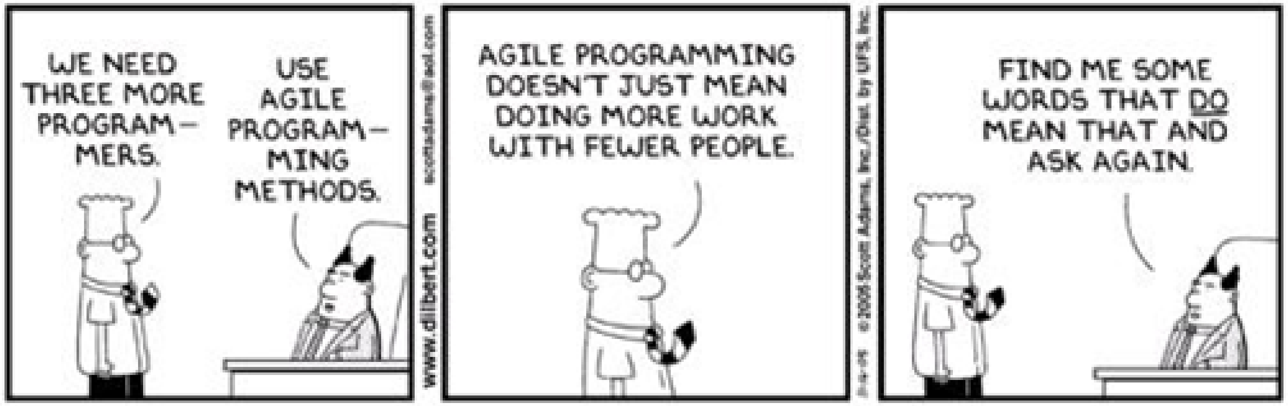
Agile Methods of Software Development

http://www-public.int-evry.fr/~gibson/Teaching/Agile/AgileMethods.pdf

AgileMethods.pptx

Dr J Paul Gibson, Dept. LOR, TSP, Evry, France


WE NEED THREE MORE PROGRAM-MERS.
USE AGILE PROGRAM-MING METHODS.
www.dilbert.com scottadams@aol.com
AGILE PROGRAMMING DOESN'T JUST MEAN DOING MORE WORK WITH FEWER PEOPLE.
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
FIND ME SOME WORDS THAT DO MEAN THAT AND ASK AGAIN.

© Scott Adams, Inc./Dist. by UFS, Inc.

Structure of the Presentation

1. Software Process and the Software Life Cycle

2. History of Agile: *More or less* a process?

3. Agile Fundamentals

4. Comparing Agile Methods

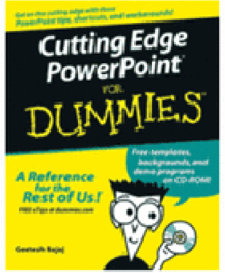
5. Agile Resources

6. Current Agile Research

Secondary Sources

I Source Texts/ References

Geetesh Bajaj. 2005. *Cutting Edge Powerpoint for Dummies*. For Dummies.



II Quotes

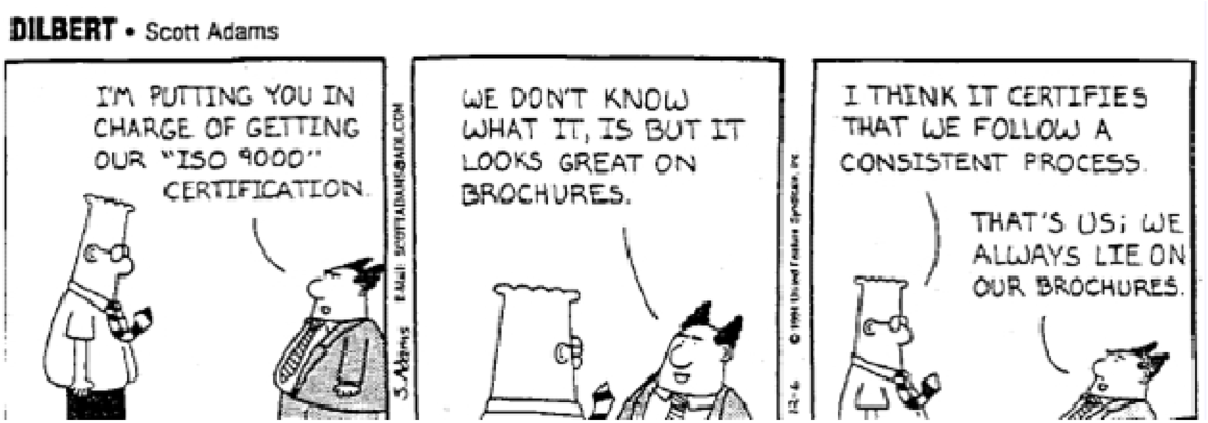
"PowerPoint makes us stupid."
Gen. James N. Mattis,
US Marine Corps,
http://www.nytimes.com/2010/04/27/world/27powerpoint.html

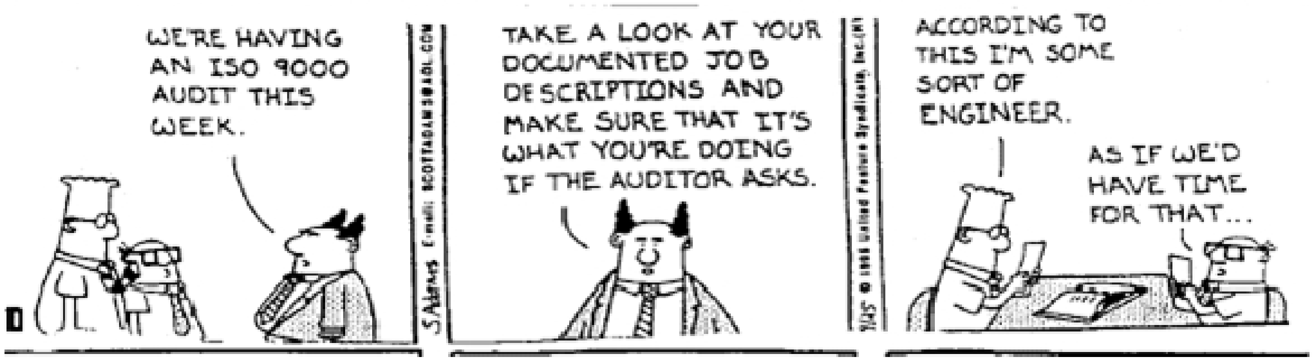
III Videos (youtube)

Boring Powerpoint (0:52)
http://www.youtube.com/watch?v=ZVFcagL1nsA

How important is a software development process?

"... the quality of the people on a project, and their organization and management, are much more important factors in the success than are the tools they use or the technical approaches they take."
Frederick P. Brooks, 1995, **The Mythical Man-Month: Essays on Software Engineering**


DILBERT • Scott Adams
I'M PUTTING YOU IN CHARGE OF GETTING OUR "ISO 9000" CERTIFICATION.
WE DON'T KNOW WHAT IT IS BUT IT LOOKS GREAT ON BROCHURES.
I THINK IT CERTIFIES THAT WE FOLLOW A CONSISTENT PROCESS.
THAT'S US; WE ALWAYS LIE ON OUR BROCHURES.


WE'RE HAVING AN ISO 9000 AUDIT THIS WEEK.
TAKE A LOOK AT YOUR DOCUMENTED JOB DESCRIPTIONS AND MAKE SURE THAT IT'S WHAT YOU'RE DOING IF THE AUDITOR ASKS.
ACCORDING TO THIS I'M SOME SORT OF ENGINEER.
AS IF WE'D HAVE TIME FOR THAT...

Why Do Software Projects Fail (Often) ?

Most often it is because of:

- A failure to properly manage the risks
- Building the wrong thing
- Being blinded by technology

Grady Booch. 1995. *Object Solutions: Managing the Object-Oriented Project*. Addison Wesley Longman Publishing Co.,

Adopting a good *software process* & *life cycle* will help address these failure modes.

Adopting a good *software process* & *life cycle* does not guarantee success.

We can never have a completely rational development process

Failure to Properly Manage The Risks

As projects progress, they often seem to lose their way:

- Unrealistic *schedules* and *plans* are drawn up
- No-one has the nerve to stand up and acknowledge *reality*
- Many problems are viewed as 'a simple *programming* matter', even when they are *process* or *architecture* concerns
- Project *direction* is set by the most *'stubborn'* participants because it is easier for *management* to let these people have their way.
- Free fall --- No one takes *responsibility* and everyone waits for the impact.
- Petty empires form … issues become *political*

Failure from Building the Wrong Thing

Projects can also lose their way because they go adrift in completely uncharted territory:

- There is no shared vision of the *problem* being *solved*.
- The (*development*) team is clueless as to the final destination
- No-one takes time to *validate* what is being built with *end-users* or *domain experts*
- Analysts *understand* the real *requirements*, but for a number of political/social reasons, this understanding never reaches the *designers/implementers*
- A false air of understanding pervades the *project*.
- Everyone will be shocked when *users* reject the delivered software.
- This is known as working in a vacuum.

Failure from Being Blinded By Technology

Don’t be blinded by the technology being used to build the software itself:

- *Tools* can break (be erroneous) … be ready for it
- Project *complexity* can grow exponentially … can your tools *scale* up accordingly?
- *Third-party suppliers of new technology* often do not deliver on promises (if at all)
- *Hardware* advances can out-run *software* development
- *Technology* can fuel changes to users’ expectations
- New languages/tools/methods are prone to premature adoption

Project Styles --- providing a focus --- moving towards a process

There are many different ways of balancing project characteristics. Certain styles are commonly seen in most industrial projects. These styles correspond to the drive towards a certain focus:

- Calendar-driven
- Requirements-driven
- Documentation-driven
- Architecture-driven
- Quality-driven

What is the Software Process?

A process is a systematic approach performed to achieve a specific purpose.

A software process is the set of activities, methods, practices, and transformations used to develop software and associated products that are released with it.

Software Process Capability is the range of expected results that are achievable by following the software process.

Software process performance is the actual result achieved in the development of software by following a software process.

Software Process Maturity is the extent to which a Software Process is defined, managed, controlled, measured and effective.

No process is perfect

Even the most successful projects seem to take longer, involve more effort, and require more *crisis management* than we really believe they ever should. We must never rely on the process pulling a project through. The process can never be completely rational:

Parnas, D. and Clements, P. 1986. A Rational Design Process: How and why to Fake It. *IEEE Transactions on Software Engineering vol. SE-12(2) p251*

- Users typically don’t know what they want
- Users typically can’t express what they want
- Requirements are incomplete and/or change
- Implementation architectures change
- We all bring intellectual/technological baggage to projects
- Systems built by humans are always subject to human error
- Fundamental limits to the amount of complexity which can be handled

What is the Software Life Cycle?

The software life cycle is the collection of phases through which a software product passes from initial conception through to retirement from service.

- Every software product has a life cycle.
- Life cycles used to be typically quite long—some software products have been "alive" for 30 years.
- Life cycles are shortening due to technological advances

Life Cycle Phases - Implicitly or explicitly, all software products go through at least the following phases:

- Requirements—determine customer needs and product constraints
- Design—determine the structure/organisation of the software system
- Coding—write the software
- Testing—exercise the system to find and remove defects
- Maintenance—correct and enhance product after customer deployment

Software Life Cycle Models

A process is a collection of activities, with well-defined inputs and outputs, for accomplishing some task.

A life cycle model is a description of a process for carrying a software product through all or part of its life cycle.

- Life cycle models tend to focus on major life cycle phases and their relationships to one another.
- Recent work on software processes has examined many aspects of development and maintenance in great detail.
- A life cycle model is a software process description, but the term life cycle model predates recent discussions of software processes.

Life Cycle Models and the Software Process

The core of any software project is the coding ---architecture, abstraction, implementation

Life cycle models revolve around this core --- how does the software evolve as the project progresses?

All life-cycle models are based on the simple idea of feedback --- synthesis and analysis are mutually defined and recursively interdependent.

The differences between the life-cycle models lie in the ways in which the feedback is organised, for example:

- Trial and error
- Exploratory Programming
- The Waterfall Model
- Iterative Feedback Model
- Prototyping
- Test-Driven
- Design/Model-Driven
- Agile

Trial and Error : “Hacking”

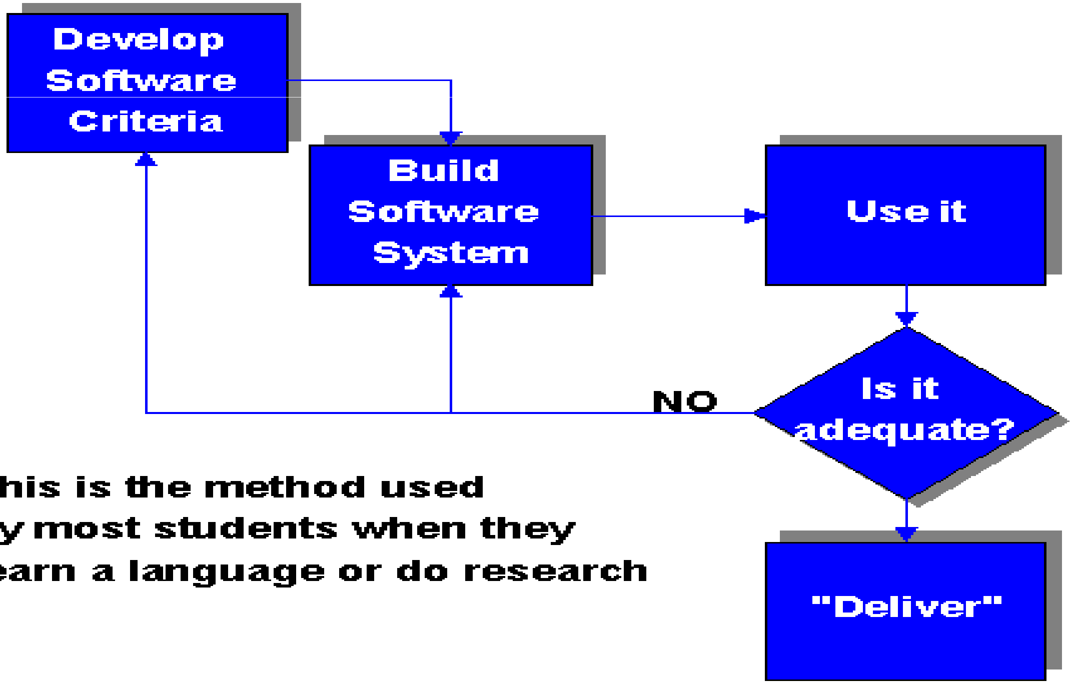
- The most primitive life cycle model is trial and error, sometimes called build-and-fix or hack-and-foist
- In this life cycle model, the first version of the system is built without planning, documentation, or control
- If the product is accepted, the developers face an interminable period of confusion, frustration, and drudgery fixing an endless stream of problems

The feedback can be very primitive --- will we accept the first and only version of the system (yes/no)

Exploratory Programming

A bit better than trial-and-error (but not much):

- it establishes feedback before delivery to customer
- it allows multiple feedback
- it separates specification from implementation



This is the method used by most students when they learn a language or do research

The Waterfall Model (dominated the 70's)

The waterfall model is the oldest life cycle model; is was proposed by Winston Royce in 1970.

