Software processes:
Putting DevOps in context

Agenda

- Best practices in software development
- Traditional methods
- Transition to “agile”
- The case for DevOps

Making software

- Correct?
- Useful?
- Developed fast?
- Cheap?
- Usable?
- Reusable?

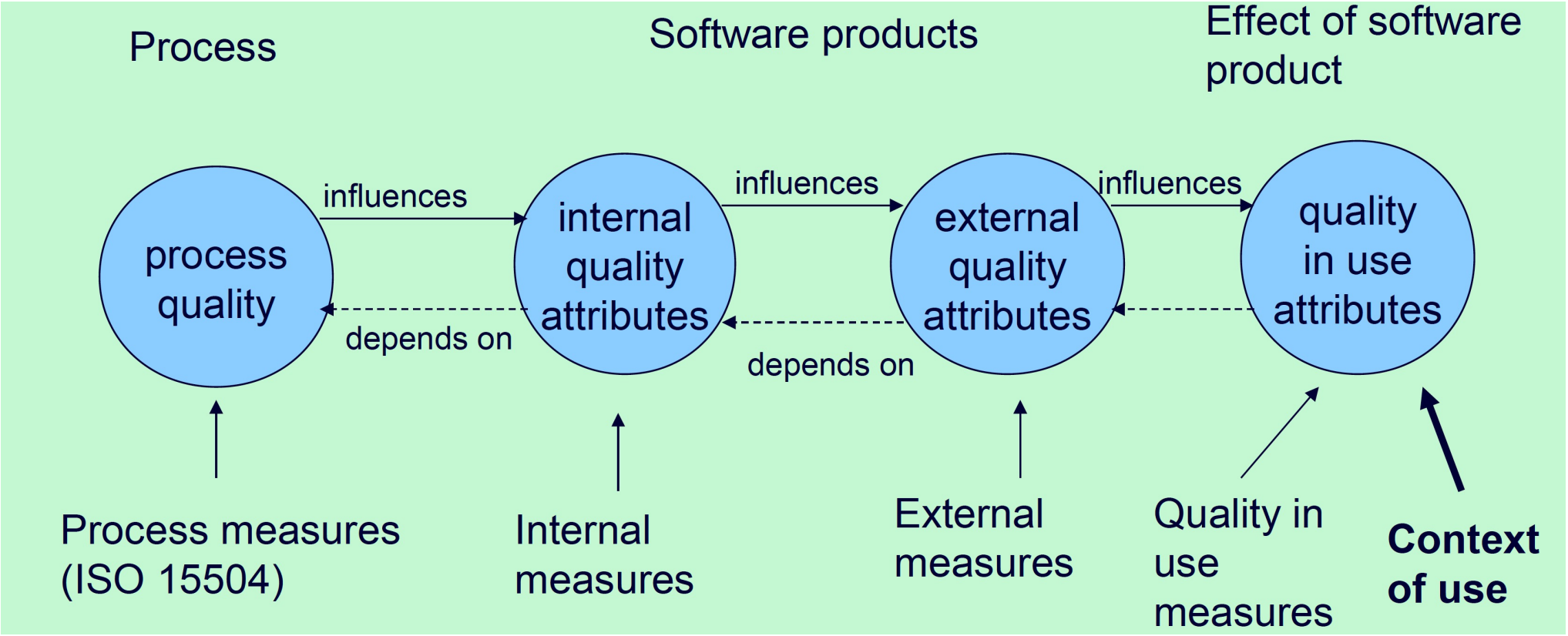
Software development: typical problems

- Inaccurate understanding of user needs
- Inability to deal with changing requirements
- Late discovery of serious flaws
- Unacceptable software performance
- Software hard to maintain or extend

→ **poor software quality**

Software quality

There are several types of software quality



Caper Jones on sw project failures

- “As to project cancellations, we cover a wider range than Standish Group because they show only IT projects. We include embedded, systems software, web applications, IT, etc.
- 10 function points = 1.86% cancels 100 function points = 3.21%
  1000 function points = 10.14% 10000 function points = 31.27%
  100000 function points = 47.57%
- The canceled projects are usually late and over budget when the plug is pulled. On average a canceled project is about 10% more expensive than a successful project of the same size and type”

Standish CHAOS report 2015: agile vs waterfall

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

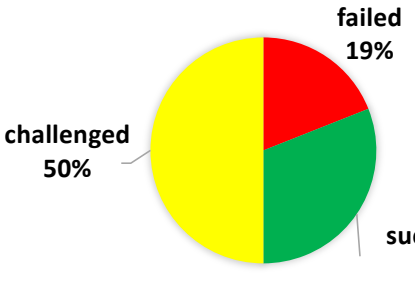
SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000

http://www.infoq.com/articles/standish-chaos-2015

Project Success
Quick Reference Card

Based on CHAOS 2020: Beyond Infinity Overview. January'2021, QRC by Henny Portman



Modern measurement
(software projects)

Good Sponsor, Good Team, and Good Place are the only things we need to improve and build on to improve project performance.

The Good Place is where the sponsor and team work to create the product. It's made up of the people who support both sponsor and team. These people can be helpful or destructive. It's imperative that the organization work to improve their skills if a project is to succeed. This area is the hardest to mitigate, since each project is touched by so many people. Principles for a Good Place are:

- The Decision Latency Principle
- The Emotional Maturity Principle
- The Communication Principle
- The User Involvement Principle
- The Five Deadly Sins Principle
- The Negotiation Principle
- The Competency Principle
- The Optimization Principle
- The Rapid Execution Principle
- The Enterprise Architecture Principle

Successful project Resolution by Good Place Maturity Level:

highly mature	50%
mature	34%
moderately mature	23%
not mature	23%

The Good Team is the project's workhorse. They do the heavy lifting. The sponsor breathes life into the project, but the team takes that breath and uses it to create a viable product that the organization can use and from which it derives value. Since we recommend small teams, this is the second easiest area to improve. Principles for a Good Team are:

- The Influential Principle
- The Mindfulness Principle
- The Five Deadly Sins Principle
- The Problem-Solver Principle
- The Communication Principle
- The Acceptance Principle
- The Respectfulness Principle
- The Confrontationist Principle
- The Civility Principle
- The Driven Principle

Successful project Resolution by Good Team Maturity Level:

highly mature	66%
mature	46%
moderately mature	21%
not mature	1%

The Good Sponsor is the soul of the project. The sponsor breathes life into a project, and without the sponsor there is no project. Improving the skills of the project sponsor is the number-one factor of success – and also the easiest to improve upon, since each project has only one.
Principles for a Good Sponsor are:

- The Decision Latency principle
- The Vision Principle
- The Work Smart Principle
- The Daydream Principle
- The Influence Principle
- The Passionate Principle
- The People Principle
- The Tension Principle
- The Torque Principle
- The Progress Principle

Successful project Resolution by Good Sponsor Maturity Level:

highly mature	67%
mature	33%
moderately mature	21%
not mature	18%

What is software quality?

- Software *functional quality* reflects how well it complies with or conforms to a given design, based on some functional requirements
- Software *structural quality* refers to how it meets non-functional requirements that support the delivery of the functional requirements

Important: Software quality can be measured!

Some software qualities

- **Robust** software: able to cope with errors during execution
- **Sustainable** software: long lasting software able to cope with changes
- **Reproducible** software: systematic use of version control during development

Good practices for quality software

- Write programs for people, not computers
- Improve software quality only after it works correctly
- Plan for finding mistakes
- Document design and purpose (ie. architecture)
- Use a tool to automate tests and check quality
- Make incremental changes, use a version control system
- Reuse code instead of rewriting it
- Collaborate (eg. by pair programming, by using an issue tracking tool, by sharing a kanban)

Poor practices for sw development

- Under-evaluation of development risks
- Insufficient (automatic) testing
- Insufficient requirements management
- Ambiguous communication among stakeholders
- Inconsistencies among requirements, designs, implementations, and tests
- Fragile software architecture
- Complexity deriving from dependencies

Examples

- Typical risk in the development cycle: misunderstanding requirements
- Typical risk in the development environment: too many manual activities
- Typical risk in the operating environment: underestimating dependencies
- Typical risk in teamwork: the team is not working well together

Traditional approach to sw developmnt

Sw development is a sequence including the following phases:

I. Requirements Analysis
II. Design
III. Coding
IV. Testing: first check the units, then the system

A development process goes through these phases **linearly**:
first all the requirements are defined, then the design is completed, and finally the code is written and tested.

The key assumptions are that when design begins, requirements no longer change. When coding starts, the design is frozen, etc.

NB: This “traditional” approach is sometimes called "waterfall development”

Personal and team process

- Personal processes are defined sets of activities that guide an individual in doing personal work. A personal process can be a wholly new creation, or a modification of an established process. Its esecution is usually based on personal experience.
- Team processes are sets of defined activities that each team member follows in the same way when performing the teamwork.
- Team processes are usually based on the collective experiences of the team members, and typically are redefined for or tailored to address the challenges faced by the team when beginning a new project

A plethora of methods

- Every software development process refers to a method (called *software process model*)
- There are more than 50 distinct sw development methods in use, and an even larger number of hybrids.
- Some of the most famous development methods include the traditional *waterfall* approach, various flavors of *agile (eg. Scrum)*, the Rational Unified Process (*RUP*), the Team Software Process (*TSP*), etc.

Making and operating software is a social activity

"multi-person construction of multi-version software"

— Parnas

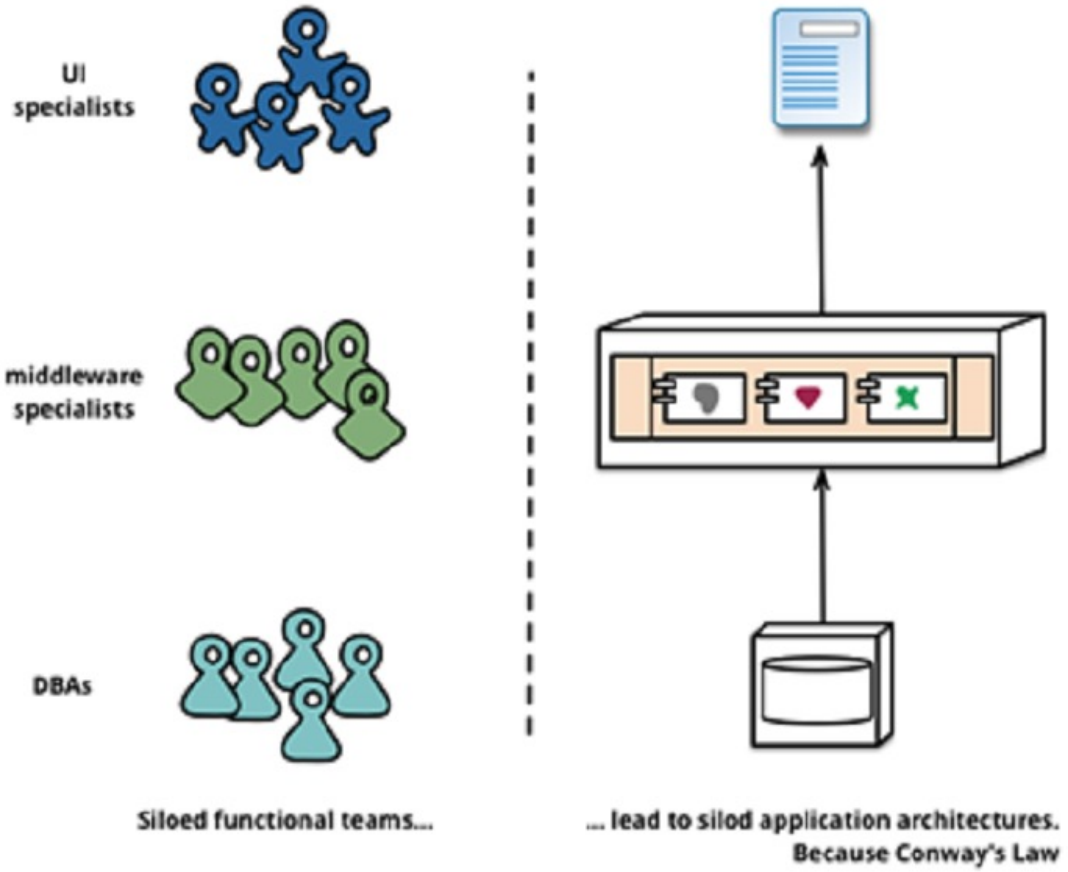
- Software is complex and difficult to build
- Successful software systems must evolve or perish
  - Changes to the software is the norm, not the exception
  - Support changes both during the development and after its deployment
- Teamwork (developers and operators)
  - Scale issue ("program well" is not enough) + communication issues: Conway's law

