Process: Linear to Agile

Claire Le Goues

Learning goals

- Understand the need for process considerations
- Select a process suitable for a given project
- Address project and engineering risks through iteration
- Ensure process quality.
- Define agile as both a set of iterative process practices and a business approach for aligning customer needs with development.
- Explain the motivation behind and reason about the tradeoffs presented by several common agile practices.
- Summarize both scrum and extreme programming, and provide motivation and tradeoffs behind their practices.
- Identify and justify the process practices from the agile tradition that are most appropriate in a given modern development process.

Administrivia

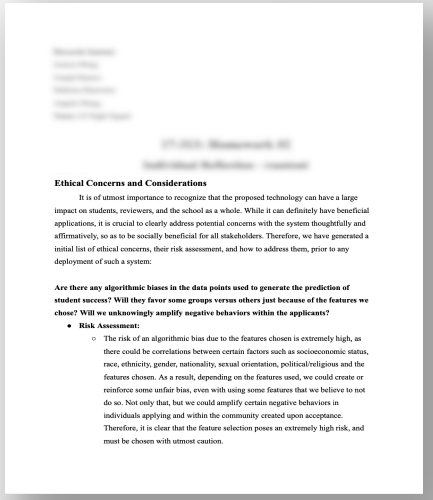
- Reflections.
- Final presentation protocol, video.

Writing for Professionals

Communication Tips

- The more important the person you are communicating with, the more limited their time. Your success depends on your ability to get your message through as quickly as possible
- Respect their time for optimal results. Plan on your document to be skimmed. Draw them to the most important parts using bold, paragraphs, indentation, etc.
- If you have a specific ask, make it as clear as possible.
- Make it easy for them to understand why you arrived at that conclusion

HW 4 Example

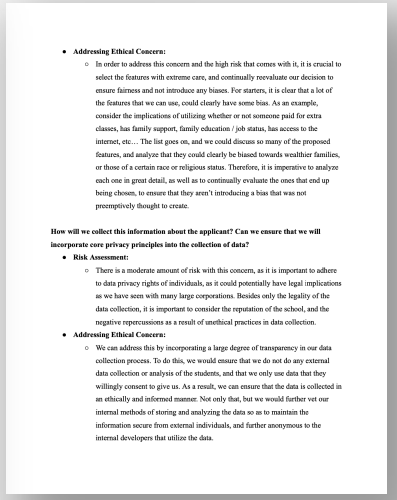


Ethical Concerns and Considerations

It is of utmost importance to recognize that the proposed technology can have a large impact on students, reviewers, and the school as a whole. While it can definitely have beneficial applications, it is crucial to clearly address potential concerns with the system thoughtfully and affirmatively, so as to be socially beneficial for all stakeholders. Therefore, we have generated a initial list of ethical concerns, their risk assessment, and how to address them, prior to any deployment of such a system:

Are there any algorithmic biases in the data points used to generate the prediction of student success? Will they favor some groups versus others just because of the features we chose? Will we unknowingly amplify negative behaviors within the applicants?

- **Risk Assessment:**
  - The risk of an algorithmic bias due to the features chosen is extremely high, as there could be correlations between certain factors such as socioeconomic status, race, ethnicity, gender, nationality, sexual orientation, political/religious and the features chosen. As a result, depending on the features used, we could create or reinforce some unfair bias, even with using some features that we believe to not do so. Not only that, but we could amplify certain negative behaviors in individuals applying and within the community created upon acceptance. Therefore, it is clear that the feature selection poses an extremely high risk, and must be chosen with utmost caution.



