



Iterative and Agile Project Management: Rethinking PMBOK, CMM and ISO 9000

University of Management and Technology

1901 North Fort Myer Drive

Arlington, VA 22209

Voice: (703) 516-0035 Fax: (703) 516-0985

Website: www.umtweb.edu



Objectives of This Presentation

This presentation has two major objectives:

- To present a nutshell view of alternative approaches to software development that differ dramatically from traditional approaches. The alternative approaches – called iterative and agile development techniques – emphasize *flexibility*, *learning* and *risk reduction* on projects.
- To show how evolving approaches to project management challenge the established wisdom of assorted standards as captured by *The PMBOK Guide*, CMM, and ISO 9000. These standards have major impacts on how people work – Are they able to keep up with new developments?



Prologue

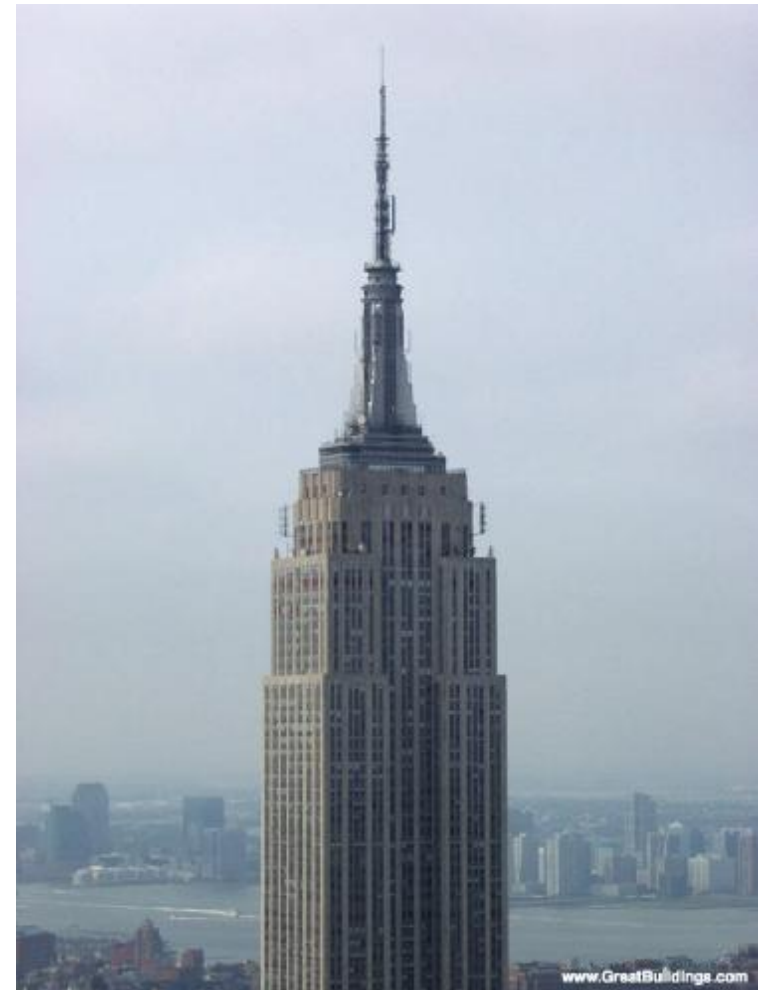


Life before Project Management

- 26 years before PERT
- Nearly 40 years before WBS
- 38 years before PMI
- 56 years before *PMBOK*

The Empire State Building was constructed in one year and two months. Its original design was developed in two weeks.

Could we match that achievement today?





Life before Project Management

- 17 years before PERT
- Nearly 30 years before WBS
- 29 years before PMI
- 47 years before *PMBOK*

The Bantam jeep was designed and built in two months.

Could we match that achievement today?





Life after Project Management: The Drive toward Maturity

- Since the mid-1980s, project management has undergone a major process of *maturity*:
 - 1969 – Project Management Institute created
 - 1984 – first PMI certification exam
 - 1987 – first *Project Management Body of Knowledge (PMBOK)*
 - 1990 – promulgation of Capability Maturity Model; PMI membership less than 8,000 people; average of 55 people per year certified in 6 years
 - 1996 – first *Guide to the Project Management Body of Knowledge*; PMP certification has begun an exponential climb
 - 2007 – PMI membership stands at one-quarter million people; half-million people have achieved certification since mid-1990s; 230,000 people are actively certified; *PMBOK Guide* is *de facto* world standard.



Life after Project Management: The Drive toward Maturity

- Maturity as a double-edged sword
 - Maturity = consistency, reliability
 - Maturity = depleted energy, prisoner of legacy, process over innovation
- In his 1983 book, *Industrial Renaissance*, William Abernathy pointed out that for companies to survive and thrive, they must undergo period episodes of *de-maturation*.
- Big Question today: Is it time to address the de-maturation of project management processes?
- Iterative and agile approaches to project management reflect a step towards de-maturation.



PART I.

NUTSHELL OVERVIEW OF ITERATIVE AND AGILE TECHNIQUES



Alternative Software Development Approaches

- There are many alternative S/W development life-cycle approaches you can employ to develop software. Here we look at three broad categories being used today:

- Traditional (specifically, the Waterfall approach)
- Iterative (specifically, RAD Application Prototyping, time-boxed development, and Rational Unified Process (RUP) development)
- Agile (XP, Scrum)



Waterfall Approach



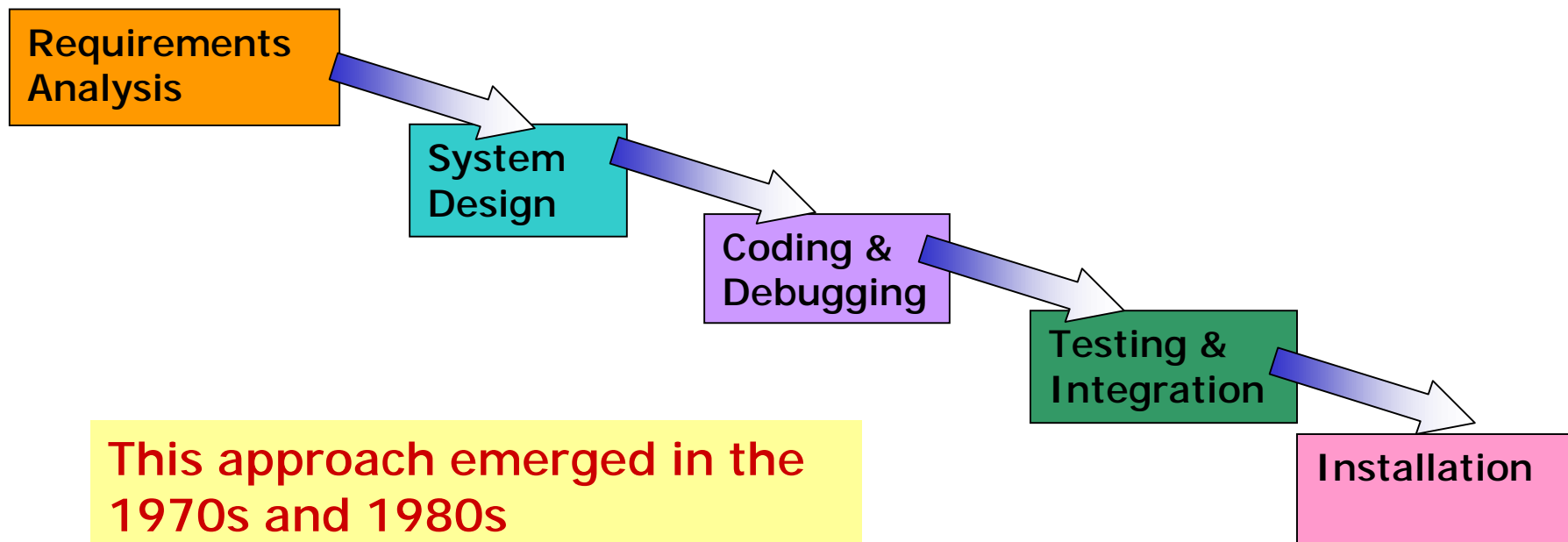
Thumbnail Sketch: Waterfall Approach

- ☉ This methodology emerged in the 1970s and 1980s as software development moved from an ad hoc effort to a more structured discipline. It is heavily process-driven.
- ☉ The Waterfall life cycle approaches software development in a step-by-step, linear fashion. First you elicit requirements for the system. Once this is done, you design the system. Then you construct the system in accordance with the design. And so on.
- ☉ Ironically, the Waterfall concept was articulated by W. W. Royce in 1970 in a publication that *criticized* linear SW development. Royce was one of the key players in developing the iterative perspective!



