Advanced Software Design
Software Project Management Methods

JonJon Clark
clrjon005@myuct.ac.za
slides and materials taken from prof edwin blake & Melissa Densmore

THE PROBLEM

The Problem
Traditional SE Methods
Modern Alternatives
Iterative SE Methods
Unified Process
Comparison & Conclusion

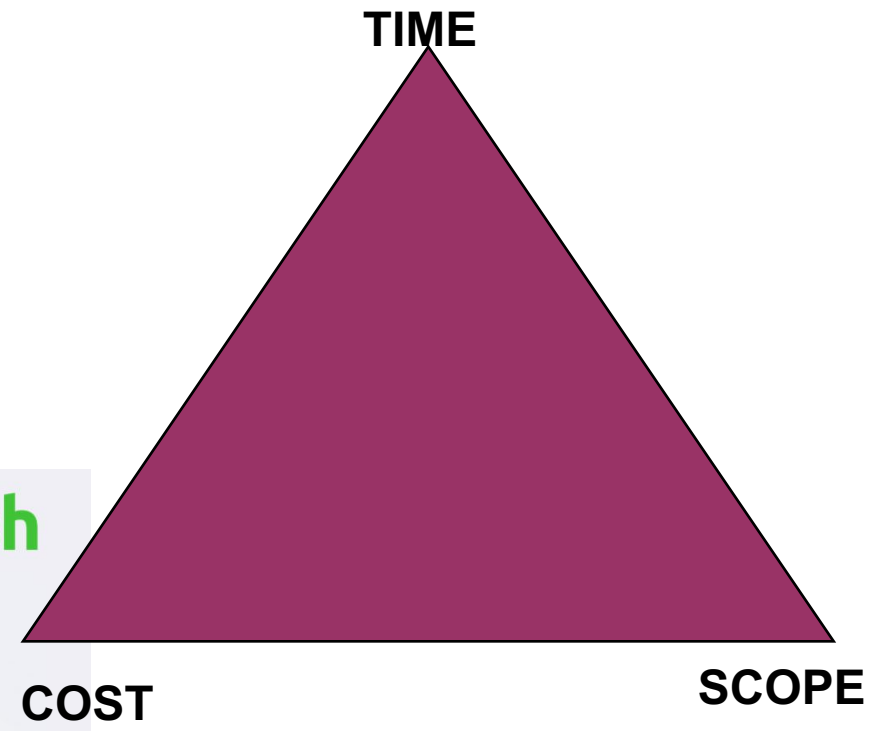
Observations

- Most common problem in software systems is not the construction, but the estimation.
- Software projects **fail to meet cost and schedule**, because those targets are wrong.
  - Costing software is difficult
- Know little about accurate estimations so targets are unreasonable
  - made by people least able to make them
    - e.g., marketers, managers and customers.
- Communication is hard when the ideas are abstract or conceptual.

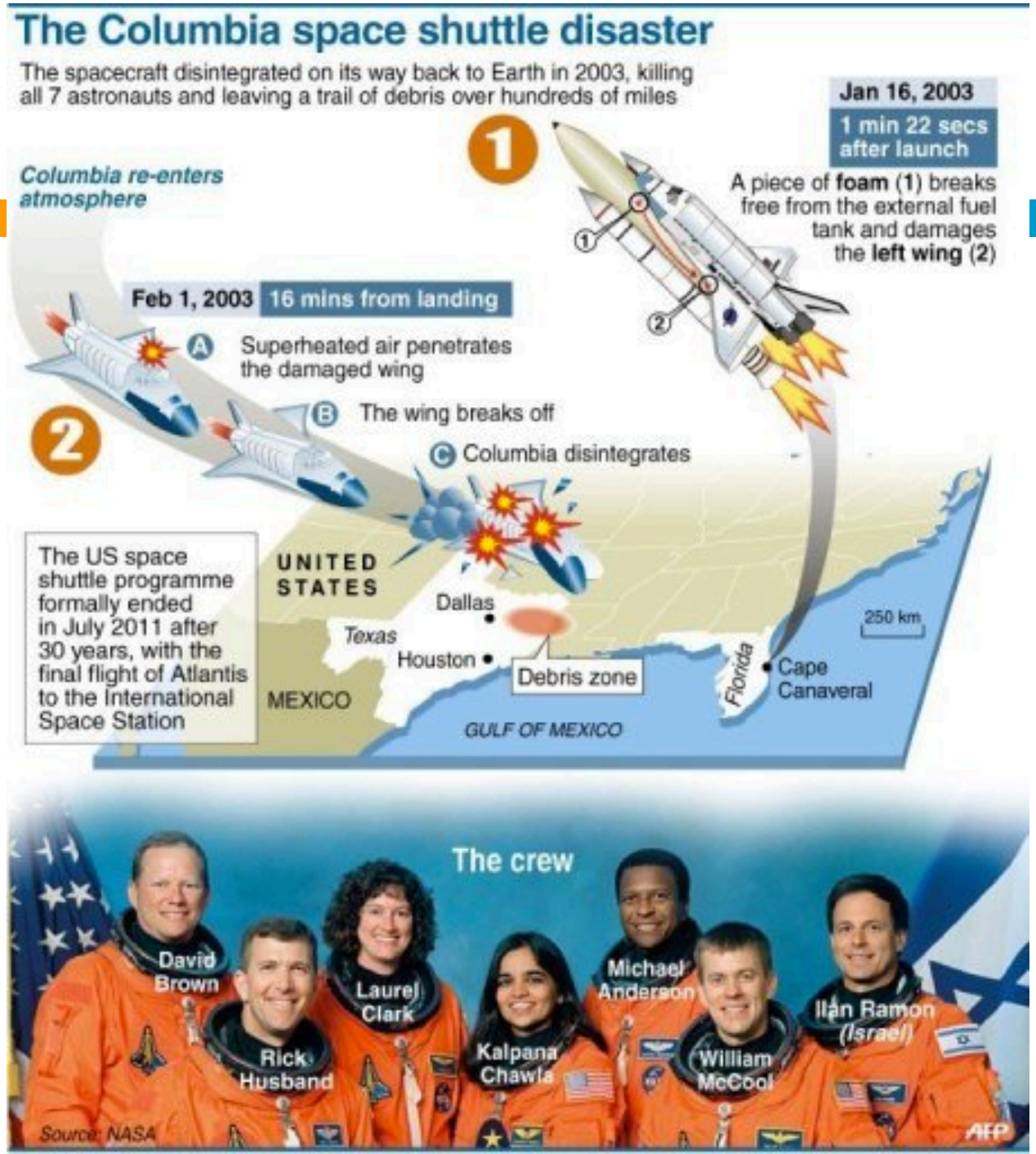
Software Engineering Triangle

- 1990s, NASA adopted the slogan:

  “Faster, cheaper, better.”



<- Lies



Faster,
better, cheaper?

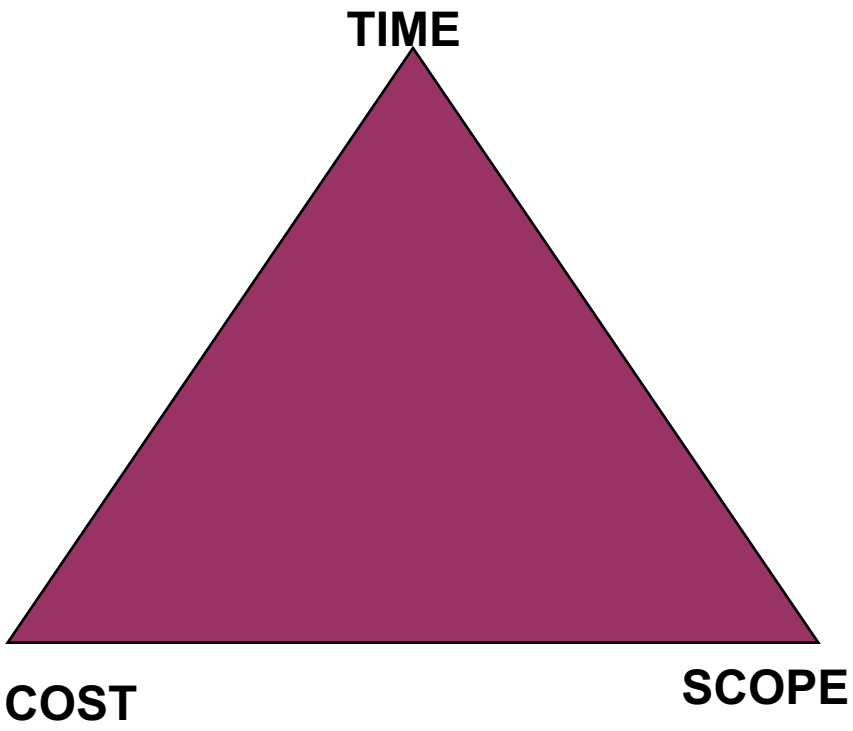
Software Engineering Triangle

- 1990s, NASA adopted the slogan:

  “Faster, cheaper, better.”
- Following a series of unsuccessful projects they abandoned the slogan.

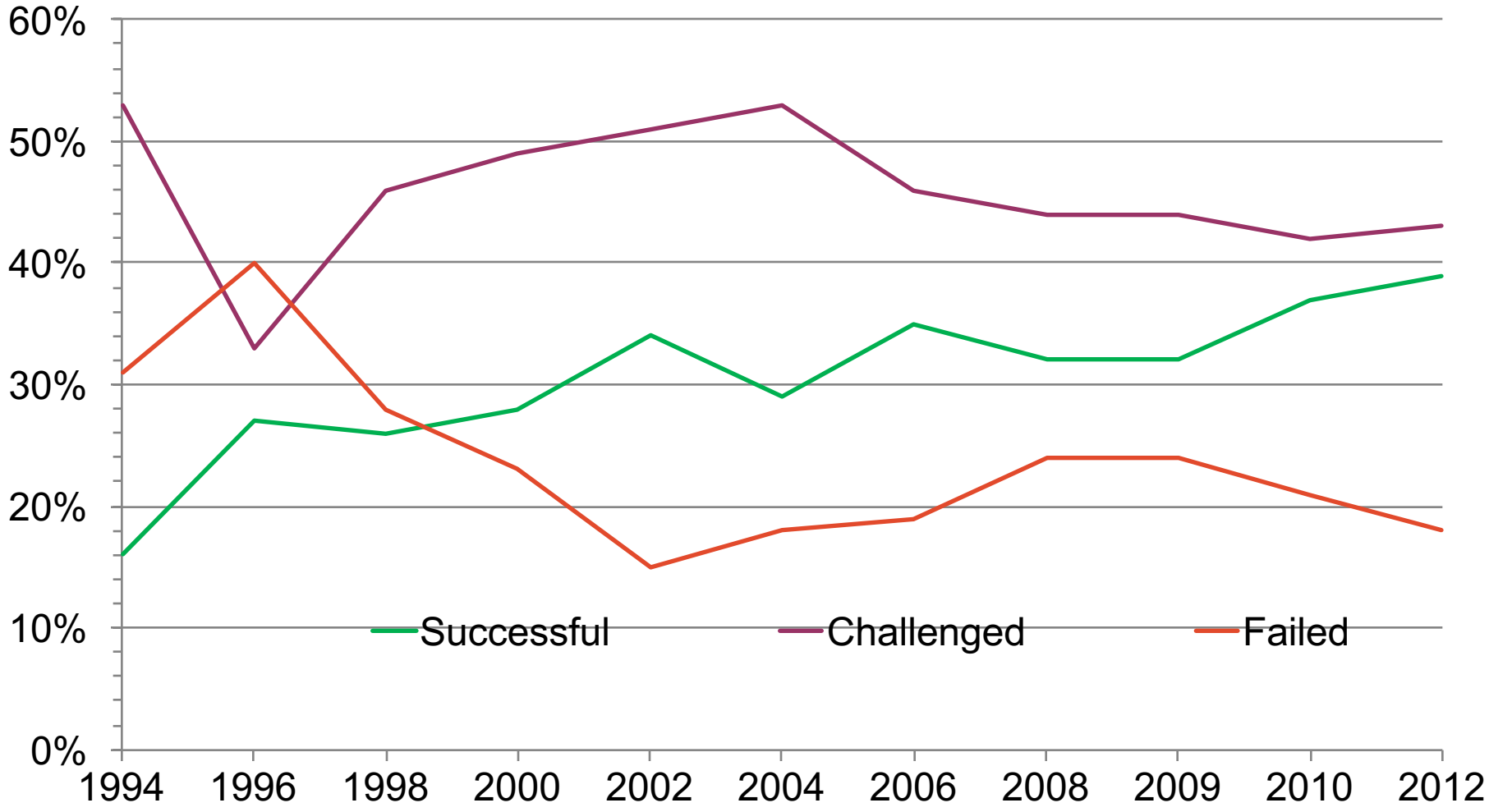
  “Faster, cheaper, better” is impossible to satisfy.
- Software engineers say that a correct statement of NASA's praiseworthy goal is

  “Faster, cheaper, better: pick two out of three.”

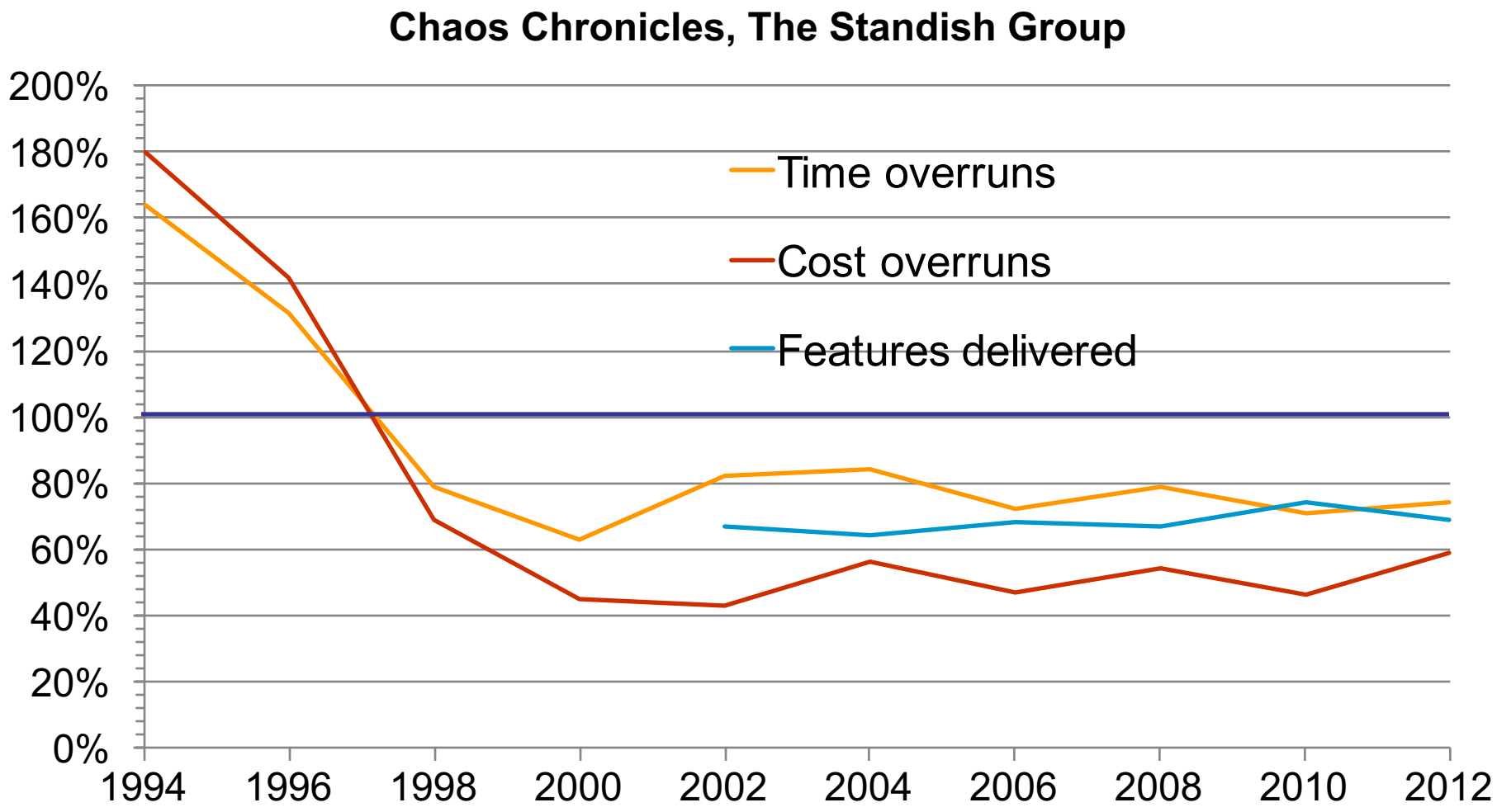


Any change to one goal must be compensated for by a change to one or both of the other goals.

Standish Group Chaos report: Project Resolution History



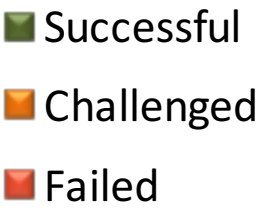
Time and Cost Overruns plus % Features Delivered



Resolution by Large and Small Projects

Small Projects

Large Projects



Small projects: less than $1 million in labour content
Large projects: more than $10 million in labour content.

