Balancing Agility and Discipline
A Guide for the Perplexed

Written by

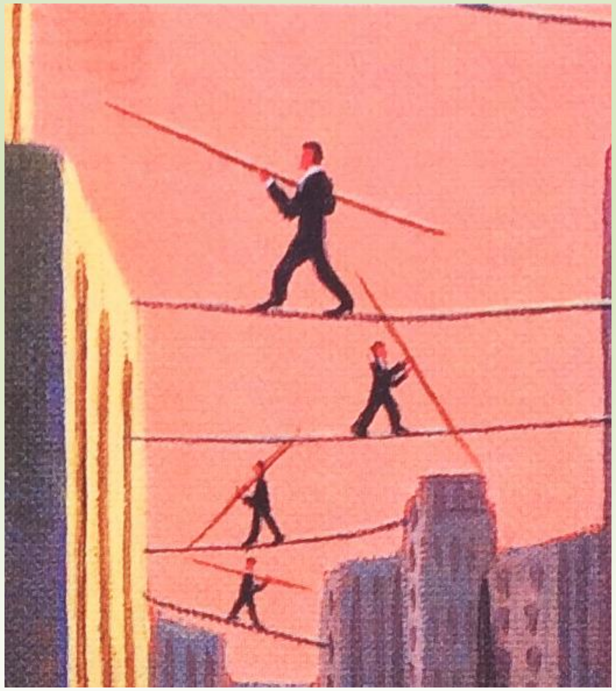
Barry Boehm & Richard Turner

August 2003

Presented by

Ben Underwood for EECS810

Fall 2015



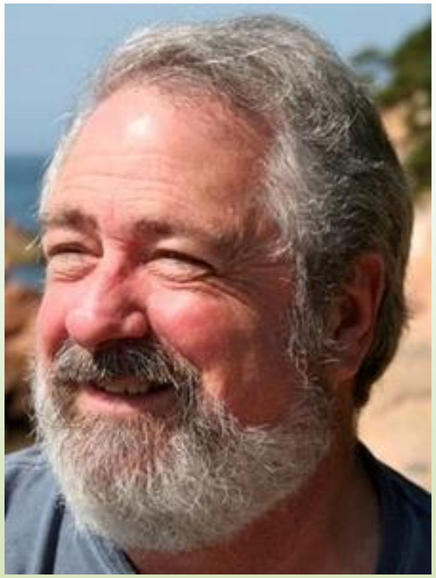
Presentation Outline
What to Expect

- Meet the Authors
- Discipline, Agility, and Perplexity
- Contrasts and Home Grounds
- A Day in the Life
- Expanding the Home Grounds: Two Case Studies
- Using Risk to Balance Agility and Discipline
- Conclusions
- Q & A

Meet the Authors
Barry Boehm

- Born 1935
- Educated in Mathematics at Harvard and UCLA
- Worked in Programming and Information Sciences in private industry and government
  - General Dynamics, Rand Corporation, TRW, and DARPA
- Currently at the University of Southern California
  - Professor of Software Engineering
  - Founding Director of USC's Center for Systems and Software Engineering

Meet the Authors
Richard Turner



- Born 1954
- Educated in Mathematics, Computer Science, and Engineering Management
- Worked in Computer Science, Technology, and Research in private industry and government
  - FAA, Systems Engineering Research Center, Software Engineering Institute, George Washington University, and more
  - One of the core authors of CMMI
- Currently at Stevens Institute of Technology
  - Professor in the School of Systems and Enterprises

Meet the Authors
Forewords

- Grady Booch
  - One of the three developers of UML
  - “…has helped me sort through… the current method wars.”
- Alistair Cockburn
  - One of the authors of the Agile Manifesto
  - “This is an outstanding book…”
- Arthur Pyster
  - COO of Systems Engineering Research Center
  - “…thoughtful analysis will help developers… sort through the agile-disciplined debate…”

Presentation Outline
Where Are We?

Discipline, Agility, and Perplexity
Main Points

- What is discipline?
- What is agility?
- The changing software environment
- Sources of perplexity
- Overview of plan-driven methods
- Overview of agile methods
- Finding middle ground

What is Discipline?

- The first sentence of the book:
  - “Discipline is the foundation for any successful endeavor.”
- The use of natural talent alone will lead to inconsistent successes at best
- The discipline of adhering to well-defined engineering processes can lead to long-term professional consistency
- Discipline provides a common and predictable organization of processes for an individual or team
- Provides strength and comfort in difficult circumstances
- Can spend a month or more planning before development

Examples of Discipline / Plan-Driven Methods

- Capability Maturity Model Integration (CMMI)
- Capability Maturity Model for Software (SW-CMM)
- Personal Software Process (PSP)
- Team Software Process (TSP)
- Cleanroom

What is Agility?

- Agility is the counterpart of discipline
- Applies memory and experience to adjust to new environments
- Reactive and adaptive
- Promotes invention and creativity
- Discipline without agility leads to bureaucracy and stagnation
- Agility without discipline is the boundless enthusiasm of a startup company before it has to turn a profit
- May only plan for days or hours before starting development

Examples of Agile Methods

- Scrum
- Adaptive Software Development (ASD)
- Lean Development (LD)
- Crystal
- Extreme Programming (XP)

The Changing Software Environment

- Software systems are growing in size and complexity
- Off-the-shelf components are playing greater roles
- Requirements are changing increasingly rapidly
- Software marketplaces are crowded
    - Quality and usability are more critical to success
    - Clients are demanding increasingly aggressive development timetables due to competition
- How do these things impact the effectiveness of different development methodologies?

Sources of Perplexity

- Multiple definitions
  - “discipline,” “agility,” and “quality”
- Method misuse
- Overgeneralization
  - Agile: XP, Disciplined: SW-CMM/CMMI
- Claims of universality
  - Both sides are guilty. No silver bullet!
- Early success stories
- Purist interpretations

Overview of Plan-Driven Methods *1 of 5*

- The "traditional" way to develop software
- History:
  - Developed concurrently by the US Dept. of Defense, IBM, Hitachi, Siemens, and others in the 1970s
  - Satellite, spacecraft, and missile development required the coordination of large numbers of interoperating components not necessarily produced by a single company or group of workers
  - Goal was to reduce chaos and lead to more predictable results
  - Based on systems engineering and quality disciplines taken from other engineering fields

Overview of Plan-Driven Methods *2 of 5*

- Originally a waterfall process with extensive documentation
  - Requirements → Analysis → Design → Coding → Testing → Operation
  - Move to next phase only after the preceding phase is verified
- More recent variations allow for incremental and iterative development, but still with extensive documentation
- Definition and management of processes is key
  - Detailed plans, workflows of prescribed activities, roles & responsibilities, and descriptions of intermediate work products

Overview of Plan-Driven Methods
3 of 5

- Important plan-driven concepts:
  - **Process capability** – ability of a process to produce planned results
  - **Organizational maturity** – a measure of process capability
  - **Process improvement** – activities to improve process capability
  - **Process group** – facilitators of a process within an organization
  - **Risk management** – an organized process to identify, assess, and quantify risks, and a plan to prevent or handle each one
  - **Validation** – confirms that the requirements are right
  - **Verification** – confirms that you are building to requirements
  - **Software system architecture** – a definition of components, connectors, and constraints to satisfy stakeholder needs

Overview of Plan-Driven Methods *4 of 5*

- Standardization provides repeatability
  - Personnel can move between projects without much retraining
  - Loss of key personnel will not doom a project
- Management support and organizational infrastructure is required
- Best characterized today by the SW-CMM
  - A road map of activities and practices to guide an organization through the software development process
  - Gained prominence in the late 1980s and early 1990s
  - Evolved into CMMI

Overview of Plan-Driven Methods *5 of 5*

- Other disciplined/plan-driven methodologies:
    - **Military standards** – DoD-STD-2167, MIL-1521, MIL-STD-498, MIL-STD-499B
    - **General process standards** – EIA/IEEE J-STD-016, ISO 9000, ISO 12207, ISO 15504
    - **Software factories** – Used by Hitachi and GE to achieve early defect reduction
    - **Cleanroom** – Used by IBM and Harlan Mills, has math-based verification
    - **PSP/TSP** – Used by SEI and Watts Humphrey, PSP advocates self-measurement, TSP is for teams

Overview of Agile Methods
1 of 9

- Internet-based economy demands speed and flexibility
- Programming viewed as a craft rather than a mechanical, industrial process
- Agile processes have less documentation, shorter cycle times, close customer involvement throughout the process, and an overall adaptive mindset
- “Chaordic” – the unification of chaos and order in a way that can’t be managed by traditional linear processes
- Widely adopted in the 2000s and 2010s

Overview of Agile Methods
2 of 9

- Best characterized by the Agile Manifesto, published by the Agile Alliance in 2001
- Four core values of the Agile Manifesto:
  1. **Individuals and interactions** over process and tools
  2. **Working software** over comprehensive documentation
  3. **Customer collaboration** over contract negotiation
  4. **Responding to change** over following a plan

Overview of Agile Methods
3 of 9

- Twelve principles of the Agile Manifesto:
    1. Satisfy customer through early and continuous delivery of software
    2. Welcome changing requirements, even late in development
    3. Deliver working software frequently; weeks or months between delivery, but preferably weeks
    4. Business people and developers must work together daily
    5. Build projects around motivated individuals; support and trust them
    6. Face-to-face conversation is the most effective communication

Overview of Agile Methods
4 of 9

7. Working software is the primary measure of progress
8. Work at a sustainable pace
9. Continuous attention to technical excellence and good design
10. Simplicity is essential
11. Self-organizing teams produce the best architectures, requirements, and designs
12. At regular intervals, the team reflects on how to be more effective, then adjusts accordingly

- Three areas of practices to enact the Agile Manifesto: Communication, Management, and Technical

Overview of Agile Methods
5 of 9

- A truly agile method must include all of the following attributes:
    - **Iterative** – Multiple development cycles
    - **Incremental** – Not all features are worked on at once
    - **Self-organizing** – Teams determine the best way to handle work
    - **Emergence** – Processes and work structures are determined during the project rather than before
- Anything less would be a lightened plan-driven process, rather than an agile methodology

Overview of Agile Methods
6 of 9

- Requires close customer involvement throughout development
- Requires a critical mass of knowledgeable and motivated team members
- Agile methodologies have mostly been used on projects with five to ten team members
- There is skepticism that pure agile methodologies can be used effectively with large, complex, or safety-critical software systems.

