Agile

Morgan Ericsson
<morgan.ericsson@chalmers.se>

This Week

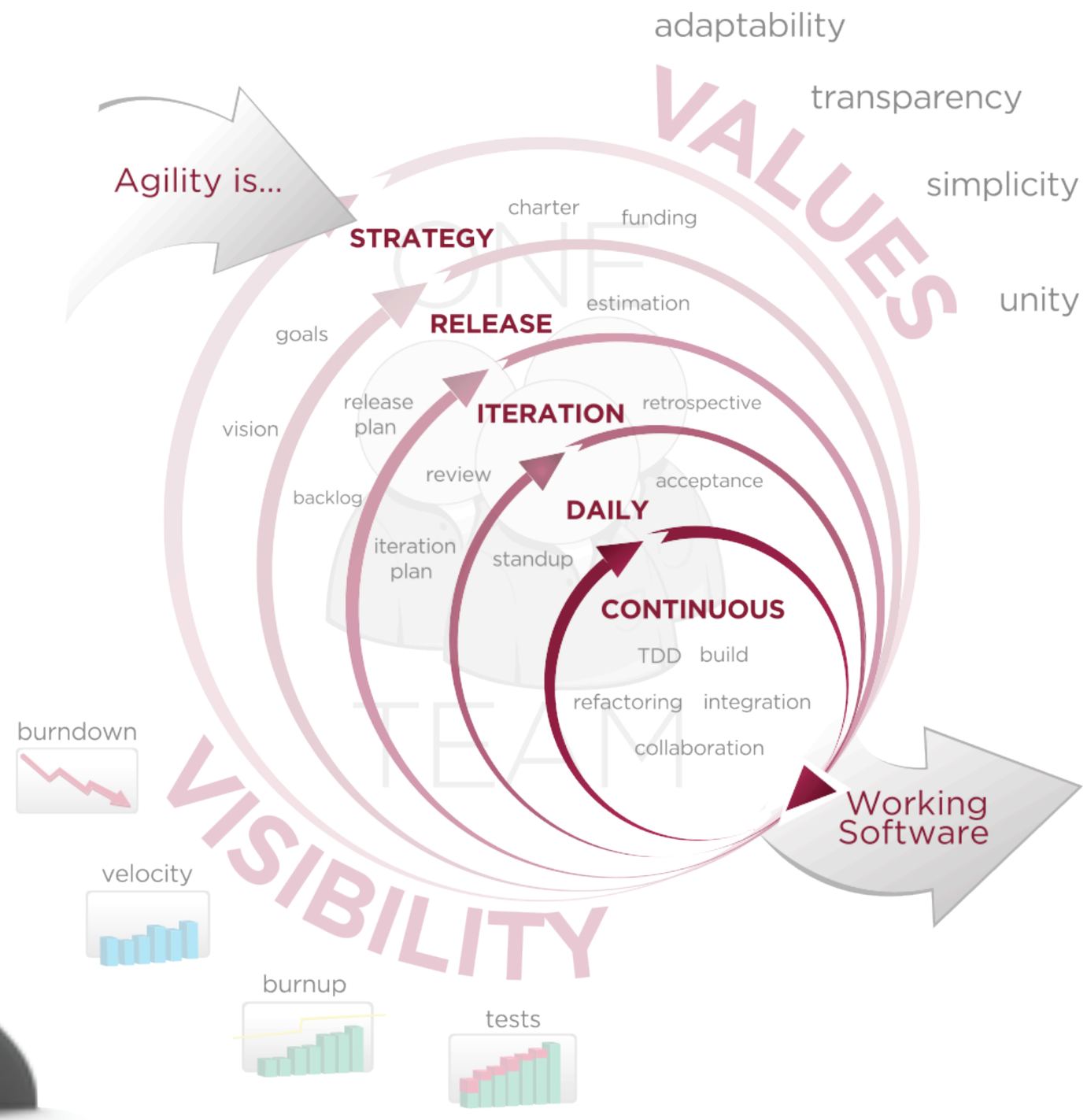
- Today, Agile + XP, and some practice
- Thursday, first meeting with customer
  - ~1h presentation, 1h per 3 groups
- Friday, lecture on Pair Programming and Automated Testing

Next Week

- First sprint!
- Tuesday, follow-up meeting with customer
- Thursday, special event (follow up from this afternoon)
- Friday, first acceptance test!
    - more about that this friday

Production vs. Creation

AGILE DEVELOPMENT



ACCELERATE DELIVERY

Agile Manifesto

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

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

Principles behind the Agile Manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Principles behind the Agile Manifesto

- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Principles behind the Agile Manifesto

- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.