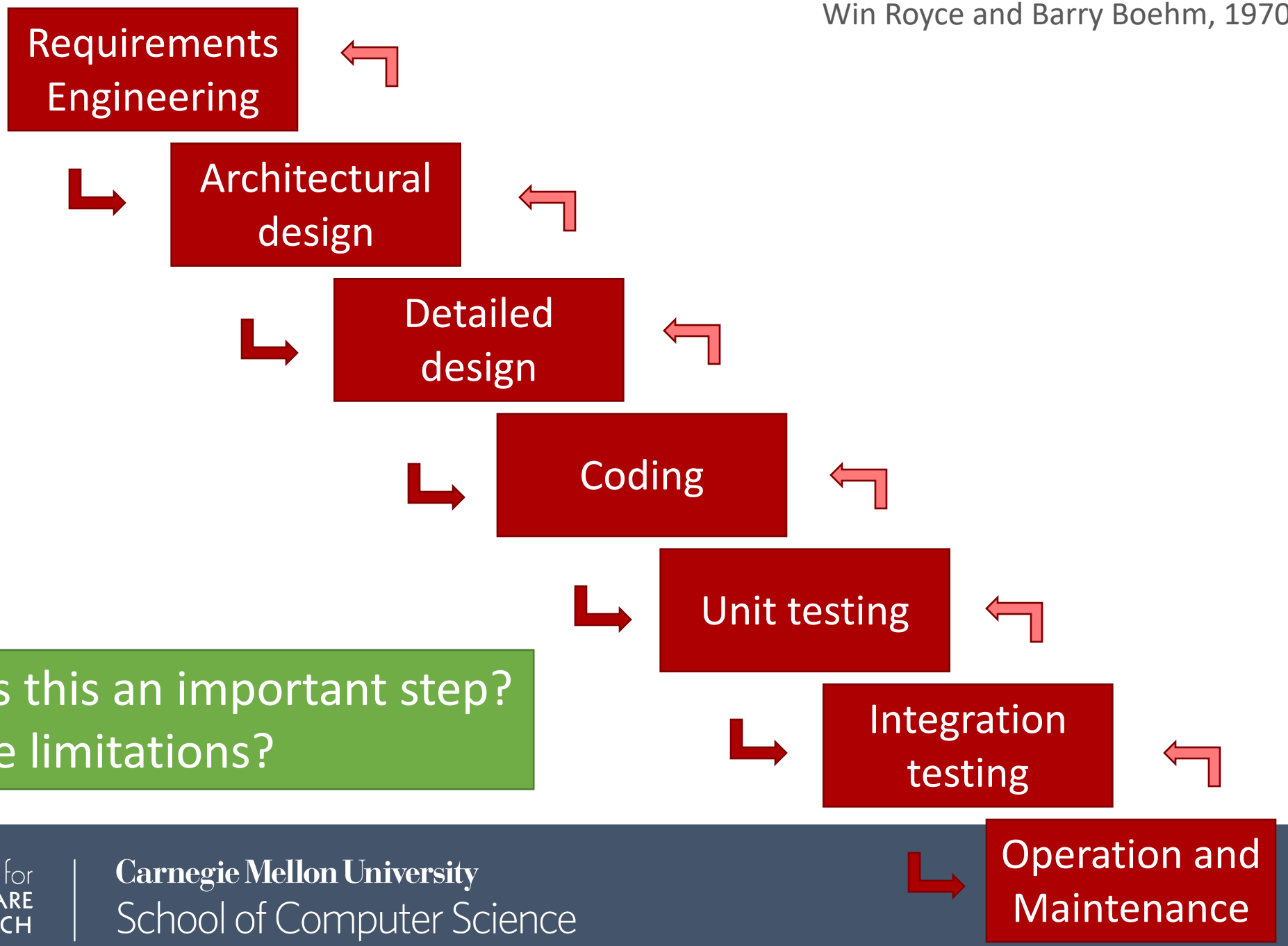
- **Addressing Ethical Concern:**
  - In order to address this concern and the high risk that comes with it, it is crucial to select the features with extreme care, and continually reevaluate our decision to ensure fairness and not introduce any biases. For starters, it is clear that a lot of the features that we can use, could clearly have some bias. As an example, consider the implications of utilizing whether or not someone paid for extra classes, has family support, family education / job status, has access to the internet, etc… The list goes on, and we could discuss so many of the proposed features, and analyze that they could clearly be biased towards wealthier families, or those of a certain race or religious status. Therefore, it is imperative to analyze each one in great detail, as well as to continually evaluate the ones that end up being chosen, to ensure that they aren't introducing a bias that was not preemptively thought to create.

How will we collect this information about the applicant? Can we ensure that we will incorporate core privacy principles into the collection of data?

- **Risk Assessment:**
  - There is a moderate amount of risk with this concern, as it is important to adhere to data privacy rights of individuals, as it could potentially have legal implications as we have seen with many large corporations. Besides only the legality of the data collection, it is important to consider the reputation of the school, and the negative repercussions as a result of unethical practices in data collection.
- **Addressing Ethical Concern:**
  - We can address this by incorporating a large degree of transparency in our data collection process. To do this, we would ensure that we do not do any external data collection or analysis of the students, and that we only use data that they willingly consent to give us. As a result, we can ensure that the data is collected in an ethically and informed manner. Not only that, but we would further vet our internal methods of storing and analyzing the data so as to maintain the information secure from external individuals, and further anonymous to the internal developers that utilize the data.

THE WATERFALL MODEL

Win Royce and Barry Boehm, 1970



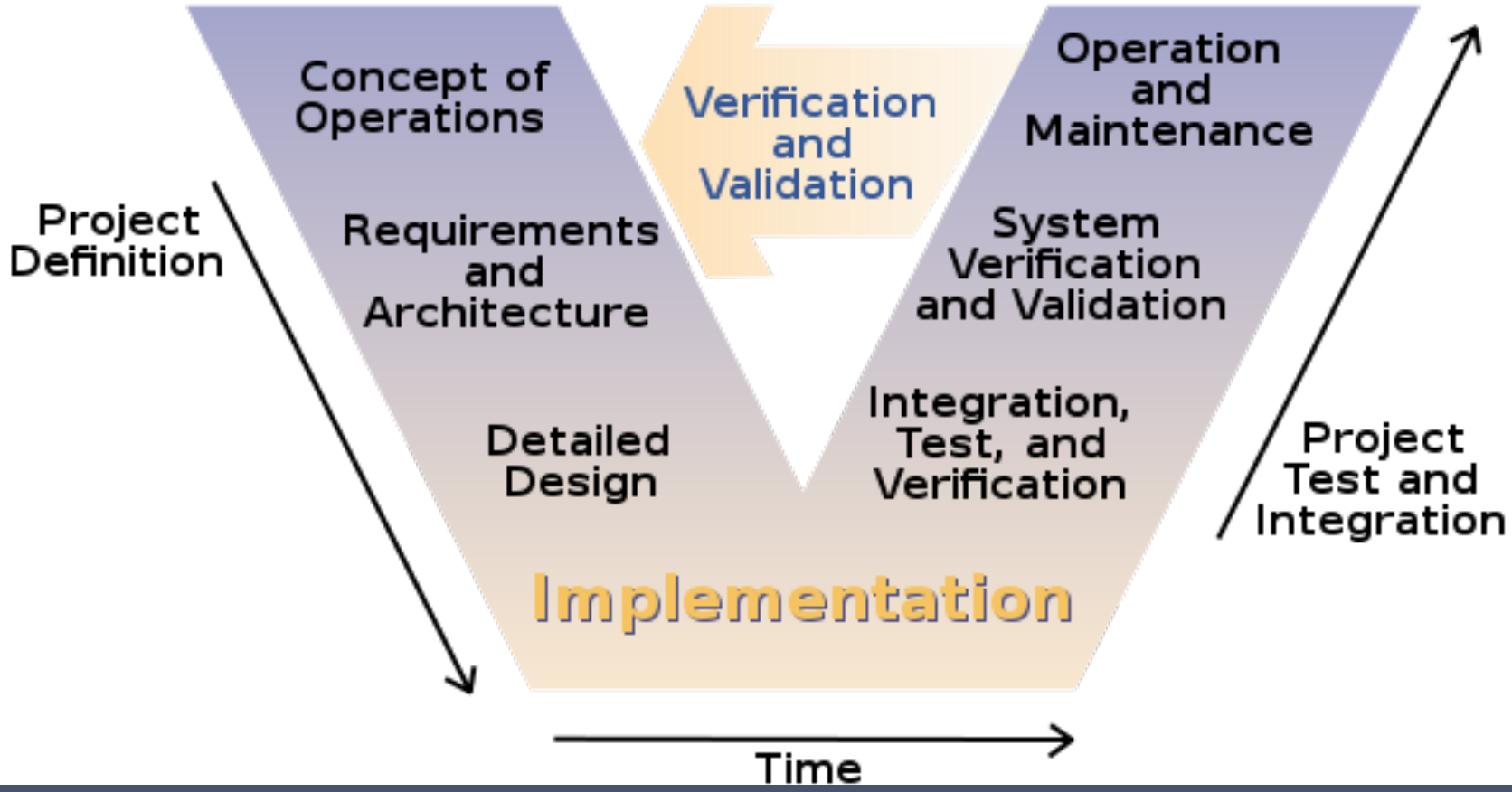
Why was this an important step?
What are limitations?

