

---

# CCPDS-R Case Study and Software Process Tailoring: Walker Royce Insights

Barry Boehm  
CS 577a Lecture

# Outline

---

- **CCPDS-R Case Study**
  - **Project Overview**
  - **Plans and Schedules**
  - **Results and Critical Success Factors**
- **Process Tailoring**
- **General Best Practices**



# Managing Successful Iterative Development Projects

A Seminar on Software Best Practices  
Version 2.3

2800 San Tomas Expressway  
Santa Clara, Ca 95051  
(408) 496-3600

# Case Study: CCPDS-R Project Overview

---

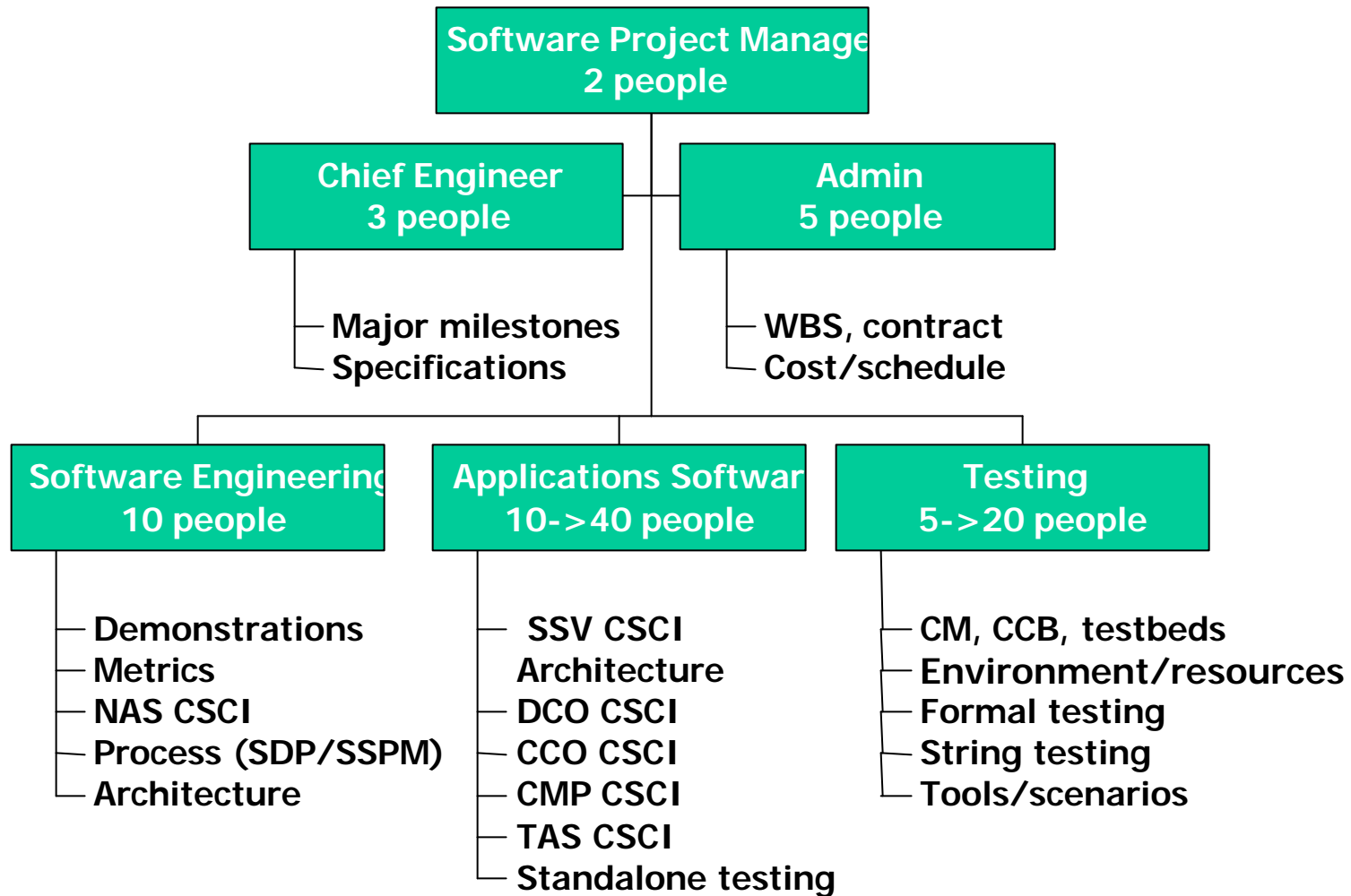
<b>Characteristic</b>	<b>CCPDS-R</b>
<b>Domain</b>	<b>Ground based C3 development</b>
<b>Size/language</b>	<b>1.15M SLOCAda</b>
<b>Average number of people</b>	<b>75</b>
<b>Schedule</b>	<b>75 months</b>
<b>Process/standards</b>	<b>DOD-STD-2167A Iterative development</b>
<b>Environment</b>	<b>Rational host DEC host DEC VMS targets</b>
<b>Contractor</b>	<b>TRW</b>
<b>Customer</b>	<b>USAF</b>
<b>Current status</b>	<b>Delivered On-budget, On-schedule</b>

# CCPDS-R Case Study Overview

---

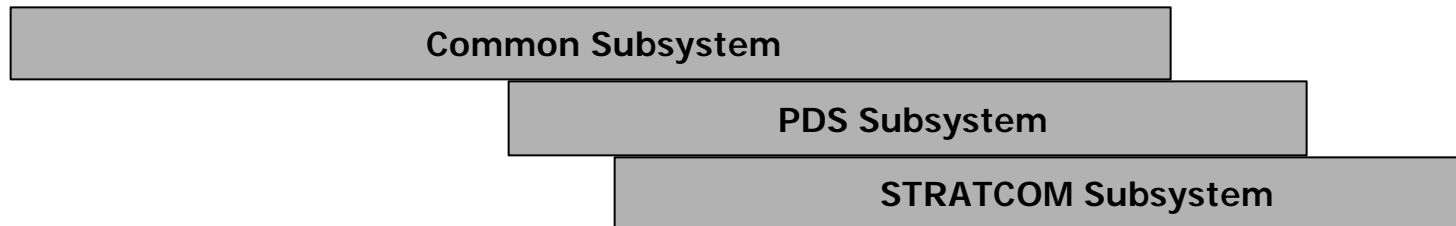
- Organization
- Subsystems/schedules
- Iteration plan/content
- Core metrics
  - Development progress (size with respect to time)
  - Test progress (evaluation criteria with respect to time)
  - Software architecture stability (numbers of objects with respect to time)
  - Subsystem stability (SCOs with respect to time)
  - Modularity (SLOC: scrap with respect to time)
  - Modularity (SLOC %of total: scrap with respect to time)
  - Adaptability (rework with respect to time)
  - Maturity (reliability with respect to time)
  - SCO volumes (raw data by CSC I)
  - Cost/effort partitions by activity
  - Conclusions

