

Agile Boot Camp



Agenda – Agile Overview Section

- Introductions
- What is Agile?
- Why Agile?
- Scrum Overview



Introductions – Start Team Building (a key component of agile)

- Name
- Role on this project
- Agile Experience
- Favorite movie

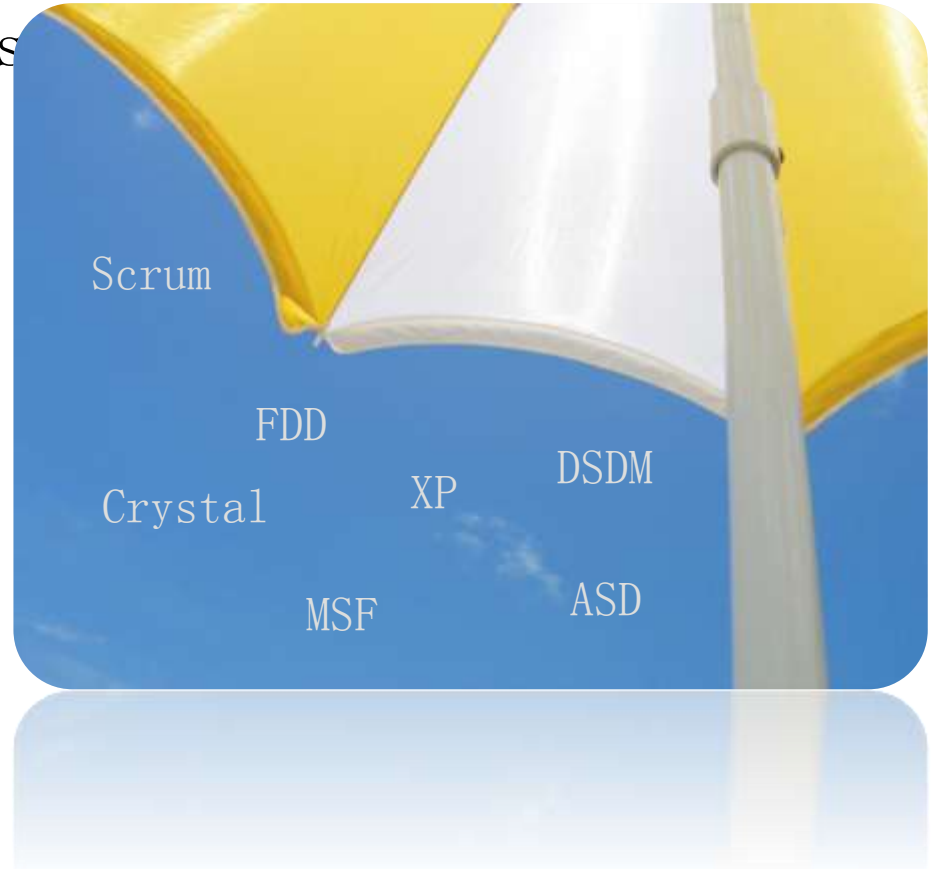


What is Agile?



What is Agile?

- Umbrella term to describe a family of methodologies
- There is no one “Agile method”
- Engineering best practices
- Leadership philosophy
- Project management methodology
- A set of values and principles



Manifesto for agile software development

A statement of values – Created 2001

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 value the items on the left more**

Source: www.agilemanifesto.org



Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Working software is the primary measure of progress.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

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.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

12 principles

Business people and developers must work together daily throughout the project.

Simplicity—the art of maximizing the amount of work not done—is essential.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

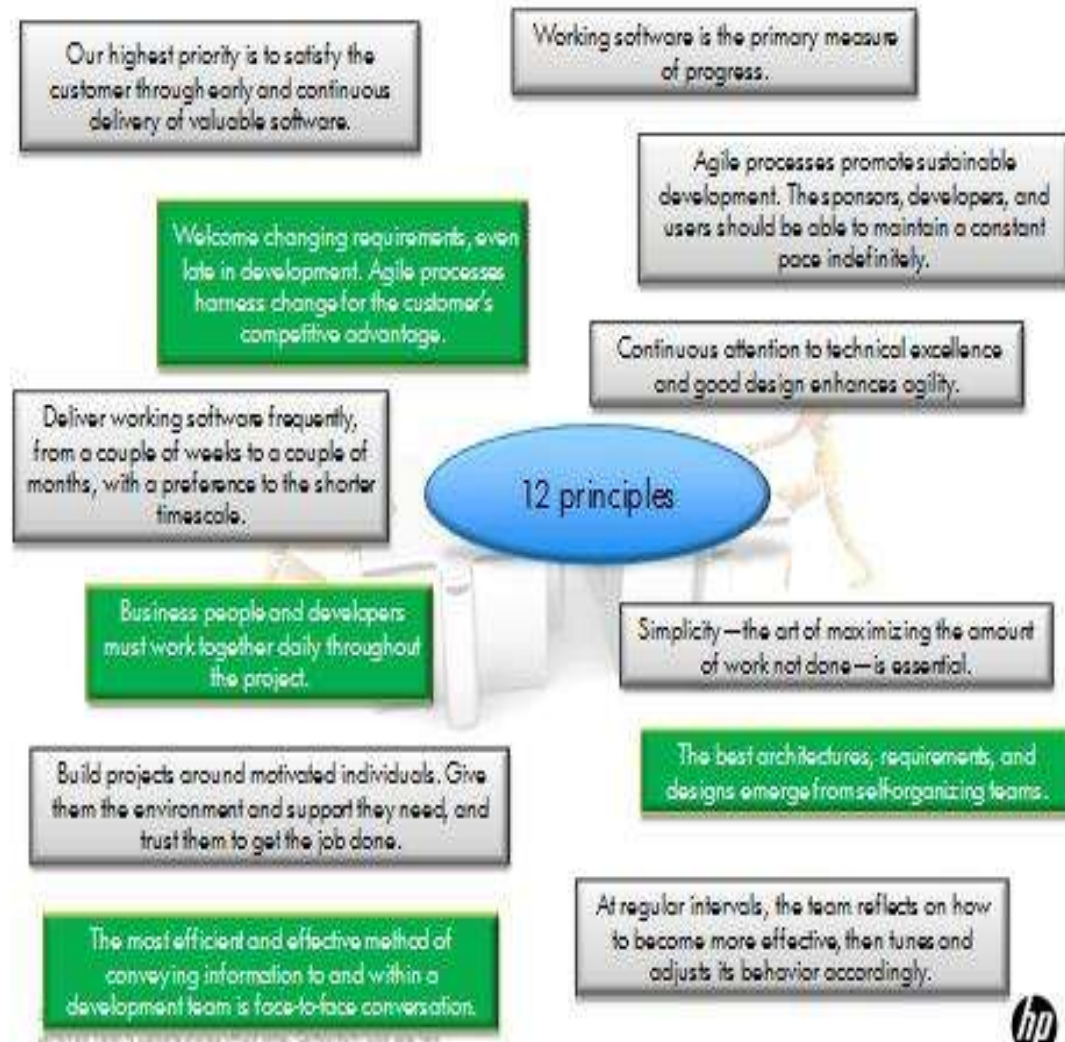
The best architectures, requirements, and designs emerge from self-organizing teams.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Exercise - Agile Principles

1. Take a few minutes to review the attached 12 principles.
2. What **3 words** would you use to summarize the 12 agile principles?
3. What do you think are the 2 most important principles and why?



What is an “Agile” Practice?

A practice, technique, or pattern of thought which enables an individual or team to deliver high-quality, working software rapidly and efficiently towards the purpose of harnessing change for the customer's business advantage



Some Agile Practices

1. Timeboxing / Fixed Sprints / Fixed Iteration Length
2. Release Planning
3. Iteration Planning / Planning Game / Sprint Planning
4. Sprint Backlog
5. Task Board
6. Definition of Done / “Done Done”
7. Daily Stand-up Meeting / Daily Scrum
8. Velocity
9. Sprint Review / Iteration Demo
10. Value Stream Mapping / Maximize ROI
11. Root Cause Analysis / 5 Whys
12. Burn Down Charts / Burn Up Charts
13. Big Visible Charts / Information Radiators
14. Retrospective / Reflection Workshop
15. Agile Documentation
16. Story/Task Decomposition
17. Remove Impediments
18. 5 Levels of Agile Planning
19. Slack
20. Expose Risk Early



Should you use all of these practices?

It is ultimately up to individuals and the team whether they **choose** to employ any specific practice (self-organization)

Whatever you decide...Practice what you preach
If it works - KEEP IT!

If it doesn't - STOP IT!

**“Do, or do not.
There is no
try.”**



Exercise – Methods and Manifesto

Q1. What are some agile development methodologies?

Q2. According to Agile Manifesto Working Software and Responding to change is not valued during agile development? Agree or Disagree?

Q3. What are some of the key practices that characterize an agile way of working?



Why Agile?



Business value of agile

Fast, efficient, quality and value

Faster Time To Market

- 78% – Time to market improved or significantly improved¹
- 37% – Agile Projects are Faster to Market²

Enhance Ability to Manage Changing Priorities

- 82% – Ability to manage change is somewhat or much higher

Increased Productivity

- 80% – Productivity is somewhat or much higher³

Higher Quality

- 73% – Quality had improved or significantly improved¹
- 84% – Defects had gone down by 10% or more¹
- 30% – Defects were down by 25% or more¹

Reduced Cost

- 44% – Development cost was improved or significantly improved^{1,3}



Source: 1. State of Agile Development, VersionOne, 2010

2. Michael Mah, QSMA, 2008

3. Agile Adoption Survey, Dr Dobb's Journal, Scott

Ambler, 2008

Satisfaction impacts of agile

Improved Stakeholder Satisfaction

- 31% – Stakeholder satisfaction was much higher¹
- 47% – Stakeholder satisfaction was somewhat higher¹

Improved Employee Engagement and Job Satisfaction

- 74% – Reported employee morale was improved or significantly improved²



Source: 1. Agile Adoption Survey, Dr Dobb's Journal, Scott Ambler, 2008

2. 3rd Annual State of Agile Adoption Survey, VersionOne,

2008
Copyright 2011 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. Confidentiality label goes here



Shopping Cart - Video

: <http://www.youtube.com/watch?v=M66ZU2PClCM>

While you are watching this video which of the agile principles can you spot the team using?