1:32pm
July 16th 1969

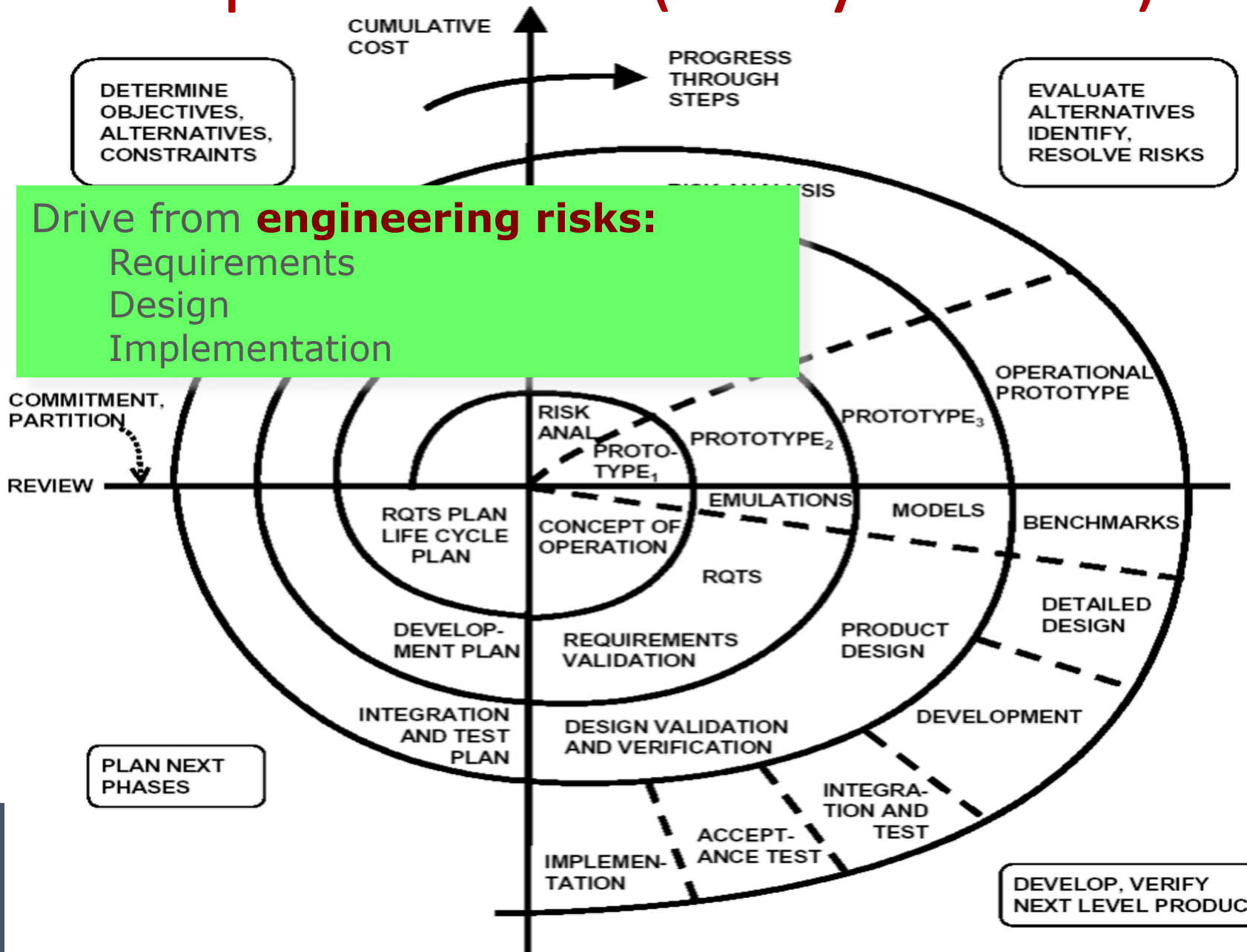
Key challenge: Change

- Software seems changeable ("soft")
- Developers prone to changes and "extra features"
- Customers often do not understand what is easy to change and what is hard
- "Good enough" vs. "optimal"

The "V" Model (80s, 90s)



The Spiral Model (Barry Boehm)



Drive from **engineering risks:**

- Requirements
- Design
- Implementation

When is waterfall appropriate?