The Waterfall Model

With the waterfall approach, you do not proceed to the next phase until the previous one is complete.





Iterative Approaches



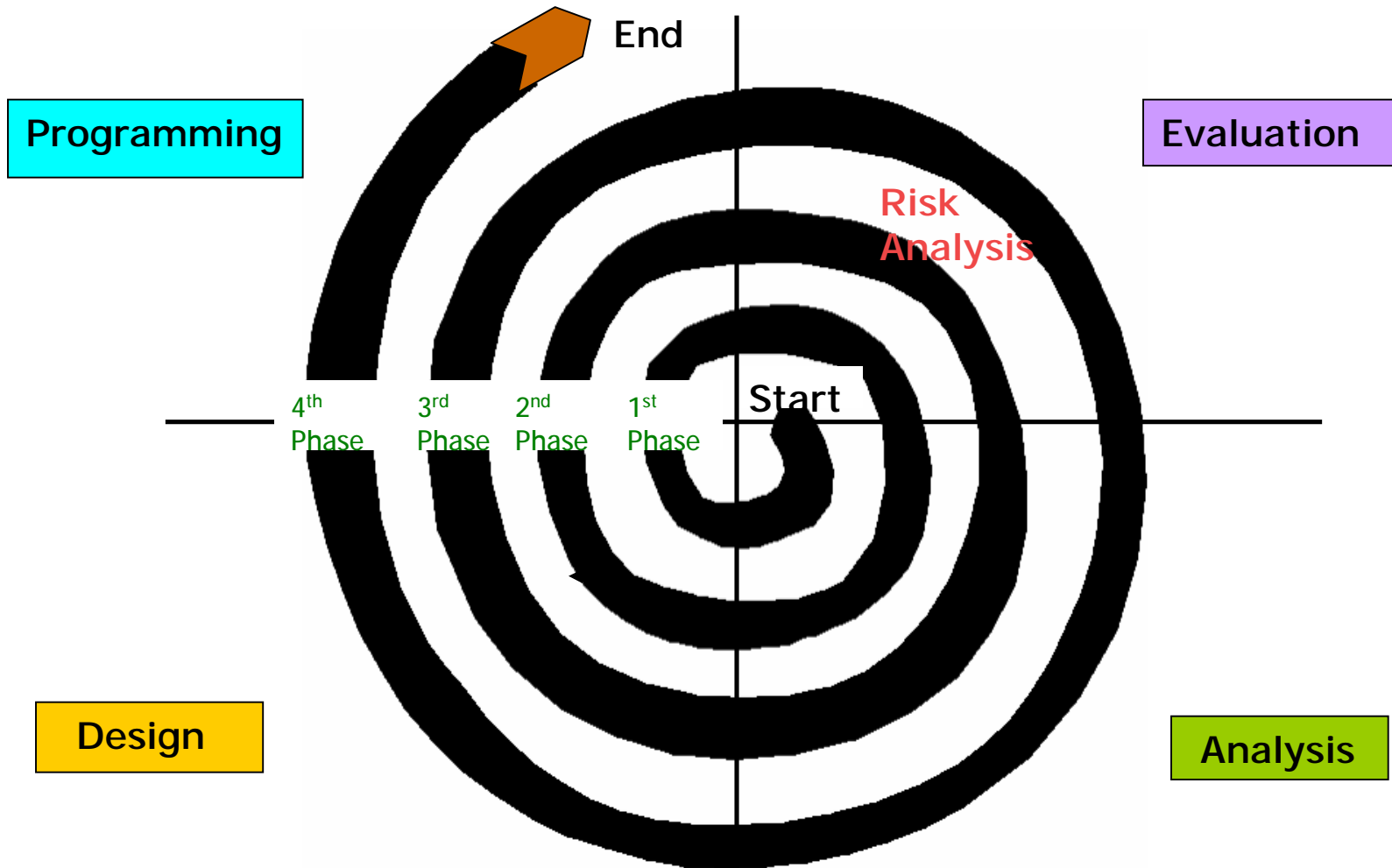
Thumbnail Sketch: Iterative Approaches

- Iterative development focuses on eating the elephant piece by piece, rather than in one big chunk (Waterfall approach).
- Iterative techniques actually pre-date the Waterfall approach. They became associated with the enormously popular *structured techniques*, which in turn were associated with top-down design.
- Many of the best known names in S/W development were proponents of iterative approaches:

- W.W. Royce (1970s)
- Harlan Mills (1970s)
- Frederick Brooks (1970s)
- Bernard Boar (1980s)
- Barry Boehm (1980s)

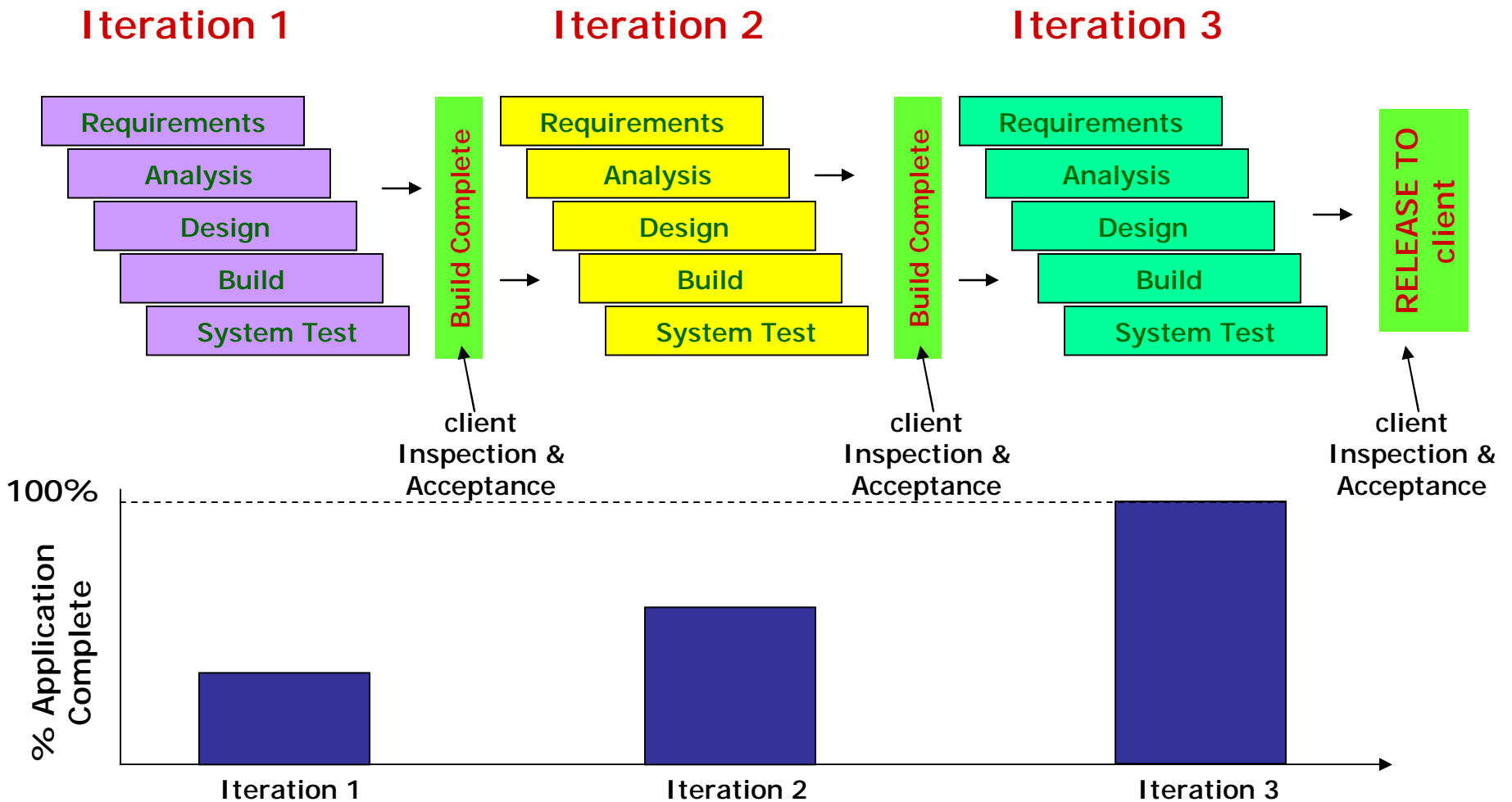


The Spiral Model (Barry Boehm)