CHAOS MANIFESTO 2013
Standish Group
www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf

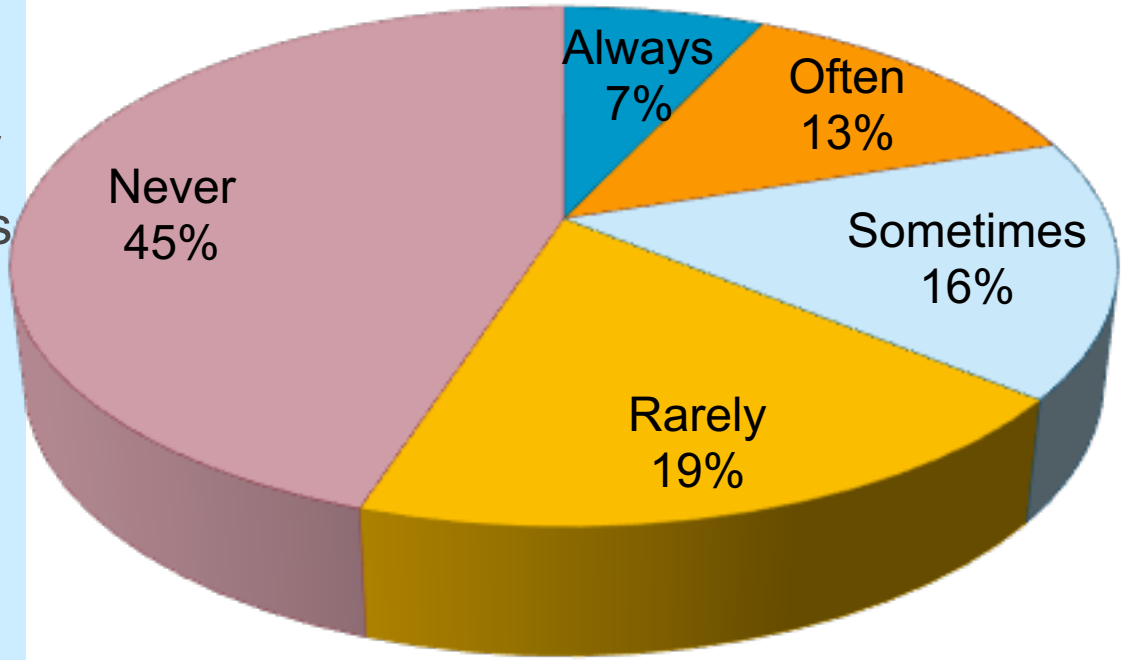
"Think Big, Act Small": Just say no (to large projects)

- Secret to project success: enforce limits on size and complexity
  - Size and complexity trump all other success factors.
- Break large projects down into a sequence of smaller ones, prioritized on direct business value.
  - Use stable, full-time, cross-functional teams that follow a disciplined agile approach.
- Quick solution is to just say no to large projects
  - more sensibly: adopt a small project strategy.
    - deliver software at lower cost and with fewer defects.
- Projects too often get too big to succeed.
  - Constantly being called on to do more for less
  - But the real key to success is doing less for less
    - splitting large projects into a sequence of small ones

Wasted Effort

More than 45% of features are never used, while another 19% are used rarely

- Almost 2/3 of the features are never or rarely used!
- Stop developing these features and double productivity!



Note: there has been increasing criticism of Standish methods. See Eveleens and Verhoef, "The Rise and Fall of the Chaos Report Figures" IEEE Software, **27**, 1, 30–36, Jan.-Feb. 2010, Doi: 10.1109/MS.2009.154 or www.cs.vu.nl/~x/chaos/chaos.pdf

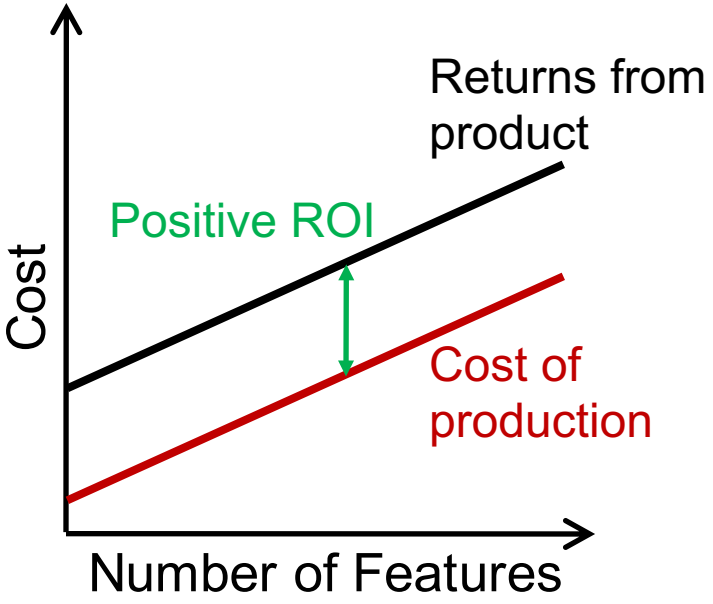
"… the Standish figures for individual organizations don't reflect reality and are highly influenced by forecasting biases. Because the underlying data has an unknown bias, any aggregation of that data is unreliable and meaningless."

YAGNI: You Ain't Gonna Need It

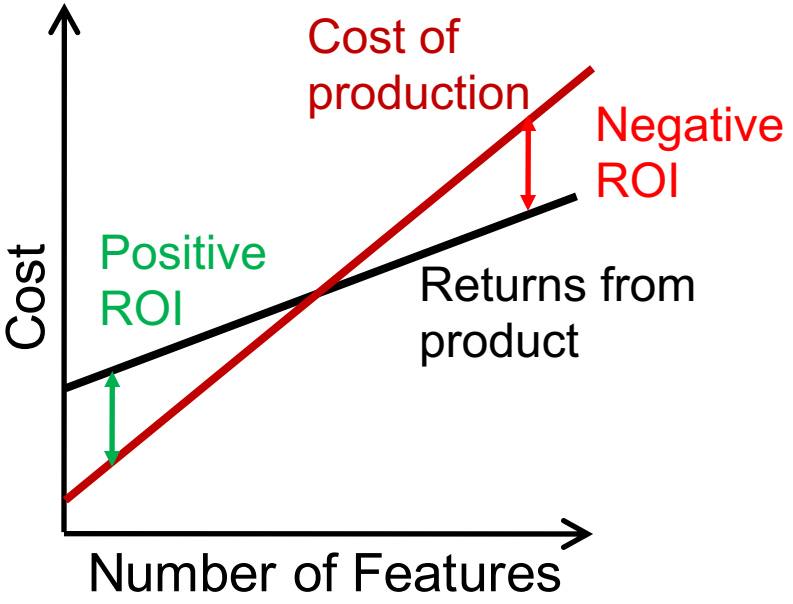
- Cry to prevent speculative development and Gold Plating (aka Bells and Whistles).
  - "I'm sure I'm going to need some additional functionality later, so I'll write it now."
- Better is to build only what you need now.
- Speculative development adds complexity to code prematurely.

Economies of Adding Features

Profitable Project: returns outpace costs

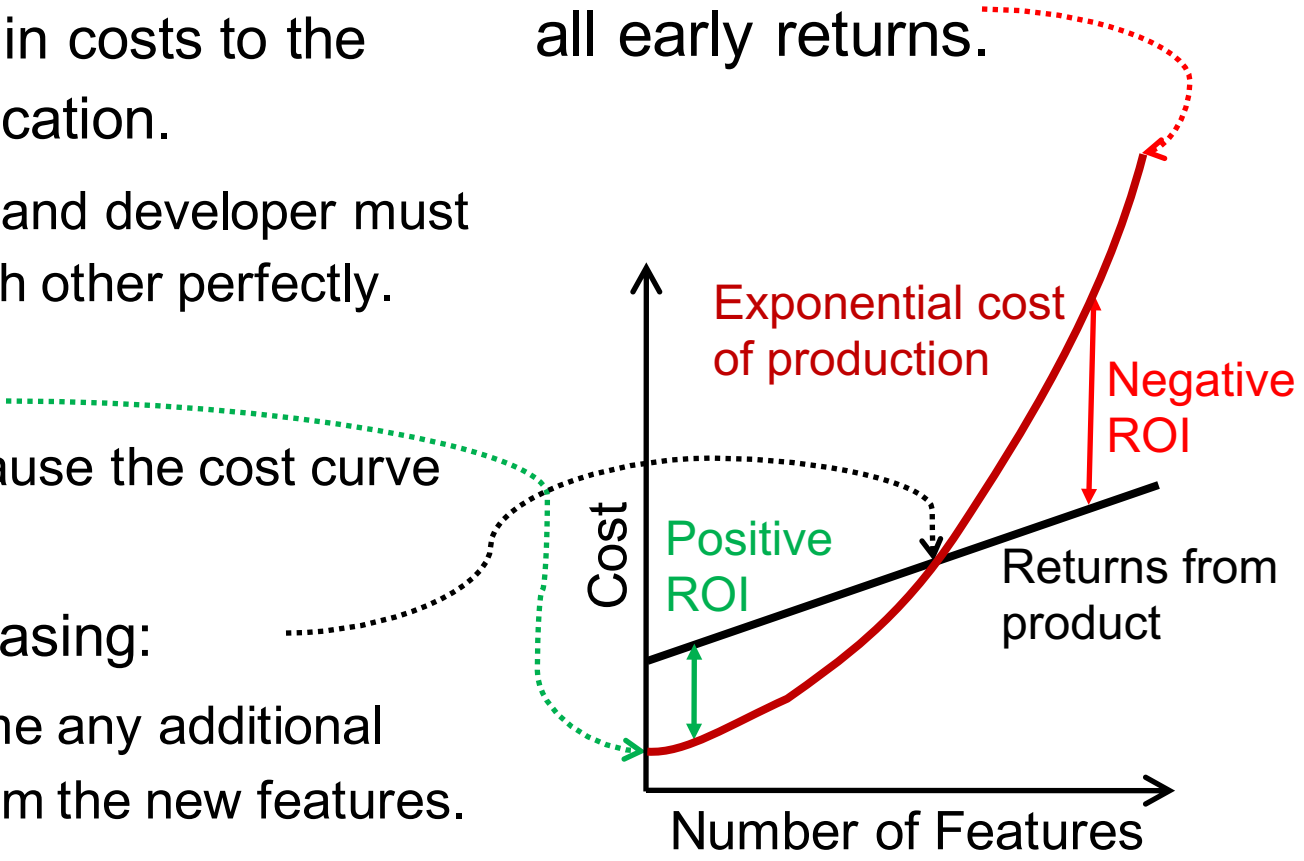


Ultimately Unprofitable Project: features become a drag



Cost Curve under most software processes is exponential

- Fred Brooks attributes the exponential rise in costs to the cost of communication.
  - E.g., Customer and developer must understand each other perfectly.
- New projects:
  - successful because the cost curve still flat.
- Costs start increasing:
  - quickly overcome any additional value added from the new features.
- Exponential costs overtake all early returns.



Problems: Bad Communication

AGE OF BOOTY: (Certain Affinity, Max Hoberman)

- “Over the course of the project there were numerous disconnects between the perceived state of the game and the actual state of the game.
- “The hardest hit were the designers, who continued fine-tuning plans for sophisticated features like matchmaking and party support long after the programmers had already made huge simplifications (and often cuts) to these systems. A combination of lack of attention to the project, poor communication, and wishful thinking led to the design team believing that several features were far more advanced than we were actually able to implement, and they did not find out the reality until very late in the project.”

Problems: The Crunch

- Crunch is the side effect of other problems and the cause of burnout.

Drawn to Life (5th Cell, Joseph Tringali)

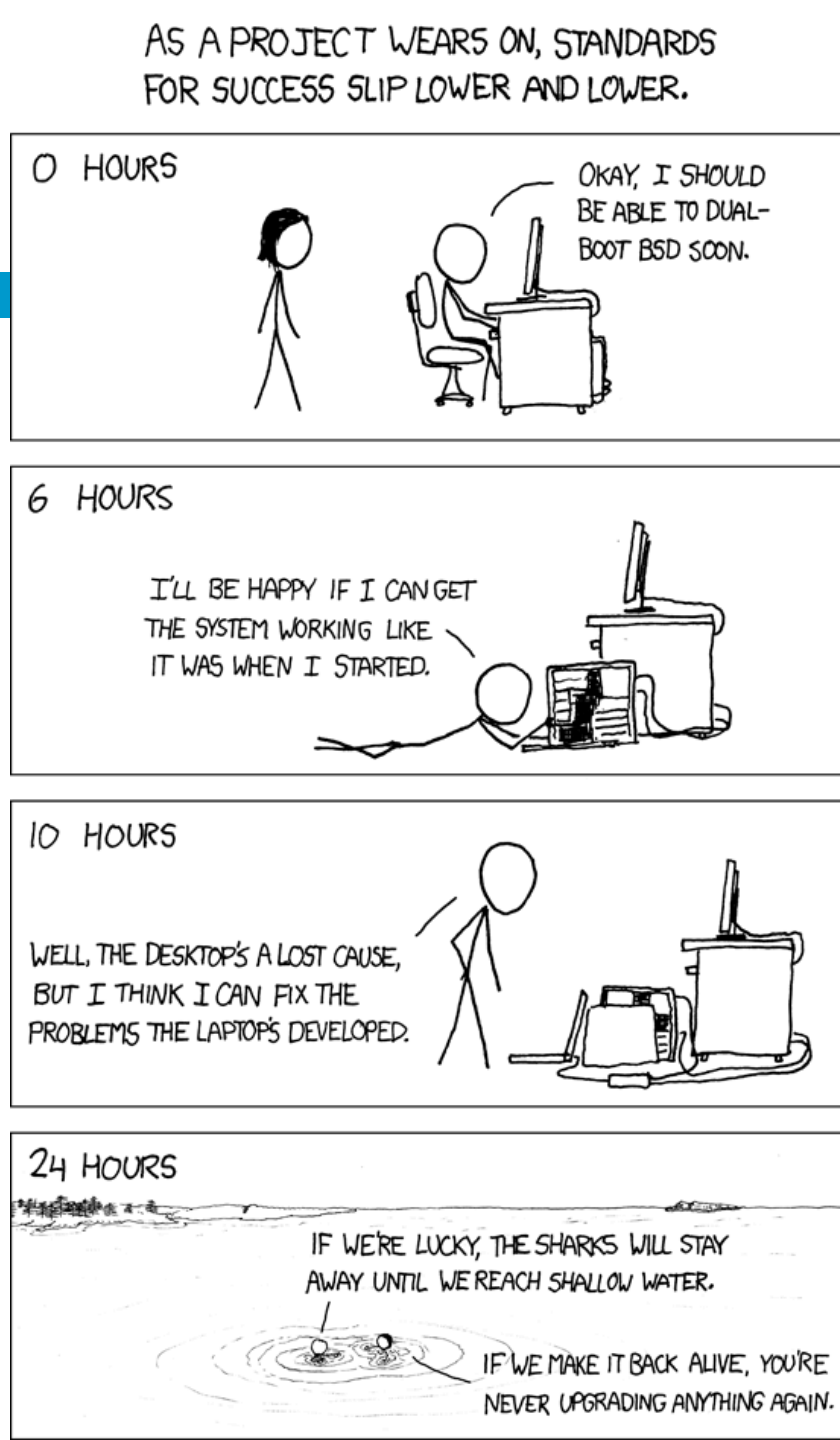
"Consistent overtime is not fun, nor is it something that should be assumed when developing a game. We ended up working late weekdays and Saturdays for the entire second half of the project.

"Our studio had the perfect storm of factors that contribute to crunch—lack of experience on the platform, an ambitious game, and a schedule that was too short. Combined with so little pre-production, we were playing catch up from day one, taking far too many shortcuts to implement the required game features."

Software Entropy, Rot & Geriatrics

- Entropy is a measure of disorder in a physical system
- Software Entropy: measure of code complexity
  - Tends to increase over time.
  - Speculative development adds complexity at the start
  - Bug fixes and enhancements increase complexity and degrade structure
    - most software applications grow at annual rates of 5%−10%
- Entropy makes it hard to
  - make changes and fixes
  - understand the code
- Cure for Entropy is:
  - **YAGNI** at the start and
  - **refactoring** as you go along.

https://xkcd.com/349/



Yak shaving—
official jargon for computer science

1. You want to generate documentation based on your git logs.
2. You try to add a git hook only to discover the library you have is incompatible and therefore won't work with your web server.
3. You start to update your web server, but realize that the version you need isn't supported by the patch level of your operating system, so you start to update your operating system.
4. The operating system upgrade has a known issue with the disk array the machine uses for backups.
5. You download an experimental patch for the disk array that should get it to work with your operating system: it works but causes a problem with the video driver. …

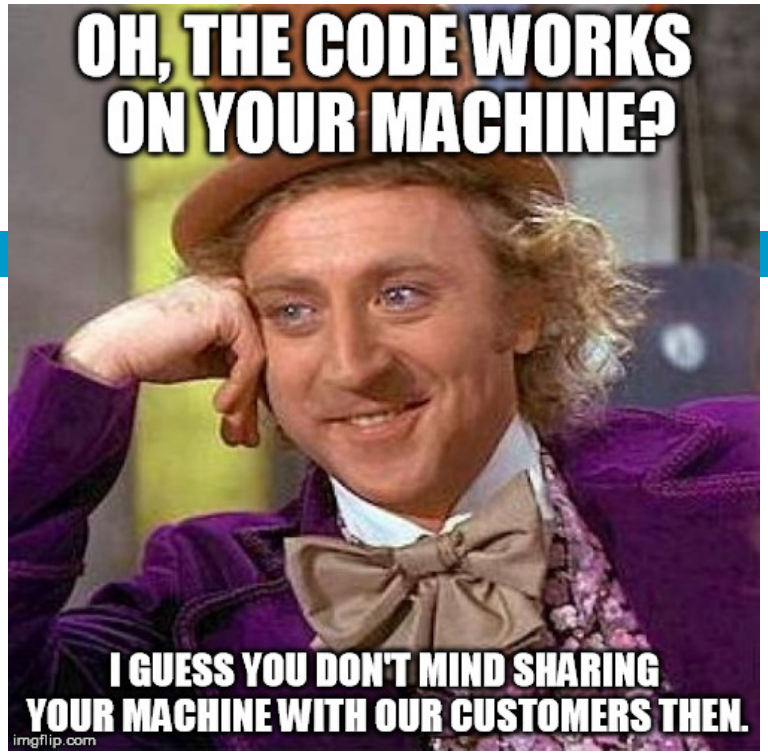
Stop: what got you started down this road? Try to figure out what shaving a yak has to do with generating documentation for Subversion logs.