1. The requirements are known in advance.
2. The requirements have no unresolved, high-risk risks such as due to cost, schedule, performance, safety, security, user interfaces, organizational impacts, etc.
3. The nature of the requirements will not change very much.
4. The requirements are compatible with all the key system stakeholders' expectations.
5. The architecture for implementing the requirements is well understood.
6. There is enough time to proceed sequentially.

Early improvement: sequencing

- Enforce earlier software considerations
- Waterfall instituted at TRW (Aerospace Govt Contractor) in 70s, with several additional recommendations for iterations (like prototypes).
- Modeled after traditional engineering
  - blueprints before construction
  - decide what to build, build it, test it, deploy
  - Reduce change
- Successful model for routine development
- Problematic at large scale
  - Requirements -> Delays -> Surprise!

Iteration!

-> Early and frequent feedback
-> Support for constant adaptation
-> Address risks first

Mitigation of risk through process interventions (examples)

- Risk-driven process
  - Prioritization and prototyping
- Architecture and design
  - Isolate/encapsulate risks
  - Follow industry standards
- Design for assurance
  - Preventive engineering
  - Codevelopment of system and evidence
- Functionality and usability
  - Prototypes , early usability labs

Key: Iterative Processes

- Interleaving and repeating
  - Requirements engineering, Risk assessment
  - Architecture and design
  - Implementation
  - Quality assurance
  - Deployment
- But when, in which sequence, and how often?
- What measurements can ground decisions?

Iteration decision

- Too slow?
- Too fast?
- -> Drive by risks and measurement data; per project decision
- Contracts?

Iteration decision

- Too slow?
  - Late reaction, reduce predictability
- Too fast?
  - Overhead, reduce innovation
- "Death spiral"
  - deferred commitment, prototypes without conclusions, missing feedback loops
- -> Drive by risks and measurement data; per project decision
- Contracts?

Process quality.

DISCUSSION: WHAT MAKES A GOOD PROCESS?

Process evaluation

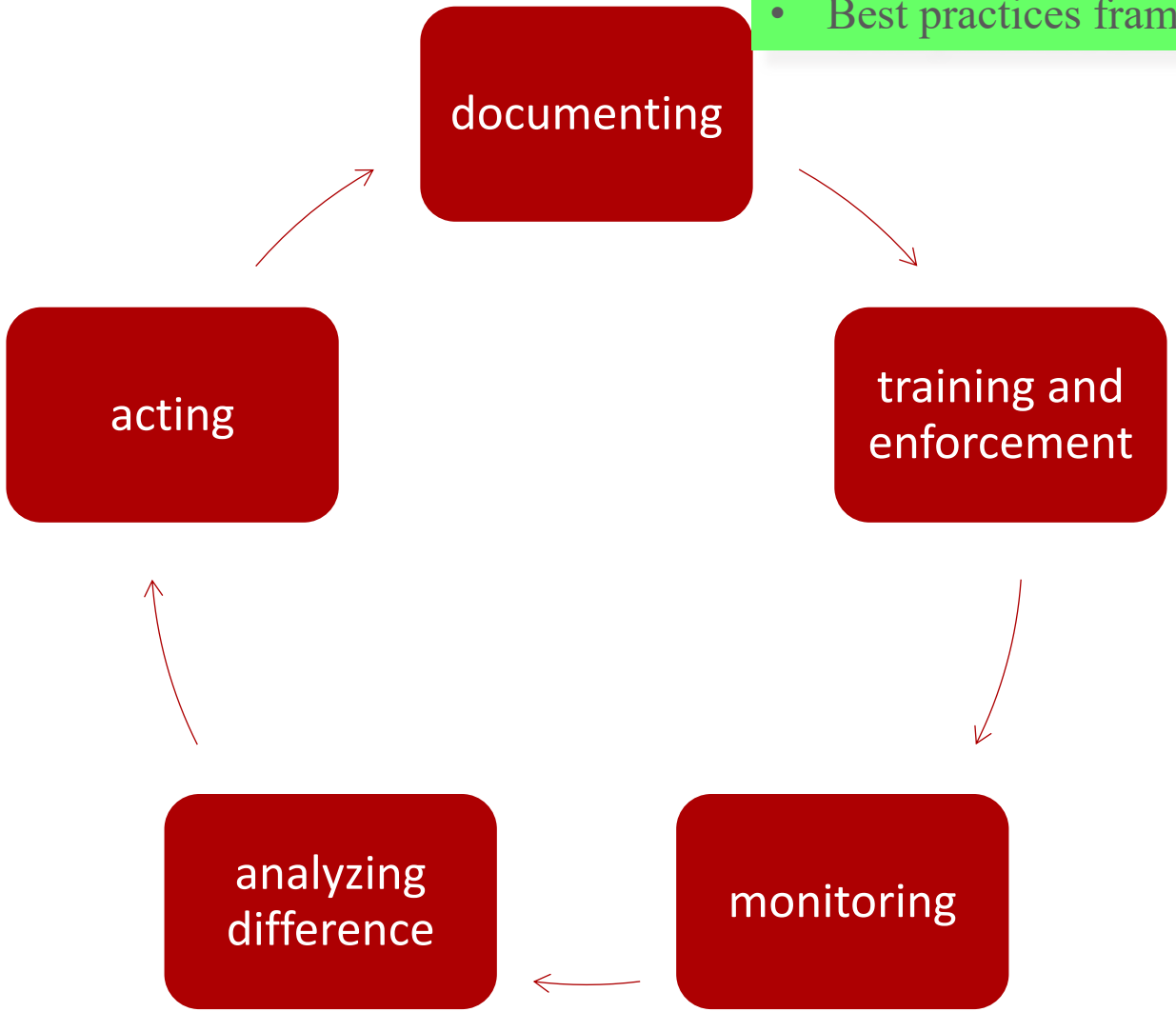
- How predictable are our projects?

