AEROSPACE
Assuring Space Mission Success

Agile Software Development in Defense Acquisition – A Mission Assurance Perspective

Dr. Peter Hantos
The Aerospace Corporation

NDIA 15th Annual Systems Engineering Conference, San Diego, California
October 22, 2012

© The Aerospace Corporation 2012

Acknowledgements

- This work would not have been possible without the support of the following people of The Aerospace Corporation
  - *Asya Campbell*
  - *Suellen Eslinger*
  - *B. Zane Faught*
  - *William S. Macaulay*
- Special thanks
  - *Steven Kropp, Florida Department of Economic Opportunity, Labor Market Statistics Center*
- Funding Source
  - The Aerospace Corporation's Aerospace Technical Investment Program (ATIP,) Software Acquisition Long Term Capability Development (LTCD) Project

AEROSPACE

Outline

- Background and Motivation
- Objectives
- Agile Software Development – The 64,000-foot View
- Still Flying High – Context and Building Blocks
- Fasten Your Seat-belt and Prepare for Landing
  - *The Life Cycle Perspective of Agile Software Development*
  - *Agile Software Development Values*
  - *eXtreme Programming (XP)*
- The State-of-Affairs - Agile Software Development in the Commercial, Market-Driven World
- Is Agility Really the Answer to Fix the Broken Acquisition System?
- Conclusions
- Acronyms
- References
- Backup

AEROSPACE

Background

- Emergence of new buzzwords in software development
  - *Competitive pressures of the 1990s forced software companies to reexamine their development processes and adopt radical approaches. As a result, the industry has been flooded with buzzwords like "internet time," "extreme," and "agile," just to mention a few*
- Management buzzwords have been flooding over the past 30 years…
  - *There has been a "bandwagon effect" of popular management movements such as total quality management (TQM,) management by objectives, reinventing government, reengineering, the balanced scorecard, lean, and Six Sigma®. However,*
    - companies that claimed excellence on the basis of these practices later turned out to be mediocre or outright failure [Paparone 2009]
  - *Consequently, a relatively recent, interesting recommendation to the Pentagon brass: "Stay away from management bestsellers…" [Erwin 2009]*

** Six Sigma has been registered in the U.S. Patent and Trademark Office by Motorola*

AEROSPACE

Motivation

- History notwithstanding…
  - *Agility seems to be a simple concept*
  - *It is commonly perceived as a virtue*
  - *Agile methods are making inroads into software development*
- Despite of Ms. Erwin's advice, Pentagon brass does not seem to be able to stay away from management bestsellers after all ☺
- Consequently, the idea of bringing agile concepts into defense acquisition requires a closer look

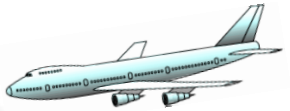
AEROSPACE

Objectives

- Attendees will be able to
  - *Name popular agile software development methods*
  - *Describe representative agile software development practices*
  - *Compare agile and traditional development methods*
  - *Assess the appropriateness of an organization's software development practices*
  - *Appreciate the spirit and usefulness of mission assurance in carrying out the evaluations of the defense contractors' software development practices*
  - *Differentiate between agility in acquisition and agility in development*

AEROSPACE

AEROSPACE
Assuring Space Mission Success



Agile Software Development - The 64,000-foot View

What is Agility?

- The narrow, dictionary definition [Collins 2012]:
  - ***Quick in movement; nimble***
- Agility implies both the capacity and capability to act immediately
  - *Agility is perceived a virtue*
  - *In business, agility is considered an important organizational capability*
- Unfortunately, in most contexts it is ill-defined or inconsistent
  - *Agility does not simply equate with speed, as the following examples show*
    - Agility may conflict with speed
      - *The Titanic's ability to turn sharply is far more likely to avert disaster than increasing its top speed charging straight ahead*
    - Agility requires speed but also requires balance
      - *E.g., in martial arts*
  - *"Lean" does not always equate with "agile"*
    - E.g., applying lean concepts might increase the rigidity of a process
      - *This rigidity results from constraining the process in order to optimize the case "right now"*

Agility is like the Elixir of Life or the Fountain of Youth – Mysterious and Elusive
~ Anonymous

AEROSPACE

Agility in Defense

- The warfighter perspective
  - *There is a confusion about the need for systems enabling **war-fighter agility** vs. the need for **agile acquisition of weapon systems***
    - No argument about the value of war-fighter agility. However,
      - *War-fighter agility can be primarily supported via weapons design and flexible architecture*
      - *Faster access to new weapons is not always the right solution*
      - *The trade-off between faster access and features is promoted, but the underlying, hidden quality concessions are always controversial and the associated decisions are very difficult*
- The acquisition perspective
  - *There are essential concerns that need to be clarified and answered*
    - To what extent would ***agile software development*** contribute to the achievement of ***agile acquisition of weapon systems?***
    - How is ***fast procurement*** different from ***agile acquisition?***
    - Under what circumstances is agile software development acceptable or even desirable for weapon systems acquisition?

For operational responsiveness we need "agile products" and not "agile processes"

AEROSPACE

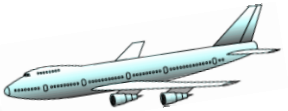
(Our) Definition of Agile Software Development

- Agile software development methods employ practices that are consistent with the Agile Manifesto's value statements and principles
  - *There are numerous, "brand-name" methods that are considered agile**
  - *However, "new" approaches are published almost every day that are mostly mix-and-match medleys of existing practices*
- History of the Agile Manifesto**
  - *Created on February 11-13, 2001 at the first meeting of agile proponents, the 17 founding members of the Agile Alliance*
- Agile values:
  - *"We are uncovering better ways of developing software by doing it and helping others doing it. Through this work we have to come to value:*
    - **Individuals and interactions** over **processes and tools**
    - **Working software** over **comprehensive documentation**
    - **Customer collaboration** over **contract negotiation**
    - **Responding to change** over **following a plan.**"

** See the backup charts; ** For the complete text see [Agile 2001]*

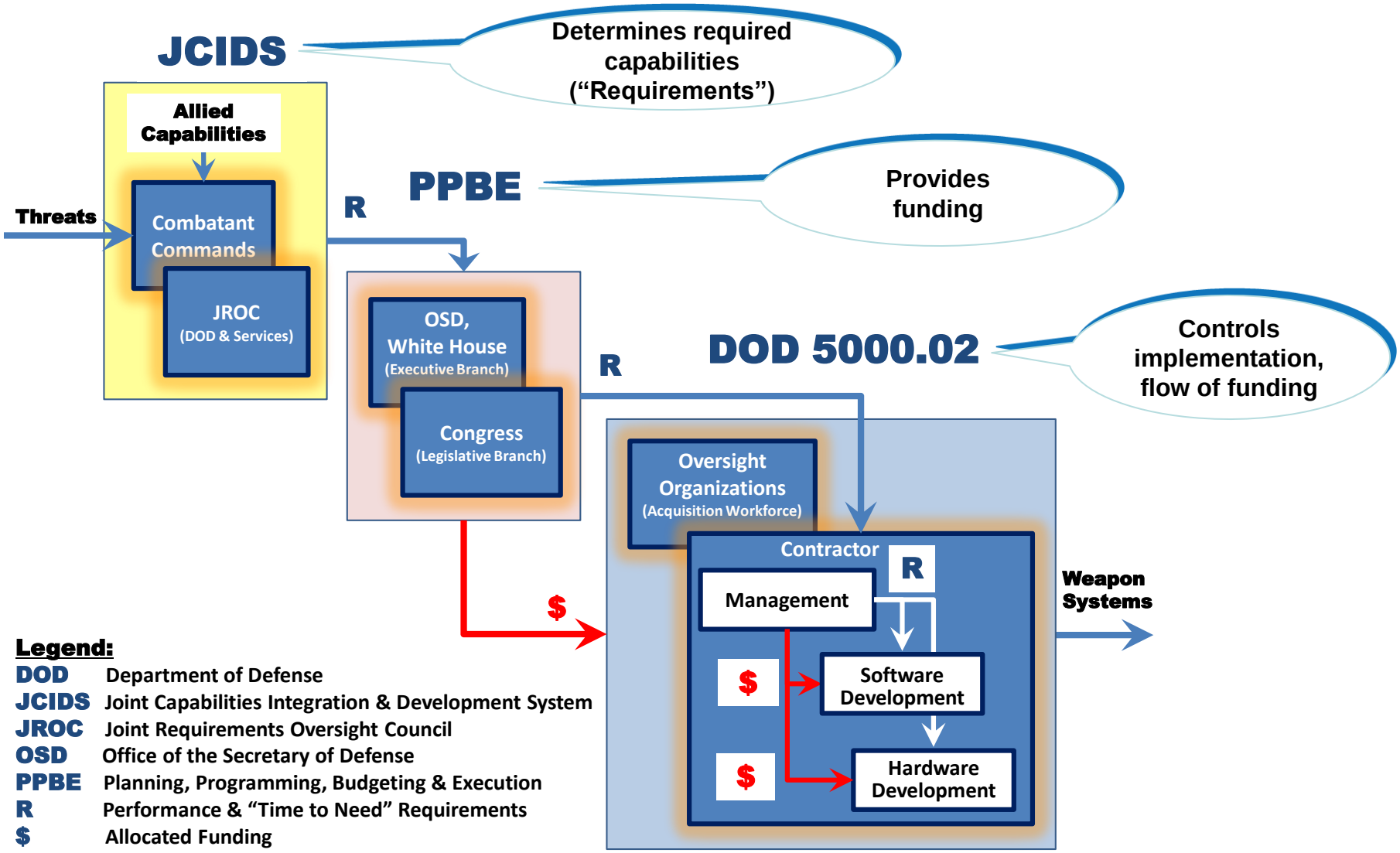
AEROSPACE

AEROSPACE
Assuring Space Mission Success



Still Flying High – Context and Building Blocks

Defense Acquisition (The Big "**A**" Acquisition Process…)