Introduction to Scrum



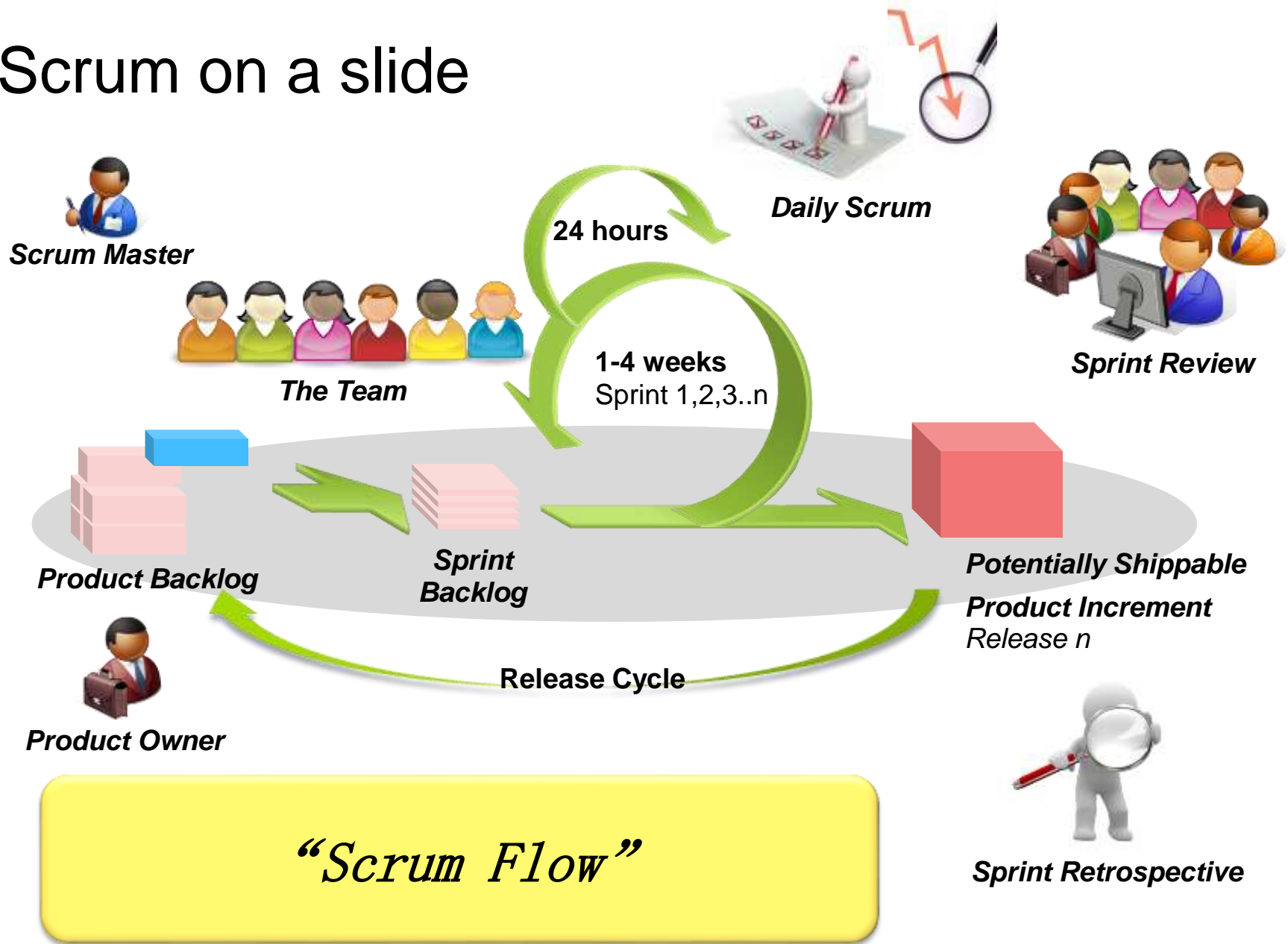
Scrum in 100 Words

-
- Scrum is an agile process that allows us to focus on delivering the highest business value in the shortest time.
 - It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
 - The business sets the priorities. Teams self-organize to determine the best way to deliver the highest priority features.
 - Every week to a month anyone can see real working software and decide to release it as is or continue to enhance it for another sprint.
-

Source: Mike Cohn



Scrum on a slide



Scrum in 10 mins Video

<http://www.axosoft.com/ontime/videos/scrum/>



Exercise – Scrum Ceremonies

Q6. What are the Four main Scrum ceremonies?

- What do you think about a team skipping a ceremony?
- What if all key members aren' t present?

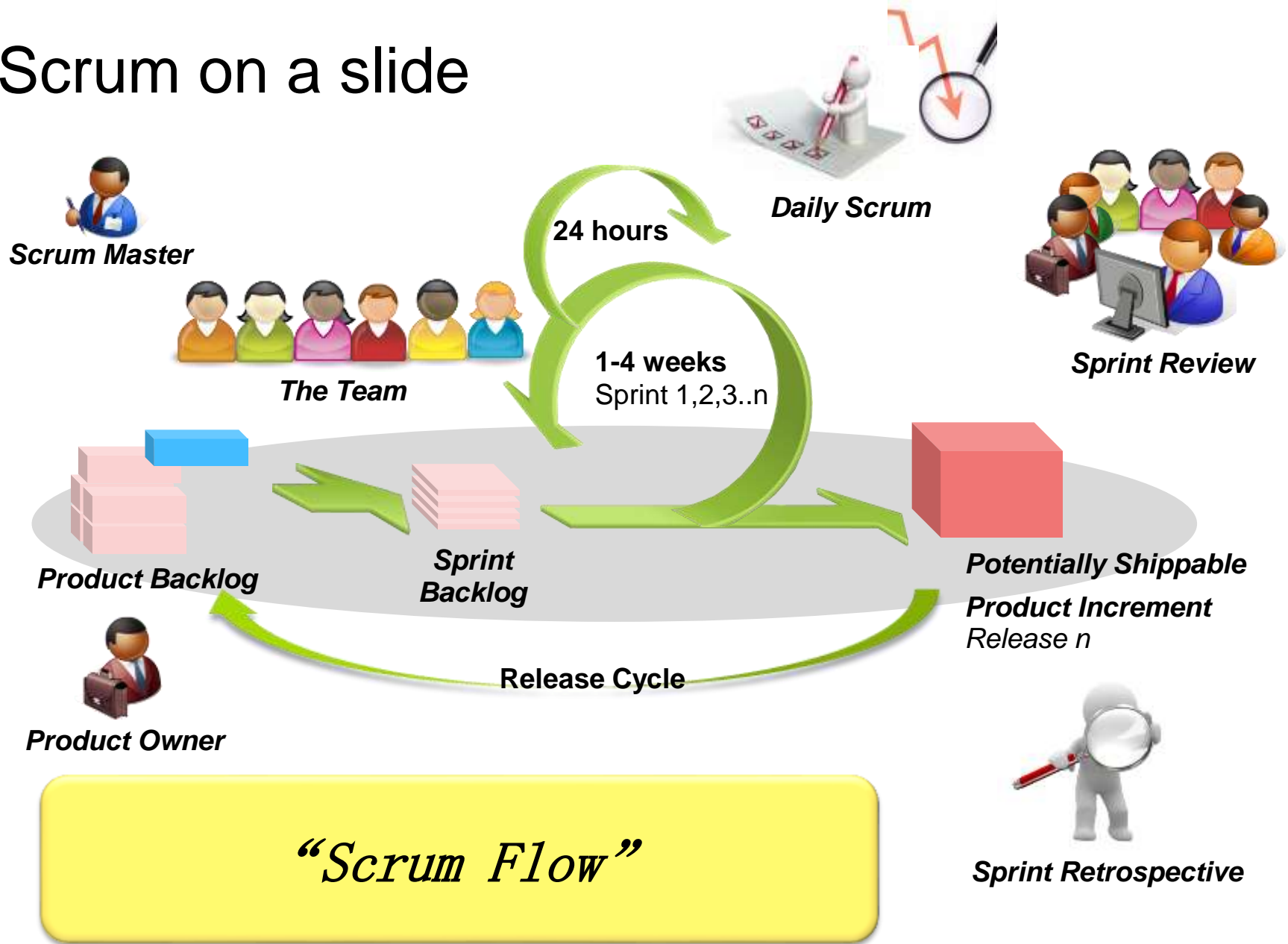
Q7. Scrum recommended sprint durations are 2-4 weeks and team size from 5 to 9 team members - Why?