This model is called a waterfall because it is usually drawn with a cascade of activities through the phases of the life cycle "downhill" from left to right:

- analysis, requirements, specification, design, implementation, testing, maintenance

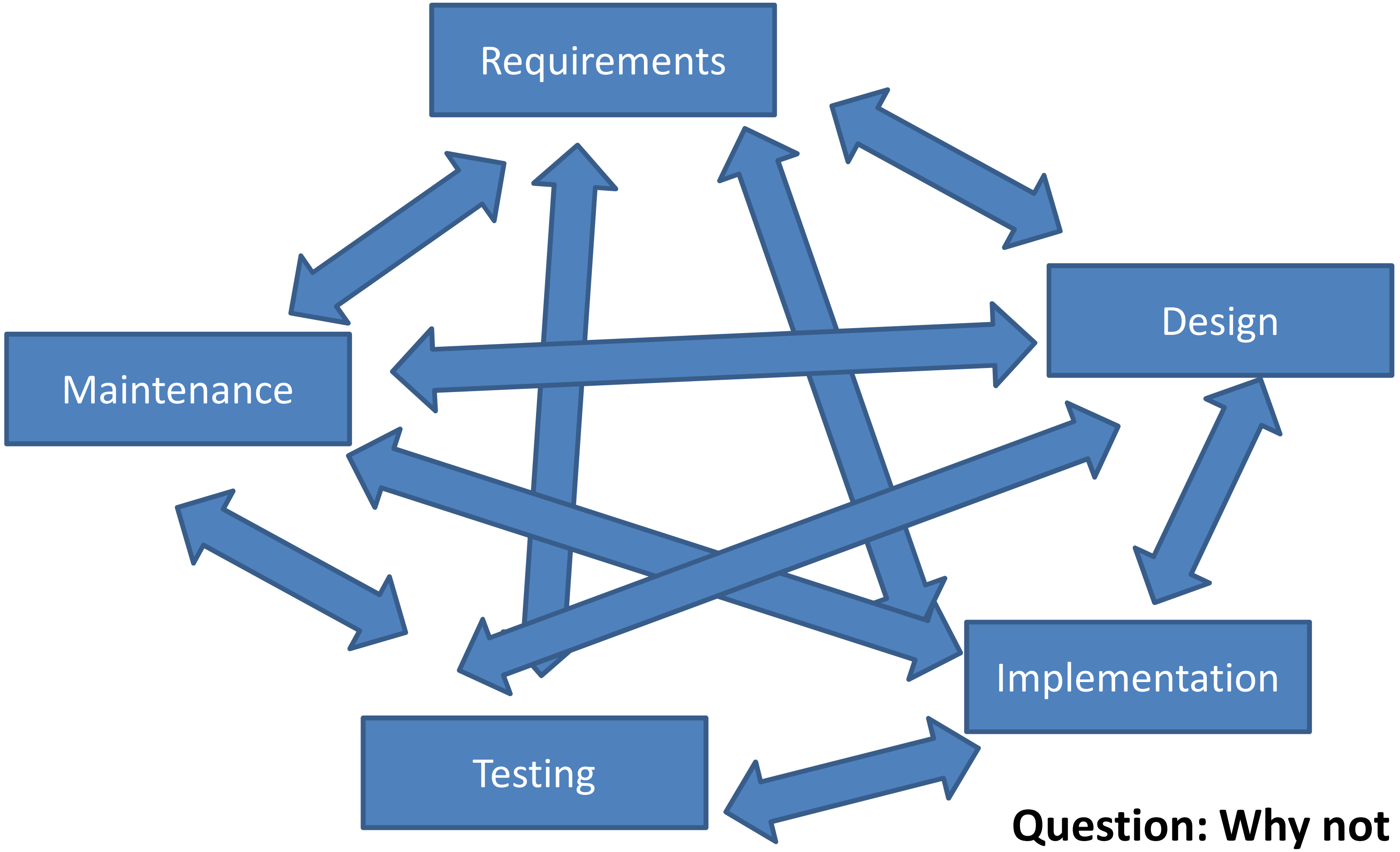
There are many versions of the waterfall model:

- the phases/activities can be structured to different levels of detail
- the feedback can be more or less flexible

W. W. Royce. 1987. *Managing the development of large software systems: concepts and techniques*. In Proceedings of the 9th international conference on Software Engineering (ICSE '87). IEEE Computer Society Press, 328-338.

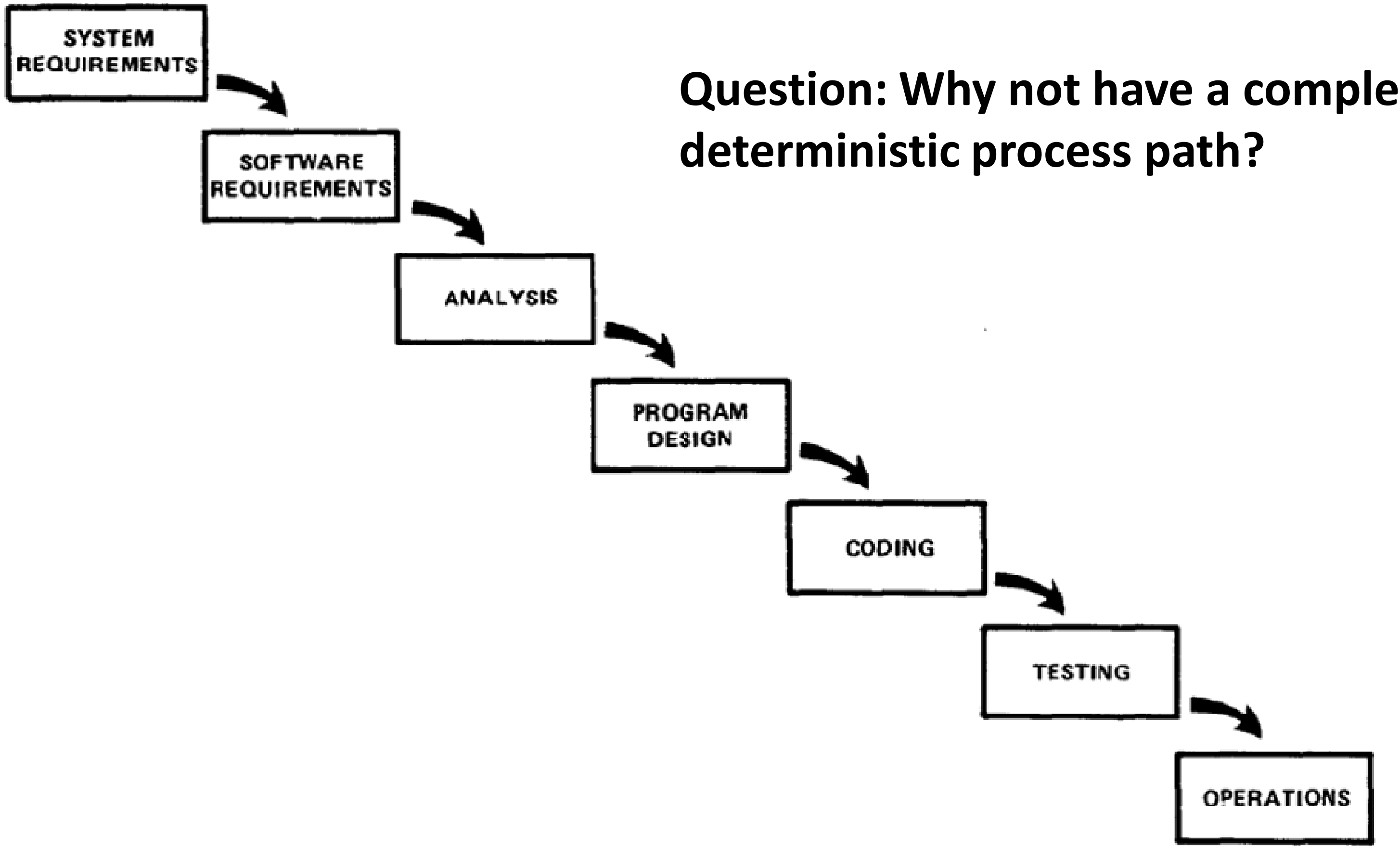
(Reprinted from *Proceedings, IEEE WESCON, August 1970, pages 1-9.)*

Life Cycle Chaos - A Complete *Feedback* Graph Between Activities



Question: Why not just reduce the number of activities to manage the complexity?

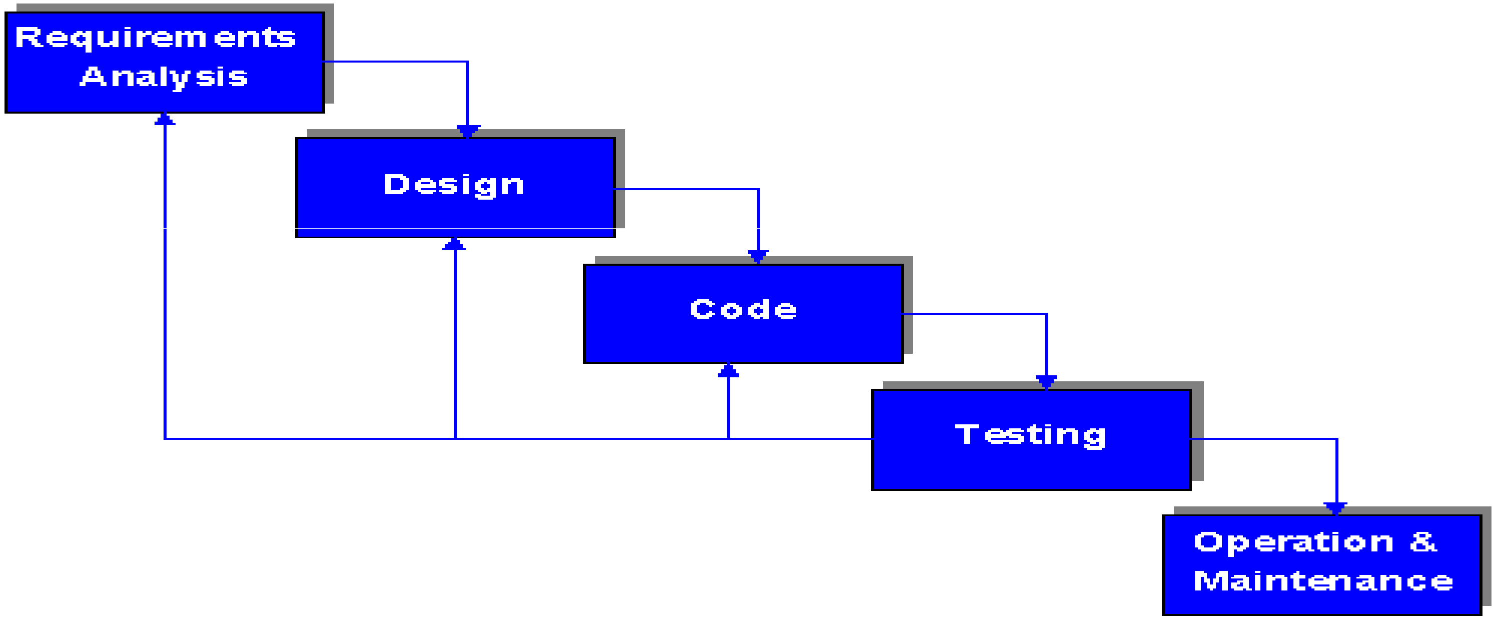
Life Cycle Ideal - (Strict) Waterfall With No Feedback



Question: Why not have a completely deterministic process path?

Non-strict Waterfall Model

Although the waterfall model stresses a linear sequence of phases, in fact there is in practice always an enormous amount of iteration back to earlier phases, a point made by the arrows leading back up the waterfall, in the following diagram.

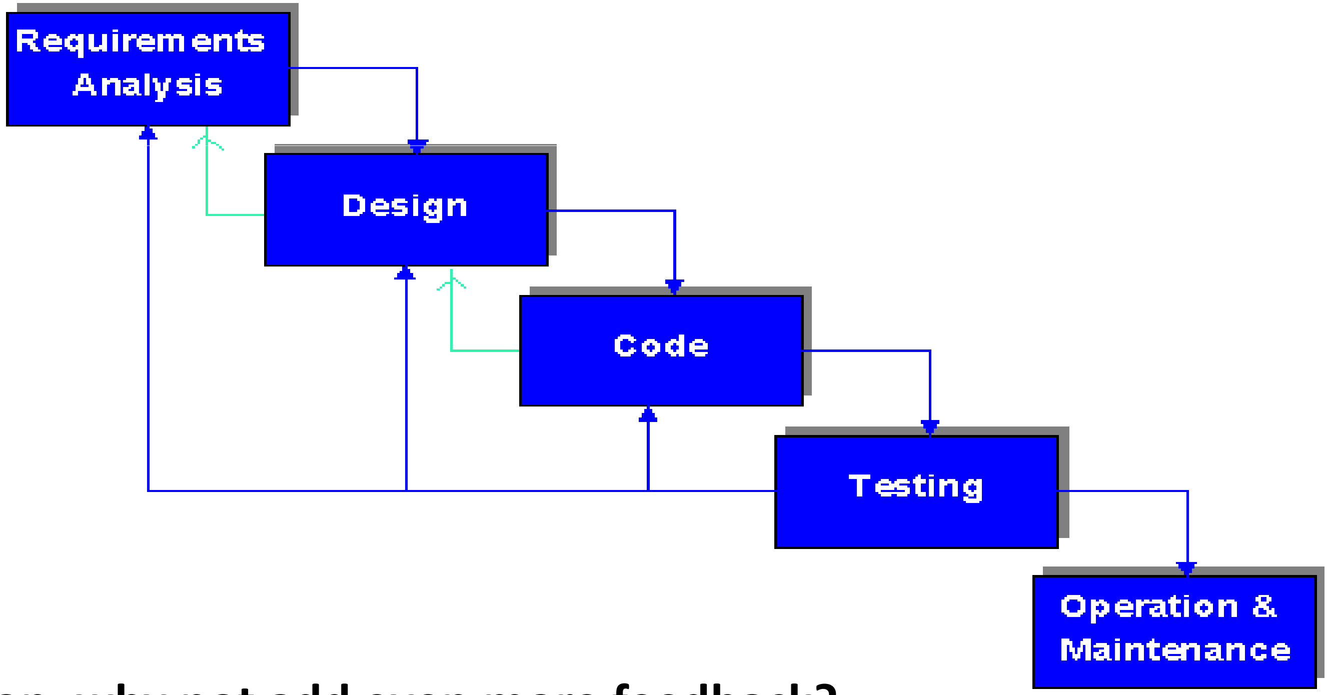


Note: In this variation, feedback is only from **testing** phase to *any* previous stage

Iterative Feedback Model

Like the non-strict waterfall method except that feedback is allowed from any phase to the previous phase.

Note that we can still jump anywhere from testing!



Question: why not add even more feedback?

Analysis of waterfall method

Strengths:

- Emphasises completion of one phase before moving on
- Emphasises early planning, customer input, and design
- Emphasises testing as an integral part of the life cycle
- Provides quality gates at each life cycle phase

Weaknesses:

- Depends on capturing and freezing requirements early in the life cycle
- Depends on separating requirements from design
- Not politically feasible in some organisations
- Emphasises products rather than processes

Away From Waterfall

Daniel D. McCracken and Michael A. Jackson. 1982. Life cycle concept considered harmful. *SIGSOFT Softw. Eng. Notes* 7, 2 (April 1982), 29-32.

“Any form of life cycle is a project management structure imposed on system development. To contend that any life cycle scheme, even with variations, can be applied to all system development is either to fly in the face of reality or to assume a life cycle so rudimentary as to be vacuous."

"The life cycle concept is simply unsuited to the needs of the 1980‘s in developing systems."

Prototyping Models (become popular in the 80's)

A prototype is a working model of (part of) a final system

Prototyping is becoming more popular all the time, and people often refer to prototypes in the literature.

Unfortunately, a variety of terminology is used, so it is often difficult to tell what is meant when people discuss prototyping.