Overview of Agile Methods
7 of 9

- Important agile concepts:
  - **Embracing change** – It allows for more creativity and quicker value
  - **Fast cycle/frequent delivery** – Many short releases force feature prioritization, quick value, and speeds emergence of requirements
  - **Simple design** – YAGNI (You Aren't Going to Need It); change is inevitable, so planning for future features is unproductive
  - **Refactoring** – Improving software without changing its behavior
  - **Pair programming** – Two programmers, one computer; forced collaboration
  - **Retrospective** – Post-iteration review of effectiveness
  - **Tacit knowledge** – In teams' minds instead of on documentation
  - **Test-driven development** – Writing tests before and during coding

Overview of Agile Methods *8 of 9*

- Agile development methodology example: **Scrum**

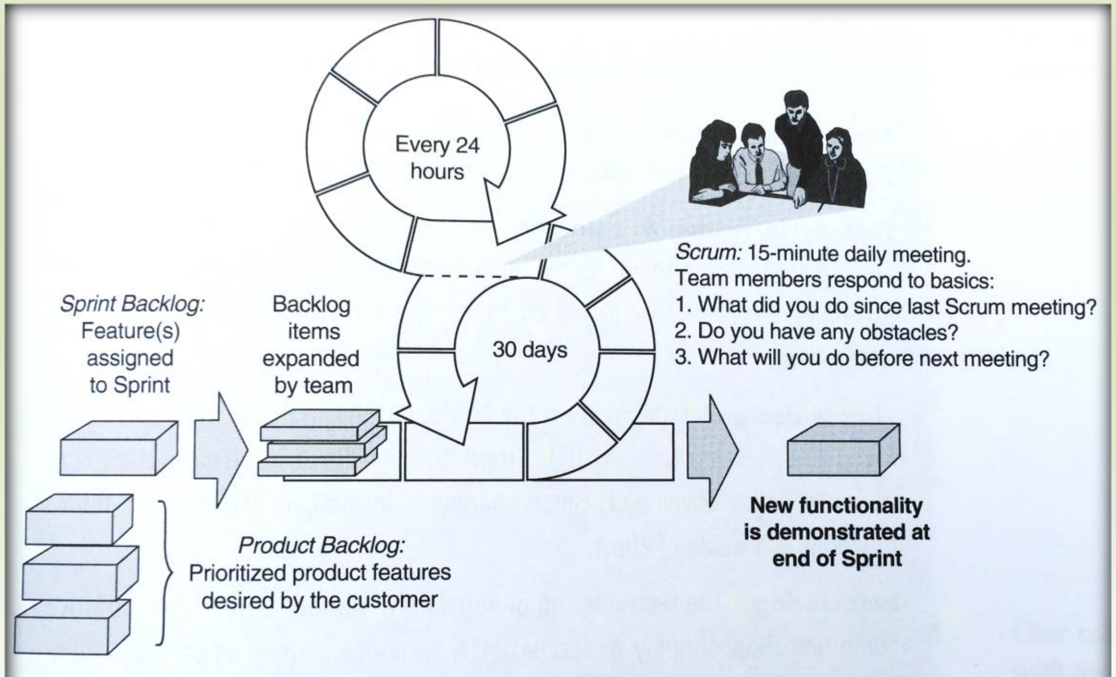

Every 24 hours
30 days
Scrum: 15-minute daily meeting.
Team members respond to basics:
1. What did you do since last Scrum meeting?
2. Do you have any obstacles?
3. What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to Sprint
Backlog items expanded by team
New functionality is demonstrated at end of Sprint
Product Backlog: Prioritized product features desired by the customer

Figure 1-2 Example Agile Process Flow (Scrum—Source: ControlChaos.com)

Overview of Agile Methods
9 of 9

- Other agile development methodologies:
    - **XP** – A fairly rigorous process that expects twelve specific practices to be followed, including pair programming
    - **ASD** – Uses feature-based planning, iterative development, customer focus-group reviews, and a collaborative management style
    - **Crystal** – Prescribes different levels of “ceremony” depending on the size of the team and criticality of the project
    - **FDD** – A lightweight process that establishes an overall object architecture and features list, then designs-by-feature and builds-by-feature

Finding Middle Ground *1 of 3*

- Home grounds are:
  - **Plan-driven** – Large, complex systems, often safety-critical or requiring high reliability. Requirements are stable and the environment predictable.
  - **Agile** – Systems and development teams are smaller, customer and users are readily available, and the requirements or environment are volatile.
- Successful, sustainable software development requires both discipline and agility

Finding Middle Ground

2 of 3

- Traditional environments need to react faster and stay relevant
- However the necessity of discipline remains, as software systems continue to grow in size and complexity
- Management should choose the development methodology for a project, based on:
  - the characteristics of the project
  - the size and capability of the development team
  - and its environment, such as budget, schedule, and criticality

Finding Middle Ground
3 of 3

- Risk is the key
  - schedule slip, cost overrun, technical failure, etc.
- Risk analysis can reveal the best methodology to choose for a project
- Ask the following about every potential development process: Is it riskier for me to apply more of this process or less of it?
- Analyzing risk in that way can lead to effective hybrid methodologies that balance discipline and agility

Presentation Outline
Where Are We?

- Meet the Authors
- Discipline, Agility, and Perplexity
- **Contrasts and Home Grounds**
- A Day in the Life
- Expanding the Home Grounds: Two Case Studies
- Using Risk to Balance Agility and Discipline
- Conclusions
- Q & A

Contrasts and Home Grounds

- What software project characteristics can be used as comparison points for the performance of agile vs. plan-driven methodologies?
- Four primary project characteristics to consider:
  - Application characteristics
  - Management characteristics
  - Technical characteristics
  - Personnel characteristics

Project Application Characteristics

- Primary Goals
  - **Agile** – Rapid value and responsiveness to change
  - **Plan-driven** – Predictability, stability, and high assurance
- Size
  - **Agile** – Small to medium teams, relatively small applications
  - **Plan-driven** – Large projects
- Environment
  - **Agile** – Turbulent, high-change environments
  - **Plan-driven** – Requirements can mostly be determined in advance, and they remain relatively stable

Project Management Characteristics

- Customer Relations
  - **Agile** – Dedicated, full-time, on-site customer representative
  - **Plan-driven** – Up-front contract negotiation
- Planning and Control
  - **Agile** – 20% of time spent planning, often done as a group
  - **Plan-driven** – Heavy documentation provides coordination
- Project Communication
  - **Agile** – Tacit knowledge, face-to-face communication
  - **Plan-driven** – Documented knowledge, written communication

Project Technical Characteristics

- Requirements
  - **Agile** – Adjustable, informal stories, prioritized by customer
  - **Plan-driven** – Stable, complete, consistent, traceable, testable
- Development
  - **Agile** – Simplest possible design for current requirements/YAGNI
  - **Plan-driven** – Big Design Up Front (BDUF) to accommodate foreseeable change
- Testing
  - **Agile** – Tests developed before code, testing is incremental
  - **Plan-driven** – Early focus is on consistent and testable requirements and architecture, for later automated testing

Project Personnel Characteristics

- Customers
  - **Agile** – CRACK customer representatives (Collaborative, Representative, Authorized, Committed, and Knowledgeable)
  - **Plan-driven** – Also require CRACK customer reps, but not full-time
- Developers
  - **Agile** – Universally talented, communicative, motivated
  - **Plan-driven** – Needs top people for design, less-capable can build
- Culture
  - **Agile** – Dev team likes wide freedom to define and solve problems
  - **Plan-driven** – Dev team prefers clear policies and procedures

Common Misconceptions
Plan-Driven Methods

- Uniformly bureaucratic
  - **Reality** – Too much bureaucracy is bad for software development
- Documentation guarantees compliance
  - **Reality** – Not necessarily
- Can succeed with lack of talented people
  - **Reality** – Can succeed with a smaller % of talented people
- High process maturity guarantees success
  - **Reality** – Documented plans provide a safety net
- Works with both foreseeable and unforeseeable change
  - **Reality** – Works best with foreseeable change

Common Misconceptions
Agile Methods

- No planning
    - **Reality** – Planning is tacit and face-to-face
- All team members must be talented
    - **Reality** – A critical mass must be highly talented
- Eliminates the cost of change
    - **Reality** – Reduces the cost of change
- YAGNI is universally safe
    - **Reality** – Risky for foreseeable change

Five Critical Project Factors

- Size – How big is the project? How big is the dev team?
- Criticality – What is the risk of under-performing?
- Dynamism – Are the requirements stable or dynamic?
- Culture – Does the dev team thrive on chaos or order?
- Personnel – What % of the dev team are experts?

Project Polar Chart

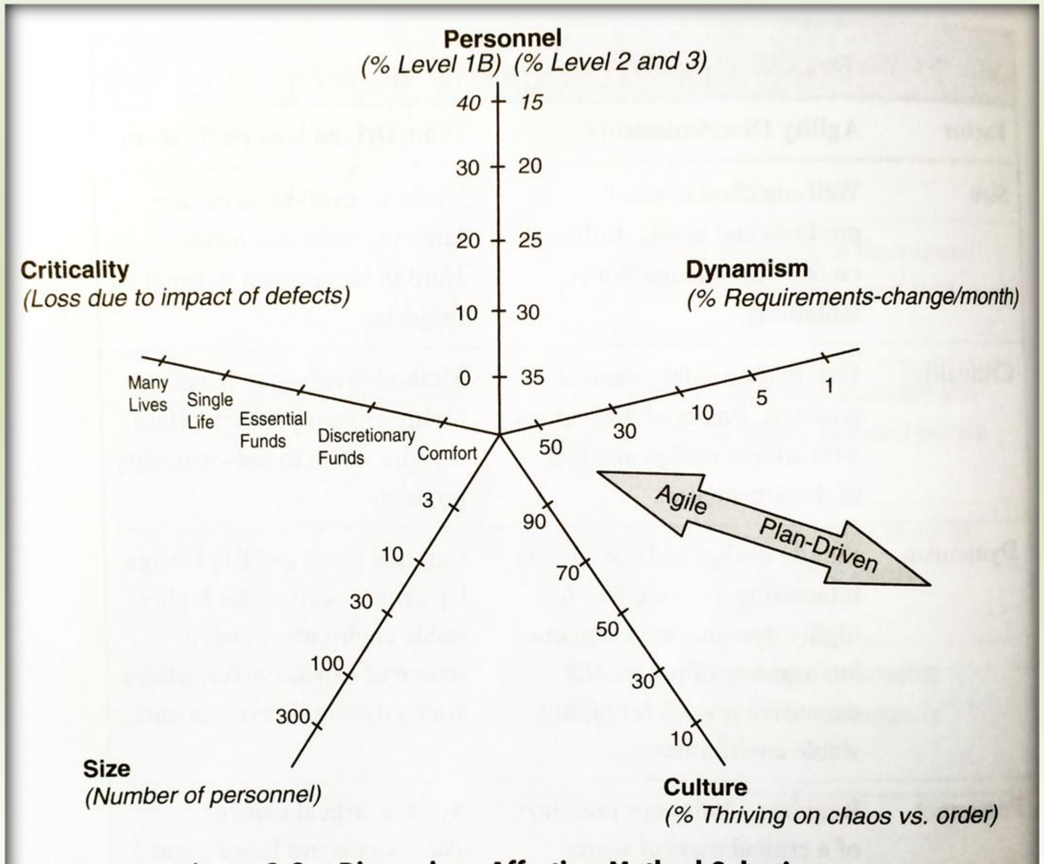


Figure 2-2 Dimensions Affecting Method Selection

Project Polar Chart Explained

- Size – Focuses on the size of the dev team
- Criticality – Maximum impact to customer of project failure
- Dynamism – % of requirements expected to change per month
- Culture – % of dev team that thrives on chaos (vs. order)
- Personnel – % of dev team at different levels of competency
  - Requires an analysis of each dev team member, using a modified version of Alistair Cockburn's levels of software method understanding (the same Cockburn from the foreword of this book)
  - More details on the next slide

Cockburn Levels of Development Personnel (Modified)

- Level 3 – Can revise/break a software method to fit a new situation
- Level 2 – Can tailor a software method to fit a new situation
- Level 1A – With training, can perform optional/discretionary software method steps
- Level 1B – With training, can perform required/procedural software method steps
- Level -1 – May have technical skills, but unwilling or unable to collaborate or follow software methods