Agile Team Roles



Scrum on a slide



Chickens and Pigs

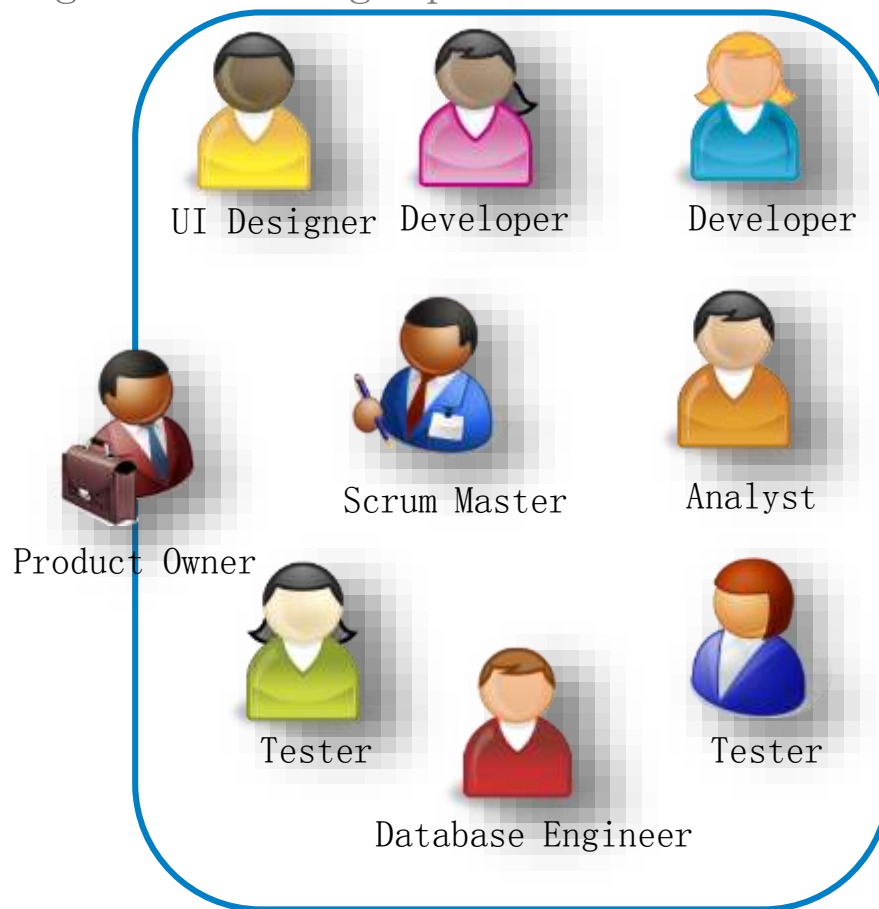


By Clark & Vizados

© 2006 implementingscrum.com

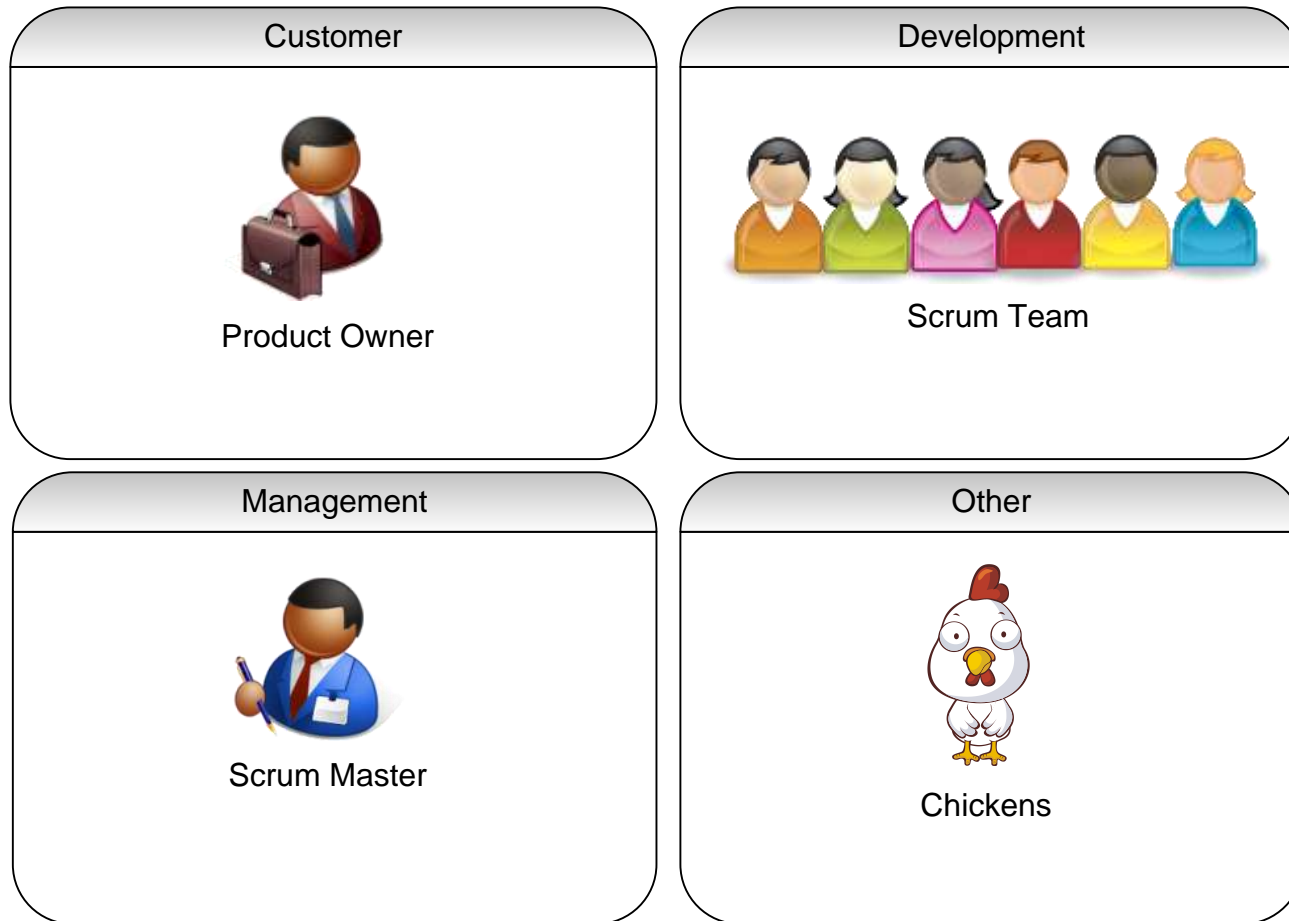
Cross-functional teams

Team members are generalizing specialists



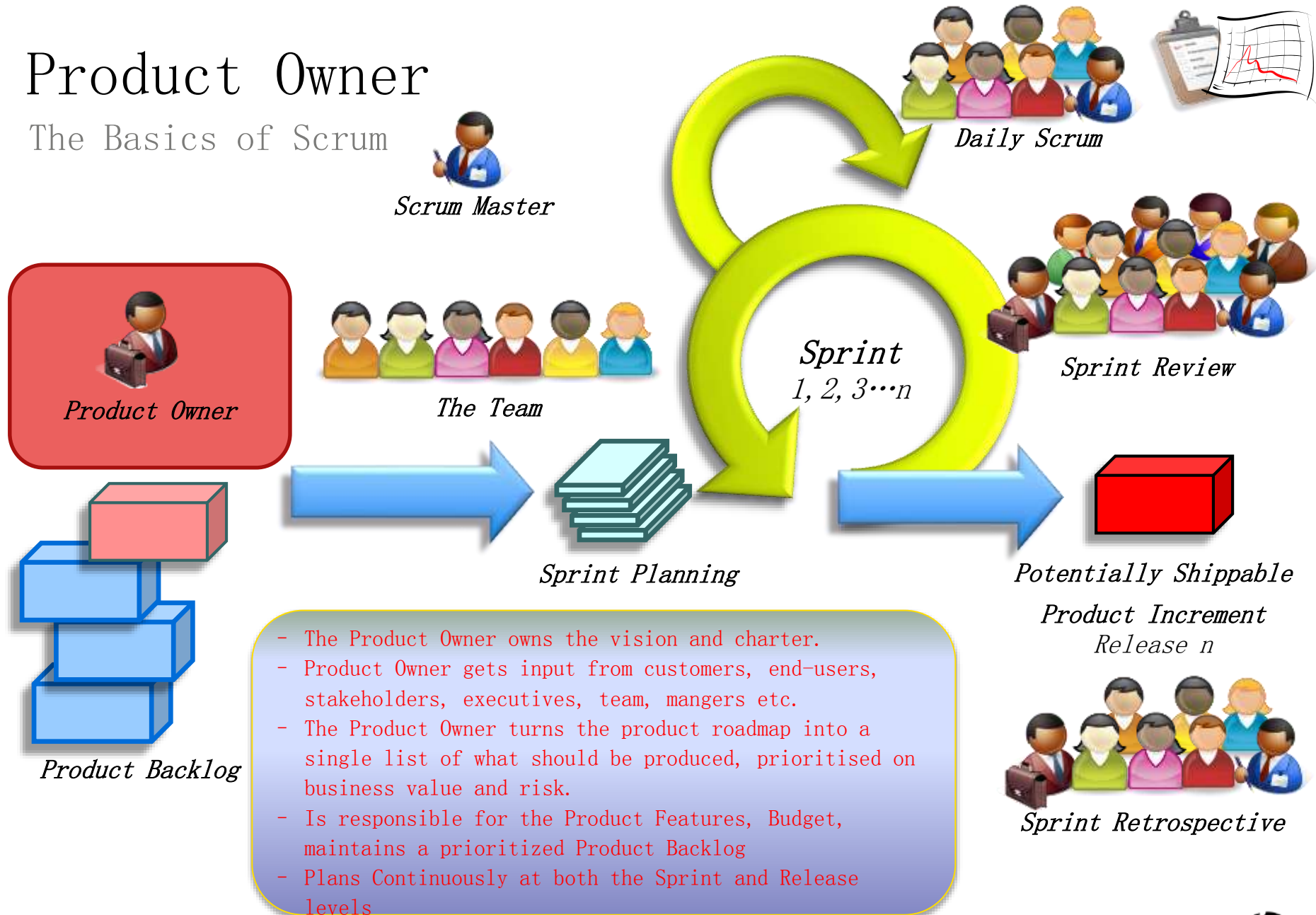
Team members are generalizing specialists - each have their primary skill such as Developer and Tester, but also have other skills. Team members may help in several areas to complete the feature.

Three Scrum Roles



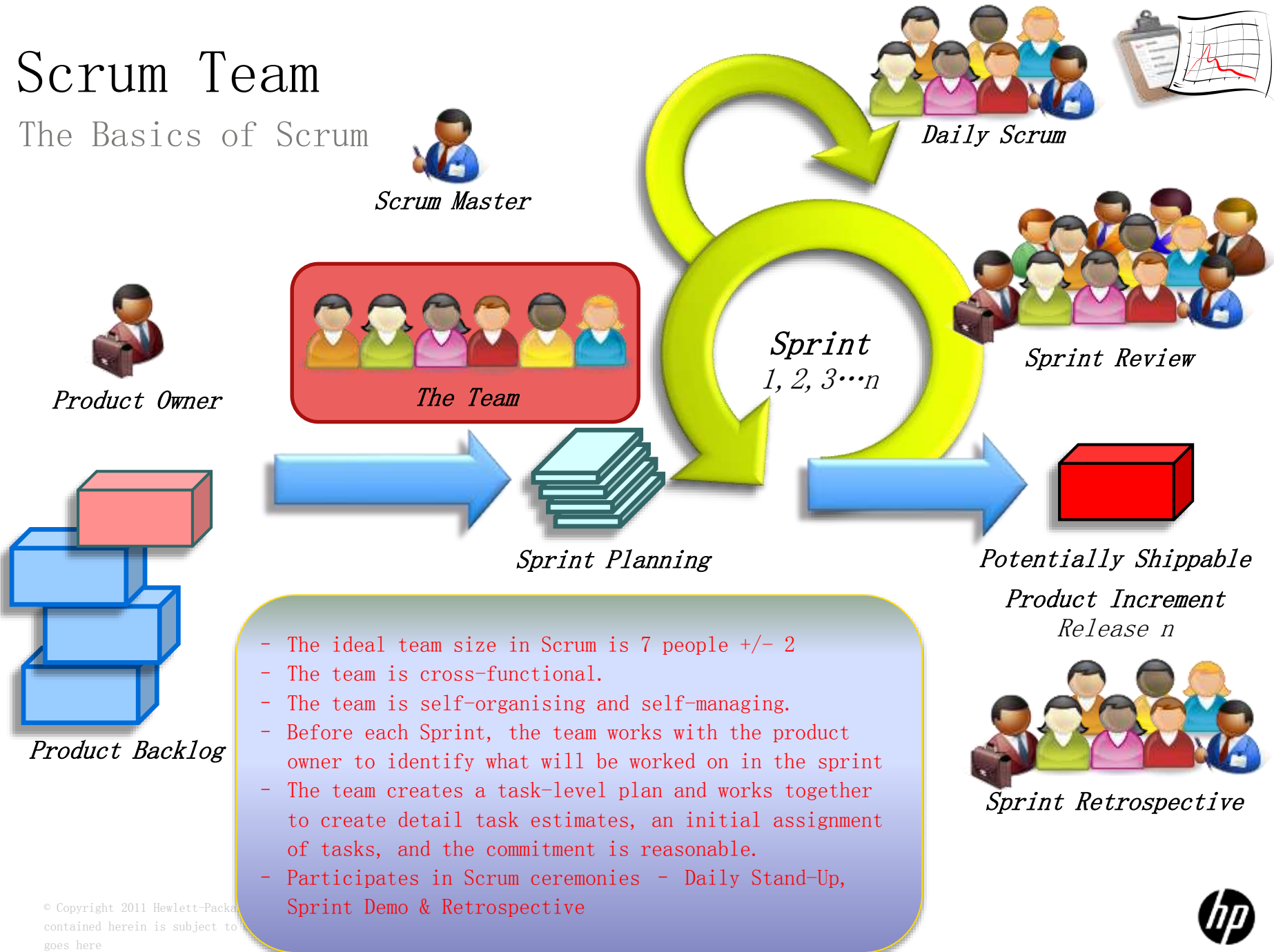
Product Owner

The Basics of Scrum



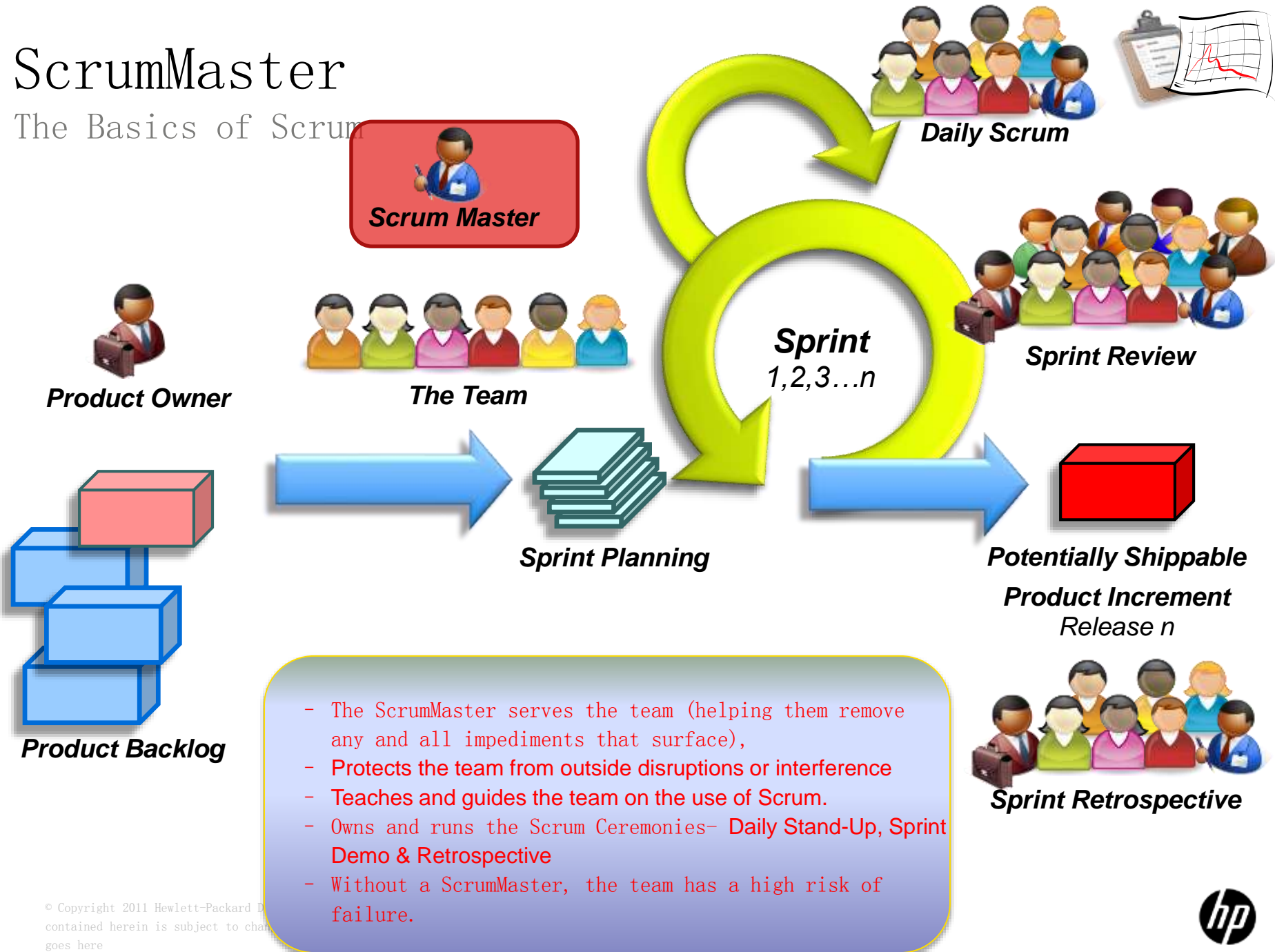
Scrum Team

The Basics of Scrum



ScrumMaster

The Basics of Scrum



Questions ??