Note there are different types of prototyping model, with different characteristics, eg:

- Rapid Prototyping
- Evolutionary Prototyping
- Operational Prototyping

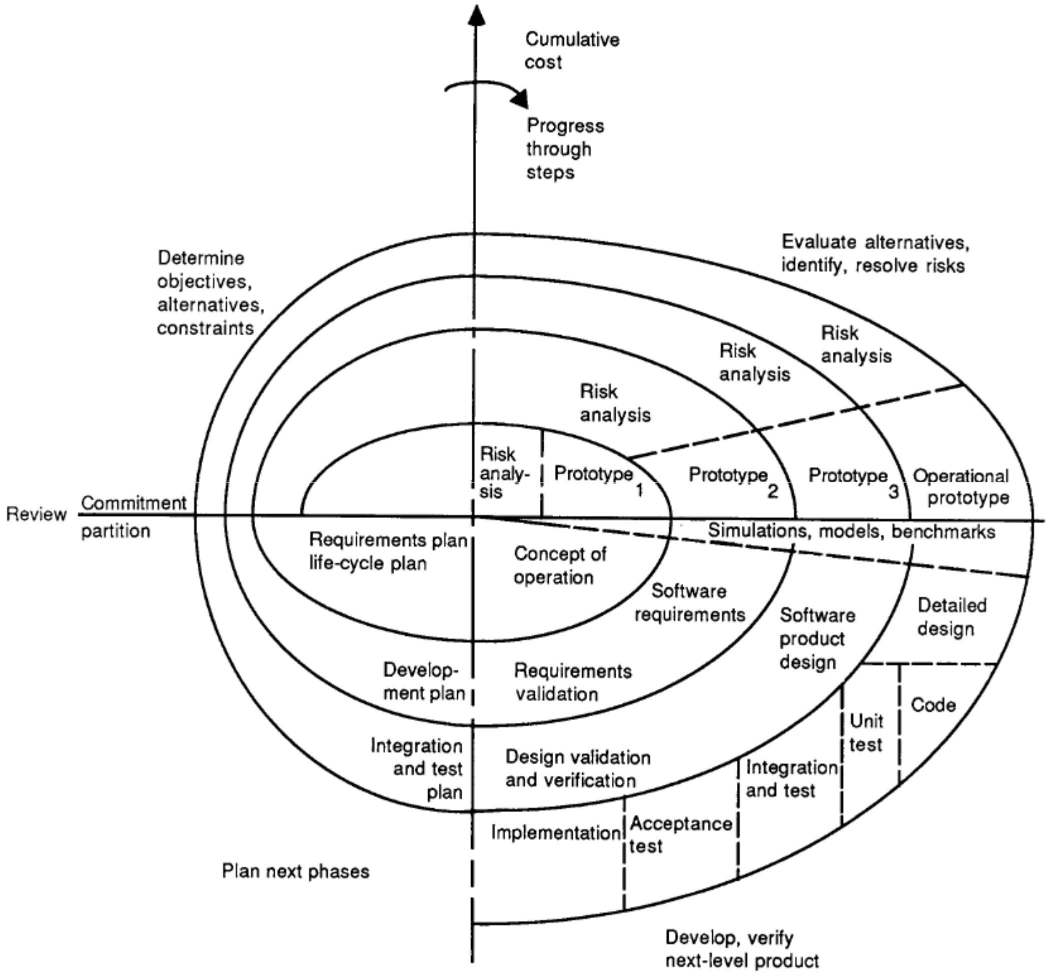
Maryam Alavi. 1984. An assessment of the prototyping approach to information systems development. *Commun. ACM* 27, 6 (June 1984), 556-563.

R. N. Burns and A. R. Dennis. 1985. Selecting the appropriate application development methodology. *SIGMIS Database* 17, 1 (September 1985), 19-23

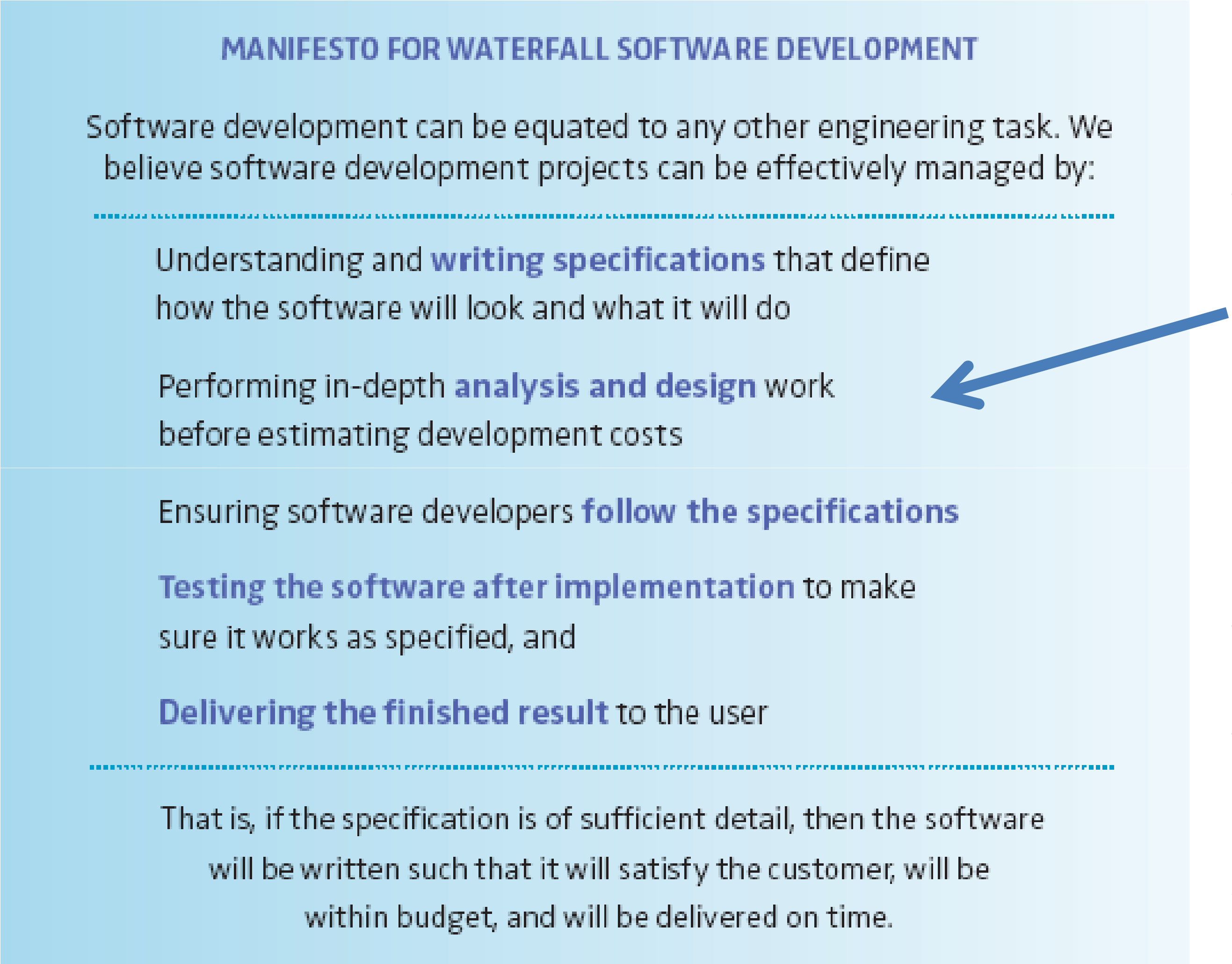
From Waterfall to Spiral (dominates the 90's)

B. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer,* May 1988, pp. 61-72.

"Stop the life cycle—I want to get off!"
"Life-cycle Concept Considered Harmful."
"The waterfall model is dead."
"No, it isn't, but it should be."



Agile: A lightweight alternative to *heavy-weight waterfall-like* processes?



MANIFESTO FOR WATERFALL SOFTWARE DEVELOPMENT

Software development can be equated to any other engineering task. We believe software development projects can be effectively managed by:

Understanding and **writing specifications** that define how the software will look and what it will do

Performing in-depth **analysis and design** work before estimating development costs

Ensuring software developers **follow the specifications**

Testing the software after implementation to make sure it works as specified, and

Delivering the finished result to the user

That is, if the specification is of sufficient detail, then the software will be written such that it will satisfy the customer, will be within budget, and will be delivered on time.

Claim:
With heavyweight processes, there is too much emphasis on analysis and design documentation.

Agile is lightweight. It combines prototyping and test-based processes: where there is continual development of code and continual validation of customer requirements

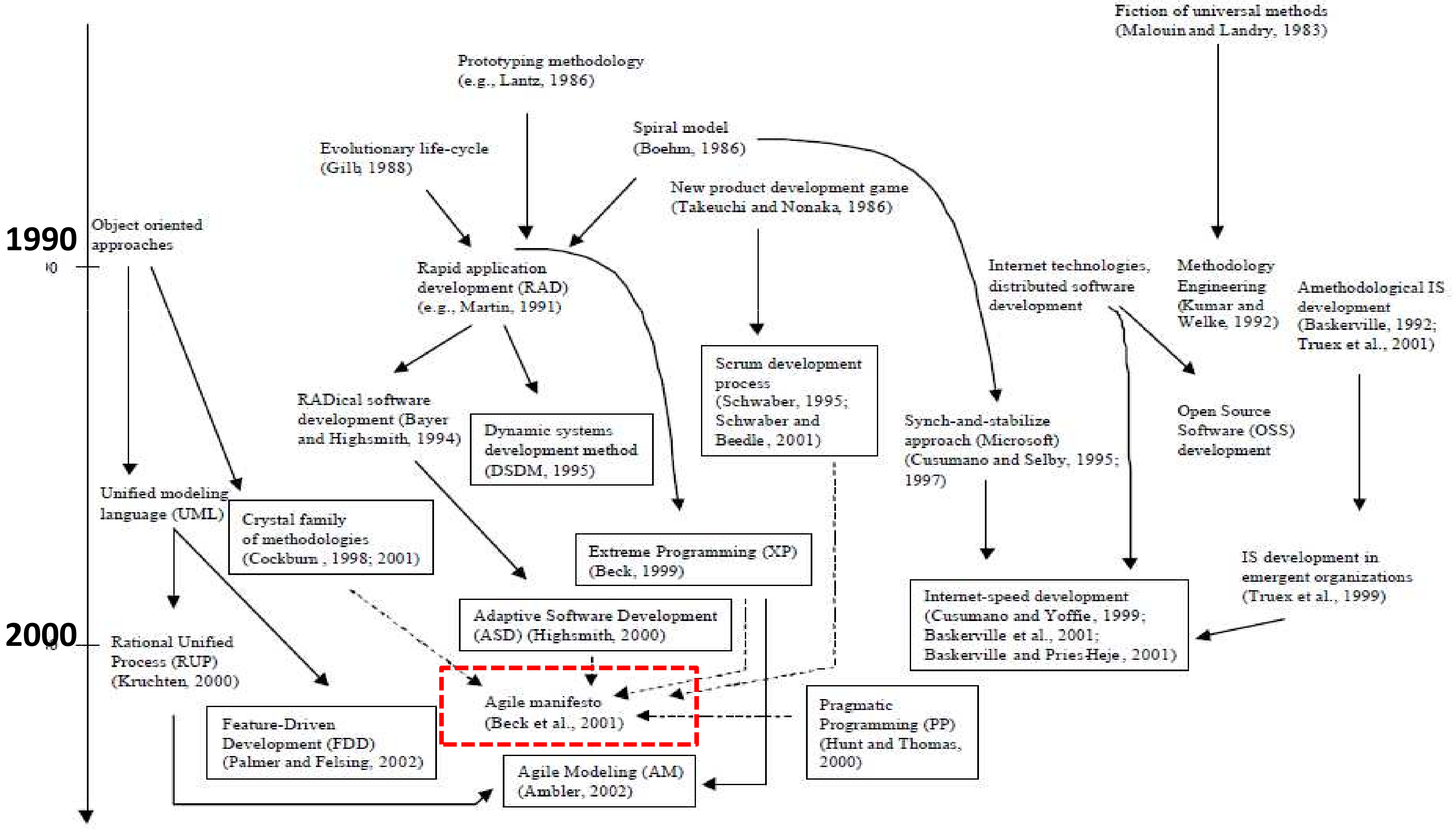
The History of Agile Methods: what can we learn?

© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com
HARDIN
search ID: pha0139
"Of course things change: Our fathers were replaced by machines— We'll be replaced with new software."

© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com
A.BACALL
search ID: aba0350
"Due to recent technological advances, everything I taught you about computers is no longer valid."

History of Agile: Where Did it Come From?

Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. 2003. New directions on agile methods: a comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering* (ICSE '03). IEEE Computer Society, Washington, DC, USA, 244-254.



1990

2000

Fiction of universal methods (Malouin and Landry, 1983)

Prototyping methodology (e.g., Lantz, 1986)

Spiral model (Boehm, 1986)

Evolutionary life-cycle (Gilb 1988)

New product development game (Takeuchi and Nonaka, 1986)

Object oriented approaches

Rapid application development (RAD) (e.g., Martin, 1991)

Internet technologies, distributed software development

Methodology Engineering (Kumar and Welke, 1992)

Amethodological IS development (Baskerville, 1992; Truex et al., 2001)

Scrum development process (Schwaber, 1995; Schwaber and Beedle, 2001)

RADical software development (Bayer and Highsmith, 1994)

Dynamic systems development method (DSDM, 1995)

Synch-and-stabilize approach (Microsoft) (Cusumano and Selby, 1995; 1997)

Open Source Software (OSS) development

Unified modeling language (UML)

Crystal family of methodologies (Cockburn , 1998; 2001)

Extreme Programming (XP) (Beck, 1999)

IS development in emergent organizations (Truex et al., 1999)

Adaptive Software Development (ASD) (Highsmith, 2000)

Internet-speed development (Cusumano and Yoffie, 1999; Baskerville et al., 2001; Baskerville and Pries-Heje, 2001)

Rational Unified Process (RUP) (Kruchten, 2000)

Agile manifesto (Beck et al., 2001)

Pragmatic Programming (PP) (Hunt and Thomas, 2000)

Feature-Driven Development (FDD) (Palmer and Felsing, 2002)

Agile Modeling (AM) (Ambler, 2002)

History of Agile

Agile software development isn't a set of tools or a single methodology, but a philosophy put to paper in 2001 with an initial 17 signatories.

Agile was a significant departure from the heavyweight document-driven software development methodologies—such as waterfall—in general use at the time.

While the publication of the "Manifesto for Agile Software Development" didn't start the move to agile methods, which had been going on for some time, it did signal industry acceptance of agile philosophy.

"*Many people may think that agile is just another software development process. Although that is true to a degree, there is a lot more to agile than just a process or just a set of practices. Agile (or agility) is more of a mindset - a way of thnking about software development.*"
Greg Smith, Ahmed Sidky, 2009, Becoming Agile: ...in an imperfect world

Manifesto for Agile Software Development

http://agilemanifesto.org/

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have 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*

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

Kent Beck:
•*Industrial experience with design patterns*. ICSE 1996, IEEE Computer Society.
•*Embracing Change with Extreme Programming*, 1999, IEEE Computer.
•*Test Driven Development: By Example*. 2002 Addison-Wesley.

Mike Beedle & Ken Schwaber:
•*Agile Software Development with Scrum* , 2001, Prentice Hall.

Arie van Bennekum

Alistair Cockburn :
•*Writing Effective Use-Cases*, 2000, Addison-Wesley.
•*The Costs and Benefits of Pair Programming,* In *Extreme programming examined*, Giancarlo Succi and Michele Marchesi (Eds.)., 2001, Addison-Wesley Longman Publishing

Ward Cunningham & KB:
A laboratory for teaching object oriented thinking, *SIGPLAN Not.* 24, 10 (September 1989)

Martin Fowler
•*Refactoring: Improving the Design of Existing Code*, 1999, Addison-Wesley.

James Grenning:

•Launching Extreme Programming at a Process-Intensive Company, 2001, *IEEE Softw.* 18

Jim Highsmith and Alistair Cockburn:

•*Agile Software Development: The Business of Innovation*, 2001, *Computer* 34,

Andrew Hunt

Ron Jeffries and Grigori Melnik:

•*TDD--The Art of Fearless Programming*, 2007, *IEEE Softw.* 24, 3

Jon Kern

Brian Marick:

•*The Craft of Software Testing: Subsystem Testing Including Object-Based and Object-Oriented Testing*. Prentice-Hall, 1994.