Iterative Development Can Incorporate Waterfall Thinking





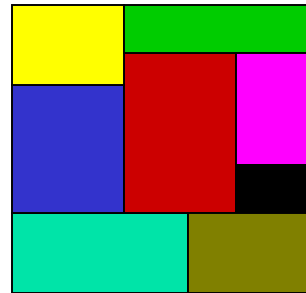
Two Variants of Iterative S/W Development

- The term *iterative S/W development* is employed to describe an evolutionary process to developing S/W, in contrast to the let's-develop-the-whole-thing approach associated with *waterfall S/W development*
- There are two variants of the iterative approach:
 - Top-down iterative development
 - Staged development

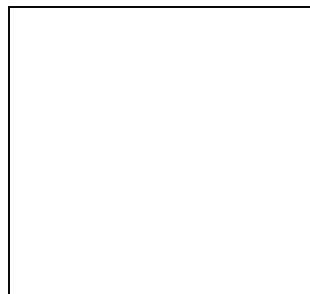


Top-down Iterative Development Approach

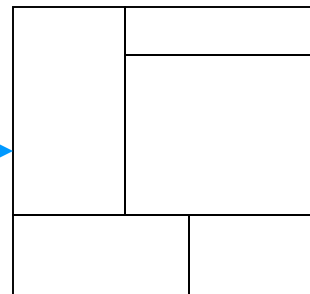
Example: RAD
Application
Prototyping



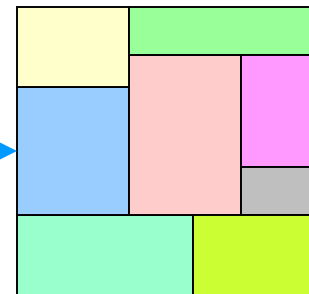
Desired Deliverable



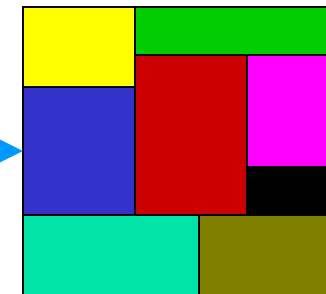
Round 0
No
functionality



Round 1
No
functionality



Round 2
No
functionality

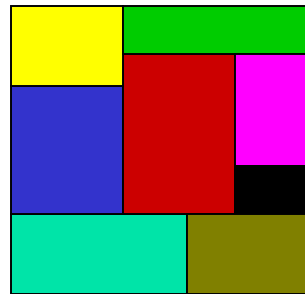


Round 3
100%
functionality

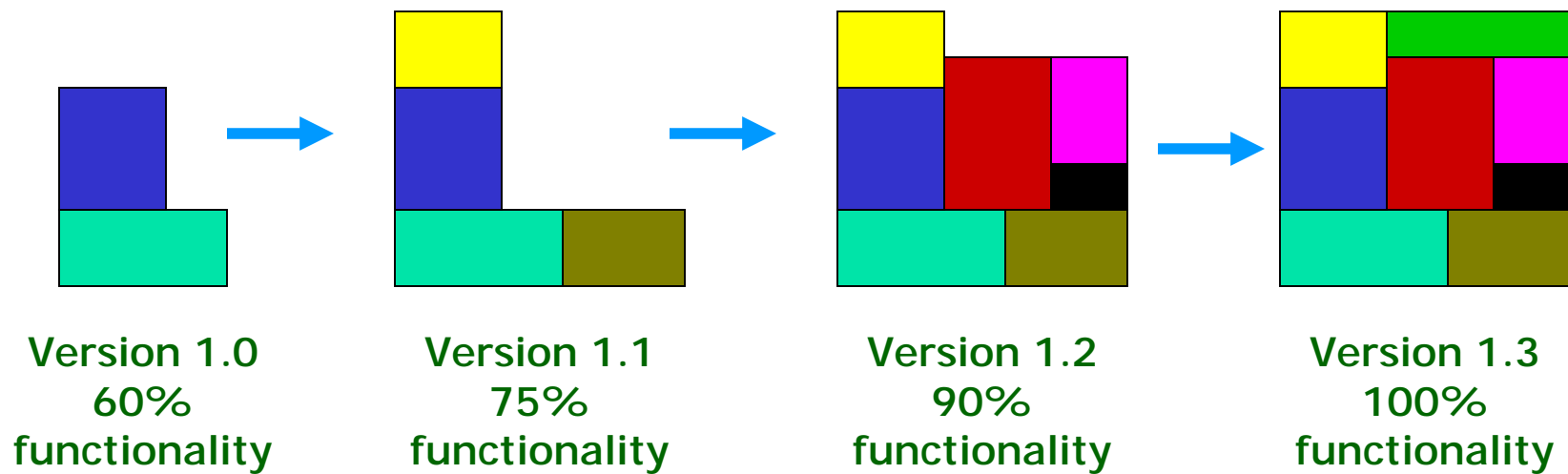


Staged Iterative Development Approach

Example: Time-boxed scheduling



Desired Deliverable





Agile Approaches



Thumbnail Sketch: Agile Approaches

- Agile S/W development approaches became popular in the 1990s. They hold that rapid change and complexity defeat attempts to develop systems built on traditional *engineering process control*, where you anticipate every possible contingency and define rules to govern them. Projects should be managed *opportunistically*.

- Problems with engineering process control:

With increased complexity and rapid change, S/W development environments became so “noisy” that this approach could no longer produce predictable results.

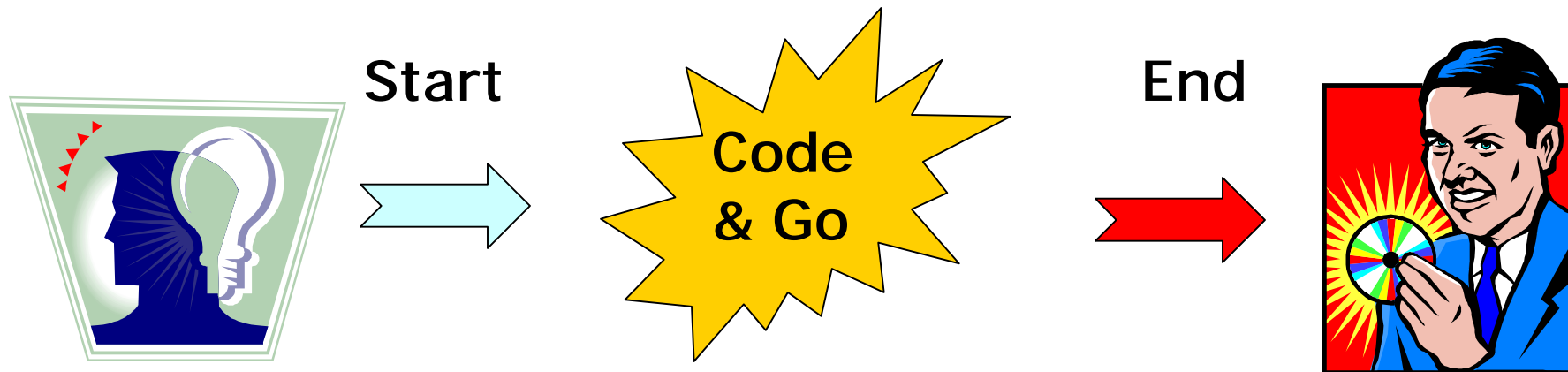
- The agile solution:

Manage change and complexity through intense communication between the development team members and users.



Concern: When Agile Becomes Code-and-Go

Critics of agile techniques are concerned that their lack of formalism may degenerate into a code-and-go approach, where you start with a concept and wind up with a product without employing discipline.





Which Approach Is Best?