Agenda – Scrum Ceremonies

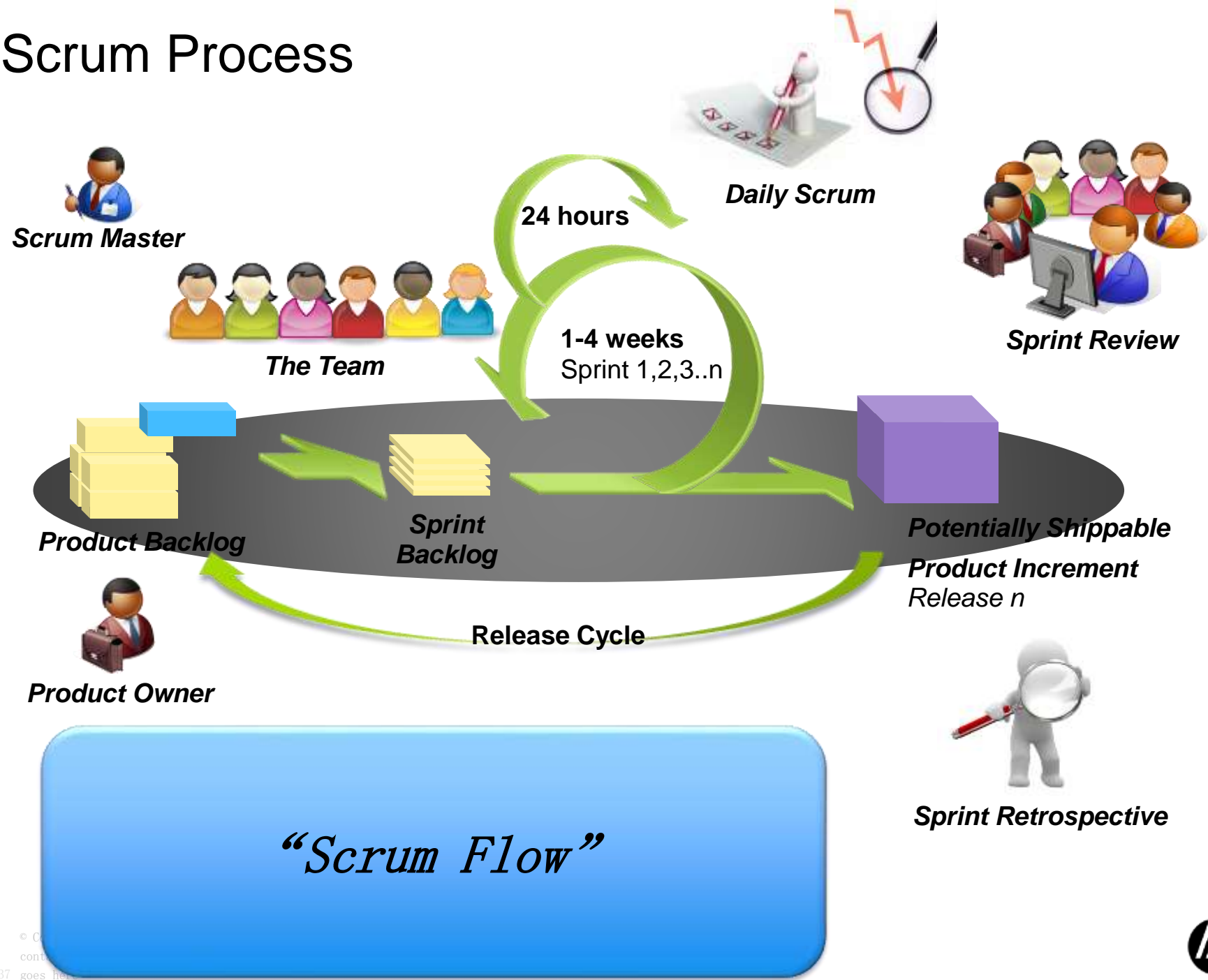
- Scrum Ceremonies
 - Scrum Planning
 - Sprint Planning
 - Daily Stand Up
 - Sprint Demo
 - Sprint Retrospective
- Typical Agile Metrics – BurnDown, BurnUp, Velocity
- Questions



Scrum Planning

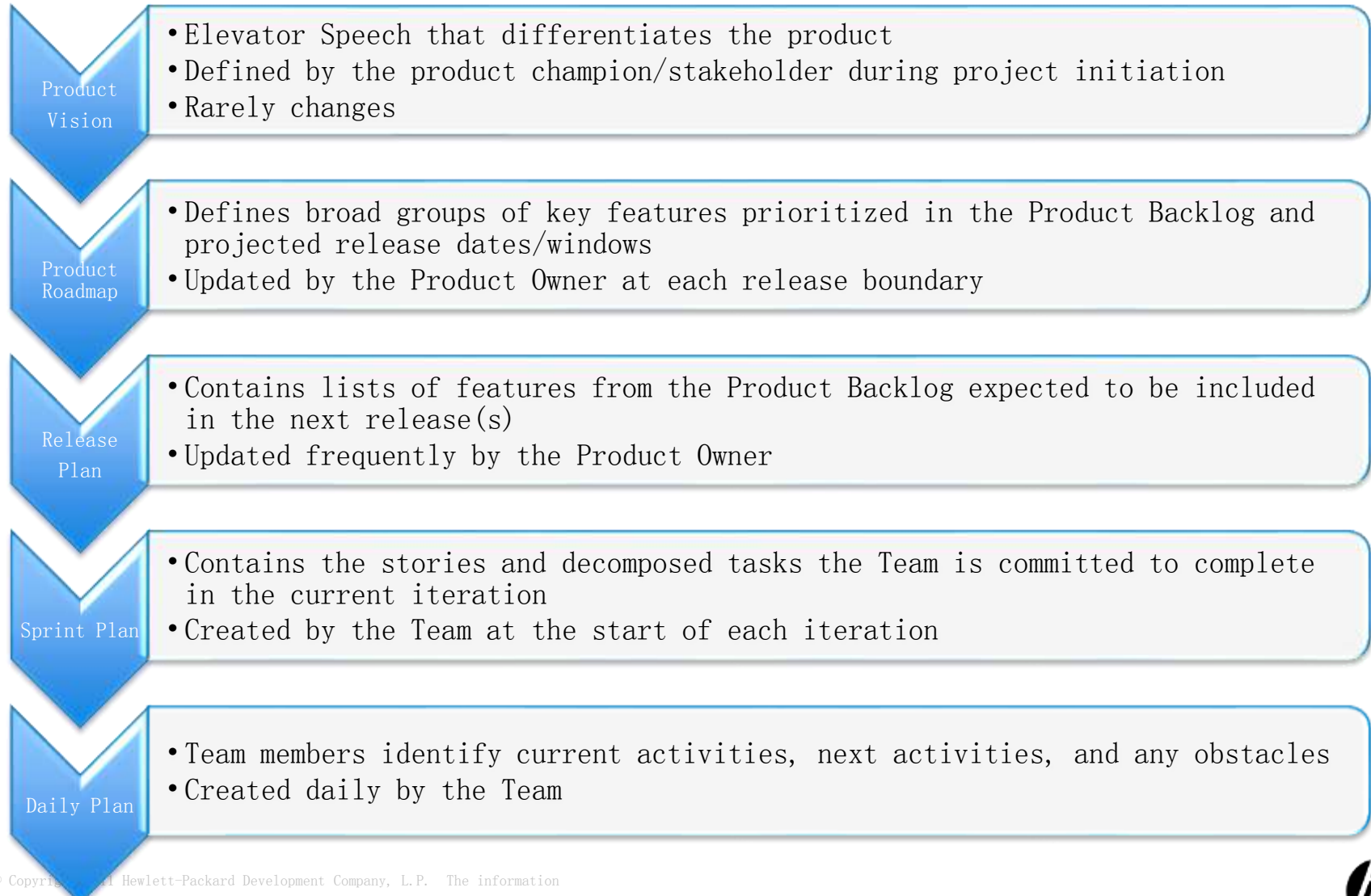


Scrum Process

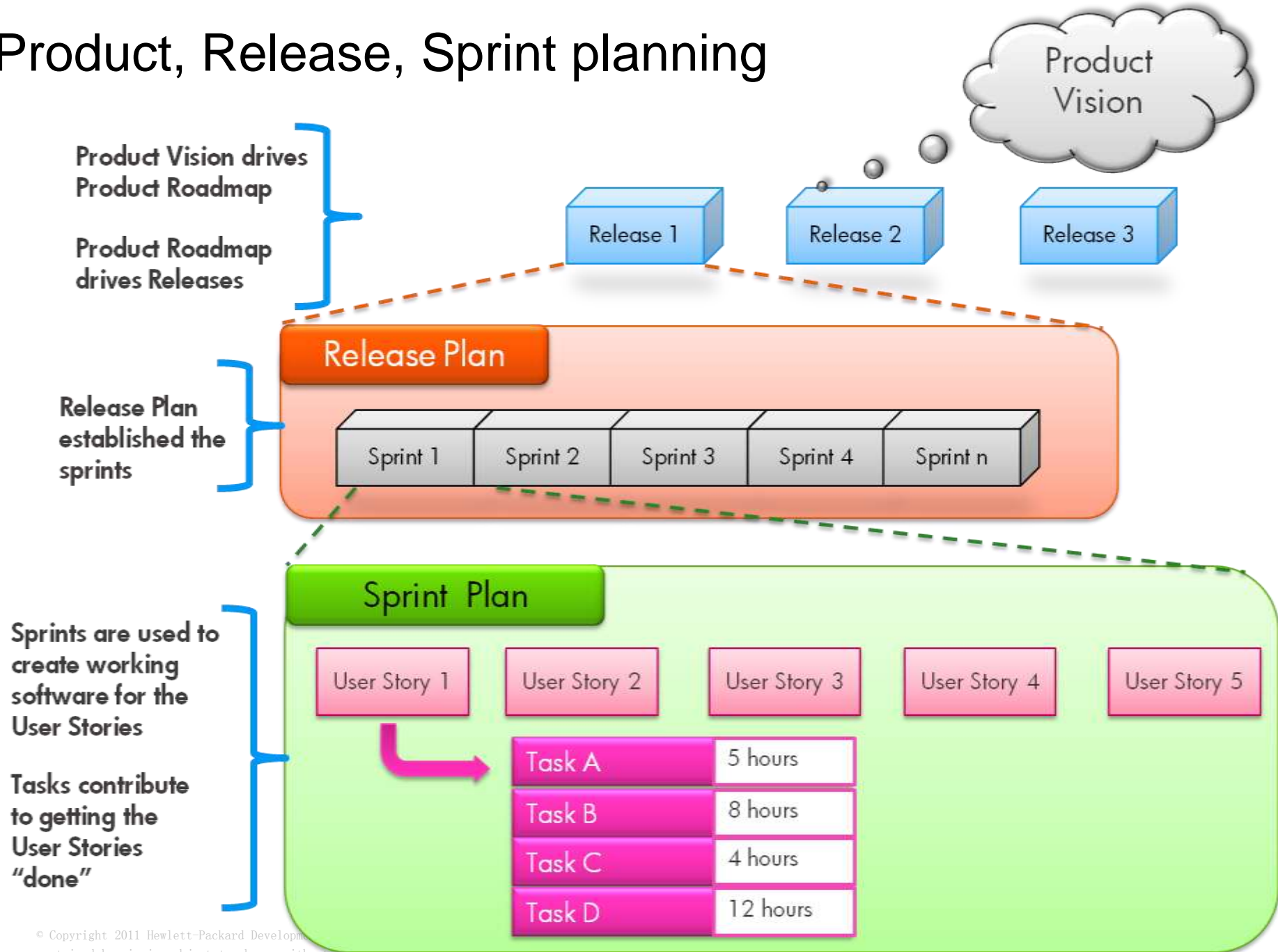


Multi-Level Planning

Continuous just-in-time planning



Product, Release, Sprint planning



Product Roadmap vs.. Release vs.. Sprint Plan

NOTE: The various levels of planning differ in purpose and characteristics. The table reflects some of these distinctions. All three are contained in a single physical Backlog.