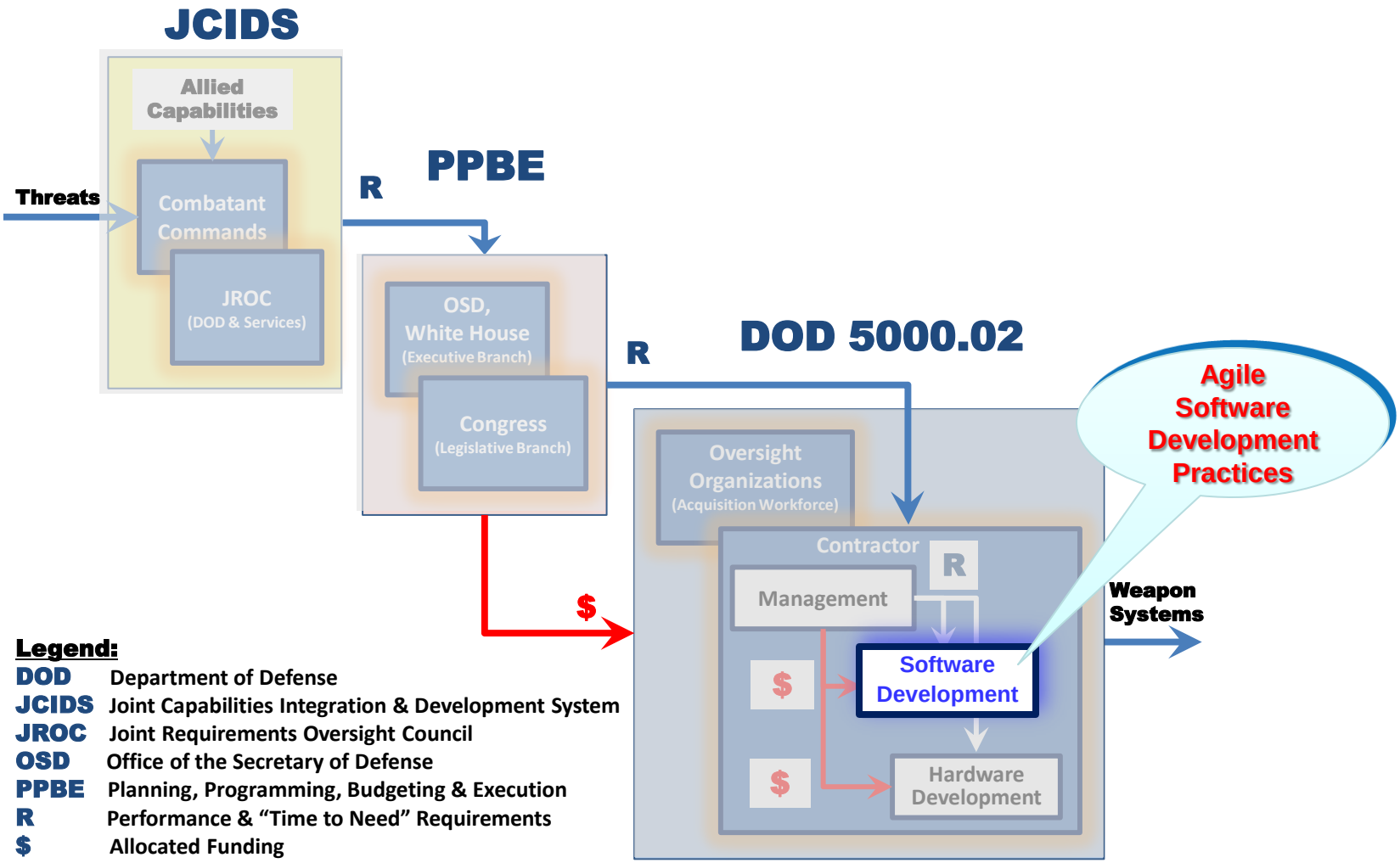


Legend:

DOD	Department of Defense
JCIDS	Joint Capabilities Integration & Development System
JROC	Joint Requirements Oversight Council
OSD	Office of the Secretary of Defense
PPBE	Planning, Programming, Budgeting & Execution
R	Performance & "Time to Need" Requirements
$	Allocated Funding

AEROSPACE

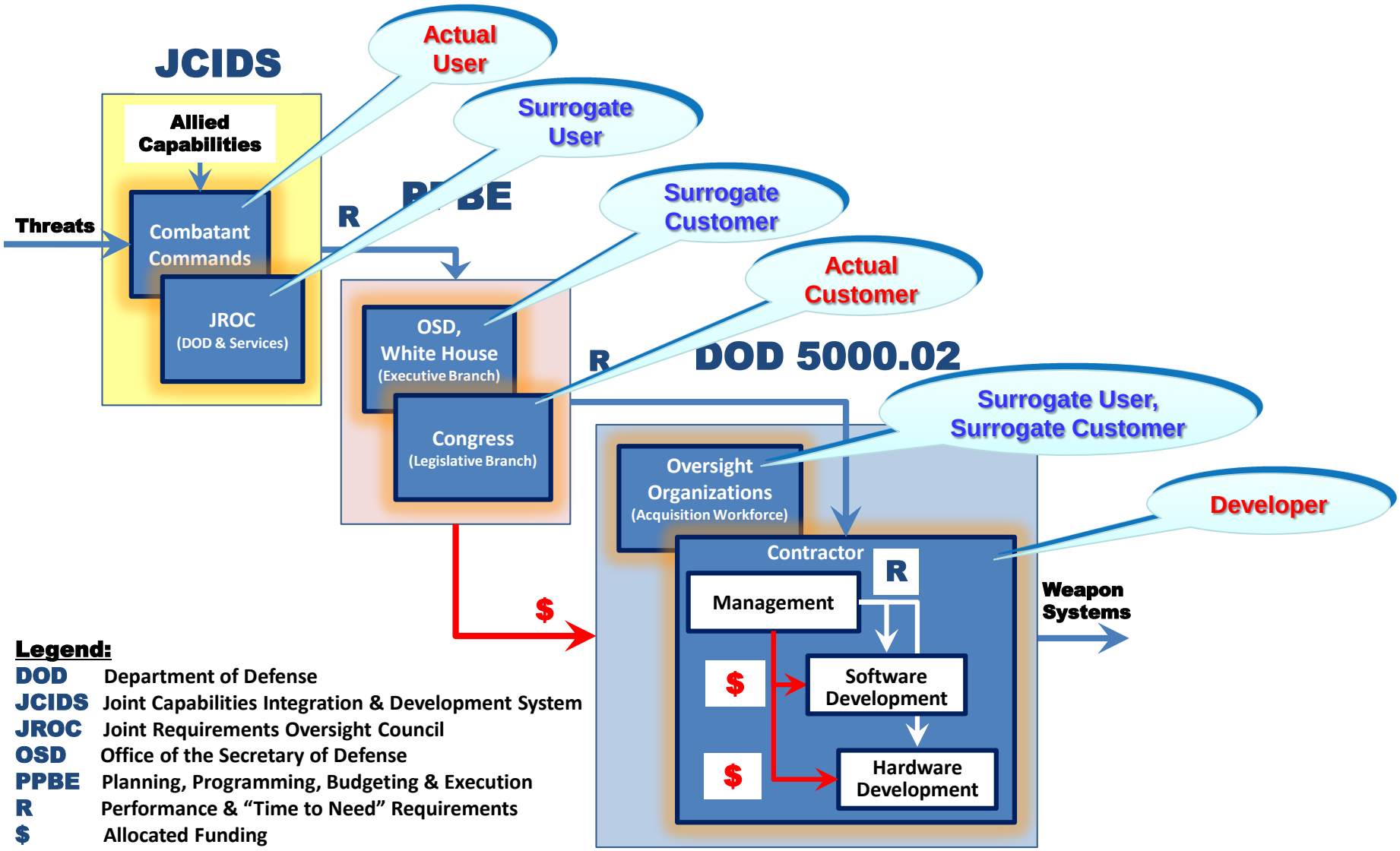
Agile Software Development in Defense Acquisition



Agile software development affects only a small fragment of the acquisition system

AEROSPACE

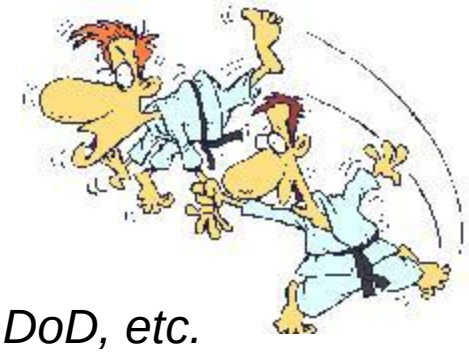
Key Stakeholders in the Big "**A**" Acquisition Process



Note how removed development is from the actual user and customer

AEROSPACE

Acquisition is a Contact Sport…



- Due to different motivation and behavior, there is a tension between
  - *Stakeholders of the Acquisition System, e.g., Congress, DoD, etc.*
  - *Stakeholders of the Oversight Organizations, e.g., Acquisition Program Offices (APOs*) and the Development Organizations (Contractors)*
  - *Stakeholders of the Development Organizations themselves*
    - Management vs. developers
    - Hardware developers vs. software developers
- Some hard facts to face
  - *Typically the conflicts are not between equals*
  - *Different stakeholders have different political weight and capabilities, hence in most cases "win-win" solutions are either not feasible or not pursued*
- New valuation considerations for agile software development practices
  - ***Potential impact on existing tensions** in the overall acquisition system*
  - ***Loyalty factor**, i.e., whose interest should be acknowledged as the most important in a particular context*

The fundamental source of tension is which stakeholder will bear the risk

** APO is a generic term; program offices are called differently in different services*

AEROSPACE

The Risk Pendulum – Who is Going to Bear the Risks?

Basic Funding Patterns*	Cost-based	Time-based	Fixed Price
Promise	Best effort	Best effort	Shall deliver
Cash flow	As incurred	As incurred	On delivery of item
Customer control	Maximal	Maximal	Minimal
Risk to contractor or developer	Low	Low	**High**
Risk to customer or management	**High**	**High**	Low

** Note that these patterns have their formal, contracting equivalents and variations in the Federal Acquisition Regulation (FAR)*

The Risk Pendulum

Customer or management

Contractor or developer

The interesting paradox is that despite higher customer control - which, perceived to drive down risk - cost-based and time-based patterns are still risky…

AEROSPACE

Mission Assurance Definitions*

- **Mission Success**
  - *The achievement by an acquired system (or system of systems) to singularly or in combination meet not only specified performance requirements but also expectations of users and operators in terms of safety, operability, suitability, and supportability*
  - *Mission success is evaluated after operational turnover, according to program-specific timelines and criteria*
- **Mission Assurance**
  - *The disciplined application of general systems engineering, quality, and management principles towards the goal of achieving Mission Success, and towards this goal, this disciplined application provides confidence in its achievement*

** Source: [Guarro 2007]*

AEROSPACE

Mission Assurance is Development Process-neutral

- Software Mission Assurance does not assume any particular software development methodology, programming language, or tools
- Mission Assurance is the responsibility of the APO, a defense acquisition oversight organization
  - *Air Force APO's enjoy direct help from multiple entities, such as*
    - Federally Founded Research and Development Centers (FFRDCs)
    - Systems Engineering and Technical Assistance (SETA) contractors
    - Systems Engineering & Integration (SE&I) contractors
- The APO's Mission Assurance activities do not assume the presence of any similar, or similarly named (i.e., "Mission Assurance") effort from the contractor
  - *If such effort exists then, from the APO's perspective, it needs to be treated as integral part of the contractor's development process*

Software Mission Assurance tasks are inherently essential for the assurance of <u>any</u> software development endeavor in defense acquisition

AEROSPACE

The Main Exposure to Mission Success: Software Defects*

- Definition of a software defect
  - *Any software attribute or characteristic that represents a deviation from specified attributes or characteristics*
  - *Software defects can cause unanticipated cost and schedule overruns and in operational systems performance deficiencies*
- Definition of a software fault
  - *A software fault is a software defect that can result in a significant system function failure during the execution of the code*
- Hardware-induced vs. software-induced failures
  - *Hardware-induced failures*
    - Software always depends on hardware; certain hardware defects might manifest themselves as software defects (e.g., a Single Event Upset (SEU) in the on-board computer's memory or registers as a result of naturally occurring cosmic rays, trapped protons and solar energetic particles)
  - *Software-induced failures*
    - Majority of such failures are rooted in software design or specification flaws; Essentially the system enters into an unanticipated and/or poorly understood operational regime