Project Polar Chart Revisited

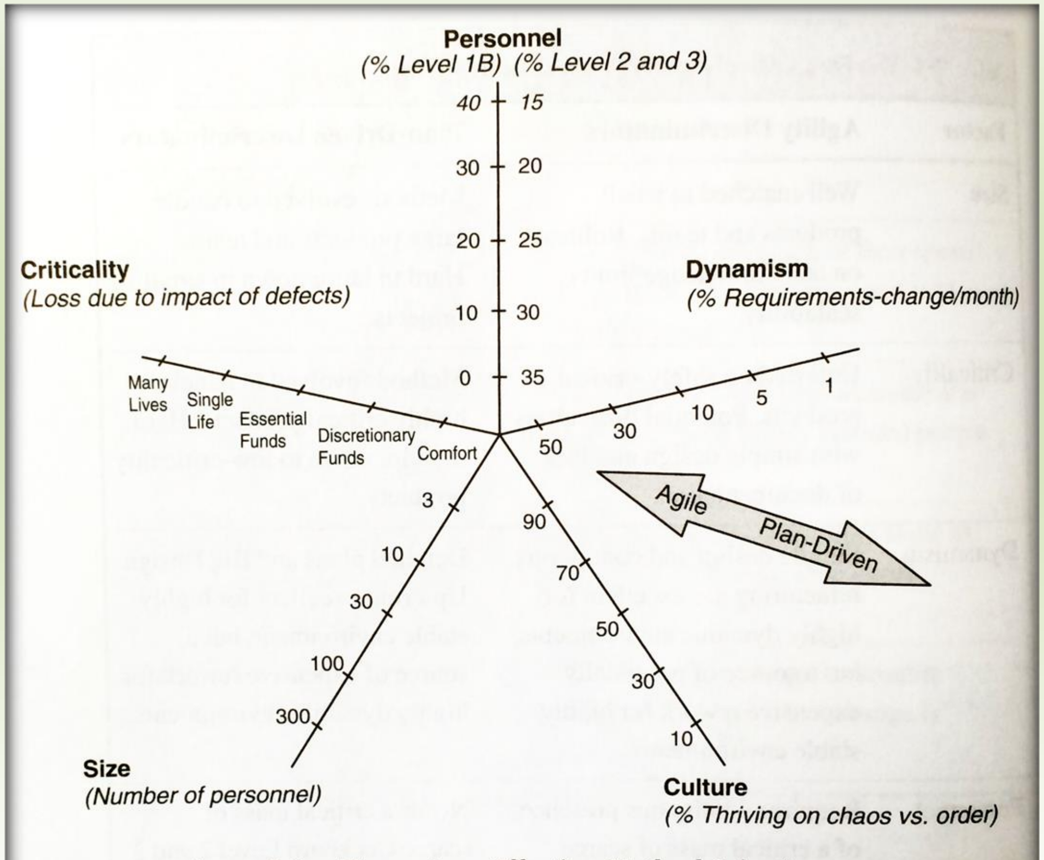


Figure 2-2 Dimensions Affecting Method Selection

Presentation Outline
Where Are We?

- Meet the Authors
- Discipline, Agility, and Perplexity
- Contrasts and Home Grounds
- **A Day in the Life**
- Expanding the Home Grounds: Two Case Studies
- Using Risk to Balance Agility and Discipline
- Conclusions
- Q & A

A Day in the Life

- What does it "feel like" to develop with agile and disciplined methods?
- Example development project is a tool that processes a complex sales reporting and inventory management file
  - Estimated at 20 KLOC and 8 months duration
- Example methods to develop that project:
  - Disciplined – a team using the PSP/TSP method
  - Agile – a team using the XP method
- Example days in each method:
  - Typical day
  - Crisis day

More Details on Each Team
Disciplined (PSP/TSP) Team – 1 of 2

- **Team Size** – 9 team members
- **Method Training** – 3-week course for all team members; additional 1-week course for two team members
- **Cockburn Level Skills** – Two level 2; Five level 1A; Two level 1B
- **Project Roles** – Each team member has a different role
  - Team Leader
  - Programmer/Analyst
  - Implementation Manager
  - Planning Manager
  - Design Manager
  - Quality/Process Manager
  - Support Manager
  - Customer Interface Manager
  - Test Manager

More Details on Each Team
Disciplined (PSP/TSP) Team – 2 of 2

- **Tools** – Web-based software for PSP/TSP data collection and reporting
- **Office Layout** – Each developer has their own office or cubicle, and their own computer. One cubicle is reserved for a dedicated testing computer. A conference room is available.
- **Project Planning**
  - Project started with a four day planning session including all team members.
  - 180 separate tasks identified and planned.
  - Process documentation rules relaxed some for the prototyping phase.
- **Project Status** – In third month. A prototype has been demonstrated to management, but nothing has been delivered to the client. Integration testing for the first phase is scheduled to begin in one week.

More Details on Each Team
Agile (XP) Team – 1 of 2

- **Team Size** – Also 9 team members
- **Method Training** – 1-week course for two team members; semi-formal in-house training for the rest of the team
- **Cockburn Level Skills** – Two level 2; Seven level 1A
- **Project Roles**
  - Coach – 1 team member
  - Customer – 1 team member
  - Tester – 1 team member
  - Tracker – 1 team member
  - Programmer – 5 team members (this is the main role in XP projects)

More Details on Each Team

Agile (XP) Team – 2 of 2

- **Tools** – Automated tool to support testing
- **Office Layout** – Open bullpen, with whiteboards and workstations for pair programming. Limited private space is provided. A comfortable lounge room is stocked with refreshments.
- **Project Planning**
  - Project started with a one day exploration session, followed by a two day planning session.
  - Story cards are documented and prioritized, two week iterations are planned.
  - Tasks are identified from stories and assigned.
- **Project Status** – Second iteration release is due next week. First release contained two less stories than planned, but customer is happy.

Disciplined (PSP/TSP) Typical Day

- **8:30am** – Organizational staff meeting
- **9-10:30am** – Unit test development; Personal code review with detailed logging of all defects and to-the-minute tracking of time spent on the task; Divisional strategic planning meeting; Review of project metrics to look for problems such as too much time spent coding vs. time spent designing, or too many lines of code being reviewed per hour.
- **11am-12:30pm** – More metric review to prepare for workshop on next project cycle; Continual to-the-minute tracking of project task time.
- **1-4:30pm** – Detailed design inspection, with number, severity, and location of defects documented.
- **2-3pm** – Customer meeting
- **3-4pm** – Integration testing meeting
- **4:30-5:30pm** – Team status meeting including Role Report, Risk Report, Goal Report, and Customer Report. Actual project "earned value" number calculated from cumulative metrics is compared to the planned value.

Agile (XP) Typical Day

- **8:30am** – Standup status and planning meeting; everyone in a circle
- **9-10:30am** – Pair programming, with test cases developed before coding; Customer meeting to discuss requirement changes using mockups; Verifying availability of funding for completion bonuses
- **11am-12:30pm** – Programmers decide to perform a spike (narrow but deep coding experiment); Refactoring to accommodate requirement changes
- **1:30-3pm** – Testing design with customer; Testing; Documentation of test design; Pair programming; Refactoring; Prototype review with customer
- **3:30-5pm** – Testing; Documentation of test results; Code corrections
- **5-5:30pm** – Team status meeting in the lounge; Documentation of progress

Disciplined (PSP/TSP) Crisis Day

- **Crisis** – On Friday, customer CEO requests a complex new report due on Wednesday.
  Changes required to multiple code modules, the database, and two GUIs.
- **9am** – Assignments are made to estimate team's ability to complete the change in time; Four hours given to complete those assignments
- **1pm** – Team determines based off of their productivity metrics that they can complete the change in time if the next release can be delayed; Customer agrees; Design tasks begin, and go smoothly due to UML tools and an original architecture which had anticipated a change like this
- **5pm** – Several teams members are assigned to work the weekend to update various project documentation like plan documents, earned value system, and test plans and procedures

Agile (XP) Crisis Day

- **Crisis** – On Friday, customer CEO requests a complex new report due on Wednesday. Changes required to multiple code modules, the database, and two GUIs.
- **9am** – Team works out the replanning strategy at their daily standup meeting; Assignments are made to write and estimate tasks
- **10:30am** – Team reconvenes to discuss the impact of the changes, works with customer to reduce the number of stories in the current release
- **11am** – Pair work on writing tests for the new features
- **1pm** – Pair programming to add new features, and to undo partially completed stories that are moving to the next release; Automated testing tool identifies errors that propagate to other parts of the system
- **5pm** – Team determines they will finish in time; no weekend work needed

A Day in the Life Summary *Differences*

- **Project tracking metrics**
  - PSP/TSP measures detailed metrics for process control, quality, and performance
  - XP metrics are for estimating progress/scheduling
- **Process prescription detail**
  - PSP/TSP has many detailed roles, scripts, forms, and exit criteria
  - XP has some strict practices but few overall guidelines
- **Reporting**
  - PSP/TSP has reports for nearly every task and phase
  - XP reports are more informal and related to scheduling
- **Customer interaction**
  - PSP/TSP is formal/contractual
  - XP is co-located/collaborative

A Day in the Life Summary
Similarities

- Both have specific team roles and responsibilities
- Both have iterative planning and execution cycles
- Both require documentation, such as performance measurement to support estimation
- Both require test development before coding
- Both support pushing back against unreasonable customer requirements when necessary. In other words, they support having the preparedness and courage to manage customer expectations appropriately, in order to meet schedules and keep failure rates low.

Presentation Outline
Where Are We?

- Meet the Authors
- Discipline, Agility, and Perplexity
- Contrasts and Home Grounds
- A Day in the Life
- **Expanding the Home Grounds: Two Case Studies**
- Using Risk to Balance Agility and Discipline
- Conclusions
- Q & A

Expanding the Home Grounds: Two Case Studies

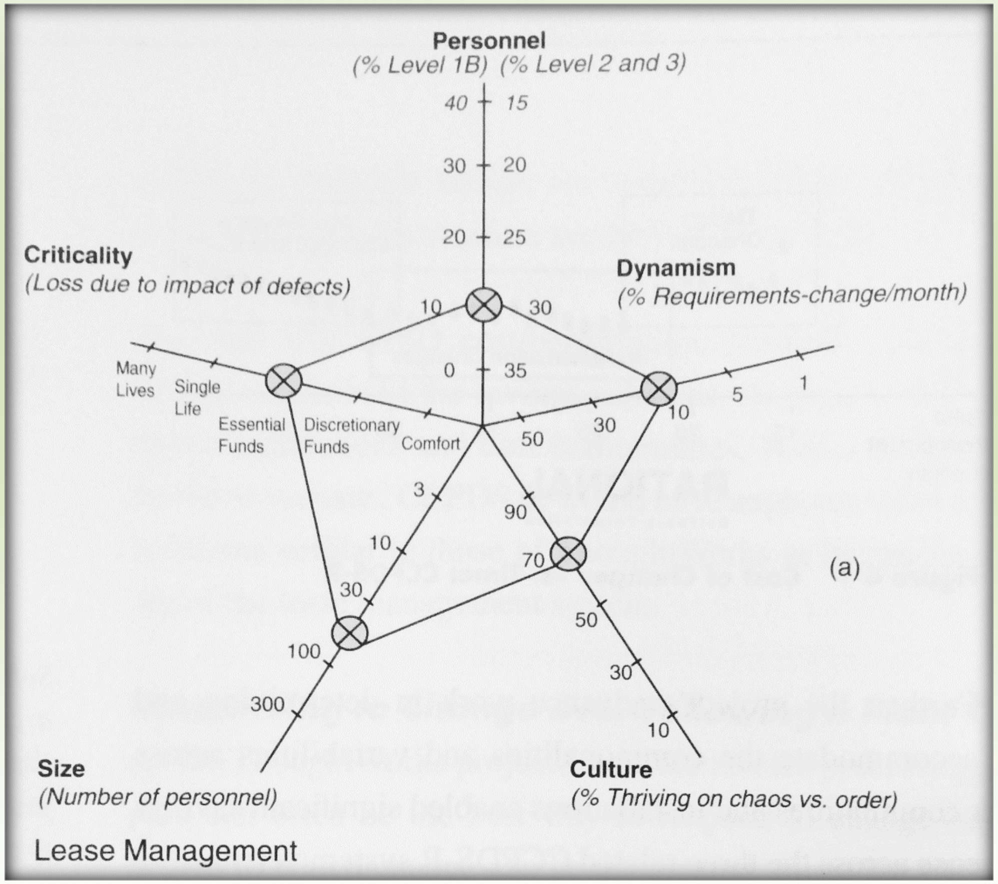
- How can methods be combined?
- Case study 1 – Lease management
  - Using plan-driven techniques to scale up an agile method
- Case study 2 – CCPDS-R
  - Using agile techniques to streamline a traditional plan-driven method