	Product Roadmap	Release Plan	Sprint Plan
Planning Horizon	6-12 months (multiple releases)	2-8 months (multiple sprints)	1-4 weeks
Items in Plan	Themes, Large Features (epics) & User Stories	Epics & User Stories	Stories & Tasks
Estimated in	Story points	Story points	Story Points/Hours
To Answer	The high-level schedule and large features of the next few product releases. It supports rough cost and schedule estimates, but not a fixed commitment.	How much might be delivered by a fixed date. When will a set of features likely be ready to release.	How much and what work needs to be done in a Sprint to deliver user stories
Who	Product Owner with product council (business), Executive Management	ScrumMaster, Scrum Team & Product Owner	ScrumMaster, Scrum Team, Product Owner



Exercise – Product Backlog

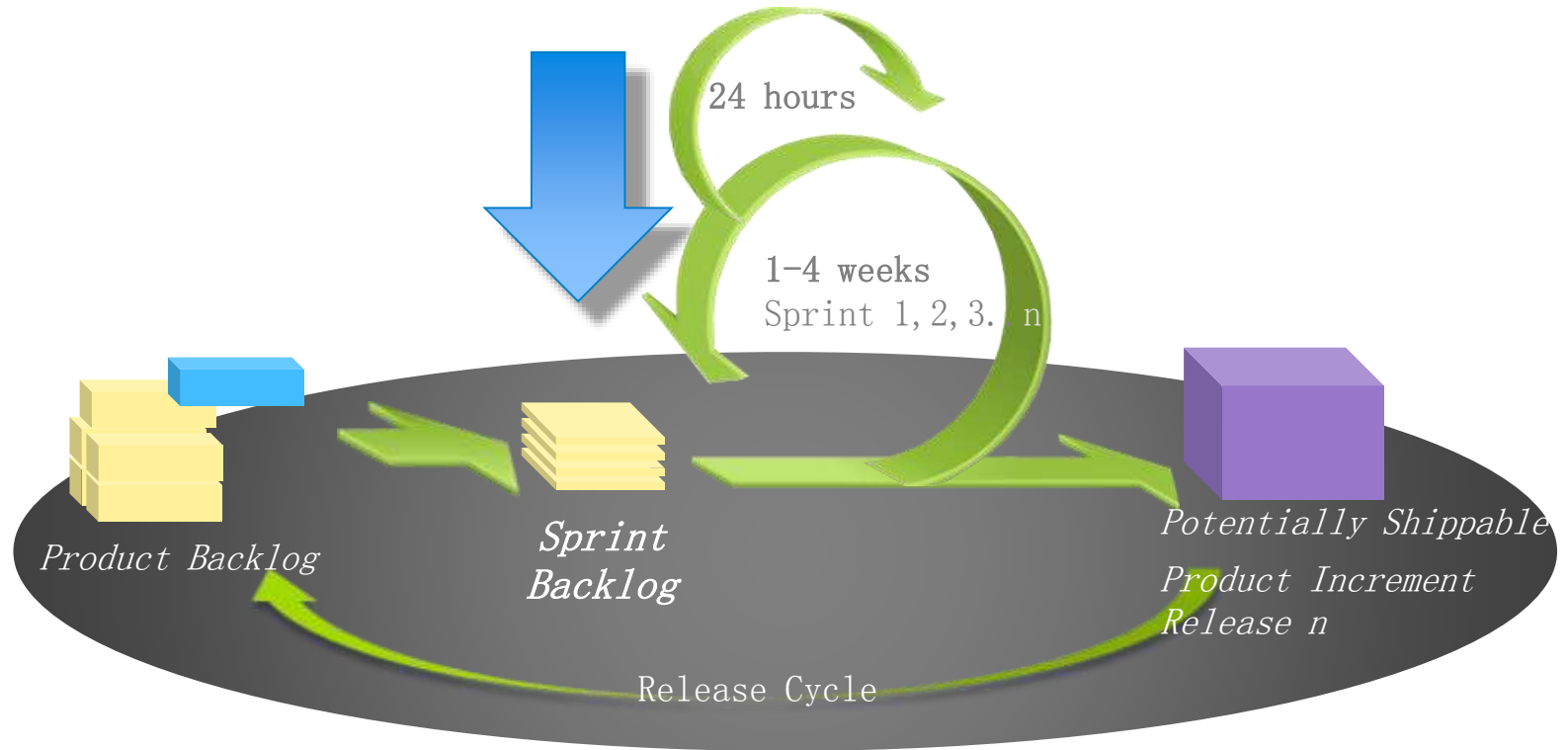
Scenario

For your project: GSIP

Review Product Vision and Product Roadmap



Sprint Planning



Sprint planning (workshop) Part 1

½ day (2 week sprint), full day (4 week sprint)

Sprint Planning Part1: Defines **what to build** in next

~~Sprint~~
Goal: Prioritized agreed upon sprint backlog

- Product Owner and Scrum Team have 2 way discussion where the:
 - Product Owner explains the top priority stories to the team, teams asks questions, enumerates dependencies on technology or other stories
 - Team sets velocity for the sprint (takes into account holidays, planned vacations, training etc.)
 - Team selects as many stories from the backlog as it believes can be delivered during the next sprint based upon projected velocity
- Team commits to the Product Owner that they will deliver the agreed upon work



Sprint planning (workshop) Part 2

Sprint Planning 2: Defines **how to build** the functionality into a product increment

Goal: Completely tasked out, estimated in hours and committed to sprint

- Team tasks out the Sprint Backlog: list all tasks to be performed to get the story to “Done Done”
- Team members sign-up for the stories they want to work on
- The team members assigned to a story estimate the hours they need to complete their tasks
- Team members verify that once tasked these stories can be completed in the sprint
- Team members collectively commit to the sprint deliverables



No changes!

- No changes are allowed during a Sprint **except** by the team
 - Priorities can only change between Sprints
 - Team makeup should only change between Sprints
- Sprint Cancellation
 - Return stories to the backlog is first choice if it doesn't jeopardize the entire sprint's goals
 - Team can cancel Sprint if they feel they are unable to meet Sprint Goal
 - Product Owner can cancel Sprint if external circumstances negate the value of the Sprint Goal
- Cancel sprint only as last resort!



Sprint backlog

A “to-do” list of tasks the team expects to do during sprint

Sourced from a prioritized product backlog

As a guideline, tasks should be a maximum of 2 days worth of work

Time tracking is not part of Scrum

Teams are measured by meeting goals, not how many hours required to meet the goal

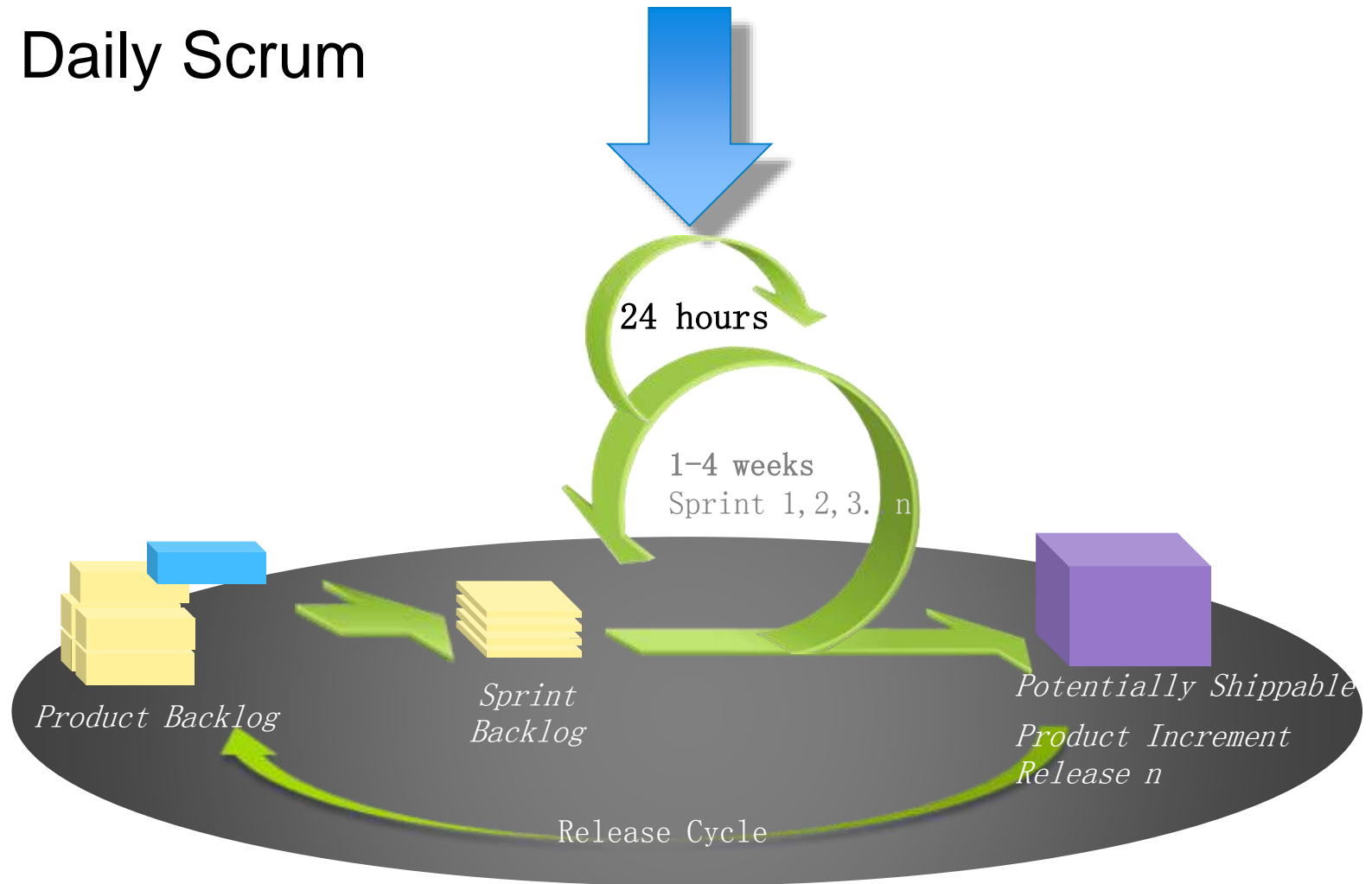
Remaining
tracking

				Hours of Effort Remaining	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
ITEM DESCRIPTION	RESPONSIBLE	STATUS	ESTIMATE	167.0	167	163	158	157	155	155	148	148	148	148
As an editor I want the system to automatically populate fields (acres, district, compartment etc.) so that I do not have to.			15.0											
Do spatial overlays	Jack J.	Completed	2.0	2	1	1	0	0	0	0	0	0	0	0
Transfer attributes	Mike C.	In Progress	6.0	6	4	2	2	2	2	2	2	2	2	2
Get target feature class	Mike C.	Completed	3.0	3	3	3	3	2	2	0	0	0	0	0
Unit test for utility classes	Bill R.	Not Started	3.0	3	3	3	3	3	3	3	3	3	3	3
Document attribute population design in Wiki	Bill R.	Not Started	1.0	1	1	1	1	1	1	1	1	1	1	1
As an editor I want contextual tools to fix GDB topology errors so that I don't have to look for them in menus, etc.			2.0											
Modify & register delete feature command w/all topology error commands	Nancy H.	In Progress	2.0	2	2	1	1	1	1	1	1	1	1	1
As an editor I want the have bad edits erased as they are done so that I do not have to deal with complex interfaces or logic to fix the problem			48.0											
Write clip utility against other features in same feature class	Nancy H.	In Progress	6.0	6	6	4	5	5	5	3	3	3	3	3
Write split utility against another feature class	Jill P.	In Progress	6.0	6	6	6	6	5	5	2	2	2	2	2
Put clip utilities into class extensions	Bill R.	Not Started	6.0	6	6	6	6	6	6	6	6	6	6	6
Put split utilities into class extensions	Bill R.	Not Started	6.0	6	6	6	6	6	6	6	6	6	6	6
Check: Doesn't intersect features in current layer	Jack J.	Not Started	6.0	6	6	6	6	6	6	6	6	6	6	6

or



The Daily Scrum



Daily Scrum (alternately called Stand-Up)

- All team members participate!
- Organized and run by the ScrumMaster
- Daily Scrum should be no more 15 minutes
- Held same place, same time, every working day
- Participants must be on time and prepared (backlog and/or story board)
- Anybody can come, but only the team and Scrum Master speak during the Scrum
- 3 questions asked to every team member:

- ✓ *What have you done since our last meeting?*
- ✓ *What will you do until our next meeting?*
- ✓ *What obstacles (blocks, issues) are getting in your way?*



Important: Not a place for discussions on how to solve the issues. Setup additional meetings following Daily Scrums to discuss resolutions to issues.

