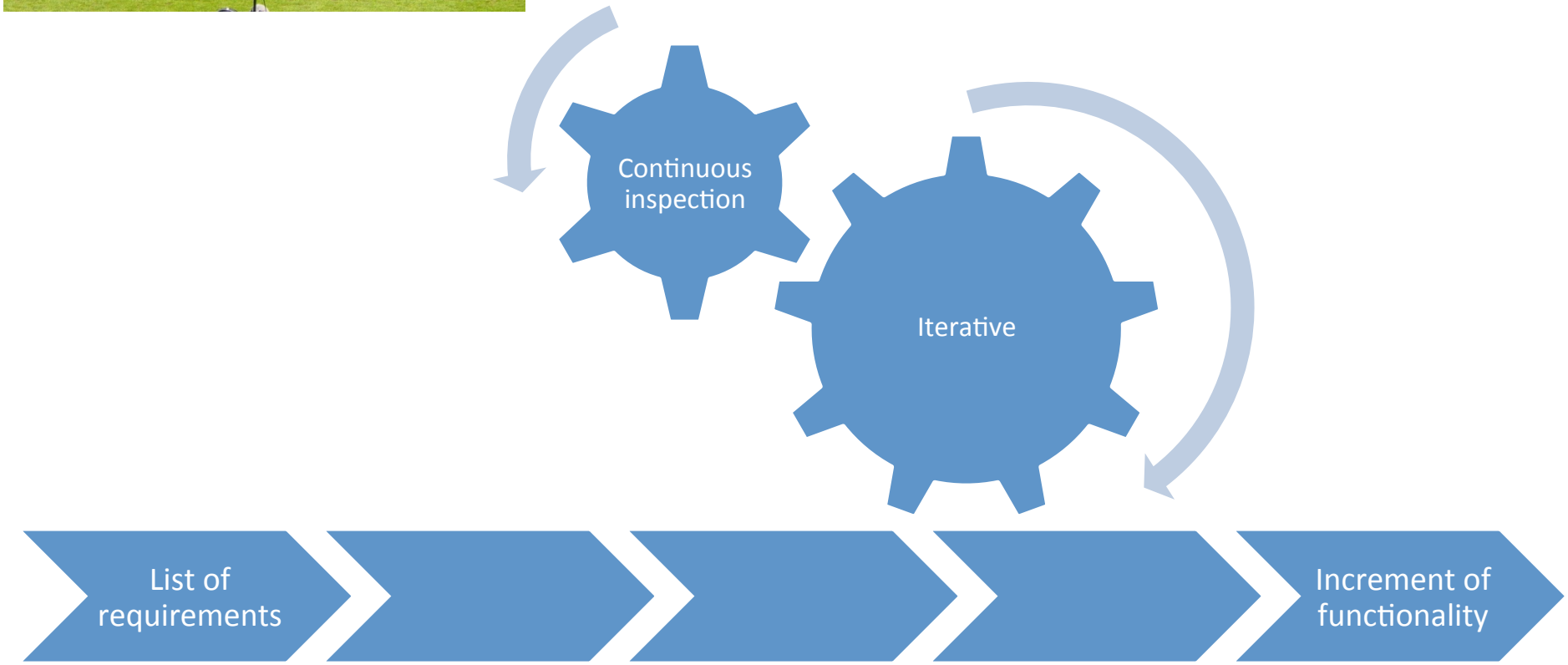
SCRUM & XP – Methodologies & Practices

Robert Feldt, 2012-03-19

Agile Dev Processes, Chalmers

Definitions



Why Scrum?

- We need to do a better job of change management. We had too many outside distractions.
- We need customer feedback during the iterative development approach we're taking.
- The users gave us a huge list of requirements. We knew we weren't going to be able to deliver everything they wanted.
- Development took place in focused chaos, and there was no one to go to with questions. We need a way to structure the chaos somehow, because all NBOs must deal with that.
- We have been used to thinking in terms of years for development; now we have to turn out products in months.
- We're chasing an emerging market. It changes weekly.
- We wasted a lot of time estimating and developing test plans for features that we never developed.
- We should have cancelled this project earlier. It took almost two years to recognize that.
- We need well-defined phases and someone who is close enough to see progress and determine what we can check off.

[Rising2000]

History of SCRUM

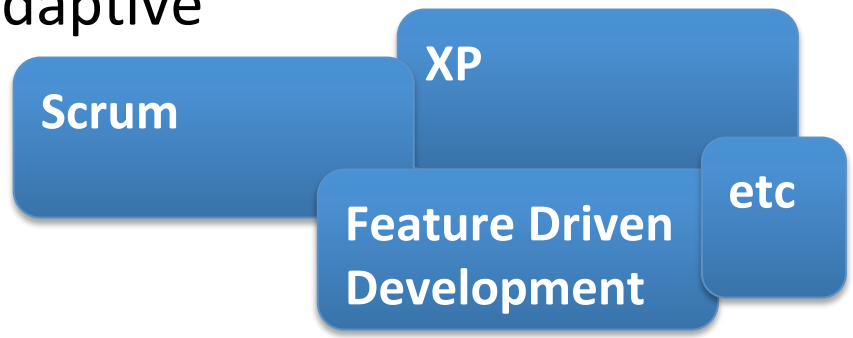
- 1986 - Hirotaka Takeuchi and Ikujiro Nonaka described a new holistic approach that increases speed and flexibility in commercial new product development
- 1991 - DeGrace and Stahl referred to this approach as SCRUM, a rugby term mentioned in the article by Takeuchi and Nonaka
- Early 1990s, Ken Schwaber used an approach that led to Scrum at his company, Advanced Development Methods
- At the same time, Jeff Sutherland, John Scumniotales, and Jeff McKenna developed a similar approach at Easel Corporation and were the first to call it Scrum
- 1995 Sutherland and Schwaber jointly presented a paper describing Scrum at OOPSLA '95 in Austin, TX, its first public appearance
- 2001, Schwaber teamed up with Mike Beedle to describe the method in the book Agile Software Development with Scrum.