- Yak shaving is dangerous because it eats up a lot of time. It also explains why estimating tasks is so often wrong: just how long does it take to fully shave a yak? Always keep in mind what you are trying to achieve, and pull the plug if it starts to spiral out of control.

Compromise if necessary
Explore alternate yaks

So we Have

- Undefined system
- Fixed resources
- Fixed time
- High quality



Goal: deliver software product to meet the clients needs, on time and within budget

Can we develop quality software under these circumstances?

- If so how?

TRADITIONAL SE METHODS

The Problem

Traditional SE Methods

Modern Alternatives

Iterative SE Methods

Unified Process

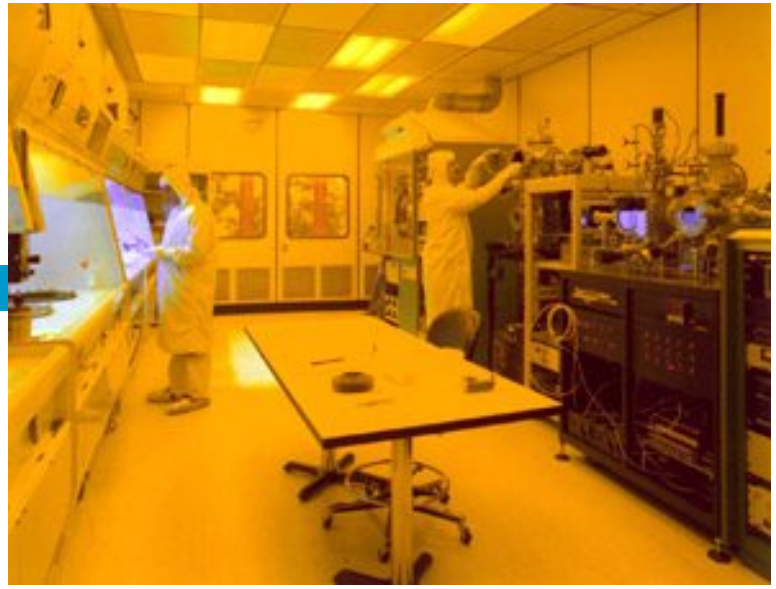
Comparison & Conclusion

Code and Fix?

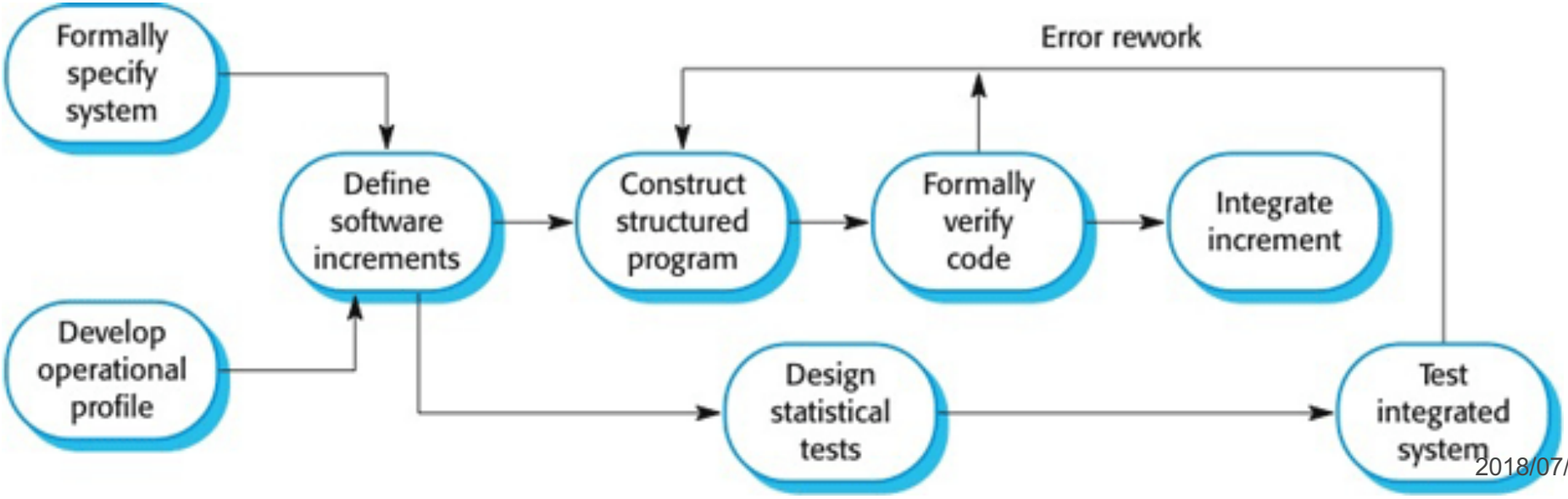
- Naïve, first approach
  - Actually lack of a methodology
- Little (zero) planning, dive straight into implementation
- Reactive
- End with bugs
  - If bugs multiply too fast to fix: "death spiral" → cancelled
  - To make it you'll have "crunch time"
    - May view as badge of honour, but results in burnout

Note to jonjon: Tell them about your Sudoku solver

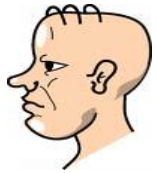
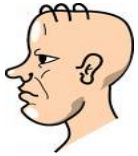
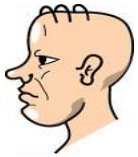
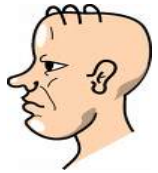
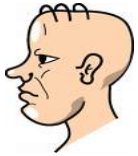
Traditional Methods



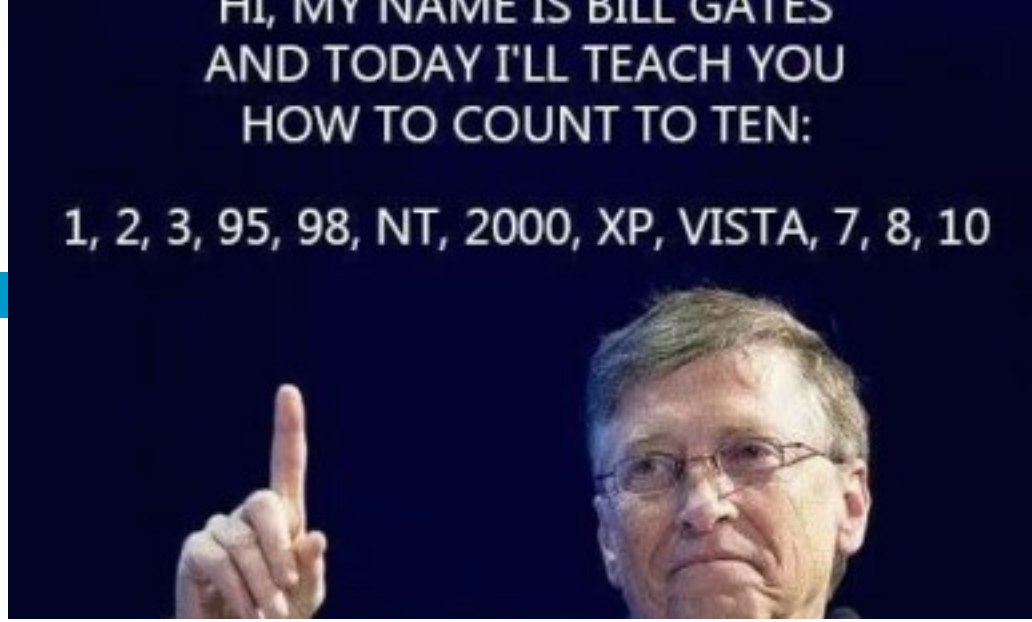
- Used for well defined systems
  - User can specify the requirements
  - Developers can then do the development
  - System is finished
  - System is launched
- For example see “Cleanroom Process”: www.SoftwareEngineering-9.com/Web/Cleanroom/

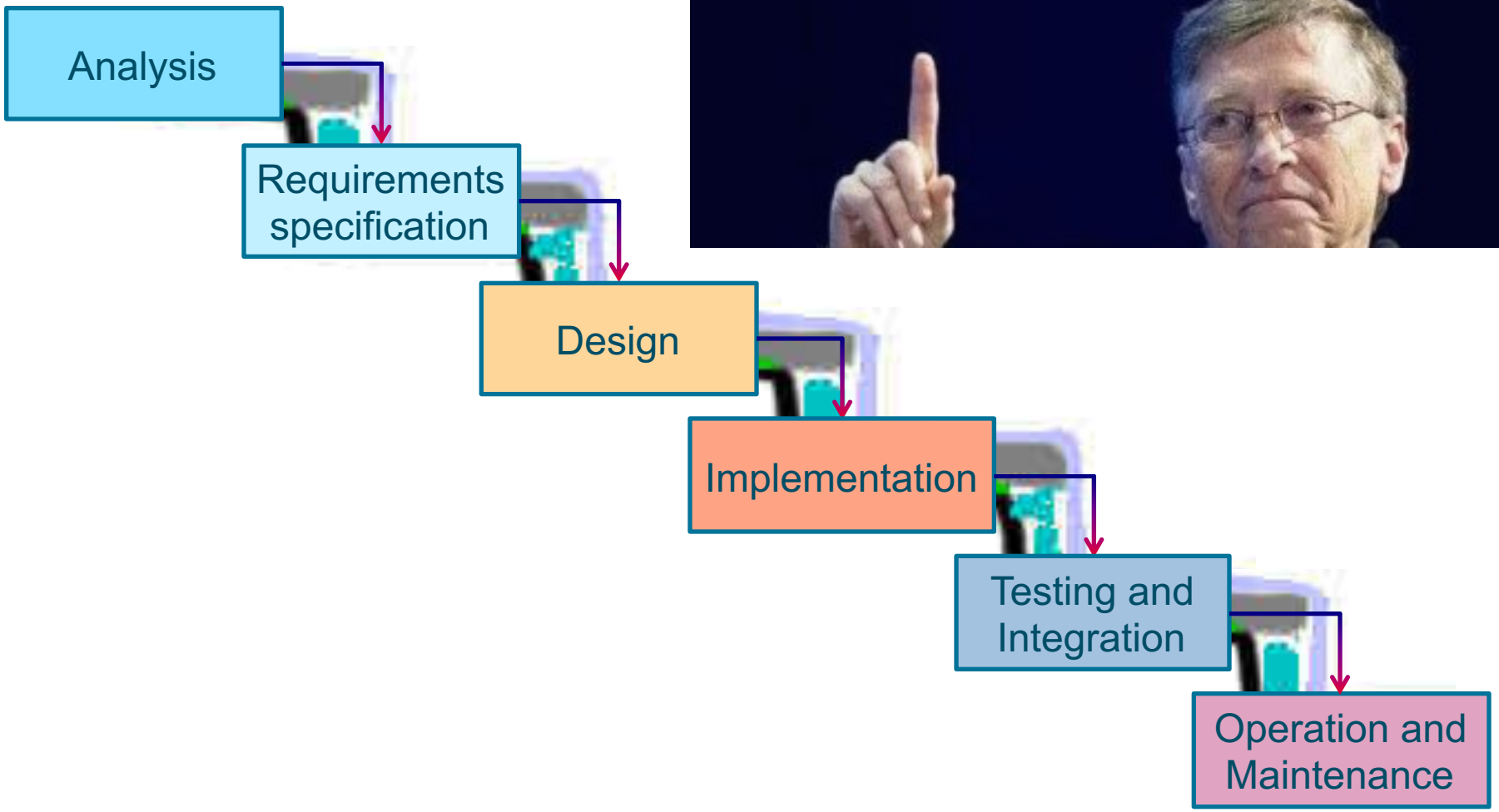


Software Development Conversation

Developer		Customer
	Sales	
	I have a need. →	
	← I can help. What is your need?	
	Requirements	
	What I need is … →	
	← I understand.	
	Design	
	If we build this will it meet your need? →	
	← Yes. Start work.	
	Build	
	What are you doing? →	
	← Programming. Please wait.	
	Test	
	I'm finished, is that what you wanted? →	
	← Let me check. Perfect!	
	Deploy	
	Let's launch it. →	
	← Call me if there are any problems.	

Waterfall Method Diagram



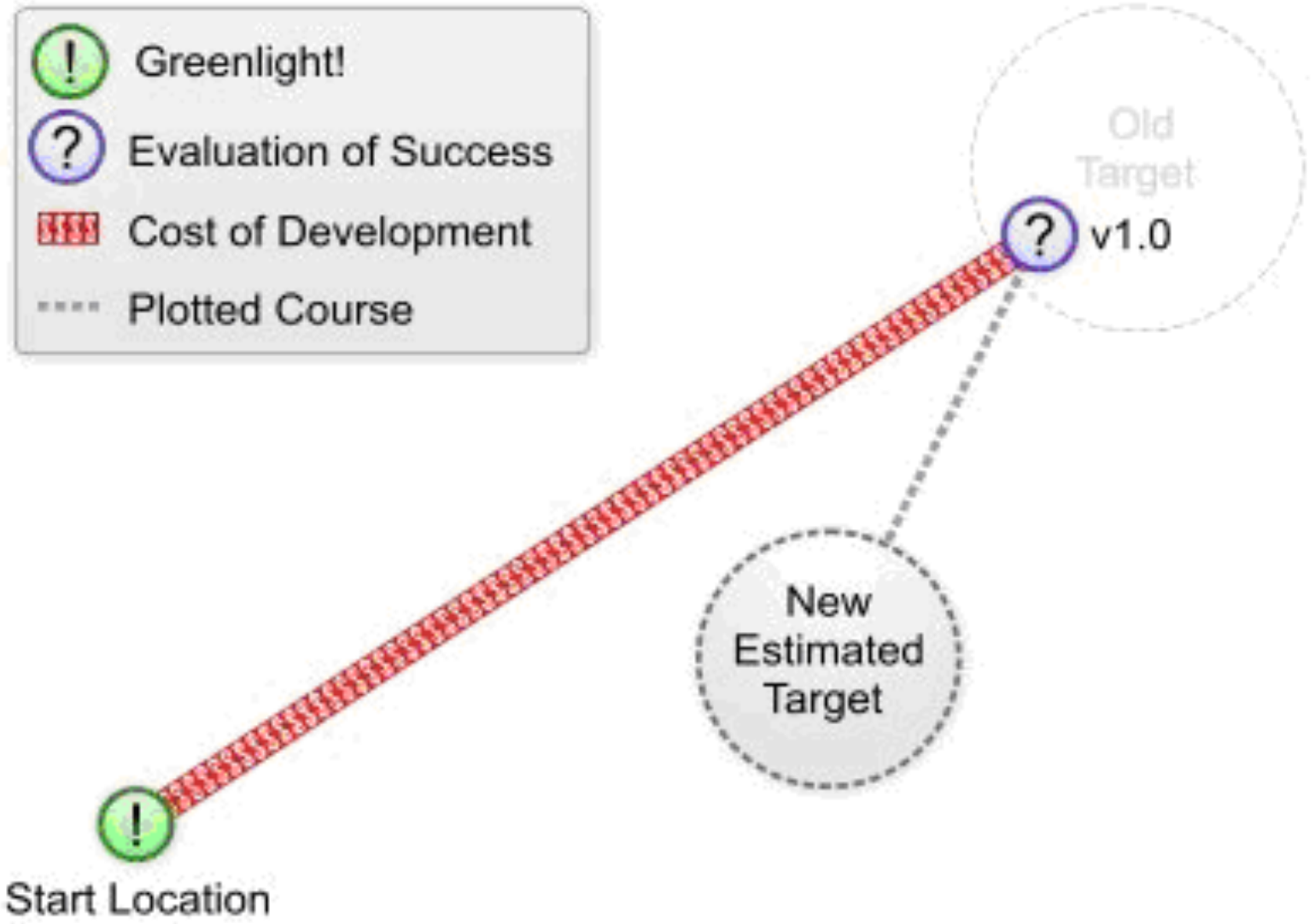


Waterfall Method

Waterfall Method

- Linear, sequential
- Teams gather requirements
- Develop the product
- Test it to see if they implemented the spec correctly.
- After release they gain insight into what the customer actually desired.
- Would be nice if it worked …