•*When Should a Test Be Automated?*, 1998.

Robert C. Martin:

- *SoapBox - eXtreme Programming Development through Dialog,* IEEE Software 17, 2000
- *Professionalism and Test-Driven Development. IEEE Software 24(3), 2007*

Steve Mellor:

- *An object-oriented approach to domain analysis*. *SIGSOFT Softw. Eng. Notes* 14, 5, 1989
- *Make models be assets. Commun. ACM 45(11): 76-78, 2002.*

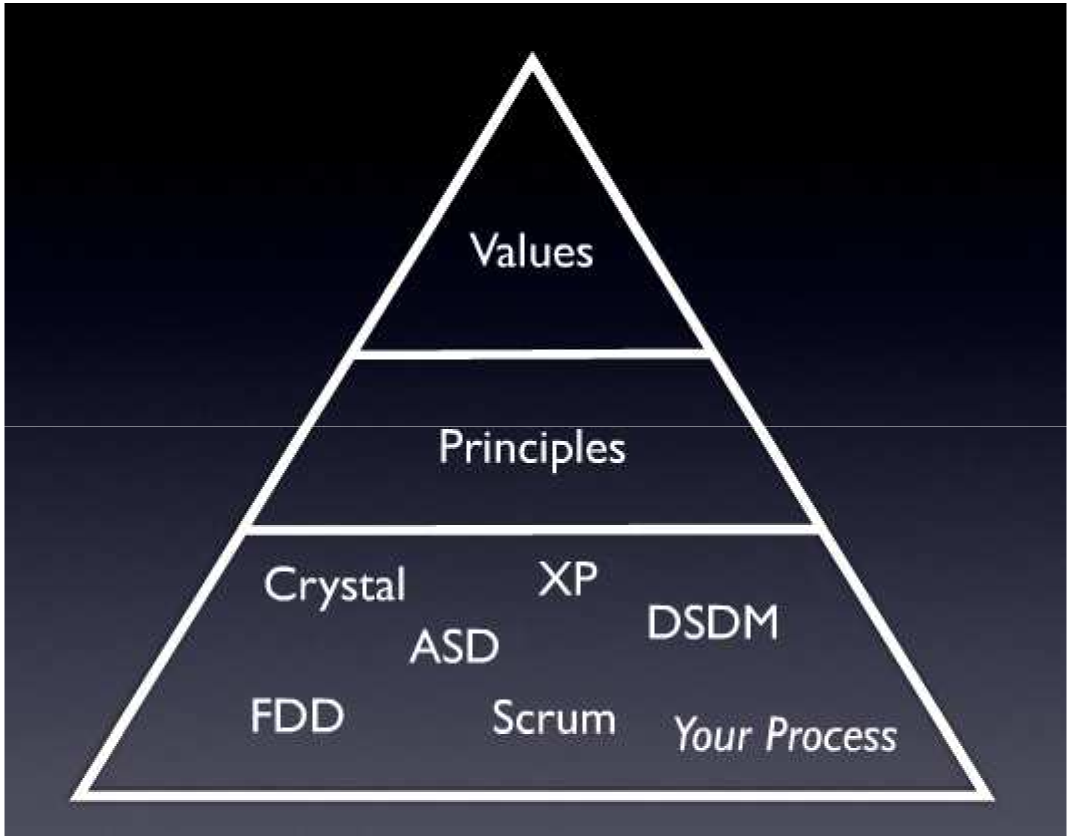
Jeff Sutherland:

- *Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies*, Cutter IT Journal, 2001
- Agile development: Lessons learned from the first scrum, Cutter IT Journal, 2004
- *Future of Scrum: Parallel Pipelining of Sprints in Complex Projects*. AGILE 2005: 90-102

Dave Thomas:

- *Orwell: a configuration management system for team programming*. In OOPSLA'88.
- *Model driven development: the case for domain oriented programming*. In OOPLSA'03.
- *MDA: Revenge of the Modelers or UML Utopia?, IEEE Software,* 21(3) , 2004
- *Agile Programming: Design to Accommodate Change. IEEE Software* 22(3), 2005

Agile: Values, Principles and Methods



Agile: 12 Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agile: 12 Principles

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity--the art of maximizing the amount of work not done--is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Fundamentals: Agile vs Waterfall

Waterfall features distinct phases with checkpoints and deliverables, while agile methods have iterations where the output of each agile iteration is working code that can be used to evaluate and respond to changing and evolving user requirements.

Waterfall assumes that it is possible to have good understanding of the requirements from the start. But in software development, stakeholders often don't know what they want and can't articulate their requirements. With waterfall, development rarely delivers what the customer wants even if it is what the customer asked for.

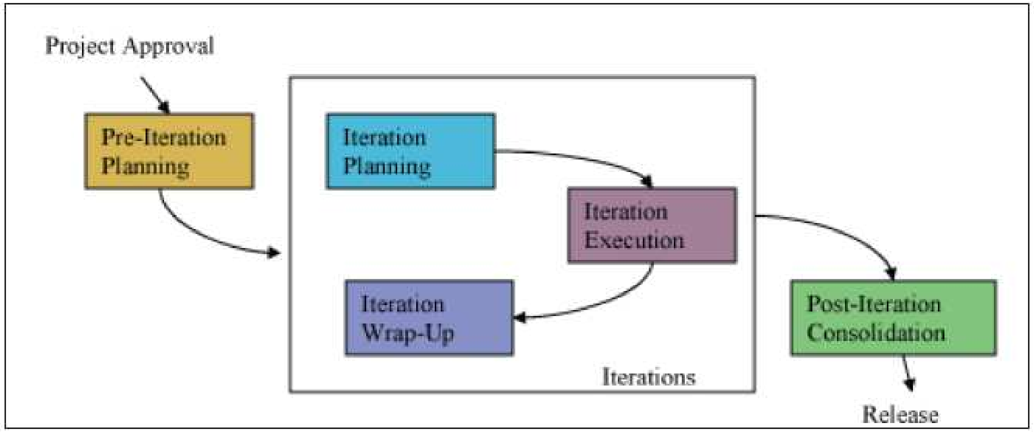
With Agile emphasis is placed on the customer and their requirements.

Fundamentals: iterations count

Agile methodologies embrace iterations.

Small teams work together with stakeholders to define quick prototypes, proof of concepts, or other visual means to describe the problem to be solved. The team defines the requirements for the iteration, develops the code, and defines and runs integrated test scripts, and the users verify the results.

Verification occurs much earlier in the development process.



Suitability of Agile Methods

There is little agreement on what types of software projects are best suited for the agile approach.

Many large organizations have difficulty moving from the traditional waterfall method to an agile one.

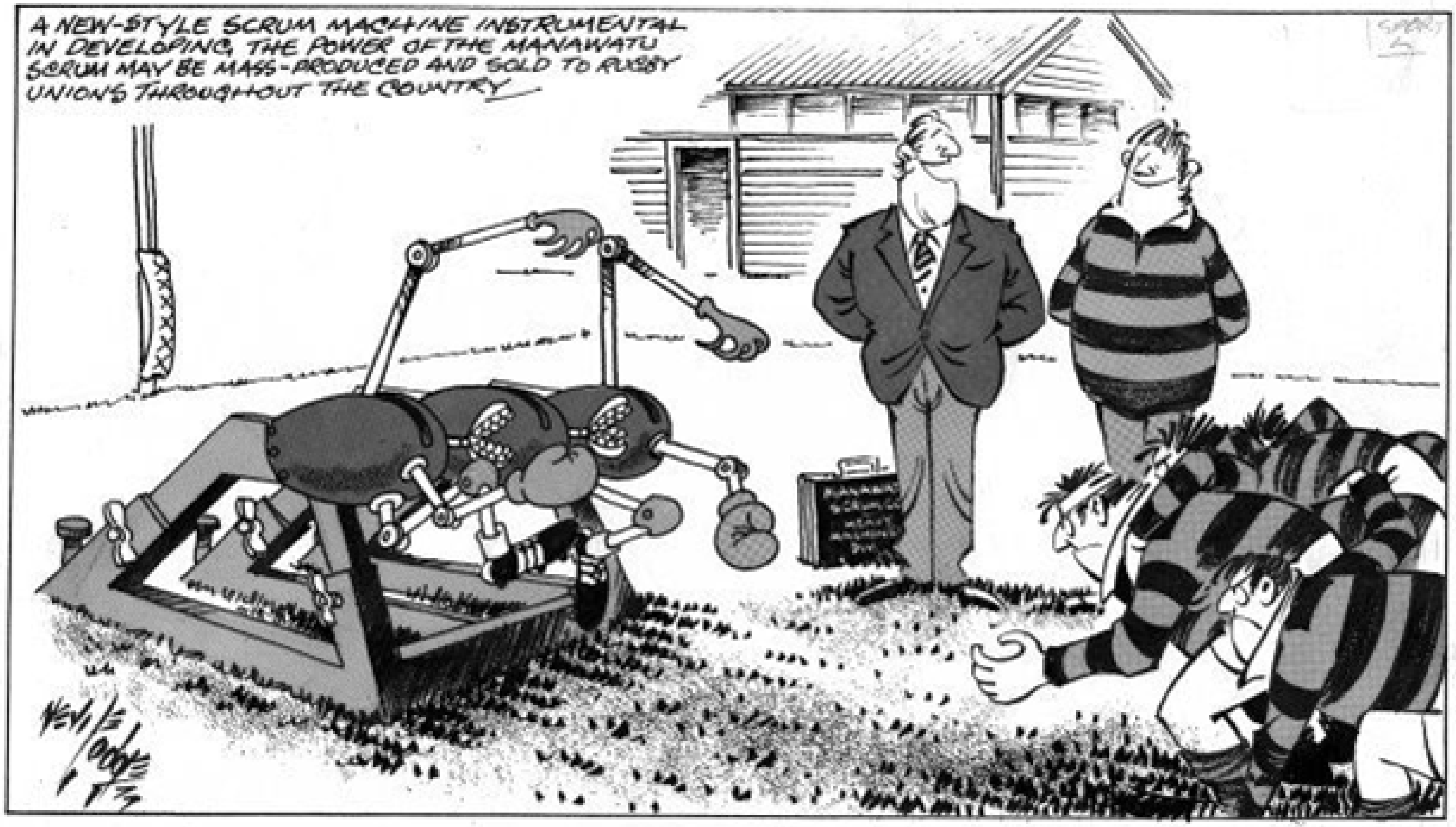
When Agile is risky:

- Large scale development (>20 developers)
- Distributed development (non-co-located teams)
- *Control-freak* companies
- Unreliable customer/client contact
- Forcing an agile process on a development team
- Inexperienced developers

Some of the most popular Agile methods:

- Scrum
- Lean Development
- Extreme programming (XP)
- Adaptive Software Development (ASD)
- Agile Modeling
- Crystal Methods
- Dynamic System Development Methodology (DSDM)
- Feature Driven Development

Scrum


A NEW-STYLE SCRUM MACHINE INSTRUMENTAL IN DEVELOPING THE POWER OF THE MANAWATU SCRUM MAY BE MASS-PRODUCED AND SOLD TO RUGBY UNION'S THROUGHOUT THE COUNTRY

"THIS IS OUR IMPROVED MARK II MODEL — GIVES FULL SCRUM SIMULATION - KNEES UP, BITING, PUNCHING, JERSEY-PULLING — THE LOT"

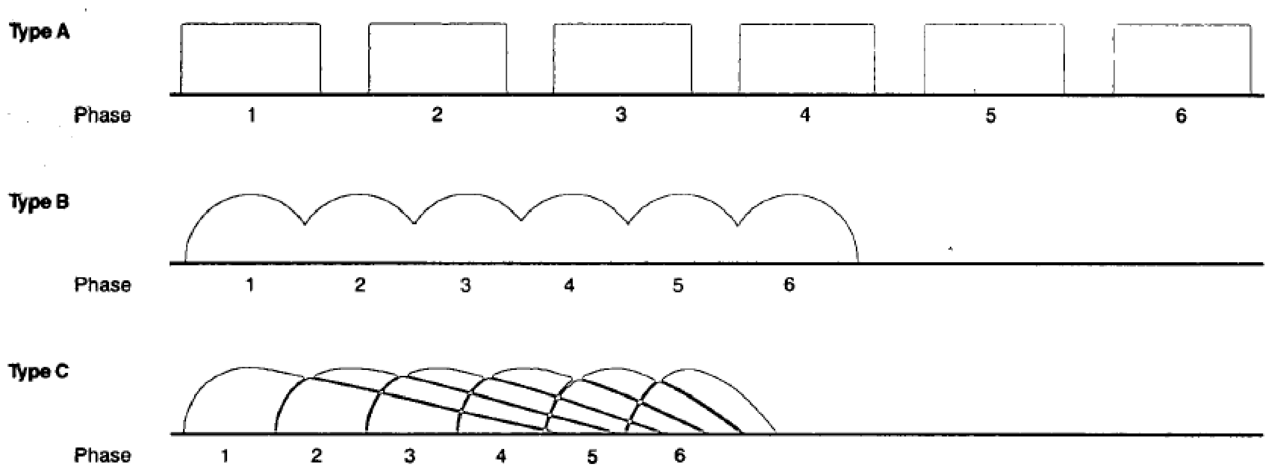
Scrum

In rugby, 'scrum' (related to "scrimmage") is the term for a huddled mass of players engaged with each other to get a job done. In software development, the job is to put out a release.

Comes from Japan and based from industrial process control theory:

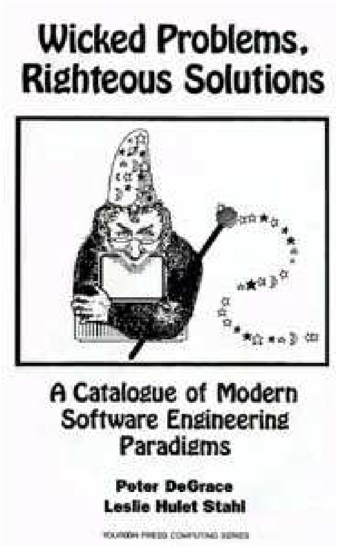
Takeuchi, Hirotaka and Nonaka, Ikujiro: *The New New Product Development Game*, Harvard Business Review, 1986.