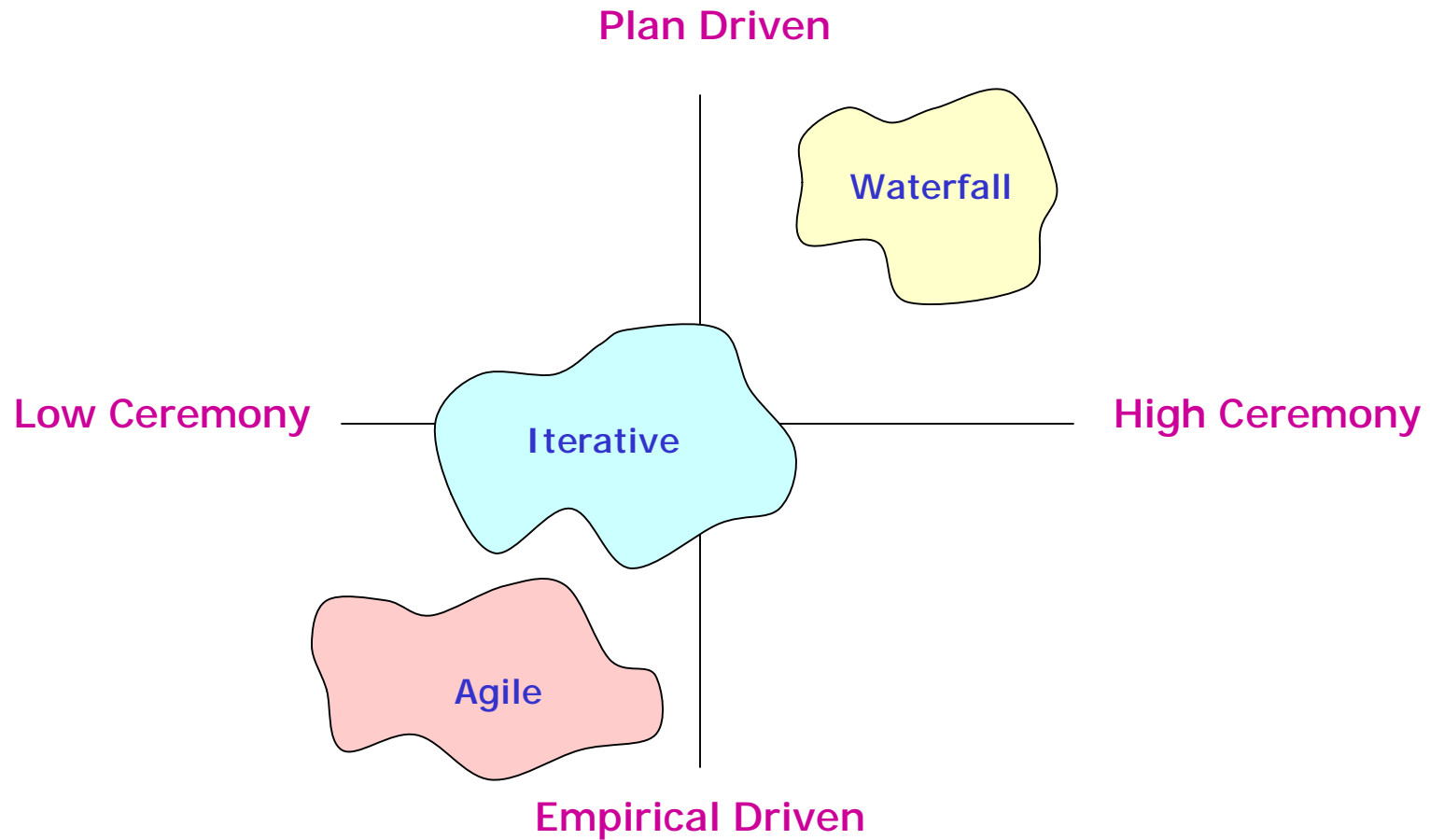
Which S/W Methodology Is Best?

- Obvious answer: There is no single *best* methodology.
- The approach employed in developing S/W must be determined *situationally*.

- If your client requires you to identify milestones and deliverables for the 18 month life of a project, you likely need to follow the *Waterfall* approach.
- If you need a flexible approach with discipline, you should consider adopting an *iterative* approach.
- If you are operating under severe time constraints or with radically changing technologies, you need to be *agile*.



Ceremony vs. Agility





Cultural Dimensions of the Different Development Approaches

Each of the three S/W development approaches carries with it cultural baggage:

- **Waterfall model**

- Comfort with linear development
- Comfort with discipline
- Dependence on processes – if you encounter problems, default to the processes

- **Iterative approaches**

- Desire to avoid eating an elephant in one gulp – rather, eat pieces
- Recognition that certainty only extends 4-8 weeks
- Recognition that development must be based on non-linear learning

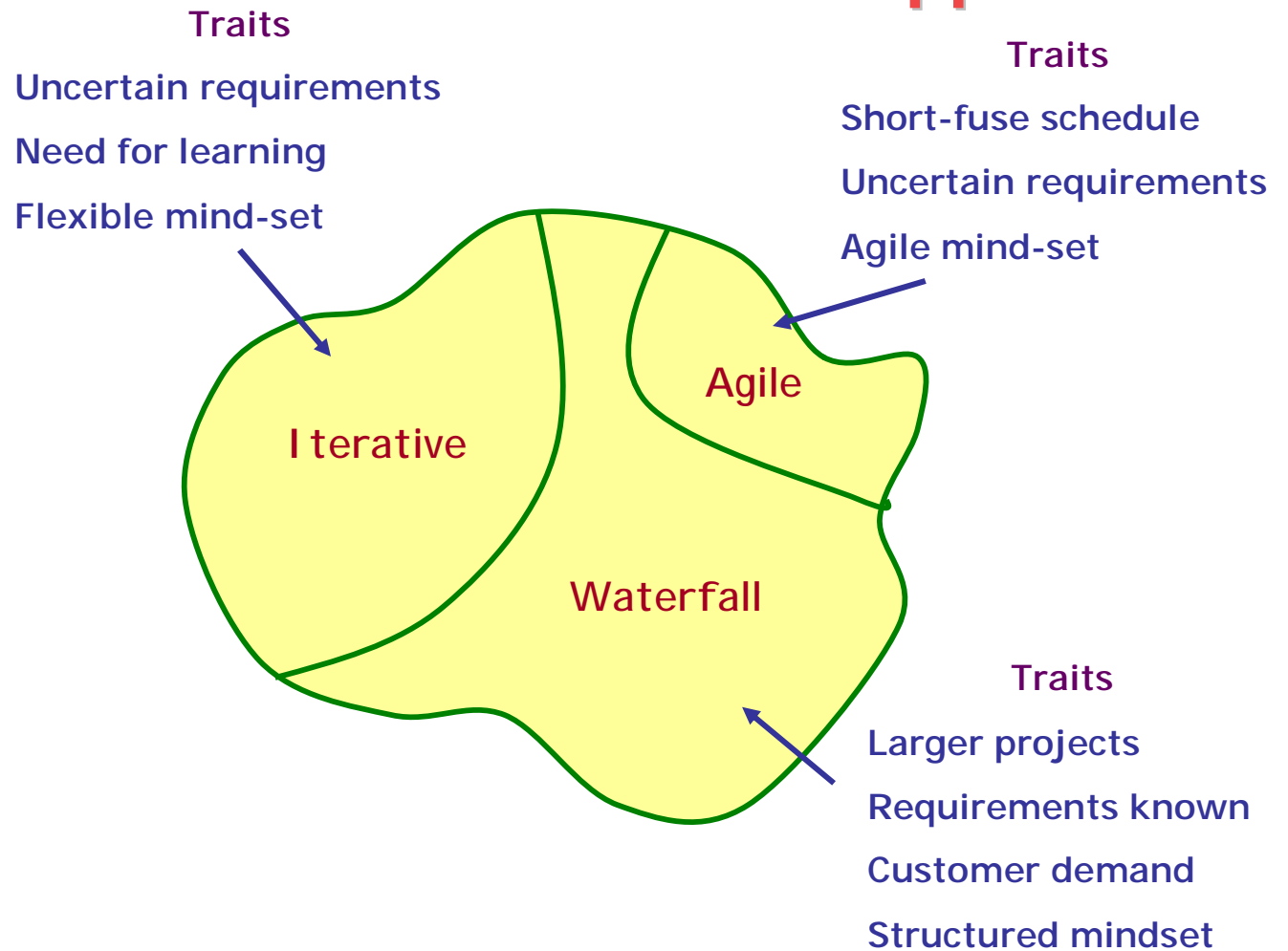
- **Agile approaches**

- Impatience with formal processes
- Heavy dependence on opportunistic development, driven by learning
- Belief that certainty only extends about a month
- Heavy belief that development must be based on non-linear, empirical learning