# CCPDS-R Organization/Responsibilities



# CCPDS-R Overview

---



<u>Project</u>	<u>Size</u>	<u>Schedule</u>
Common months	353 KSLOC	60
PDS months	254 KSLOC	32
STRATCOM months	552 KSLOC	36
Total months	1159 KSLOC	75

*Software effort: 75 people average*

# Common Subsystem Build Content

---

Primitives and support software

Architecture, test scenarios, models

Critical thread applications, displays

Mission algorithms, non-critical thread applications, displays

Communications interfaces, final test scenarios

Associate contractor interface

*Reliability and performance-critical  
component development*

*Reliability and performance-critical  
component testing and maturation*

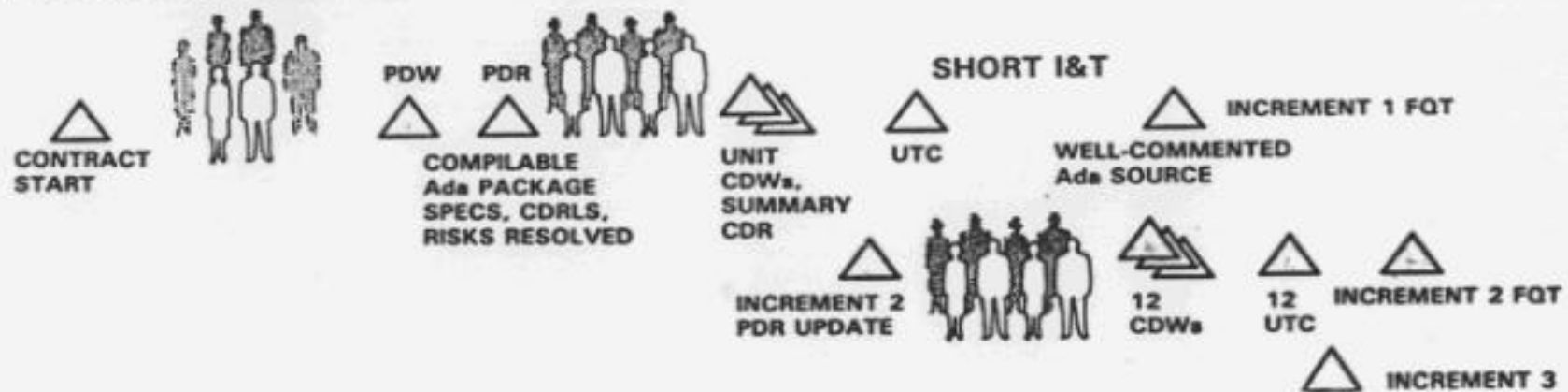
# Ada Process Model for Large-System RAD