"Stop running the relayy race and take up rugby"



Speed Up development

Scrums for wicked problems


Wicked Problems, Righteous Solutions
A Catalogue of Modern Software Engineering Paradigms
Peter DeGrace
Leslie Hulet Stahl
YOURDON PRESS COMPUTING SERIES

"the best adaptive project management approach for wicked software development projects"

Peter DeGrace and Leslie Hulet Stahl. *Wicked Problems, Righteous Solutions*. 1990. Yourdon Press

Many of the systems problems facing software developers have all the characteristics of wicked problems:

1. **There is no definitive formulation of a wicked problem.**
2. **Wicked problems have no stopping rule.**
3. **Solutions to wicked problems are not true-or-false but good-or-bad**
4. **There is no immediate and no ultimate test of a solution to a wicked problem.**
5. **Every implemented solution to a wicked problem has consequences.**
6. **Wicked problems do not have a well-described set of potential solutions.**
7. **Every wicked problem is essentially unique.**
8. **Every wicked problem can be considered a symptom of another problem.**
9. **The causes of a wicked problem can be explained in numerous ways**.
10. **The planner (designer) has no right to be wrong.**

Scrum: early research

Ken Schwaber, *SCRUM Development Process*, OOPSLA Business Object Design and Implementation Workshop, 1997

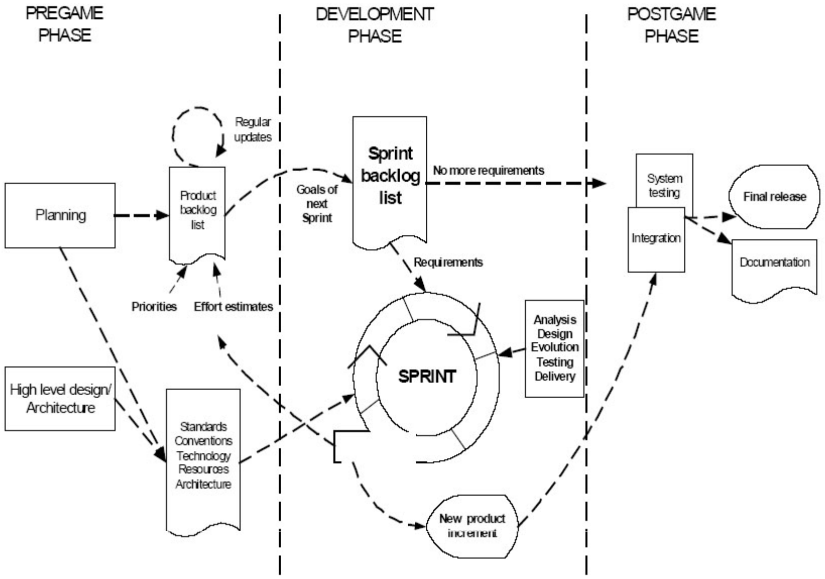
M. Beedle et al., *SCRUM: An Extension Pattern Language for Hyperproductive Software Development*, Pattern Languages of Program Design 4, N. Harrison, B. Foote, and H. Rohnert, eds., Addison-Wesley, 2000,

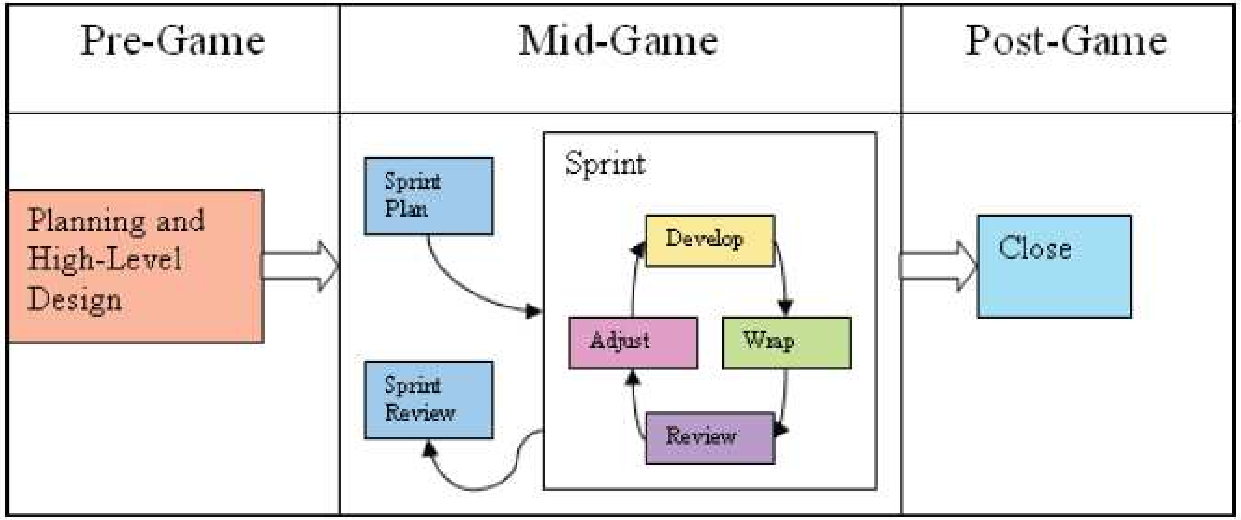
Linda Rising and Norman S. Janoff. *The Scrum Software Development Process for Small Teams*. *IEEE Softw.* 17, 4. 2000.

Jeff Sutherland, *Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies*, Cutter IT Journal, 2001

Mike Beedle & Ken Schwaber, *Agile Software Development with Scrum* , 2001, Prentice Hall.

Scrum Phases





Main scrum concepts:

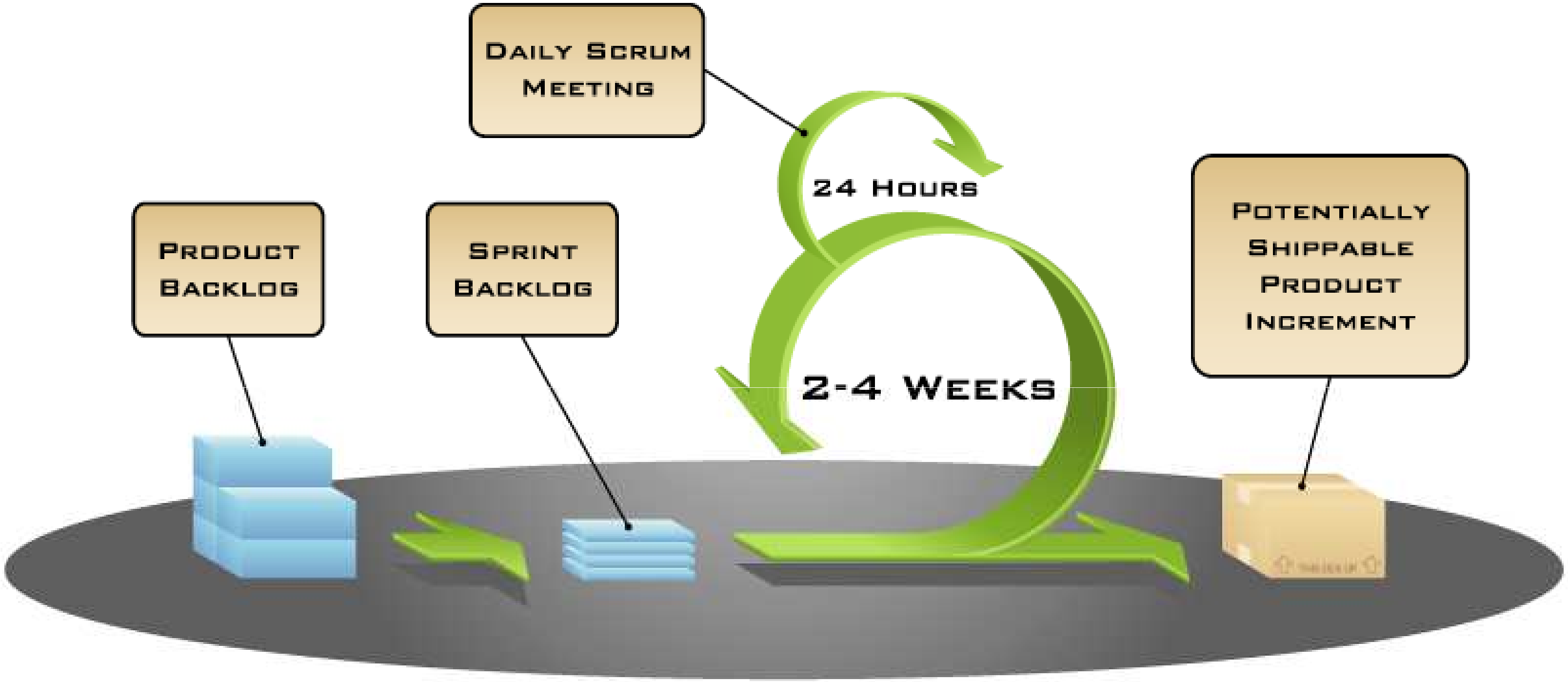
Burndown chart. This chart, updated every day, shows the work remaining within the sprint. The burndown chart is used both to track sprint progress and to decide when items must be removed from the sprint backlog and deferred to the next sprint.

Product backlog. Product backlog is the complete list of requirements—including bugs, enhancement requests, and usability and performance improvements—that are not currently in the product release.

ScrumMaster. The ScrumMaster is the person responsible for managing the Scrum project. Sometimes it refers to a person who has become certified as a ScrumMaster by taking ScrumMaster training.

Sprint backlog. Sprint backlog is the list of backlog items assigned to a sprint, but not yet completed. In common practice, no sprint backlog item should take more than two days to complete. The sprint backlog helps the team predict the level of effort required to complete a sprint.

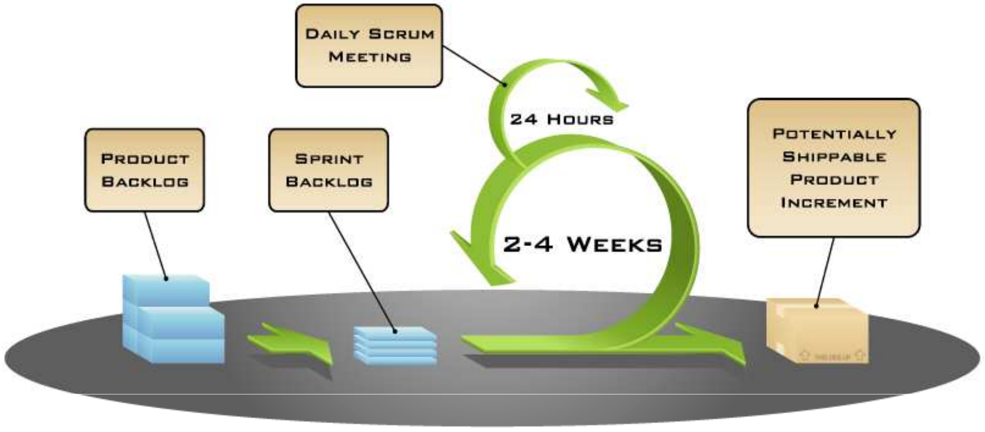
Scrum Meetings



Copyright © 2005, Mountain Goat Software

The backlog is key: it is populated during the planning phase of a release and defines the scope of the release

Scrum Meetings



Copyright © 2005, Mountain Goat Software

The development process is divided into a series of short iterations called sprints. Before each sprint, the team members identify the backlog items for the sprint. At the end of a sprint, the team reviews the sprint to articulate lessons learned and check progress.

During a sprint, the team has a daily meeting called a scrum. Each team member describes the work to be done that day, progress from the day before, and any blocks that must be cleared. To keep the meetings short, the scrum is supposed to be conducted with everyone in the same room—standing up for the whole meeting.

When enough of the backlog has been implemented so that the end users believe the release is worth putting into production, management closes development. The team then performs integration testing, training, and documentation as necessary for product release.

Scrum Meetings

We stand up to keep the meeting short

The daily stand-up meeting (also known as a "daily scrum", a "daily huddle", "morning roll-call", etc.) is simple to describe:

The whole team meets every day for a quick status update. We stand up to keep the meeting short.

But this short definition does not really tell you the subtle details that distinguish an effective stand-up from a waste of time. So how can you tell?

For experienced practitioners, when things go wrong with the stand-up, they will instinctively know what to adjust to fix the situation.

For novices, when things go wrong, it is much less likely that they'll figure out what to do... and it’s much more likely that, given no assistance, they will simply abandon the practice altogether.

This would be unfortunate since well-run stand-ups add significant value to teams.

Scrum Meetings

The goals of the daily stand-up are GIFTS

There are several goals for a daily stand-up meeting:

- To help start the day well
- To support improvement
- To reinforce focus on the right things: the baton not the runners
- To reinforce the sense of team
- To communicate what is going on

As a mnemonic device, think of **GIFTS**:
Good Start, **I**mprovement, **F**ocus, **T**eam, **S**tatus

The purpose is not to meet... it is to improve.
-- Joe Ely, "More on Daily Start-Up Meetings"

Scrum Meetings

Who attends?.... All Hands

People and representatives from various areas wish to know about and/or contribute to the status and progress of the project. Communicating status in multiple meetings and reports requires a lot of duplicate effort.

Therefore

Replace some or all of the meetings and reports with the daily stand-up. Anyone who is directly involved in or wants to know about the day-to-day operation of the project should attend the single daily stand-up meeting.

But

People not directly involved can disrupt the stand-up if they are unclear about what is expected behaviour. This may be addressed by simply informing new participants and observers of expected norms beforehand.

In The Perfect World, *Work Items Attend*

Also Known As: Story-focused stand-up - if the stories are so important to the project, **they** ought to be the ones speaking in the standup

Scrum Meetings

What do we talk about?

Yesterday Today Obstacles

Also Known As: Three Questions

Some people are talkative and tend to wander off into **Story Telling**. Some people want to engage in **Problem Solving** immediately after hearing a problem. Meetings that take too long tend to have low energy and participants not directly related to a long discussion will tend to be distracted.

Therefore

Structure the contributions using the following format:

- What did I accomplish yesterday?
- What will I do today?
- What obstacles are impeding my progress?

Scrum Meetings

Walk the (Improvement) Board.
Also Known As: Blockage Board, Impediment Board,

Obstacles raised in the stand-up are not removed or otherwise addressed in a timely fashion.
Therefore
Post raised obstacles to an **Improvement Board**. This is a publicly visible whiteboard or chart that identifies raised obstacles and tracks the progress of their resolution. An **Improvement Board** can be updated outside of stand-ups and serves as a more immediate and perhaps less confronting way to initially raise obstacles.

The simple act of writing an issue down and therefore explicitly acknowledging it is a very reliable way to reduce drawn out conversations. So even if not everyone agrees that any particular item is an obstacle, it is worth simply writing it down for discussion after the meeting has ended.

Including an occurrence count with each raised obstacle highlights which issues are generally more important to deal with first.

Scrum Meetings

What order do we talk in? … establish a rule in advance, eg:

- **Last Arrival Speaks First**
- **Round Robin**
- **Pass The Token**
- **Take a Card**
- **Follow The Board**

Scrum Meetings

Where and when and how?