** Definitions courtesy of Myron Hecht [Guarro 2008]*

AEROSPACE

Software-induced Failure* Types

- Deterministic vs. random failures
  - *Deterministic ("Bohrbugs")*
    - Repeatable
    - Traceable to root cause(s) under control of developer or user
      - *Deterministic failures <u>can be</u> prevented through the use of a disciplined development process*
  - *Random ("Heisenbugs")*
    - Not repeatable; many such failures can be fixed by reset
    - Caused by transient states of the software (timing, buffer overflows, queues, memory leaks, etc.)
    - Indistinguishable from single event upsets (SEUs,) power fluctuations or hardware timing errors
- Recoverable vs. non-recoverable software failures (space example)
  - *Recoverable software failures are events that occur in spacecraft processors that cause a loss or performance degradation of the bus or payload, which can be restored via either <u>onboard or ground corrective</u> actions*

Application of a disciplined development process itself is <u>not</u> a guarantee for preventing random failures or mitigating recoverable failures

** For sake of simplicity they will be referenced as **software failures***

AEROSPACE

Preventing Random Software Failures

- The following approach is recommended*
  - ***Collect software failure data during integration testing***
    - Use relevant operational profiles, not just requirements, to define test plans
    - Record software operating time
    - Record all failure events
    - Collect recovery time and data to determine the probability of recovery
  - ***Select an appropriate software reliability model***
    - This model will be used to extrapolate behavior from test data
  - ***Evaluate parameters***
    - Software behavior must be analyzed and validated via formal, systematic means that take into account a variety of nominal and off-nominal operational scenarios
  - ***Integrate findings into a system stochastic or reliability model***

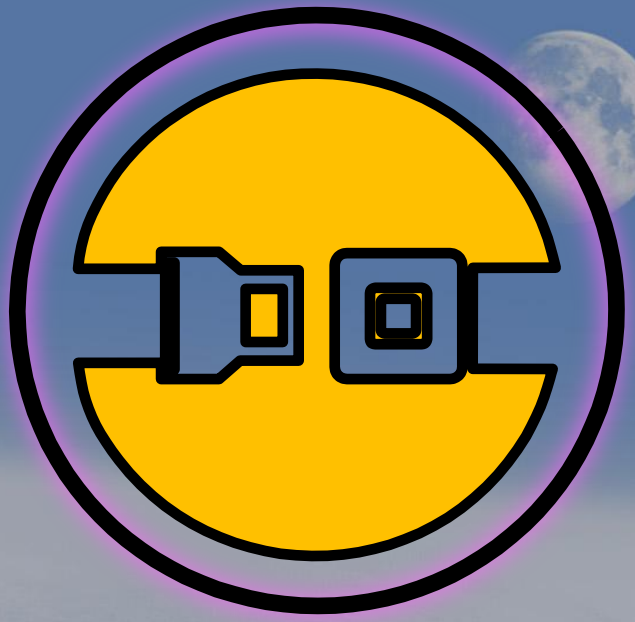
Most likely the contractors use similar approaches; verifying the correctness of the contractors' analyses is a critical mission assurance task

** Source: [Guarro 2008]*

AEROSPACE

AEROSPACE
Assuring Space Mission Success

Well, the high-altitude cruising is over…



Fasten your seat-belt and prepare for landing!

The Life Cycle Perspective of Agile Software Development

AEROSPACE

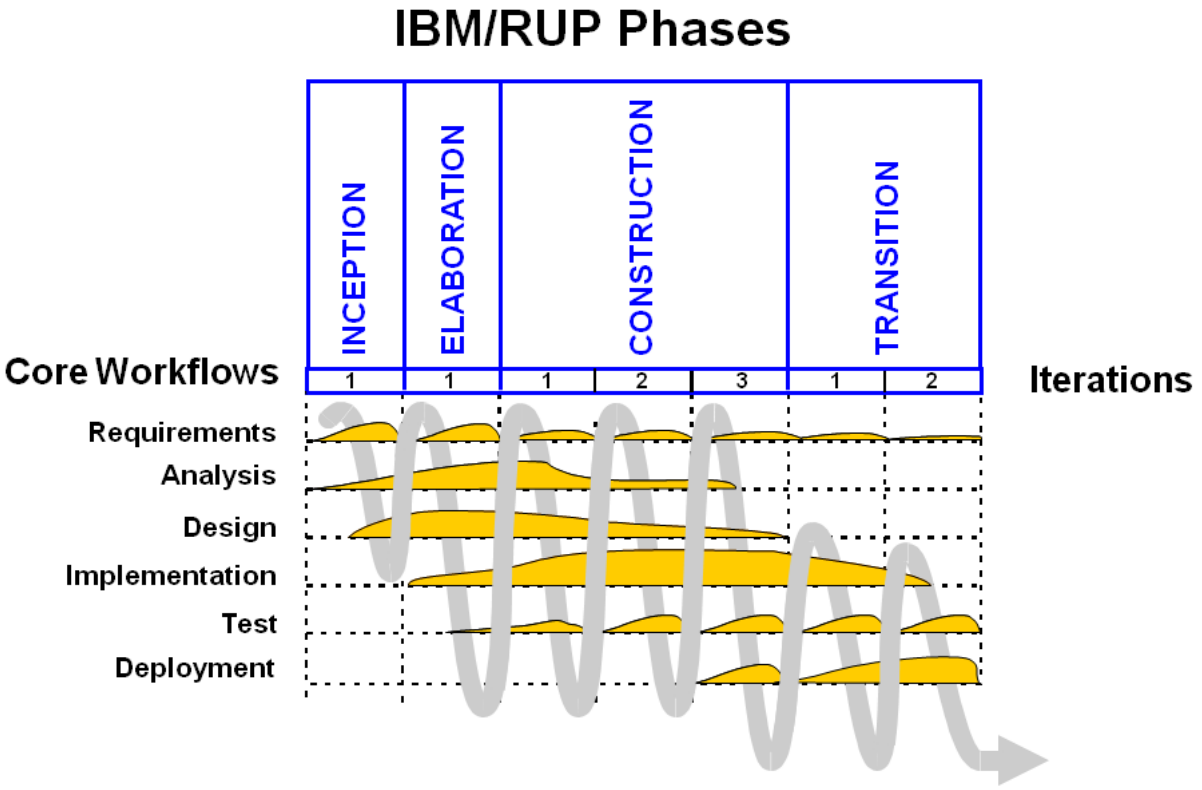
Agile Process Example: Scrum*

- Scrum is a lean approach to software development
  - *Simple "inspect and adapt" management framework, using time-boxing*
  - *Based on the scrum metaphor for new product development [Takeuchi 1986]*
  - *No declared, method-specific development practices*
  - *"Backlog" is a metaphor for requirements*

** The process was first formalized by Ken Schwaber [Schwaber 95]*

AEROSPACE

In Contrast, an Iterative-Incremental Process, IBM/RUP



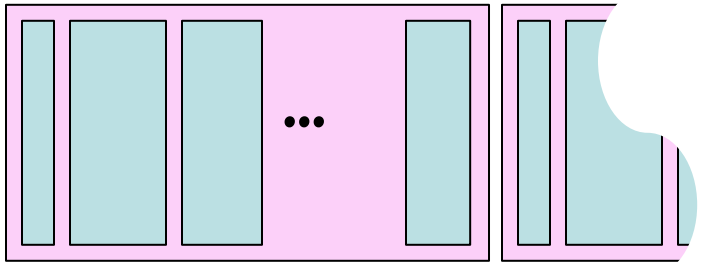
- The Rational Unified Process (RUP) is a comprehensive process model*
  - *Workflows are essentially life cycle processes with detailed description*
  - *The process encompasses modern development principles [Royce 1998]*

** [Jacobson 1999]; Renamed to IBM/RUP after the acquisition of Rational Corp. by IBM*

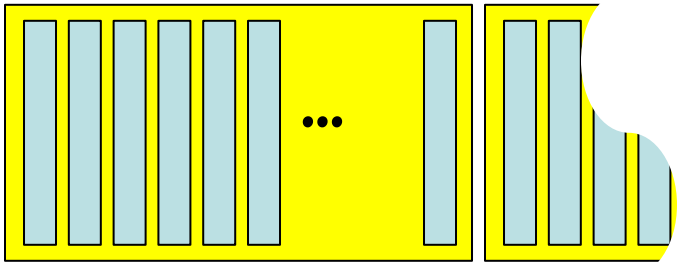
AEROSPACE

After We Remove the Fluff (i.e., the Metaphors…)

Iterative-Incremental Development (IID)
Content (Requirements) Driven



Time-box (“Agile”)
Calendar (“Clock”) Driven



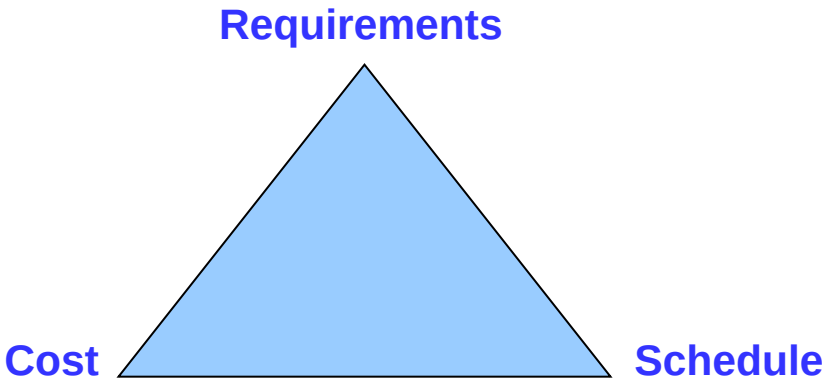
Factors to be compared	IID	Time-box
Iteration/Increment duration	varying	set
Iteration content in the context of an increment	Planned up-front	Not planned up-front
Difficulty of iteration planning	moderate	easy
Difficulty of increment planning	**difficult**	**difficult**
Micro-estimation fidelity	moderate	higher than IID
Macro-estimation fidelity	high	**low**
Naturally fitting contracting pattern	cost-based	**time-based**

Red flag marks the customers’ primary concerns

AEROSPACE

Main Time-box Risk: Violating the Iron Triangle Principle

- What is the Iron Triangle Principle?
  - *One can only fix two of the **cost, requirements, and schedule** triad; any attempt to pre-determine all three results in an non-executable plan*

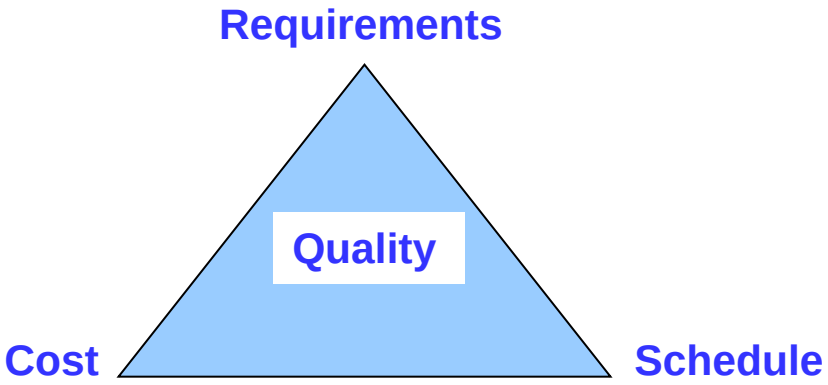


- (Unfortunately) typical scenario: the number of time-boxes, like sprints in Scrum, and ultimately the launch-date are pre-determined
  - *At that point cost (i.e., manpower loading,) is also pre-determined*
- Risks:
  - *No guarantees that all desired requirements can be fully implemented; In fact, the adaptive process will successively defer and drop requirements*