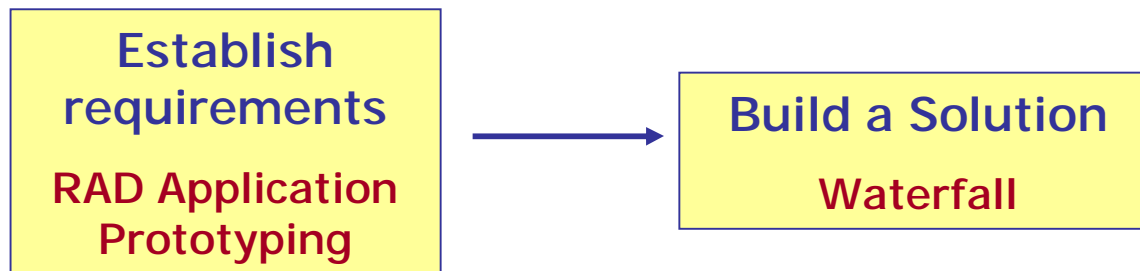
SDLC Is Not an Either/Or Decision: A Portfolio Approach

Selecting a proper SDLC is not an either/or proposition. A portfolio approach to employing SDLCs can be taken.





SDLC Is Not an Either/Or Decision: Combining SDLCs



Example of Combining SDLCs

In order to identify customer-focused requirements, RAD application prototyping can be employed. Once customer-focused requirements are well-defined, the system can be built with the a Waterfall approach, incorporating careful planning and high ceremony.



Some Specific Methodologies

**Time-Boxed Scheduling, RAD
Application Prototyping, RUP, and
Scrum**



Time-boxed Scheduling



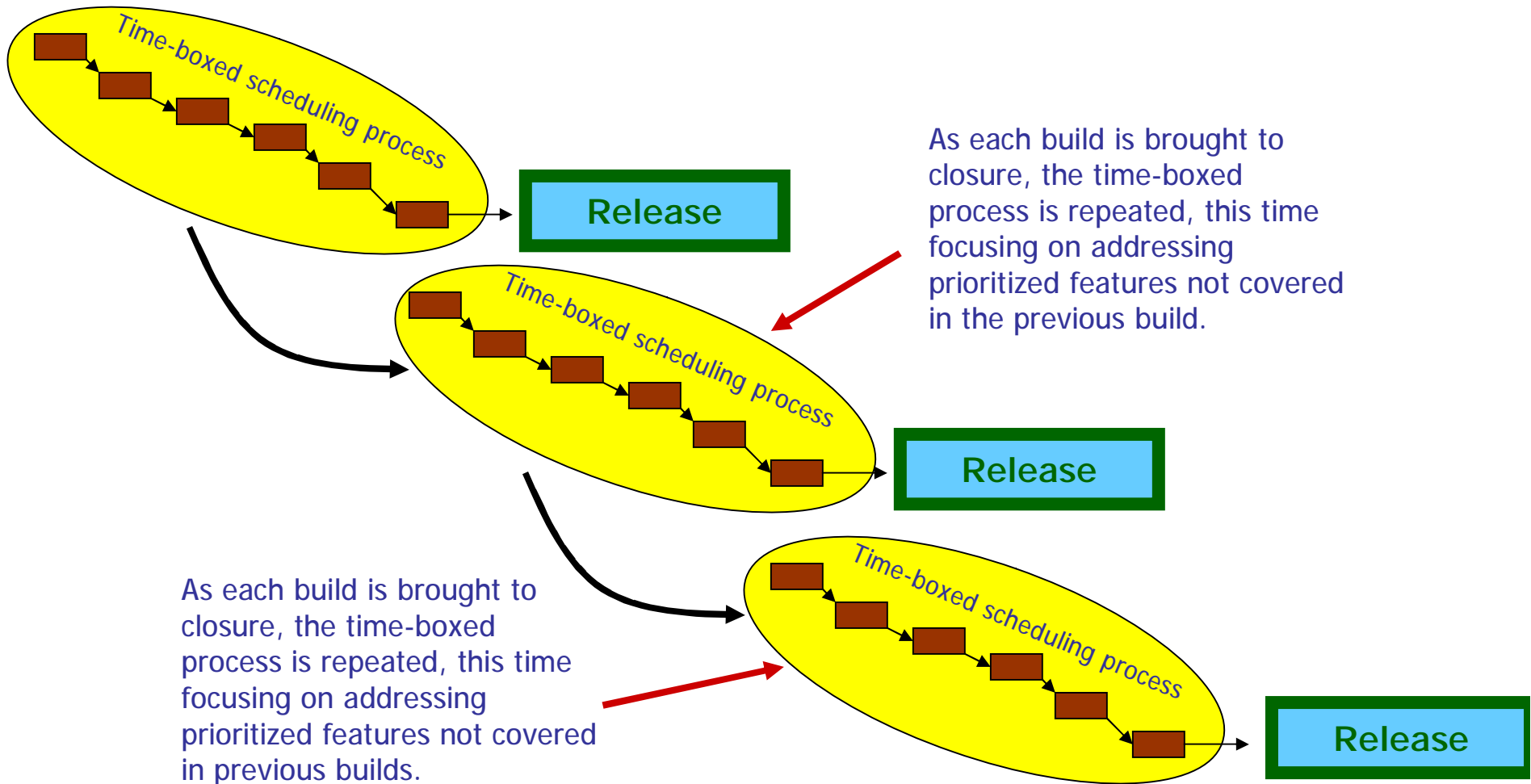
Time-boxed Scheduling

- ☛ With time-boxed scheduling, producing deliverables by a specific short-fuse target date is paramount
- ☛ To produce the deliverable quickly, both clients and developers must recognize that “you can’t have it all” – some requirements will need to be trimmed from the client’s wish list
- ☛ If clients insist on keeping features that have been dropped, these features can be included in future versions of the deliverable. Thus time-boxed scheduling lends itself to iterative S/W development, where new releases are created with each iteration.
- ☛ Key issue: How to determine which requirements to keep and which to drop?
 - Decisions must be made jointly by clients and developers





Time-boxed Scheduling Extended to Produce Releases Iteratively





Opportunistic Scheduling



Opportunistic vs. PERT/CPM Scheduling

- The best known project management tools are the Gantt/PERT/CPM charts, which lie at the heart of scheduling packages, such as MS Project.
- Agile and iterative S/W development practitioners – such as those people using time-boxed scheduling and Scrum – often need to abandon rigidly pre-determined Gantt/PERT/CPM logic in favor of *opportunistic scheduling*.

- If testers are suddenly available, use them now, if appropriate.
- If a piece of equipment arrives sooner than expected, integrate it into your solution now, if appropriate.