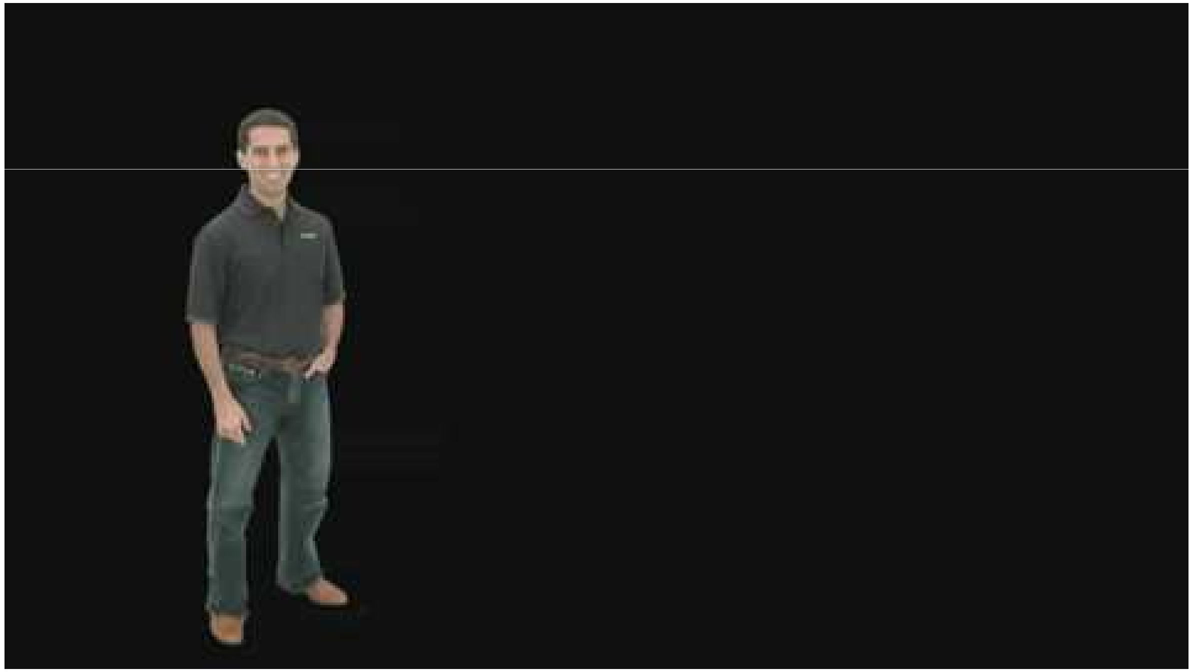
- **Meet Where the Work Happens**
- **Same Place, Same Time**
- **At the start of the day? … or not?**
- **Stand Up close to each other**
- **Prepare in advance**
- **Fifteen minutes or less … Take problem solving off-line**
- **Encourage Autonomy (rotate the faciltatior?)**

Scrum

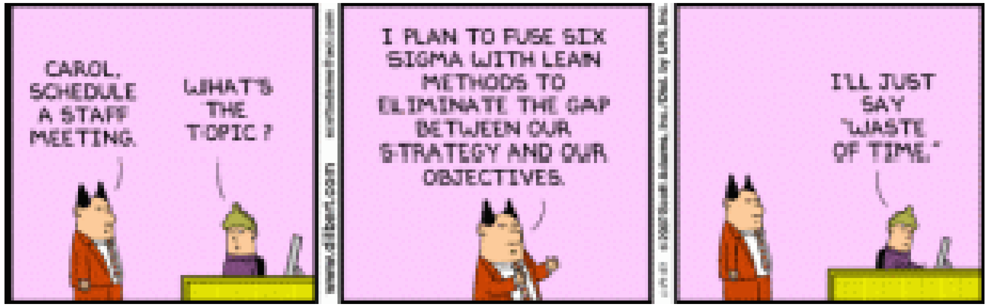
Pros	Cons
Scrum uses small Sprints to effectively break the system down into smaller components and divided among teams.	Scrum only works when management “trust the developers to use own judgment “to accomplish task. Key attributes of scrum is light and subtle control. So if development team is young and immature, Scrum is risky.
A key activity in scrum is the “daily scrum meetings”, which help team members to show evidence of task completion and allows for continuous improvement, thereby enabling rapid, bottom-up engineering	Scrum is ideally designed for company with “currently existing agile methods.” Therefore a company must already have some working knowledge of agile methods before using Scrum.

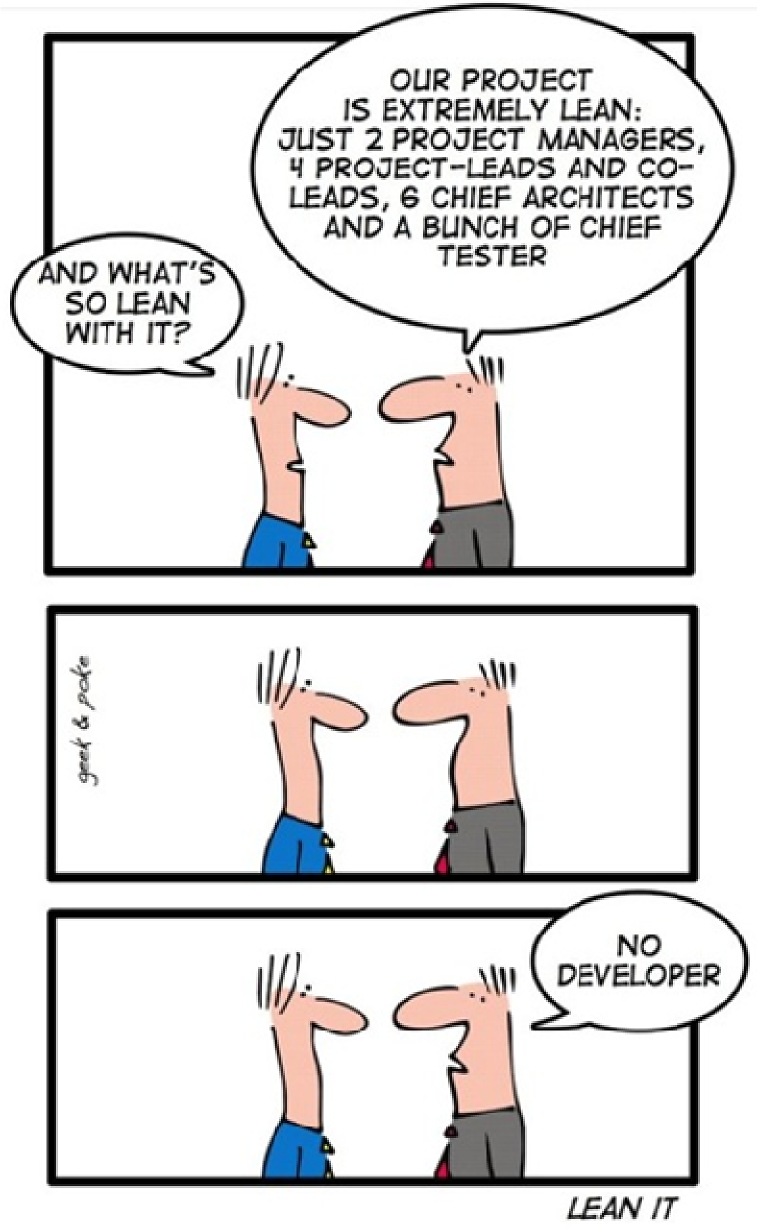
Scrum: a video

Scrum Master in Under 10 Minutes *(8:00)*
http://www.youtube.com/watch?v=ZVFcagL1nsA



Lean development


CAROL, SCHEDULE A STAFF MEETING.
WHAT'S THE TOPIC?
www.dilbert.com
I PLAN TO FUSE SIX SIGMA WITH LEAN METHODS TO ELIMINATE THE GAP BETWEEN OUR STRATEGY AND OUR OBJECTIVES.
I'LL JUST SAY "WASTE OF TIME."


AND WHAT'S SO LEAN WITH IT?
OUR PROJECT IS EXTREMELY LEAN: JUST 2 PROJECT MANAGERS, 4 PROJECT-LEADS AND CO-LEADS, 6 CHIEF ARCHITECTS AND A BUNCH OF CHIEF TESTER
geek & poke
NO DEVELOPER
LEAN IT

Lean development

Lean software development is a translation of lean manufacturing principles and practices to the software development domain. Adapted from the Toyota Production System, a pro-lean subculture is emerging from within the Agile community:

Mary Poppendieck and Tom Poppendieck. 2003. *Lean Software Development: An Agile Toolkit*. Addison-Wesley Longman Publishing

Mary Poppendieck and Tom Poppendieck. 2006. *Implementing Lean Software Development: From Concept to Cash (The Addison-Wesley Signature Series)*. Addison-Wesley Professional.

Mary Poppendieck. 2007. *Lean Software Development*. In Companion to the proceedings of the 29th International Conference on Software Engineering (ICSE COMPANION '07). IEEE Computer Society

Lean Software Development – key ideas

- Life would be so much easier if customers would just stop changing their minds.
- Let customers delay their decisions about exactly what they want as long as possible, and when they ask for something, give it to them so fast they don't have time to change their minds.
- Great designs come from great designers, and great designers understand that designs emerge as they develop a growing understanding of the problem, not collecting mass amounts of requirements.
- Deliver working system as fast as possible.

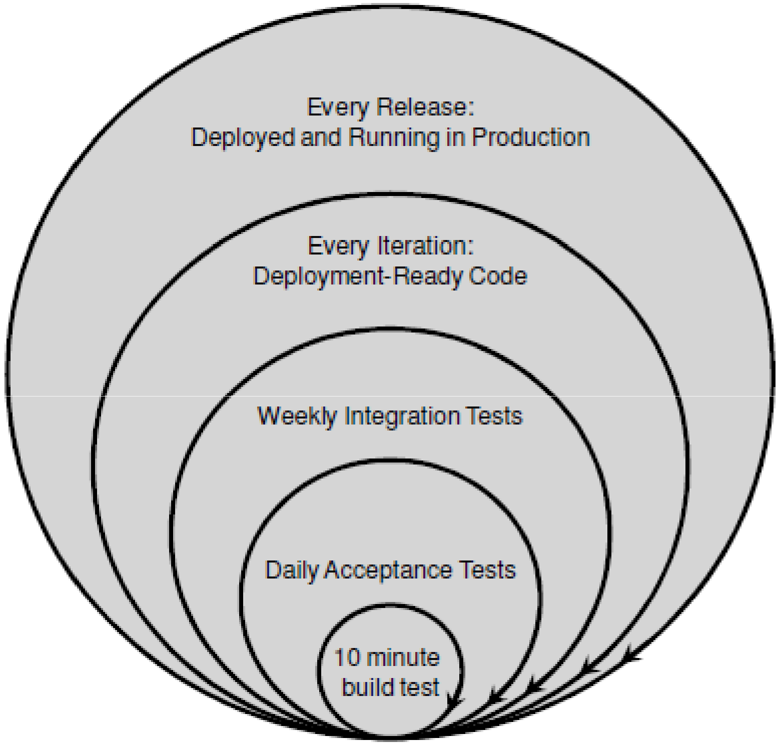
QUESTION: Do you agree with this?

Lean Software Development

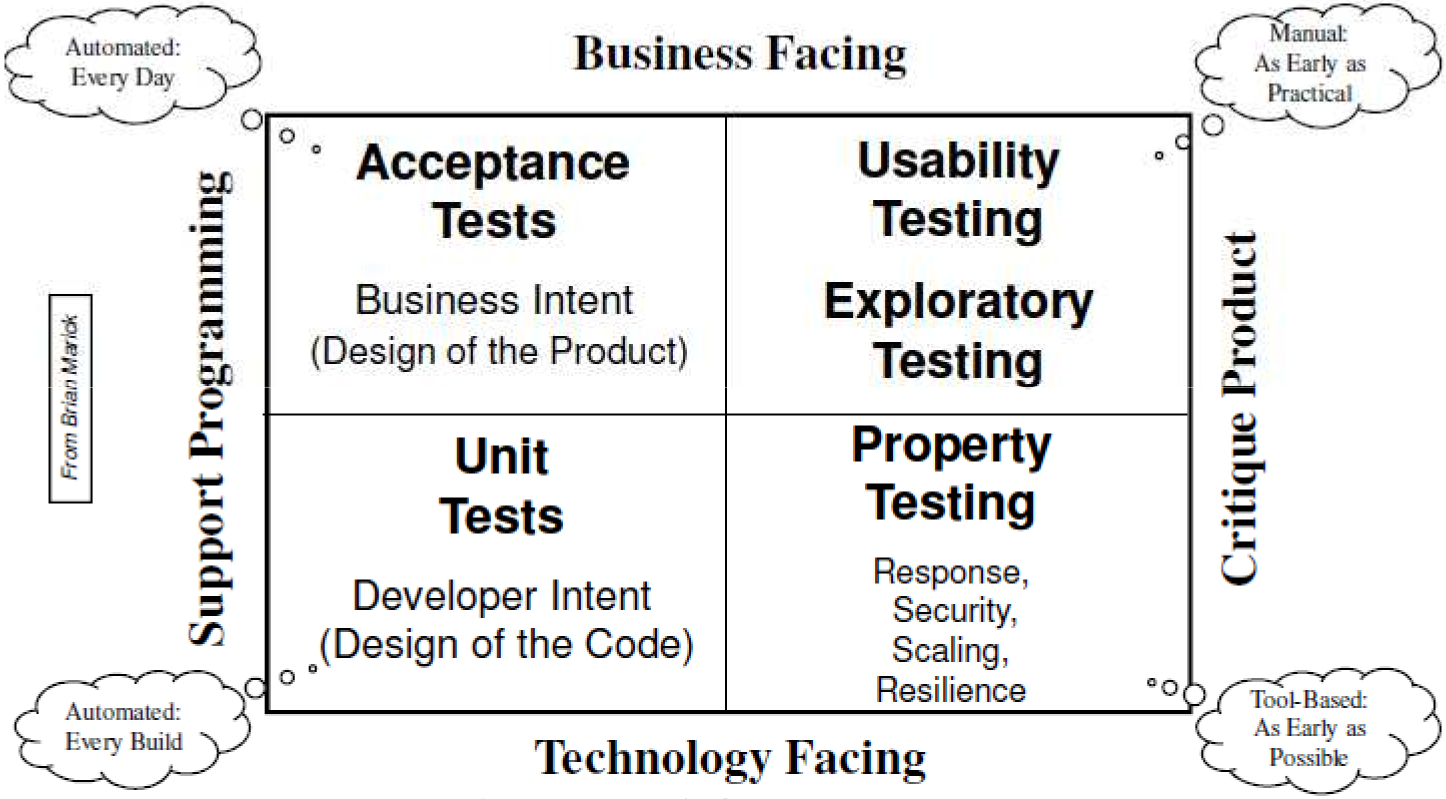
1. **Eliminate waste.** Any activity that does not “pay for itself” in reduced effort elsewhere in the system and should be removed
2. **Amplify learning.** Developers always need to learn new methods to produce the most robust system.
3. **Decide as late as possible.** The benefits of making a decision at the last minute is it avoids making the wrong decision early and then having to fix it later.
4. **Deliver as fast as possible.** This principle is that if you deliver very quickly, it reduces the chance of requirement changing. Which can cost a great deal if the change is late in the development.
5. **Empower the team**. In order to get people to take responsibility, get motivated, and gel as a team, they need to be responsible for the outcome and authorized to make it happen.
6. **Build integrity in:** It is key that the system maintains integrity through out the development cycle. That means integration test, unit testing and general testing is a must, particularly from the customer.
7. **See the whole.** Don’t break the system down into parts, but keep it as a whole.

Lean software development: focus on testing

- ✓Every few minutes
  - × Build & run unit tests
  - × ***STOP*** if the tests don’t pass
- ✓Every day
  - × Run acceptance tests
  - × ***STOP*** if the tests don’t pass
- ✓Every week
  - × Run production testes
  - × ***STOP*** if the tests don’t pass
- ✓Every iteration
  - × Deployment-ready code
- ✓Every Release
  - × Deploy and run in production



Lean software development: focus on testing



Automated: Every Day

Business Facing

Manual: As Early as Practical

From Brian Marick

Support Programming

Critique Product

Acceptance Tests Business Intent (Design of the Product)	**Usability Testing** **Exploratory Testing**
Unit Tests Developer Intent (Design of the Code)	**Property Testing** Response, Security, Scaling, Resilience

Automated: Every Build

Technology Facing

Tool-Based: As Early as Possible

Lean Software Development

Pros	Cons
Thinking the system as a whole, though difficult for complex system, helps to guarantee consistency and integrity of the system. Reduces integration time since since is developed as singular unit.	With large system or complex system, the only way for developers to visualize its construction is through partitioning the system. But LSD suggest the opposite, which can be difficult to accomplish.
The work team designs its own processes, makes its own commitments, gathers the information needed to reach its goals, and polices itself to meet its milestones.	Deciding as late as possible can have adverse affect to the schedule. This can hurt parallel development and increase implementation time.