Defined Process Control vs. Empirical Process Control

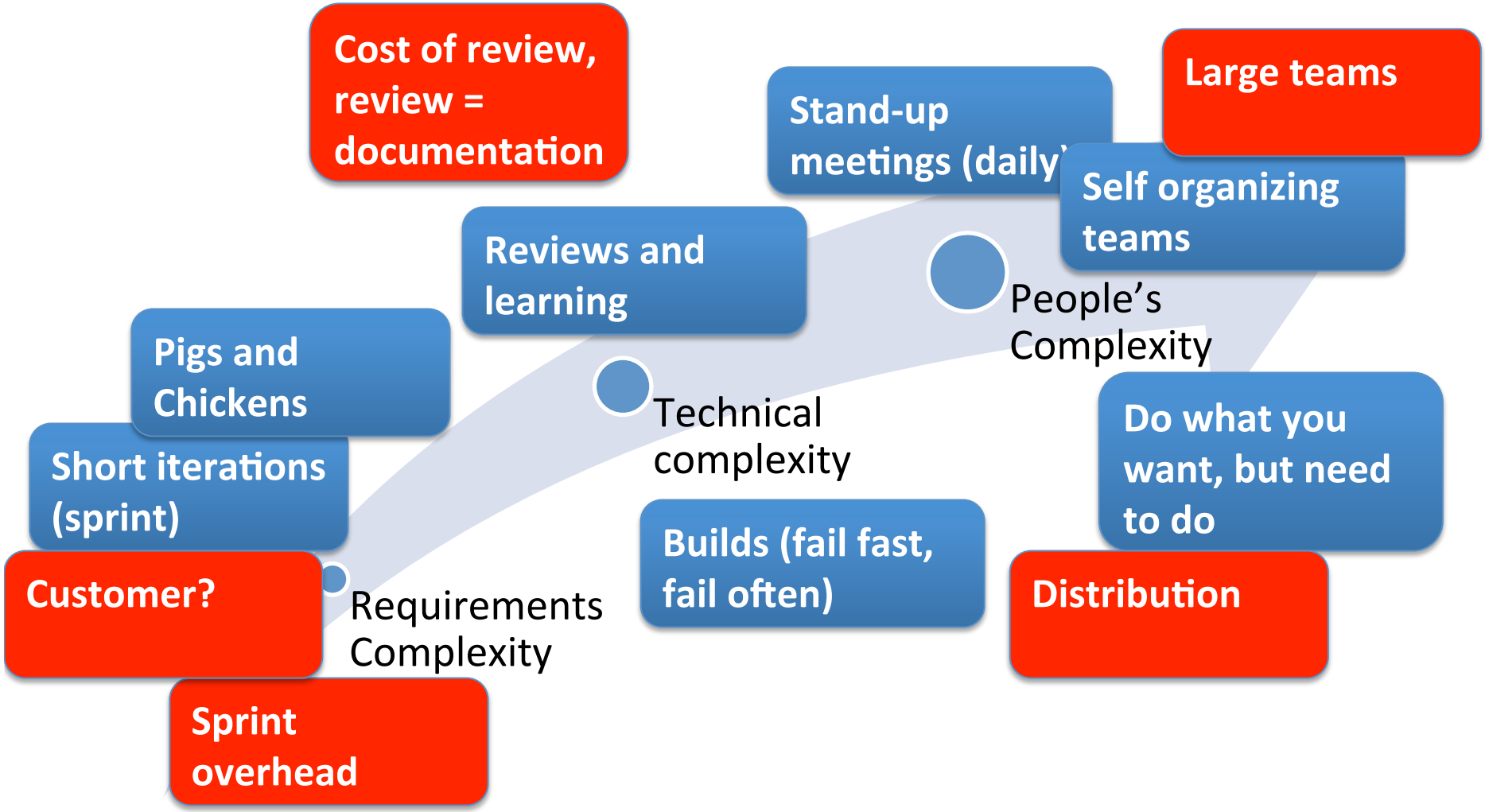
- Laying out a process that repeatedly will produce acceptable quality output is called *defined process control*
- When defined process control cannot be achieved because of the complexity of the intermediate activities, something called empirical process control has to be employed

- Defined Process control (“non-agile”)
  - Planning heavy
  - Assumes (more) static environment
  - Longer iterations
  - Change Management intensive
  - Typical pre-study heavy
  - Assumes good estimations (and as we know… estimations are negotiations)…
  - Process and Management → Control over Actual work (often seen as bureaucratic)

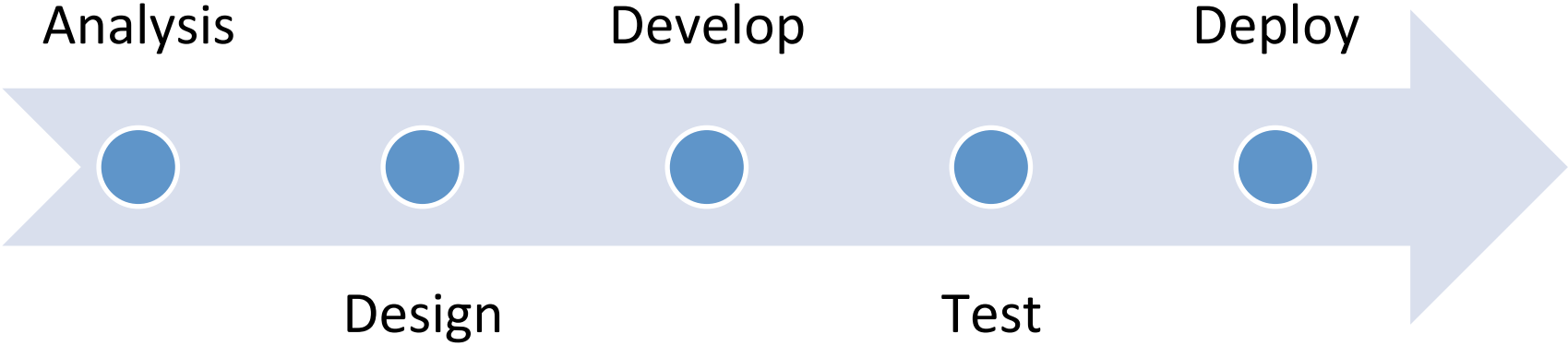
- Empirical Process control (Agile)
  - Change is reality
  - Shorter iterations
  - Problem vs. solution space (empowering the developers)
  - Just-enough (management, documentation etc)
  - Self organizing teams
  - Continous “customer” interaction
  - NOT UNPLANNED rather adaptive