Principles behind the Agile Manifesto

- Simplicity – the art of maximizing the amount of work not done – is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Methodologies & Practices

- Scrum
- eXtreme Programming (XP)
- Kanban

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.

Resing and Janoff (2000)

Roles

Commitment vs. Involvement

The Pigs

- Product owner
  - Represents interests of everyone with a stake in the project and resulting system
  - Responsible for initial/ongoing funding, initoal overall requirements, ROI objectives, release plans

The Pigs

- Scrum master
  - responsible for the Scrum process
  - teaching Scrum to everyone involved in the project
  - implements 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.

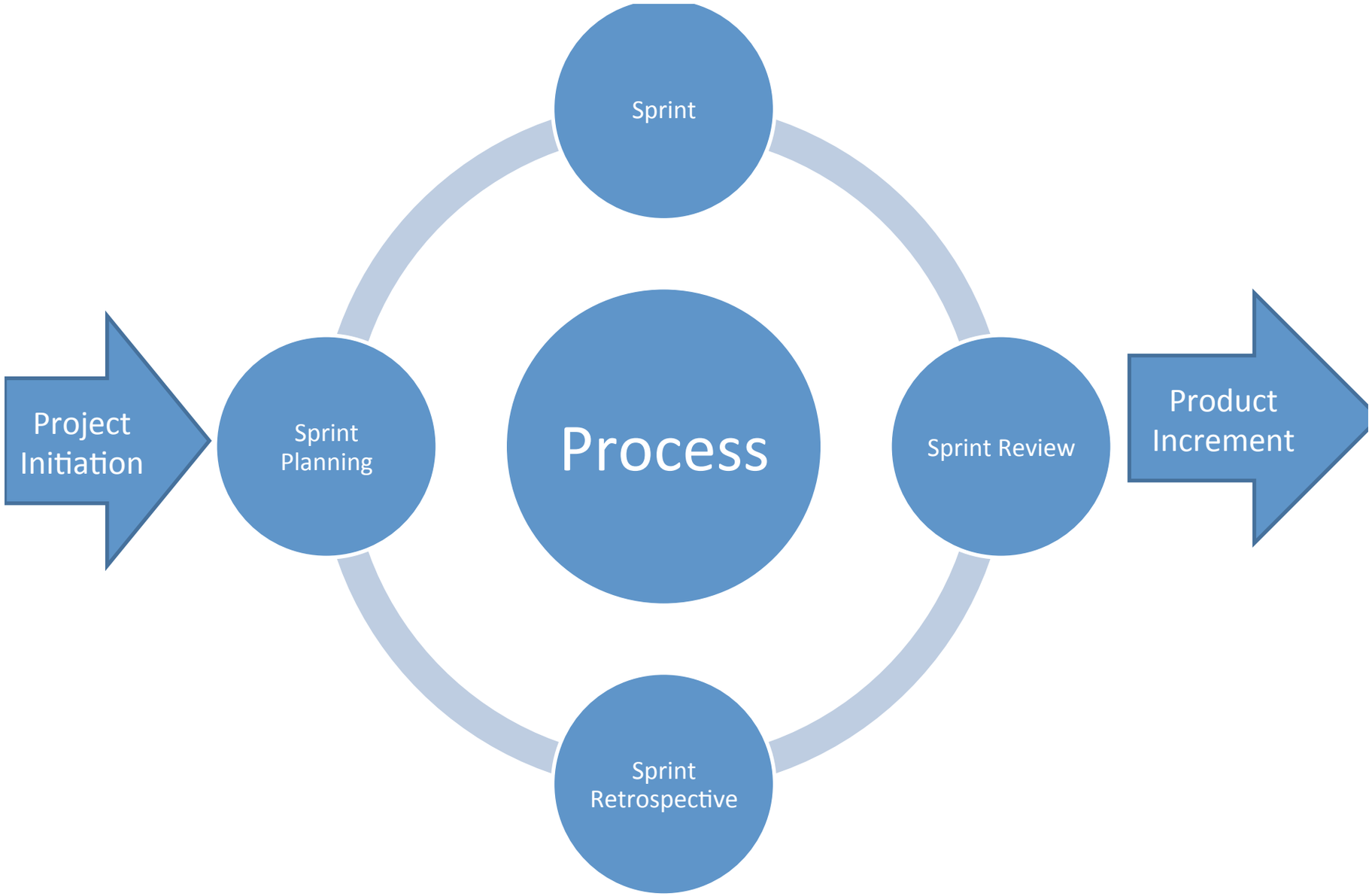
The Pigs

- Scrum team
  - Self-managing, self-organizing, and cross-functional
  - responsible for figuring out how to turn a list of requirements into an increment of functionality

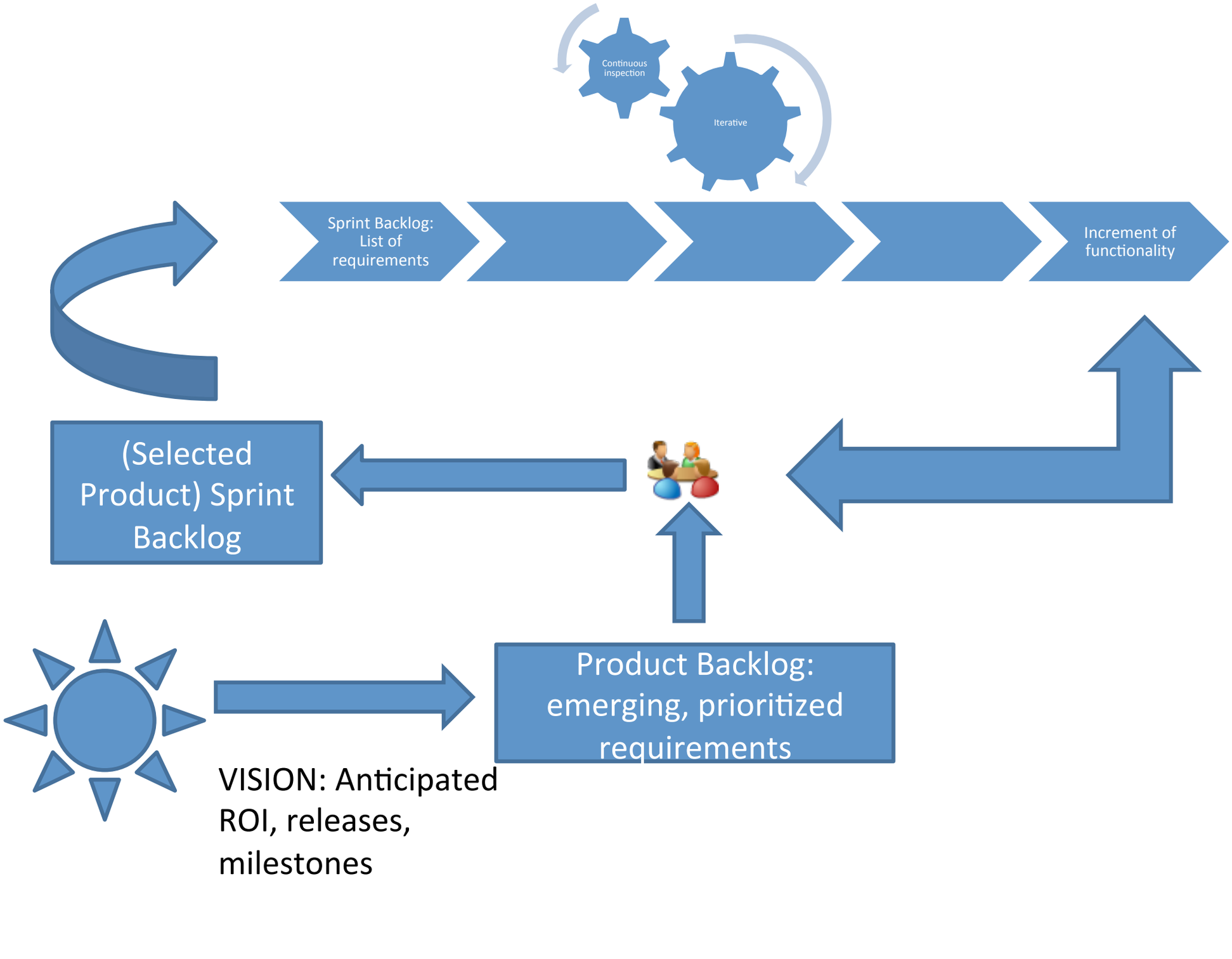
The Chickens

- Stakeholders
- Users

Scrum Process



Scrum Process



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

Product Backlog

- The requirements for the system or product being developed by the project(s) are listed in the Product Backlog
- Product Owner is resonsible for the contents, prioritization, and availability of the Product Backlog