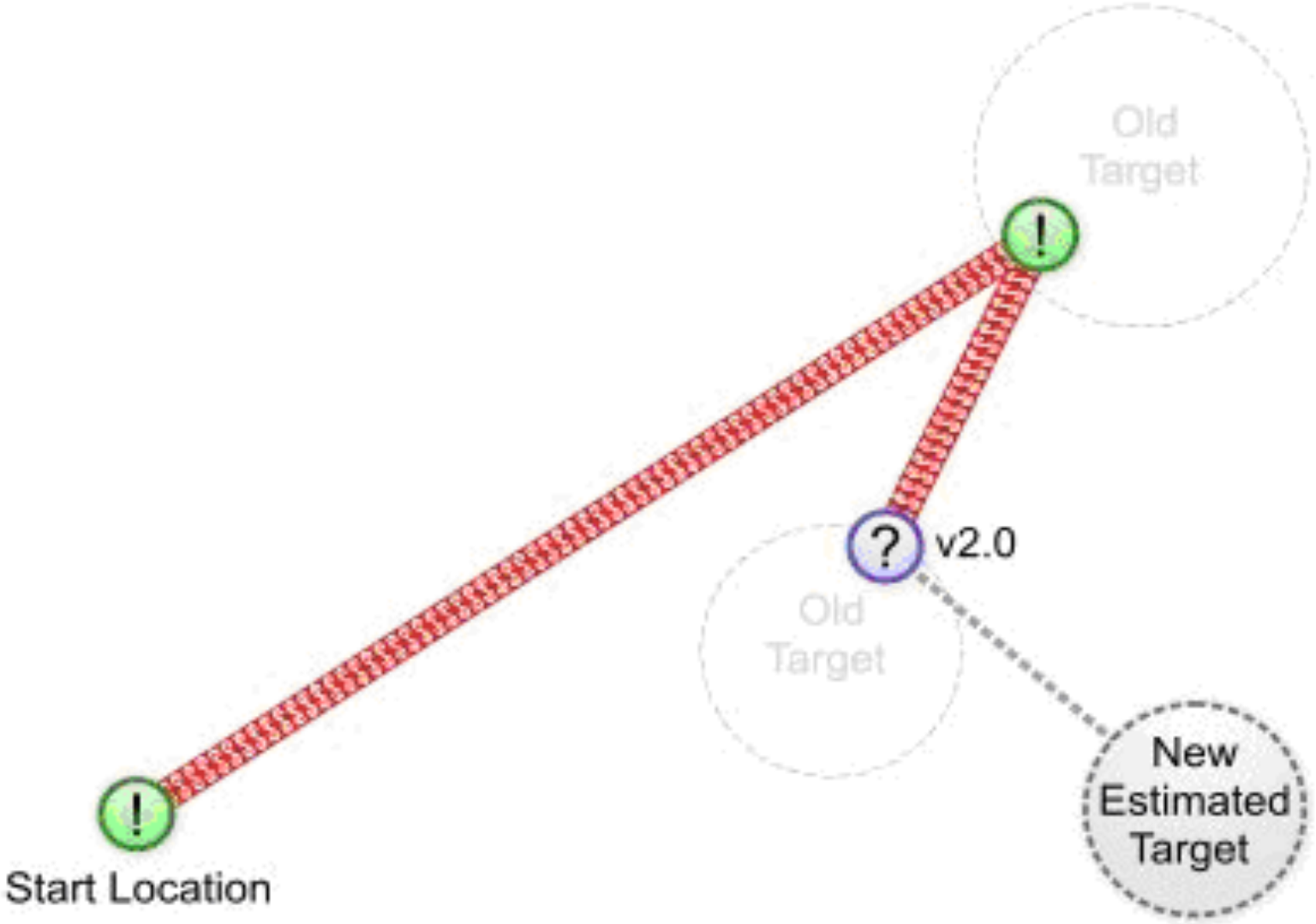
Waterfall — Version 1



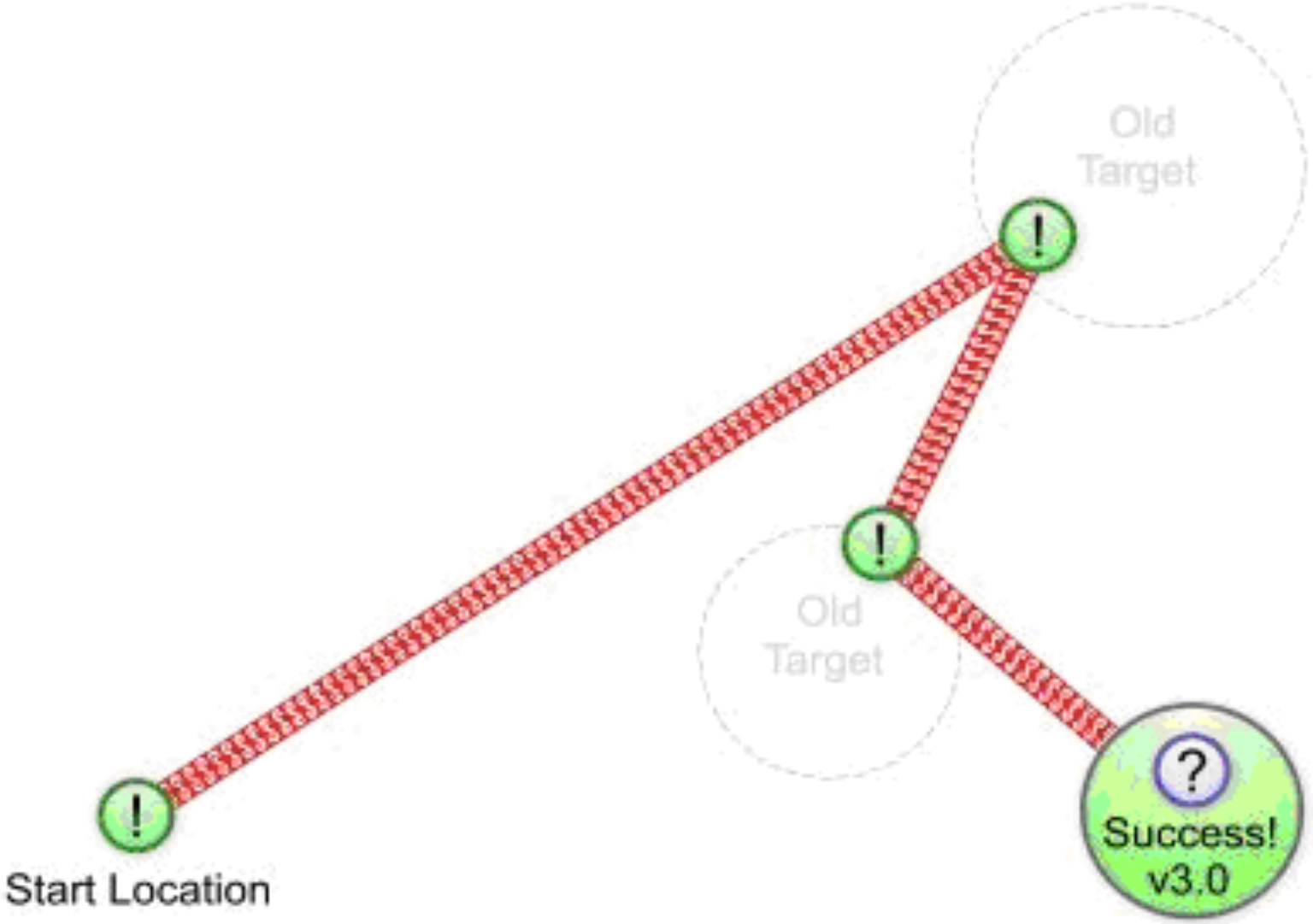
Waterfall — Version 2

- However the target has changed.
  - not all is lost.
  - given more money you can try again.
  - there is a good chance the team learned quite a bit about what their customers actually desired.
    - next rocket has a better chance of landing closer to the actual customer needs

Waterfall — Version 2



Waterfall — Version 3



Waterfall Concepts

- Software as an Engineering discipline
  - “Do it right the first time”.
- The more design time reduces risk
  - by planning upfront you identify problems early and avoid mistakes
    - the longer analyse a system, the more edge cases you’ll discover
      - often design elaborate systems for problems that do not really exist

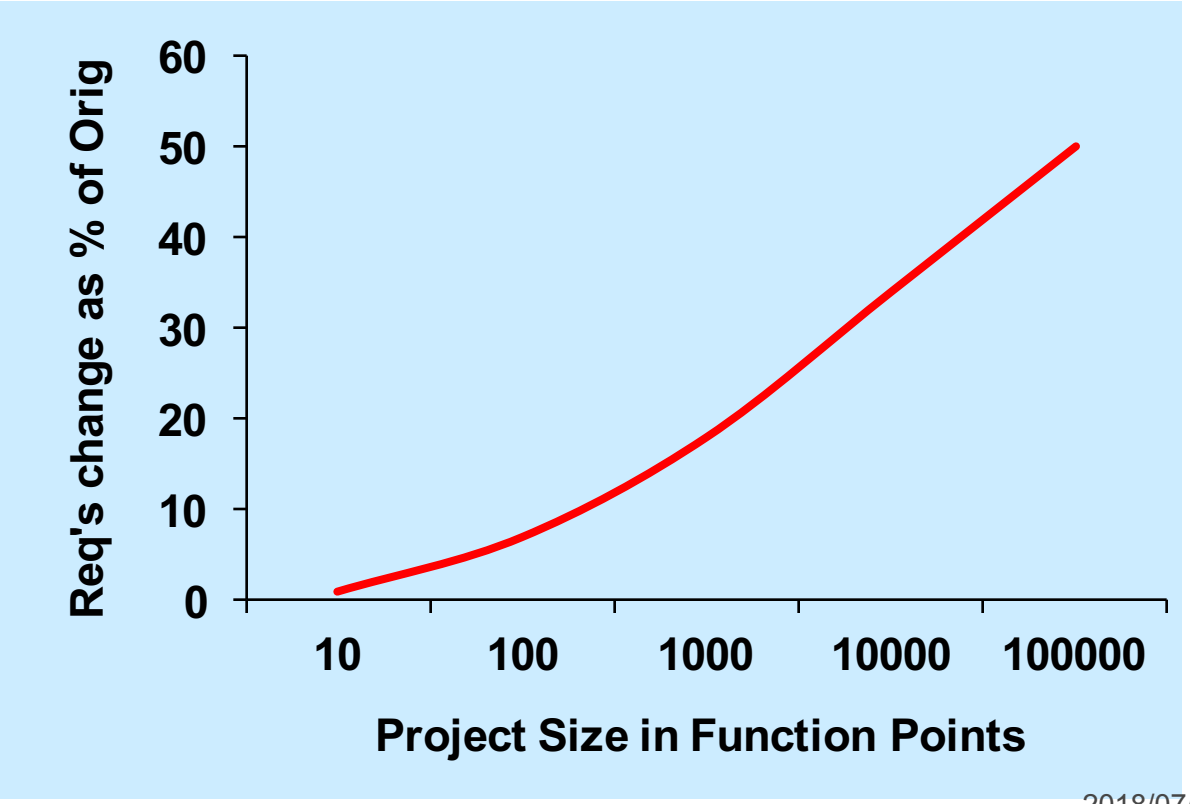
Change and the Waterfall Method

- The cost of change increases exponentially with time
  - conservative design decisions motivated by fear of change
    - a change late in the process costs 1000 times as much as a change early in the process
      - five minutes to write a spec
      - two days to program the feature
      - two weeks to test it before deployment
      - month to write a patch that fixes a problem after deployment

Change and Feasibility

- Is it feasible first to define the whole problem, then design the entire solution, then build the software, and then test the product?

From: *Applied Software Measurement*, Capers Jones, 1997. Based on 6,700 systems.



Waterfall

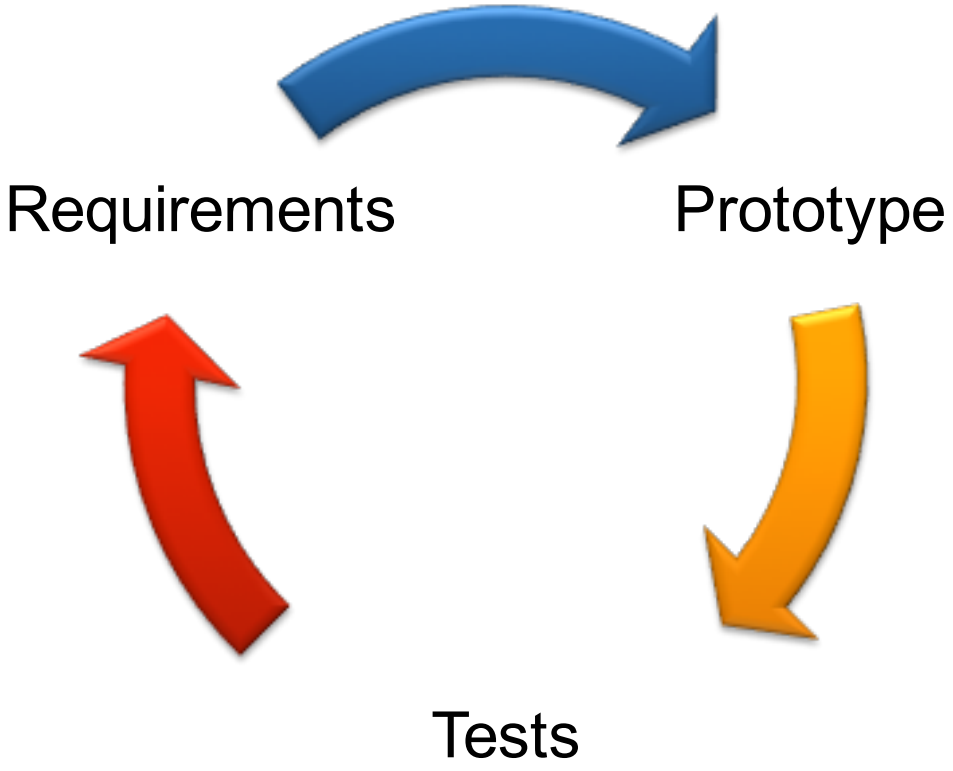
MODERN ALTERNATIVES

The Problem
Traditional SE Methods
Modern Alternatives
Iterative SE Methods
Unified Process
Comparison & Conclusion

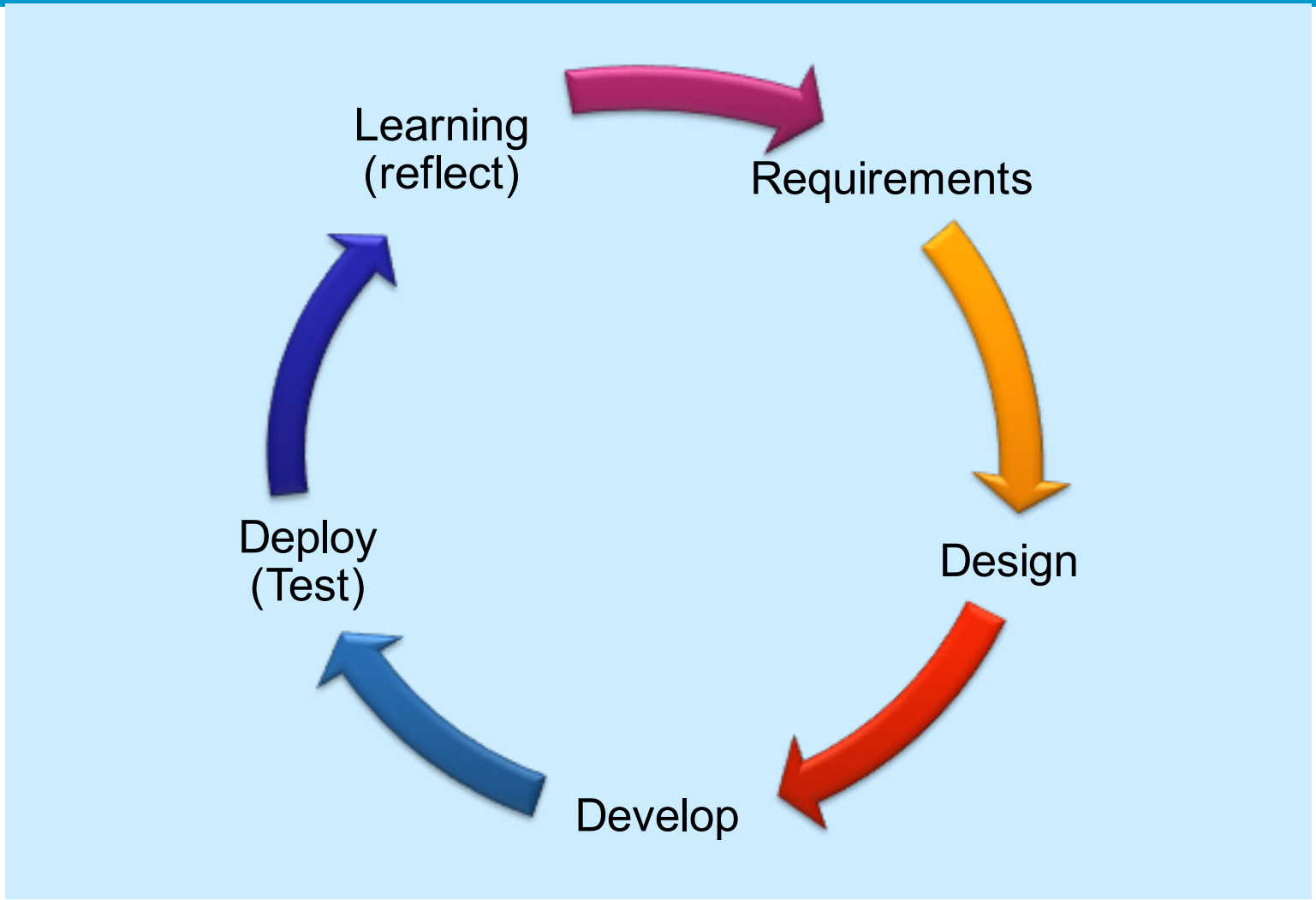
More Modern Processes

- More lightweight that waterfall
  - **Less documentation**
  - **Fewer procedures**
- **Don’t release only one version at the end**.
  - Parallel development
  - Produce of prototypes
- Only do what is required
  - No adding in extra requirements
- Design for change
  - Change is inevitable ensure you can handle it