- 33% of organizations collect productivity and efficiency data
- 8% collect quality data
- 60% do not monitor their processes

Process improvement loop

High-level approaches:

- Opportunistic, based on double-loop learning.
- Analytic, based on measurement + principles
- Best practices frameworks



Agile Software Development Is …

Both:

- a set of software engineering best practices (allowing for rapid delivery of high quality software)
- a business approach (aligning development with customer needs and goals)

Brief History of Agile

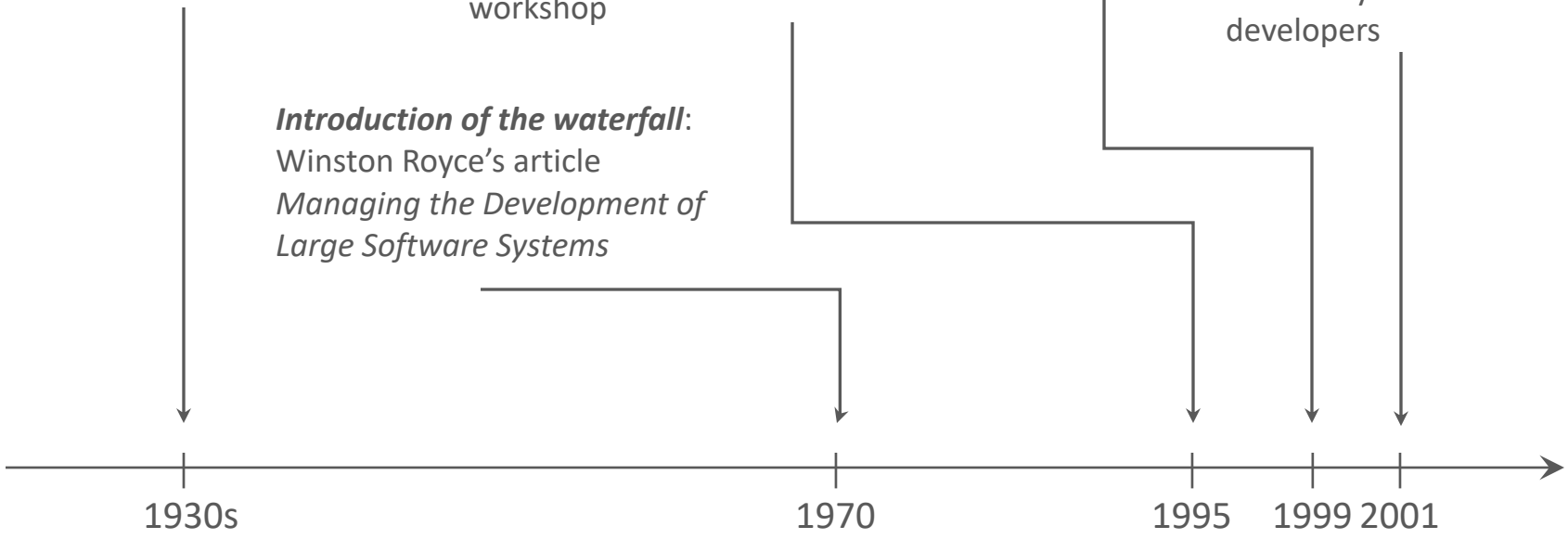
Inception of Iterative and Incremental Development (IID): Walter Shewhart (Bell Labs, signal transmission) proposed a series of "plan-do-study-act" (PDSA) cycles

Introduction of Scrum: Jeff Sutherland and Ken Schwaber presented a paper describing the Scrum methodology at a conference workshop

XP reified: Kent Beck released *Extreme Programming Explained: Embrace Change*

Introduction of "Agile": The Agile Manifesto written by 17 software developers

Introduction of the waterfall: Winston Royce's article *Managing the Development of Large Software Systems*



Agile in a nutshell

- A project management approach that seeks to respond to change and unpredictability, primarily using incremental, iterative work sequences (often called “sprints”).
- Also: a collection of practices to facility that approach.
- All predicated on the principles outlined in “The Manifesto for Agile Software Development.”

The Manifesto for Agile Software Development (2001)

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

The Twelve Principles of Agile Software Development

Individuals and interactions

1. Projects are built around motivated individuals, who should be trusted
2. Face-to-face conversation is the best form of communication (co-location)
3. Self-organizing teams

Working software

4. Working software is delivered frequently (weeks rather than months)
5. Working software is the principal measure of progress
6. Sustainable development, able to maintain a constant pace
7. Continuous attention to technical excellence and good design
8. Simplicity—the art of maximizing the amount of work not done—is essential

Customer collaboration

9. Customer satisfaction by rapid delivery of useful software
10. Close, daily cooperation between business people and developers

Responding to change

11. Welcome changing requirements, even late in development
12. Regular adaptation to changing circumstances

Agile Practices

- Backlogs (Product and Sprint)
- Behavior-driven development (BDD)
- Cross-functional team
- Continuous integration (CI)
- Domain-driven design (DDD)
- Information radiators (Kanban board, Task board, Burndown chart)
- Acceptance test-driven development (ATDD)
- Iterative and incremental development (IID)
- Pair programming
- Planning poker
- Refactoring
- Scrum meetings (Sprint planning, Daily scrum, Sprint review and retrospective)
- Small releases
- Simple design
- Test-driven development (TDD)
- Agile testing
- Timeboxing
- Use case
- User story
- Story-driven modeling
- Retrospective
- On-site customer
- Agile Modeling
- 40-hour weeks
- Short development cycles
- Collective ownership
- Open workspace
- Velocity tracking
- Etc.