## “ALL TOO FREQUENT PROCESS MODEL”

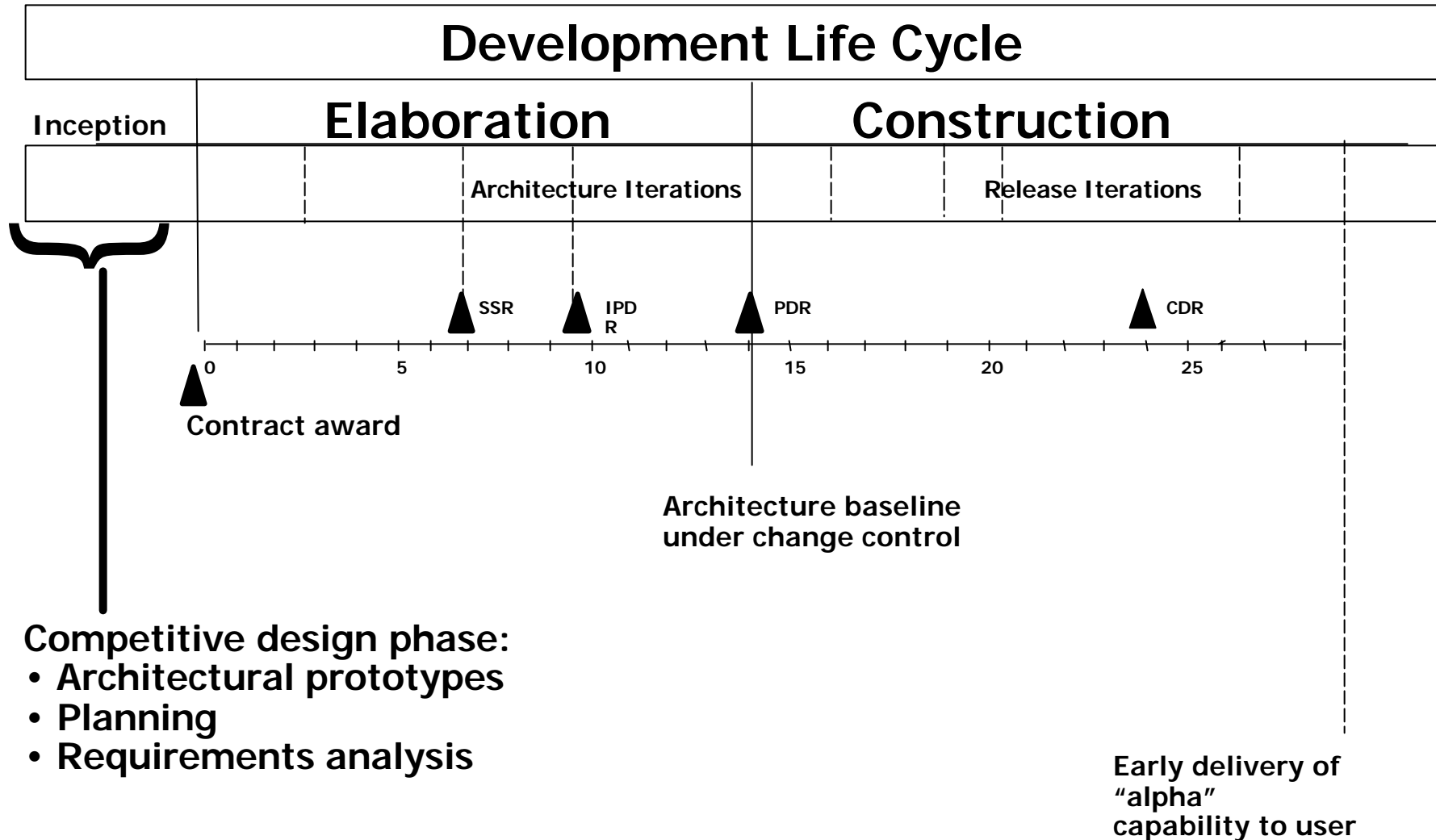


## Ada PROCESS MODEL

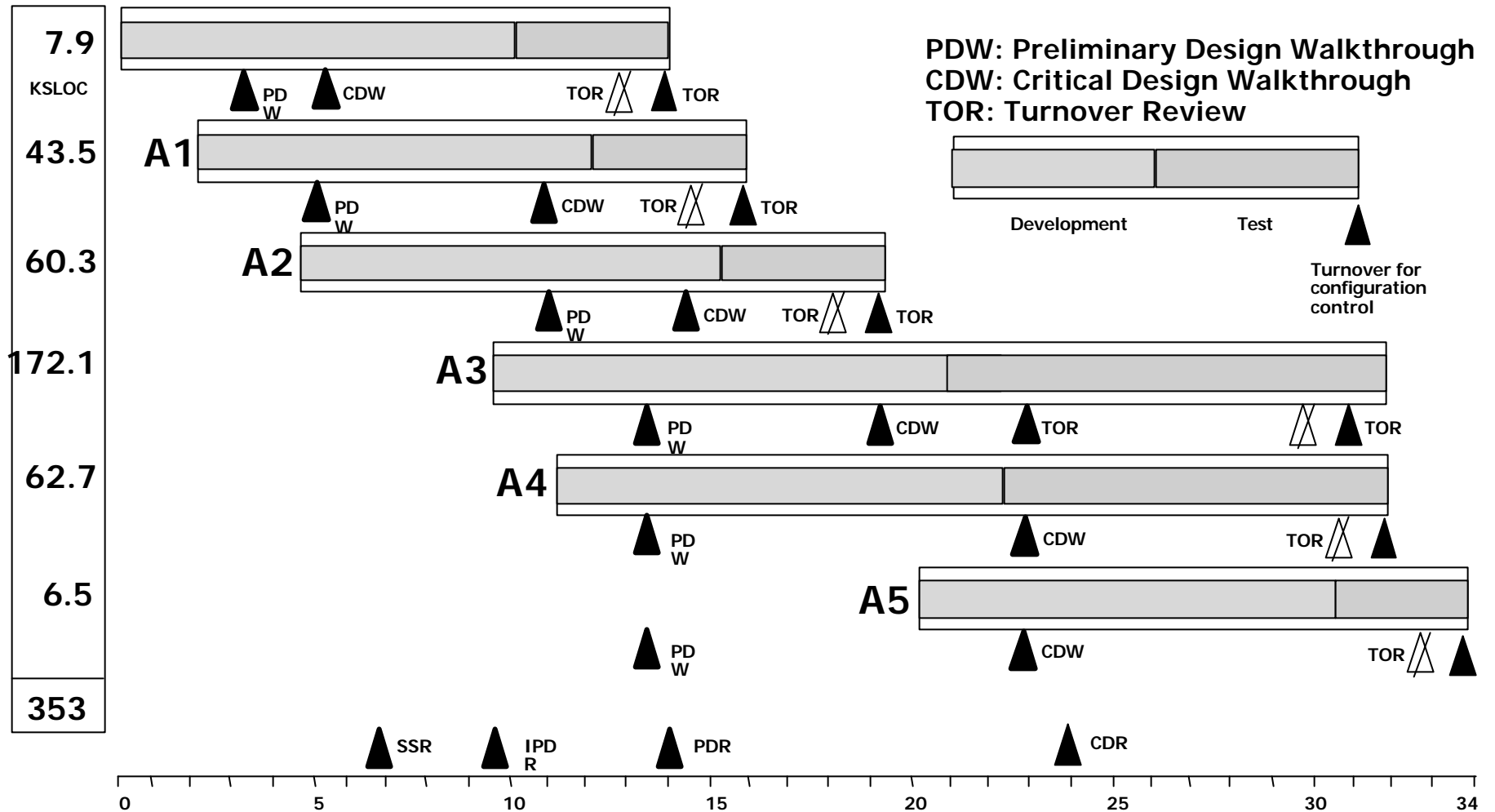
SMALL, TOP TEAM WORKING ARCHITECTURE, RESOLVING HIGH-RISK ITEMS



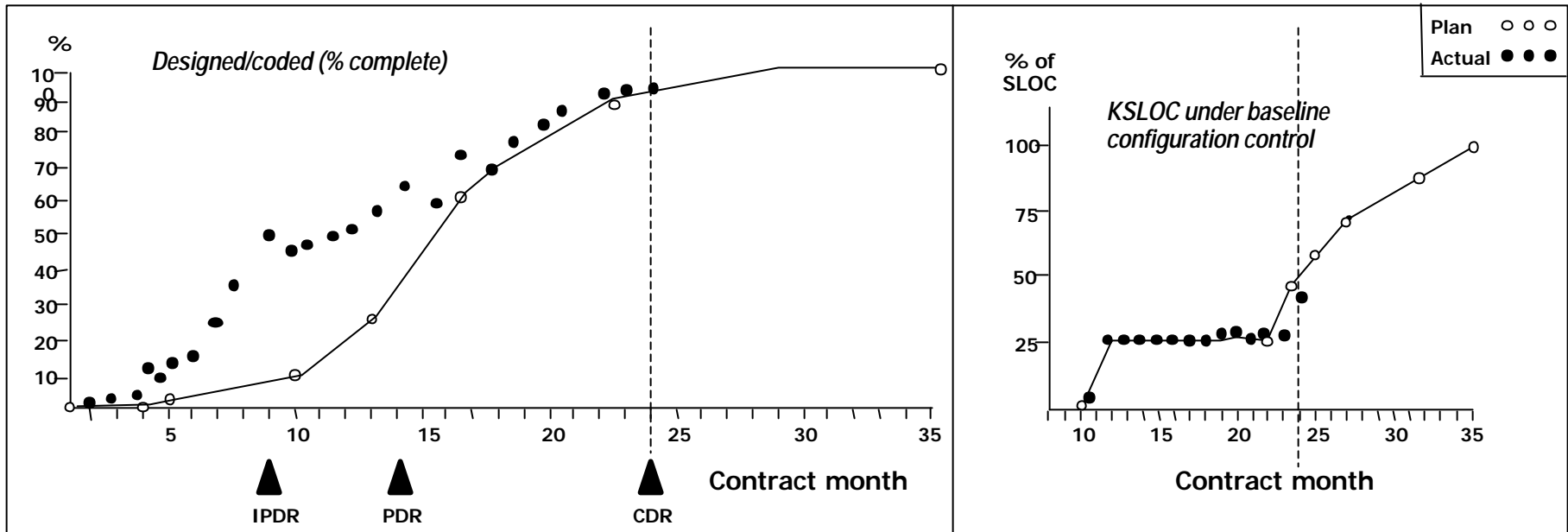
# Common Subsystem Macroprocess



# Common Subsystem Microprocesses



# Common Subsystem Progress



- CDR progress:

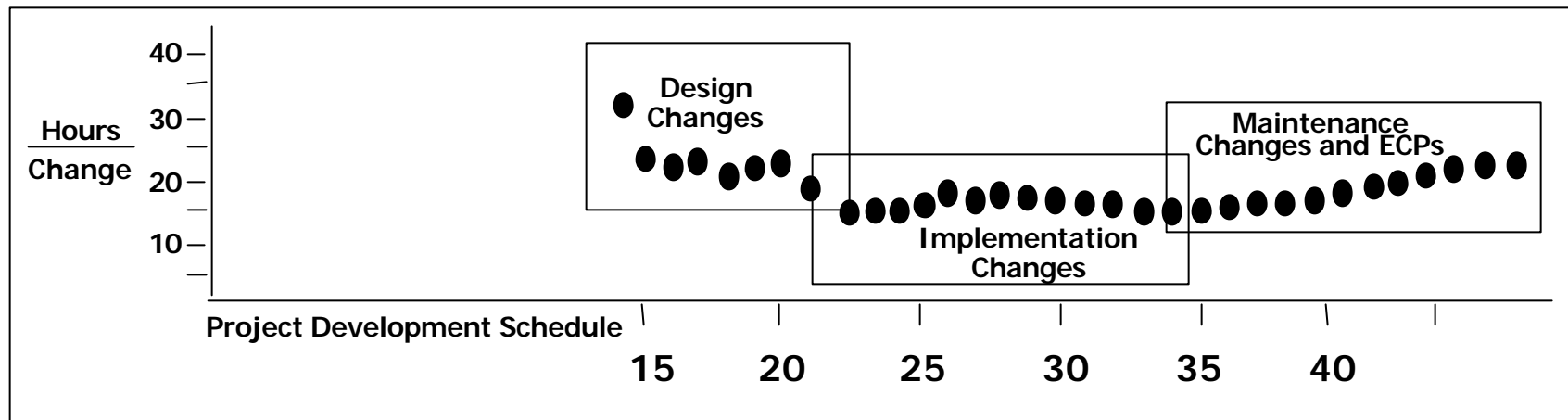
### Traditional Approach

### CCPDS-R Approach

Software design	→	Complete	Complete
Code development	→	10%	94%
Baseline under change control	→	Negligible	47%
Formal test	→	0%	12%
Performance assessment	→	Modeling	80% of operational software demonstrated

# Common Subsystem Adaptability

- Architecture first