Alternative Ideas: Prototyping

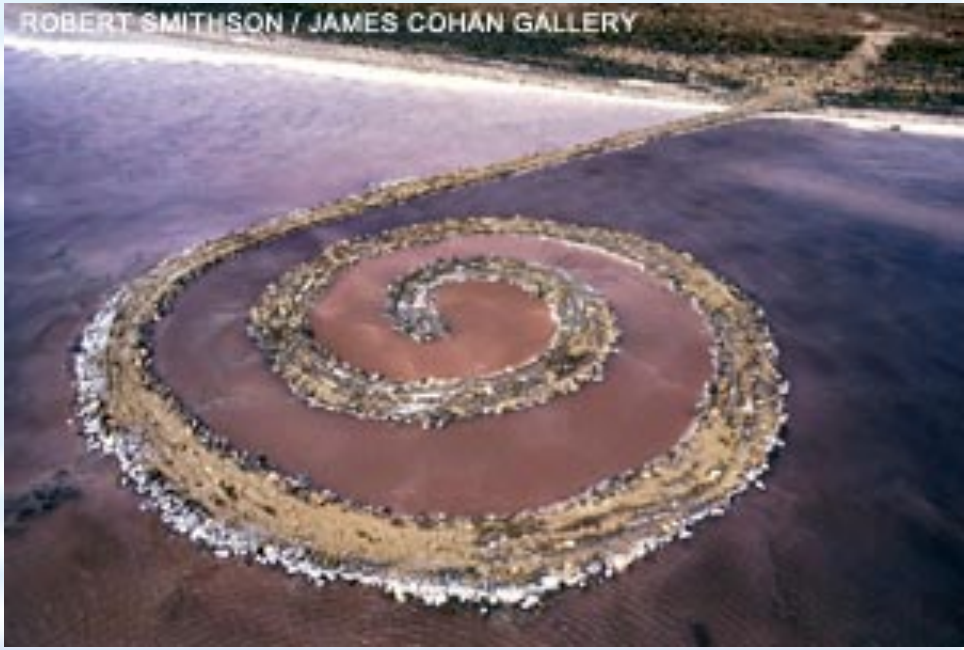


Rapid Application Development



ITERATIVE SE METHODS

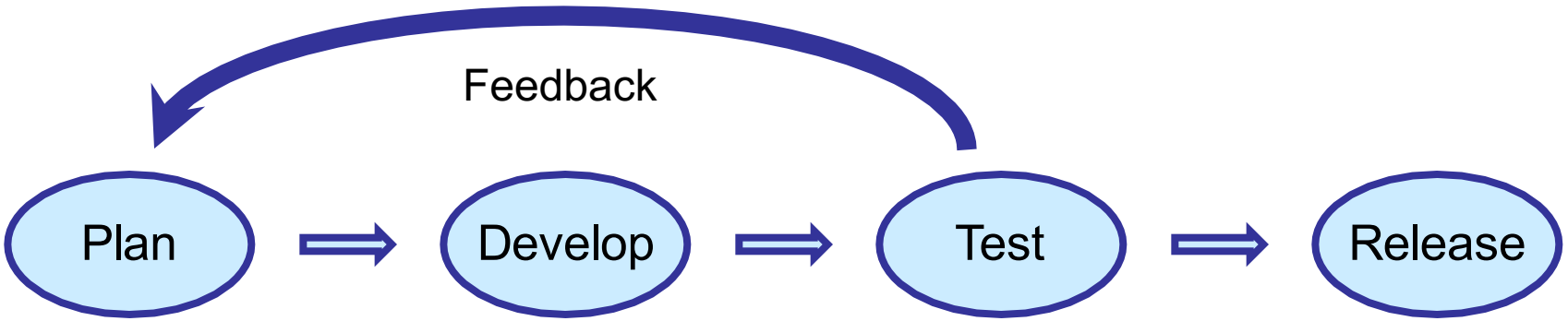
The Problem
Traditional SE Methods
Modern Alternatives
Iterative SE Methods
Unified Process
Comparison & Conclusion



Iterative or Incremental Process

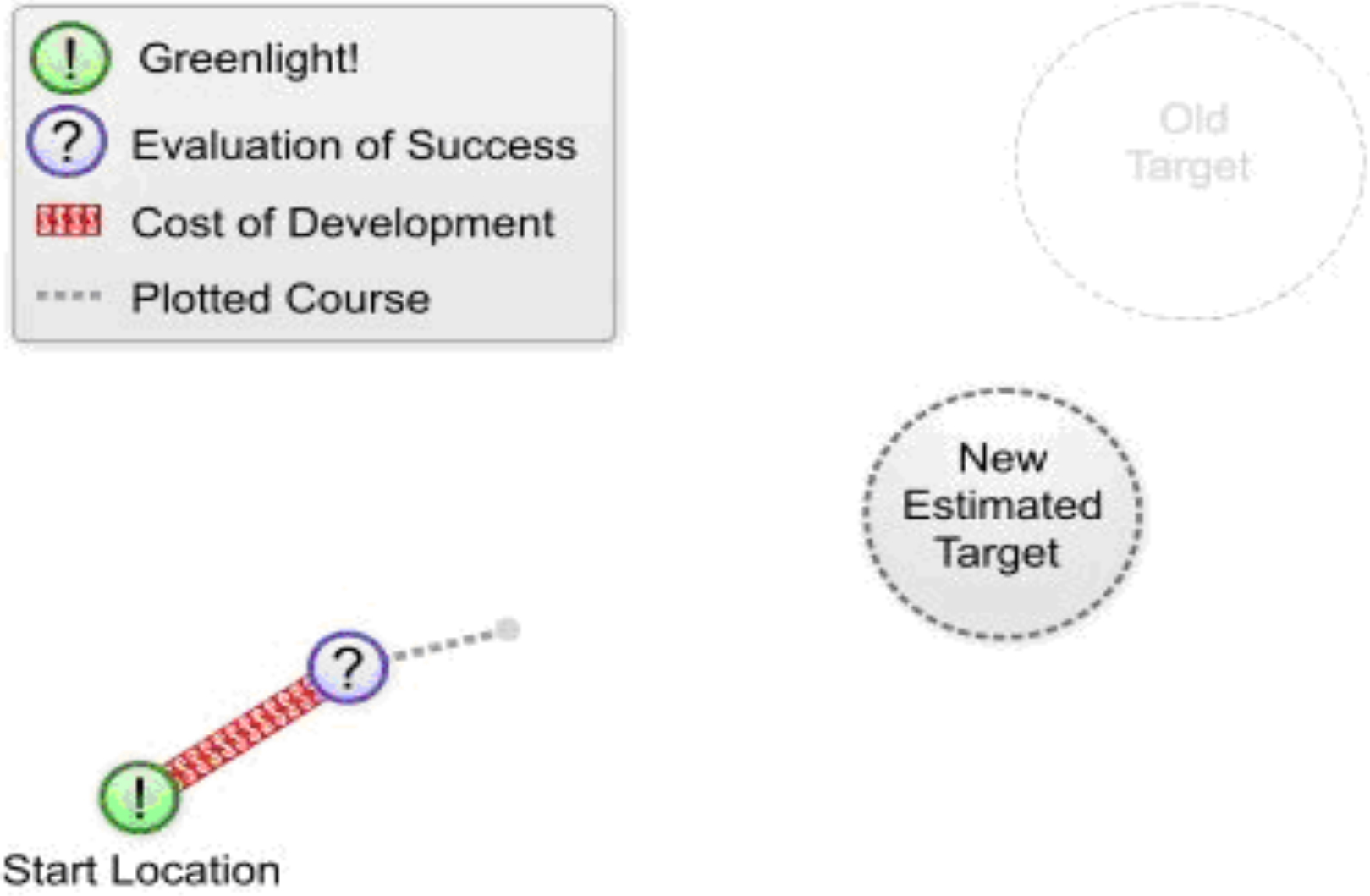
- Plan development
- Undertake development
- Generate a prototype
- Get user feedback
- Develop again

Iterative software development:
a process that reaches the goal in a series of ever improving delivery cycles



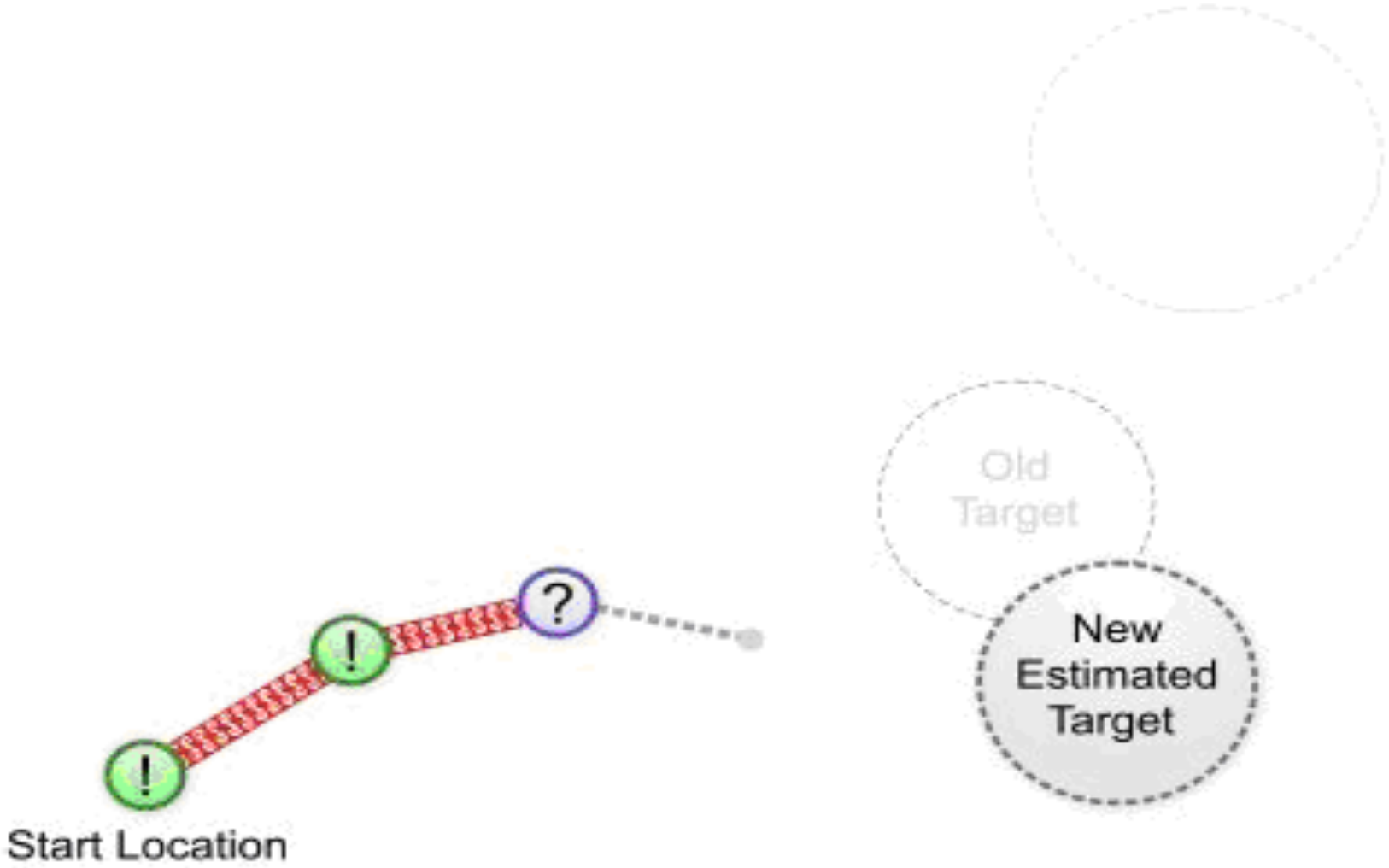
Iterative Process

Iteration 1



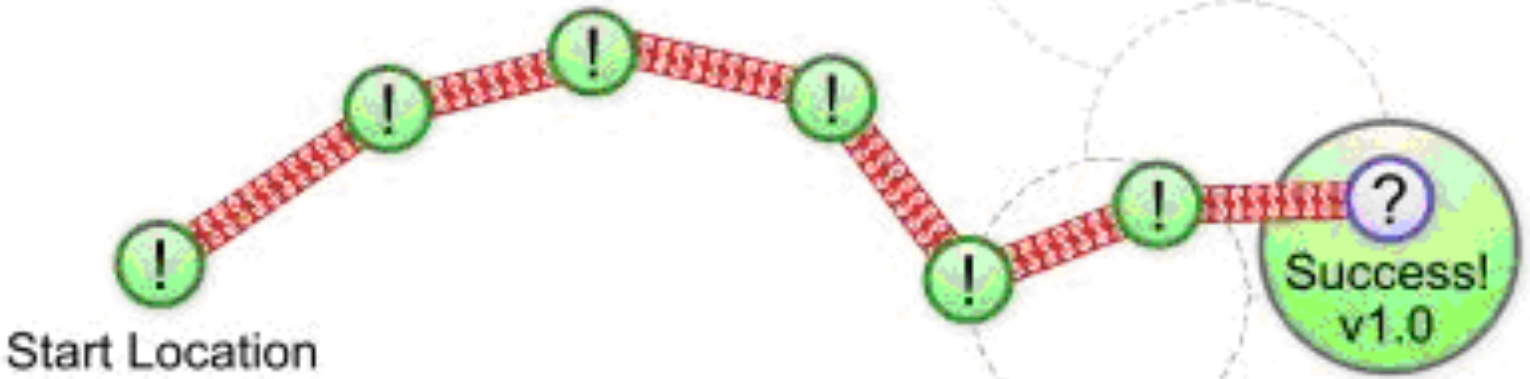
Iterative Process

Iteration 2

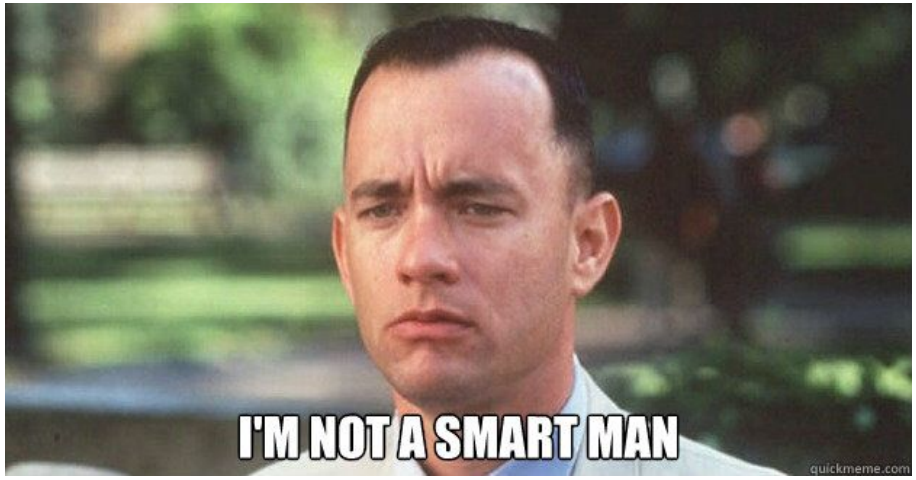


Iterative Process

Success



Waterfall — Version 3



Iterative Process — The Big Difference

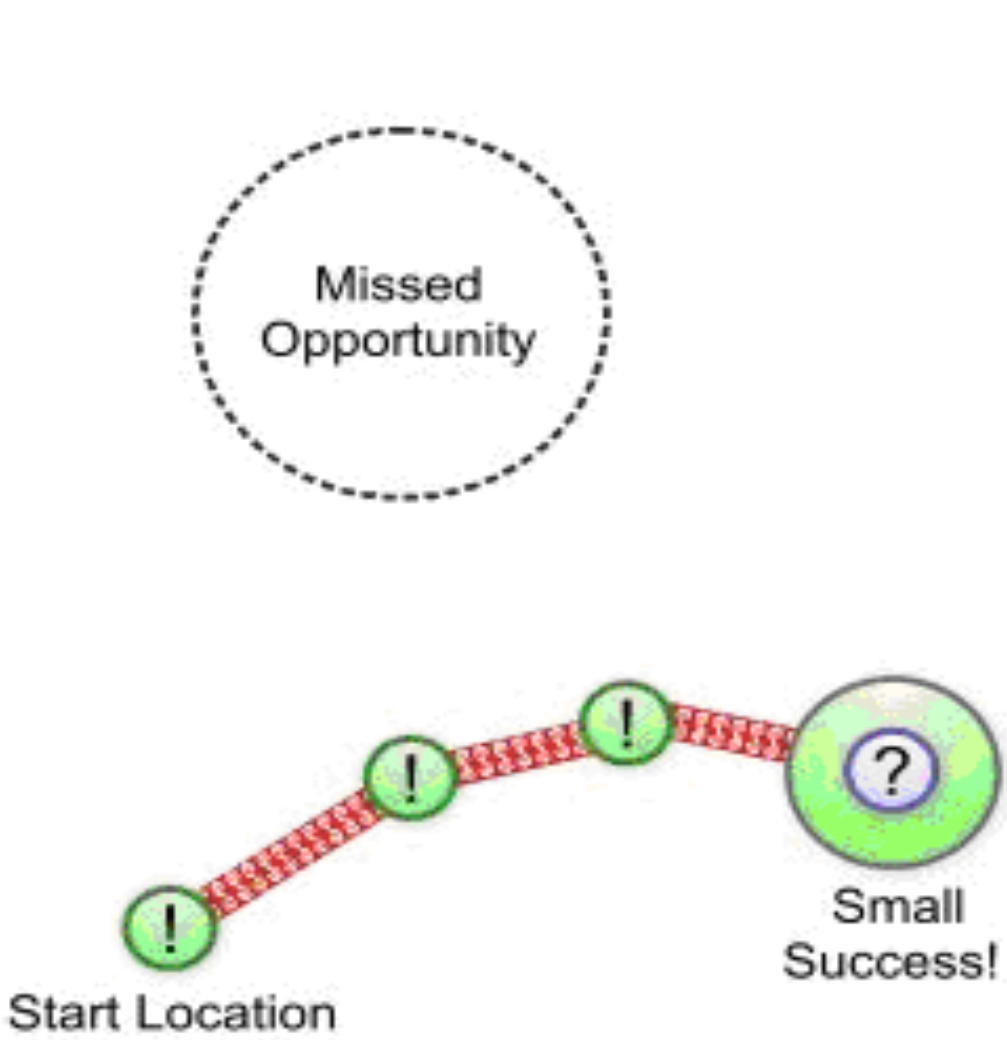
- Instead 2 to 18 months to create and evaluate a concept
- Build and show a new version to users every 2 to 4 weeks.
- Requires
  - **team members** close together and **close to customer**.
  - team members agree on good ideas over a period of hours, not months.
  - teams become experts through intense hands-on problem solving and testing.
- Ends up with real systems meeting the users' needs, not their “perceived” needs.