Conway's Law

- The law: *Organizations that design software systems are constrained to produce designs that are copies of the communication structures of these organizations*
- Examples:
  - If you have four groups working on a compiler, you'll get a 4-pass compiler
  - If a development team is set up with a SQL database specialist, a JavaScript/CSS developer and a C# developer they will produce a system with three tiers: a database with stored procedures, a business middle tier and a UI tier

Conway's law

- The organizational structure drives a particular software architecture. The software architecture drives a particular organizational structure
- People who work closely together and communicate frequently will create software that reflects this and vice versa



Conway's law

- Communication between human beings is complex, and each person has a limited amount of time and energy for communication. Therefore, when an issue is complex and requires cooperation, we need to divide our organization to improve communication efficiency.
- The system design in which the members of an organization work depends on the communication between the members. Managers can adjust the division mode to implement different ways of communication between teams, which will influence the system's design.
- If a subsystem has clear external communication boundaries, then we can effectively reduce the communication costs, and the corresponding design will be more appropriate and efficient.
- There is need to continuously optimize a complex system with the help of error tolerance and resilience. Do not expect up front big and all-embracing designs or architectures, as their development occurs in an iterative manner.

Conway's law

- Leverage all possible tools to improve the communication efficiency, such as Slack, Github, and Wiki. Communicate with only the people involved. Each person and each system must have clear duties. You must know whom to turn to in case of an issue, to ensure accountability.
- Design a system in the MVP mode, verify and optimize the system in an iterative manner, and ensure that the system is elastic.
- Adopt a team that aligns with your system design and streamline the team if possible. A plausible recommendation is that whenever possible, set up teams by departments so that each team is autonomous and communicational. Clarify the departmental boundaries to reduce external communication costs. Each small team must be responsible for its module throughout the entire module life cycle. Prevent vague boundaries and shifting the responsibility. Set up the "inter-operate, not integrate" relationship between the teams.
- Develop small and efficient teams, as the costs increase and the efficiency decrease when the number of team members goes up. Jeff Bezos, CEO of Amazon, had a funny rule of thumb: if two pizzas are not enough for a team, the team is oversized. Typically, a small product team of an Internet company consists of 7 to 8 people. (These include people in charge of front-end and back-end tests, interactions, and user research. Some people may have multiple task assignments.)

Process models

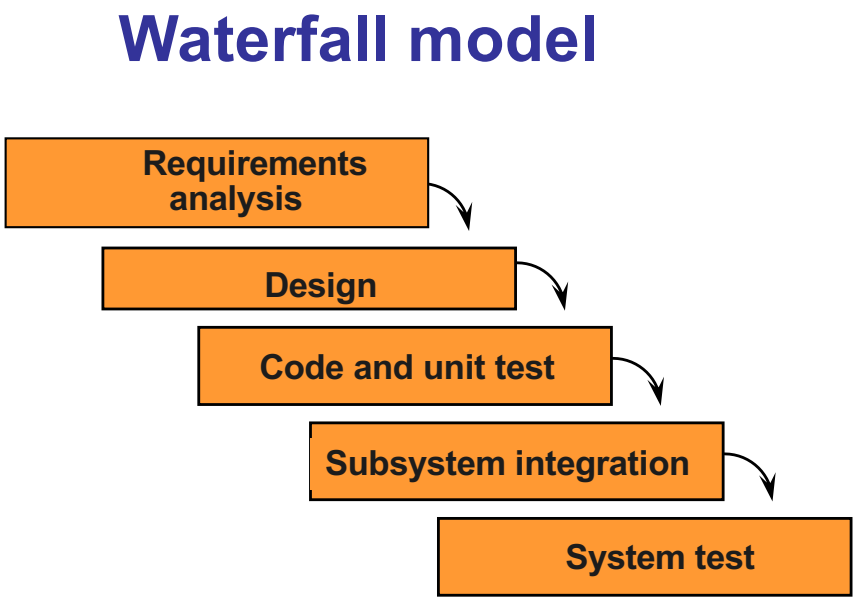
- A method, or process, is a way of working: whenever you do something, you're following a process
- Some processes are ad hoc and informal; others are written and follow some *process model*

Models for the software process

A model for a sofware development process is a method describing the roles, the tasks, and the documents (*artifacts*) to be developed

- Waterfall (planned, linear) eg. MIL498
- Spiral (planned, iterative) eg. RUP
- Agile (estimated, test driven) eg. XP, Scrum, SAFe
- DevOps (test driven, automated, measured)

Waterfall characteristics



- One way communicatons
- Delays confirmation of critical risk resolution
- Measures progress by assessing work-products that are poor predictors of time-to-completion
- Delays and aggregates integration and testing
- Precludes early deployment
- Frequently results in major unplanned iterations

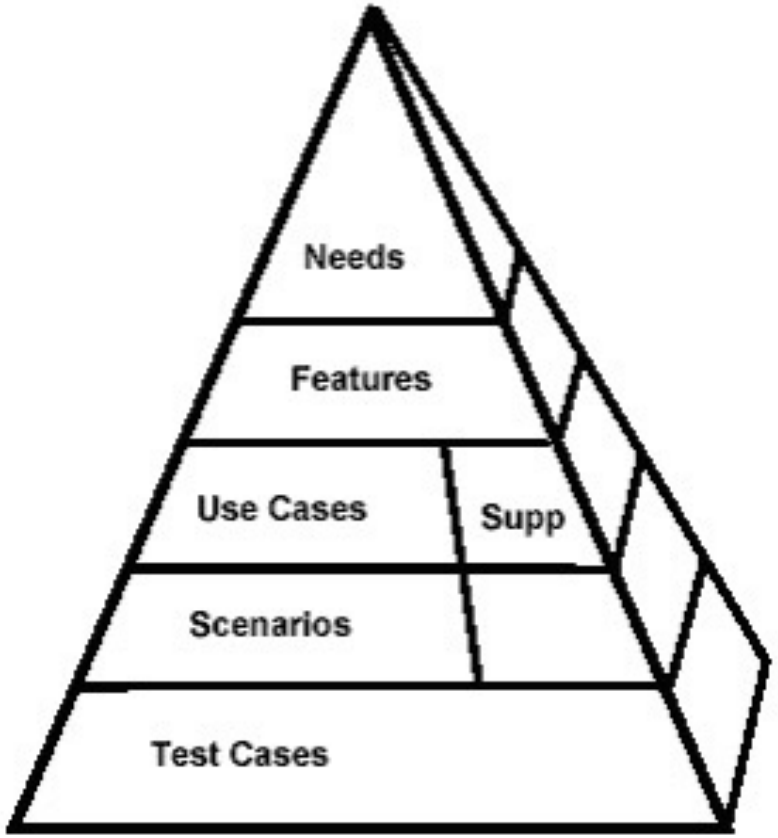
The requirements pyramid

Some user has some needs

Needs could be satisfied by “features” that some system must have

Each feature corresponds to a need and is a collection of requirements

Features and requirements can be aggregated in “scenarios”: after the code is built, testing it in the scenario will prove that its features satisfy the user’s needs



www.ibm.com/developerworks/rational/library/04/r-3217

Problems with the waterfall lifecycle

1. "Real projects rarely follow the sequential flow that the waterfall model proposes. *Iteration* always occurs and creates problems in the application of the paradigm"

2. "It is often *difficult* for the customer *to state all requirements* explicitly. The classic life cycle requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects."

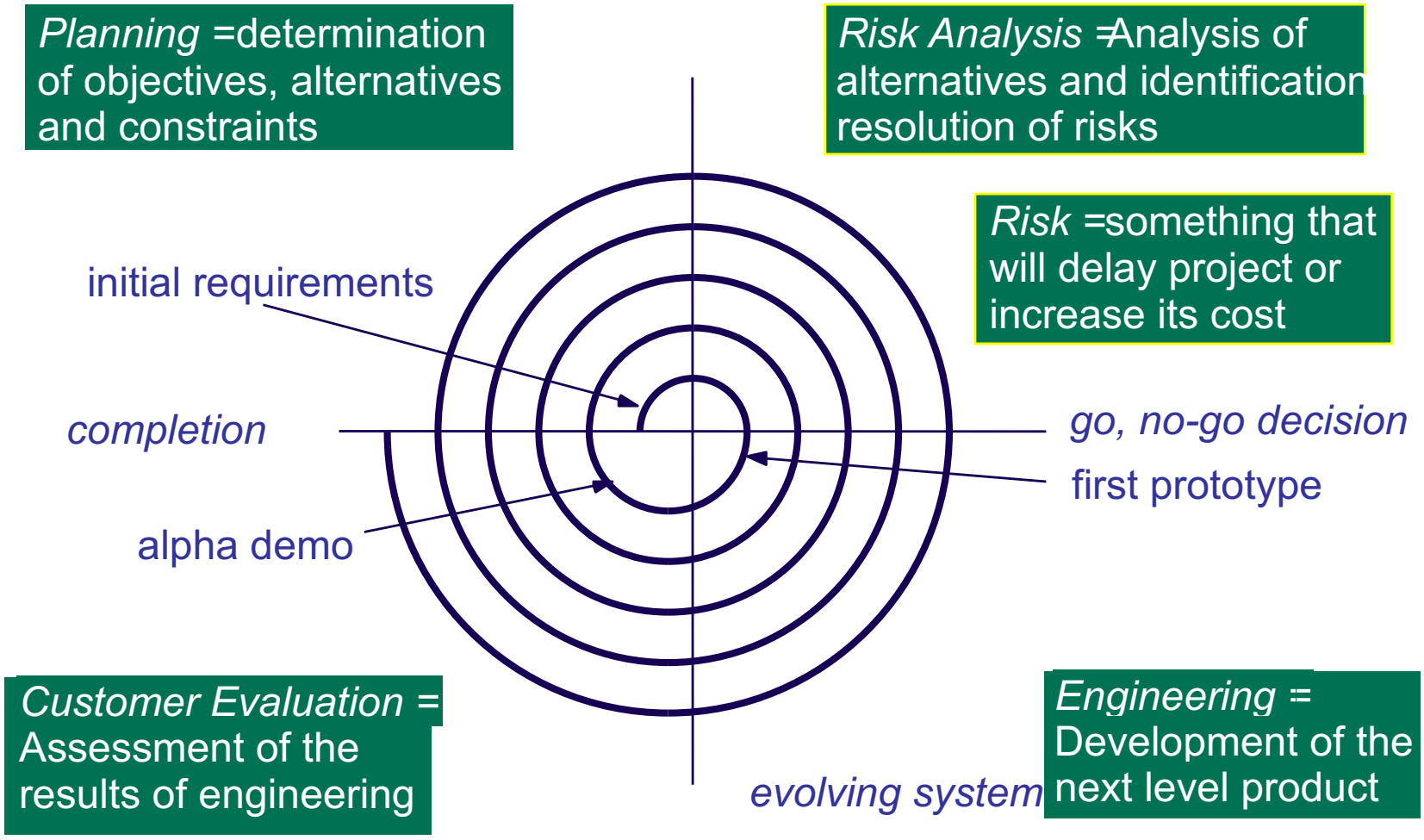
3. "The customer must have patience. A *working version* of the program(s) will not be available until *late in the project* timespan. A major blunder, if undetected until the working program is reviewed, can be disastrous."