  - Integration during the design phase
  - Demonstration-based evaluation
- Configuration baseline change metrics:



# Some General Conclusions

- Common subsystem subsidized:

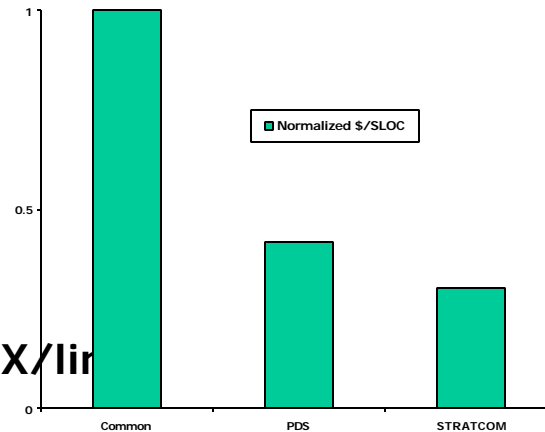
- Internal reuse
- Common process/tools
- Overall systems engineering
- Cost/SLOC:

Common  
PDS

STRATCOM

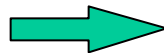
\$X/line  
\$.40X/line

\$.33X/line

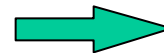


- Productivity/quality improved with each subsystem.

Common



PDS



STRATCOM

- This is the real indicator of a mature (level 3/level 4) process.

- Personnel volatility was low on the common subsystem.

- It was much higher on the PDS and STRATCOM subsystems.