Opportunistic Scheduling

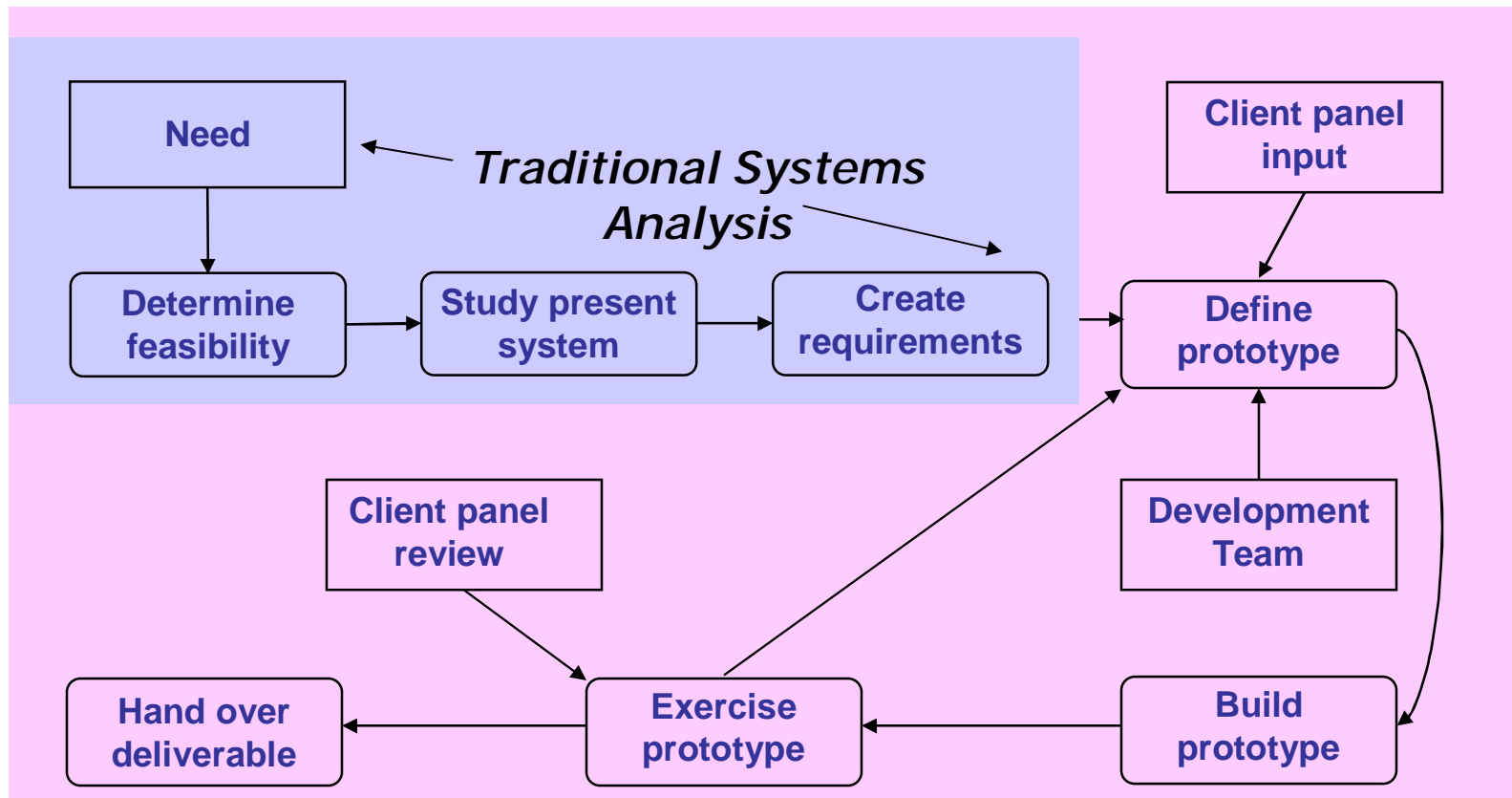
- If we need to speed things up, we may need to move away from rigid PERT/CPM task sequences:
 - Can we do things in parallel that we would normally do sequentially?
 - Can we identify resources that may be able to help us speed up the work effort?
 - Can the team figure out ways to do the job more effectively?
 - Can we identify new technologies that build our effectiveness?
 - Can we identify new opportunities that will enable us to do work earlier than scheduled (e.g., the unplanned availability of extra resources; the early arrival of equipment)?
 - In prioritizing features to deliver, what features do clients most want?
 - Can we cut back on functionality that adds little value?



RAD Application Prototyping



The Big Picture





The Myth of Slower Development Time

- ☉ One of the severest criticisms of application prototyping is that it stretches out development time, since at the end of the prototyping effort, all you have is a set of requirements. You still need to build the system!
- ☉ This criticism is largely without merit. By getting the requirements right with application prototyping, you gain client acceptance and avoid producing systems that are rejected by clients, or that entail large amounts of changes in order to get things right. By gaining client acceptance through client involvement, project life-cycles can be shortened dramatically.
- ☉ Warning: Clients may initially see the process as being slow. They need to be educated that because it dramatically improves requirements development, it can in fact be much quicker than traditional development approaches that entail substantial rework.



Rational Unified Process (RUP)

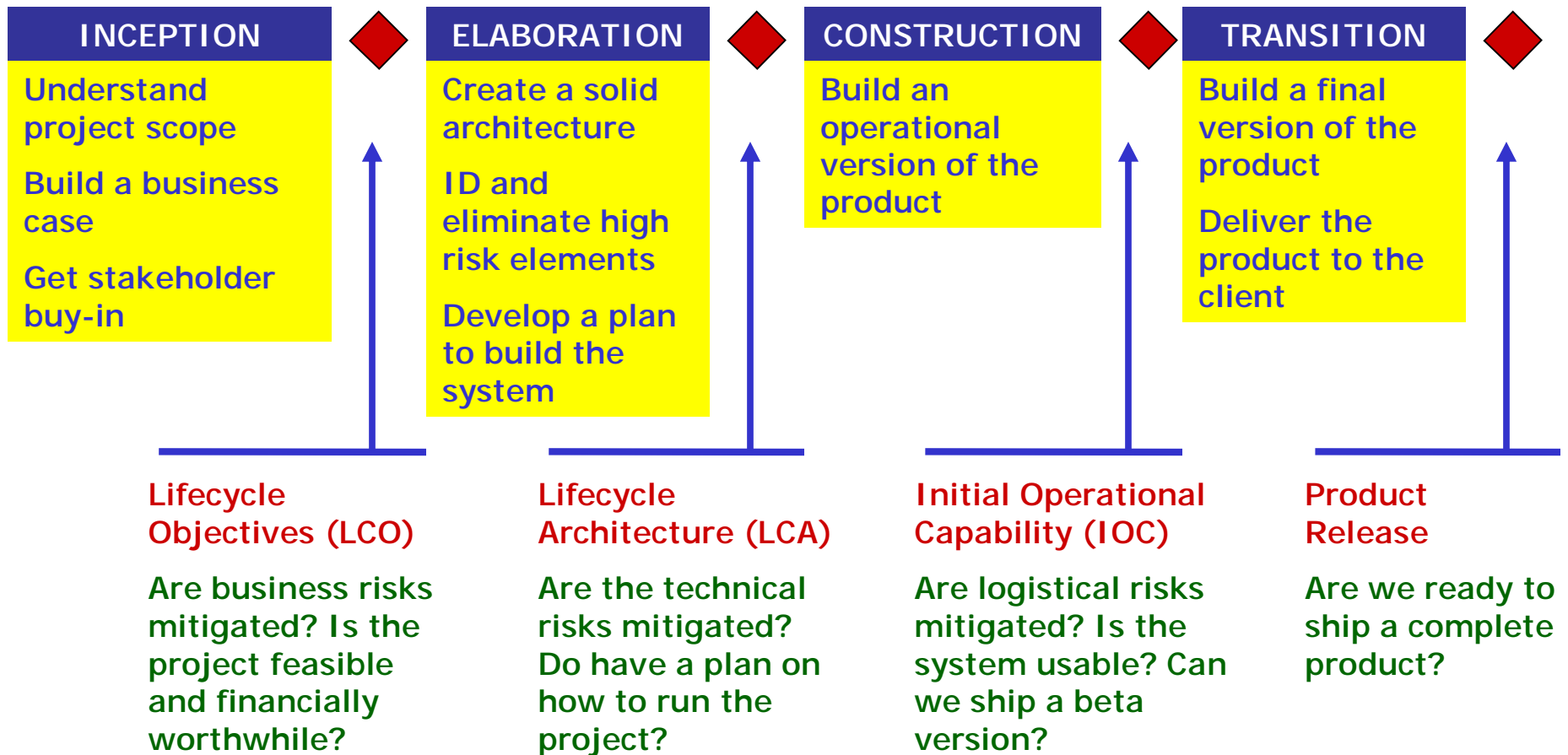


Some Distinguishing Features of RUP

- The Rational Unified Process (RUP) emerged in the 1980s as a spin-off of Boehm's spiral development model.
- There are many approaches that S/W developers can take these days. What distinguishes RUP from the other approaches? There are five features that RUP focuses on that differentiate it. These are:
 - Mitigate risk as early as possible
 - Baseline an executable architecture as early as possible
 - At the end of each iteration, make sure you have produced demonstrably executable S/W
 - Recognize that change is inevitable – Don't be rigid. Prepare yourself to accommodate change as early as possible
 - Build your system with components