Product Backlog

- 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

Sprint Backlog

- 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
- The Team compiles an initial list of these tasks in the second part of the Sprint planning meeting

Sprint Backlog

- 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 have not yet been appropriately defined
- Only the Team can change it
- Highly visible, real-time picture of the current Sprint

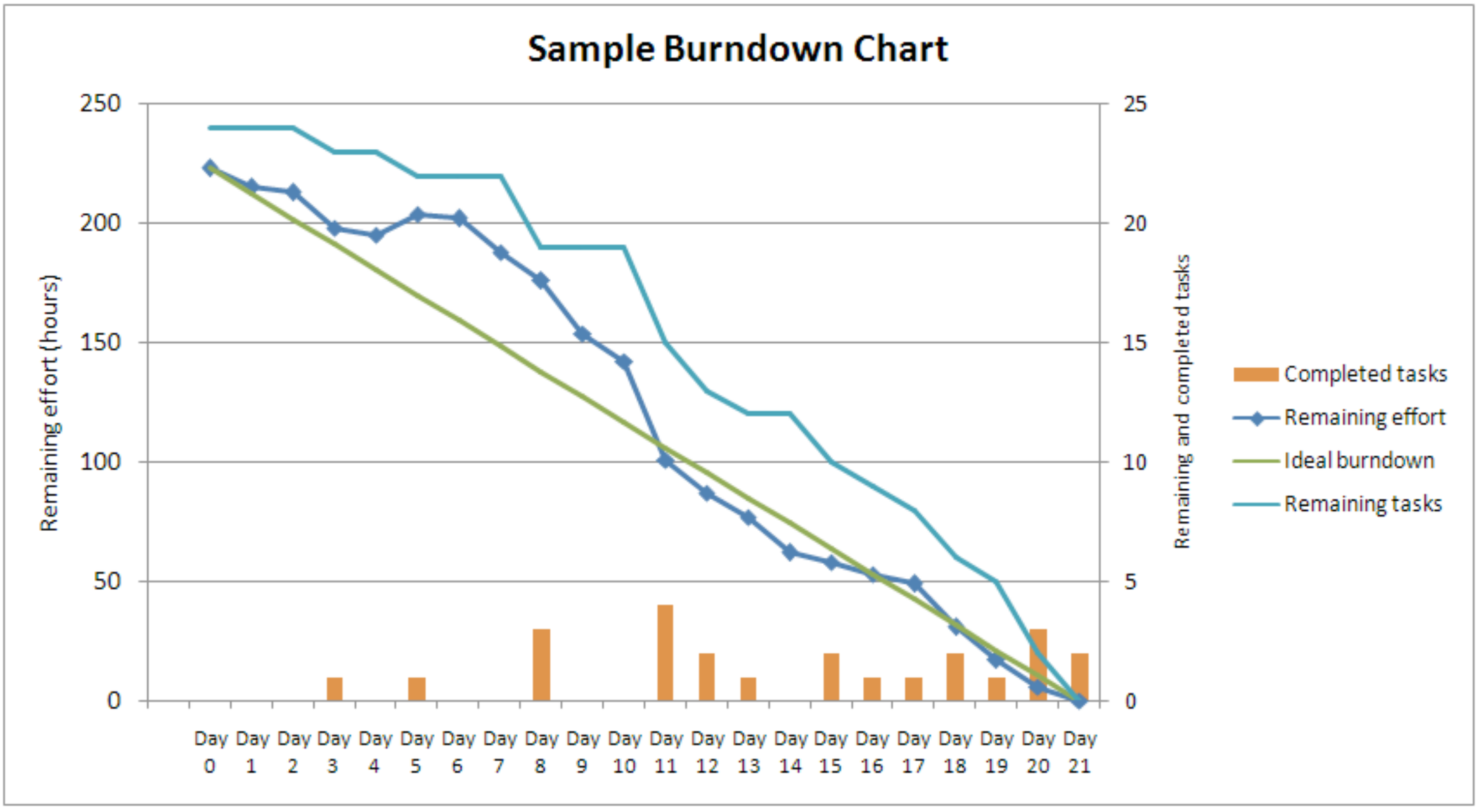
Burndown Chart

- Shows the amount of work remaining across time
- The Team compiles an initial list of these tasks in the second part of the Sprint planning meeting