Sample Task board

Recommended: all collocated teams have a board even if they are using an automated tool



Another Sample Task board



Scrum Meeting In Action

Typical Scrum

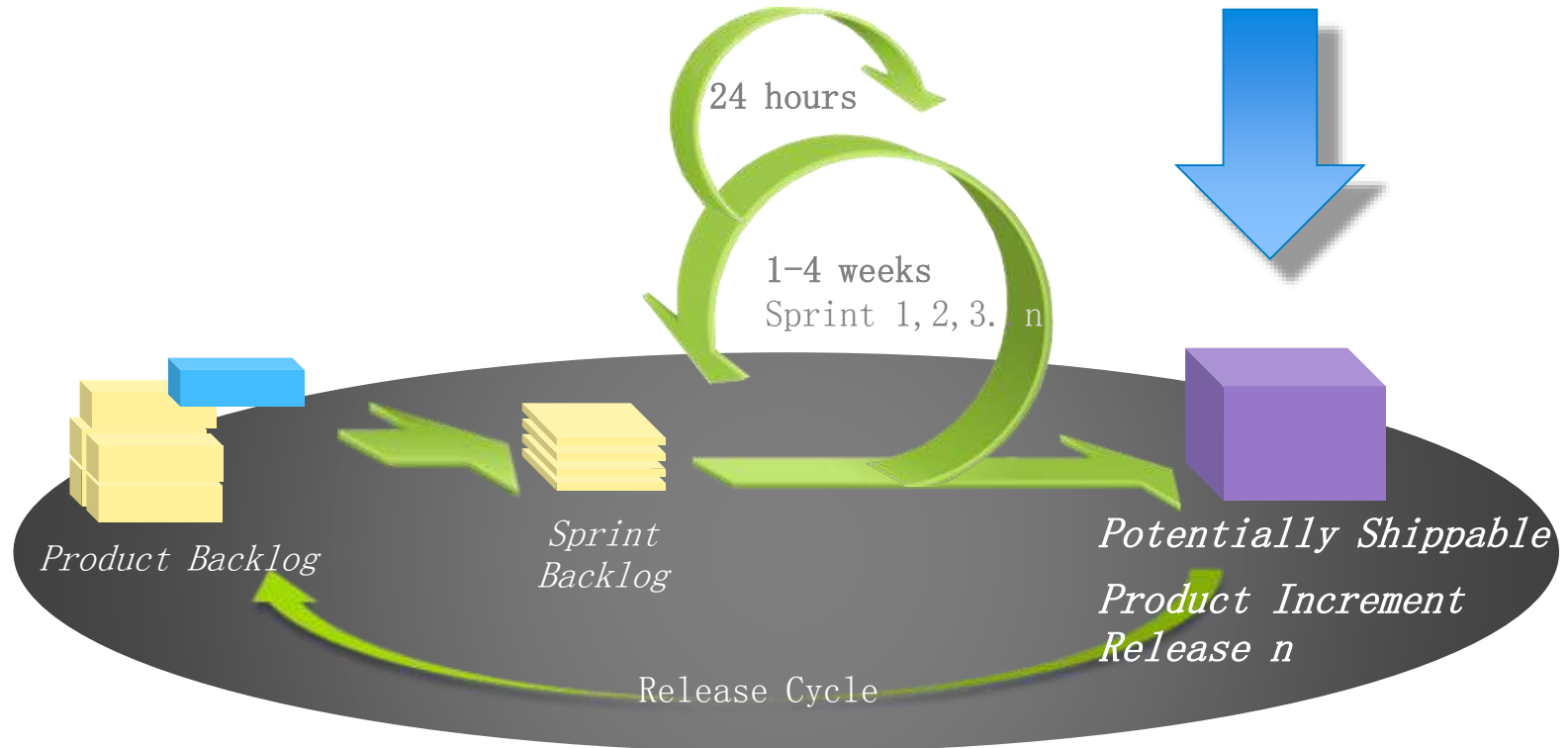
<http://www.youtube.com/watch?v=tyN-3lrftGY&feature=related>

Dysfunctional Scrum

<http://www.youtube.com/watch?v=B3htbxIkzzM>



The Sprint Review (Demo) & Retrospective



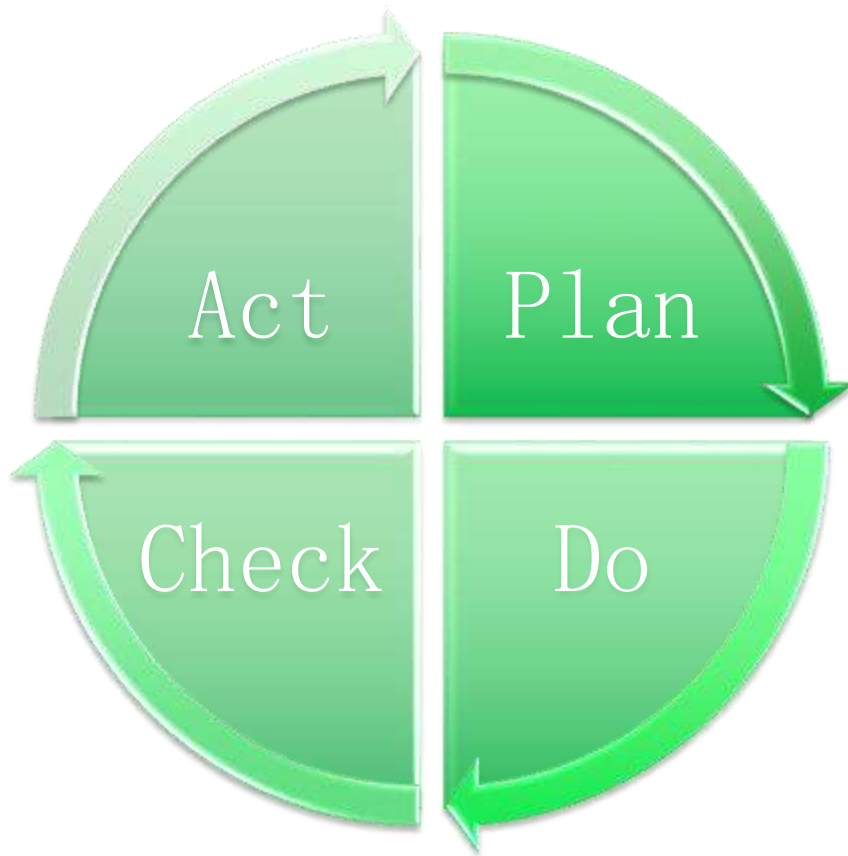
Sprint Review (aka Demo)

- Team presents work done in the sprint
- Demonstration of potentially shippable increment of product functionality
- Presented by the team to the Product Owner and customers/users
- What have we achieved?
- Should show finished functionality
- What is missing?
- Maximum of 2 hours for demonstration
- Minimum of formality
 - No PowerPoint presentations, Preference for working software



Retrospectives

Are part of the quality improvement cycle



Plan

- Release and Sprint Planning

Do

- Team and customer collaborate to elaborate on requirements, do some design, some coding, some testing in one Sprint to deliver a product increment

Check

- Daily stand-up and end of Sprint Review and Retrospective. Inspect how the team went.

Act

- Adapt the way the team works based on what was learned from the Retrospective

Sprint Retrospectives

Regular reflection and adaption

- the most important element of Agile practices!

- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly
- Periodically, the entire team including owners/end users
 - reflects on the results of the process
 - learn from that examination
 - adapt the process and organization to produce better results
- The team decides what is working well, what isn' t, and what one thing to do differently next time
- The shorter the iterations and releases - the faster the learning



Sprint Retrospective

Characteristics:

- Held at the end of every Sprint
- Facilitated by Scrum Master (or other neutral party)
- Team reflects on the sprint
- What went well?
- What could be improved for next sprint?
- What will we do differently for the next sprint?
- What actions (with owners) are we taking in the next sprint?
- Maximum of 2 hours



Sample Retrospective

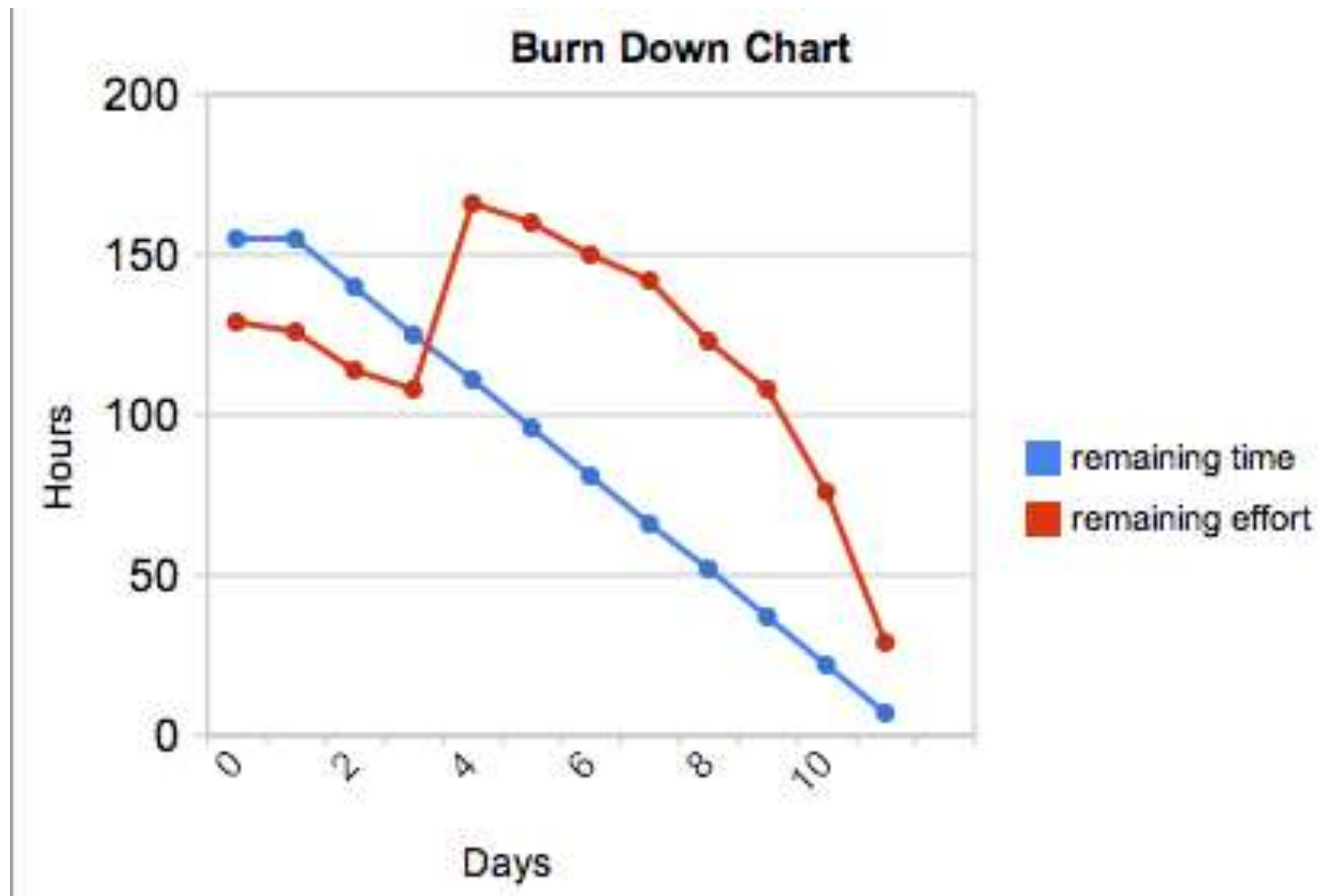
What went well?	What didn't go well?	Where does the team want to concentrate improvement?	What action is the team taking in the next sprint
<ul style="list-style-type: none"> •BA trip to Dallas, good discussions •Better interaction with development and test •Good clarity of data model to be used •End result of refactoring resulted in a better product •BA being present and being able to review test results in person •RESTful services integration 	<ul style="list-style-type: none"> •Finding lots of technical issues that are required to complete functional work •Unable to process SSIM with 7K lines when testing IVI •Refactoring of code and validation functional work at same time •Determining size of technical tasks required and dependencies •Testing pushed to end of sprint again •Issue tracking of code •Test server needing to be restarted 	<ul style="list-style-type: none"> •Get more input from technical specialist during story review •Do KT prior to doing refactoring work and split work up among development team •Saying NO, not taking on too much work •Explore more negative test cases •Do code review earlier in sprint, give time to implement comments 	<ul style="list-style-type: none"> •Plan for more BA visits •Issue tracking of code, , root cause analysis to try and prevent recurring bugs •Get insight into test server to know what is causing server to break