— *Pressman, SE, p. 26*

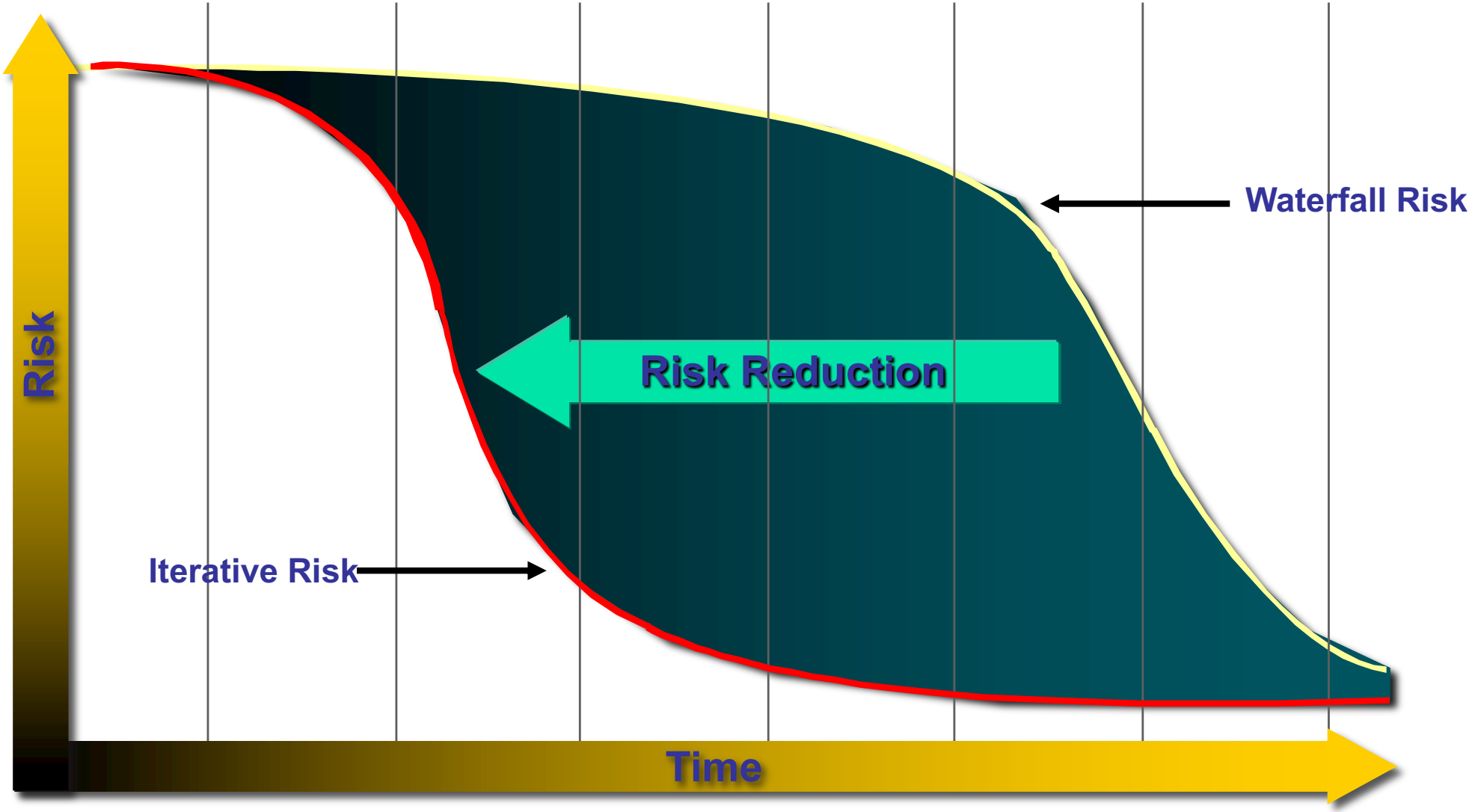
Iterative models

- All iterative models start with a plan to iterate the analysis, design and implementation
  - You will not get it right the first time, so integrate, validate, and test as frequently as possible
  - More than one iteration of the software development cycle may be in progress at the same time
  - All iterative models have the goal to mitigate development risks involving more frequently the customer

The spiral lifecycle



Risk: waterfall vs iterative

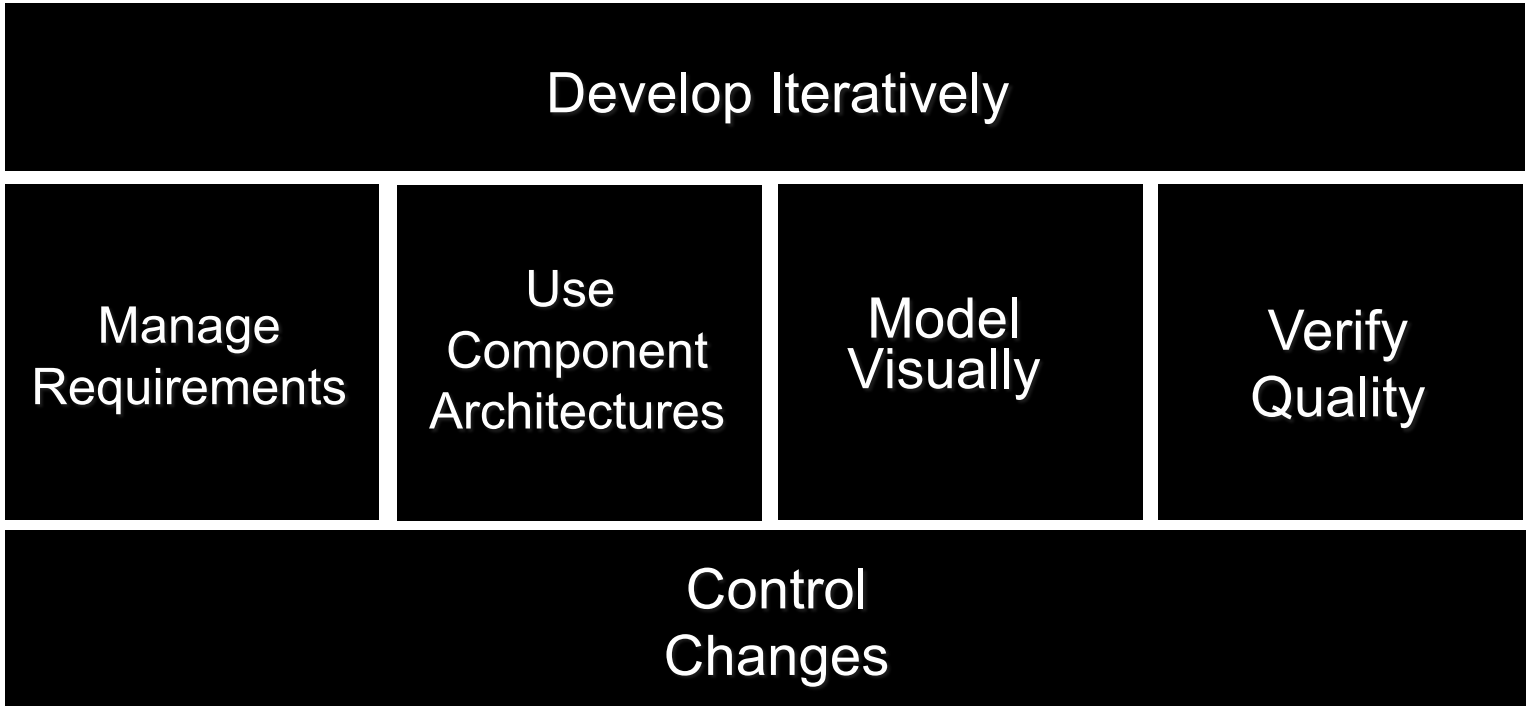


Iterative development

Plan to *incrementally* develop (i.e., prototype) the system

- If possible, *always have a running version* of the system, even if most functionality is yet to be implemented
- *Integrate* new functionality as soon as possible
- *Validate* incremental versions against user requirements.

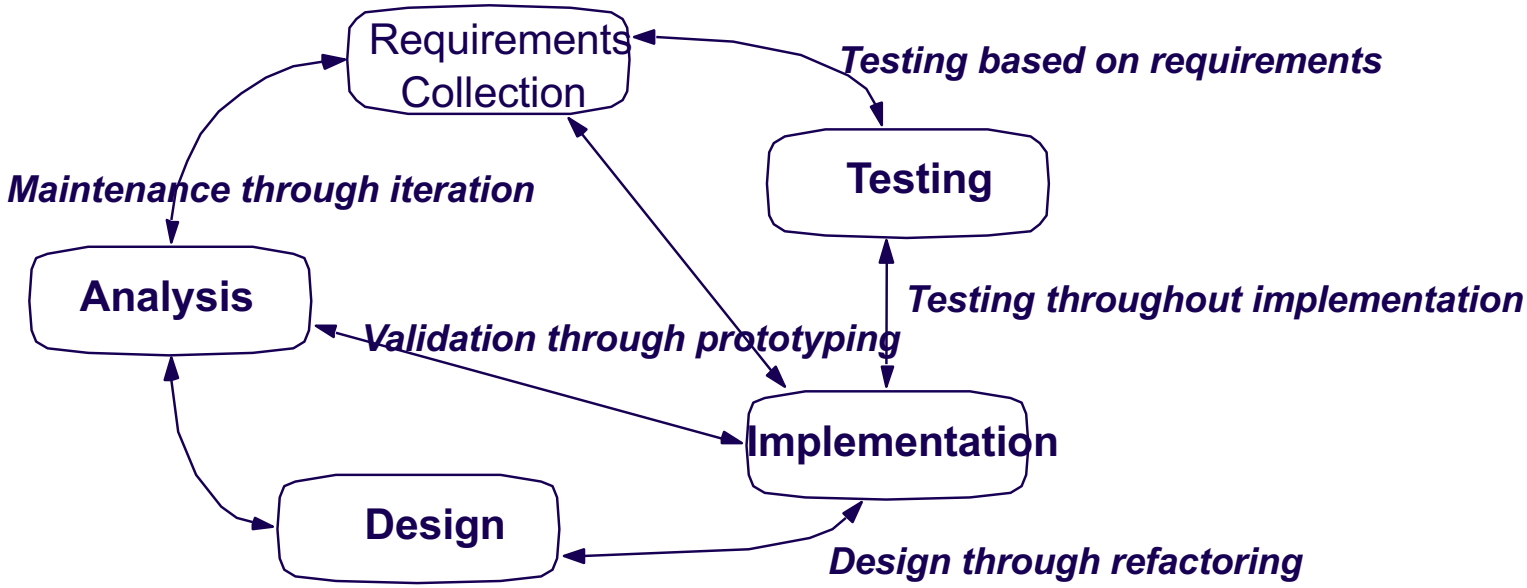
Good practices of software development in RUP