AEROSPACE

Main Time-box Risk: Violating the Iron Triangle Principle

- What is the Iron Triangle Principle?
  - *One can only fix two of the **cost, requirements, and schedule** triad; any attempt to pre-determine all three results in an non-executable plan*



- (Unfortunately) typical scenario: the number of time-boxes, like sprints in Scrum, and ultimately the launch-date are pre-determined
  - *At that point cost (i.e., manpower loading,) is also pre-determined*
- Risks:
  - *No guarantees that all desired requirements can be fully implemented; In fact, the adaptive process will successively defer and drop requirements*
  - *Since the process is already over-constrained, delivering predictable quality is also a challenge*

AEROSPACE

Agile Software Development Values

AEROSPACE

Examining Agile Software Development Values

- Agile software development values revisited
  - ***Individuals and interactions*** *over* ***processes and tools***
  - ***Working software*** *over* ***comprehensive documentation***
  - ***Customer collaboration*** *over* ***contract negotiation***
  - ***Responding to change*** *over* ***following a plan***

- During the analysis the following, typical figures were considered
  - *Space vehicle (embedded, large, including bus software and payload(s)):*
    - ~512 Thousand Delivered Source Instructions (KDSI)
  - *Ground systems:*
    - Space Shuttle software ~2,000 KDSI
    - Satellite control systems software ~4,700 KDSI
  - *The development of* ***512 KDSI*** *would require roughly a* ***6,420 person-month*** *effort, spreading over* ***41 months****, involving* ***~157 full-time equivalent software personnel***

AEROSPACE

Individuals and Interactions Over Processes and Tools

- Let's focus on processes first
  - *Agile proponents believe that one should only declare and rely on **practices** instead of **processes** to increase the agility of software development*
    - A practice usually refers to an individual activity while a process is an aggregate structure of multiple activities
  - *Relying only on practices certainly ensures a greater level of flexibility, however*
    - This flexibility comes with unavoidable ambiguities and may create tension amongst the stakeholders
      - *Consider the example's 157 developers working shoulder-to-shoulder*
      - *Consider the problems of concurrent hardware-software development*
  - *In pursuing mission success we found that even the use of so-called mature processes, such as defined by the CMMI®, proved to be inadequate*

The government must make a robust software standard contractually compliant [Eslinger 2006]

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University

AEROSPACE

Lean

- The term “lean production” was coined in the 80’s [Krafcik 1988]
  - *The underlying ideas represent the so-called lean thinking about processes*
- Current (mis)use of the term
  - *Lean is a popular buzz-word for general cost cutting efforts*
  - *Lean may be used in conjunction with Six Sigma®, another, also manufacturing-rooted, process improvement method (“Lean Six Sigma”)*
    - Unfortunately, this term is misleading: “lean” does not mean applying lean thinking to Six Sigma but using Six Sigma tools to carry out lean practices
- Key principles of lean systems thinking [Rule 2011]
  - *Understand value from the stakeholders’ perspective*
  - *Identify all steps in the value stream*
  - *Enable value to flow smoothly*
  - *Respond to the pull of stakeholder demand*
  - *Continuously seek perfection*
- Mission assurance exposure
  - *Difficult to sort out what is really important due to stakeholder conflicts*
  - *Lean Six Sigma rule of thumb is that usually only 5% of total process cycle time adds value to outputs; mission assurance is valued low by developers*

® Six Sigma is registered in the U.S. Patent and Trademark Office by Motorola

AEROSPACE

Major Areas in a Typical Software Development Standard*

System and Software Architecture
Human Systems Integration
Interoperability and Standardization
Reliability, Safety, Information Assurance
Project Planning and Oversight
SW Development Environment
System Requirements Analysis
SW Requirements Analysis
SW Design
SW Implementation and Unit Testing
Unit Integration and Testing
SW Qualification Testing

Transition to Operations and Maintenance
Software Configuration Management
SW Peer Review/Product Evaluation
SW Quality Assurance
Corrective Action
Joint Technical and Management Reviews
Risk Management
SW Management Indicators (Metrics)
Security and Privacy
Subcontractor Management
Interface with Software IV&V Agents

The "lean" question: Which ones do not add value? Which ones to get rid off?

* *Source: [Adams 2005]*

AEROSPACE

What Does My Dentist Know About Mission Assurance?

Sign in my dentist's office:
"Brush only those teeth you wish to keep…"

AEROSPACE

Individuals and Interactions Over Processes and Tools-2

- Tools
  - *The typical 3-4 year long development and a minimum 5-10 year long operation and sustainment for a space vehicle require strong tools support*
    - Development must be based on an architecture-first approach
      - *Architecture modeling artifacts need to be documented with rigorous notation and handled with appropriate (preferably visual) modeling tools*
      - *The dynamics of concurrent workflows by different teams working on shared artifacts necessitates a rigorously controlled change management environment*
    - Tools are also necessary to keep all the engineering information in different formats synchronized and to support bi-directional traceability
      - *System requirements, software specifications, design models, source code, executable code, scripts, test cases, test data, etc.*
    - **True change freedom** cannot be realistically achieved without the support of an appropriate, integrated environment [Royce 1998]

Even in a stable labor force tacit knowledge sharing is not sufficient

AEROSPACE

Work Force Volatility

- The work force in the information sector is very volatile* even during recessions when the overall, net employment change is lower than average

Periods of Recession**	Information Sector		Federal Sector	
	Hires	Separations	Hires	Separations
2001-2002	36.5%	43.3%	19.75%	19.4%
2008-2010	23.7%	27.8%	22.13%	21.0%

- How to interpret the data
  - *Unfortunately, the Bureau of Labor and Statistics (BLS) is not collecting the exact data we would be interested, i.e., programming-related turnover in the defense industry*
  - *However, one can see that the turnover rate is quite high even in the federal sector, which is considered less volatile than the private sectors*
  - *Additionally, the BLS database does not track internal, company turnover*

Insisting on tacit knowledge sharing is inappropriate in case of such a volatile work force

** Source: Bureau of Labor and Statistics database; ** [Bruyere 2011]*

AEROSPACE

Working Software Over Comprehensive Documentation

- Agile proponents essentially do not dispute that documentation plays an important role in software development [Ambler 2011]
  - *Author makes a point from an agile perspective that customers must understand the total cost of ownership (TCO) for a document, and they must explicitly decide to invest in that document*
  - *This a good advice under any circumstances, of course*
- **However, this value statement is about interim progress assessment**
  - *The idea is not new; modern processes are already using the demonstration-based approach to assess intermediate artifacts [Royce 1998]*
- The concern regarding the agile approach is the impact on the customer
  - *Principle #8 of the Agile Manifesto represents a strong imposition on the customer: "Sponsors, developers, and users maintain a constant pace". Unfortunately, maintaining such pace is not feasible on large projects*
  - *Issues:*
    - Embedding users/customers with the necessary expertise into every team
    - Users/customers need to approve technical decisions in the short cycles
    - Coordination of an extensive network of user/customer representatives

In short, this agile value does not scale up in a large project

AEROSPACE

Customer Collaboration Over Contract Negotiation

- As it was shown, actual users and customers are far removed from the development organization
  - *JROC, DOD, and Congress are high-inertia organizations with complex, bureaucratic processes for interaction*
  - *These are stakeholders with different political weights; building true collaborative relationships is difficult if not impossible*
- With the current, rigid “upstream” relationship the flexibility of the surrogate customer is very limited
  - *Agile development will not improve the agility of the acquisition process; in fact, insisting on developer agility may exacerbate the existing tensions*
- It is an unfortunate fact of life that when things do not go well, collaborative resolution becomes less and less feasible
  - *The stakeholders have their own, different risk perspectives and motivations and their differences cannot be easily reconciled via voluntary actions*
- You would not remodel your kitchen without a detailed contract, so why would you deemphasize the importance of contracts for billion-dollar weapon system acquisitions?
  - ***Well, actually we did it in the 1990s, it was called “Acquisition Reform”***

AEROSPACE

Responding to Change Over Following a Plan

- The essential motivation is the recognition that solution details to complex problems cannot be successfully determined up-front
  - *This is not a new idea; that's why modern, but pre-agile software development methods are adaptive and use iterative/incremental processes. How requirements risks are handled in modern methods:*
    - On micro-level: The emphasis during the planning of iterations is on facilitating a successively refined understanding of requirements
    - On macro-level: New or changing requirements are expected to be handled via **evolutionary acquisition and development** strategies
- Agile principle #2 ("**Welcoming changing requirements**") is directly flowing from the discussed agile value statement
  - *Unfortunately, this is a disingenuous statement, to say the least*
    - In reality, everybody likes to work on stable grounds with clear, unchanging expectations; Don't you?
- However, if anyone still has doubts, listen to Yogi Berra:

"If you don't know where you are going, you will wind up somewhere else"

AEROSPACE

Beyond Unavoidable Requirements Volatility

- Even though Yogi Berra was right, a certain level of requirements volatility is unavoidable
  - *Consequently, whatever process is used, some level of flexibility is needed to deal with such volatility*
- However, lack of control may still lead to the erosion of process discipline
  - *"Just because you have a detailed requirements specification that has been reviewed and signed off, that doesn't mean that the development team will read it, or if they do, that they will understand it, or if they do, that they will choose to work to the specification." ~~~ Scott W. Ambler [Ambler 2007]*

Only diligent mission assurance can prevent this from happening

AEROSPACE

eXtreme Programming

AEROSPACE

eXtreme Programming (XP)*

- What is eXtreme Programming?
  - *XP is a light-weight, low-ceremony software development methodology*
    - Based on Kent Beck's early experiences at Daimler Chrysler Corporation
- Why is it Extreme?
  - *Does not involve bungee cords; no relationship to Windows XP either…* ☺
  - *XP adopts well-known software development practices and attempts to take them to their logical extremes*
    - Example: The "You Aren't Gonna Need It" (**YAGNI**) Concept
      - *YAGNI is a general refrain when someone suggests building functionality for the system that is not present in the current requirements set. The assumption is that it can be added later if it becomes necessary*
      - *YAGNI supposed to be the opposite of "Big Design Up-Front"* ***BDUF***
      - *However, remember the importance of diligent, strategic architecting and design we described earlier to prevent random software failures*

BDUF might have its problems, but from a mission assurance perspective we need at least a balanced approach; "extreme" is not really desirable

** Source: [Beck 2000]*

AEROSPACE

XP Practices (Practice Details in the Backup Section)

Original Practices in 2000	Mission Assurance Exposure
Planning game, on-site customer	Burdensome customer participation
Small releases	Implied customer responsibility for validation
Metaphor	None
Simple design	Rigid use of the YAGNI principle
Continuous integration & testing	Implied customer responsibility for success
Refactoring	Practice is not a replacement for proper architecting
Pair Programming	Practice is not a replacement for formal inspections
Collective code ownership	Practice does not scale up
40-hour work week	None
Coding standards	None