40-hour Weeks

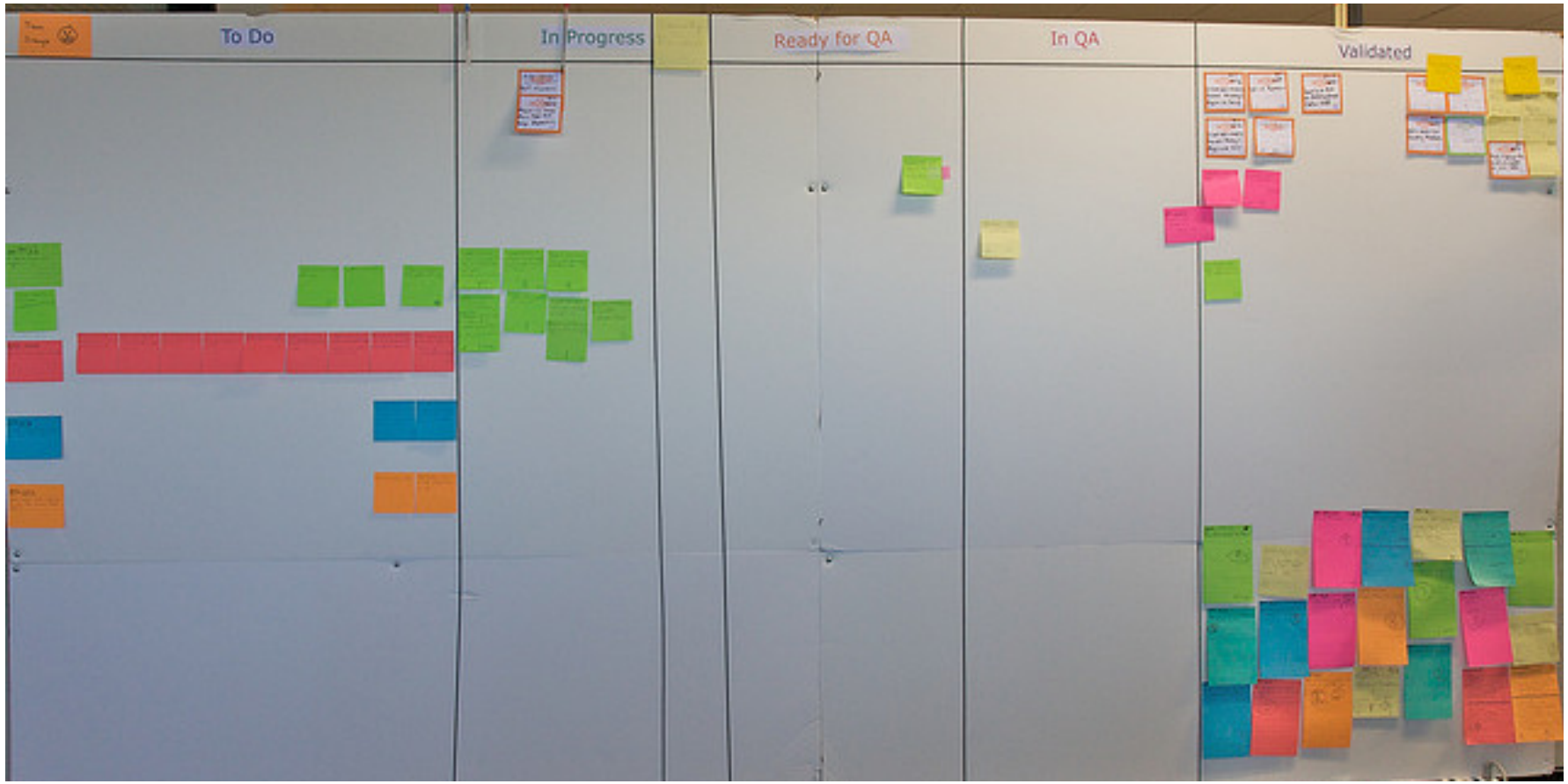
No one can work a second consecutive week of overtime. Even isolated overtime used too frequently is a sign of deeper problems that must be addressed.

Planning Poker

Collective Ownership

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

Kanban Board



Simple Design

“Say everything once and only once”:

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.

On-site Customer

A customer sits with the team full-time.

Pair Programming

Short development cycle

The software development process is organized in a way in which the full software development cycle—from design phase to implementation phase to test and deployment phase—is performed within a short timespan, usually several months or even weeks.

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.

Continuous Integration (CI)

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.

Test-driven development

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.

Open workspace

Solving Software Development Problems with Agile Practices

	Problem in Software Development	Agile Methods That Mitigate It
1.	Requirement changes during the development process	Close relation with customer, short development cycle, small releases, planning poker, Kanban board
2.	Scope creep	Short development cycle, small releases, planning poker
3.	Architecture erosion	Collective ownership, pair programming
4.	Under- or overestimation (time and budget), sticking to the plan	Close relation with customer, planning poker, short development cycle, small releases
5.	Bringing in new developers (time and effort for their training), steep learning curve	Collective ownership (pros & cons), planning poker
6.	Change of management during the development process	Close relationship with customer
7.	Introducing new bugs as you develop software	40-hour week, collective ownership, short development cycle, small releases, tests, CI, pair programming

Solving Software Development Problems with Agile Practices* (contd.)