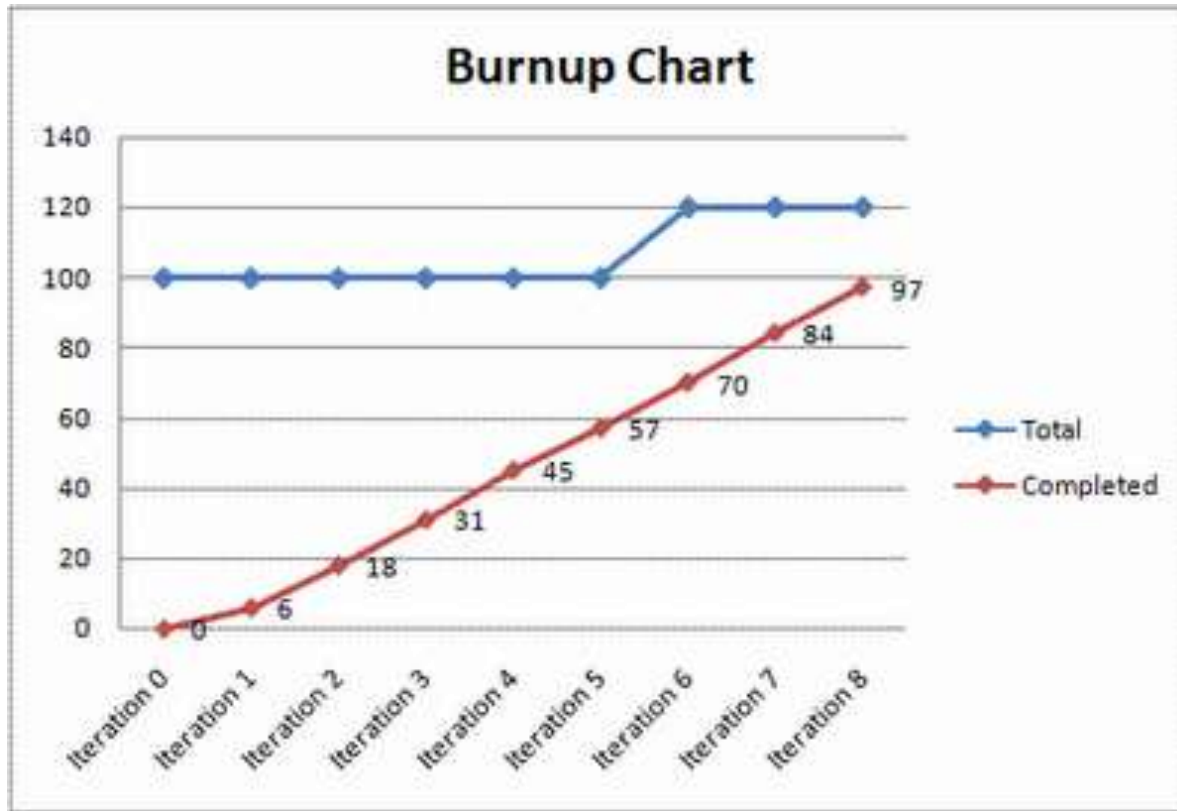
Typical Scrum Metrics



BurnDown Chart



Sample BurnUp Chart



Velocity

Is used as an indication of capacity of future sprints

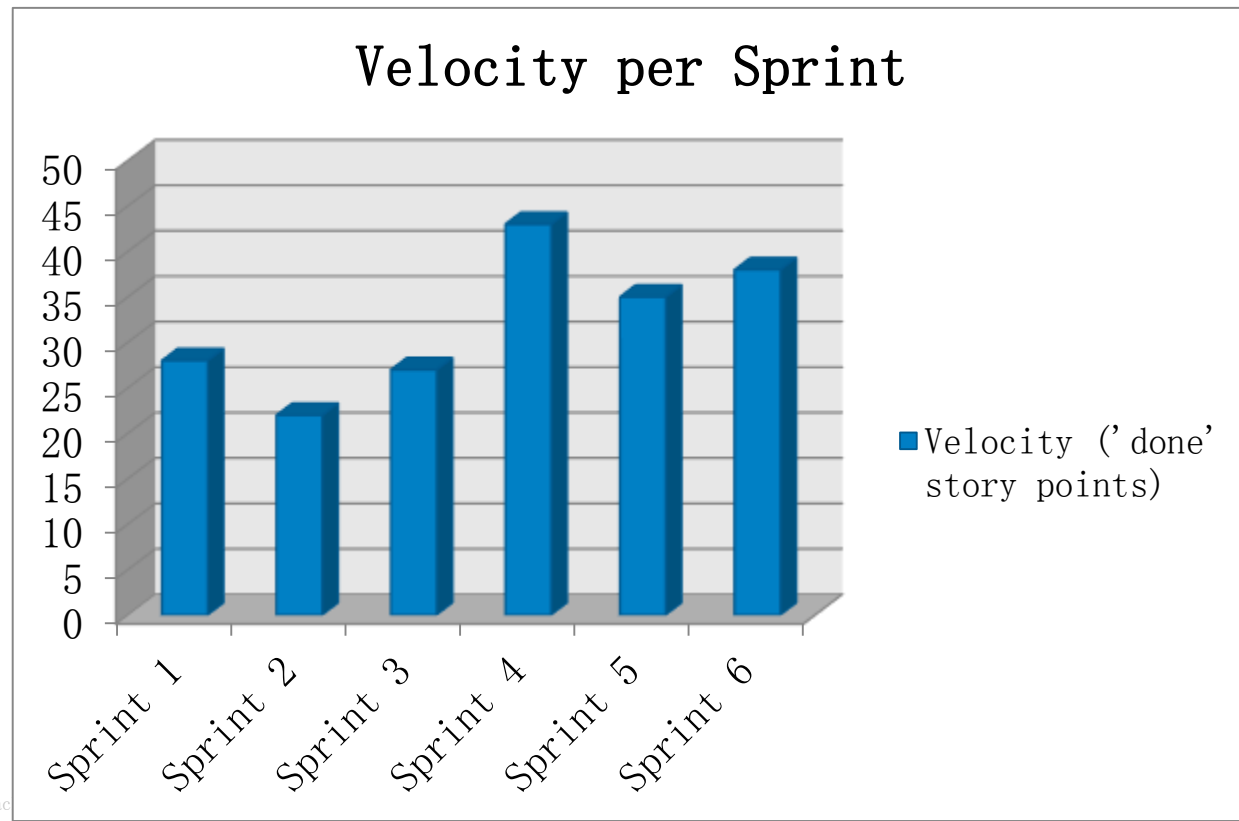
Velocity is a measure of the team's amount of work they can do in each sprint

Velocity is measured in the same units as feature estimates

Velocity is affected by the team's capacity

Velocity tends to increase dramatically initially then becomes relatively stable

For predicting use an average velocity from the last 3-4 sprints



Definition of “Ready”

Prevents sprint churn

Don't let anything that's not READY into your Sprint

Describes the Backlog and User Stories before the Sprint

Quality checklist

Scrum Team decides if OK

For example, “ready” might be:

Detail	Size
All elements defined (e.g. wireframe, test data etc.)	Small enough for the sprint
Meets “Invest” Criteria	Granularity ok
No technical issues	
Any dependencies resolved	



Definition of “Done”

Ensures real, working product

- Don't let anything that's not DONE leave your Sprint
- Describes the product at the end of the Sprint
- Quality checklist
- Static constraints and requirements
- Can vary over time
- It's important for teams to define what is implied by “done” or “complete.”
- **For example, “done” might be:**



Engineering	Testing	Documentation
Working Software	Tests automated	Release notes created
Unit Tested/Code reviewed	Meets acceptance test criteria	Online help/User Docs created
Code integrated into CI and quality metrics meet standards	Reviewed and accepted by Product Owner	Design Documentation created
All defects identified in the sprint resolved	Performance/Load Tested	

Review - Ready & Done

Let's take a few moments to review or draft the definition of ready and done



Questions ??



Agile Estimating and Backlogs



Agenda – Backlog, User Stories, Estimating and Testing

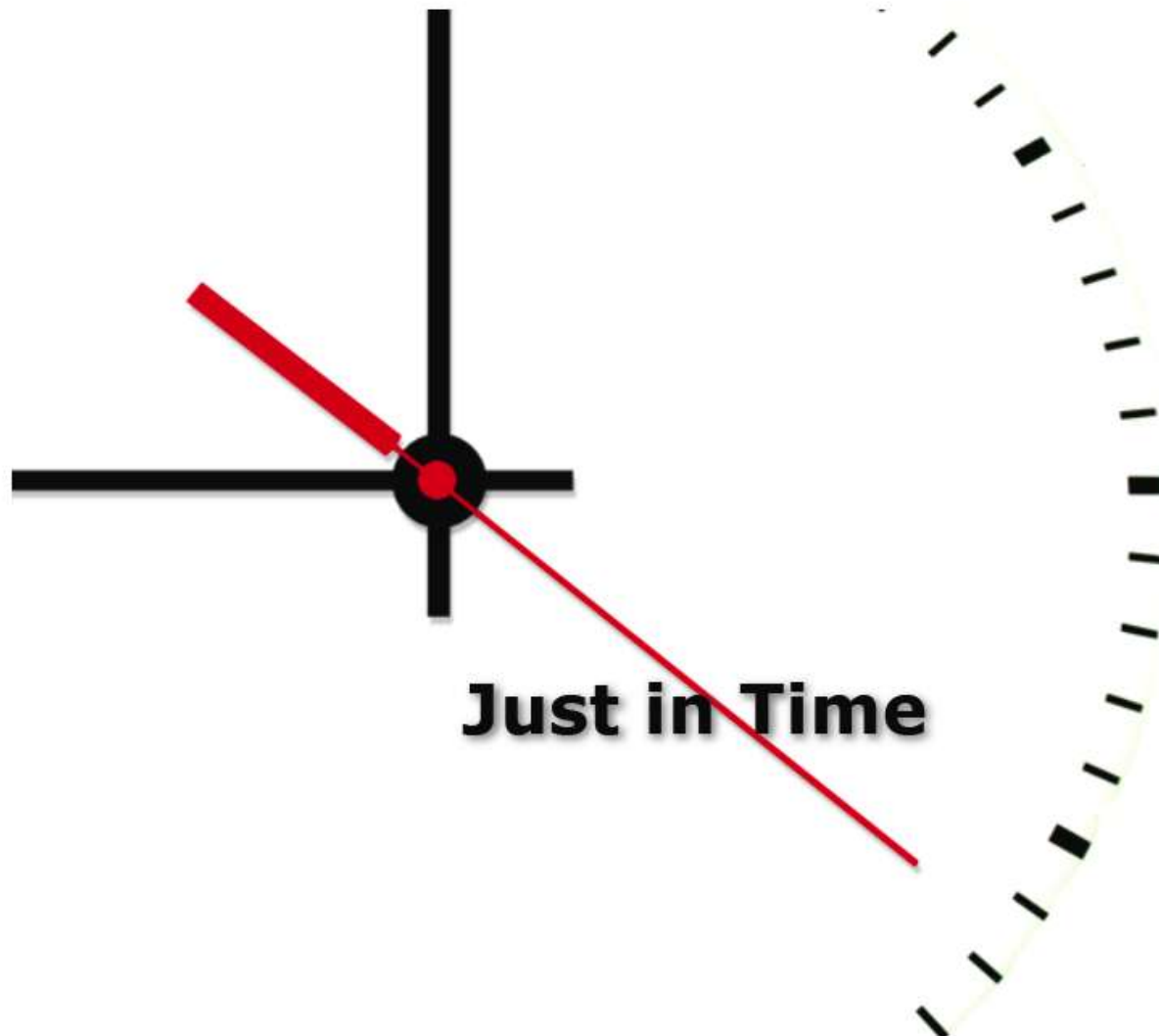
- Backlogs: Product, Release and Sprint
- Agile Requirements and Analysis
- User Stories
- Agile Estimating
- Agile Testing
- HP Agile Edge Review
 - Templates (project plan, team norms, audit criteria)



Agile Requirements Management



Agile requirements are...



Agile requirements Management is different

- Agile thinking recognizes that perfect plans cannot be created, no matter how much effort is invested “Planning is essential but Plans are useless”
- Instead, do just enough, just-in-time:
 - Develop a shared vision with the stakeholders
 - Create a prioritized list of the most valuable features
 - Understand the prioritization and relative sizing of those features
 - Work out a candidate architecture and design (not detailed design)
 - Craft a plan that identifies the Minimum Marketable Feature (MMF) set
- Agree with stakeholders that you will work together to stay within the parameters of that plan.
- The overall objective is to unlock as much value as possible for the user community, while working at a sustainable pace.



Backlogs – Product/Release/Sprint



The Product Backlog

The Product Backlog is a prioritized list of the all the work necessary to bring the product to life:

- Functional and non-functional requirements
- New features to be developed
- Feature enhancements
- Engineering improvement goals
- Exploratory or research work
- Known defects to existing stories

If it is not in the product backlog, it will not be done!