Gibbs, Project Management with the IBM Rational Unified Process: Lessons From the trenches

Iterative development

In practice, development is always iterative,
and *most* activities can progress in parallel



Incremental iterations in DevOps

- In a DevOps environment with continuous delivery and deployment capabilities, iterations can be measured in hours or minutes.
- Iterative development reduces complexity and work in progress, which are important elements in implementing lean software development concepts.
- The term *incremental* means that the product evolves incrementally – step-by-step – with each development iteration.
- In other words, the term *increment* implies the release of a new slice of functionality with each iteration

Testing: not a task for the developer

- You have built a program
- You debug the program: when you stop? Answer: when you are tired
- Conflict of interest: Programmers are not really interested in proving their programs faulty or having bad quality

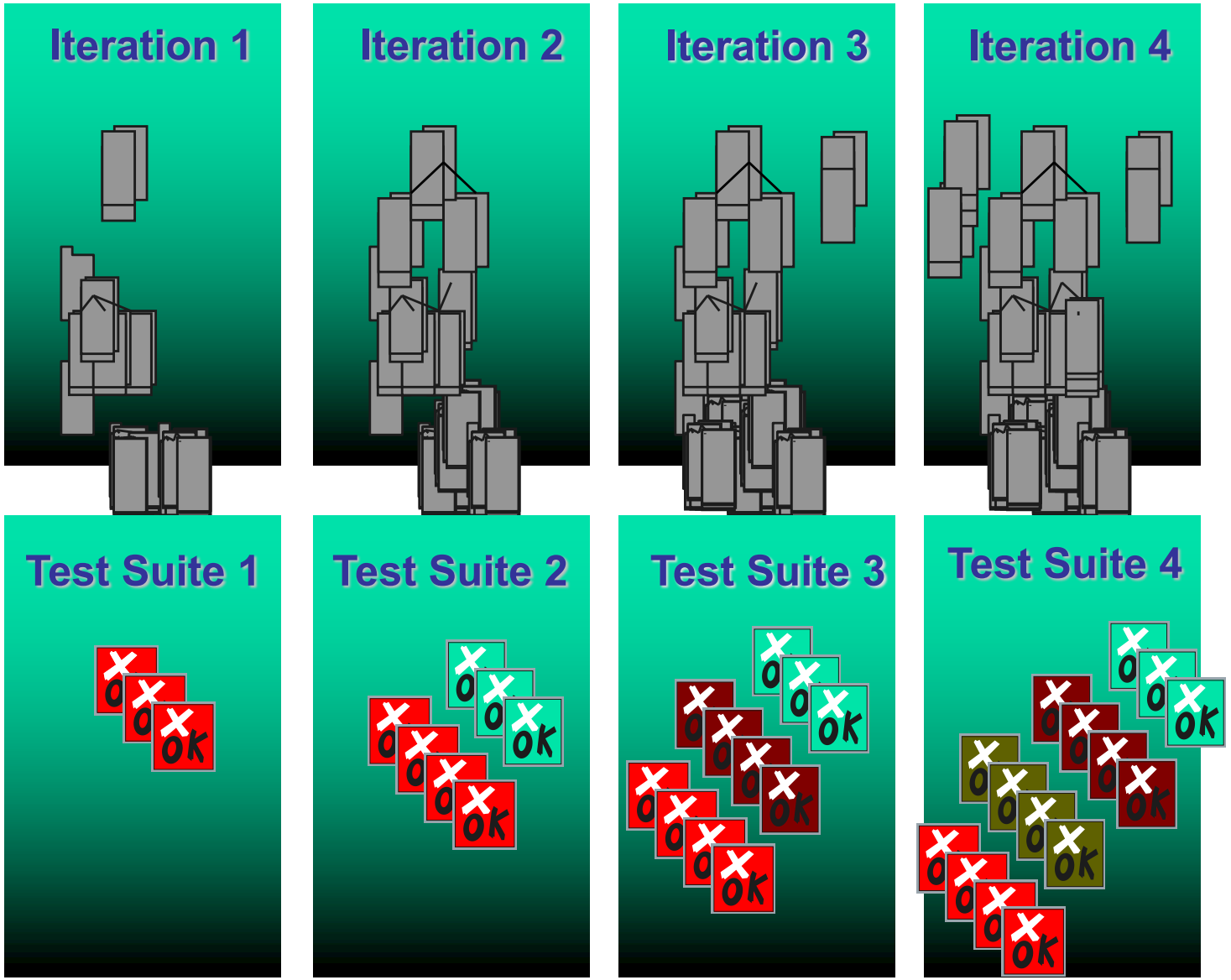
Testing before designing

- What is software testing? an investigation conducted to provide information about the quality of some software product
- In planned process models testing happens after the coding, and checks if the code satisfies the requirements
- What happens if we define the tests before the code they have to investigate?

Test each iteration

Requirements, models and code

Tests



Agile methods

Agile methods are a familly of process models which all recommend:

- **Frequent releases** of the product being developed
- **Continuous participation** of the customer to the activities of the development team
- **Reduced** development **documentation**
- **Continous evaluation** of the obtained value and of the risks of changes

Agile ethics

- www.agilemanifesto.org

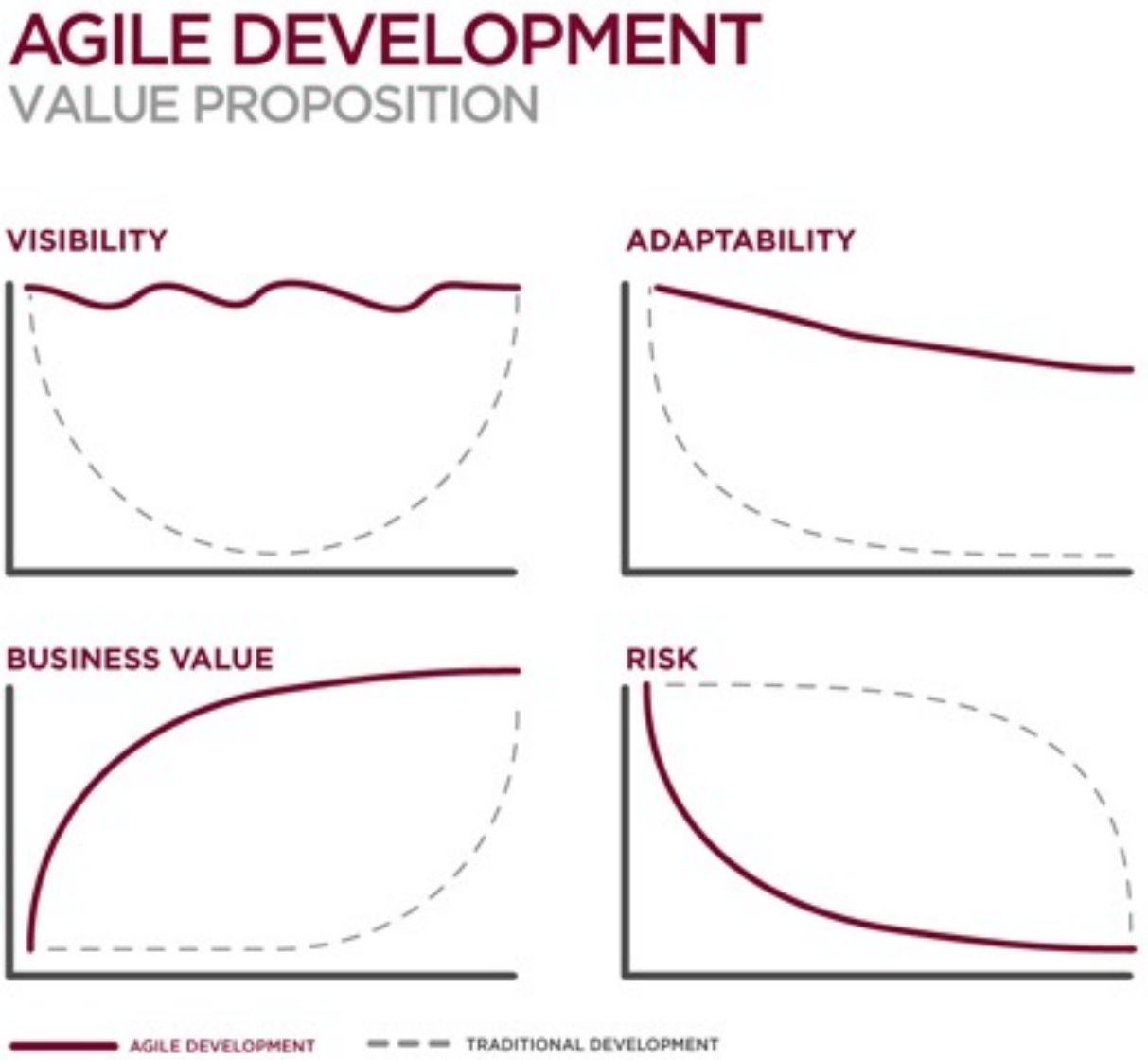
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

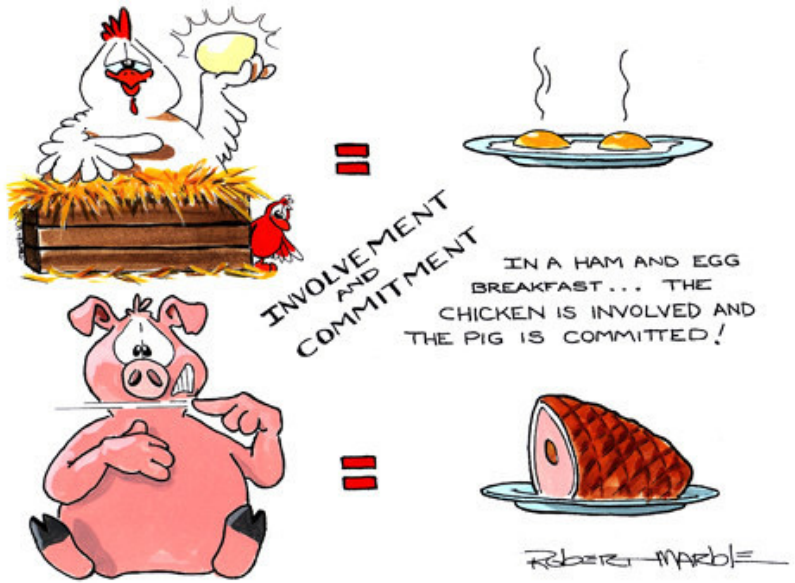
That is, while there is value in the items on the right, we prefer the items on the left.