Lease Management
Case Study 1

- The project was an enterprise resource solution for the leasing industry
- 1000 story cards, 500 KLOC
- 50 team members (30 developers, 20 customer domain experts) writing in J2EE
- Used a traditional plan-driven development method for the first 18 months. Found it ineffective for this project.
- Switched to XP method, and used that for the next 3 years

Lease Management

Polar Chart



Lease Management
Three Issues Implementing XP

- The team recognized and resolved three main issues with using the XP method for their project
  - **Issue 1** – The effort to develop or change the project's stories increased with time and the number of stories
  - **Issue 2** – Trusting people to get everything done on time was incompatible with the project's fixed schedule
  - **Issue 3** – Simple design and YAGNI didn't scale up easily to the large project

Lease Management
Issue 1 – Tacit Knowledge Problems

- Tacit knowledge was a difficult approach to maintain with:
  - The large size of the project. Changing a lease involved working with 100 objects.
  - Changing personnel. Developers changed between iterations.
- Time to develop a story card increased in later iterations
- Also in later iterations, functions would pass their individual tests, but often fail integration testing
- **Solution** - More high-level architectural planning needed; but it can still be modified at any stage of development

Lease Management
Issue 2 – Schedule Pressure Problems

- Customer representatives were less than ideal; didn’t fully understand end user needs
- The time necessary for integration wasn’t properly planned for
- Schedule pressure side effects:
  - Work went undone because all hoped someone else would do it
  - “Tragedy of the commons” – Barely enough time for assigned work, so no one sees to the common need
- **Solution** – More precise list of tasks created for each story

Lease Management

Issue 3 – Simple Design & YAGNI Problems

- Constant refactoring due to new invoice format requirements
- Different tests required similar test drivers. Repeated development of each similar test drivers was time-consuming.
- **Solution** – Implemented a software development pattern for these problems. More up-front work, so violated YAGNI, but saved time overall, especially given that the need/benefit was foreseeable

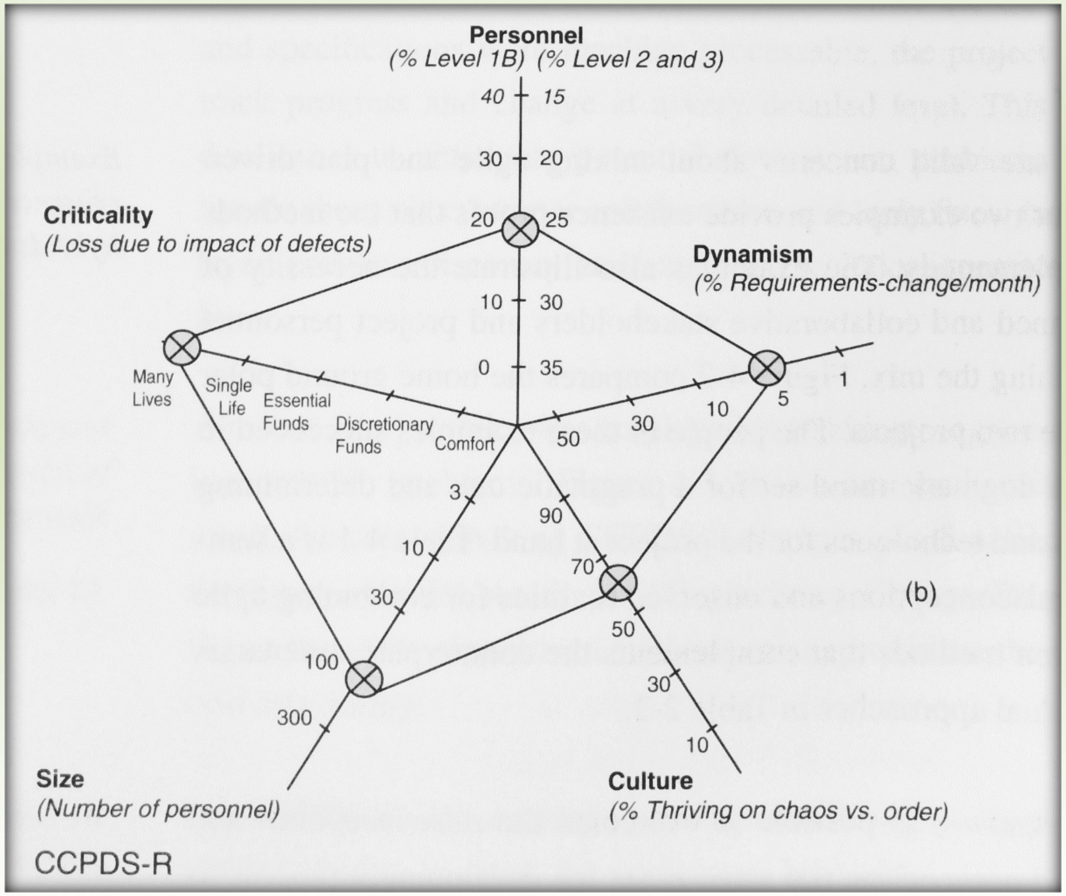
Lease Management
Summary

- The project team customized an enhanced XP method for their large project
- Tacit knowledge promotes agility, but has scaling problems
- YAGNI is risky with large projects and foreseeable change
- Plan-driven elements were to XP added in needed areas:
    - High-level architectural plans
    - More detailed itemizing and monitoring of tasks for milestone completion
    - Use of design patterns rather than YAGNI to accommodate foreseeable repeated work

CCPDS-R
Case Study 2

- The project was rebuilding the hardware/software for the US early missile warning system command center, the Command Center Processing and Display System Replacement (CCPDS-R)
- Used by the US Space Command, US Strategic Command, US National Command Authority, and all nuclear-capable Commanders in Chief
- 1000 KLOC, 75 team members writing in Ada
- 4-year development contract, required to follow a relatively strict DoD quality process, but used a relatively lean/agile RUP
- Numerous high risk requirements and infrastructure challenges

CCPDS-R
Polar Chart



CCPDS-R
Agility Added to Plan-Driven Method

- Added agility in four categories, corresponding with the four value propositions of the Agile Manifesto:
  - **1** – “Individuals and interactions over processes and tools”
  - **2** – “Working software over comprehensive documentation”
  - **3** – “Customer collaboration over contract negotiation”
  - **4** – “Responding to change over following a plan”

CCPDS-R
Agile Value 1 – Individuals & Interactions

- Specific DoD processes and method management tools were necessary, but where possible:
  - They chose the processes and tools that best served the project’s stakeholders
  - They adapted the processes and tools to the needs of individuals and interactions, rather than vice versa
- Automated verification tools
- Performance bonuses for developers
  - Increased motivation and teamwork; decreased developer turnover
- Specific tasking was based on developer skill level

CCPDS-R
Agile Value 2 – Working Software

- Changed a DoD-required Preliminary Design Review from a document review process to a working software prototype review
    - And negotiated for enough time to accomplish that
- Automatically generated DoD-required documentation where possible
    - And generated them in machine-processable formats where possible
- Developed tools to support continuous testing

CCPDS-R
Agile Value 3 – Customer Collaboration

- Used an Ada version of COCOMO to:
  - Estimate and negotiate the development schedule
  - Negotiate awards for the project’s high performers
  - Identify and design for re-use opportunities between’s modules for the project’s various user groups

CCPDS-R
Agile Value 4 – Responding to Change

- Using machine-generated and machine-processable plan and specification documentation made it:
  - Easier to update various documentation when necessitated by a change
  - Easier to identify potential downstream problems when a change occurred
- Planned component re-use limited the impact of some changes

CCPDS-R
Summary

- Rational Unified Process (RUP) guidelines currently lack specificity in regards to how to balance agility and discipline, so for now, project leaders must be either ambitious enough or experienced enough to do so
- The CCPDS-R project team was ambitious enough and sophisticated enough to determine the best tools and techniques for the project at hand
  - They added key agile concepts to a project that had the constraint of requiring a high degree of discipline
  - As a result they delivered on time/budget and to a satisfied customer

Presentation Outline
Where Are We?

- Meet the Authors
- Discipline, Agility, and Perplexity
- Contrasts and Home Grounds
- A Day in the Life
- Expanding the Home Grounds: Two Case Studies
- **Using Risk to Balance Agility and Discipline**
- Conclusions
- Q & A

Using Risk to Balance Agility and Discipline

- A way to plan for a project to incorporate agility and discipline in proportion with a particular project's needs
- Based on the risks of using agile or plan-driven methods for the project in question
- The five-step, risk-based method developed by Boehm and Turner for this purpose relieson the ability of key development team members to understand organizational capabilities, and to identity and collaborate with project stakeholders
- Have to balance the risks of doing too little of each thing, with the risks of doing too much of it

