, we have a problem.”

• Have been in “crisis” since the advent of “big” software
  – What we want for software development
    • Low risk and predictability
    • Lower costs and proportionate costs
    • Faster turnaround
  – What we have:
    • High risk, high failure rate
    • Unpredictable schedule, cost, effort
  – Characterized by lack of control
• Three road blocks to getting beyond the crisis:
  1. Complex software is essentially difficult
  2. Software development is pre-industrial
  3. Traditional SE processes only look at part of the problem
The SSE Hypothesis

- Effective control of developments in most real companies requires a strategic view of software development
  - (i.e., one that spans development across multiple products, over multiple development cycles, and across organizational boundaries)
- Lemma 1: Necessary control is unobtainable at the project level
- Lemma 2: Some key development problems are artifacts of the way we decompose the problem (i.e., the narrow view)

1. Software is Essentially Difficult

- In *No Silver Bullet* Fred Brooks provides a perspective on why we have not been able to solve the “software crisis”
  - Identifies two kinds of development difficulties
    - Essential difficulties - part of the essence of the problem
    - Accidental difficulties - introduced by imperfect practice
  - Thesis: developing complex software systems is *essentially difficult – but not as difficult as we make it!*
    - “The hard part of building software [is] the specification, design and testing of [its] conceptual constructs, not the labor of representing it and testing the fidelity of the representation”
  - Lemmas:
    1. Most new SE technology addresses primarily “accidental difficulties”
    2. Improvements here are at the point of diminishing returns
2. Software is Pre-Industrial

<table>
<thead>
<tr>
<th>Pre-Industrial</th>
<th>Post-Industrial</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The Craftsman</strong></td>
<td><strong>The Factory</strong></td>
</tr>
</tbody>
</table>

Pre-Industrial vs. Post-Industrial

**Pre-Industrial**
- Craftsman builds product
  - Builds one product at a time
  - Each product is unique, parts are not interchangeable
  - Quality depends on craftsman’s skill – product of training, experience
  - Unbounded opportunities for error
- Focus on individual products
  - Customization is easy
- Scaling is difficult
  - Parts are not interchangeable
  - No economy of scale
  - Control problems rise exponentially with product size!

**Post-Industrial**
- Products produced by machines
  - Quality depends on machines & manufacturing process
  - Production requires little training or experience
- Focus on developing the means of production
  - Craftsman builds means to build product (tools, factory)
  - Customization is difficult
- Easily scales
  - Parts are interchangeable
  - Products are alike
  - Economies of scale apply
The “Big Project” Problem

- Control problems rise exponentially with product size!
- Each interface must be hand fit

3. We Only Look at Part of the Problem

This view is an impediment to progress

- Know that choices here
  - Are affected by prior choices, experience, products
  - Will affect future capabilities: business, technical

- Omits critical context information
  - How do the activities relate to short or long term business goals?
  - How do the decisions made here affect future products?
  - How does this development relate to others in the organization?

- Factors affecting ability to control the process are outside the scope
  - Factors affecting the process are implicit
  - Factors affected by the process are nowhere evident
Consequence: Merry-Go-Round of Sequential Development

- Knowledge is not institutionalized (tacit)
- Doomed to repeat lessons of the past

Sequential Development Over Time

... a result of “tactical software engineering”
If only we could stop and think…

“Here is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels there really is another way, if only he could stop bumping for a moment and think of it”

— A. A. Milne

A Road Less Traveled

Going around (rather than through) the software productivity barriers
Qualities of a Solution

• Essential nature of the problem
  – Building conceptual structures is essentially difficult
  – There are inherent control problems in building large, complex systems from hand-crafted parts
  – Key problems are outside the traditional life cycle scope
  => Cannot solve the problem within traditional life-cycle view

• Implied nature of the solution
  1. Change what we build (product) - reuse conceptual structures
  2. Change how we build (process)
     • Use an industrial model of development
     • Expand the development scope to encompass the whole problem
  3. Change who does what (organization) – organize to build families of products and processes

I. Change What We Build to Systematically Reuse Conceptual Structures

• Observation: sequential development is inefficient
  – Much of software “development” is really re-development.
    • Software inevitably exists in many versions
    • Seldom develop truly new applications
  – Implication: typically much in common among systems we build … but very little is reused!

• Real progress depends on reusing (not re-creating) conceptual structures
  – Exploit commonality to develop reusable assets for system families
  – Develop infrastructure to support rapid production of instances of a family from reusable assets
  – Avoid the essential difficulties of creating conceptual structures by reusing them
II. Exploit an Industrial Development Model

**Pre-Industrial**

1. Build Product (by hand)

**Post-Industrial**

1. Develop Means of Production
   - Design Manufacturing process
   - Build Factory

   Factory (Means of Production)

   Build Products

   Product

Historical note: Eli Whitney’s vision and invention transformed the product development process.

---

**Product-Line Development**

- A product-line development strategy sets out to maximize reuse over a family of similar software systems.
  - Reuse is planned and systematically executed
  - Common assets are developed as a distinct product
  - The process focuses on developing means of production before any particular product

**A Product-Line Process**

- Develop Means of Production
  - Design development process
  - Build production environment

- Build Applications (Family Members)

- Application Engineering Environment (Means of Production)

- Application

Build reusable assets
Product-Line Development Over Time

III. Extend the Development Scope

- Shift from “tactical” to “strategic” view of development
- “Tactical” development maximizes local objectives
  - Focuses on a particular product
  - Key SE issue: how to make tradeoffs to achieve product-relevant goals (improved quality, reduced cost)
  - Concerns outside the project are ignored
- “Strategic” development focuses on maximizing global return on investment
  - Focuses on overall business objectives
  - Key SE issue: how to make trade-offs to maximize overall return
    - May trade-off near-term goals to maximize overall return
    - E.g., pay upfront to develop reusable architecture
  - One project is viewed as only one part of the business
Components of Strategic Development

- Process
  - Encompasses multiple product developments
  - Reorganized to evolve software product families
- Products
  - Production environment as product
  - Processes as product (concurrent engineering)
- Organization
  - Link strategic business goals to technical decisions
    - Management, marketing, customer come into the loop
  - Organizational support for strategic asset base
    - Support common assets independent of any deliverable
    - Provide budget, schedule, authority, ownership
  - Capital investment in means of production

Déjà vu all over again…
Meta-Software Engineering

- Re-engineering the SE process
- Definition: a development process is “strategic” if it considers and invests in work products beyond the scope of the current product development
  - Focus on intellectual and managerial control over multiple developments over time
  - Product-line software development is a type of strategic process
- Key Question: how do we transition from a “tactical” development approach to a “strategic” one?
  - Are there principles, processes, and methods supporting strategic software engineering?
  - Is there a transition path?

Existing Technology Suffices

- Logical extension of a common set of SE principles
  - design for ease of change
  - information hiding
  - abstraction
  - separation of concerns
  - developing programs as families of systems
- What differs is how they are applied
  - Apply to strategic goals
  - Apply to SE tools and methods themselves (recursively)
- Requires no new technology but does require
  - Deep understanding of SE principles
  - Thinking differently about SE problems
Questions?