Note to jonjon: Tell them about your current ML projects

Iterative Process — The Benefits

- Favoured by small start up companies
  - greatly reduce the risk of a project failing.
    - only one shot at the target, steer your way to success using information instead of launching blindly into the unknown.
  - Long term, iterative development delivers more value sooner, with lower overall risk
- Project is intensely focused
  - in completing only high priority features, many alternative concepts never get explored.
    - good ideas can be lost

Iterative Process — The Danger



Examples of Iterative Software Development Methods

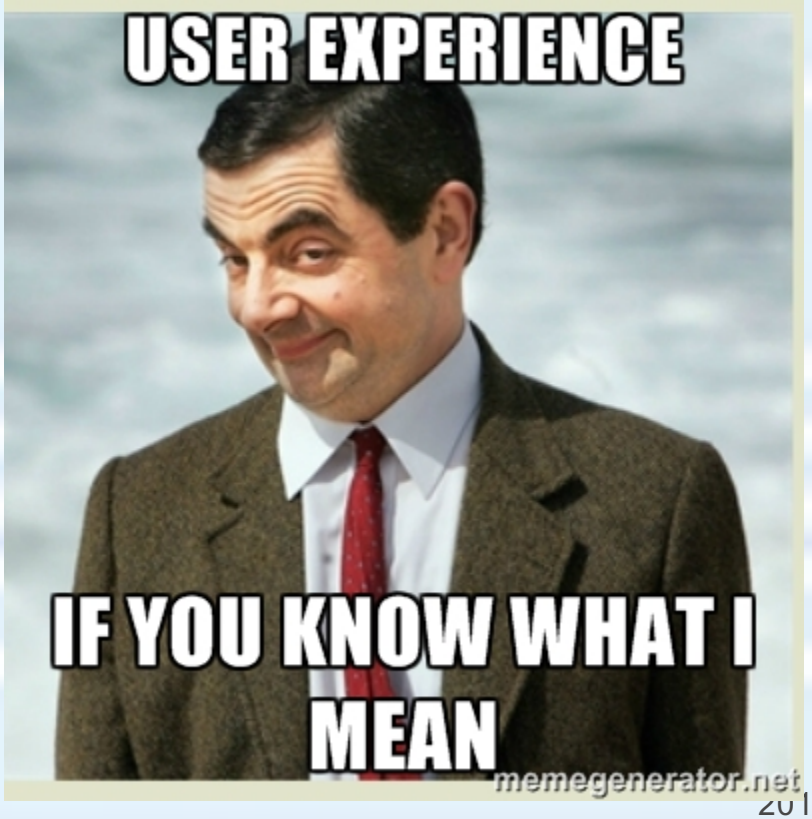
Some important flavours with a few key features:

- Agile software development ("Agile Manifesto")
  - Mini software projects; face-to-face communication
- Rapid Application Development (RAD — James Martin)
  - Voice of customer; Collaborative; rigid schedule
- Extreme Programming (XP — Kent Beck)
  - Design on the fly; unit testing of all code, pair programming; refactoring
- Scrum (Takeuchi, Nonaka and, later, Schwaber)
  - Facilitated teams scrum down in short iterations (sprints); empirical process


MAKE IT AGILE
made on imgur

UNIFIED PROCESS

The Problem
Traditional SE Methods
Modern Alternatives
Iterative SE Methods
Unified Process
Comparison & Conclusion



Iterative Development and the Unified Process

- *(Rational) Unified Process* (RUP or UP) is a process for building high quality object-oriented systems.
- Central Idea: Iterative Development
  - The life of a system stretches over a series of cycles, each resulting in a product release.

Iterative Development I

- Development as a series of short mini-projects: iterations.
- Each iteration gives a tested, integrated & executable system.
- An iteration forms a short (2-6 weeks) complete development cycle:
  - Requirements
  - Analysis
  - Design
  - Implementation
  - Integration and System Test

Note to jonjon: tell them about your acronym technique

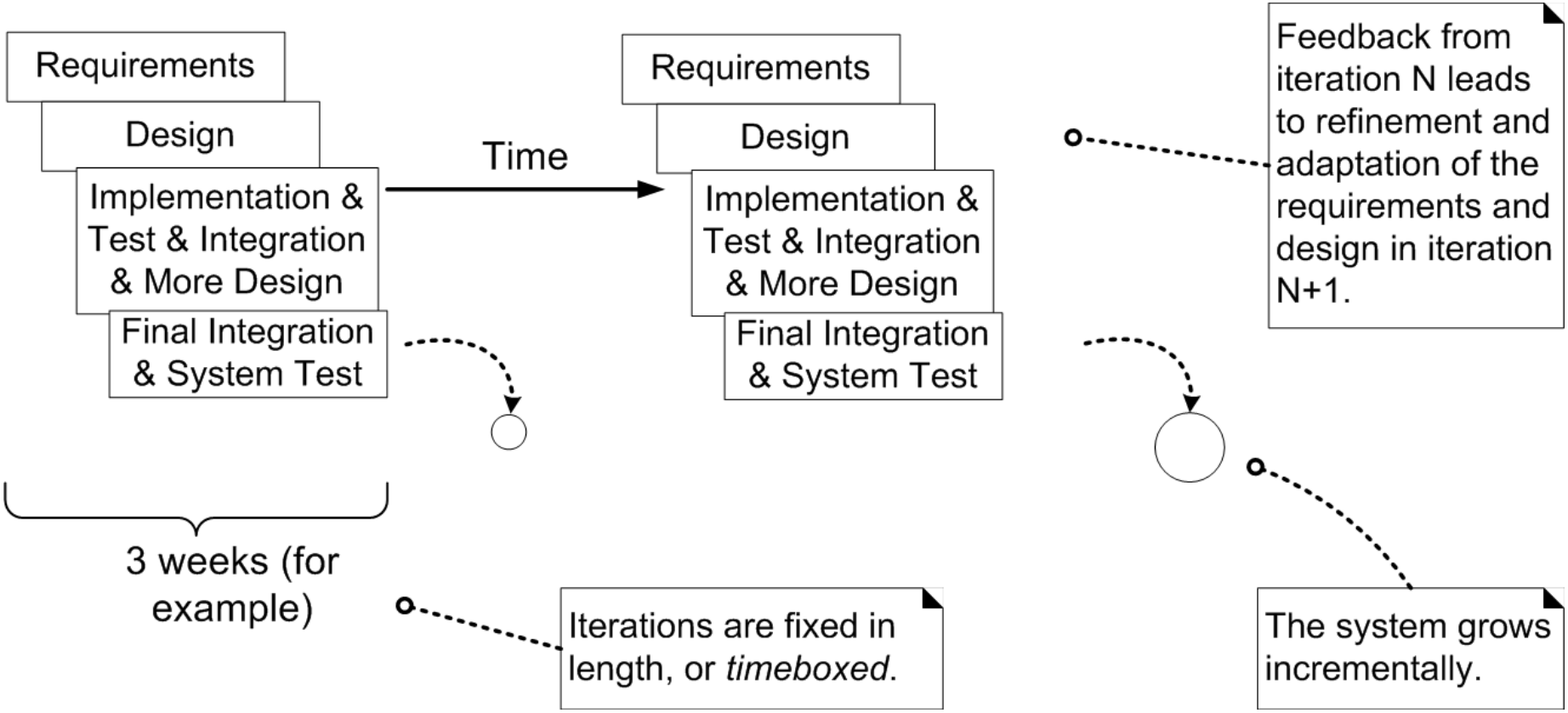
Iterative Development II

- Iterative lifecycle is based on the successive enlargement and refinement of a system
  - multiple iterations with feedback and adaptation.
- System grows incrementally over time, iteration by iteration.
  - may not be eligible for production deployment until after many iterations.

Iterative Development III

- Output of an iteration is not an experimental prototype but a **production subset of the final system.**
- Each iteration tackles new requirements and incrementally extends the system.
- An iteration may occasionally revisit existing software and improve it.

Iterative Development IV



Central Unified Process Ideas

- **Iterative Development** is number one!
- Others:
  - **Tackle high risk items early**
  - Continuous engagement of users
  - Core architecture built in early iterations
  - Continuous verification of quality: test
  - Apply use cases continuously
  - Model software with UML
  - Carefully manage requirements
  - Control changes

Unified Process phases

Inception — Define the scope of project.

- feasibility

Elaboration — Plan project, specify features, baseline architecture.

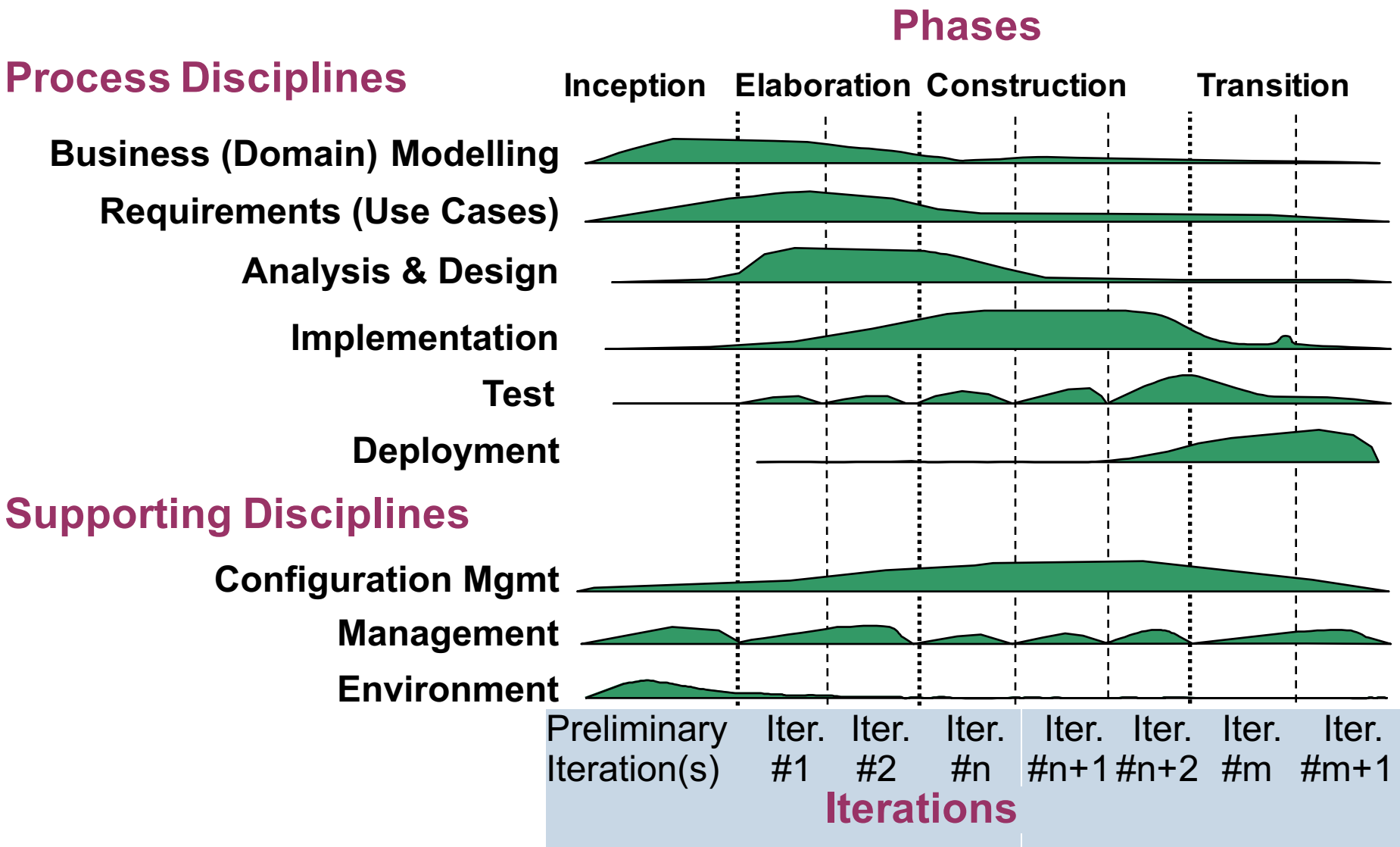
Construction — Build the product

- Refine vision, implement core, resolution of high risks, identify major requirements
- Several iterations (3 in book)

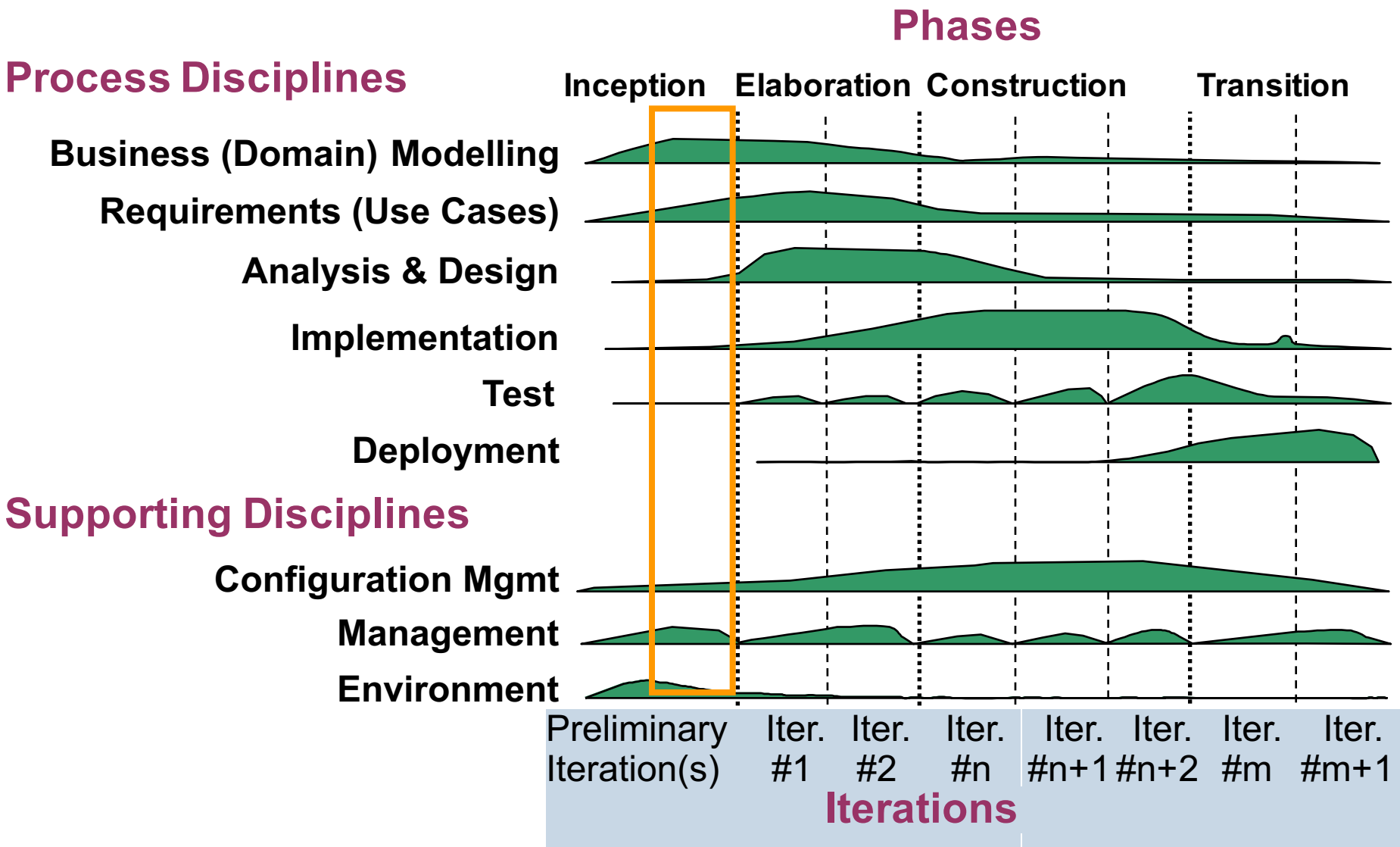
Transition — Transfer the product into end user community

- (deployment, release)

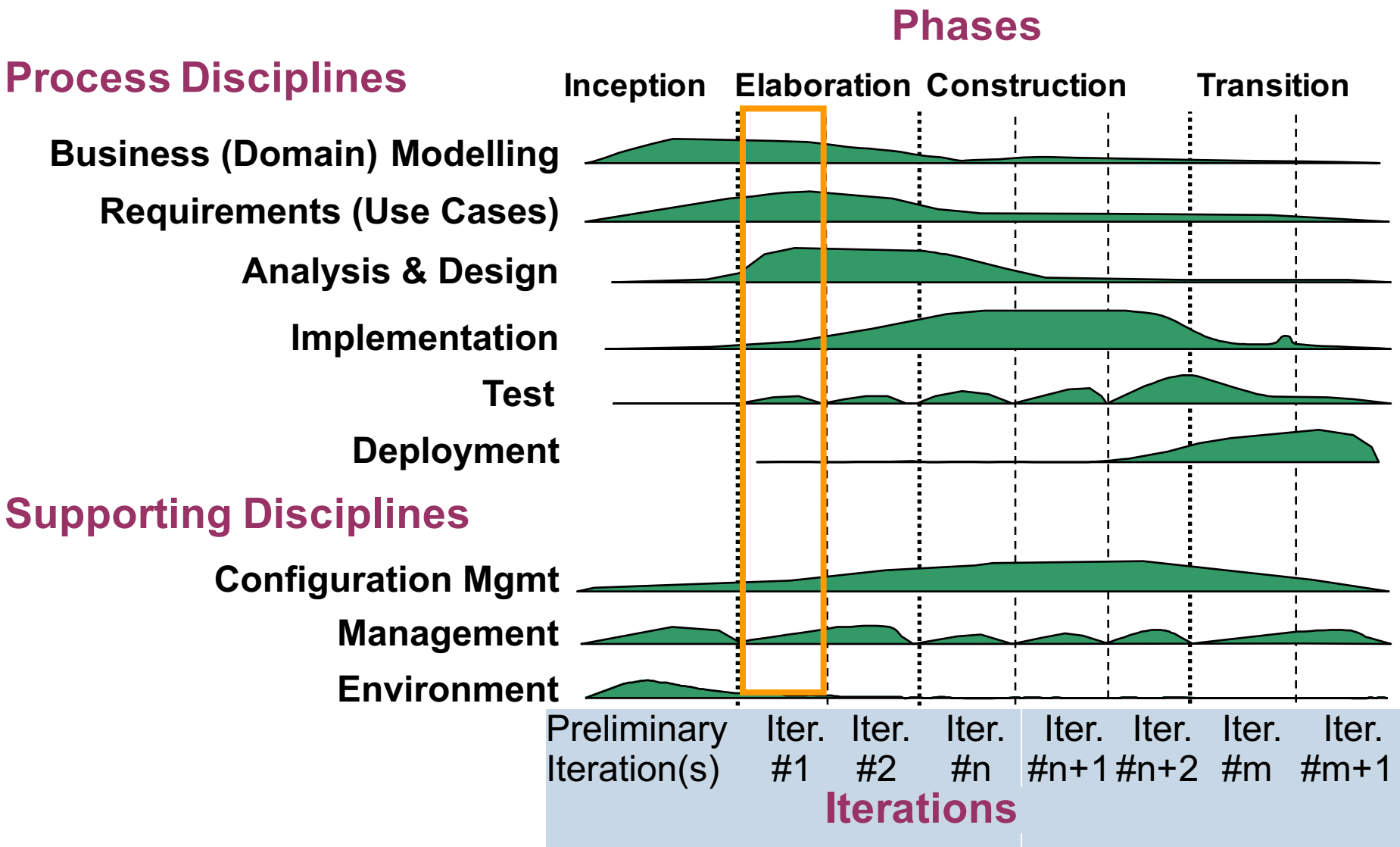
Iterative Development and the Unified Process