Burndown Chart

- 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
- Highly visible, real-time picture of the current Sprint

Burndown Chart



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

Project Initiation

- Release Plan
  - based on the product backlog and priori7zed items
- Sprint Team
  - Product owner
  - Scrum master
  - The Team

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, commit to Product Backlog for the next Sprint
  - 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, the Team plans out the Sprint
  - 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

Sprint / Daily Scrum

- 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

- 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

- 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
  - the functionality increment are for inspection,
  - the adaptation is the modified Product Backlog

Scrum Retrospective

- 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

Scrum Advice

- 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.

eXtreme Programming

- 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 and directly useful code
- Requirements, user story, written on index cards
- Estimate development time per story, prio on value
- Dev starts with discussion with expert user

eXtreme Programmign

- 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

XP practices

- Collective Code Ownership
- Coding standards
- Sustainable Pace (40-hour week)
- The Planning Game
- Simple Design
- Design Improvement (Refactoring)
- Metaphor

Whole Team

- 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.

Small Releases

- 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.

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

Test-Driven Development

- 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

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

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
  - skills spreading
  - ...

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

Coding standards

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

Sustainable Pace

- 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

- 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

User Stories

- One or more sentences in the everyday or business language
- Captures what a user does or needs to do as part of his or her job function
- Quick way of handling requirements without formalized requirement documents
- Respond faster to rapidly changing real-world requirements