Five-Step Method for Balancing Agile and Plan-Driven Methods

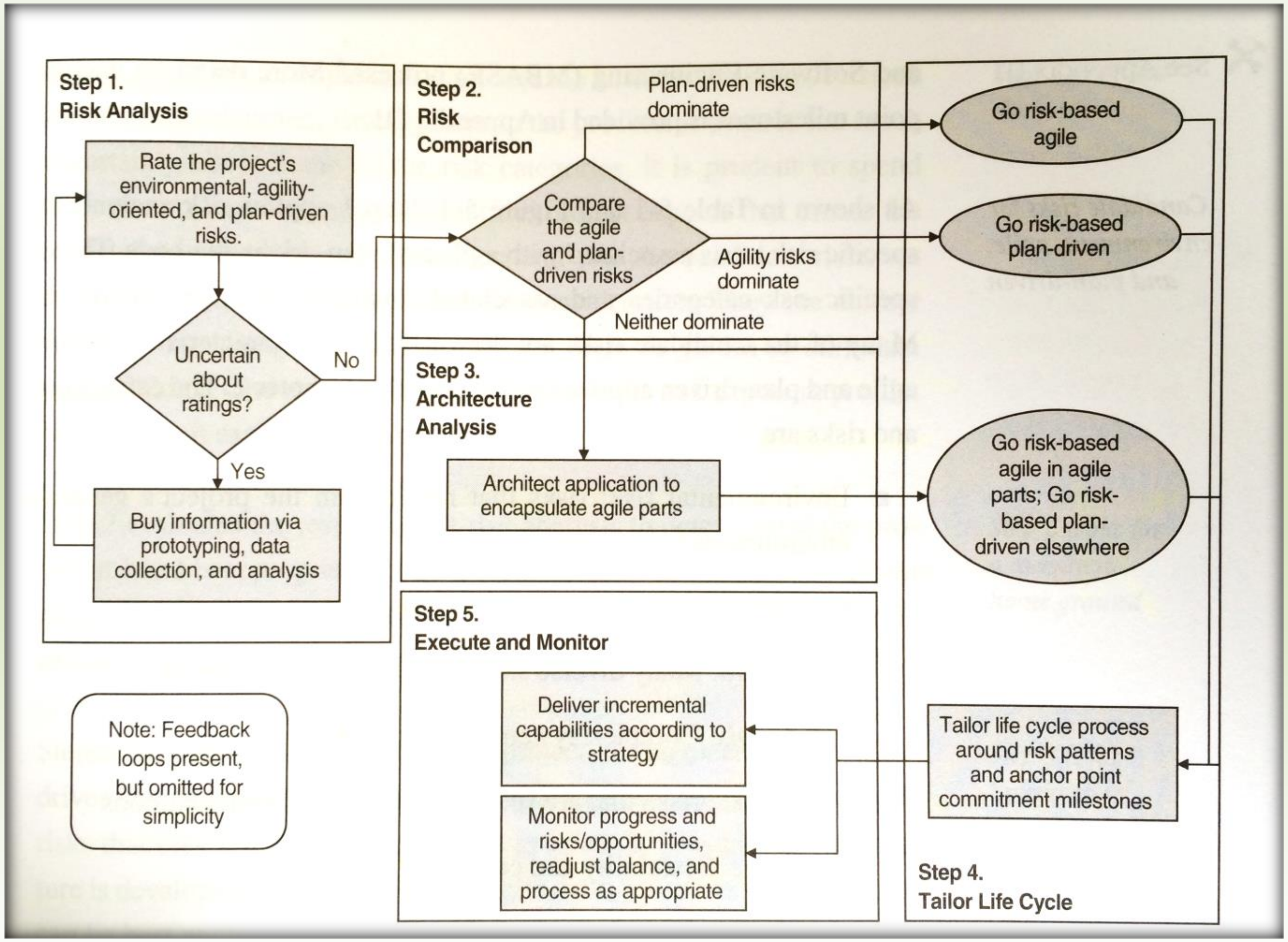
1 of 2

- **Step 1** – Identify and rate the environmental, agile, and plan-driven risks. If uncertain, try prototyping or other data collection methods. Use the polar chart to visualize the data.
- **Step 2a** – If agility risks dominate plan-driven risks, use a risk-based plan-driven development method.
- **Step 2b** – If plan-driven risks dominate agile risks, use a risk-based agile development method.
- **Step 3** – If the risks are mixed, create a project-specific hybrid development method. You could architect the application to encapsulate the agile parts, use a risk-based agile method for the encapsulated agile parts, and use a risk-based plan-driven method elsewhere. When in doubt, it is safest for plan-driven to be the default.

Five-Step Method for Balancing Agile and Plan-Driven Methods *2 of 2*

- **Step 4** – Create an overall project strategy. This is a project management and development plan which integrates risk mitigation plans for each individual risk identified in step 1.
- **Step 5** – Continually improve your capabilities. This includes improving the development capabilities, value-oriented capabilities, communication capabilities, and expectations management capabilities.
  - Always track progress with respect to plans and apply corrective action whenever the opportunity arises.

Five-Step Balancing Method Flowchart



Step 1.
Risk Analysis
Rate the project's environmental, agility-oriented, and plan-driven risks.
Uncertain about ratings?
No
Yes
Buy information via prototyping, data collection, and analysis
Step 2.
Risk Comparison
Plan-driven risks dominate
Compare the agile and plan-driven risks
Agility risks dominate
Neither dominate
Go risk-based agile
Go risk-based plan-driven
Step 3.
Architecture Analysis
Architect application to encapsulate agile parts
Go risk-based agile in agile parts; Go risk-based plan-driven elsewhere
Step 5.
Execute and Monitor
Deliver incremental capabilities according to strategy
Monitor progress and risks/opportunities, readjust balance, and process as appropriate
Tailor life cycle process around risk patterns and anchor point commitment milestones
Step 4.
Tailor Life Cycle
Note: Feedback loops present, but omitted for simplicity

Five-Step Method for Balancing Agile and Plan-Driven Methods

Step 1 Detail – 1 of 2

- Categories of environmental risks
    - **E-Tech** – Technology uncertainties
    - **E-Coord** – Many diverse stakeholders to coordinate
    - **E-Cmplx** – Complex system of systems
- Categories of agile risks
    - **A-Scale** – Scalability and criticality
    - **A-YAGNI** – Extensive refactoring due to use of simple/YAGNI design
    - **A-Churn** – Loss of tacit knowledge due to personnel turnover
    - **A-Skill** – Not enough people skilled in agile methods

Five-Step Method for Balancing Agile and Plan-Driven Methods

Step 1 Detail – 2 of 2

- Categories of plan-driven risks
  - **P-Emerge** – Emergent requirements
  - **P-Change** – Rapid changes in technology, market conditions, etc.
  - **P-Speed** – Need for rapid results
  - **P-Skill** – Not enough people skilled in plan-driven methods
- Risk analysis provides the basis for decision-making about the project
- If not enough is known about the risks, it is prudent to spend the time now to figure it out. You can prototype or use other methods to collect more risk data.

Five-Step Method for Balancing Agile and Plan-Driven Methods

Step 2a/2b Detail

- Evaluate the results of step 1’s risk analysis
- The project may be appropriate for a purely agile or purely plan-driven method
- If so, select a purely agile or purely plan-driven development methodology
- Even with pure methodologies, account for the risks that were identified

Five-Step Method for Balancing Agile and Plan-Driven Methods

Step 3 Detail

- If the project is not appropriate for purely agile or purely plan-driven, segment the parts of the application that are appropriate for agile.
- Use a risk-based agile methodology for the segmented parts
- Use a risk-based plan-driven methodology for everything else
    - Plan-driven methodology is the default state

Five-Step Method for Balancing Agile and Plan-Driven Methods
Step 4 Detail

- Develop an overall project strategy to address the risks identified in step 1
- Create risk resolution strategies for each of the risks and integrate those into the strategy
- Recommended next step is holding a Life Cycle Architecture Review

Five-Step Method for Balancing Agile and Plan-Driven Methods

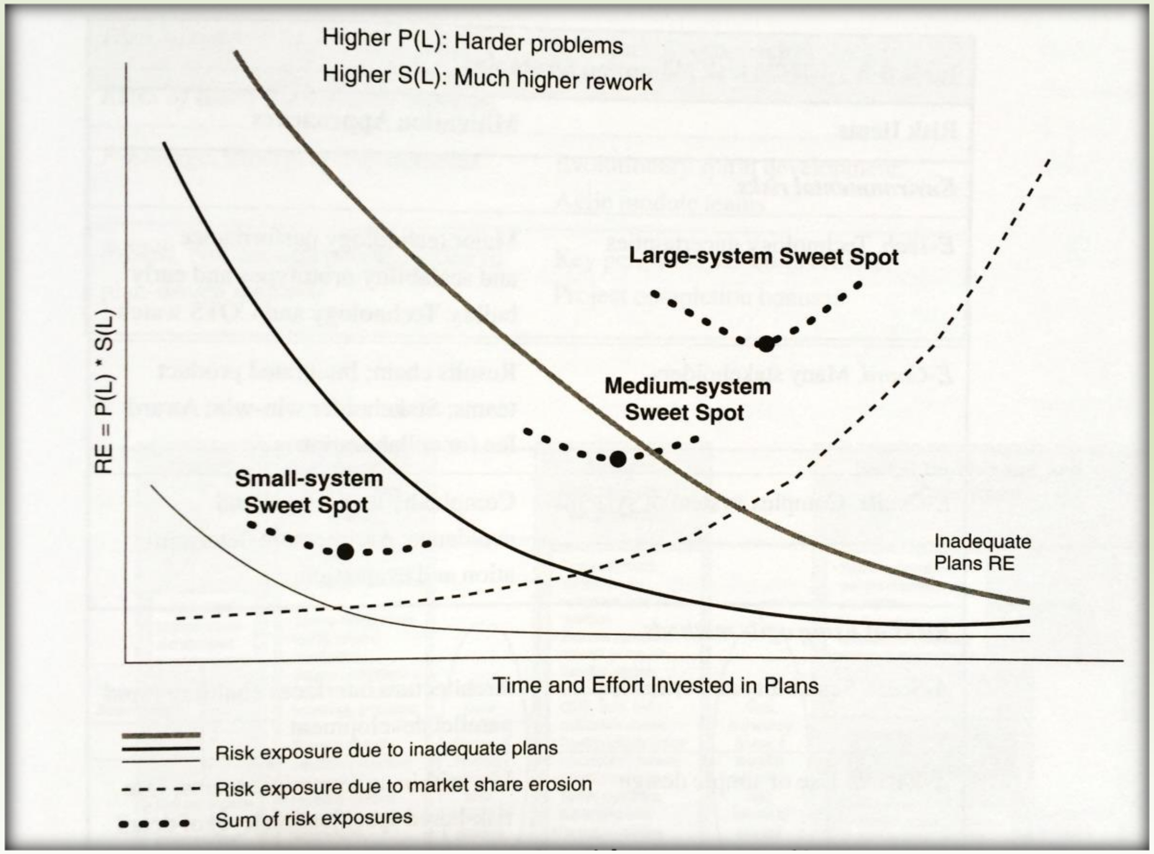
Step 5 Detail

- No decision is perfect or valid forever
- Management should constantly monitor and evaluate the environment and the performance of its selected processes
- New risks or opportunities can emerge, and re-adjust as necessary
- Similar to agile practice called “reflection”

Risk Exposure
Definition

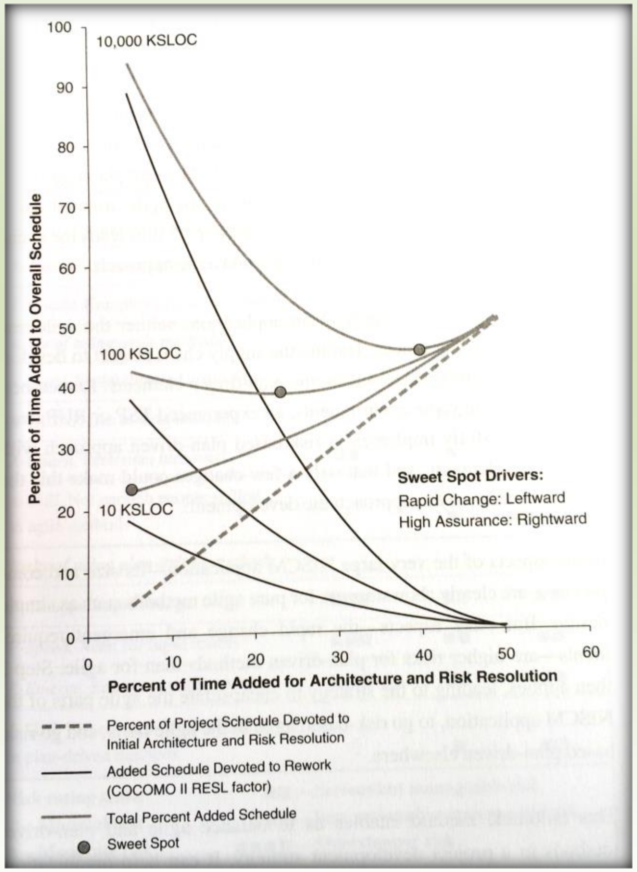
- Risk Exposure equals the probability of the loss times the size of the loss
- RE = P(L) x S(L)
- Losses can include
  - Revenue
  - Reputation
  - Quality of life
  - Life itself!
  - Any other value-related attributes

Risk Exposure
Variation by Time Spent Planning



How Much Time to Spend Planning?