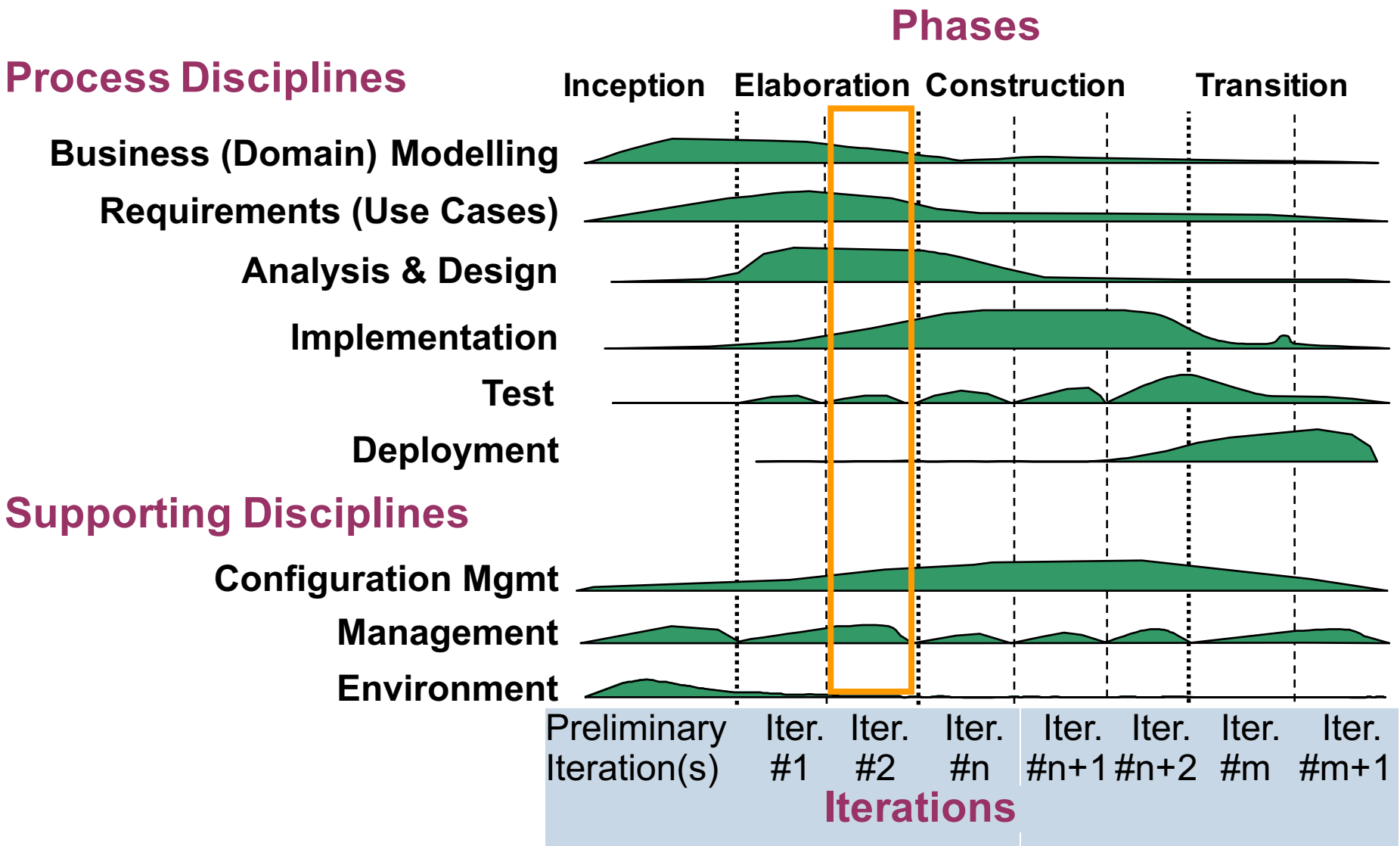
Iterations


Phases
Process Disciplines
Inception
Elaboration
Construction
Transition
Business (Domain) Modelling
Requirements (Use Cases)
Analysis & Design
Implementation
Test
Deployment
Supporting Disciplines
Configuration Mgmt
Management
Environment
Preliminary Iteration(s)
Iter. #1
Iter. #2
Iter. #n
Iter. #n+1
Iter. #n+2
Iter. #m
Iter. #m+1
Iterations

Iterations

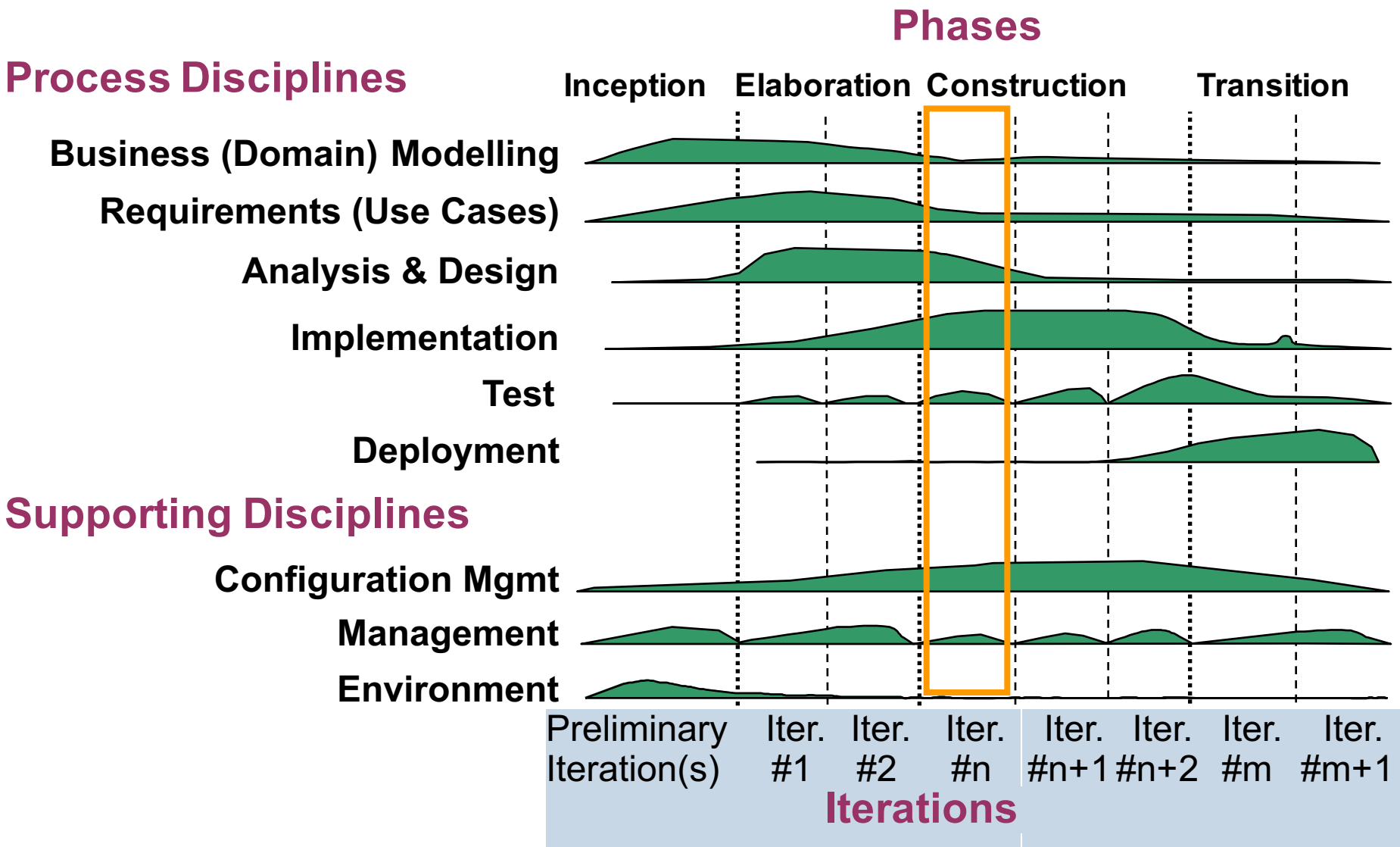

Phases
Process Disciplines
Inception
Elaboration
Construction
Transition
Business (Domain) Modelling
Requirements (Use Cases)
Analysis & Design
Implementation
Test
Deployment
Supporting Disciplines
Configuration Mgmt
Management
Environment
Preliminary Iteration(s)
Iter. #1
Iter. #2
Iter. #n
Iter. #n+1
Iter. #n+2
Iter. #m
Iter. #m+1
Iterations

Iterations


Phases
Process Disciplines
Inception
Elaboration
Construction
Transition
Business (Domain) Modelling
Requirements (Use Cases)
Analysis & Design
Implementation
Test
Deployment
Supporting Disciplines
Configuration Mgmt
Management
Environment
Preliminary Iteration(s)
Iter. #1
Iter. #2
Iter. #n
Iter. #n+1
Iter. #n+2
Iter. #m
Iter. #m+1
Iterations

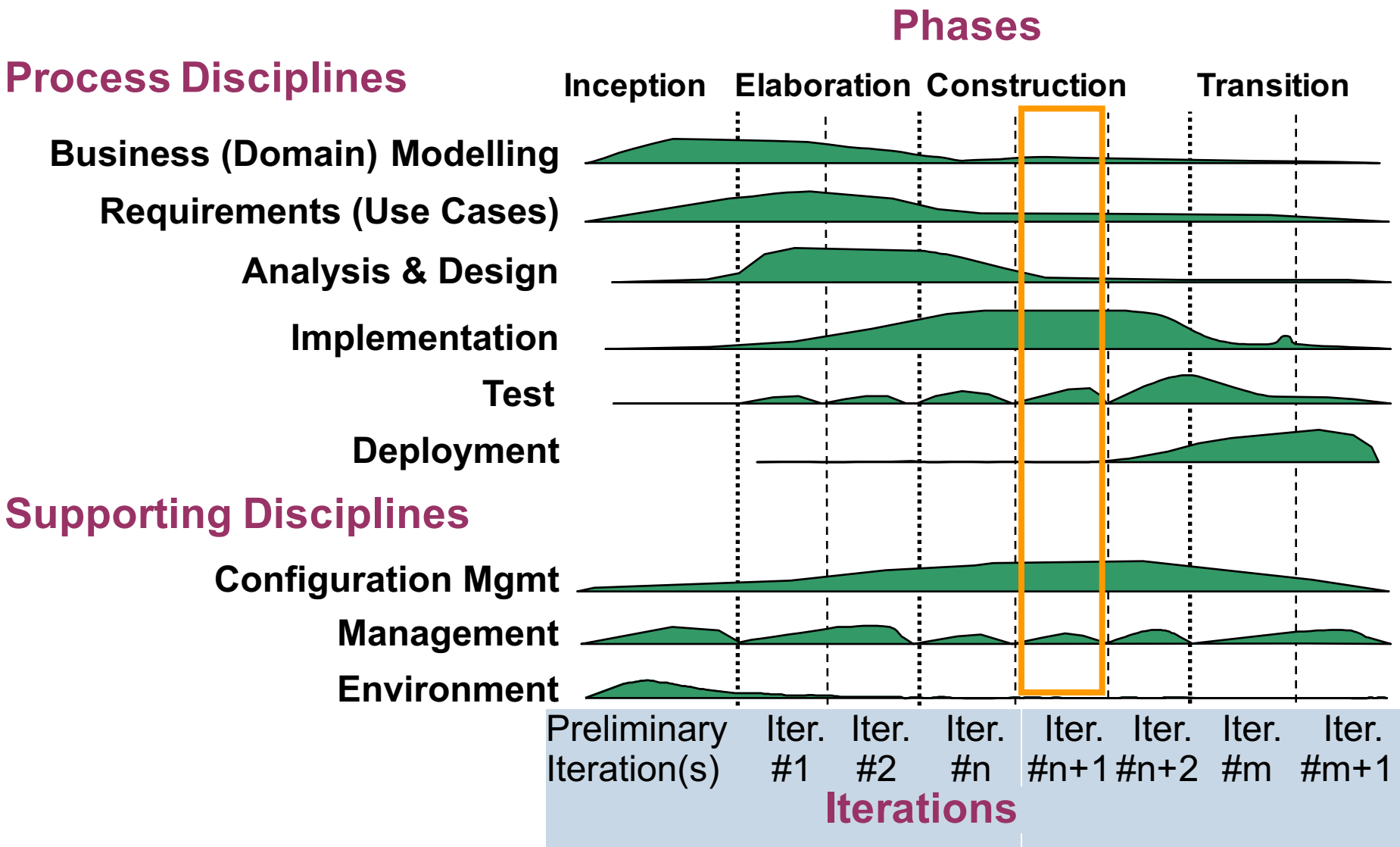
Iterations

n



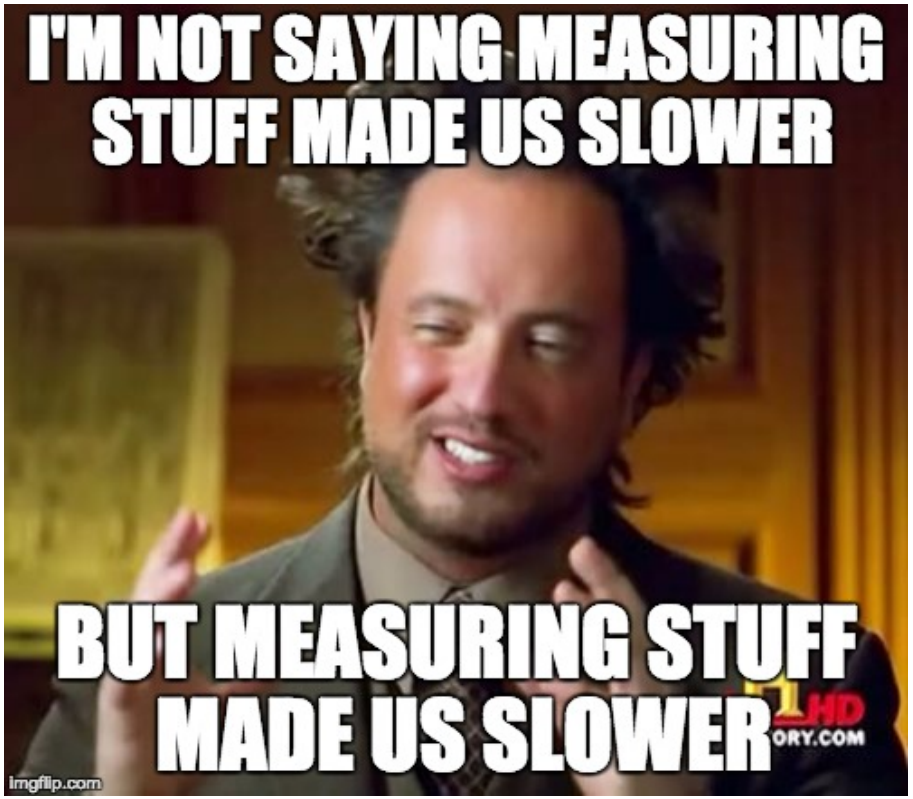
Iterations

n+1



Artefacts

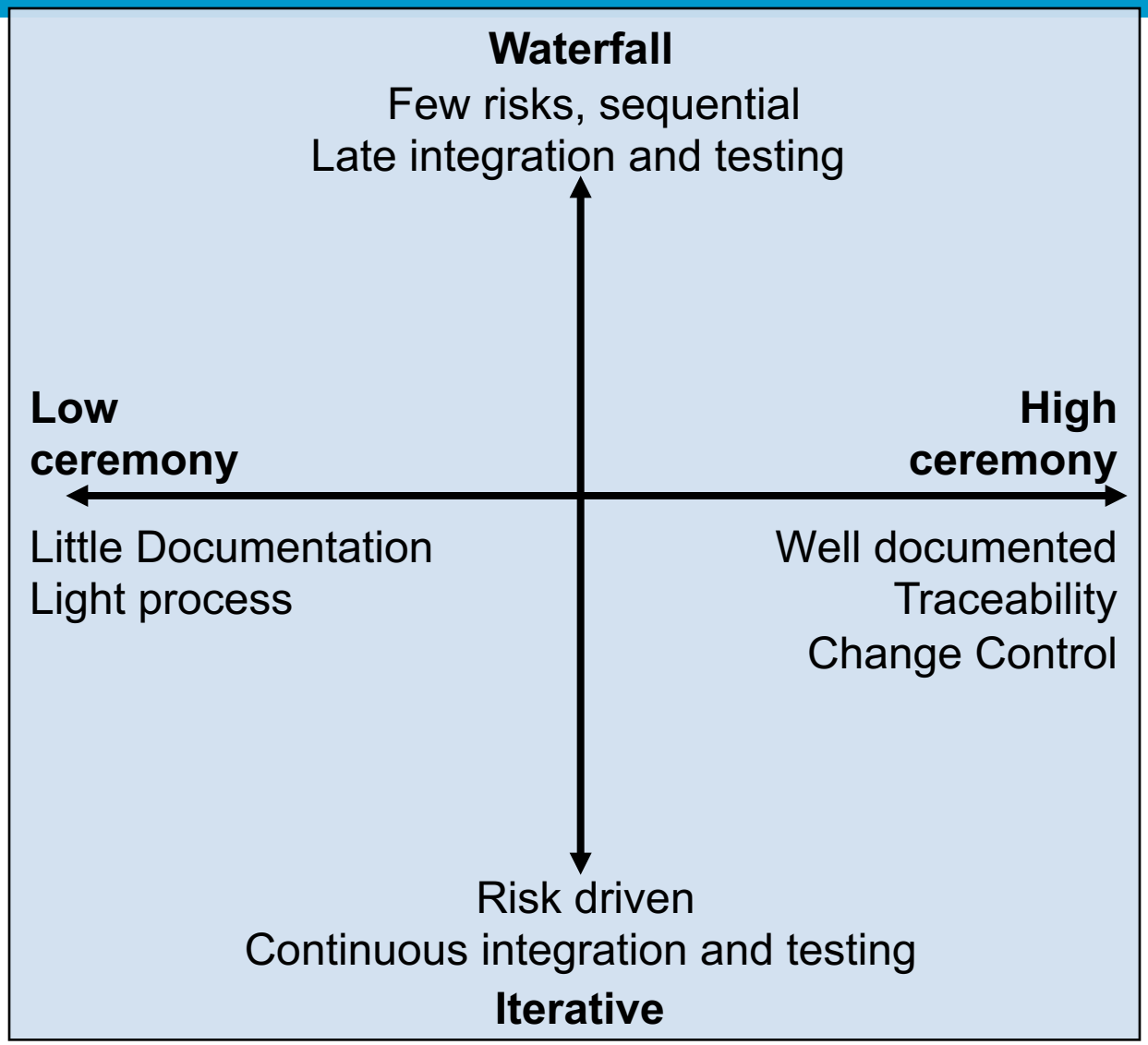
- Docs, diagrams, code, etc. that track our progress
- Everything is optional!
- Best kept electronically on website
- Following can start in inception:
  - Use-case model, vision, supplementary specification, glossary, s/w development plan, development case