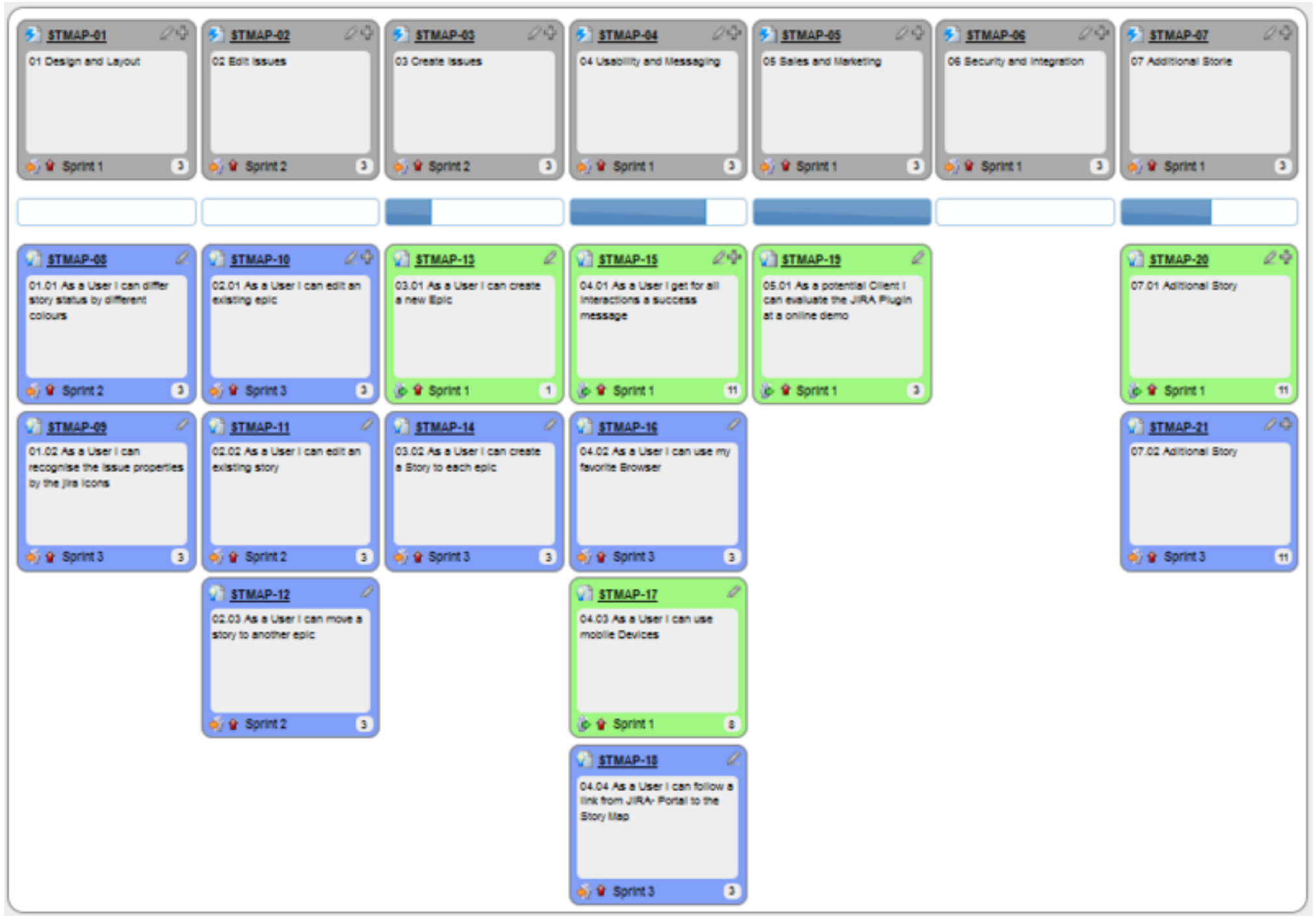
User Stories

- "As a <role>, I want <goal/desire> (so that <benefit>")
- "As <who> <when> <where>, I <what> because <why>."
- *“As a user, I want to search for my customers by their first and last names.”*
- *“As a user closing the application, I want to be prompted to save if I have made any change in my data since the last save.”*

Benefits

- Represent small chunks of business value that can be implemented in a period of days to weeks.
- Needing very little maintenance.
- Allowing projects to be broken into small increments.
- Being suited to projects where the requirements are volatile or poorly understood. Iterations of discovery drive the refinement process.
- Making it easier to estimate development effort.
- Require close customer contact throughout the project so that the most valued parts of the software get implemented.

Story Maps


STMAP-01
01 Design and Layout
Sprint 1
3
STMAP-02
02 Edit Issues
Sprint 2
3
STMAP-03
03 Create Issues
Sprint 2
3
STMAP-04
04 Usability and Messaging
Sprint 1
3
STMAP-05
05 Sales and Marketing
Sprint 1
3
STMAP-06
06 Security and Integration
Sprint 1
3
STMAP-07
07 Additional Storie
Sprint 1
3
STMAP-08
01.01 As a User I can differ story status by different colours
Sprint 2
3
STMAP-10
02.01 As a User I can edit an existing epic
Sprint 3
3
STMAP-13
03.01 As a User I can create a new Epic
Sprint 1
1
STMAP-15
04.01 As a User I get for all interactions a success message
Sprint 1
11
STMAP-19
05.01 As a potential Client I can evaluate the JIRA Plugin at a online demo
Sprint 1
3
STMAP-20
07.01 Aditional Story
Sprint 1
11
STMAP-09
01.02 As a User I can recognise the issue properties by the jira icons
Sprint 3
3
STMAP-11
02.02 As a User I can edit an existing story
Sprint 2
3
STMAP-14
03.02 As a User I can create a Story to each epic
Sprint 3
3
STMAP-16
04.02 As a User I can use my favorite Browser
Sprint 3
3
STMAP-21
07.02 Aditional Story
Sprint 3
11
STMAP-12
02.03 As a User I can move a story to another epic
Sprint 2
3
STMAP-17
04.03 As a User I can use mobile Devices
Sprint 1
8
STMAP-18
04.04 As a User I can follow a link from JIRA- Portal to the Story Map
Sprint 3
3

Until Next Time

- Form teams (not people, but name, schedule, etc...)
- Sit down with Max and/or Erik to discuss what you should do (if you haven't already)
- Write user stories
- Sign up for pivotal tracker and invite Max or Erik to your project
- Set up Github repo anv invite Max or Erik

Next time

- Working with user stories
- Good and bad stories
- Planning
- Follow up