Sweet Spot Depends on System Size



The Balancing Method in Practice
Three Sample Projects

- SupplyChain.com
  - Medium-sized application
  - Intermediate complexity
- Event Planning
  - Small application
  - Relatively non-critical
- National Information System for Crisis Management (NISCM)
  - Very large application
  - Highly critical

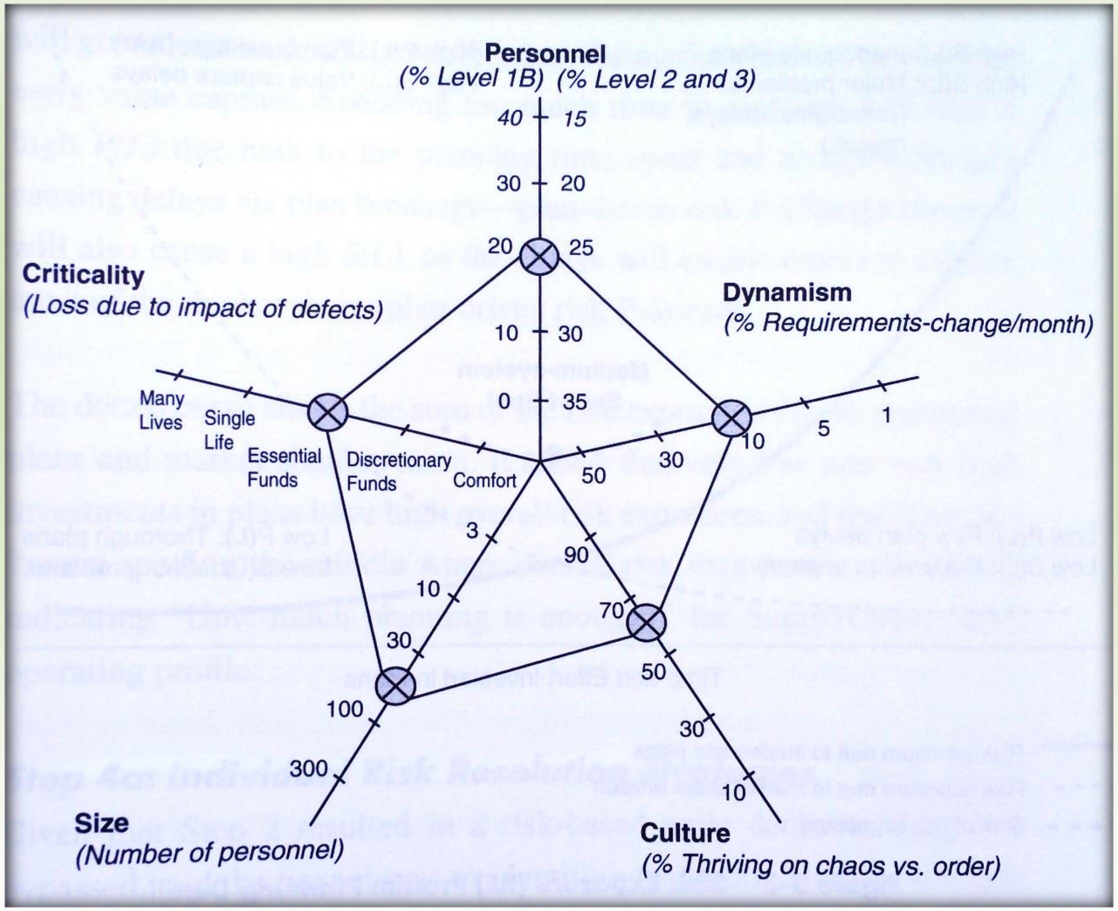
SupplyChain.com
Application Characteristics

- Supply chain management software
- For large manufacturing companies with complex networks of suppliers
- Includes some COTS packages (commercial off the shelf software), but mostly custom-built for each manufacturer
- Dev team – 50 people in multiple locations and sometimes in multiple organizations
- Primary objective is to provide a rapid increase in value to the manufacturer by increasing supply chain speed, dependability, and adaptability

SupplyChain.com
Five-Step Balancing Method Analysis

- Step 1 – E-Tech, E-Coord, E-Cmplx, A-Scale, A-YAGNI, A-Churn, P-Change, P-Speed, P-Emerge, P-Skill
- Step 2 – Agile and plan-driven risks are roughly equal
- Step 3 – A hybrid approach could work. However, they selected risk-based agile due to the primary objective of a rapid increase in value, and the dev team's agile culture
- Step 4 – Detailed project strategy created
- Step 5 – Risk/opportunity management team formed

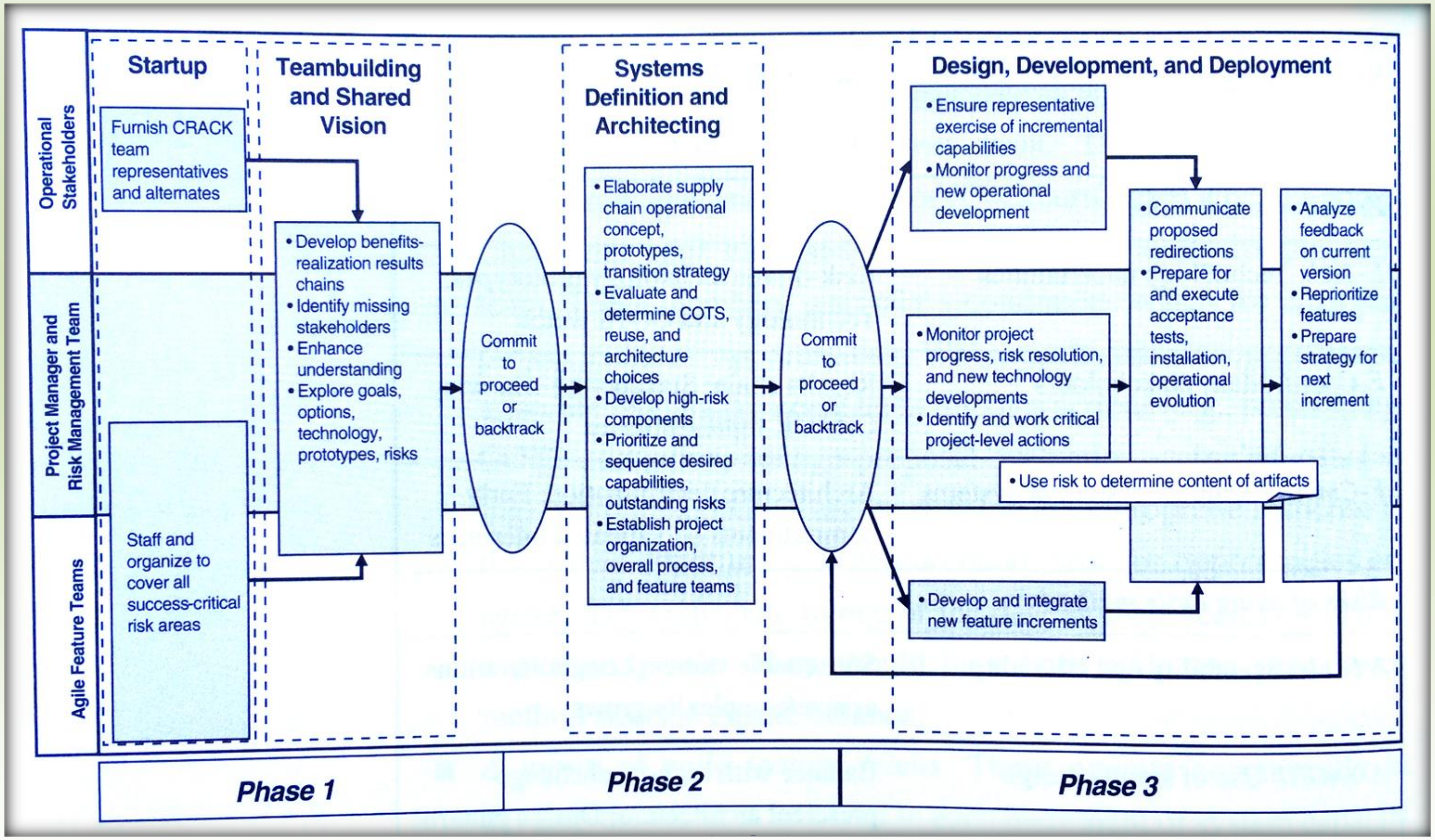
SupplyChain.com
Polar Chart



SupplyChain.com
Risk Mitigation

- E-Tech – Prototyping and ongoing review of emerging COTS
- E-Coord – Results Chain coordination, ensuring CRACK representatives
- E-Cmplx – Early commitments on validated interfaces
- A-Scale – Longer iterations as size and complexity grow
- A-YAGNI – Foreseeable change in isolated design modules
- A-Churn – Pair programming, project completion bonuses
- A-Skill – (low risk on this project)
- P-Change – Short iterations, simple design
- P-Speed – Short iterations, pair programming
- P-Emerge – Short iterations, dedicated customer
- P-Skill – Risk management team with agile and plan-driven skills

SupplyChain.com
Project Strategy



Operational Stakeholders
Project Manager and Risk Management Team
Agile Feature Teams
Startup
Furnish CRACK team representatives and alternates
Staff and organize to cover all success-critical risk areas
Teambuilding and Shared Vision
• Develop benefits-realization results chains
• Identify missing stakeholders
• Enhance understanding
• Explore goals, options, technology, prototypes, risks
Commit to proceed or backtrack
Systems Definition and Architecting
• Elaborate supply chain operational concept, prototypes, transition strategy
• Evaluate and determine COTS, reuse, and architecture choices
• Develop high-risk components
• Prioritize and sequence desired capabilities, outstanding risks
• Establish project organization, overall process, and feature teams
Commit to proceed or backtrack
Design, Development, and Deployment
• Ensure representative exercise of incremental capabilities
• Monitor progress and new operational development
• Monitor project progress, risk resolution, and new technology developments
• Identify and work critical project-level actions
• Develop and integrate new feature increments
• Communicate proposed redirections
• Prepare for and execute acceptance tests, installation, operational evolution
• Analyze feedback on current version
• Reprioritize features
• Prepare strategy for next increment
• Use risk to determine content of artifacts
Phase 1
Phase 2
Phase 3

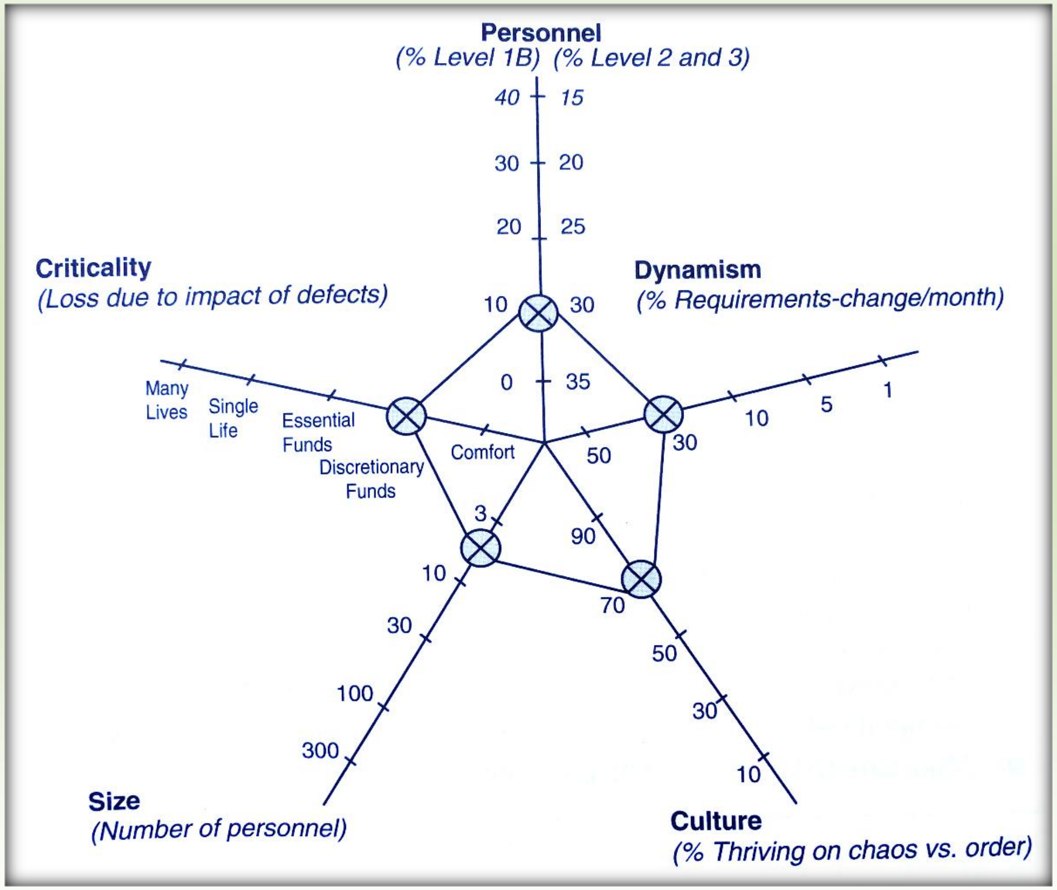
Event Managers Inc.
Application Characteristics

- Event planning software
- For the management of the infrastructure and operations of conferences and conventions
- Includes a single COTS package
- Dev team – 5 people, startup company, one location
- Primary objective is rapid value for Event Managers Inc.