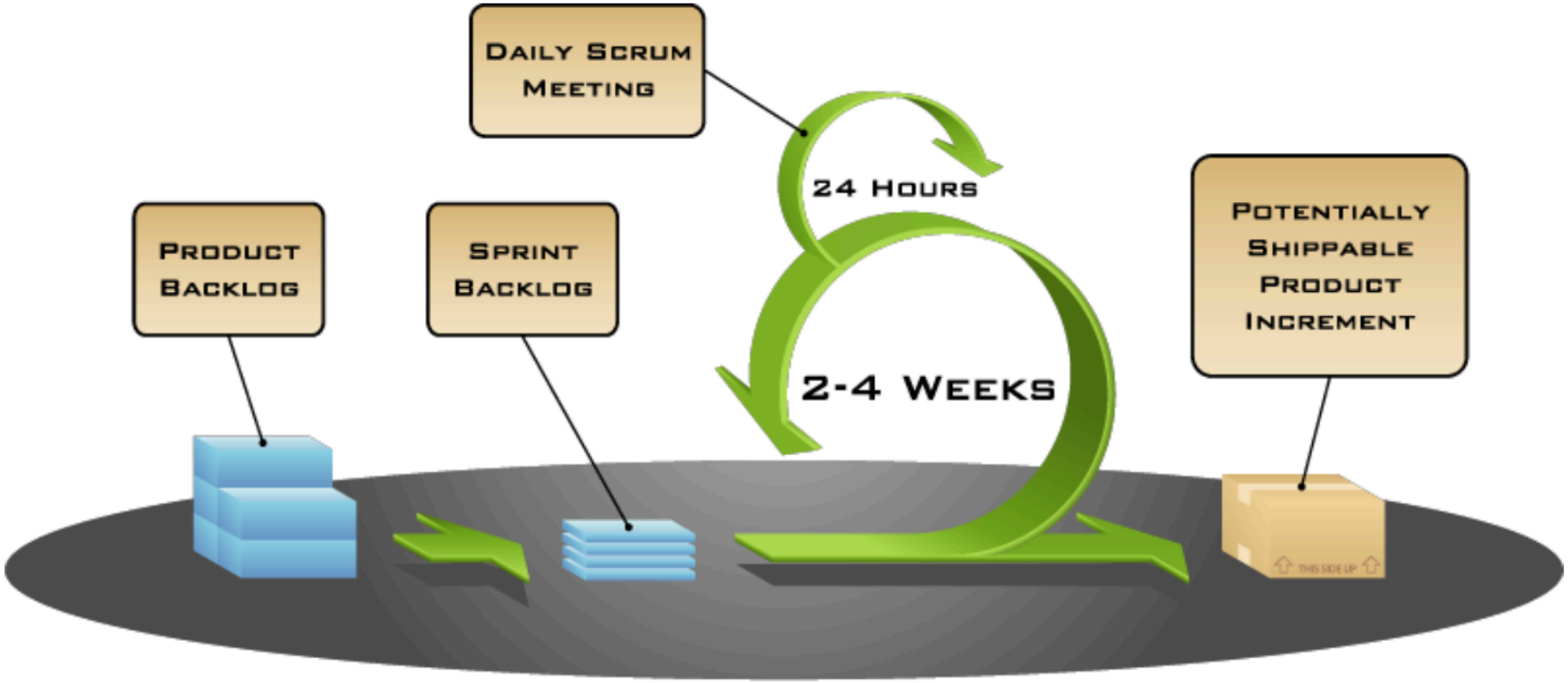
	Problem in Software Development	Agile Methods That Mitigate It
8.	Challenge of communication	Close relation with customer
9.	Developer turnover	Collective ownership (pros & cons), 40-hour week
10.	Integration issues	Collective ownership
11.	Difficulty of tracking bugs	Collective ownership, short development cycle, small releases, CI, tests
12.	Disagreement between developers	Close relation with customer
13.	Scheduling problems (global team)	Close relation with customer
14.	"Groupthink" (tendency of developers to agree with one another, common thinking among them), fear of hurting the feelings of other developers	Planning poker, pair programming
15.	Challenges with integrating with legacy code	Collective ownership

* This is an expanded, but still not comprehensive list.