# General Conclusions (Continued)

---

- CCPDS-R adhered to 2167A
  - Process could have been much more efficient
  - Early (initially unplanned) delivery (month 30) of a useful increment
- Moderate contractual requirements volatility
  - Heavy non-ECP requirements/design volatility
  - Only 1 major ECP (complete overhaul of the input message set)
  - Contractor/customer/user rapport avoided an inefficient contracts bureaucracy
- Personnel skill:
  - Strong project management, planning, and architecture teams
  - Cooperative customer/user
  - Average application/test team

# CCPDS-R Software Technology Thrusts

---

	Current Thrust	CCPDS-R Approach
<b>E</b>	<ul style="list-style-type: none"> <li>• Integrated tools</li> <li>• Open systems</li> <li>• Hardware performance</li> <li>• Automation</li> </ul>	<ul style="list-style-type: none"> <li>• DEC/Rational/custom tools</li> <li>• VAX/DEC dependent</li> <li>• Several VAX family upgrades</li> <li>• Custom-developed change management system, metrics tools, code auditors</li> </ul>
<b>Size</b>	<ul style="list-style-type: none"> <li>• Reuse, COTS</li> <li>• Object-oriented</li> <li>• Higher level languages</li> <li>• CASE tools</li> <li>• Distributed middleware</li> </ul>	<ul style="list-style-type: none"> <li>• Common architecture primitives, tools, processes across all subsystems</li> <li>• Message-based, object-oriented architecture</li> <li>• 100%Ada</li> <li>• Custom automatic code generators for architecture, message I/O, display format source code generation</li> <li>• Early investment in NAS development for reuse across multiple subsystems</li> </ul>
<b>P</b>	<ul style="list-style-type: none"> <li>• Iterative development</li> <li>• Process maturity models</li> <li>• Architecture-first</li> <li>• Acquisition reform</li> <li>• Training</li> </ul>	<ul style="list-style-type: none"> <li>• Demonstration, multiple builds, early delivery</li> <li>• Level 3 process prior to SEI CMM definition</li> <li>• Executable architecture baseline with CM at PDR</li> <li>• Excellent customer/contractor/user teamwork, highly tailored 2167A for iterative development</li> <li>• Mostly OJT and internal mentoring</li> </ul>

# CCPDS-R MBASE Models

---

- **Success Models**
  - Reinterpreted DOD-STD-2167a; users involved
  - Award fee flowdown to performers
- **Product Models**
  - Domain model and architecture
  - Message-passing middleware (UNAS)
- **Process Models**
  - Ada process model and toolset
  - Incremental builds; early delivery
- **Property Models**
  - COCOMO cost & schedule
  - UNAS - based performance modeling
  - Extensive progress and quality metric tools

# Agend

## a

---

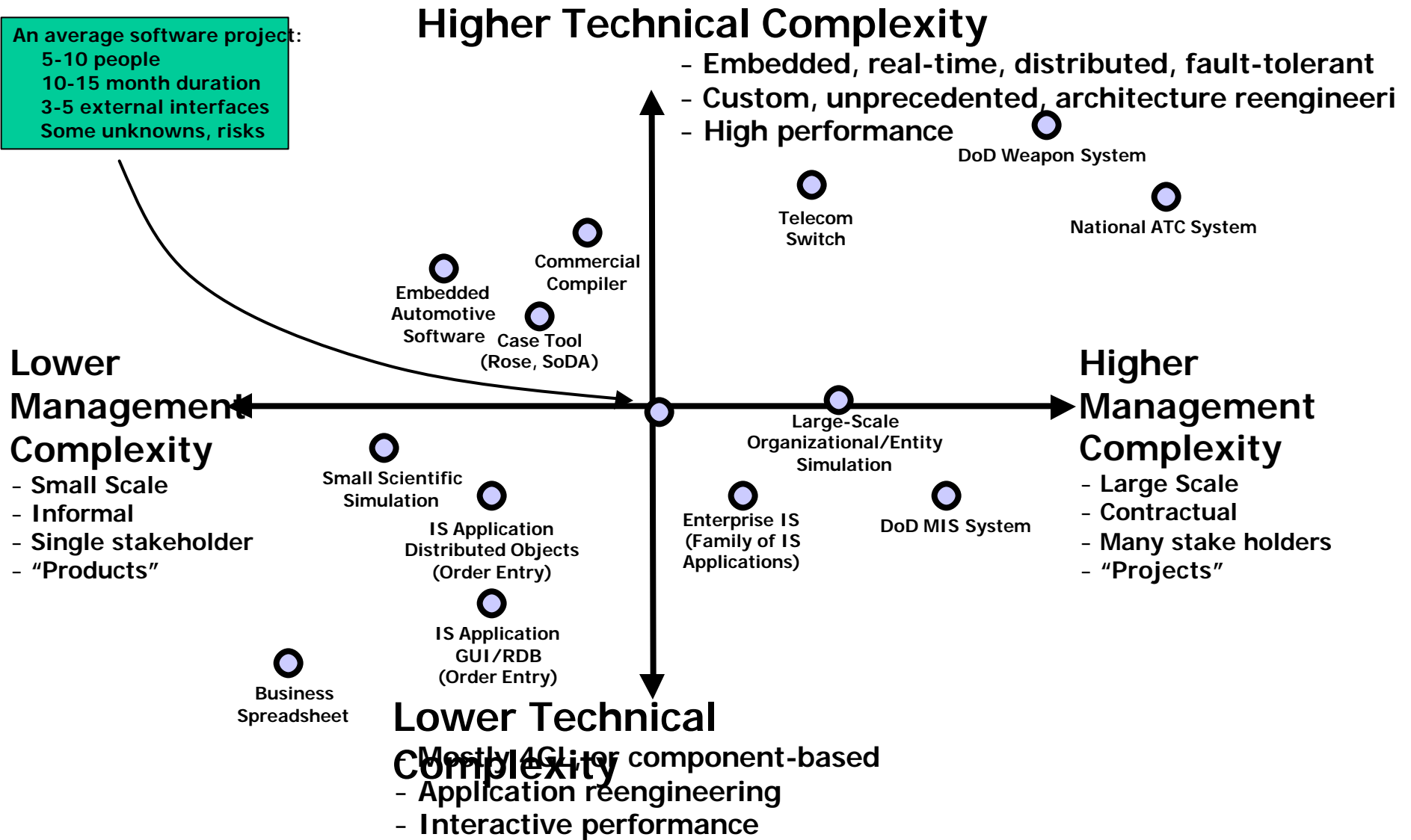
- Motivation
  - The search for software best practices
  - Software economics
- Technology overviews
  - Iterative development
  - Architectures
  - SEI CMM and ISO 9000
- Software process
  - The Rational Objectory process
  - An organizational process
  - Tailoring the process
  - The acquisition process
- Experience and results
  - General observations
  - Case Studies and real-world metrics
- Conclusions