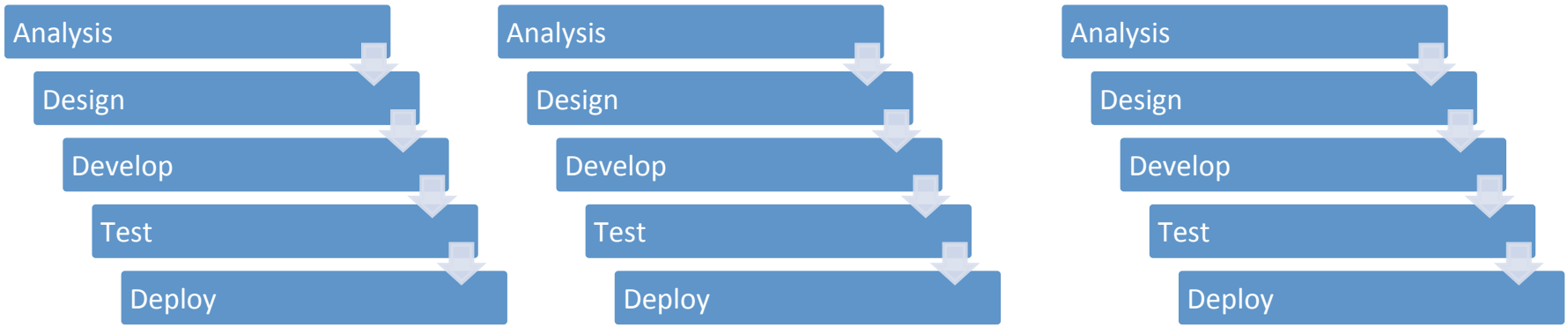
Handling complexity



Traditional Waterfall Approach

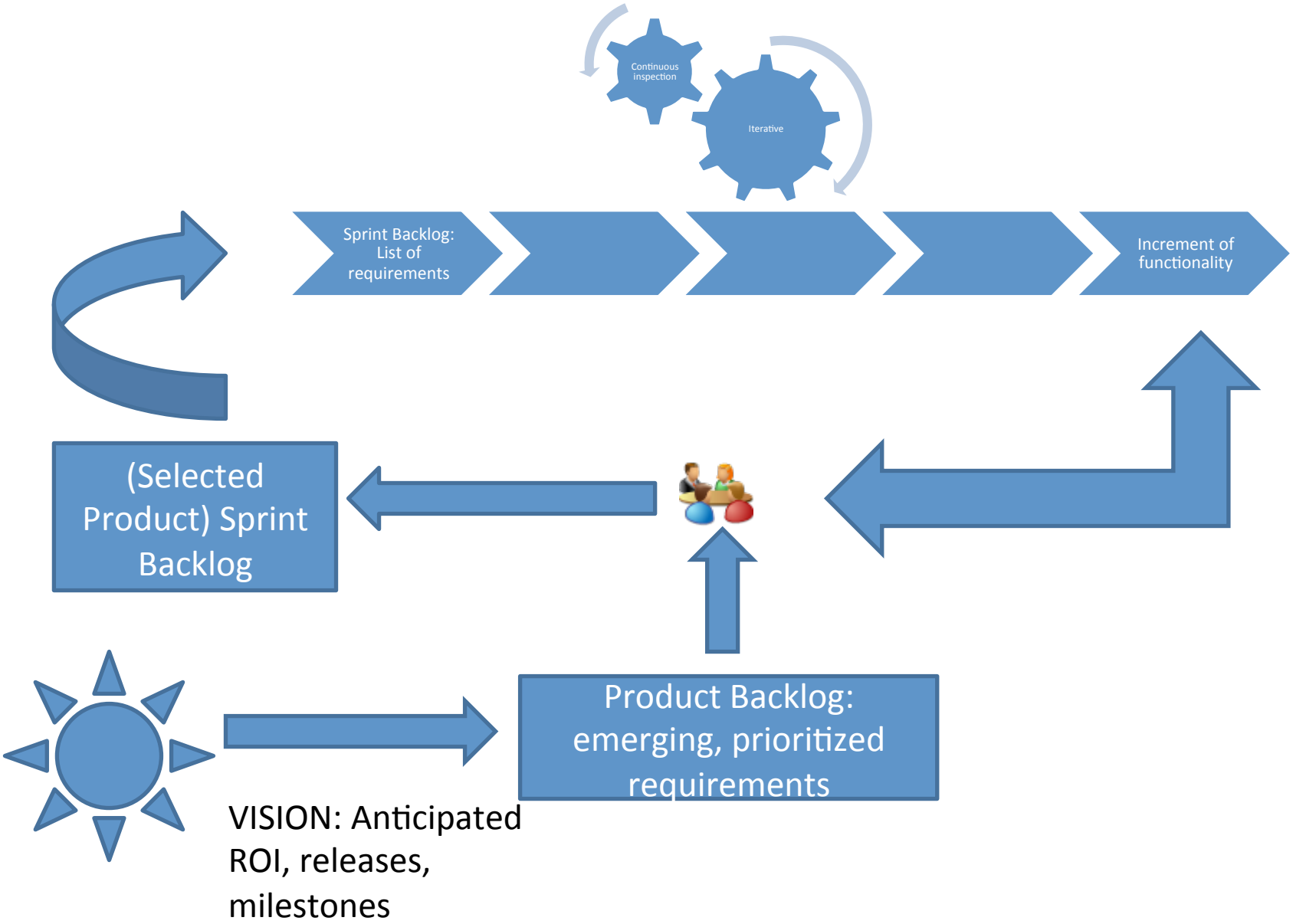


Agile Approach



Most Agile methodologies have similar concepts

SCRUM – What is the Process?



Roles

Pigs – Involved in the project

- Product owner
  - Represents interests of everyone with a stake in the project and resulting system
  - Responsible for initial/ongoing funding, initial overall requirements, ROI objectives, release plans
- Scrum master
  - responsible for the Scrum process
  - teaching Scrum to everyone involved in the project, for implementing Scrum so that it fits within an organization's culture and still delivers the expected benefits for ensuring that everyone follows Scrum rules and practices.
- Scrum team
  - Self-managing, self-organizing, and cross-functional, and they are responsible for figuring out how to turn a list of requirements into an increment of functionality

Chickens – interested in the project

- Stakeholders
- Users

Artifacts

Product Vision

- What are the aims and objectives of the planned product
- Which markets to cover,
- Which competitors to compete,
- What is product's differentiation etc

Artifacts (Cont...)