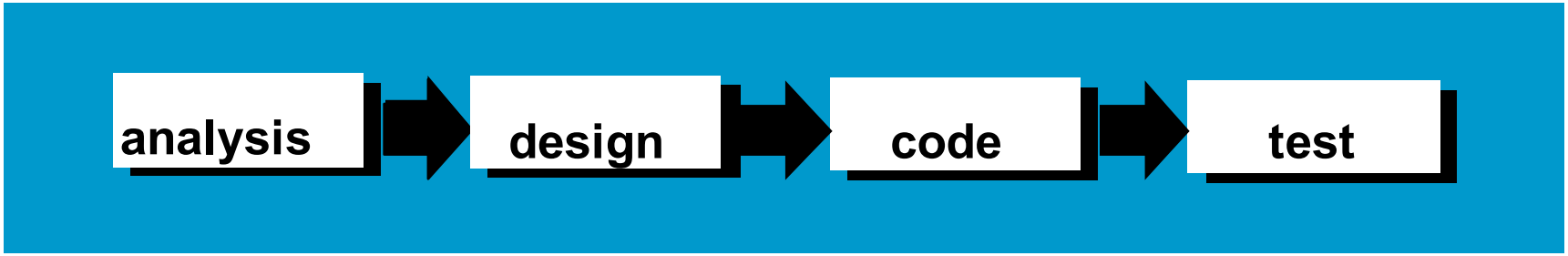
COMPARISON & CONCLUSION

The Problem
Traditional SE Methods
Modern Alternatives
Iterative SE Methods
Unified Process
Comparison & Conclusion

Process Comparison

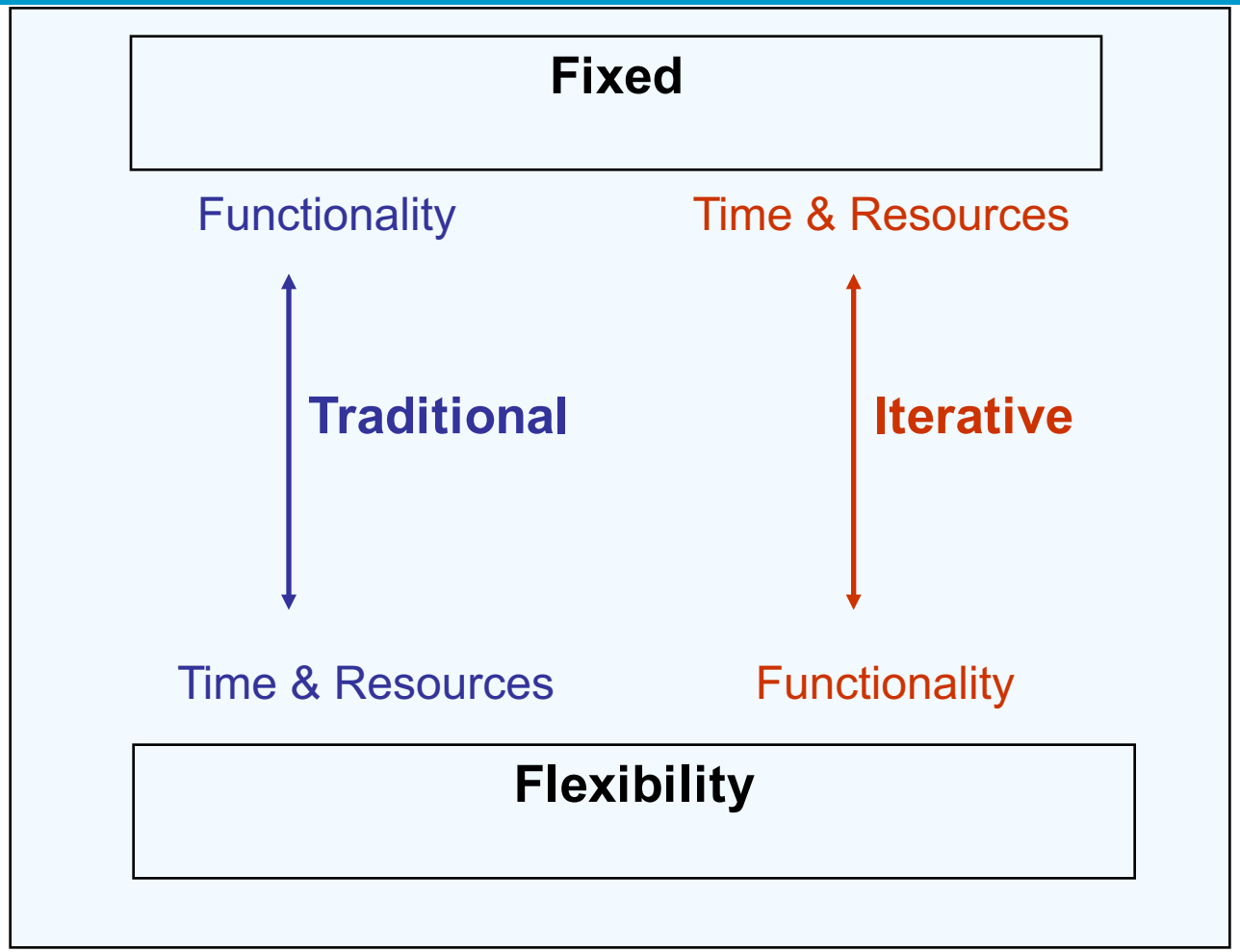


Process Models



- A framework of tasks applied during software engineering:
  - Linear (Waterfall): Based on conventional engineering
  - Prototyping: Build a system to clarify requirements
  - Rapid Application Development (RAD): Well-defined 60-90 day projects
  - Incremental: Deliver increasing functionality at each iteration
  - Spiral (Boehm): Similar set of tasks applied for each turn of the spiral.
  - Component based: Aimed at producing and reusing O-O components
  - Agile: Embrace change and adapt to it and keep things simple.

Process Comparison

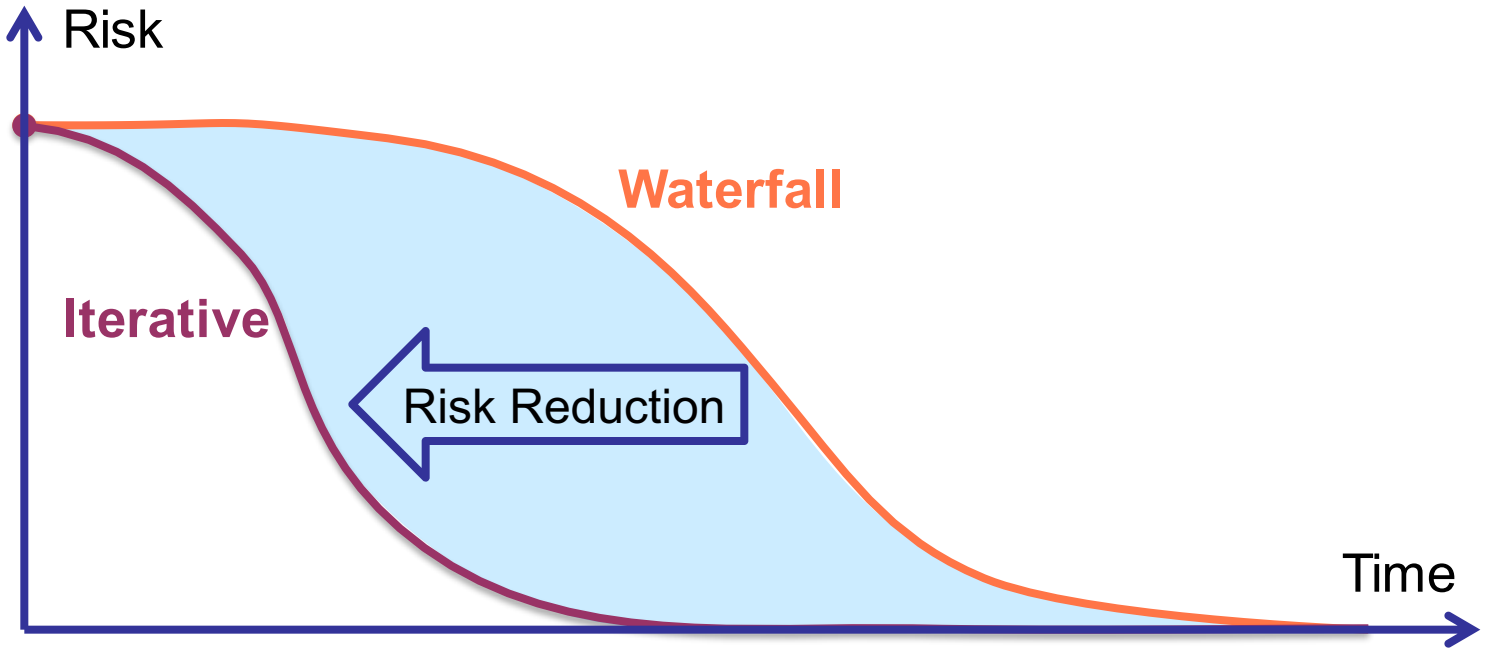



WHAT IF I TOLD YOU
YOU COULD GET CLEARER PRODUCT DIRECTION BY TALKING TO YOUR USERS
ip.com

Reduce Risk

- Iterative methods attempt to reduce risk by bringing versions out early
- For further discussion see

  www.projectsmart.co.uk/reducing-risk-increasing-probability-of-project-success.php

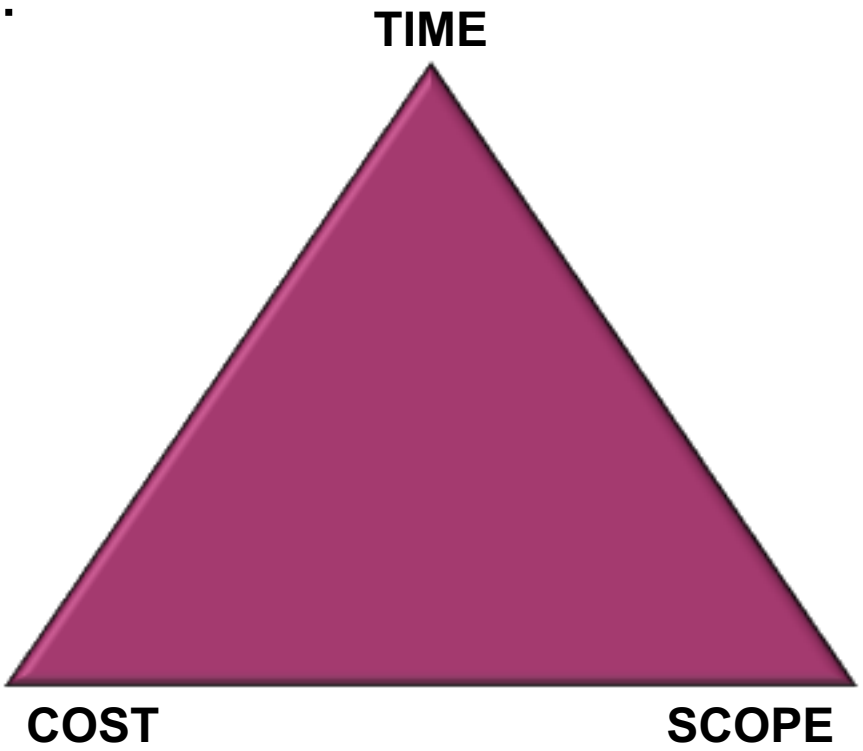


Conclusion

- Consider only iterative technologies
- Agile techniques
  - Small time cycles
  - Many prototypes
  - Meet user requirements
  - Timescale is adopted by the development team

Alternative to the SE Constraint Triangle?

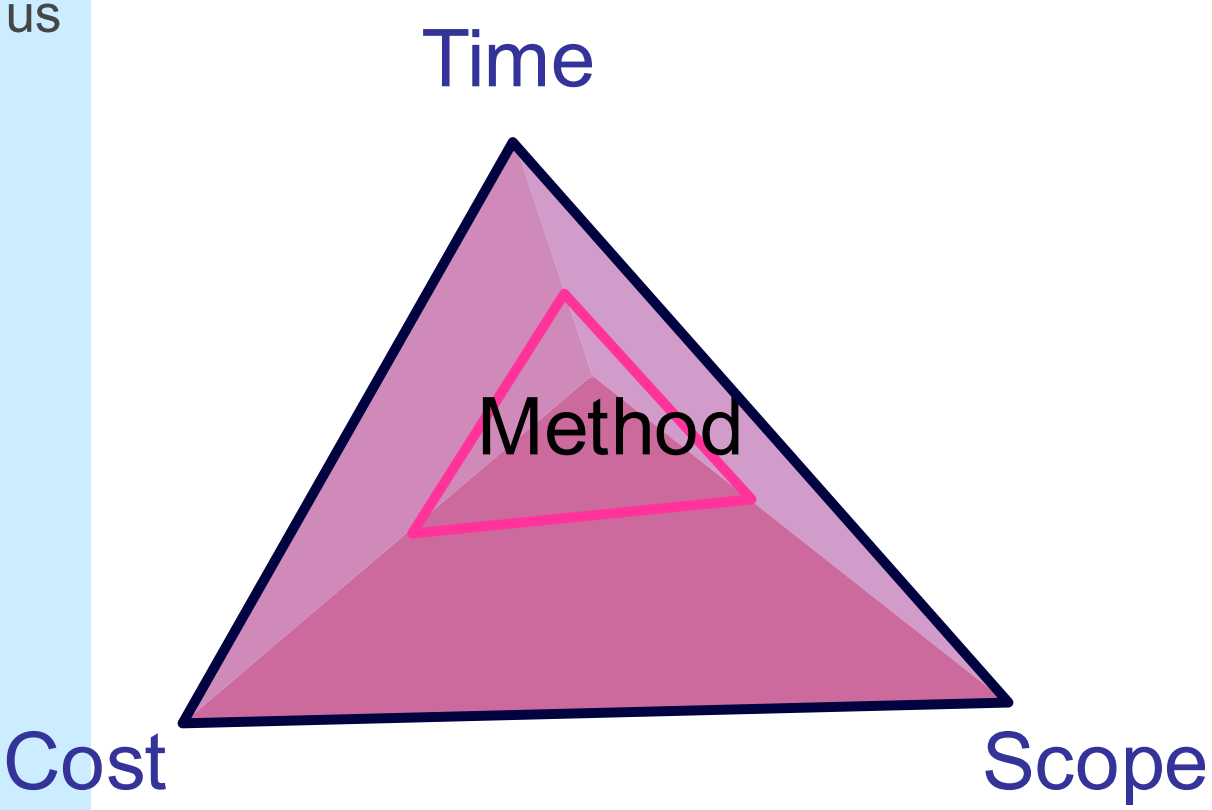
- Remember:



- What other way can we look at this?

Method is a third dimension!

Can use method to move us into another Cost/ Scope/ Time regime.



Benefits of iterative development

- Early reduction of risk: technical, requirements, objectives, usability, etc.
- Early visible progress
- Early feedback
  - user engagement, and adaptation
  - better meets the real needs
- Managed complexity: No very long and complex steps
- Get a robust architecture
  - Architecture can be assessed and improved early.
- Handle evolving requirements
  - Users provide feedback to operational systems.
  - Responding to feedback is an incremental change.
- Allow for changes: System can adapt to problems
- Learn and apply lessons within the development process