Warning:
Management people tend to prefer the things on the right over the things on the left

Agile value proposition



Chicken and pigs



Agile development processes

- There are many agile development methods; most minimize risk by developing software in short amounts of time
- The requirements are initially grouped in stories and scenarios
- Then the tests for each scenario are agreed with the user, before any code is written
- Each code is tested against its scenario tests, and integrated after it passes its unit tests

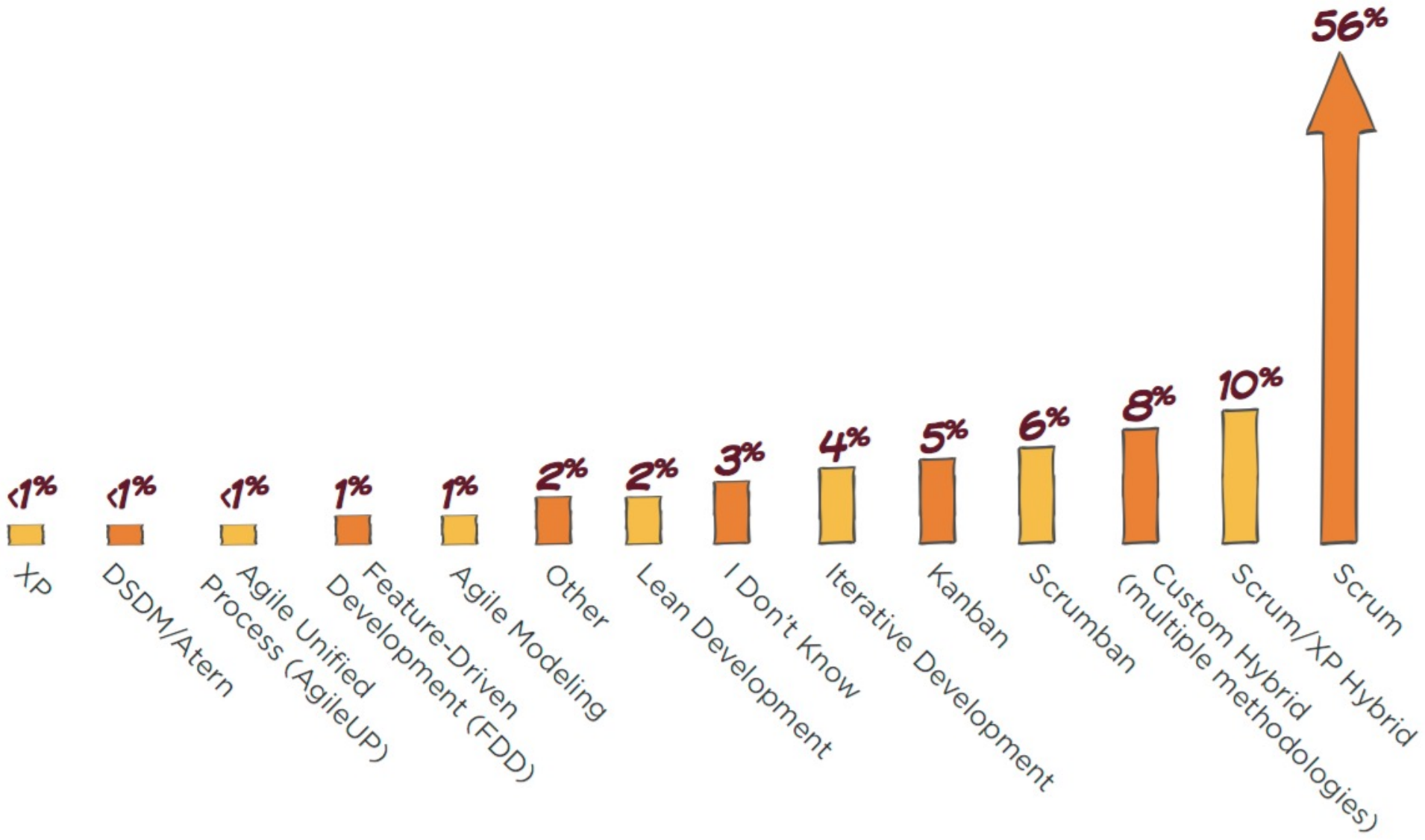
Anti-methodology?

The Agile movement is not anti-methodology, in fact, many of us want to restore credibility to the word *methodology*.
We want to restore a balance.

We embrace *modeling*, but not in order to file some diagram in a dusty corporate repository.
We embrace *documentation*, but not hundreds of pages of never-maintained and rarely-used tomes.
We *plan*, but recognize the limits of planning in a turbulent environment.

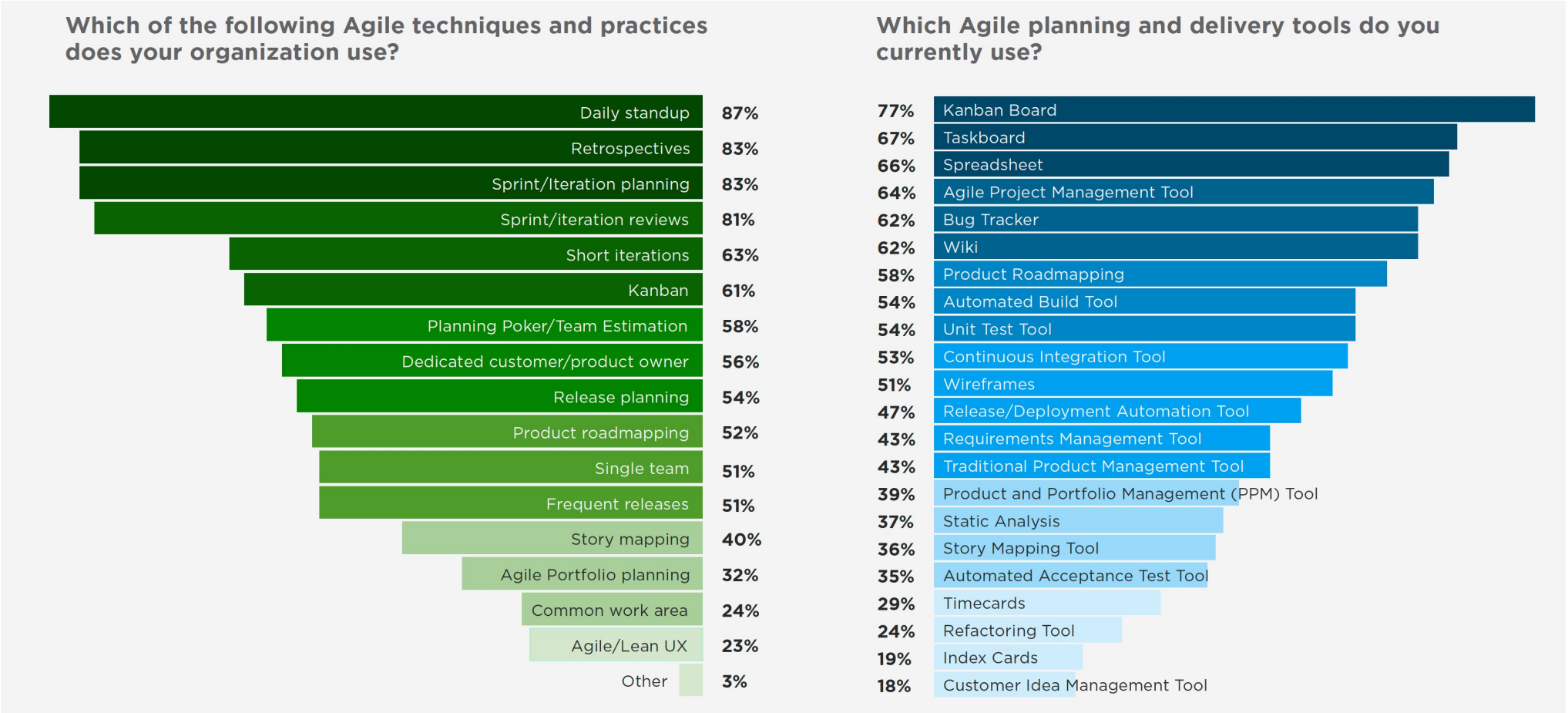
Jim Highsmith, *History: The Agile Manifesto*

Popularity of agile methods



Source: VersionOne, 9th State of agile survey, 2015

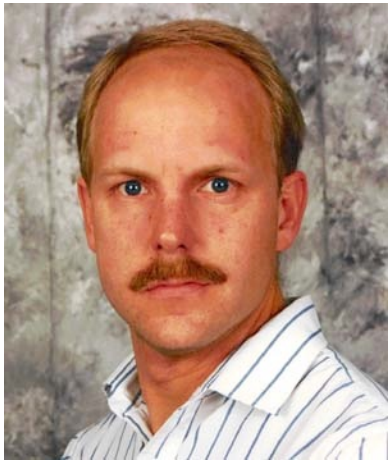
Popularity of practices and tools



Source: Digital.AI, 15th State of agile survey, 2021

eXtreme Programming (XP)

“Extreme Programming is a discipline of software development based on values of *simplicity*, *communication*, *feedback*, and *courage*”.