Practice Changes in 2004*	# of Practices
No change	**2**
Eliminated or deemphasized	**4**
Renamed or changed	**2**
New practice introduced	**11**

** [Beck 2004]*

AEROSPACE

Mission Assurance Consequences

- A reviewer's opinion about the 2nd edition of Beck's book:
  - *"… the 2nd edition describes a new process that is different from the process Beck describes in the first book. It seems, he has invented a new process (based on his experience with XP) and gave it the same name"**
- The importance of process documentation and use of standards
  - ***XP is a good example of how fluid the agile field still is and how difficult it is to pin-down specific practices***
  - *High-quality, detailed process documentation is needed to mitigate up-front agile process ambiguities; carrying out the oversight function is very difficult without documented, agreed upon terminology and processes*
  - *The customer must understand that (s)he will only get what (s)he explicitly asks for; after the contract is signed, the customer is at the mercy of the contractors and will be separately charged for every request that is deemed to be "new"*
  - *For a more detailed analysis see the earlier mentioned report [Eslinger 2006]*

Use of standards is one of the most effective tools for the customer to go on record with process-related expectations

* *Source: [Stansell 2004]*

AEROSPACE

The State-of-Affairs - Agile Software Development in the Commercial, Market-Driven World

AEROSPACE
Assuring Space Mission Success

Top 12 Reasons Named in 2010 for Adopting Agile*

- Accelerate time to market
- Enhance ability to manage changing priorities
- Increase productivity
- Enhance software quality
- Improve alignment between information technology (IT) and business objectives
- Improve project visibility
- Reduce risk
- Simplify development process
- Enhance software maintainability and extensibility
- Improved team morale
- Reduce cost
- Improve and increase engineering discipline

Some of these expectations are clearly counter-intuitive, showing a lack of true understanding of these methodologies

* *[VersionOne 2010]*

AEROSPACE

Top 12 Concerns in 2010 About Adopting Agile*

- Loss of management control
- Lack of upfront planning
- Management opposed to change
- Lack of documentation
- Lack of predictability
- Lack of engineering discipline
- Development team opposed to change
- [Lack of] engineering talent
- Inability to scale
- Regulatory compliance
- Reduced software quality
- Other

It is not on the list, but one of the main concerns should be lack of consistent metrics and reliable data to verify if any of the objectives stated on the previous slide have been met

* *[VersionOne 2010]*

AEROSPACE

Agile Software Development from a Commercial Perspective

- Using agile software development is a business strategy, based on a particular value proposition
- Choosing a development method should be based on the home grounds of the organization and project, characterized by the following factors*
  - *Project size (Expressed by the number of development personnel involved)*
  - *Criticality (Loss due to impact of defects)*
  - *Level of software understanding in development personnel*
  - *Dynamism (%Requirements-change/month)*
  - *Culture (Thriving on chaos vs. preferring order)*
- When these factors are considered, agile software development certainly seems to be a promising approach for **small, low criticality projects with rapidly changing requirements, where the organization's culture embraces high degrees of freedom, and the developers are highly experienced**

Unfortunately, the applicability of these methods outside of the above described home ground has not yet been proven

* *[Boehm 2004]*

AEROSPACE

The Agile Manifesto Principles – the “New” Way of Developing Software

(1) Early and continuous delivery to satisfy customers
(2) Welcoming changing requirements
(3) Delivering working software frequently
(4) Close collaboration with business people
(5) Motivation of developers through trust
(6) Using face-to-face conversations to convey information
(7) Working software is the primary measure of progress
(8) Sponsors, developers, and users maintain a constant pace
(9) Continuous attention to good design
(10) Simplicity, maximizing the amount of work not done
(11) The best work is always expected from self-organizing teams
(12) Team reflection and behavior adjustment at regular intervals

AEROSPACE

The “Old”(?) Way - Principles of Software Management*

(1) Architecture-first approach
(2) **Iterative life-cycle process**
(3) Component-based development
(4) Establish a change management environment
(5) Enhance change freedom through tools that support round-trip engineering
(6) Rigorous, model-based notation
(7) Objective quality control
(8) **Demonstration-based approach to assess intermediate artifacts**
(9) **Intermediate releases with evolving levels of detail**

Do #2, #8, and #9 ring any bells?

** Source: [Royce 1998]*

AEROSPACE

There is No Silver Bullet – Particularly in Software Engineering! ☺

Everybody quotes Fred Brooks* but still, we keep trying to make one…

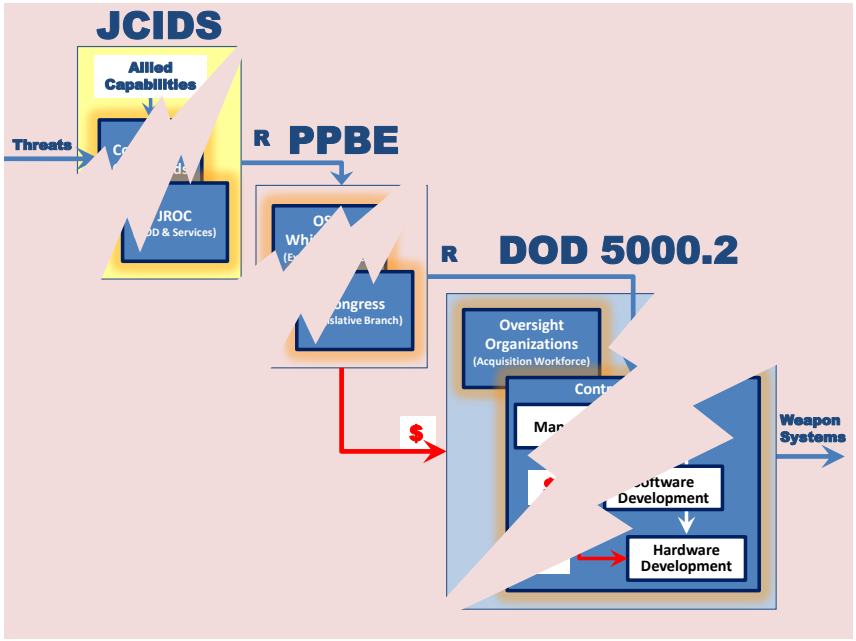
* *[Brooks 1987]*

AEROSPACE

Is Agility Really the Answer to Fix the Broken Acquisition System?

AEROSPACE
Assuring Space Mission Success

How do We Know that it is Broken?



- Defense Acquisition Performance Assessment (DAPA) summary in 2006
  - *"As early as 1971 it has been identified that [defense] acquisition processes have significant shortcomings leading to loss of confidence by congress and the defense community"*
  - *"Many improvements to the DOD's acquisition system have been made as a result of past reviews …* ***However, the ability to deliver operational performance of major systems within predicted cost and schedule has not improved over the last 20 years****"*

AEROSPACE

Selected* DAPA Recommendations in 2006

- Replace the Joint Capability Integration Development System (JCIDS) with a new, **two-year** recurring planning process based on the **15-year extended plans** submitted by combatant commands
- **Stabilize** the Planning, Programming, Budgeting, and Execution (PPBE) process
- Introduce a new requirements process with **2-year duration**
- Establish a distinct, **stable** Program Funding Account
- Increase program **predictability**
- Program all accounts to a **high, 80/20 confidence level**
- **Establish very early** a realistic capability delivery rate
- **Establish very early** all test plans
  - *Complete Test & Evaluation Management Plan (TEMP) and Initial Operational Testing & Evaluation Plan (IOT&EP) prior Milestone B*

Clearly, the DAPA panel valued stability and predictability as opposed to agility

** There were more recommendations but those did not have potential agile implications*

AEROSPACE

Acquisition Problems Identified in 2011 by the Government Accountability Office (GAO)*

- **Alternatives not considered**
  - *Clearly, no relationship to agile development*
- **Funding unstable**
  - *Agile development might be helpful in dealing with unstable funding, but only at the price of delaying or dropping requirements*
- **Inadequate contracting strategy**
  - *The report is referring to the failure of Total System Performance Responsibility (TSPR) and lack of evolutionary strategies in certain acquisitions; neither has agile software development implications*
- **Inadequate contractor oversight**
  - *This concern is also related TSPR; While some agile principles would embed more government personnel in the development process, due to lack of contracting rigor this involvement would be costly and ineffective*
  - ***Also, increasing the acquisition work force has been suggested, but in the current climate of drastic budget cuts it is not feasible***
    - $148–$178B DOD cuts are planned between 2012 and 2016**
    - Additionally, due to looming sequestration, $492B DOD cuts over 10 years

** [Chaplain 2011], **[Weisgerber 2011]*

AEROSPACE

More, GAO-identified Acquisition Problems

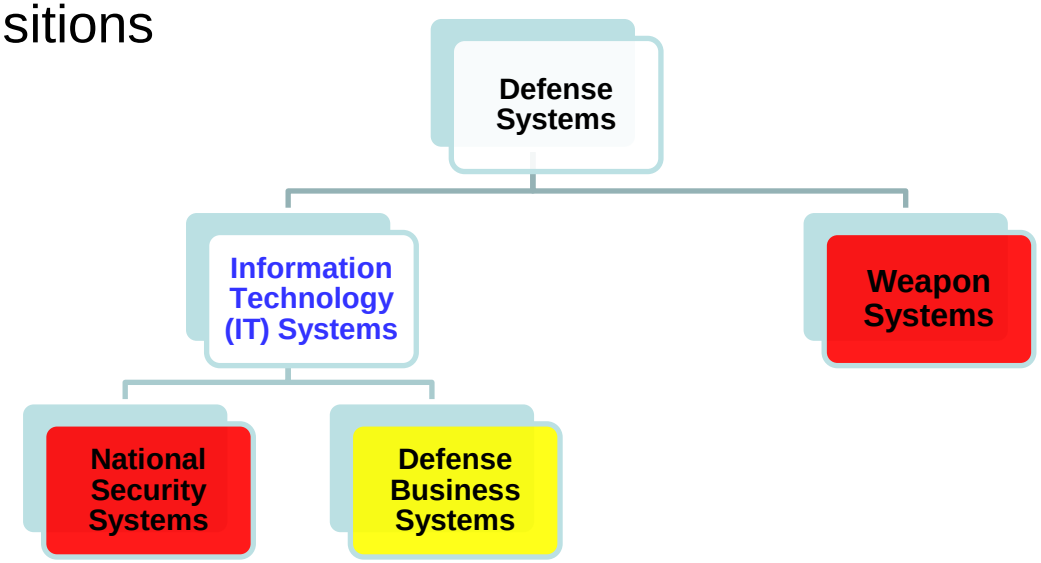
- **Optimistic cost and schedule estimates**
  - *The operative word seems to be "optimistic", which has nothing to do with the details of development methodologies. Additionally, due to the difficulties with macro-estimation in agile development, one can expect further dissatisfaction with the accuracy of cost and schedule estimates*
- **Requirements unstable**
  - *Due to its adaptive nature agile development is supposed to help with handling unstable requirements. However, regardless of the implemented agile project management strategy, volatile requirements will yield inaccurate cost and schedule estimates, ultimately resulting in customer dissatisfaction*
- **Software needs poorly understood**
  - *This is also a requirements and early architecting issue. Again, selected agile development practices do facilitate the gradual, more effective discovery of software-level requirements, but still, software estimates, particularly the early ones, will be grossly inaccurate*
- **Technology immature**
  - *This has not been a software issue on the reviewed acquisitions*

Agility and agile software development still do not seem to be the answers

AEROSPACE

Section 804

- A recent directive by Congress to shape the future of defense acquisitions is
  - ***Public Law 111-84, The National Defense Authorization Act for Fiscal Year 2010; Section 804. Implementation of New Acquisition Process for Information Technology Systems***
- The directive targets the streamlining of Defense Business System (DBS) Acquisitions



Weapon systems and National Security Systems acquisitions are still carried out according to DOD 5000.02

AEROSPACE

Does Section 804 Direct the Incorporation of Agile Methodologies in DOD Software Acquisition?