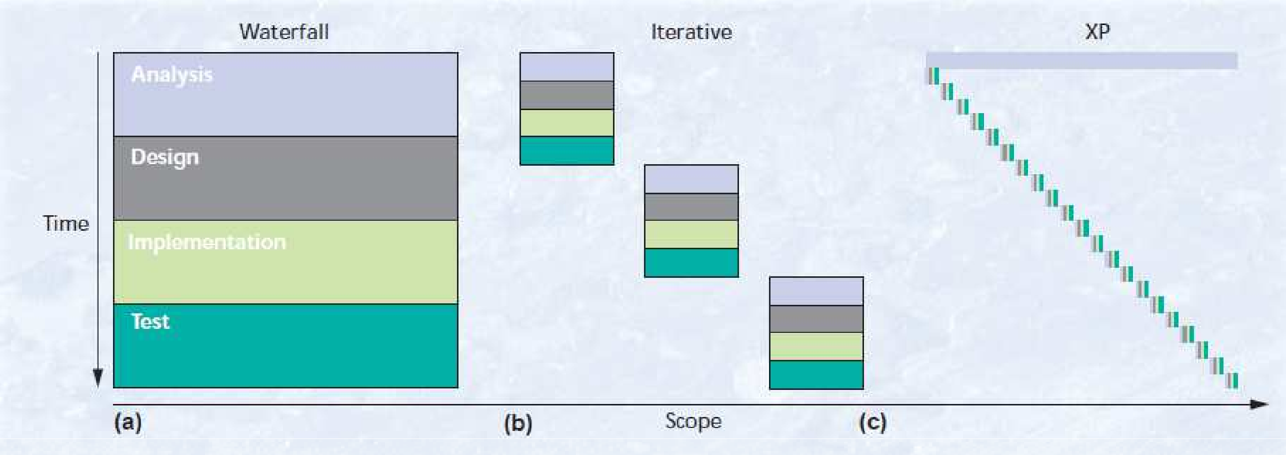
Extreme Programming (XP)

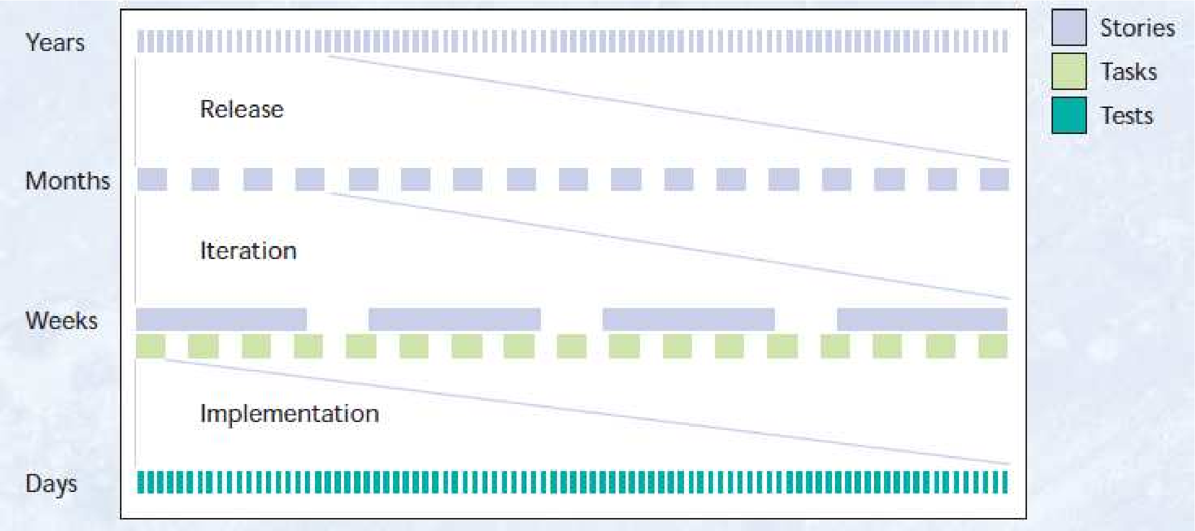
I'LL GO UP AND FIND OUT WHAT THEY NEED AND THE REST OF YOU START CODING!
SHELTON

XP is not this *extreme*!

Extreme Programming

From: ***Embracing Change with Extreme Programming*****, Beck, 1999**





Extreme Programming (XP)

***Embracing Change with Extreme Programming*, Beck, 1999**

You can't program until you know what you're programming.

XP considers the period before a system first goes into production to be a dangerous anomaly in the life of the project and to be gotten over as quickly as possible. However, every project has to start somewhere.

You put the overall analysis together in terms of stories, which you can think of as the amount of a use case that will fit on an index card. Each story must be business-oriented, testable, and estimable.

A month is a good long time to come up with the stories for a 10 person-year project.

Extreme Programming (XP)

***Embracing Change with Extreme Programming*, Beck, 1999**

How do you know what you should be programming at any particular time?

The customer picks the next release by choosing the most valuable features (called stories in XP) from among all the possible stories, as informed by the costs of the stories and the measured speed of the team in implementing stories.

The customer picks the next iteration's stories by choosing the most valuable stories remaining in the release, again informed by the costs of the stories and the team's speed.

The programmers turn the stories into smaller-grained tasks, which they individually accept responsibility for.

Then the programmer turns a task into a set of test cases that will demonstrate that the task is finished.

Working with a partner, the programmer makes the test cases run, evolving the design in the meantime to maintain the simplest possible design for the system as a whole.

XP Practices I (Usually associated with Agile)

Planning game. Customers decide the scope and timing of releases based on estimates provided by programmers. Programmers implement only the functionality demanded by the stories in this iteration.
Small releases. The system is put into production in a few months, before solving the whole problem. New releases are made often—anywhere from daily to monthly.
Metaphor. The shape of the system is defined by a metaphor or set of metaphors shared between the customer and programmers.
Simple design. At every moment, the design runs all the tests, communicates everything the programmers want to communicate, contains no duplicate code, and has the fewest possible classes and methods. This rule can be summarized as, "Say everything once and only once."
Tests. Programmers write unit tests minute by minute. These tests are collected and they must all run correctly. Customers write functional tests for the stories in an iteration. These tests should also all run, although practically speaking, sometimes a business decision must be made comparing the cost of shipping a known defect and the cost of delay.
Refactoring. The design of the system is evolved through transformations of the existing design that keep all the tests running.
Continuous integration. New code is integrated with the current system after no more than a few hours. When integrating, the system is built from scratch and all tests must pass or the changes are discarded.

XP Practices II (also found outside Agile)

Pair programming. All production code is written by two people at one screen/keyboard/mouse.

Collective ownership. Every programmer improves any code anywhere in the system at any time if they see the opportunity.

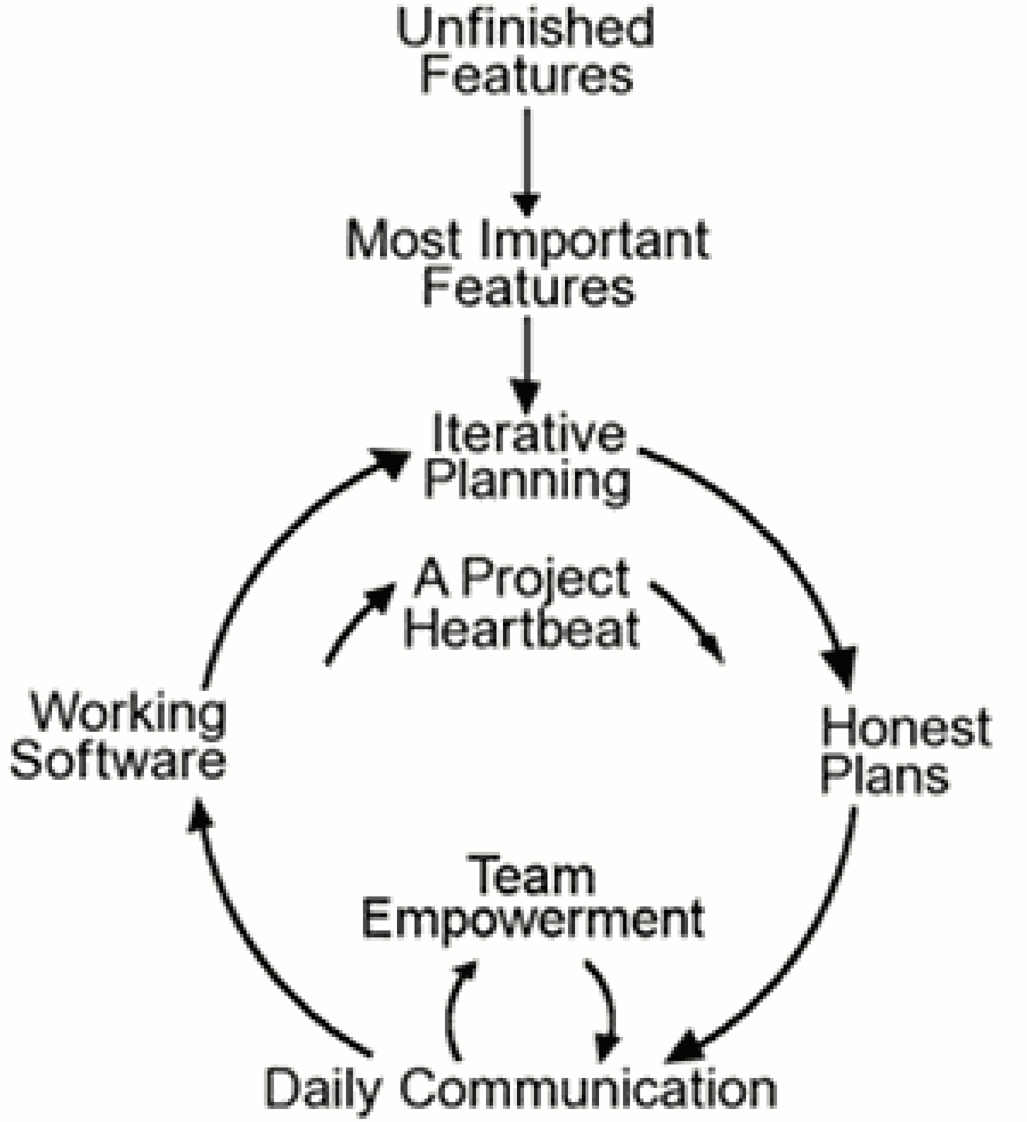
On-site customer. A customer sits with the team full-time.

40-hour weeks. No one can work a second consecutive week of overtime. Even isolated overtime used too frequently is a sign of deeper problems that must be addressed.

Open workspace. The team works in a large room with small cubicles around the periphery. Pair programmers work on computers set up in the center.

Just rules. By being part of an Extreme team, you sign up to follow the rules. But they're just the rules. The team can change the rules at any time as long as they agree on how they will assess the effects of the change.

XP – Daily Communication is Key



Extreme Programming (XP)

For XP to be a success, critical expertise is required In:

Building User Stories - A user story describes problems to be solved by the system being built. These stories must be written by the user and should be about three sentences long. User stories do not describe a solution, use technical language, or contain traditional requirements-speak.

Turning stories into code - Because user stories are short and somewhat vague, XP will only work if the customer representative is on hand to review and approve user story implementations.

Turning stories into test code – Unit testing is central to XP, where two twists on conventional testing strategies make tests far more effective: Programmers write their own tests and they write these tests before they code.

Evolving design – Must not break existing tests and must also support scaleable incremental development

Extreme Programming (XP)

Beck acknowledges a wide number of influences that led to the development of XP. A selection of the most accessible are:

C. Alexander, *The Timeless Way of Building,* Oxford University Press, New York, 1979.

W. Cunningham, “Episodes: A Pattern Language of Competitive Development,” *Pattern Languages of Program Design 2,* J. Vlissides, ed., Addison-Wesley, New York, 1996.

I. Jacobsen, *Object-Oriented Software Engineering, Addison-Wesley, New York,* 1994.

T. Gilb, *Principles of Software Engineering Management, Addison-Wesley, Wokingham,* UK, 1988.

B. Boehm, “A Spiral Model of Software Development and Enhancement,” *Computer,* May 1988, pp. 61-72.

R. Coyne, *Designing Information Technology in the Postmodern Age, MIT Press, Cambridge, Mass., 1995.*

T. DeMarco and T. Lister, *Peopleware,* Dorset House, New York, 1999.

The pros of XP

- Done well, XP improves teamwork.
    - It builds true competency in all team members.
    - It makes for an enjoyable and honest work day.
    - It gets people out of their cubes and talking to one another.

- TDD teaches developers about how to write quality code and how to improve their notions of design; it helps them to improve estimates. It improves the resumes of developers.

- It gives management many tools, including predictability, flexibility of resources, consistency, and visibility into what's really going on.

- It gives customers the ability to see whether or not a company can deliver on its promises.

- You don't spend a lot of time in stupid, wasteful meetings, and you don't produce a lot of useless documents

The cons of XP

- Design becomes implicit rather than explicit
- Relying on emergent design is risky
- It is very hard to write good tests
- Too frequent iterations can compromise quality
- To do it well you need to do it often – so its hard to introduce successfully

Extreme Programming (XP): video

Extreme Programming (Animated Video) (4:12)
http://www.youtube.com/watch?v=X_2PfTvXBeA

Adaptive Software Development (ASD)

The software development community has a […]dichotomy. One is represented by the more traditional deterministic development, derived from management practices rooted in nineteenth-century Newtonian physics of stability and predictability -- or in Arthur's terms, decreasing returns. [The other]is about the second world -- unpredictable, nonlinear, and fast.

American Programmer, Volume X, No. 1; January 1997.
http://www.adaptivesd.com/articles/messy.htm

Adaptive Software Development (ASD)

Adaptive Software Development grew out of rapid application development work by Jim Highsmith and Sam Bayer.

Principle of ASD: continuous adaptation of the process to the work at hand is the normal state of affairs.

Speculate - the paradox of planning – it is more likely to assume that all stakeholders are comparably wrong for certain aspects of the project's mission, while trying to define it.

Collaboration - balancing the work based on predictable parts of the environment (planning and guiding them) and adapting to the uncertain

Learning -challenge all stakeholders - knowledge is gathered by making small mistakes based on false assumptions and correcting those mistakes

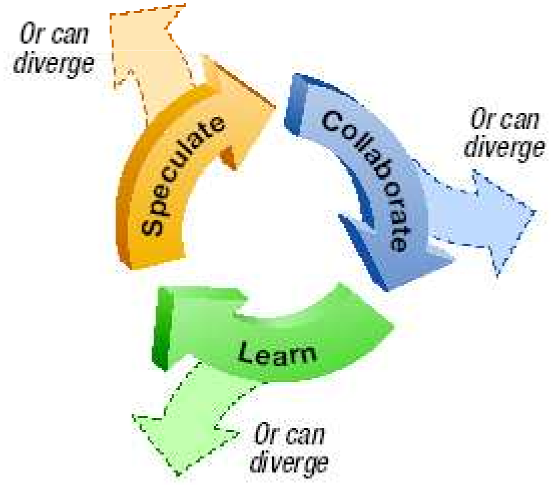


FIGURE 1 The adaptive lifecycle

Agile Modeling

Agile Modeling is a supplement to other Agile methodologies such as: Extreme Programming, Agile Unified Process, Scrum

AMDD is the agile version of Model Driven Development (MDD)

Scott Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, 2002, John Wiley

“In my opinion, generative MDD is a lost cause for the current generation of developers. Agile MDD will be a struggle to pull off, but at least it has a chance of succeeding.”