Event Managers Inc.
Five-Step Balancing Method Analysis

- Step 1 – E-Tech, A-Churn, A-Skill, P-Change, P-Speed, P-Emerge
- Step 2 – Plan-driven risks dominate; agile method selected
- Step 3 – n/a
- Step 4 – Simple project strategy created
- Step 5 – Team reflection after each release

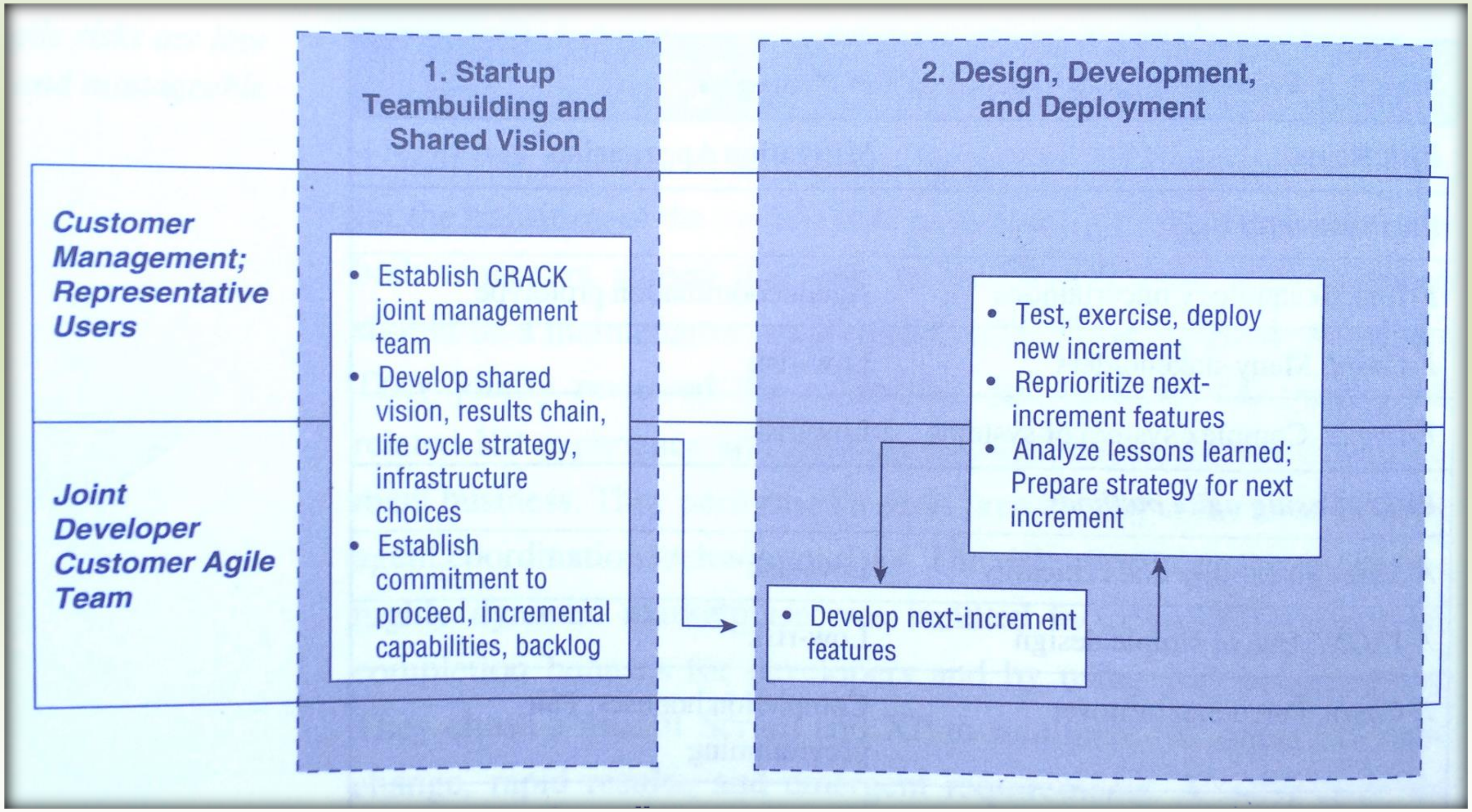
Event Managers Inc.
Polar Chart



Event Managers Inc.
Risk Mitigation

- E-Tech – Prototyping
- E-Coord – (low risk on this project)
- E-Cmplx – (low risk on this project)
- A-Scale – (low risk on this project)
- A-YAGNI – (low risk on this project)
- A-Churn – Pair programming, project completion bonuses
- A-Skill – Hire a maintenance programmer to work for the customer
- P-Change – Short iterations, simple design
- P-Speed – Short iterations, pair programming
- P-Emerge – Short iterations, dedicated customer
- P-Skill – (low risk on this project)

Event Managers Inc.
Project Strategy



1. Startup Teambuilding and Shared Vision
2. Design, Development, and Deployment
Customer Management; Representative Users
Joint Developer Customer Agile Team
• Establish CRACK joint management team
• Develop shared vision, results chain, life cycle strategy, infrastructure choices
• Establish commitment to proceed, incremental capabilities, backlog
• Test, exercise, deploy new increment
• Reprioritize next-increment features
• Analyze lessons learned; Prepare strategy for next increment
• Develop next-increment features

NISCM
Application Characteristics

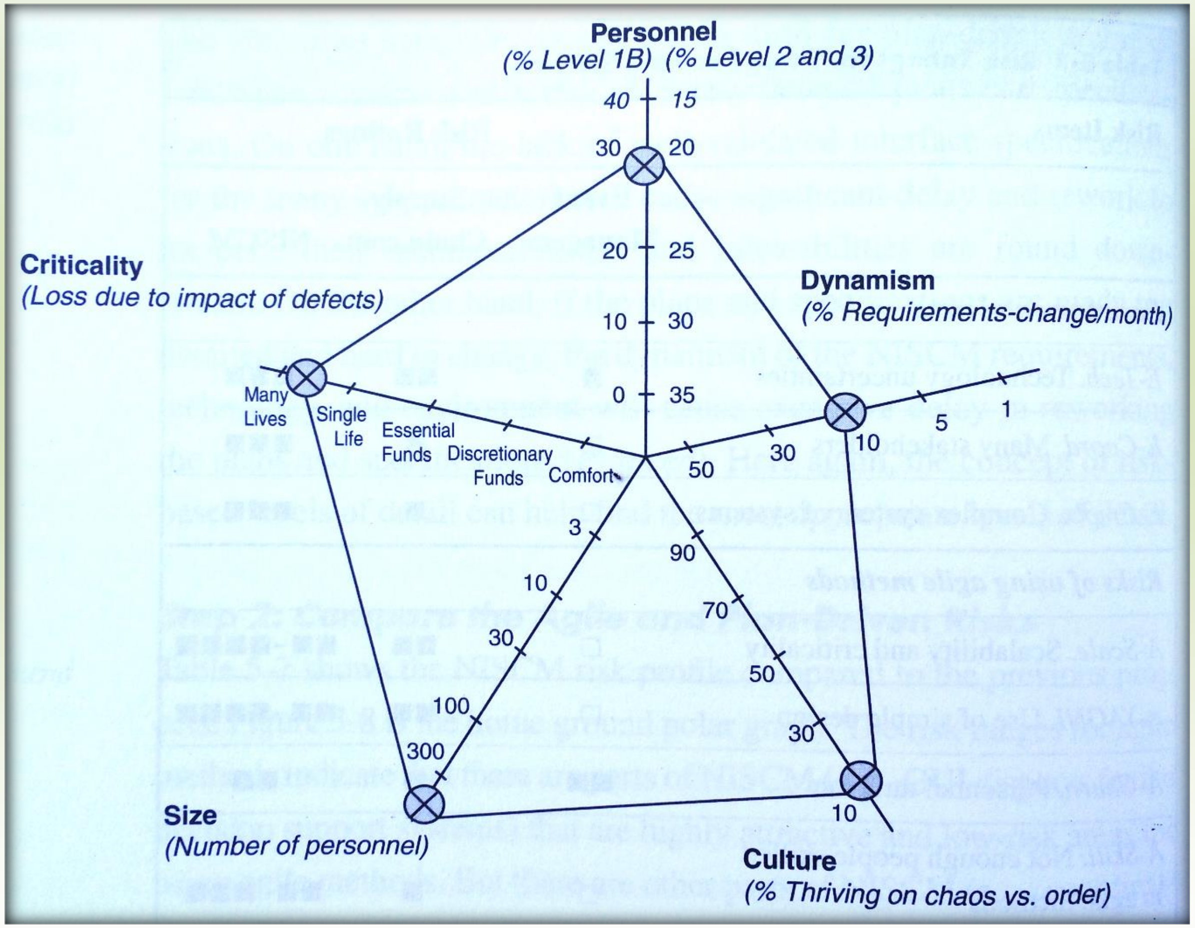
- Crisis management software
- For the coordination of multiple public agencies and private sector resources to manage multiple types of crises; natural disasters, financial, biomedical, weapons, infrastructure, etc.
- Includes many COTS packages; a system of systems
- Dev team – 500 people in multiple locations and multiple organizations
- Primary objective is rapid response, safety, security, scalability, and adaptability

NISCM
Five-Step Balancing Method Analysis

- Step 1 – E-Tech, E-Coord, E-Cmplx, A-Scale, A-YAGNI, A-Churn, A-Skill, P-Change, P-Speed, P-Emerge, P-Skill
- Step 2 – Complex system involves nearly all possible risks
- Step 3 – A hybrid approach is selected with plan-driven as the default, but some encapsulated agile parts (subcontractor development)
- Step 4 – Detailed project strategy created
- Step 5 – Large project and risk management team formed