### Tailoring the process

- ↓ Present the range of applicability
- ↓ Same invariant spirit/policy
- ↓ Different priorities
- ↓ Different implementations

# Classifying Systems

An average software project:  
 5-10 people  
 10-15 month duration  
 3-5 external interfaces  
 Some unknowns, risks



# Process Tailoring

---

## Higher Technical Complexity

- More emphasis on domain experience
- Longer inception/elaboration phase
- More iterations, risk management
- Less predictable costs/schedules

Lower  
Management  
Complexity

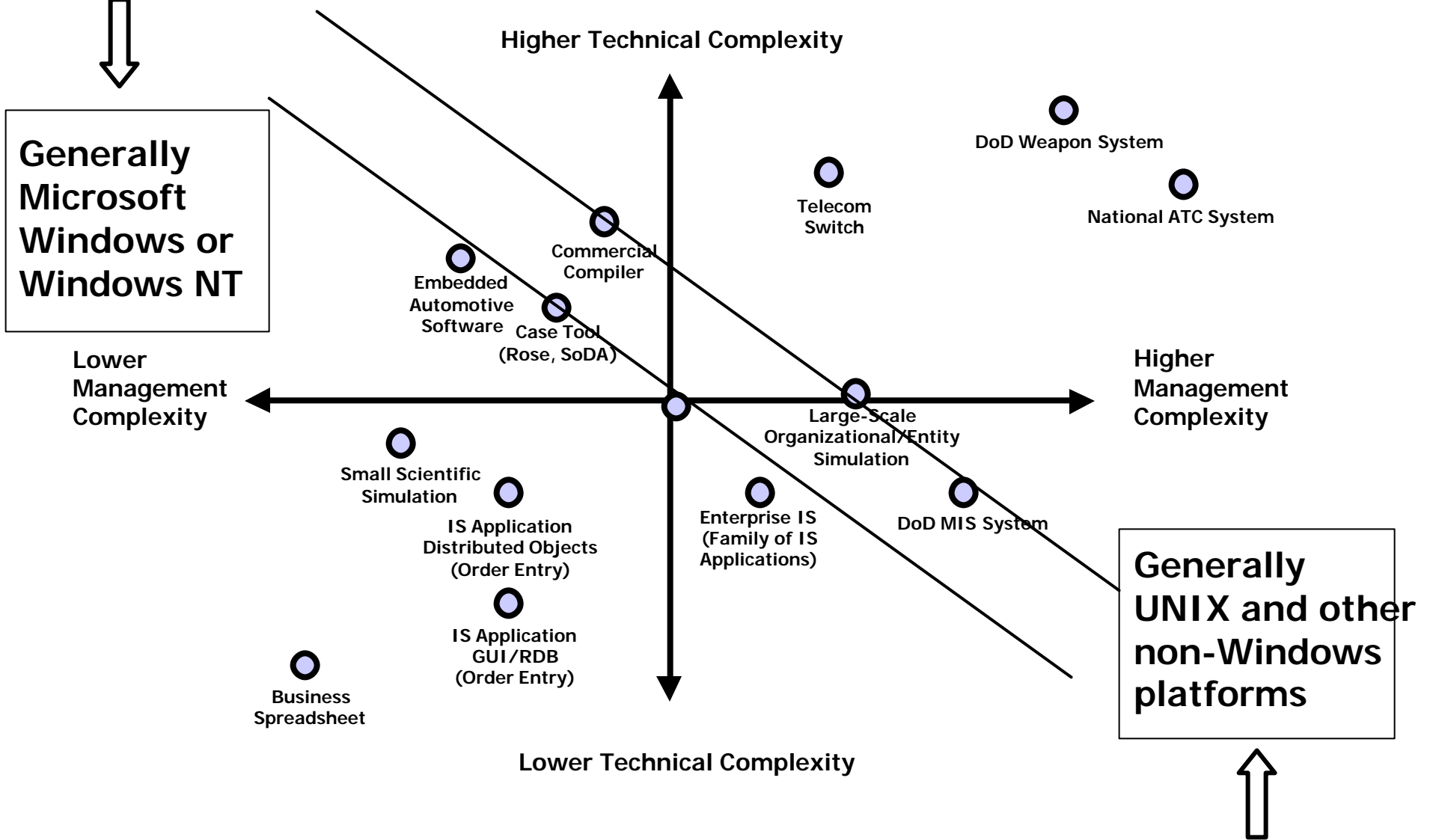
Higher  
Management  
Complexity

- More emphasis on management perspective
- More process formality/ceremony
- More emphasis on teamwork and win/win
- Longer Inception/elaboration

## Lower Technical Complexity

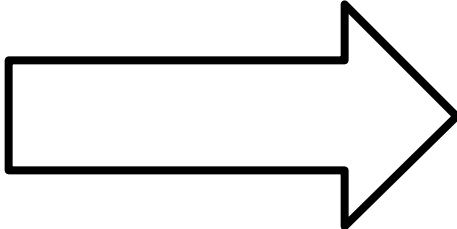
- More emphasis on existing assets/knowledge base
- Shorter inception/elaboration phase
- Fewer iterations
- More predictable costs/schedules

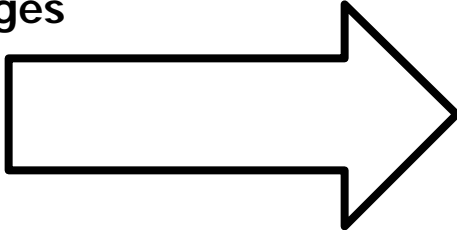
# UNIX vs. Windows



# Products vs. Projects

---

- **Emphasis and focus are different.**
  - **Products: Market-constrained, small teams, driven by technical challenges**
    - Usability
    - Reusability
    - Portability
    - Adaptability

**Varying levels are freely traded off with cost and time-to-market**
  - **Projects: Contract constrained, large teams, driven by management challenges**
    - Completeness
    - Performance
    - Reliability

**Fixed levels are required at fixed costs and schedules**
- **The same process themes apply.**
  - However, priorities are different.