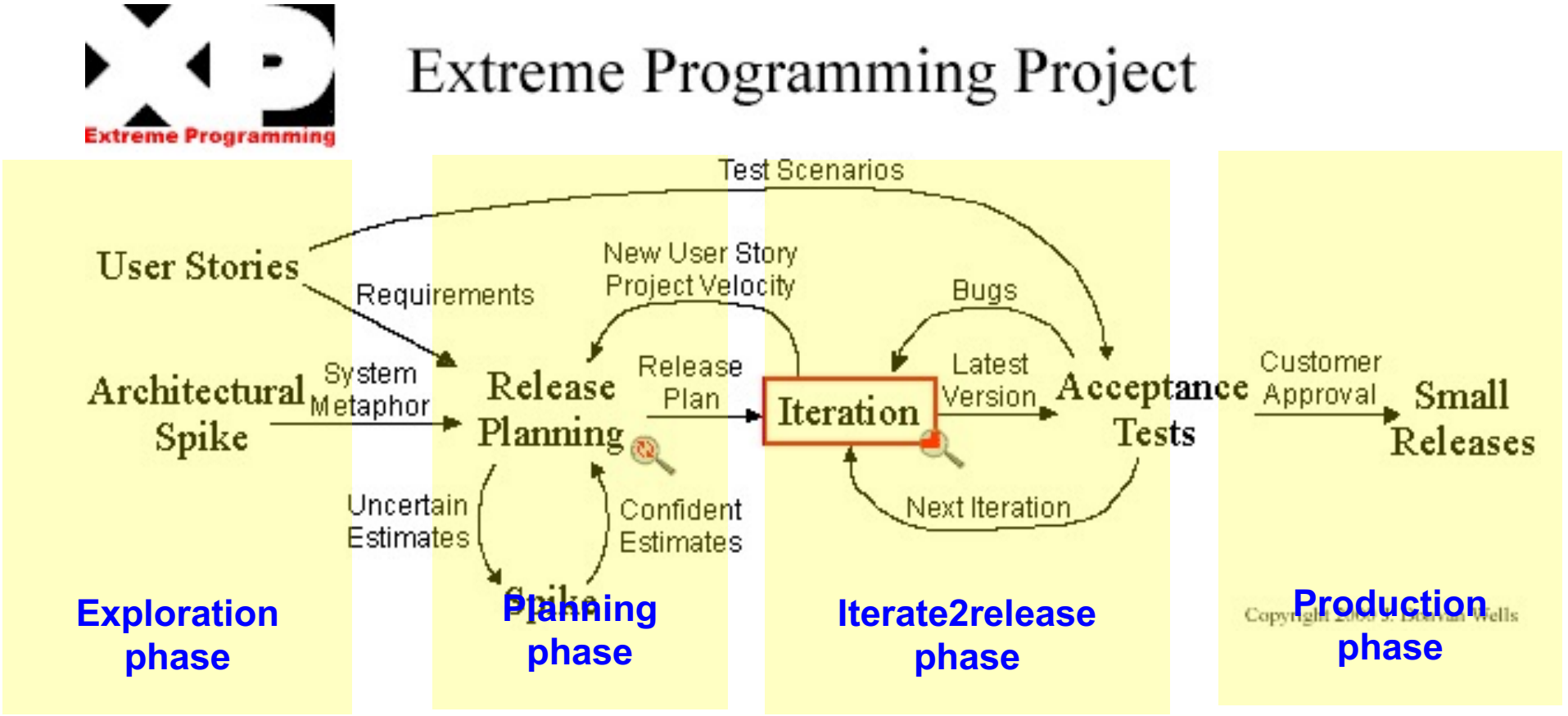
Kent Beck

XP: the developing team

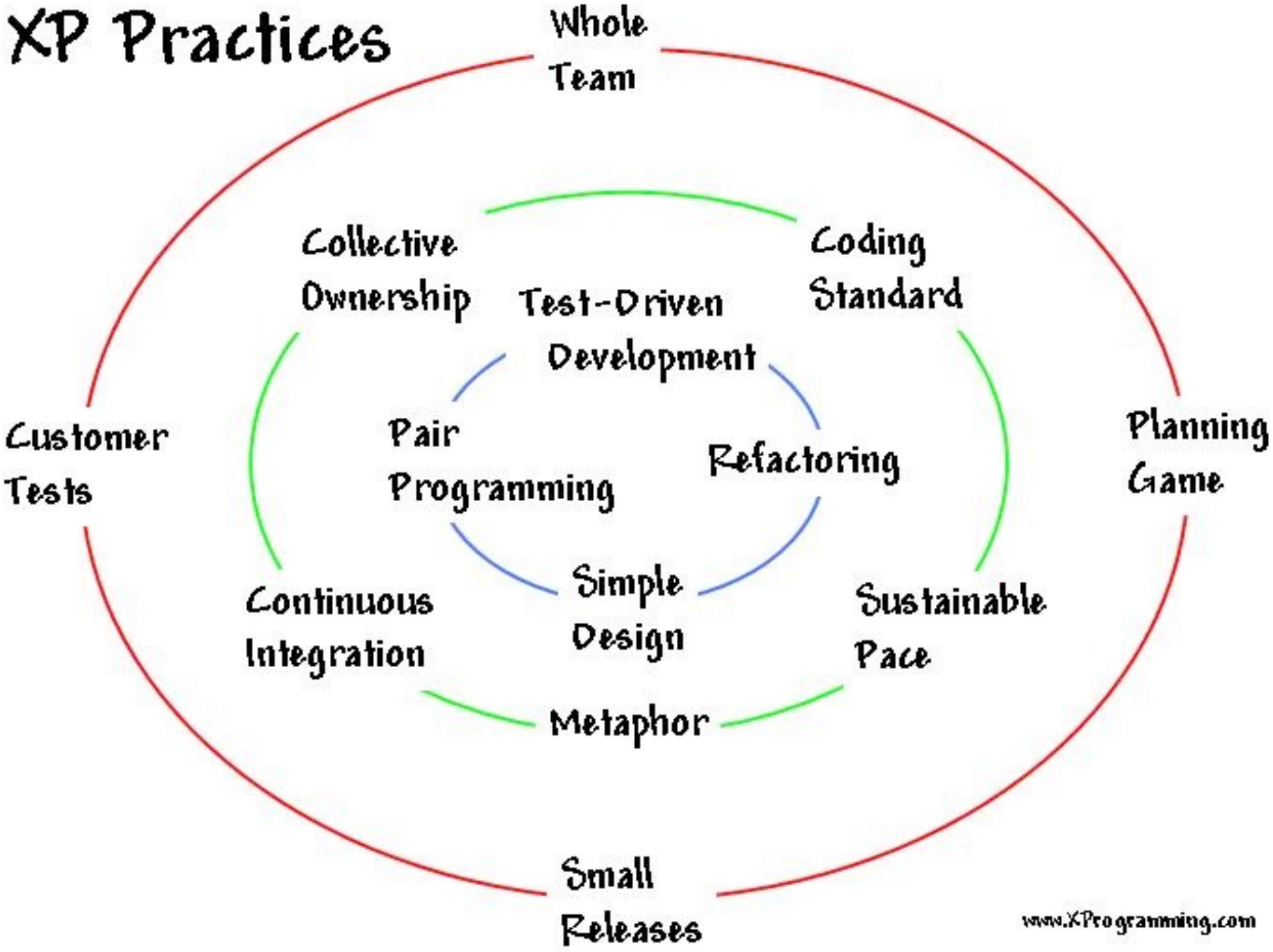
- The team is small (less than 10 people usually) and includes a representative of the customer;
- All members work in the same room
- The team approach is based on simplicity, transparency (everybody knows all) and flexibility

eXtreme Programming (XP)

www.extremeprogramming.org



XP Practices



www.XProgramming.com

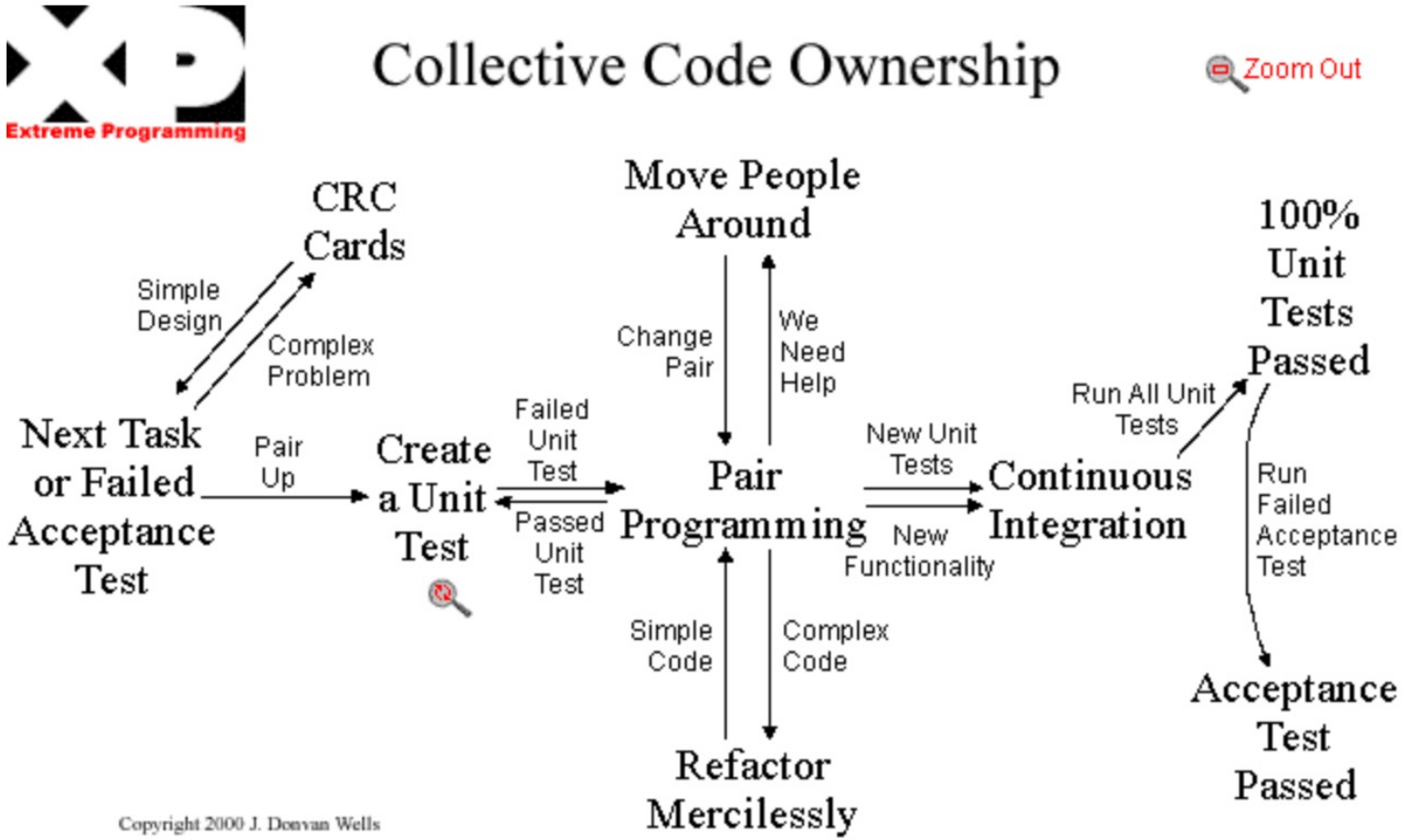
XP practices

- On-site customers
- Requirements are given as “user stories”
- Daily planning game
- Small releases
- “Test first, then code” (and customer tests)
- Collective ownership of the code
- Pair programing
- Systematic use of coding standards
- The architectural metaphor
- Refactoring
- Continuous integration
- Simple design
- Work 40 hours per week

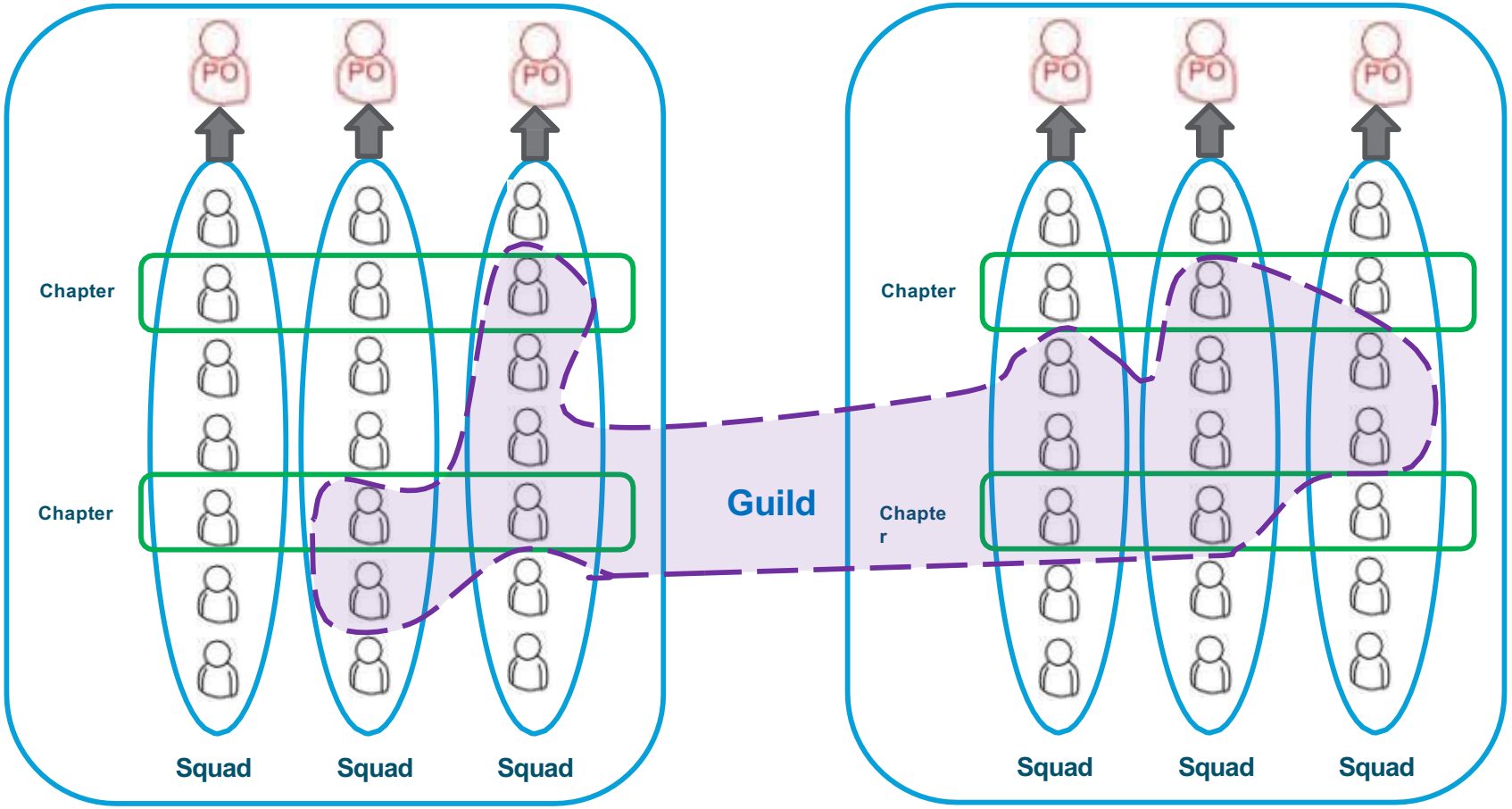
Agile for multiple teams

- A large company usually has several products, each developed by one or more teams
- Original Scrum is for one *small* team (3-7 people), so how can we coordinate several teams?