Product Backlog

- **Definition**: The requirements for the system or product being developed by the project(s) are listed in the Product Backlog.
  Responsible: Product Owner for the contents, prioritization, and availability of the Product Backlog.
- **Properties**:
  - Never complete
  - Merely an initial estimate of the requirements
  - Evolves as the product and the environment in which it will be used evolves
  - Dynamic - management constantly changes it to identify what the product needs to be appropriate, competitive, and useful.
  - Exists as long as a product exists

Artifacts (Cont…)

Sprint Backlog

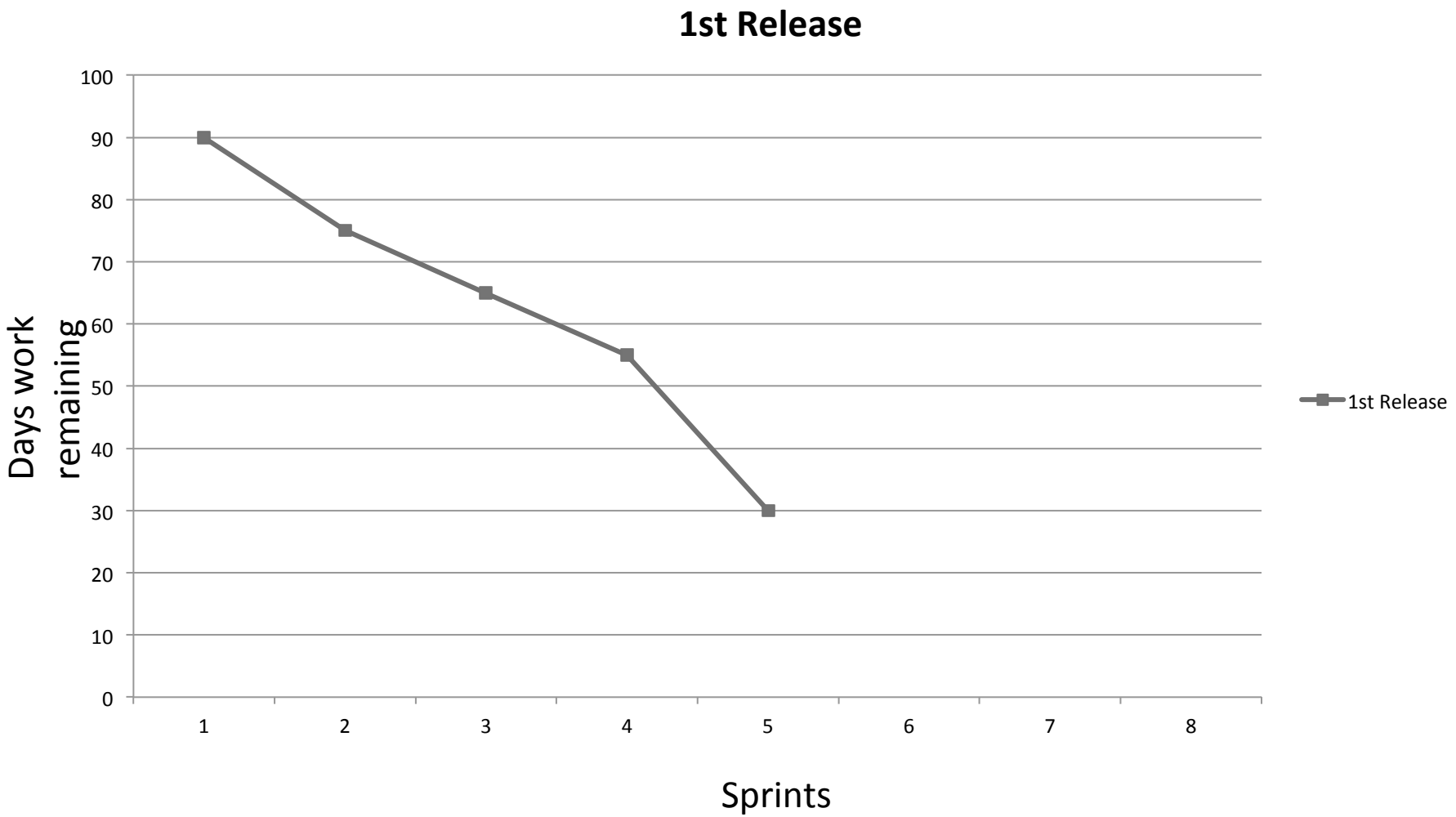
- **Definition**: defines the work, or tasks, that a Team defines for turning the Product Backlog it selected for that Sprint into an increment of potentially shippable product functionality
- **Responsible**: The Team compiles an initial list of these tasks in the second part of the Sprint planning meeting
- **Properties**:
  - Should contain tasks such that each task takes roughly 4 to 16 hours to finish
  - Tasks longer than 4 to 16 hours are considered mere placeholders for tasks that haven't yet been appropriately defined
  - Only the Team can change it
  - Highly visible, real time picture of the current Sprint