NISCM

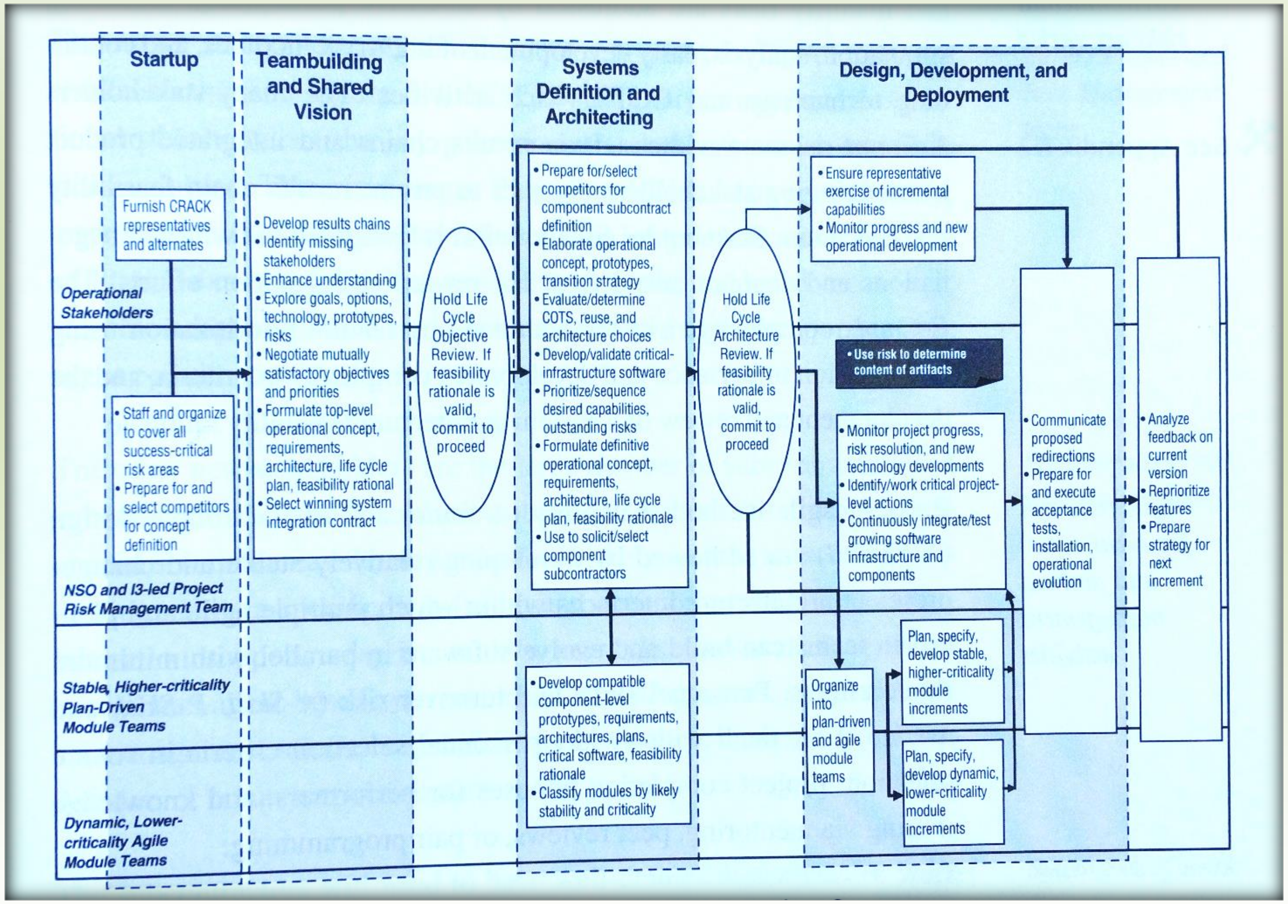
Polar Chart



NISCM
Risk Mitigation

- E-Tech – Prototyping, technology and COTS watch
- E-Coord – Results Chain coordination, award a fee for collaboration
- E-Cmplx – Complexity assessment and avoidance
- A-Scale – Design to allow for rapid parallel development
- A-YAGNI – Use only within agile module teams
- A-Churn – Peer reviews, pair programming, project completion bonuses
- A-Skill – Key personnel selection criteria, project completion bonuses
- P-Change – Encapsulated agile modules, award a fee for agility
- P-Speed – Design to allow for rapid parallel development
- P-Emerge – Evolutionary spiral development, agile module teams
- P-Skill – Key personnel selection criteria, project completion bonuses

NISCM
Project Strategy


Startup
Teambuilding and Shared Vision
Systems Definition and Architecting
Design, Development, and Deployment
Operational Stakeholders
NSO and I3-led Project Risk Management Team
Stable, Higher-criticality Plan-Driven Module Teams
Dynamic, Lower-criticality Agile Module Teams
Furnish CRACK representatives and alternates
• Staff and organize to cover all success-critical risk areas
• Prepare for and select competitors for concept definition
• Develop results chains
• Identify missing stakeholders
• Enhance understanding
• Explore goals, options, technology, prototypes, risks
• Negotiate mutually satisfactory objectives and priorities
• Formulate top-level operational concept, requirements, architecture, life cycle plan, feasibility rational
• Select winning system integration contract
Hold Life Cycle Objective Review. If feasibility rationale is valid, commit to proceed
• Prepare for/select competitors for component subcontract definition
• Elaborate operational concept, prototypes, transition strategy
• Evaluate/determine COTS, reuse, and architecture choices
• Develop/validate critical-infrastructure software
• Prioritize/sequence desired capabilities, outstanding risks
• Formulate definitive operational concept, requirements, architecture, life cycle plan, feasibility rationale
• Use to solicit/select component subcontractors
• Develop compatible component-level prototypes, requirements, architectures, plans, critical software, feasibility rationale
• Classify modules by likely stability and criticality
Hold Life Cycle Architecture Review. If feasibility rationale is valid, commit to proceed
• Ensure representative exercise of incremental capabilities
• Monitor progress and new operational development
• Use risk to determine content of artifacts
• Monitor project progress, risk resolution, and new technology developments
• Identify/work critical project-level actions
• Continuously integrate/test growing software infrastructure and components
Organize into plan-driven and agile module teams
Plan, specify, develop stable, higher-criticality module increments
Plan, specify, develop dynamic, lower-criticality module increments
• Communicate proposed redirections
• Prepare for and execute acceptance tests, installation, operational evolution
• Analyze feedback on current version
• Reprioritize features
• Prepare strategy for next increment

Presentation Outline

Where Are We?

- Meet the Authors
- Discipline, Agility, and Perplexity
- Contrasts and Home Grounds
- A Day in the Life
- Expanding the Home Grounds: Two Case Studies
- Using Risk to Balance Agility and Discipline
- **Conclusions**
- Q & A

Conclusions

Top six conclusions

1. Neither agile nor plan-driven methods provide a silver bullet
2. Agile and plan-driven methods have home grounds where one clearly dominates the other
3. Future trends are toward application developments that need both agility and discipline
4. Some balanced methods are emerging
5. It is better to build your method up than to tailor it down
6. Methods are important, but potential silver bullets are more likely to be found in areas dealing with people, values, communication, and expectations management

No Agile or Plan-Driven Method Silver Bullet
Conclusion 1 of 6

- It will always be a challenge to cope with the complexity of software development
- Agile methods address specific needs well, but do not scale up well to large or safety-critical projects
- Plan-driven methods handle large and safety-critical projects well, but often do not address all project needs

Agile and Plan-Driven Method Home Grounds

Conclusion 2 of 6

- Plotting five critical factors on the polar chart can help determine where a project or organization is relative to agile and plan-driven method home grounds
  - Criticality
  - Personnel skill
  - Requirement dynamism
  - Organizational culture
  - Number of personnel

Future Application Development Will Need Both Agility & Discipline
Conclusion 3 of 6

- Large projects can no longer count on low rates of requirement dynamism
- Agile development is becoming more mainstream, so it will need to be able to handle complexity
- Tailorable methods that can combine the benefits of agility and discipline will be in increasingly high demand

Balanced Agility-Discipline Methods are Emerging
Conclusion 4 of 6

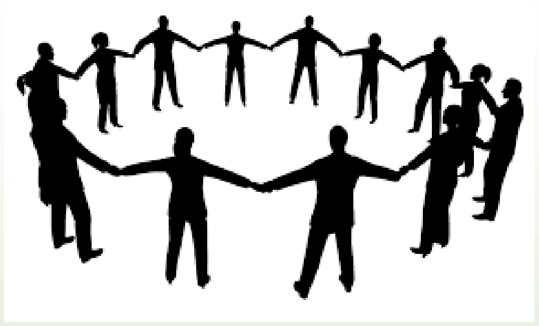
- Newer, lighter versions of the Rational Unified Process
- Built-up agile methods such as Crystal Orange, Dynamic Systems Development Method (DSDM), Feature-Driven Development (FDD), and Lean Development
- Risk-driven balancing method described in this book for tailoring project-specific hybrid development methods

Build Your Method Up – Don't Tailor It Down

Conclusion 5 of 6

- Plan-driven methods can be tailored down, but in practice they often aren't even when they should be
  - Human nature of non-expert management
  - Trying to play it safe
- Agilists have the right idea in starting with a minimal set of practices, and adding more where they can be justified by a cost-benefit analysis of each

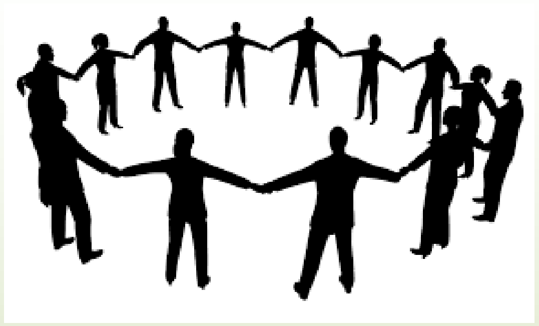
Focus Less on Methods – More on People, Values, Communication, and Expectations Management *Conclusion 6 of 6*



- Agilists are right to value individuals and interactions over processes and tools
  - Multiple studies have shown since the early 1970s that people-related factors dominate other software cost and quality drivers
- Focus less on methods, more on *people*
  - Sofware engineering is
    - Of the people – People develop software for other people's needs
    - By the people – People identify needs; other people develop for those needs
    - For the people – People pay the bills for development; people use the software
  - Must overcome "separation of concerns" legacy

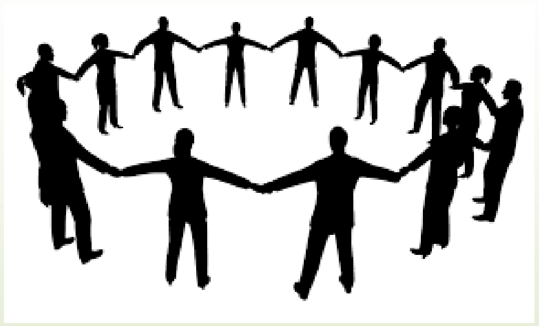
Focus Less on Methods – More on People, Values, Communication, and Expectations Management

Conclusion 6 of 6



- Focus less on methods, more on *values*
  - Reconcile the various value propositions of stakeholders
  - Challenge is all requirements are considered important
  - Agilists are right to focus on pleasing clients/stakeholders
- Focus less on methods, more on *communication*
  - "I'll know it when I see it" (IKIWISI) remains a common problem in the requirements phase
  - Software development is a cooperative game of invention and communication

Focus Less on Methods – More on People, Values, Communication, and Expectations Management *Conclusion 6 of 6*



- Focus less on methods, more on *expectations management*
  - Most software professionals do not naturally do well at expectations management. They have a strong desire to please and avoid confrontation.
  - This unfortunately makes them pushovers for aggressive customers and managers who try to get more software for less time and money
  - Short development iterations address this need well

What Can Be Done Next About Balancing Agility and Discipline? *1 of 2*

- Do a self-assessment to determine what your organization characteristics are, and what your stakeholders want them to be
- How will you cope with future trends?
  - Increased pace of change and therefore the need for agility
  - Increased need for dependability and therefore the need for discipline
  - The need to satisfy stakeholders and outperform competitors
  - Increasing scarcity of high development talent relative to need
  - COTS integration, web technology evolution, distributed and mobile operations, and virtual collaboration

What Can Be Done Next About Balancing Agility and Discipline? *2 of 2*

Five Step Balancing Method – Not “yet another development method,” but rather a way for determining how to best balance agility and discipline in your own local software development environment.

- **Step 1** – Identify and rate risks using polar chart.
- **Step 2a** – If agility risks dominate plan-driven risks, use a risk-based plan-driven development method.
- **Step 2b** – If plan-driven risks dominate agile risks, use a risk-based agile development method.
- **Step 3** – If the risks are mixed, create a project-specific hybrid method.
- **Step 4** – Create an overall project strategy which mitigates each risk.
- **Step 5** – Continually track progress and improve your capabilities.

Presentation Outline

Where Are We?

Q & A

Any Questions?