Agile model driven development is good enough, Scott Ambler, IEEE Software, 2003

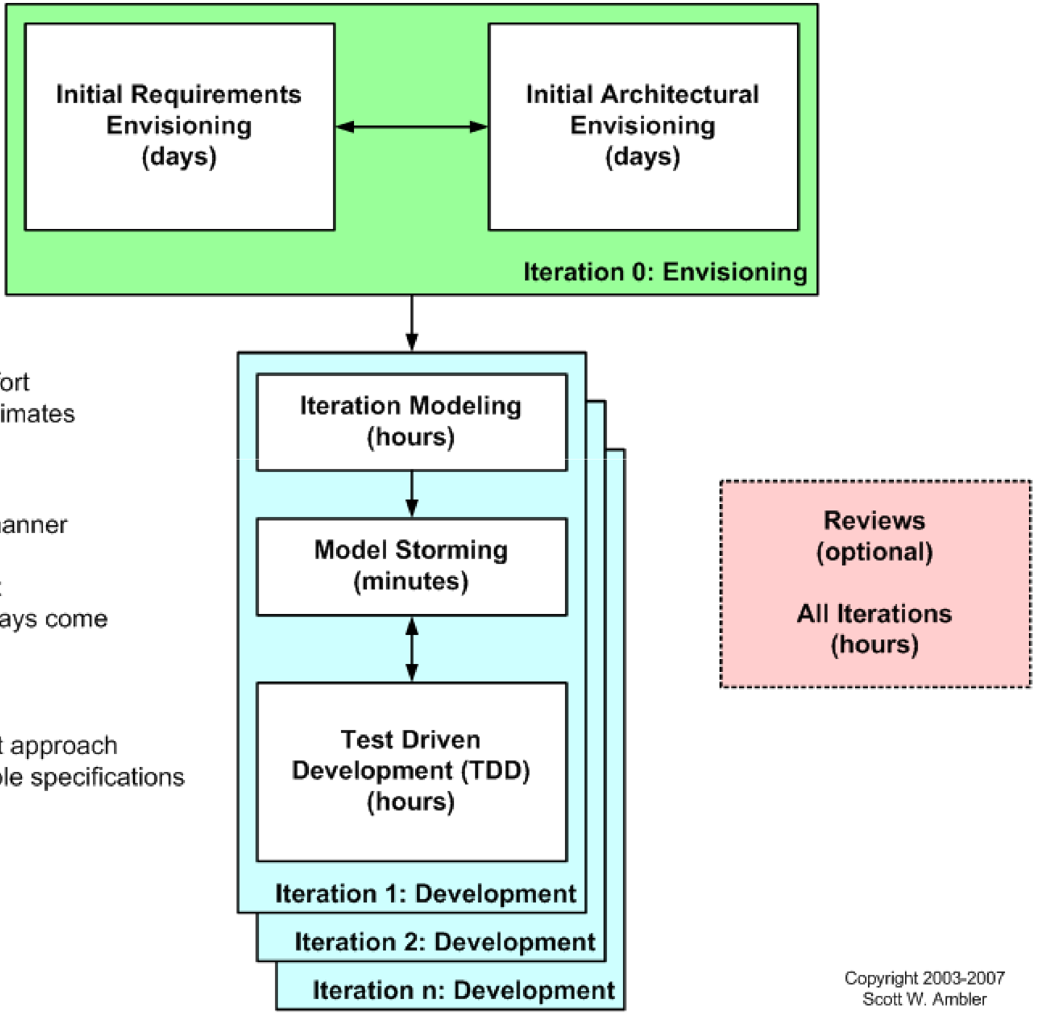
Agile Modeling

- Identify the high-level scope
- Identify initial "requirements stack"
- Identify an architectural vision

- Modeling is part of iteration planning effort
- Need to model enough to give good estimates
- Need to plan the work for the iteration

- Work through specific issues on a JIT manner
- Stakeholders actively participate
- Requirements evolve throughout project
- Model just enough for now, you can always come back later

- Develop working software via a test-first approach
- Details captured in the form of executable specifications



Copyright 2003-2007
Scott W. Ambler

Crystal Methods

Alistair Cockburn developed the Crystal family of software development methods as a group of approaches tailored to different size teams.

Alistair Cockburn. *Crystal Clear a Human-Powered Methodology for Small Teams* (First ed.), 2004, Addison-Wesley Professional.

Crystal Methods

Alistair Cockburn developed the Crystal family of software development methods as a group of approaches tailored to different size teams.

Crystal is seen as a family because Cockburn believes that different approaches are required as teams vary in size and the criticality of errors changes.

Despite their variations all crystal approaches share common features:

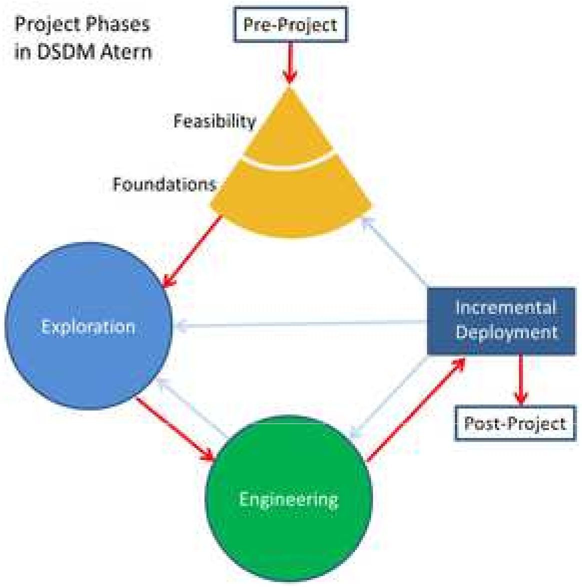
- All crystal methods have three priorities: safety (in project outcome), efficiency, habitability (developers can live with crystal).

- They also share common properties, of which the most important three are: Frequent Delivery, Reflective Improvement, and Close Communication.

The goal of Crystal : the least amount of process you can do and still succeed with an underlying assumption of low-discipline that is inevitable with humans.

Crystal requires less discipline than extreme programming, trading off less efficiency for a greater habitability and reduced chances of failure.

Dynamic System Development Methodology (DSDM)



A commercial « product » to aid with management of Agile approaches.

It is an agile project delivery framework, primarily used as a software development method.

DSDM was originally based upon the rapid application development (RAD) method.

James Martin. 1991. *Rapid Application Development*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA.

Feature Driven Development

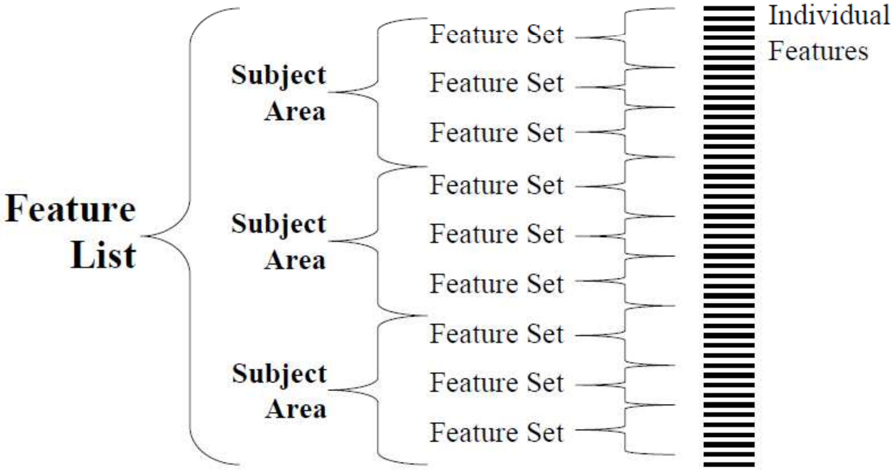
Evolved from the Coad Method, in the late 1990s

Peter Coad and Jeff De Luca **later published a brief outline of FDD in their** **1999 book *"Java Modeling in Color with UML"*.**

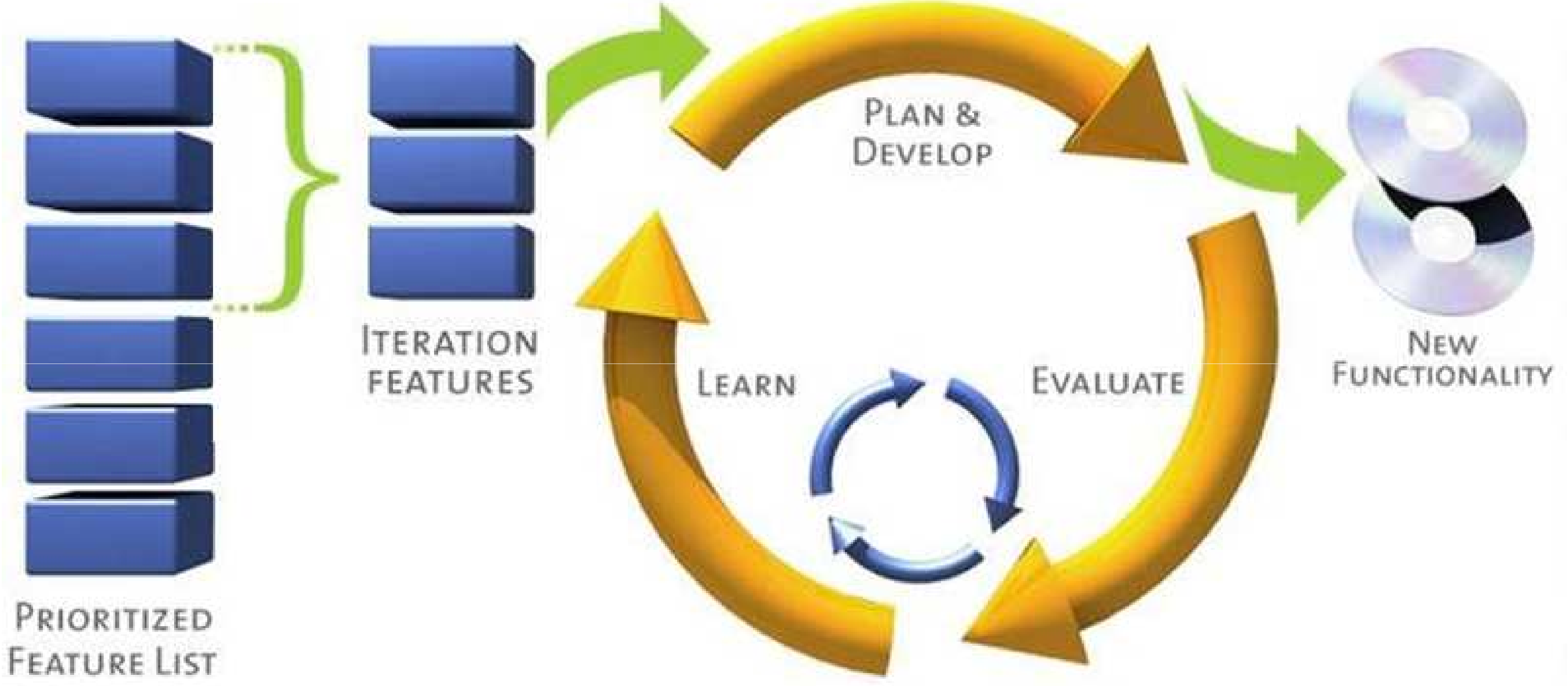
Stephen Palmer with Mac Felsing **wrote the definitive textbook*, "A Practical Guide to Feature Driven Development"* two years later**.

Features:

Are tiny
Map directly onto an object domain model
Can be coded directly
Can be assembled in component sets.



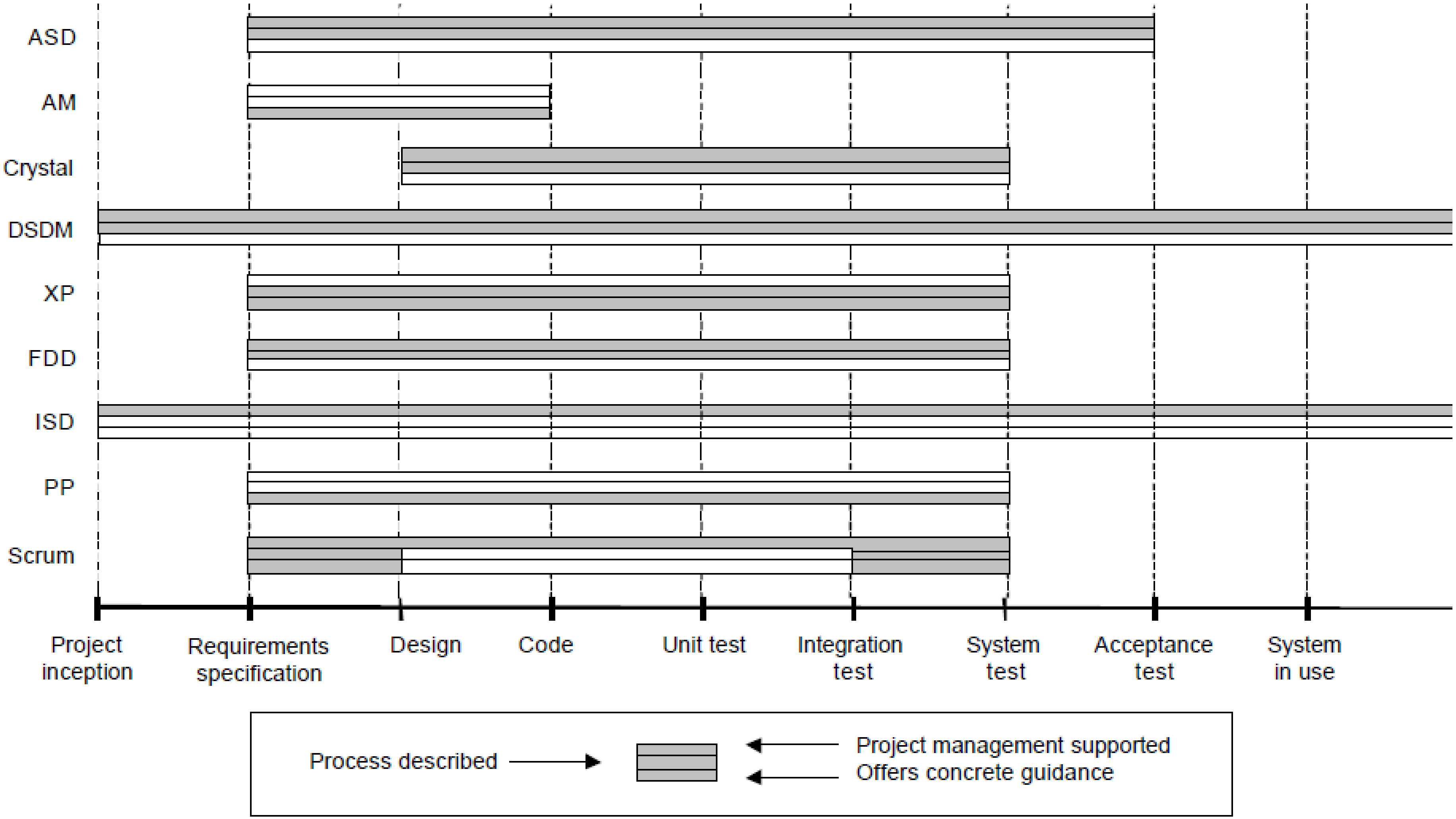
Feature Driven Development: looks pretty agile!



I would not recommend this for beginners to Agile development as it is hard to manage interactions between features.

Comparing Agile Methods

Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. 2003. New directions on agile methods: a comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering* (ICSE '03). IEEE Computer Society, Washington,



http://en.wikipedia.org/wiki/Agile_software_development

http://www.agilealliance.org/

http://agilemanifesto.org/

http://scrummethodology.com/

http://www.extremeprogramming.org/

http://www.leansoftwareinstitute.com/

Agile research topics

- **Agile Formal Methods**
- **Service-oriented Agile for the cloud**
- **Pair programming**
- **Agile and CMMi**
- ***Agile and MDD/MDA***
- ***Agile and automated testing***
- ***Teaching agile***
- ***Agile – (when) does it work?***

QUESTIONS ….?

TODAY WE'LL DISCUSS WAYS TO IMPROVE OUR WORKFLOW PROCESS.
AS YOU KNOW, A GOOD PROCESS IS A SUBSTITUTE FOR GOOD EMPLOYEES.
THE ULTIMATE GOAL IS TO SIMPLIFY OUR PROCESSES SO MUCH . . .
THAT WE CAN TRAIN CHICKENS TO DO YOUR JOBS IN RETURN FOR PELLETS.
WE'LL BEGIN BY DISCUSSING OUR PROCESS FOR FUNDING NEW PROJECTS.
COULD ANY PART OF OUR PROCESS BE REPLACED BY, FOR EXAMPLE, RINGING A BELL WITH YOUR BEAK?
YES, BUT ONLY THE PART THAT YOU DO.
THERE'S A WRINKLE IN THE PLAN.
PELLET
DilbertCartoonist@gmail.com
© 2009 Scott Adams, Inc./Dist. by UFS, Inc.
www.dilbert.com

Feedback helps me improve my teaching process!