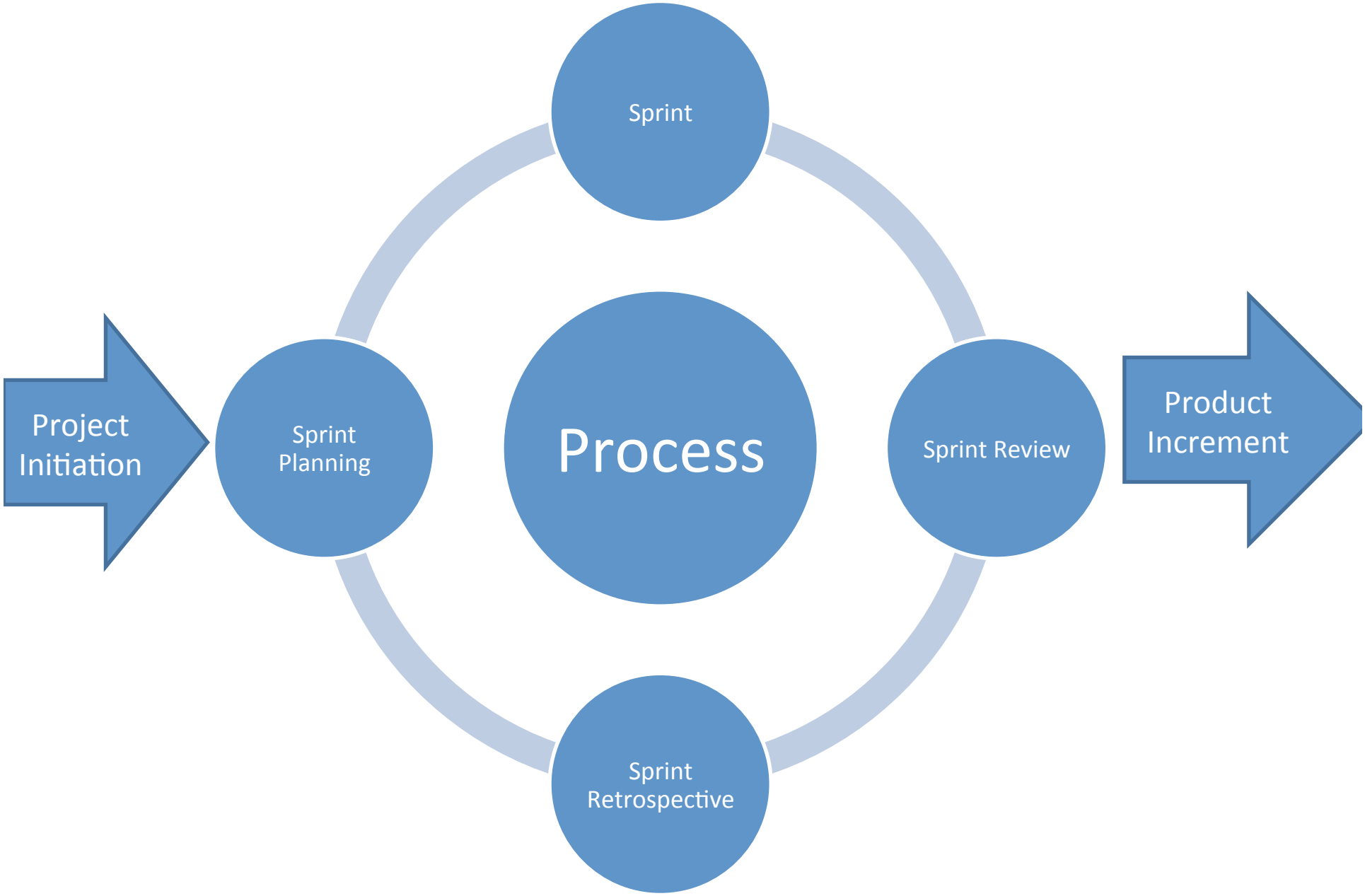
Artifacts (Cont…)

Burndown Chart

- **Definition**: shows the amount of work remaining across time
- **Responsible**: The Team compiles an initial list of these tasks in the second part of the Sprint planning meeting
- **Properties**:
  - Excellent way of visualizing the correlation between the amount of work remaining at any point in time and the progress of the project Team(s) in reducing this work
  - Allows to “what if” the project by adding and removing functionality from the release to get a more acceptable date or extend the date to include more functionality
  - Only the Team can change it
  - Highly visible, real time picture of the current Sprint

Artifacts (Cont...)




Project Initiation
Sprint Planning
Sprint
Process
Sprint Review
Product Increment
Sprint Retrospective

Project Initiation

Product Vision

- might be vague at first, perhaps stated in market terms rather than system terms, but becomes clearer as the project moves forward

Product Backlog

- list of functional and nonfunctional requirements that, when turned into functionality, will deliver this vision
- Prioritized so that the items most likely to generate value are top priority

Release Plan

- Based on the product backlog and prioritized items

Sprint Team

- Product owner
- Scrum master
- The Team

Project Initiation (Cont…)

Team Formation

- Introductions and backgrounds
- Team name
- Team room and daily Scrum time/place
- Development process for making product backlog done
- Definition of “Done” for product and Sprint Backlog items
- Rules of development
- Rules of etiquette, and
- Training in conflict resolution

Sprint Planning Meeting – Part 1

Purpose: Commit to Product Backlog for the next Sprint

Steps

- Calculate The Team capacity. Every resource is 100% allocated less 10% for forward looking Product Backlog analysis and 10% for severity 1 issues
- Commit to Product Owner as much backlog as the Team believes it can turn into a “Done” increment in the Sprint

Sprint Planning Meeting – Part 2

Purpose: the Team plans out the Sprint

Steps

- Self managing teams requiring a tentative plan to start the Sprint
- Tasks that compose this plan are placed in a Sprint Backlog

Sprint

Daily Scrum: the team gets together for a 15-minute meeting

Each member answers

- What have you done on this project since the last Daily Scrum meeting?
- What do you plan on doing on this project between now and the next Daily Scrum meeting?
- What impediments stand in the way of you meeting your commitments to this Sprint and this project

Do not forget

- It is the inspect and adapt process control for the Team
- The 3 questions provide the information the Team needs (inspect) to adjust its work to meet its commitments

Sprint (cont…)

What does it mean when a team member says “done”

- Code adheres to the standard
- Is clean
- Has been refactored
- Has been unit tested
- Has been checked in
- Has been built
- Has had a suite of unit tests applied to it

Sprint Review

Purpose

- Team presents what was developed during the Sprint to the Product Owner and any other stakeholders who want to attend
- Collaborative work session to inspect and adapt: the most current Product Backlog and the functionality increment are for inspection, the adaptation is the modified Product Backlog

SCRUM means Visibility

- Visibility of Progress, Process and Sociology
- Daily SCRUM: to feel the tone, attitude, and progress of a Sprint
- Sprint review meeting: monthly insight into whether the project is creating valuable functionality, as well as the quality and capabilities of that functionality
- Product Backlog: details a project's requirements and lists them in order of priority
- Formal reports: end of each Sprint, static snapshot of the project's progress