Continuous Integration in XP

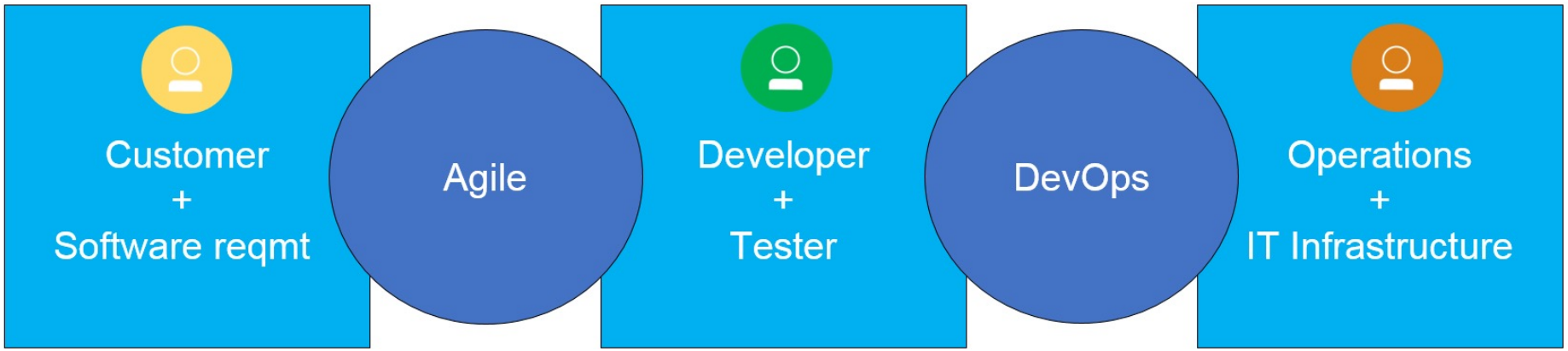


The Spotify model: Tribes, Squads, Chapters & Guilds



Continuous integration (CI) is a central focus of Spotify's ecosystem. Spotify depends on CI as an internal process to reduce the time and effort required by each feature integration—and to successfully deliver a product version suitable for release at any moment

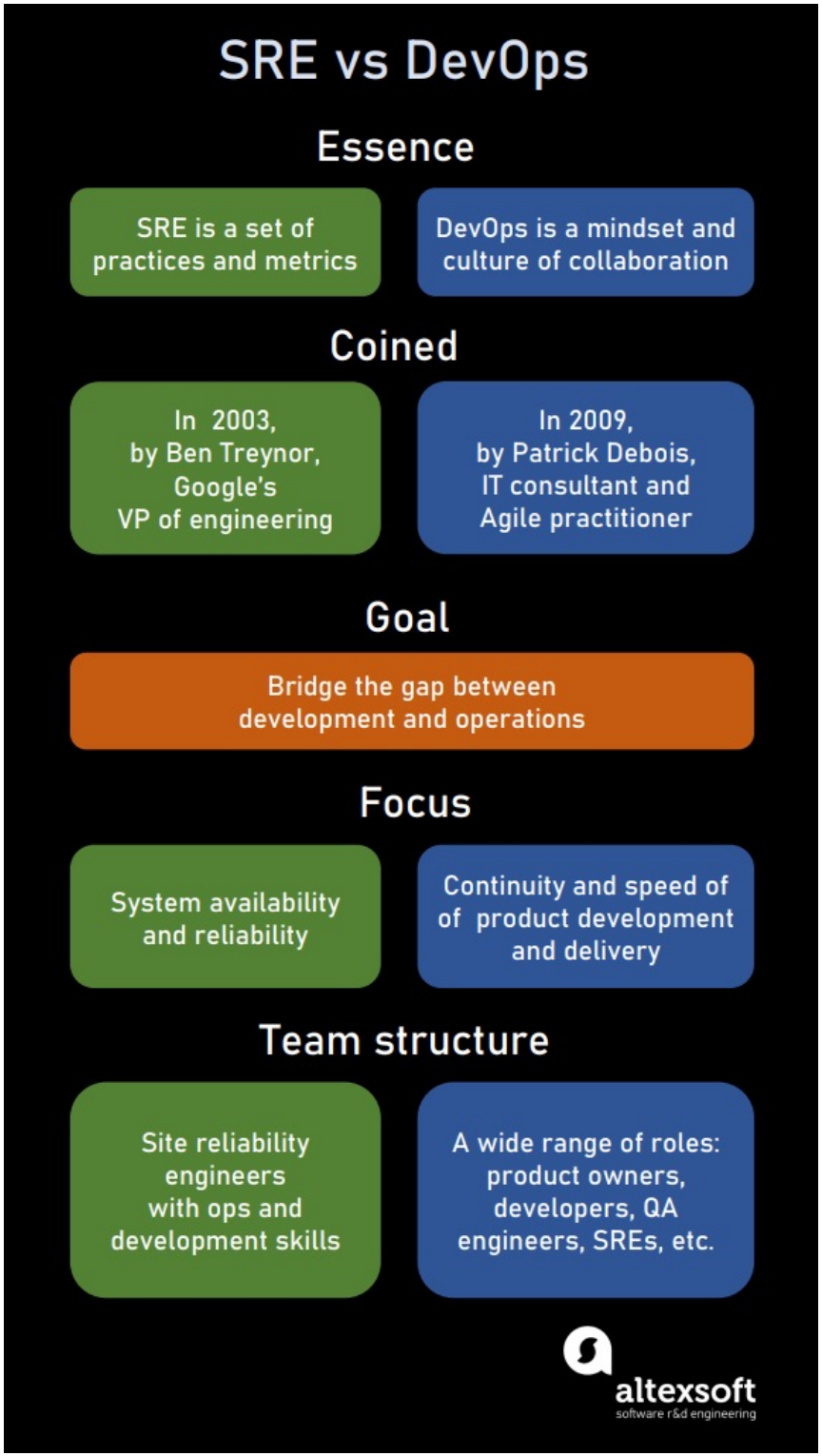
From Agile to DevOps



DevOps variants: Site Reliability Engineering SRE at Google (2004)

DevOps was introduced in 2009 (talk by Flickr people)

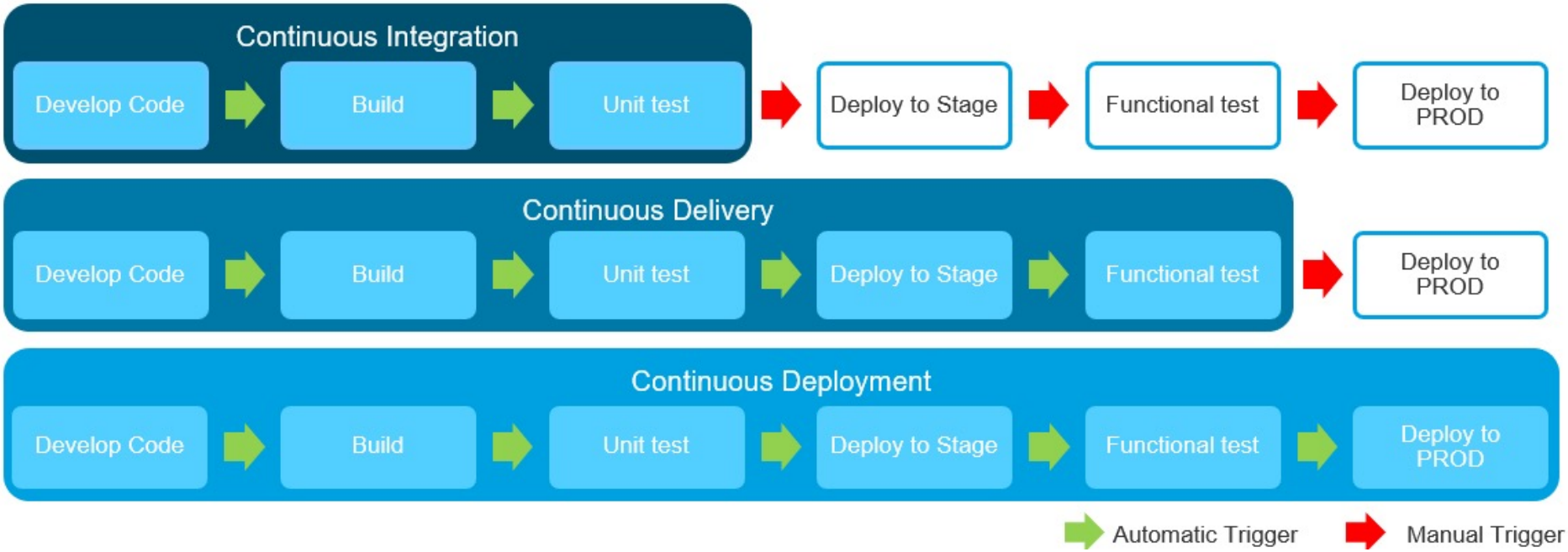
https://www.altexsoft.com/blog/devops-vs-site-reliability-engineering/