# Products vs. Projects

---

- Top 10 discriminations of success/failure:

## Products

1. Architecture team skill
2. Object-oriented technology
3. Programming team skill
4. Iterative development
5. Change management rigor
6. User/customer focus
7. Environment integration
8. Languages/standards
9. Management team skill
10. Megaprogramming

## Projects

1. Iterative development
2. Architecture team skill
3. Management team skill
4. Acquisition process/rapport
5. Megaprogramming
6. Change management rigor
7. Environment integration
8. Languages/standards
9. Object-oriented technology
10. Programming team skill

# Product Discriminators

---

- **Architecture/programming team skills are paramount.**
  - **Innovation and heroic efforts are critical**
- **Management skill is less important**
  - **There are few constraints. Time-to-market, function, quality, and cost are negotiable.**
  - **Tradeoffs can be decided upon within common corporate goals.**
- **Object-oriented technology is key to technical success.**
  - **Reuse, adaptability and portability are necessities**
- **Acquisition process/rapport is generally unimportant.**
  - **User/customer focus is still vital**
  - **The market is moldable; customers appears after development.**

# Project Discriminators

---

- Iterative development and acquisition process are key.
  - R&D (resolving unknowns in requirements, design, and technology) is separate from production (development of useful increments).
- Architecture team skill is very important.
  - #1 driver of technical quality, feasibility, and application production
- Management skill is very important.
  - Many constraints (cost, schedule, function, quality); continuous negotiation and team building
- Applications team skill is less important.
  - A good architecture can be implemented by an average team.

# Best Practices Cross

## Reference

---

1. Make Quality #1.
2. High-quality software is possible.
3. Give products to customers early.
4. Determine the problem before writing the requirements.
5. Evaluate design alternatives.
6. Use an appropriate process model.
7. Use different languages for different phases.
8. Minimize intellectual distance.
9. Put techniques before tools.
10. Get it right before you make it faster.
11. Inspect code.
12. Good management is more important than good technology.
13. People are the key to success.
17. Plan to throw one away.
19. Design for change.
20. Design without documentation is not design.
26. Don't test your own software.

# Our Preferred Wording

---

1. Define quality commensurate with your objectives
2. Make quality assurance a team goal, not a separate discipline
3. Give products to customers early.
4. Determine the problem before writing the requirements.
5. Evaluate design alternatives.
6. Tailor the process to the scale/domain/complexity of the project
7. Minimize the number of life-cycle representation formats.
8. Minimize intellectual distance (use an object oriented method).
9. Put techniques before tools.
10. Get it working before you make it faster.
11. Use inspections judiciously
12. Good management is more important than good technology
13. People are the key to success.
17. Encourage experimentation and exposure of design issues.
19. Design for change.
20. Build self-documenting products rather than self-serving documents.
26. Plan and execute a test program from day 1.

# Recurring Themes of Success

---

- Customer-user-contractor teamwork is essential
  - Relationships are non-adversarial.
- Managers are performers.
  - They author their own plans.
- Requirements/designs are fluid and tangible.
  - Maturity evolves where designs are “guilty until proven innocent.”
  - Real issues are surfaced and resolved systematically.
  - Change management overrides change avoidance.
- CM/QA is everyone’s job, not a separate discipline.
- Performance issues arise early in the life cycle.
  - “Something” is performing.
- Low attrition of good people is a sign of success.

# Recommendations to Buyers

---

- Try to acquire a good product and have your contractor make a good profit
- **Install an on-line acquisition environment:**
  - Demand delivery of executable evolving product versions.
    - ▲ Get users involved early and continuously.
  - Add value through your own engineering support.
    - ▲ Do your own metrics analysis.
    - ▲ Construct scenarios.
- **Avoid attrition and adversarial relationships.**
- **Strive for 80% solution early, then evolve toward 100%**
  - Separate the “need” from the “want.”

# Recommendations to Contractors

---

- **Iterative development exposes important trends early.**
  - Good early performance breeds continuous success.
  - Poor early performance is almost impossible to turn around.
    - ▲ Get the right management team.
    - ▲ Get the right architecture team.
- **Provide adequate capital environments.**
  - Iterative development is much more (computer) resource intensive
  - Opportunities for automation must be exploited.
- **Enable round-trip engineering.**
  - Integrated environments and compatible tools.

# Rational Process Summary

---

	Development Life Cycle			
	Inception	Elaboration	Construction	Transition
<b>Objective</b>	Identify Revenue opportunity	Risk resolution	Development	User satisfaction
<b>Product</b>	Business case	Architecture and Production plan	Useful increments	Version updates
<b>Cost estimate</b>	Prototyping Object points	Early design Function points	Post-architecture Source lines	Maintenance Annual change traffic
<b>Risk Focus</b>	Technical and economic feasibility	Schedule	Cost	Deployment perturbations

# The Importance of Automation

---

- **Process maturity depends on capable environments.**
  - Metrics must be automatically collected by the environments.
    - ▲ It's the best way software engineers will accept them.
  - Change management must be automated and enforced.
    - ▲ It's the best way to manage multiple iterations.
    - ▲ It's the best way to enable change freedom.
    - ▲ Change is the fundamental primitive of iterative development.
  - An architecture-first and demonstration-based process is enabled by off-the-shelf architecture components and middleware.
    - ▲ It's the best way to remain hardware/topology-independent.
  - Tools must be pre-integrated.
    - ▲ It's the best way to maintain consistency and traceability.
  - Testing/documentation must be mostly automated.
    - ▲ It's the best way to attain change freedom