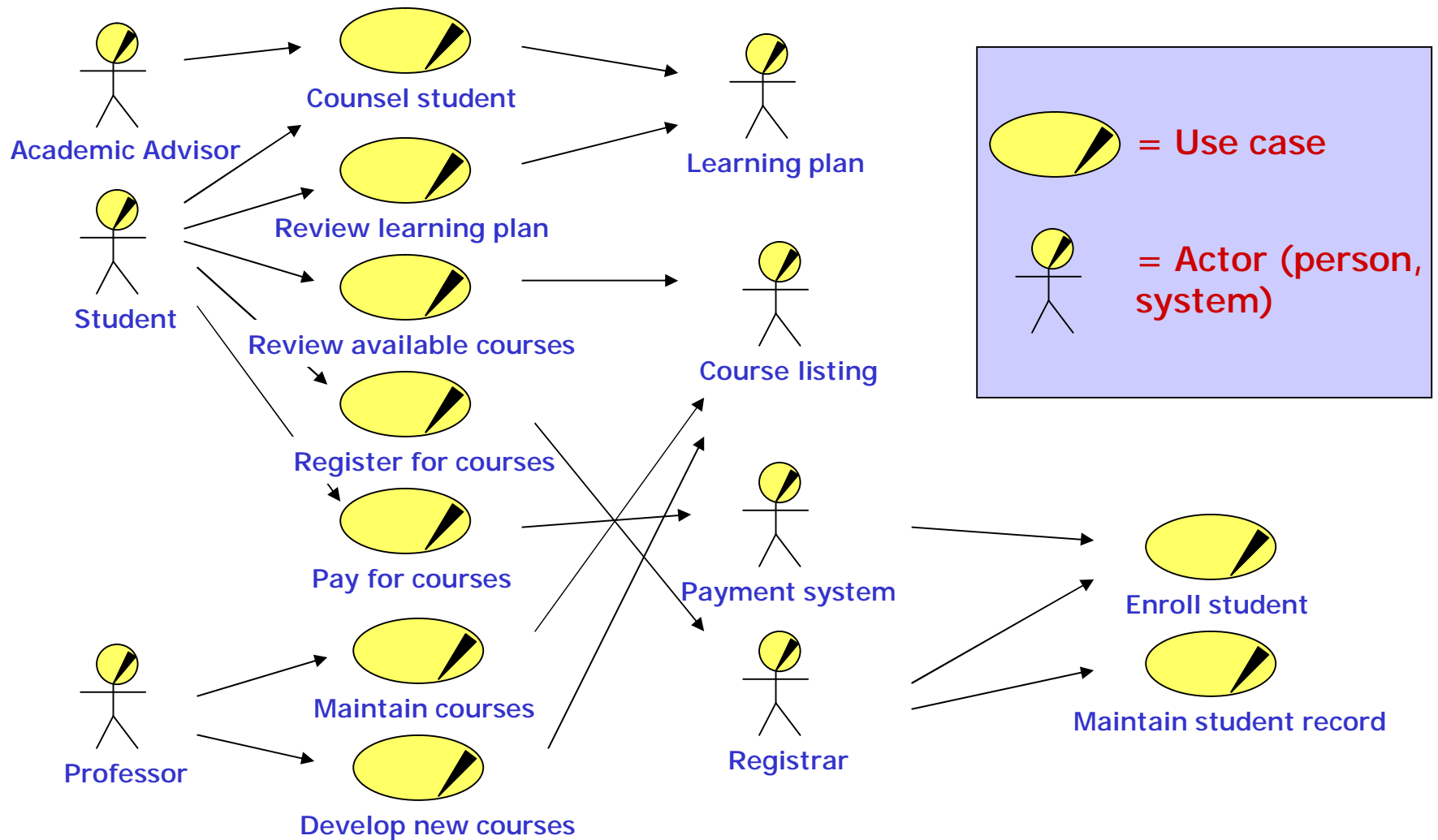


RUP Phases: An Overview





User Input with Use Case Diagrams: Student Course Registration





Example of Agile Development: Scrum

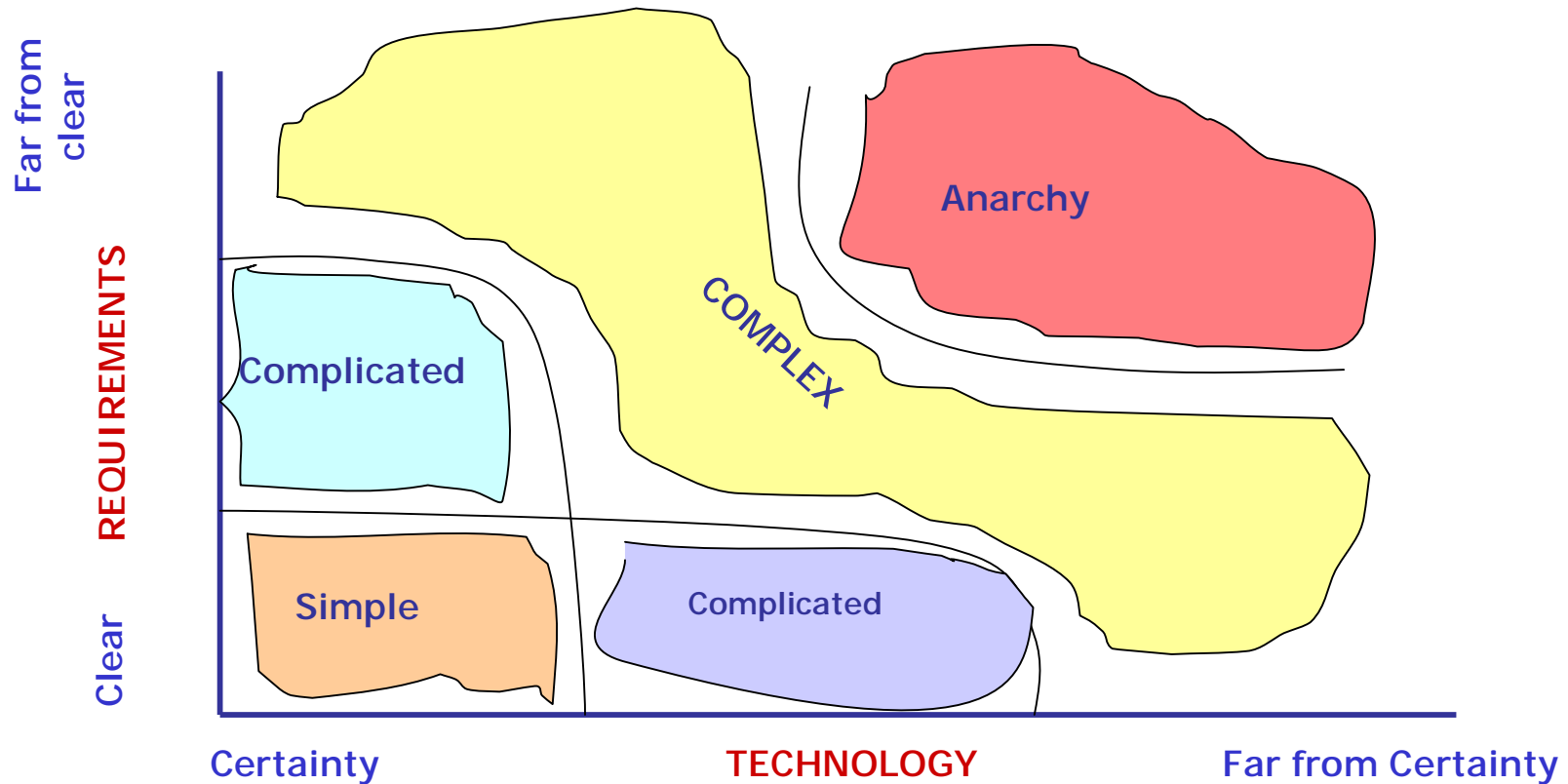


Introduction

- The best known of the agile development techniques is Scrum
- Scrum's basic premise is that beginning in the 1980s, S/W development efforts became “noisy” along three dimensions:
 - Requirements: There is uncertainty and change regarding what the product should address
 - Technology: Technologies being computerized, as well as computer technologies, are changing so rapidly that few people understand them expertly
 - People: People factors are poorly understood – skills levels vary among S/W team members, different people interpret information differently, team dynamics contribute to success and failure, etc.