- Why are we even asking the question?
  - *Quotes from the November 14-15, 2011 National Defense Industry Association (NDIA) Agile Scrum Workshop's invitation*
    - "The law [Section 804] directs the incorporation of Agile methodologies in DOD software acquisition … Agile cannot fail. Unequivocally, Agile cannot fail."
- However what Section 804 actually requires is an acquisition process for DBS acquisitions only, with the following characteristics
  - ***Early and continual involvement of the user***
  - ***Multiple, rapidly executed increments or releases of capability***
  - ***Early, successive prototyping to support an evolutionary approach***
  - ***Modular open systems approach (MOSA)***
- Section 804 requires a new acquisition process but congress cannot (and should not) legislate a software development process

The wording is indeed inspired by agile ideas, but the connection to specific agile software development practices is very weak or nonexistent

AEROSPACE

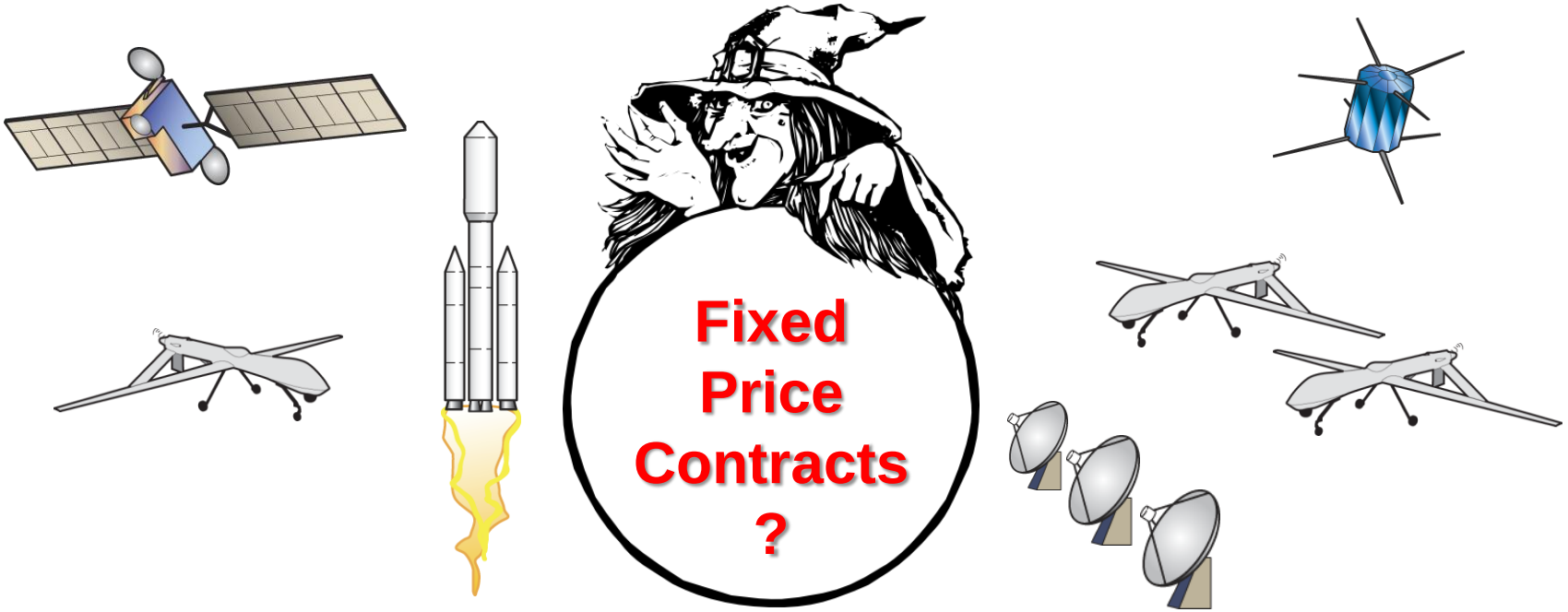
More on Silver Bullets - Spiral Development vs. Agile Methods

- Spiral development is a precedence for a fiasco when the government tried to mandate a development process
  - *Spiral development, first introduced in 1988, is one of the most complex and most powerful adaptive process models*
  - *It looked so promising that in 2003 it was declared as the DOD's preferred development strategy in the DOD 5000.2 instructions*
  - *However, people struggled with the implementation of the process and the lack of success culminated in the elimination of the spiral model from the DOD 5000.02 instructions in 2008*
- Agile development is essentially an "extreme" adaptive process model
  - *Apparently an adaptive process model is being pushed again, without any attempts to understand why spiral development failed and what it takes to manage major acquisitions with adaptive approaches*