# Making a Positive Economic Impact

---

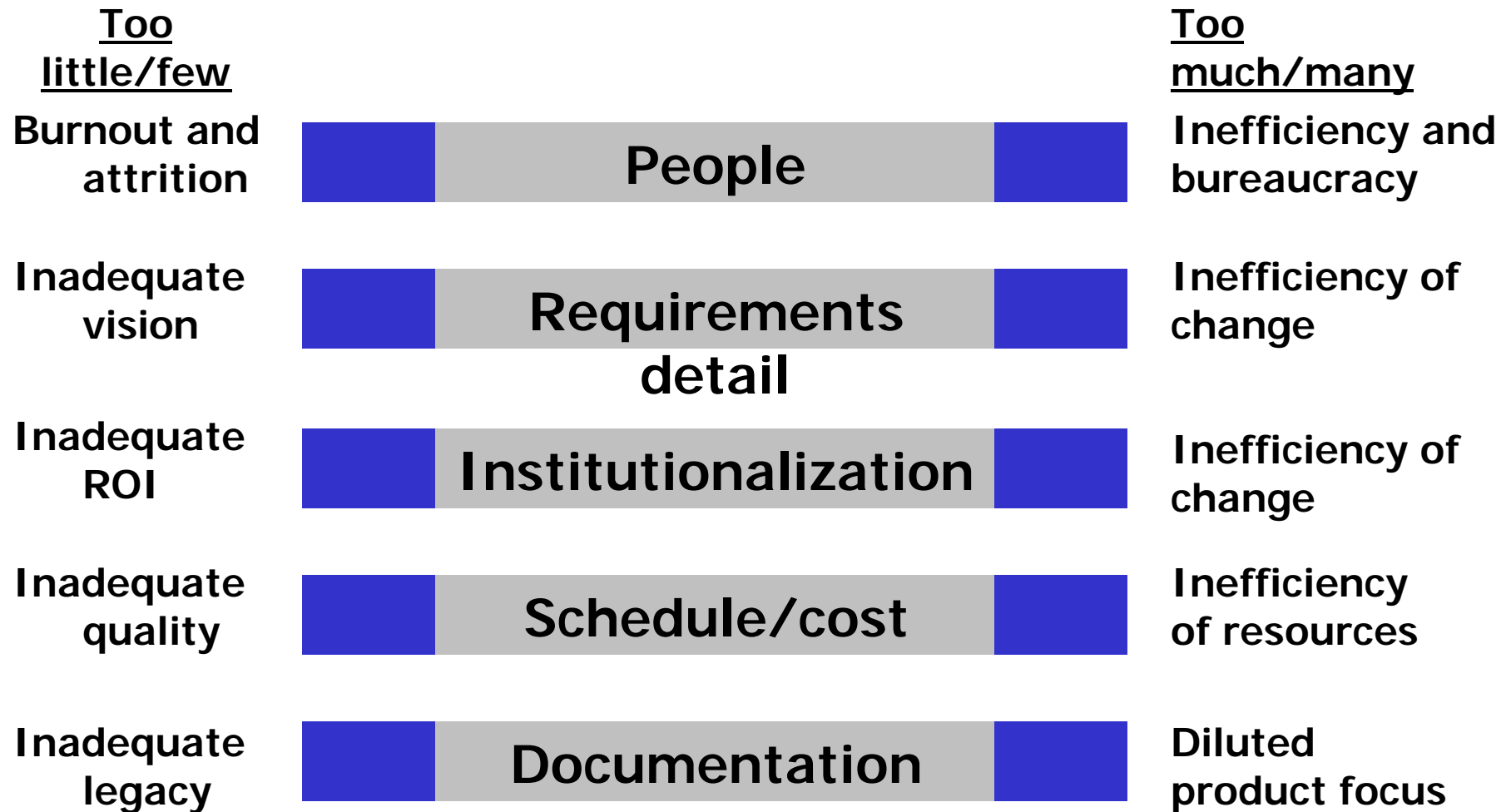
$$\text{Cost} = (\text{Environment}) * (\text{Size})^{(\text{Process})}$$

- **Environments are the key to reduced “coefficients.”**
  - More automation and integration
  - Change management and change freedom
- **Objects technology is the key to reduce “Size.”**
  - Reuse, off-the-shelf products, automatic code generation
  - Line-of-business architectures
- **Iterative development is the key to reduced “exponents.”**
  - Architecture-driven, continuous integration, tangible quality/progress

**Maintain a balance of emphasis on all 3 thrusts.**

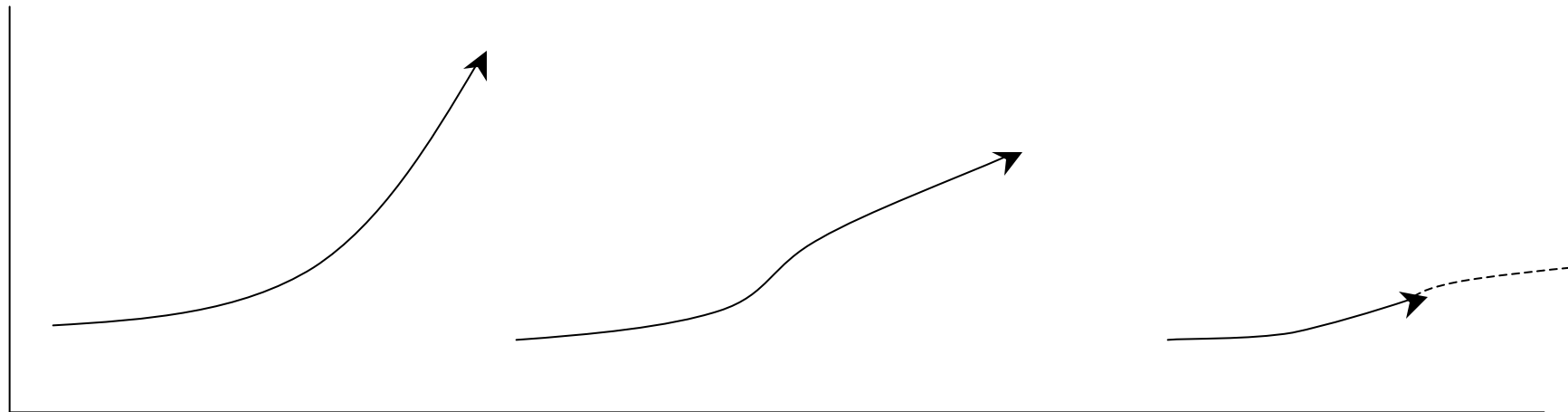
# Software Management Balance is Key

---



# Next-Generation Environments

---



## All R&D

- Custom tools
- Ad hoc methods and processes
- Numerous languages and formats
- Manual integration
- Programming focus

## R&D separate from construction

- Off-the-shelf tools
- Defined methods and processes
- Few languages and formats
- Some integration
- Design/integration focus

## R&D with automated construction

- Off-the-shelf tools
- Defined methods and processes
- Few languages and formats
- Highly integrated
- Architecture focus