Scrum Retrospective

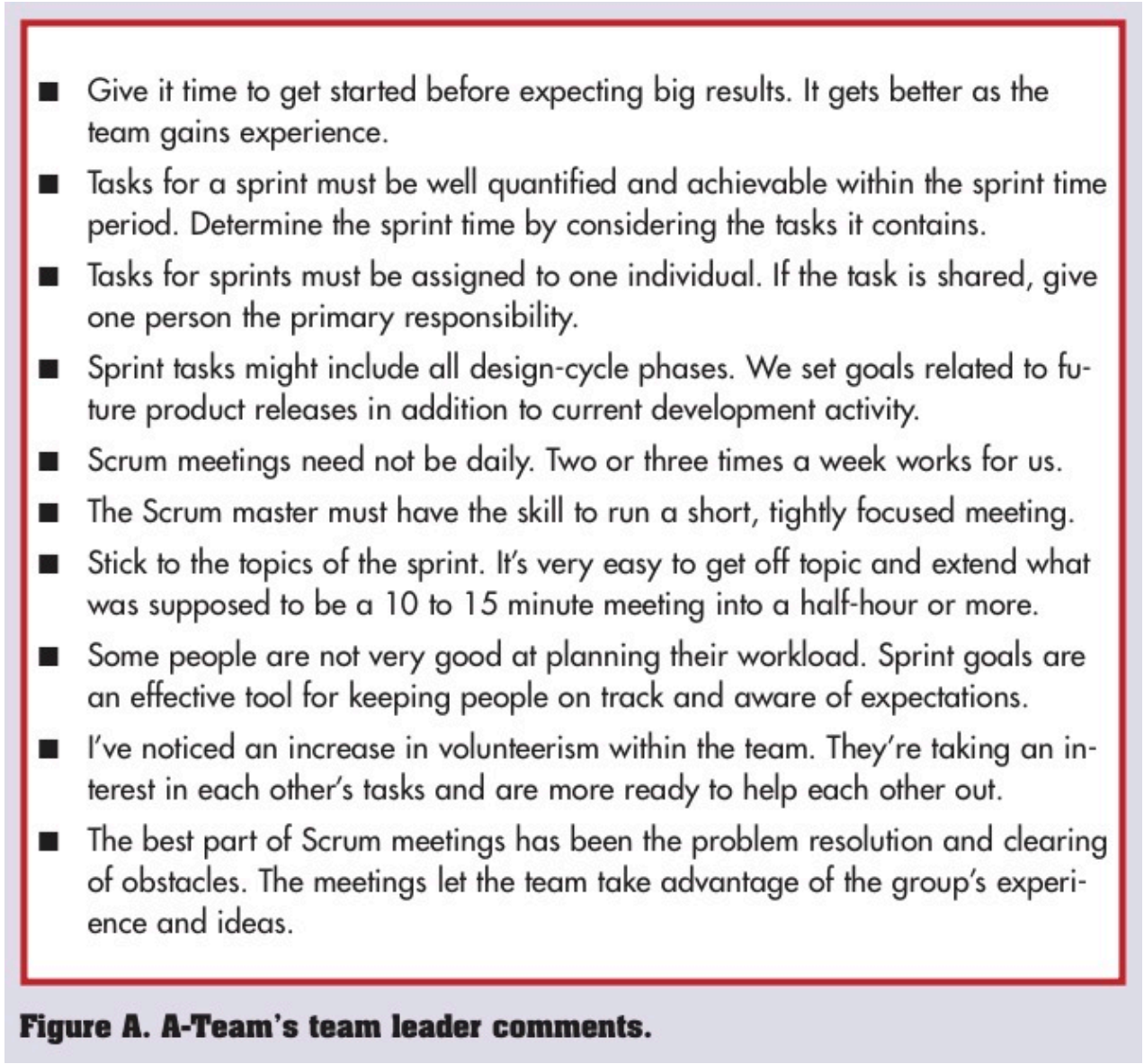
Purpose

- Scrum master encourages the Team to revise, within the Scrum process framework and practices, its development process to make it more effective and enjoyable for the next Sprint

Do not forget

- Should be time-boxed to 1-3 hours
- While discussing issues the Team figures out itself how to address the issues

Scrum Team Leader comments



- Give it time to get started before expecting big results. It gets better as the team gains experience.
- Tasks for a sprint must be well quantified and achievable within the sprint time period. Determine the sprint time by considering the tasks it contains.
- Tasks for sprints must be assigned to one individual. If the task is shared, give one person the primary responsibility.
- Sprint tasks might include all design-cycle phases. We set goals related to future product releases in addition to current development activity.
- Scrum meetings need not be daily. Two or three times a week works for us.
- The Scrum master must have the skill to run a short, tightly focused meeting.
- Stick to the topics of the sprint. It's very easy to get off topic and extend what was supposed to be a 10 to 15 minute meeting into a half-hour or more.
- Some people are not very good at planning their workload. Sprint goals are an effective tool for keeping people on track and aware of expectations.
- I've noticed an increase in volunteerism within the team. They're taking an interest in each other's tasks and are more ready to help each other out.
- The best part of Scrum meetings has been the problem resolution and clearing of obstacles. The meetings let the team take advantage of the group's experience and ideas.

Figure A. A-Team's team leader comments.

[Rising2000]

Scrum Effects

Att arbeta med Scrum:

Påverkan på arbetslag, kommunikation och social miljö

Stefan Hedberg

Scrum Effects

Abstract

Systems development methods has always been an important part of the systems development process. It affects not only the product, but also the organization itself. Today more and more companies are abandoning the more rigid traditional methods for the more flexible agile ones. A transition like this is bound to affect the product and the organization, but in what ways? Is it a difficult transition to make, and how does it affect social environment and dynamics? Does it change how people communicate information to each other? This study was conducted at a local branch of a larger international IT-company. The company had recently switched from a traditional waterfall method to the agile method Scrum. The study's aim was to see if and how this transition caused any changes in the way that the company develops their artifacts. A characteristic for Scrum is that people collaborate in small self-organizing teams. How is this affecting the social structures and the relationships between individuals? The study is of qualitative nature, where semi-structured interviews were conducted with people from the different Scrum teams. The study has shown that this kind of transition takes time, but that people are positive towards its benefits. The fact that individuals are no longer isolated with their own task, but are instead working closely together with their team members has led to, perhaps less, but better and more accurate products. To be in a tight team with a common goal has also made people more content with their social exchange and work situation, which positively affects moral. The physical closeness to the team has led to improvements in the communication and collaboration between teammembers. However it has also had the effect that the communication and collaboration with the other teams have deteriorated, due to the teams having differing ways of working, differing norms and differing goals.

eXtreme Programming

- Introduced by Ward Cunningham, Kent Beck, and Ron Jeffries.