DevOps values: CALMS

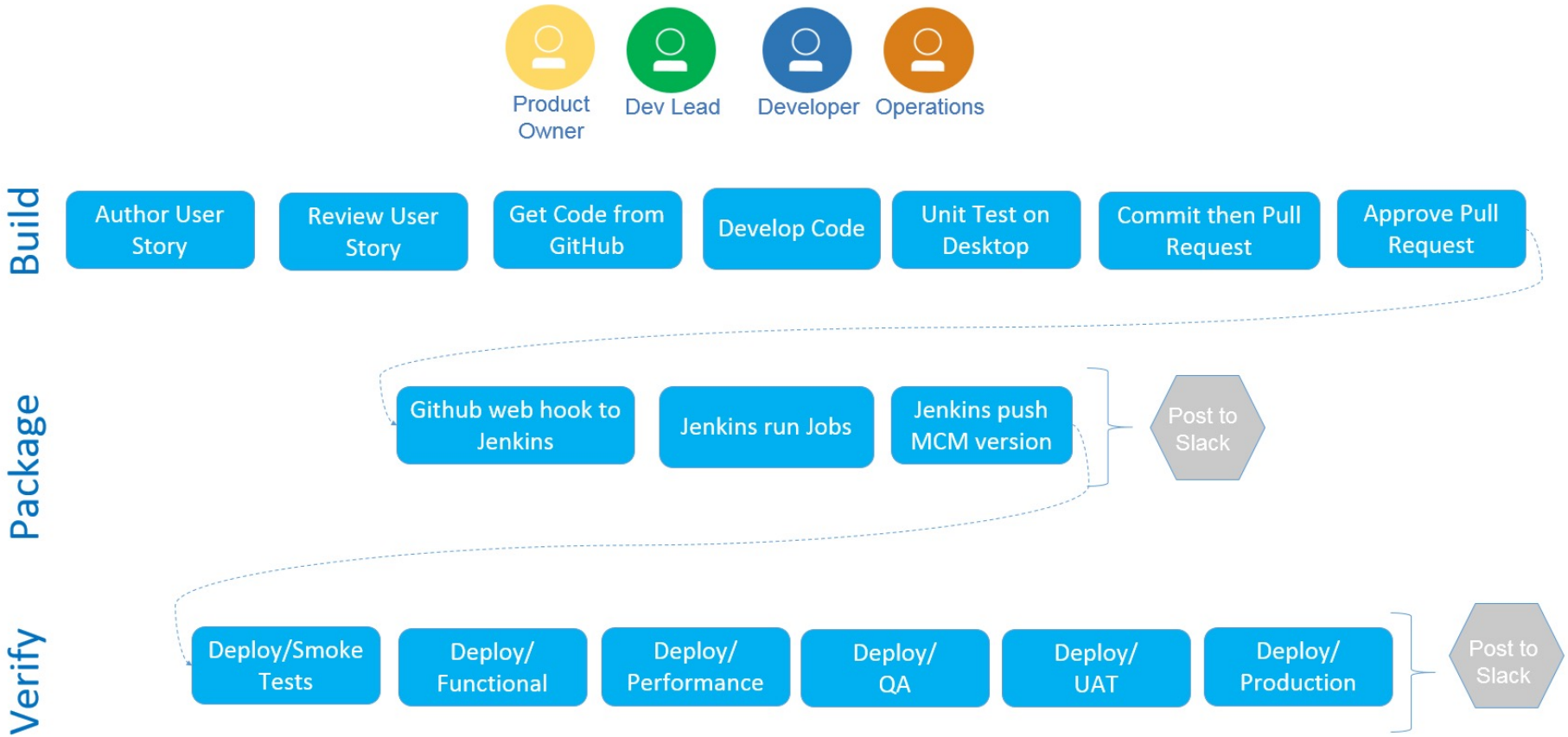
- **Culture**
  - Focus on people
  - Embrace change and experimentation
  - Operations integrated throughout the development process.
- **Automation**
  - Continuous delivery
  - Infrastructure as code
  - Driving entire lifecycle with the automation of processes, tasks, and decisions.
- **Lean**
  - Focus on producing value for end-user
  - Small batch sizes
  - Service broken into smaller pieces and released frequently.
- **Measurement**
  - Measure everything
  - Show the improvement
  - Integrated measurement and management designed into the entire development lifecycle.
- **Sharing**
  - Open information sharing
  - Collaboration and communication
  - Feedback built into the process at several steps but with emphasis on production, especially customer experience

Continuous: integration, delivery, deployment

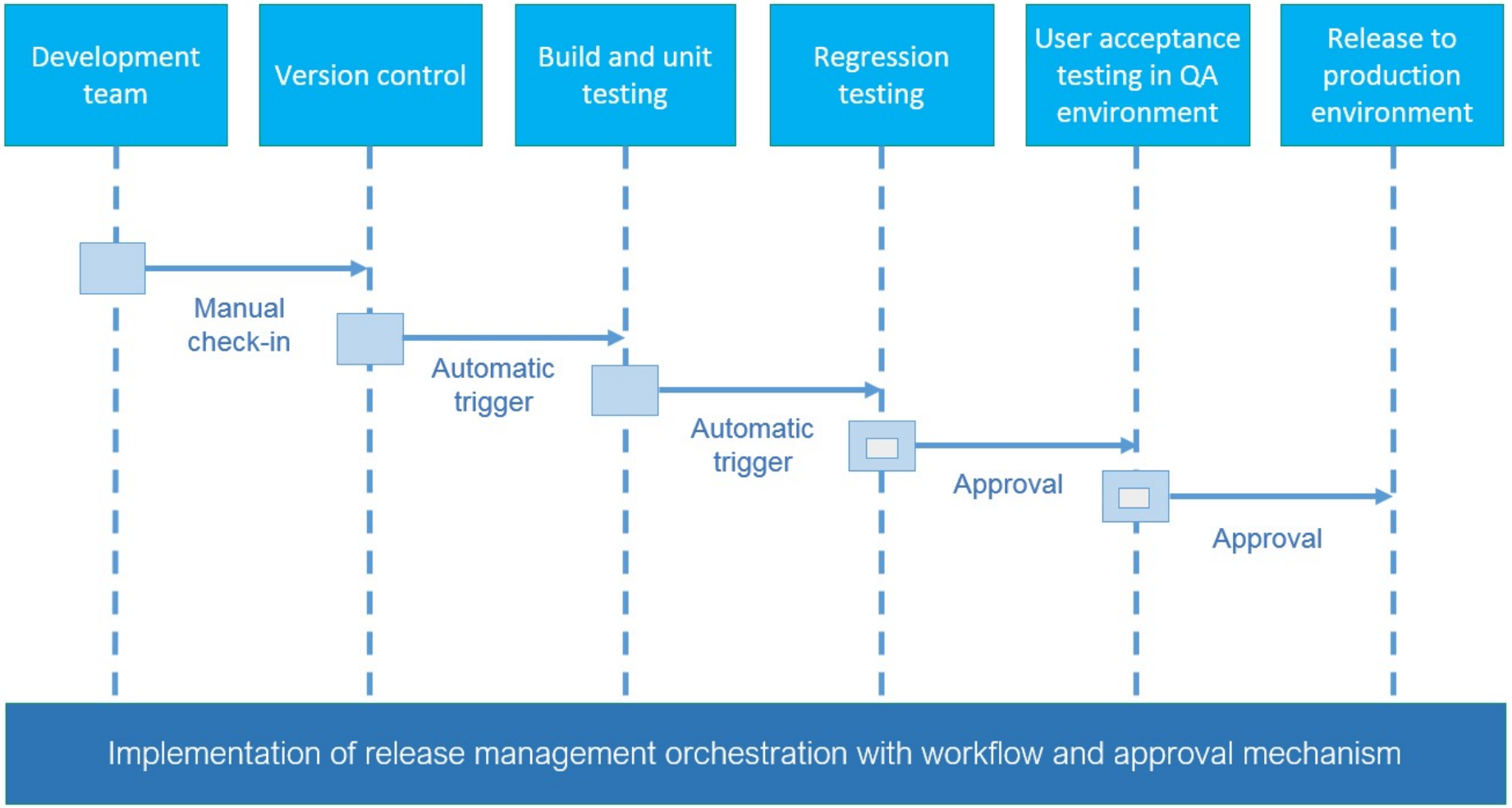


https://medium.com/ibm-garage/devops-adoption-approach-plan-and-design-be3d1ba67c8

A DevOps process

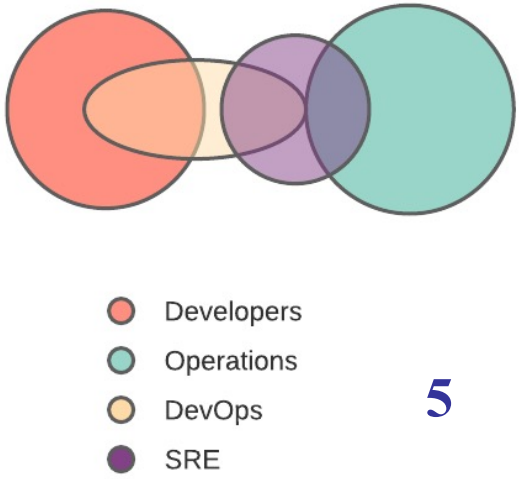


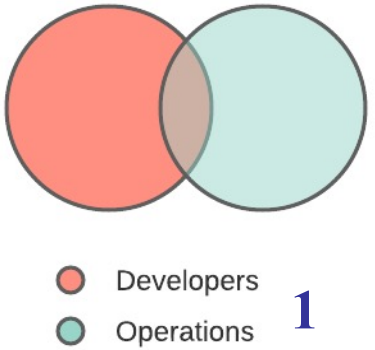
DevOps automation

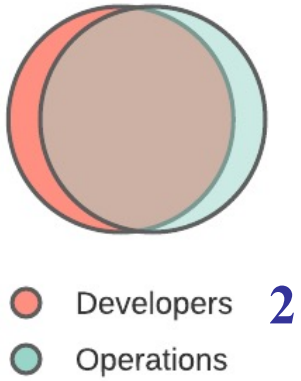


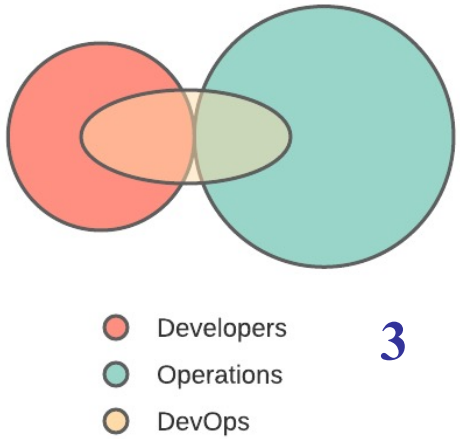
Team topologies

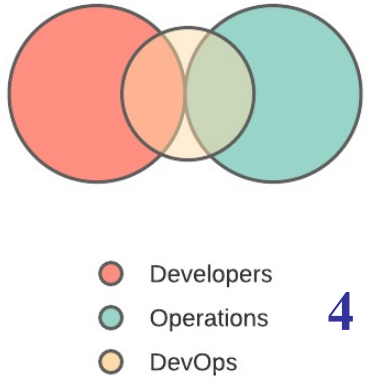
1. Development and operations collaboration
2. Shared operations
3. DevOps as a service
4. DevOps advocacy
5. Site reliability engineering (SRE)
6. Container driven

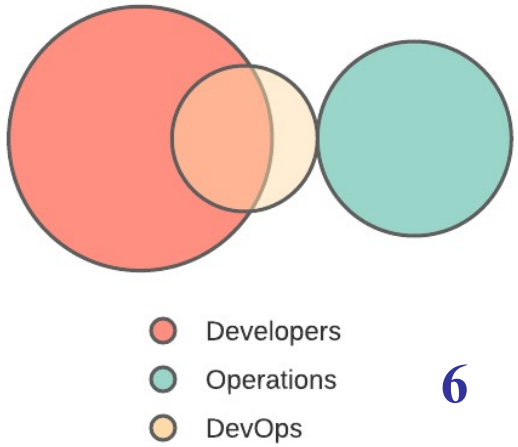












Conclusions

Minimizing paperwork and accelerating development speeds are top goals of agile

The term “agile” is ambiguous and there are many flavors of agile: DevOps is agile

We assume that: DevOps teams divide projects into sprints, use daily status meetings and post mortem analysis, practice self tracking, use systematically automation tools

References

- Humphrey, Managing the Software Process
- Brooks, The Mytical Man-Month
- Beck, Extreme Programming Explained
- Poppendieck, Lean Software Development: an Agile toolkit

Useful sites

- www.scrumalliance.org
- www.controlchaos.com
- www.digital.ai
- www.devops.com
- tulip.co/blog/agile-manufacturing-vs-lean-manufacturing/

Questions?