Scrum

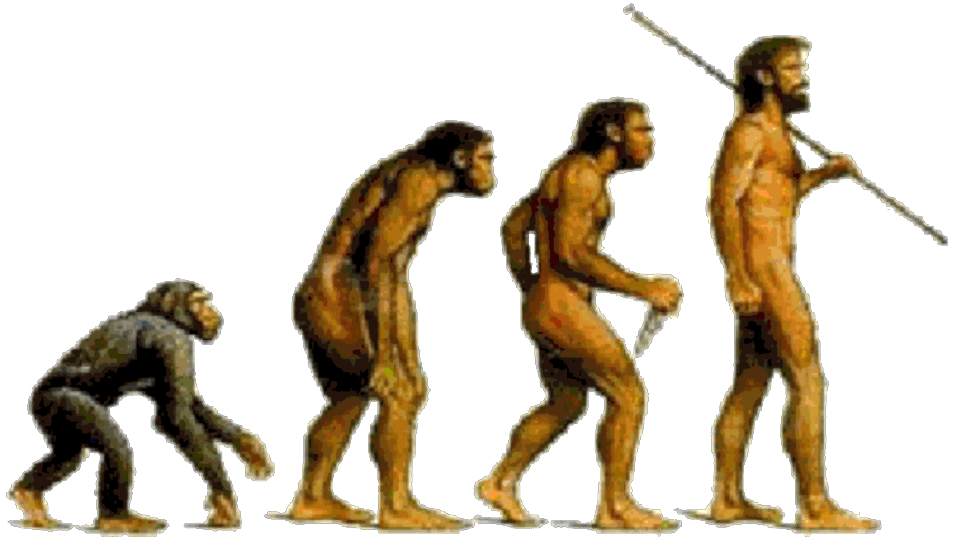
Customer, team, scrum master

Scrum Process

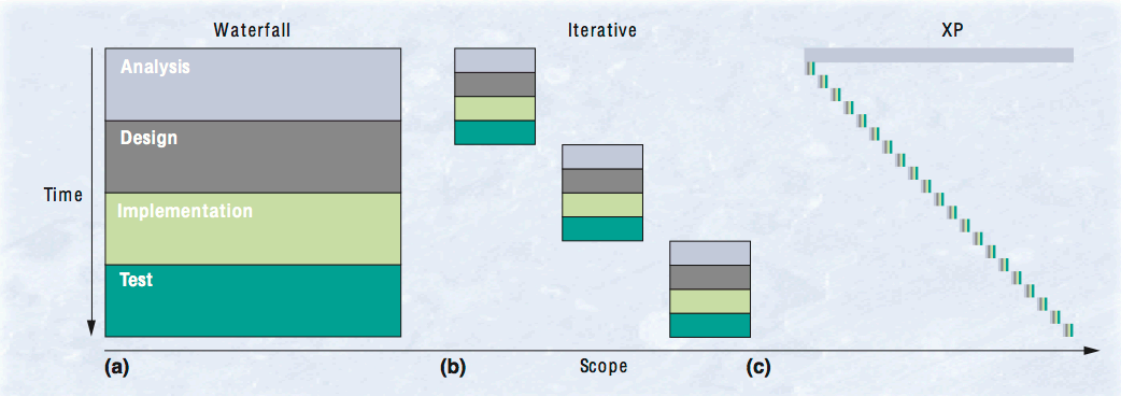


Extreme Programming (XP)

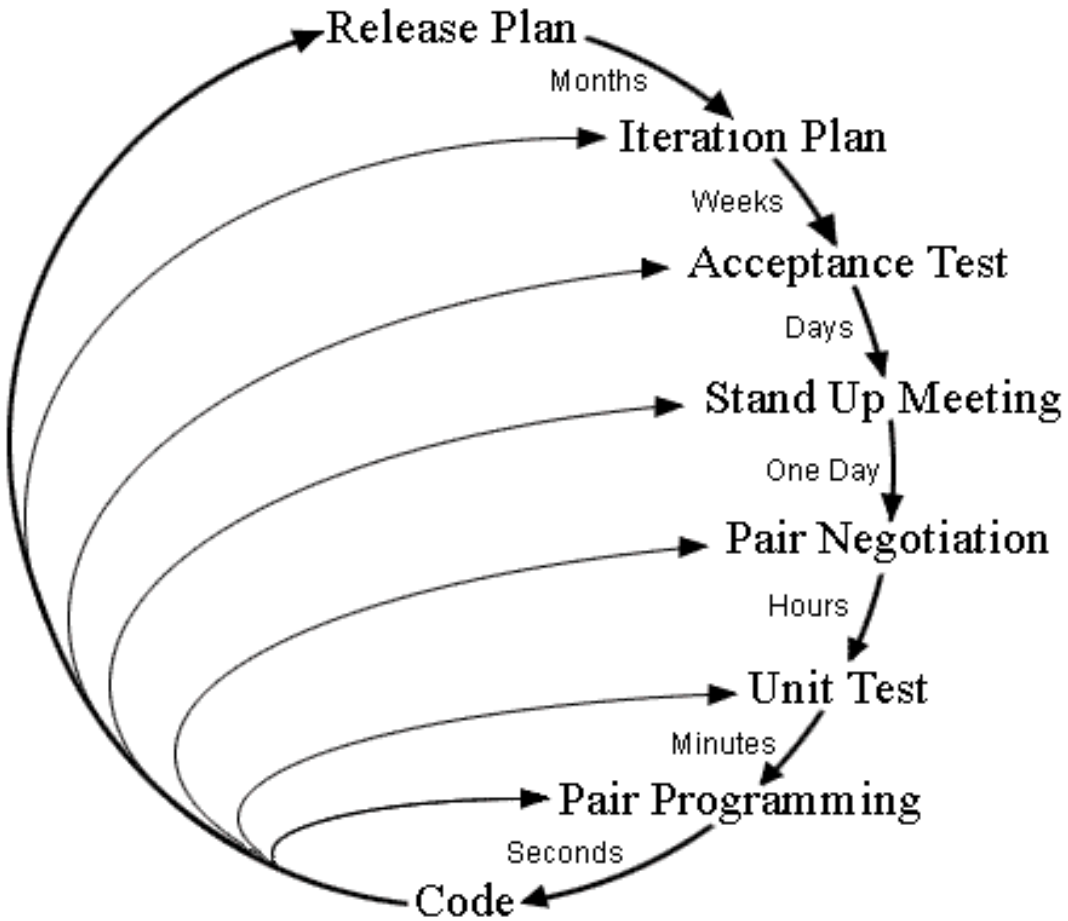
Human evolution



XP evolution



Extreme Programming (XP)



XP Practices (subset of Agile!)

- TDD (test-first approach).
- Planning game: 1-3 week iterations, one iteration at a time, customer decides which user stories to use
- Whole team/on-site customer: “customer speaks with one voice.” Customer may be a whole team.
- Small releases, with valuable functionality, to guard against unhappy customers.
- System metaphor is a single shared story of how it works. (Sort of like architecture)
- Simplest thing that possibly works (coding for today)
- Refactor all the time, because you don’t have up-front design before programming.
- Collective ownership. Everyone is responsible for everything. If a programmer sees something she doesn’t like, she can go change it. Task ownership is individual.
- Pair programming. can code alone for nonproduction code like prototypes
- Continuous Integration. A day of development at most.
- Sustainable pace. 40 hour work weeks.
- Coding standards, Especially since all code can change at all times.

Universal Credit

CASE STUDY

The Twelve Principles of Agile Software Development

Individuals and interactions

1. Projects are built around motivated individuals, who should be trusted
2. Face-to-face conversation is the best form of communication (co-location)
3. Self-organizing teams

Working software

4. Working software is delivered frequently (weeks rather than months)
5. Working software is the principal measure of progress
6. Sustainable development, able to maintain a constant pace
7. Continuous attention to technical excellence and good design
8. Simplicity—the art of maximizing the amount of work not done—is essential

Customer collaboration

9. Customer satisfaction by rapid delivery of useful software
10. Close, daily cooperation between business people and developers

Responding to change

11. Welcome changing requirements, even late in development
12. Regular adaptation to changing circumstances