- XP is what it says, an extreme way of developing software.
  - If a practice is good, then do it all the time.
  - If a practice causes problems with project agility, then don’t do it.

eXtreme Programming

- Team = 3-10 programmers + 1 customer
- Iteration => tested & directly useful code
- Req = User story, written on index cards
- Estimate dev time / story, prio on value
- Dev starts with discussion with expert user
- Programmers work in pairs
- Unit tests passes at each check-in
- Stand-up meeting daily: Done? Planned? Hinders?
- Iteration review: Well? Improve? => Wall list

XP practices

- Whole Team (Customer Team Member, on-site customer)
- Small releases (Short cycles)
- Continuous Integration
- Test-Driven development (Testing)
- Customer tests (Acceptance Tests, Testing)
- Pair Programming
- Collective Code Ownership
- Coding standards
- Sustainable Pace (40-hour week)
- The Planning Game
- Simple Design
- Design Improvement (Refactoring)
- Metaphor

Whole Team

A.K.A: Customer Team Member, on-site customer

- Everybody involved in the project works together as ONE team.
- Everybody on the team works in the same room. (Open Workspace)
- One member of this team is the customer, or the customer representative.

Pair Programming

- All program code is written by two programmers working together; a programming pair.

- Working in this manner can have a number of positive effects:
  - Better code Quality
  - Fun way of working
  - Information spreading
  - Skills spreading
  - …

Test-Driven development

A.K.A: Unit tests, Testing

- No single line of code is ever written, without first writing a test that tests it.

- All tests are written in a test framework like JUnit so they become fully automated.

Customer tests

A.K.A: Acceptance Tests, Testing

- The customer (or the one representing the customer) writes tests that verifies that the program fulfills his/her needs

Continuous Integration

- Daily build
  - A *working* new version of the complete software is released internally every night.

- Continuous build
  - A new version of the complete software is build as soon as some functionality is added, removed or modified.

Collective Code Ownership

- All programmers are responsible for all code.
- You can change any code you like, and the minute you check in your code somebody else can change it.
- You should not take pride in and responsibility for the quality of the code you written yourself but rather for the complete program.

Sustainable Pace

A.K.A: 40-hour week

- Work pace should be constant throughout the project and at such a level that people do not drain their energy reserves.
- Overtime is not allowed two weeks in a row.

Simple design

- Never have a more complex design than is needed for the current state of the implementation.

- Make design decisions when you have to, not up front.

Design Improvement

A.K.A: Refactoring

- Always try to find ways of improving the design
- Since design is not made up front it needs constant attention in order to not end up with a program looking like a snake pit.
- Strive for minimal, simple, comprehensive code.

Metaphor

- Try to find one or a few metaphors for your program.
- The metaphors should aid in communicating design decisions and intends.
- The most well known software metaphor is the desktop metaphor.

Coding standards

- In order to have a code base that is readable and understandable by everybody the team should use the same coding style.

Small releases

A.K.A: Short cycles

- The software is frequently released and deployed to the customer.
- The time for each release is planned ahead and are never allowed to slip. The functionality delivered with the release can however be changed right up to the end.
- A typical XP project has a new release every 3 months.
- Each release is then divided into 1-2 week iterations.

Extra material

Sprint(Cont...)

What is required ?

- Source control
- Continuous integration
- Unit testing
- Feature testing

Product Backlog Maintenance

Agile at ICST


Certified Agile Tester
Pragmatic, Soft Skills Focused, Industry Supported

Can agile be certified?

Concept

We are well aware that agile team members shy away from standardized trainings and exams as they seem to be opposing the agile philosophy. However, agile projects are no free agents; they need structure and discipline as well as a common language and methods. Since the individuals in a team are the key element of agile projects, they heavily rely on a consensus on their daily work methods to be successful.

All the above was considered during the long and careful process of developing a certification framework that is agile and not static. The exam to certify the tester also had to capture the essential skills for agile cooperation. Hence a whole new approach was developed together with the experienced input of a number of renowned industry partners.

www.agile-tester.org

The Training

All Days: Daily Scrum and Soft Skills Assessment
Day 1: History and Terminology: Agile Manifesto, Principles and Methods
Day 2: Planning and Requirements
Day 3: Testing and Retrospectives
Day 4: Test Driven Development, Test Automation and Non-Functional
Day 5: Practical Assessment and Written Exam

Training Provider

FIND your training provider at www.agile-tester.org
BECOME training provider by contacting cat@isqi.org

Yes!

www.agile-tester.org

The Exam

To become a Certified Agile Tester you have to succeed in three different ways:

/ A social skills assessment on capacity for teamwork
/ An exam, which requires free answering – no multiple choice questions
/ A practical section where your testing skills are put to the test

Exam Provider

International Software Quality Institute www.isqi.org

www.agile-tester.org

Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review

Adnan Causevic, Daniel Sundmark, Sasikumar Punnekkat
Mälardalen University, School of Innovation, Design and Engineering,
Västerås, Sweden
{adnan.causevic, daniel.sundmark, sasikumar.punnekkat}@mdh.se

Abstract — **Test driven development (TDD) is one of the basic practices of agile software development and both academia and practitioners claim that TDD, to a certain extent, improves the quality of the code produced by developers. However, recent results suggest that this practice is not followed to the extent preferred by industry. In order to pinpoint specific obstacles limiting its industrial adoption we have conducted a systematic literature review on empirical studies explicitly focusing on TDD as well as indirectly addressing TDD. Our review has identified seven limiting factors viz., increased development time, insufficient TDD experience/knowledge, lack of upfront design, domain and tool specific issues, lack of developer skill in writing test cases, insufficient adherence to TDD protocol, and legacy code. The results of this study is of special importance to the testing community, since it outlines the direction for further detailed scientific investigations as well as highlights the requirement of guidelines to overcome these limiting factors for successful industrial adoption of TDD.**