Requirements, Technology, and Levels of Complexity



From Schwaber and Beedle, *Agile Software Development with Scrum*, Prentice-Hall, 2002, p. 93



Engineering Process Control Lacks Value in Highly Dynamic Environments

- Engineering process control entails defining the steps of a process in detail, and checking frequently to make sure the steps are followed and the desired outcomes are achieved
- Problem: As systems become more complex, it becomes impossible to define processes in detail. Consider a typical situation facing processes today:
 - In general, if you encounter $A + B + C$, carry out process X1.
 - However, if C lies below a threshold value of 1,212 googles, then carry out process X2.
 - Also, if the combined value of $A + B$ is more than 122% greater than the value of C, then carry out process X3,
 - except when the value of C is greater than 1,345 googles, in which case carry out process X4



Opportunistic Scheduling: Lessons from Scrum

- Scrum is the best known of the agile S/W development techniques. It is based on making sure that team members continually communicate with each other. The theory is that if all the team members know what their fellow team members are doing, they are more likely to recognize opportunities for speeding up and improving their collective work effort.
 - Team members meet each day during daily *scrum sessions*. They also meet after completing an iteration of effort (which they call a *sprint*), to see what they did right and wrong.
 - The team also receives guidance from a *scrum master*, who oversees the scrum management process. A goal of the scrum master is to identify how to speed things up and enable the team to function more effectively. The *product backlog* is maintained by the *product owner*.
 - Bottom line: With scrum, there is a constant search for opportunities to speed things up and to increase team effectiveness. Development is driven by continual *learning*.



Conclusions about Agile and Iterative Techniques



Some Challenges to Iterative and Agile Approaches

- Some unresolved issues regarding iterative and agile approaches:
 1. How to deal with increasing scale of projects when using iterative and agile techniques
 2. How to manage agile teams when team members are geographically dispersed (i.e., virtual teams)
 3. How to plan for iterative and agile efforts (e.g., How do you establish budgets for such projects?)
 4. Distinguishing between:
 - Outputs associated with iterations
 - Enhancements
 - Scope creep
 5. Determining whether “average” players can function effectively on agile teams
- NOTE: The fact that Waterfall approaches allow you to plan projects in detail does not make the plan right!! It does not mean you got the requirements right!!



Iterative and Agile Development Are Based on Learning

- The iterative and agile approaches to software development are not new. Some of the best known players in the history of software development espoused iterative and agile approaches – including Royce, Boehm, Mills, and Boar.
- The underlying premises of iterative approaches dovetail nicely with current management thinking, which stresses that in an era of complexity and rapid change, organizations must be *learning organizations*. With iterative approaches, learning is gained from iteration to iteration and is incorporated into the product being developed.



PART II. IMPACTS OF EVOLVING PM PRACTICES ON PM STANDARDS



The Standards Dilemma

- An important function of standards is to establish consistent practices in a profession. For example:
 - *PMBOK Guide* – Project management standards
 - Capability Maturity Model (CMM) – Software development standards
 - ISO 9000 – Quality standards
- The dilemma: Standards generally reflect current practice. In a sense, by doing so they look backward. As practices evolve, standards need to adjust to the changes – otherwise, they become a barrier to progress.



Example of the Challenge to *PMBOK* Presented by Agile and Iterative Practices

- The *PMBOK Guide* (2004) and PMI's *Practice Standard for Work Breakdown Structures* (2nd. Ed, 2006) emphasize the central role of WBSs in planning projects.
- While WBS construction is well-suited to the Waterfall approach to S/W development, how well suited is it to agile and iterative approaches?

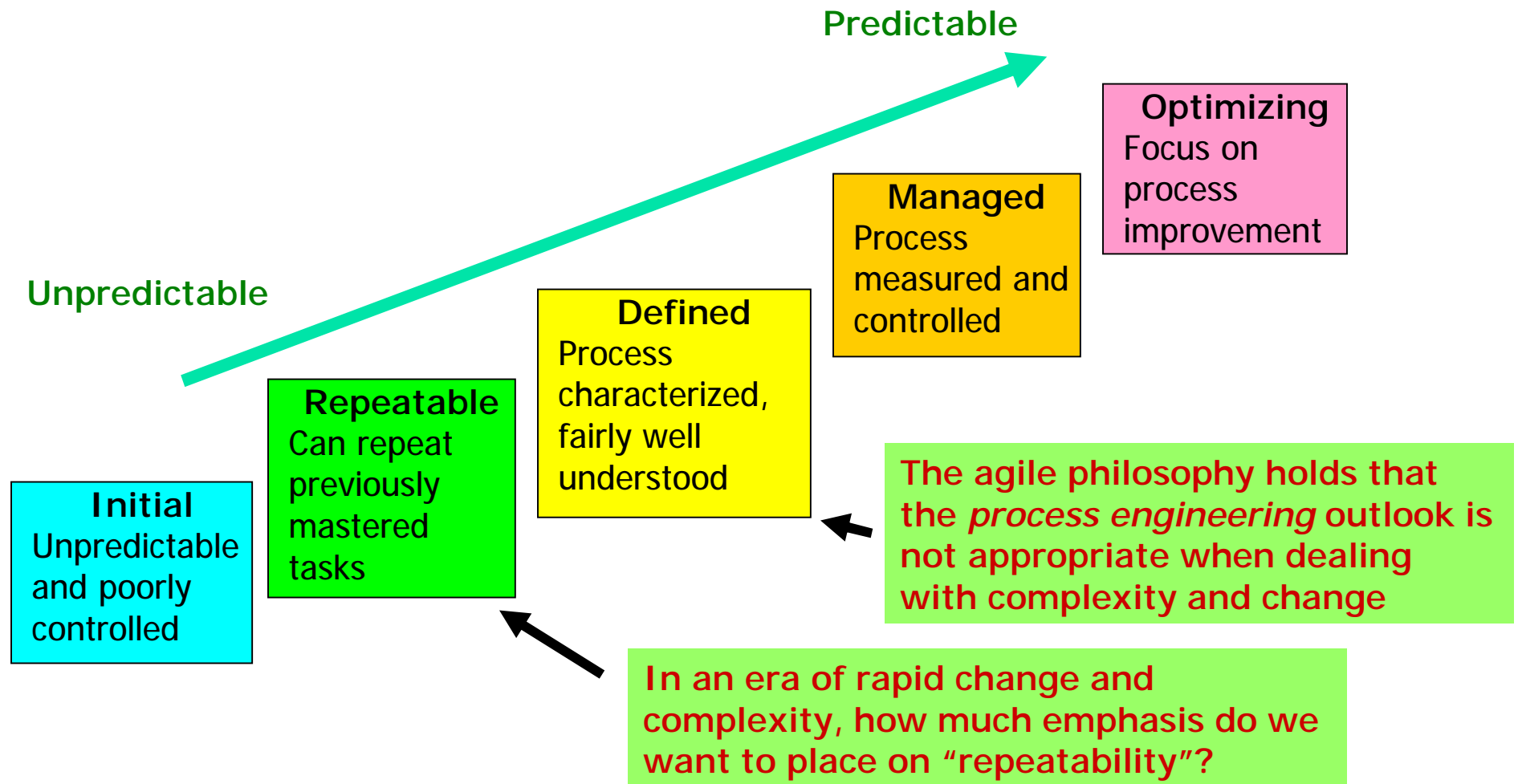


Example of the Challenge to *CMM* Presented by Agile and Iterative Practices

- The Capability Maturity Model has become the standard for introducing “maturity” in software development in organizations. Its primary premise is that S/W development should move from *ad hoc* efforts to controlled processes.
- While CMM Level 2 (Repeatable) and Level 3 (Defined) are suitable to the Waterfall approach, how well-suited are they to agile and iterative approaches?



Stages of Maturity in the CMM





Example of the Challenge to *ISO 9000* Presented by Agile and Iterative Practices

- As is the case with CMM, the ISO 9000 quality perspective strives to achieve consistent output by use of well-defined processes. Ultimately, enterprises that receive ISO 9000 certification are evaluated on the basis of their processes rather than the products they produce.
- Yet agile and iterative development methodologies are based on highly flexible processes. With agile practices, the direction taken by the project may be determined day-by-day (e.g., through daily scrums). Certainly, all agile and iterative practices make heavy use of *learning* and take advantage of unforeseen *opportunities* in order to define future directions taken by projects.



Last Word

- When selecting a development approach to executing projects, you need to take into account a variety of factors, including:
 - People: Working style and psychological proclivities of team members; requirements of customers
 - Project: Complexity, size, and time-frame of the project, in addition to the risk associated with the project effort and the product being produced
- When selecting a project development approach, you are not facing an either/or situation. A blended approach often makes sense.
- It is good from time to time to reconsider the standards we adopt to carry out our projects. What worked yesterday may not work today – effective management requires periodic episodes of de-maturation.



Thank you!