"It is déjà vu all over again"
~~~ Yogi Berra

AEROSPACE

What May the Future Bring?



- There is numerous evidence that the government wants to do away with cost-plus contracts and wants to pursue fixed price contracts*
  - *Presidential memorandum*
  - *Office of Management and Budget (OMB) memorandum*
  - *Revised procurement regulations*
  - *Additional, administrative barriers to cost-plus contracts*

** Source: [Chierichella 2012]*

AEROSPACE

However, Further Analysis is Needed

- Fixed price contracts are appropriate to acquire mature products
- In any other situations fixed-price contracts are contra-indicated, e.g.,
  - *One of a kind weapon system development*
    - The development of such systems is always pushing the technology envelope
  - *Information Technology (IT) system development*
    - Most of the time IT systems end up being one of a kind systems, regardless of the contractor's sales pitch and buzzwords like web-based, Cloud, Service Oriented Architecture (SOA,) etc.
    - These systems need to be viewed as **system of systems** with emergent characteristics and inter-twined elements across multiple organizations or even multiple military, intelligence, or other government departments
  - *Unfortunately, due to foreseeable budget cuts and additionally, due to the looming danger of sequestration the push for fixed price contracts might prevail*

However, remember the Iron Triangle? Pursuing agile development and fixed price contracts at the same time is not feasible

AEROSPACE

AEROSPACE
Assuring Space Mission Success

Conclusions

Conclusions - 1

- Continuing problems in the software enterprise (earlier the symptom was called the “software crisis”) forced organizations to continuing experimentation with new development methods
  - *Part of this experimentation is manifested in the rediscovery and sometimes just renaming of known processes*
  - *Experimentation is further fueled by the “bandwagon effect”*
  - *Unfortunately, there is no sufficient data with acceptable quality available to properly characterize the emerging agile methods and establish a reliable performance baseline*
- In the meantime, defense acquisitions of software-intensive systems are still struggling and there is no effective solution in sight
  - *The demand for bigger and more sophisticated weapon systems is constantly increasing while the scaling problem of processes and the management of the continuously growing scope are not resolved*
  - *Also, a tendency for blind copying of industry practices is present due to a persistent opinion that “industry knows what to do and we should just adopt industry practices”*
    - Unfortunately, the associated risks are not well understood and in some cases are explicitly covered up

AEROSPACE

Conclusions - 2

- What Acquisition Program Office personnel needs to do
  - *Continuously educate itself on the emerging development methods*
  - *In the contracting phase must insist on the use of robust development standards*
    - The government should not settle for vague references to agile programming; must insist on a detailed Software Development Plan (SDP) that fully characterizes all planned life cycles, their internal relationships, and the planned implementation details of all life cycle processes and associated activities
    - Mission success criteria and synergy with mission assurance needs must be used to validate the SDP before acceptance for the contract
  - *In the contract monitoring phase must implement an effective mission assurance program*
    - Mission assurance is essentially an ingrained instrumentation of the development process; it is a necessity and must not be allowed to be viewed by the development organization as a “nice to have”, negotiable feature

AEROSPACE

Conclusions - 3, or What You Really Need to Remember…

"The temptation to 'cut corners,' even in the name of being efficient or 'expedient,' is ever-present, especially in a global business that is economically unforgiving...

That is why 'getting it right' must be a 24/7 commitment.“

~ Dr. Wanda Austin, President and CEO, The Aerospace Corporation

AEROSPACE

Acronyms

APO	Acquisition Program Office
ATIP	Aerospace Technology Program
BDUF	Big Design Up Front
CEO	Chief Executive Officer
CMMI	Capability Maturity Model Integration
COTS	Commercial Off-the-shelf
DAPA	Defense Acquisition Performance Assessment
DoD	Department of Defense
FAR	Federal Acquisition Regulation
FFRDC	Federally Funded Research & Development Center
GAO	General Accountability Office
GUI	Graphical User Interface
IBM	International Business Machines
IID	Iterative-Incremental Development
IOT&EP	Initial Operational Testing & Evaluation Plan
IT	Information Technology
IV&V	Independent Verification & Validation
JCIDS	Joint Capabilities Integration & Development System
JROC	Joint Requirements Oversight Council

KDSI	Thousand Delivered Source Instructions
LTCD	Long Term Capability Development
MOSA	Modular Open System Architecture
NDIA	National Defense Industry Association
OSD	Office of the Secretary of Defense
OT&E	Operational Test & Evaluation
PPBE	Planning, Programming, Budgeting & Execution
RAD	Rapid Application Development
RUP	Rational Unified Process
SE&I	Systems Engineering & Integration
SETA	Systems Engineering and Technical Assistance
SEU	Single Event Upset
SW	Software
TCO	Total Cost of Ownership
TEMP	Test & Evaluation Management Plan
TQM	Total Quality Management
TSPR	Total System Performance Responsibility
XP	eXtreme Programming
YAGNI	You Aren't Gonna Need It

References - 1

Adams 2005	Adams, R. J., et al, *Software Development Standard for Space Systems*, The Aerospace Corporation Technical Report TOR-2004(3909)-3537, Revision B, March 11, 2005
Agile 2001	Agile Alliance, *Manifesto for Agile Software Development*, 2001, <http://www.agilealliance.org>
Ambler 2006	Ambler, S. W., The Agile Unified Process, <*http://www.ambysoft.com/unifiedprocess/agileUP.html*>
Ambler 2007	Ambler, S. W., Agile Documentation Strategies, *Dr. Dobb's Journal*, February 05, 2007
Ambler 2011	Ambler, S. W., Agile/Lean Documentation: Strategies for Agile Software Development, <*http://www.agilemodeling.com/essays/agileDocumentation.htm*>
Baird 2002	Baird, S., Extreme Programming Practices in Action, December 6, 2002, *<http://www.informit.com/articles>*
Beck 2000	Beck., K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley 2000
Beck 2004	Beck., K., *Extreme Programming Explained: Embrace Change (2nd Edition),* Addison-Wesley, 2004
Boehm 2004	Boehm, B., Turner, R., *Balancing Agility and Discipline – A Guide for the Perplexed*, Addison-Wesley, 2004
Brooks 1987	Brooks, F., No Silver Bullet – Essence and Accidents of Software Engineering, *IEEE Computer* 20(4):10-19, April 1987
Bruyere 2011	Bruyere, C. N., et al, Employment dynamics over the last decade, *Monthly Labor Review*, August 2011, pp 16-29
Chaplain 2011	Chaplain, C. T., *DOD Delivering New Generations of Satellites, but Space System Acquisition Challenges Remain*, Government Accountability Office Report GAO-11-590T, May 11, 2011
Chierichella 2012	Chierichella, J., The Budget Crunch, Fixed Price Contracts, and Lessons of the Past, *National Defense*, July 2012
Cockburn 2004	Cockburn, A., *Crystal Clear: A Human-powered Methodology for Small Teams*, Addison-Wesley, 2004
Cohen 2011	Cohen, J., Does Pair programming Obviate the Need for Code Review? <*http://www.softwarequalityconnection.com/2011/04/does-pair-programming-obviate-the-need-for-code-review/*>, April 1, 2011
Collins 2012	*Collins Free Online English Dictionary*, http://www.collinsdictionary.com/dictionary/english/agile

AEROSPACE

References - 2

DAPA 2006	*Defense Acquisition Performance Assessment (DAPA) Report*, March 2006
DoD 2008	*DoD 5000.02, Instructions on the Operation of the Defense Acquisition System*, Signed 8 December 2008
INCOSE 2003	*INCOSE Systems Engineering Handbook*, INCOSE-TP-2003-016-02, Version 2a, June 1, 2004
Erwin 2009	Erwin, S. I., Pentagon brass: Stay away from management bestsellers, *National Defense*, August 1, 2009
Eslinger 2006	Eslinger, S., *Mission Assurance-driven Processes for Software-intensive Ground Systems*, The Aerospace Corporation Technical Report ATR-2006(8056)-1, September 30, 2006
Guarro 2007	Guarro, S. B. and Tosney, W.F. (editors), *Mission Assurance Guide*, The Aerospace Corporation Technical Report TOR-2007(8546)-6018, 1 July 2007
Guarro 2008	Guarro, S. B. and Hecht, M., Risk and Reliability Assessment of Software-Intensive Systems, *Space Systems Engineering and Risk Management Symposium*, Los Angeles, California, 26 February 2008
Hannay 2009	Hannay, J. E., et al, The effectiveness of pair programming: A meta-analysis, *Information and Software Technology* 51(2009), pp 1110-1122
Hedges 1981	Hedges, Larry V., Distribution theory for Glass's estimator of effect size and related estimators, *Journal of Educational Statistics* 6 (2): 107–128, 1981
Highsmith 2000	Highsmith, J. A., *Adaptive Software Development – A Collaborative Approach to Managing Complex Systems*, Dorset House Publishing, 2000
Jacobson 1999	Jacobson, I., et al, *The Unified Software Development Process*, Addison-Wesley, 1999
Jacobson 2006	Jacobson, I., The Essential Unified Process – an Introduction, <http://www.ivarjacobson.com/essup.cfm>
Krafcik 1988	Krafcik, J. F., Triumph of the lean production system, *Sloan Management Review* 30 (1): 41–52, 1988
MSF 2006	*Microsoft Solution Framework for Agile Software Development Process Guidance*, <http://www.microsoft.com/downloads>
Nosek 1998	Nosek, J. T., The Case for Collaborative Programming, *Communications of the ACM*, March 1998/vol.41, No. 3
Palmer 2002	Palmer, S.R., and Felsing, J.M., *A Practical Guide to Feature-Driven Development*, Prentice Hall, 2000

AEROSPACE

References - 3

Palmer 2010	Palmer, S. R., Inspections, <*http://www.step-10.com/SoftwareProcess/General/InspectionNotes.html*>
Paparone 2009	Paparone, C.R., From Not-So-Great to Worse – The Myth of Best Practice Methodologies, *Defense AT&L*, July-August 2009
Poppendieck 2003	Poppendieck, M., Poppendieck, T., *Lean Software Development: An Agile Toolkit for Software Development Managers*, Addison-Wesley, 2003
Poppendieck 2006	Poppendieck, M., Poppendieck, T., *Implementing Lean Software Development: From Concept to Cash*, Addison-Wesley, 2006
Royce 1998	Royce, W., *Software Project Management - A Unified Framework*, Addison-Wesley, 1998
Rule 2011	Rule, P. G., What do we mean by "Lean?", <*http://www.smsexemplar.com/wp-content/uploads/20110921-what-do-we-mean-by-lean-v1b-IncludingNotes.pdf*>
Schwaber 1995	Schwaber, K., Scrum Development Process, Business Object Design and Implementation, OOPSLA '95 Workshop Proceedings, *Springer-Verlag Telos*, 1997
Stack 2008	What is a metaphor in the context of XP?, blog, < *http://stackoverflow.com/questions/211557/what-is-a-metaphor-in-the-context-of-xp*>
Stansell 2004	Stansell, J., "Extreme Programming Explained: Embrace Change Second Edition" book review, <*http://c2.com/cgi/wiki?ExtremeProgrammingExplainedEmbraceChange*>
Stapleton 2003	Stapleton, J., *DSDM: Business Focused Development*, Addison-Wesley, 2003
Takeuchi 1986	Takeuchi, H., Nonaka, I., The New Product Development Game, *Harvard Business Review*, January-February 1986
VersionOne 2010	*The State of Agile Development – State of Agile Survey 2010*, VersionOne, 2010
Voas 2001	Voas, J., Faster, better, and cheaper, *IEEE Software* 18 (3) (2001) 96–99
Weisgerber 2011	Weisgerber, M., DoD expects up to $100B more in cuts, White Paper, *Federal Times*, March 28, 2011

AEROSPACE

AEROSPACE
Assuring Space Mission Success

Backup

Representative Agile Software Development Methods

- Agile UP (Agile Unified Process) [Ambler 2006]
- ASD (Adaptive Software Development) [Highsmith 2000]
- Crystal Clear [Cockburn 2004]
- DSDM (Dynamic Systems Development Method) [Stapleton 2003]
- Ess UP (The Essential Unified Process) [Jacobson 2006]
- XP (eXtreme Programming) [Beck 2000], [Beck 2004]
- FDD (Feature-Driven Development) [Palmer 2002]
- Lean Software Development [Poppendieck 2003, Poppendieck 2006]
- MSF (Microsoft Solution Framework) for Agile Development [MSF 2006]
- Scrum [Schwaber 1995]

AEROSPACE

XP Practices

- The original* XP practices
  - *The planning game*
  - *Small releases*
  - *Metaphor*
  - *Simple design*
  - *Continuous integration*
  - *Continuous testing*
  - *Refactoring*
  - *Pair programming*
  - *Collective code ownership*
  - *40-hour work week*
  - *On-site customer*
  - *Coding standards*

** This list is based on [Beck 2000]*

AEROSPACE

The Planning Game

- The planning game is a metaphorical name for requirements engineering and increment/iteration planning
  - *It is essentially a meeting where the team is working through a stack of index cards that contain the user stories*
  - *Each required feature is described and elaborated in a user story (another metaphor…)*
- Responsibilities during the planning game*

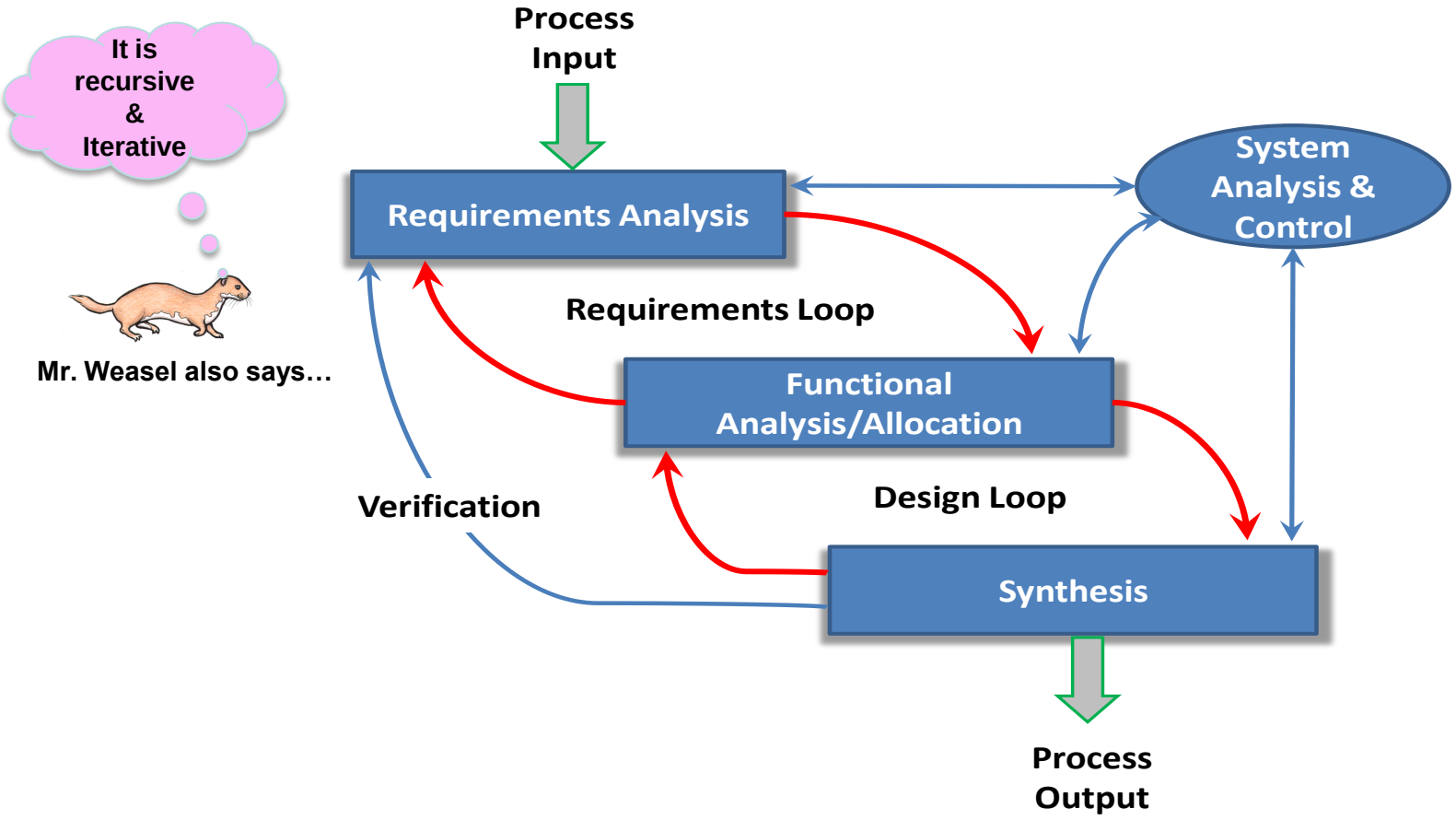
Customer	Developer
Define scope of the release	Estimate how long each user story will take
Define order of delivery	Communicate technical impacts of implementing requirements
Set dates and times of release	Break down user stories into tasks and allocate work

** Source: [Baird 2002]*

AEROSPACE

The Planning Game - 2