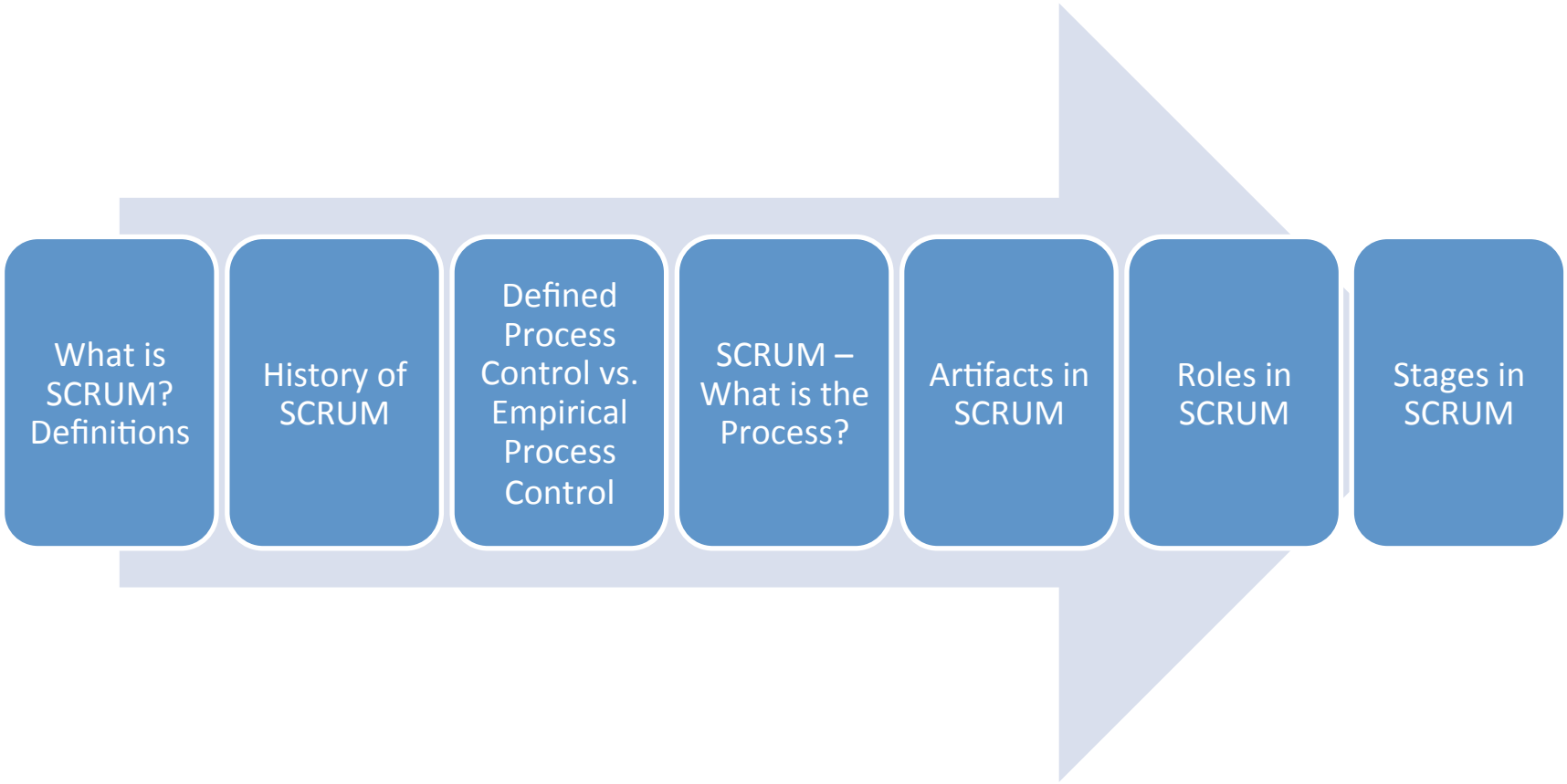
Keywords: Test driven developmen; systematic review; agile software development; unit testing; empirical studies.

in a particular organisation. The specific research question we address in this paper is:

RQ: *Which factors could potentially limit the industrial adoption of TDD?*

In order to identify such limiting factors, a systematic literature review of empirical studies on TDD was undertaken. Partly based on concerns of an insufficient number of studies due to publication bias [3], the review was not restricted to studies reporting on failure to implement TDD. Instead, we decided to expand the scope of the study and to systematically search for primary empirical studies of TDD, including (1) studies where TDD was the main focus, (2) studies where TDD was one of the investigated practices, and (3) studies where TDD was used in the experimental setting while investigating something else. In case any of the studies reported issue(s) with any specific factors, this was noted. By qualitatively and quantitatively analysing the reported issues on TDD within the selected papers, we have identified a number of limiting factors.

Outline (SCRUM part)



Artifacts (Cont…)

Impediments List

- **Definition**: shows the hurdles in day to day SCRUM team work
- **Responsible**: The Team compiles and maintains the list of impediments and how to remove them
- **Properties**:
  - Highly visible, real time picture of the issues faced by the SCRUM teams
  - Impediments brought into light daily
  - The Team itself decides how to solve them

Velocity of SCRUM Team

- Velocity is a measurement of how much the team gets done in an iteration
- Velocity is what actually got done in the last iteration not what is planned
- Can be calculated at the end of each Sprint with the features implemented vs. features planned

Artifacts (Cont...)

Increment of Potentially Shippable Product Functionality

- **Definition**: Scrum requires Teams to build an increment of product functionality every Sprint
- **Responsible**: The Team
- **Properties**:
  - Must be potentially shippable
  - consist of thoroughly tested, well-structured, and well-written code that has been built into an executable and that the user operation of the functionality is documented, either in Help files or in user documentation
  - Only the Team can change it
  - Highly visible, real time picture of the current Sprint

Tricks of the Trade

Consensus on “Done”

- Teach SCRUM team to manage itself
- Understand all aspects of what the team is doing and frequently correlate its activities in order to deliver a completed set of functionality
- To manage itself, a team must have a plan and report against that plan
- Testing is not someone else’s problem, it’s the SCRUM team problem
- Separately delineate testing activities in the Sprint Backlog until the team understands the meaning of the word “Done”