- However, the needed, overall systems engineering process that provides the context for software development is more complex [INCOSE 2003]



Well, Mr. User, are you ready to take direct responsibility for the progress?

AEROSPACE

Small Releases

- Start with the smallest feature set, release early and often
- Duration
  - *Releases may be provided every 1-3 months*
- Concerns
  - *The earlier mentioned customer problem*
    - The need for excessive participation and associated responsibility in the planning and validation of these releases are not feasible
  - *Scaling issue*
    - In large systems it might be difficult to come up with a finite, incremental feature set to field tangible releases that the customer could appreciate

Having small, internal releases is a good engineering practice but the customer should not be responsible for validating these releases

AEROSPACE

Metaphor

- Each project supposed to have an organizing metaphor
  - *Metaphors facilitate the dialog between the user and developer*
  - *Metaphors serve as a bridge between the terminology of the customer's domain and the software engineering jargon*
  - *A metaphor of the metaphor: "Tribal Language"*
- Example metaphor
  - *"Describing an agent-based information retrieval system we might say that this program works like a hive of bees, going out for pollen and bringing it back to the hive"**
- This practice is quite benign (as opposed to "extreme") and its cost is negligible. However, its value has not been proven.

Use of metaphors do not seem to represent any risks

* *Source: [Stack 2008]*

AEROSPACE

Simple Design

- Keep the design as simple as possible for the moment and don't add features that are not needed for current functionality
  - *The reasoning behind this practice is that if a feature is not valuable now, it is not worth the investment until it becomes valuable*
  - *Simple design is the practice-level implementation of the earlier introduced YAGNI concept and the avoidance of the supposedly bad approach of big design up-front (BDUF)*
- Keeping designs simple is a good idea in general
- However, the operative phrase in this definition is "**for the moment**"
  - *Remember Heisenbugs? Prudent consideration for all the overarching, nonfunctional requirements (like reliability, availability, etc.) requires extensive up-front design and thorough follow-up during development*

A shortsighted, "extreme" implementation of this practice might lead to a Mission Assurance exposure

AEROSPACE

Continuous Integration and Continuous Testing

- Continuous Integration
  - *Integrate with the whole system as often as feasible*
- Continuous testing
  - *Unit testing and acceptance testing are alternating according to the rhythm of the process, which is driven by the duration of the applied timeboxes*
    - Unit tests, written by developers to test functionality as they implement it
      - *Conceptually, it is not different from any other approaches*
      - *In agile development a **test-driven strategy** is preferred where the unit test suite is developed before coding starts and the execution of these tests is automated – no particular mission assurance exposure here*
    - Acceptance tests
      - *The tests themselves supposed to be specified by the user/customer*
      - *The user/customer has to observe all tests or review test runs*
        - In either case the user/customer is expected to approve test results according to the dictated process' rhythm
        - **However, see our earlier interim progress tracking concerns:**

This is an undue burden on the customer – continuous acceptance tests are not feasible in a large project

AEROSPACE

Refactoring

- Refactoring is a technique to improve code without changing functionality
  - *It is a declared XP virtue to refactor late in the design to increase performance*
- Examples
  - *Repartitioning the code to smaller, easier to maintain chunks*
  - *Renaming some variables to be more descriptive*
  - *Re-evaluating the need for temporary variables*
  - *Extracting common behavior into a single code segment*
  - *Candidates for refactoring may be found via the "smell test"*
    - Large program segments or classes
    - Deeply nested code
    - Long parameter list
    - Presence of switch (case) statements
    - Redundant code (e.g., a class that does not seem to do anything,) etc.
- Risks
  - *Every technique that changes a running or working system is not immune to introducing errors, even if it is claimed that no functionality is impacted*
  - *"Refactoring in the small" can be helpful but "refactoring in the large" does not make sense and it is a dangerous practice*

Refactoring must not be used as a replacement for proper architecting

AEROSPACE

Pair Programming

- Collaborative programming is not a new idea; it has been explored before*
- Pair programming is a collaborative technique to ensure quality code
  - *People are paired-up at a workstation and working together*
  - *However, it is not like a piano-duet on the computer keyboard* ☺

  - *The members of the pair have different roles and those roles may change*
  - *People may change pairs too as needed*
- Pair programming is one of the most debated agile practices
  - *Its effectiveness is evaluated on the following three dimensions when it is compared to solo programming: Effects on **quality**, **duration**, and **effort.***

** See [Nosek 1998]*

AEROSPACE

The Effectiveness of Pair Programming

- The reported results are based on a meta-analysis of 18 detailed studies*
  - *The goal of a meta-analysis is to estimate the overall, combined effect*
  - *Rigorous meta-analysis ensures the standardization of the reported data sets and provides comparable effect estimates*
  - *In meta-analysis, rather than computing a simple mean, more weight is assigned to studies that carry more information*
- Authors used two different statistical models; we present their conclusions for a so-called “fixed-effects model”
  - *A fixed-effects model assumes an unknown but fixed population*
  - *All 18 studies are seen as data drawn from the same population and variances between individual studies are viewed as results of subject variability*

Caveat: Effort, duration, and quality are not well-defined in general and are operationalized in very diverse ways

* *Source: [Hannay 2009]*

AEROSPACE

Reported Meta-analytic Effects of Pair Programming

- A little statistics
  - *The standardized measure of effect size is Hedges' **g****
    - An effect size of .5 indicates that the mean of the pair programming group's distribution is half a standard deviation larger than the mean of the reference group's (the solo programmer's) distribution
      - *Effect sizes larger than 1.0 are "large", 0.38 - 1.00 are "medium", and 0 - 0.37 are "small"*
- Effect sizes from the meta-analysis

Effect on	Effect Size [*g*]	Description of Effect
Quality	**+0.23**	Small significant positive
Duration	**+0.40**	Low-medium significant positive
Effort	**- 0.73**	Medium significant negative

- In plain English: Minor quality improvement and some schedule compression can be achieved at the price of somewhat higher cost

Even in plainer English, "Faster, Better, Cheaper" does not work here either**

** [Hedges 1981]; ** [Voas 2001]*

AEROSPACE

Mission Assurance Risk in Pair Programming

- XP, while does not explicitly forbids formal inspections, treats them as redundant and unnecessary
  - *XP proponents claim that inspection happens all the time through pair programming*
  - *However, that pair programming is a general improvement over formal inspections (also called Peer Reviews) remains unproven**
- Unique benefits of formal inspections
  - *Inspectors' independence from the creator of the inspected work product*
    - "The issue is closeness, not ability. That's why every writer needs an editor**"
      - *Note that we used the term "work product", which is broader than "code"*
  - *Knowledge transfer (although should not be treated as a training vehicle…)*
  - *Improving the process, like adding items to checklists, recommending tools like a static code analyzer, recommending changes to coding standards, etc.*
  - *Reevaluating assumptions that were made earlier about requirements*
  - *Capturing and evaluating quality metrics, identifying common problem areas*

Despite of its positive impact on quality, pair programming is not an acceptable replacement for formal inspections

** [Palmer 2010]; ** [Cohen 2011]*

Collective Code Ownership

- No single person "owns" a module
  - *Any developer is expected to be able to work on any part of the code-base at any time*
- In theory this is a good practice regardless of the used software development methodology
- Caveat: In reality the practice does not scale up
  - *There are limits to how much code evolution can somebody follow real-time*
  - *Also, programmers are no longer equal – like in medicine, high-level specialization is the current reality*
    - Specialization examples
      - *Database developers, Graphical User Interface (GUI) developers, algorithm developers, networking specialists, infrastructure specialists (formerly called “system programmers”,) etc.*

Collective code ownership, if applied properly, has a positive impact

AEROSPACE

40-Hour Work Week

- Programmer welfare is considered important
  - *XP development is considered a stressful environment*
  - *Programmers should go home on time*
    - Up to one week of overtime is allowed (Note that this is an XP guidance and not a Human Resources (HR) policy)
    - Consecutive weeks of overtime is a sign that the process might be failing
- My take
  - *Not just XP but software development in general is a stressful endeavor*
  - *Everybody should go home on time, not just programmers …* ☺

AEROSPACE

On-site Customer

- According to this practice, the developers have continuous access to a real, live customer
  - *Note that this is different (and much more involved) than the traditional Rapid Application Development (RAD) approach, where the customer primarily participated in early prototyping*
  - *It is also different from the prevailing, periodical program management reviews where customer representatives are present*
- In case of large, geographically distributed teams this expectation is not feasible
  - *Development of large systems usually involves geographically distributed teams; the distributed structure of the organization is essentially a liability and source of numerous risks that need to be dealt with*
- The excessive burden on government personnel makes the practice also infeasible

However, the main risk is the underlying issue that the customers are now made implicitly responsible for all decisions and progress

AEROSPACE

Coding Standards

- The written code must be homogeneous
  - *One should not be able to tell by looking at the code who on the team wrote or corrected a piece of it*
  - *This practice is closely related to Collective Code Ownership*

Following coding standards is an unconditionally good practice regardless of the software development methodology used

AEROSPACE

XP Practice Evolution – New XP Practices*

- The planning game
  - ***Quarterly Cycle*** *and* ***Weekly Cycle*** *are replacing the old practice*
- Small releases
  - ***Incremental Deployment*** *and* ***Daily Deployment*** *are introduced*
- Metaphor
  - *It was always the least understood practice and now it is* ***eliminated***
- Simple design
  - ***Incremental Design*** *and* ***Single Code Base*** *are introduced*
- Continuous integration
  - *No change*
- Continuous testing
  - *Emphasis on* ***Test-First Programming***
- Refactoring
  - ***Eliminated*** *as a formal practice; became part of* ***Incremental Design***
- Pair programming
  - *No change*

*** See [Beck 2004] for the description of new XP practices*

AEROSPACE

New XP Practices-2

- Collective code ownership
  - *It is now called **Shared Code***
- 40-hour work week
  - ***Eliminated**, **Energized Work** and **Slack** replaces this practice*
    - Energized Work is a reinterpretation of the sustainable pace concept
    - Slack means to mark things that can be dropped if you get behind
- On-site customer
  - ***Sit Together**, **Whole Team**, and **Real Customer Involvement** practices were introduced*
- Coding standards
  - *Not called out anymore but it is still a foundation of Shared Code*

- There are more new practices; a new value; and several new principles but we were only focusing on the evolution of the original 12 practices

AEROSPACE

Use of Trademarks, Service Marks and Trade Names

Use of any trademarks in this material is not intended in any way to infringe on the rights of the trademark holder. All trademarks, service marks, and trade names are the property of their respective owners.

The clip art on slides 8, 15, 20, and 76 is courtesy of Animation Library

The clip art on slide 16 is courtesy of Florida's Educational Clearing House

The clip art on slide 24 is courtesy of Mountain Goat Software

The clip art on slide 34 is courtesy of PicGifs

The clip art on slide 51 is courtesy of Wallpapers4Desktop

The clip art of a witch on slide 60 is courtesy of All-free-download

All other clip arts on slide 60 are owned by The Aerospace Corporation

The clip art on slide 74 is courtesy of Elee Kirk

The picture on slide 80 is courtesy of Dr. David Bader