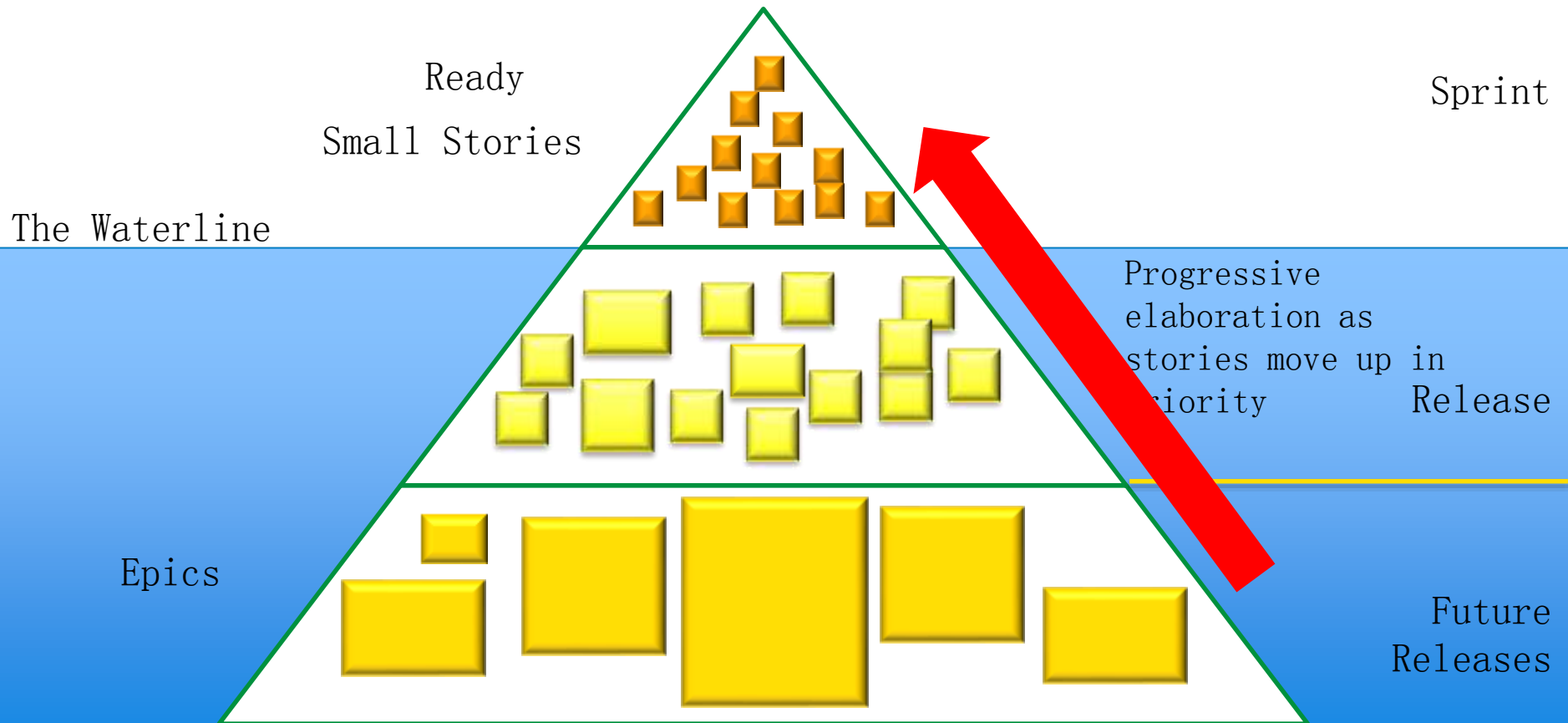
Backlog item	Estimate (story points)
As a registered user, I want to be able to view my bookings	8
As a registered user, I want to be able to reset my password	3
As a guest user, I want to be able to submit a booking enquiry	5
As an portal admin, I want to be able to produce a report of submitted forms	13
As a marketing specialist, I want to define promotional offers	20
As a registered user, I want to update my personal details	8
As the system, I want to timeout a user' s session after a period of inactivity	2
...	...
...	...

The Product Backlog iceberg

Size of backlog items varies across time

Large user stories are called Epics

- Epics should be decomposed before they are allocated to a Sprint



HP - Agile Accelerator Taskboard v5.0 - Windows Internet Explorer

HP - Agile Accelerator Taskboard v5.0

Project : Alm_aa_v50_adv_demo
Release : Release 1.0
Sprint : Sprint 1.0
Progress:

12%

Show Sprint Navigation

Show Story Summary

Show Owner Summary

Show Burn-down Chart

Show Cumulative Chart

Add New Story

All Owners

Hide Completed Tasks

Hide Dev Tasks

Hide Test Tasks

Hide Epic Stories

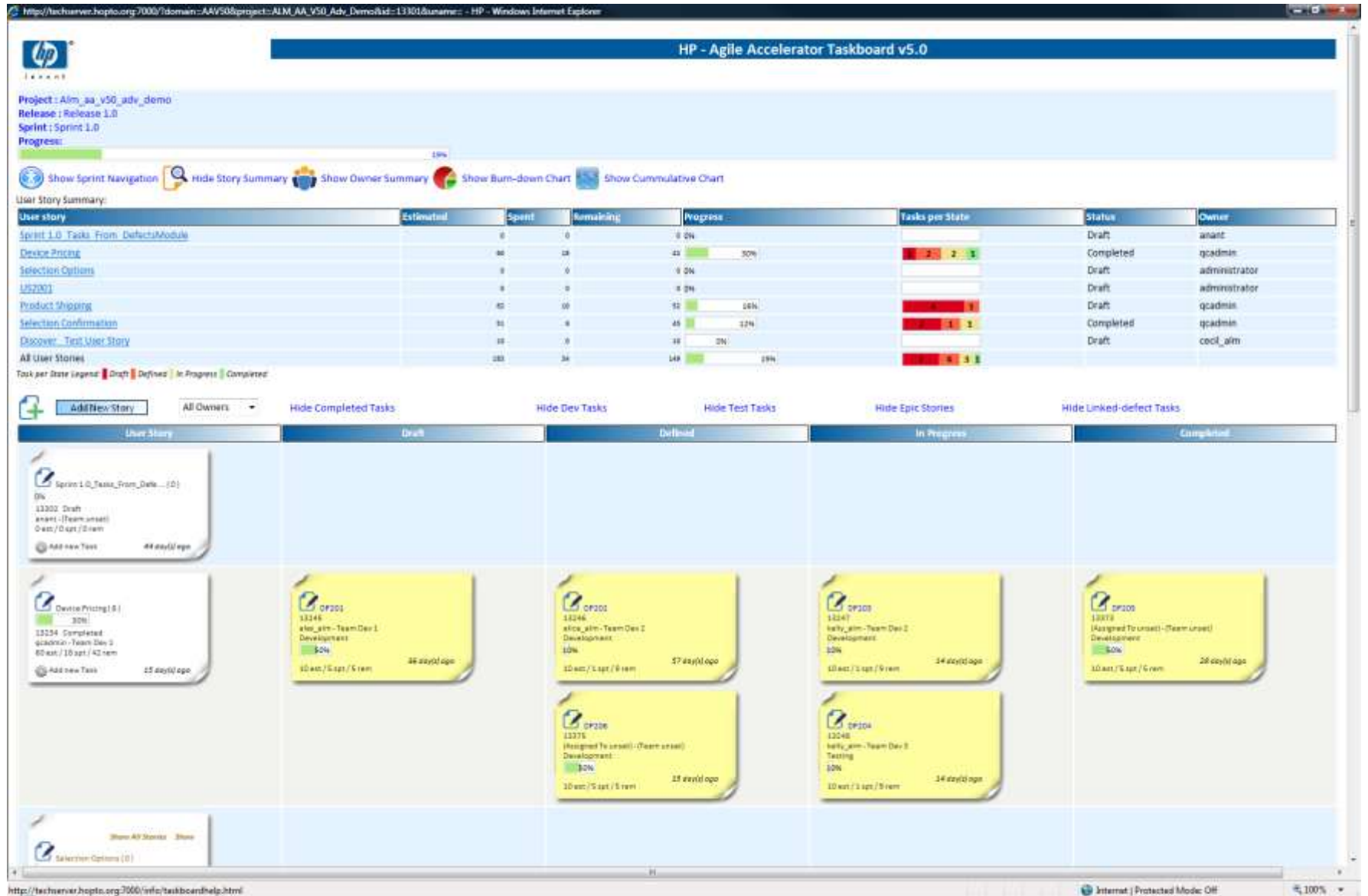
Hide Linked-defect Tasks

User Story	Draft	Defined	In Progress	Completed
<div> <div>Sprint 1.0_Tasks_from_Defe... {0}</div> <div>0%</div> <div>13994 Draft administrator - (Team:unset) 15 est / 0 spt / 15 rem</div> <div>Add new Task 2 hour(s) ago</div> </div>	<div> <div>Test 14076</div> <div>(Assigned To:unset)-(Team:unset) (Task Type:unset) Linked Defect</div> <div>0%</div> <div>15 est / 0 spt / 15 rem</div> <div>2 hour(s) ago</div> </div>			
<div> <div>US 1 {8}</div> <div>12%</div> <div>14065 Draft administrator - Team Dev 1 45 est / 6 spt / 39 rem</div> <div>Add new Task 2 hour(s) ago</div> </div>	<div> <div>Task 1 14067</div> <div>andy_alm - (Team:unset) Development 30%</div> <div>10 est / 2 spt / 8 rem</div> <div>1 hour(s) ago</div> </div>	<div> <div>Task 2 14068</div> <div>alice_alm - (Team:unset) Development 30%</div> <div>10 est / 2 spt / 8 rem</div> <div>2 hour(s) ago</div> </div>		
	<div> <div>Task 2 14071</div> <div>paul_alm - (Team:unset) Development 0%</div> <div>15 est / 0 spt / 15 rem</div> <div>2 hour(s) ago</div> </div>			
	<div> <div>Test_Task 1 14072</div> <div>(Assigned To:unset)-(Team:unset) Testing 0%</div> <div>0 est / 0 spt / 0 rem</div> <div>2 hour(s) ago</div> </div>			

Done

Internet | Protected Mode: Off

100%



Creating User Stories

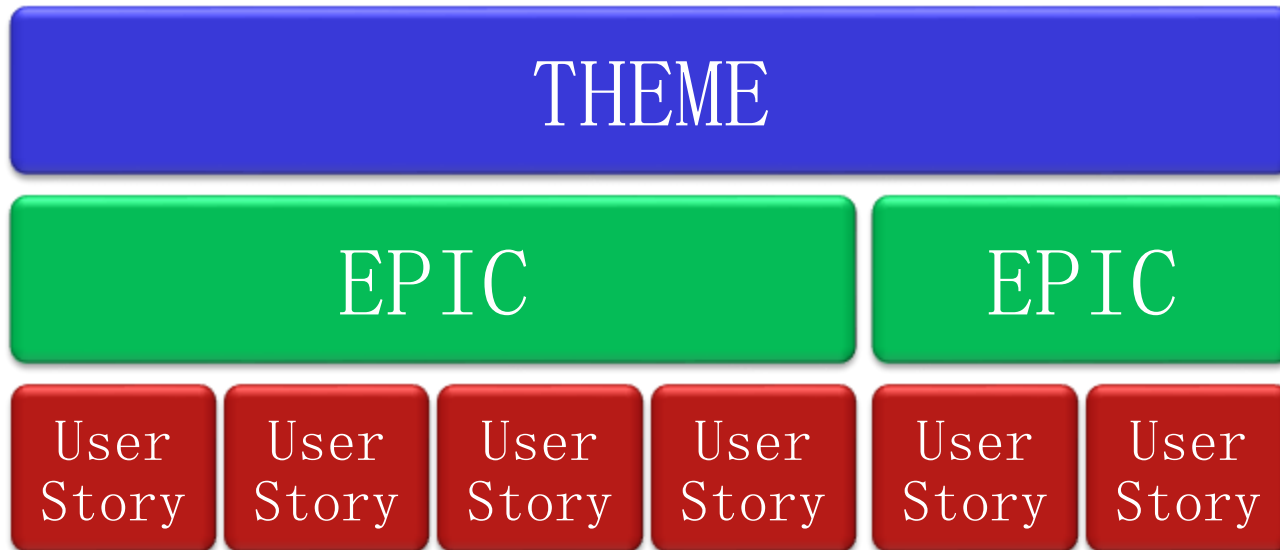


Themes

Grouping of related items in the product backlog

Themes act as placeholders for product functionality

- They structure the backlog
- Aid prioritization
- Easy to access information
- Should contain 2 to 5 coarse grained requirements (epics) – enough information to understand what it will take to bring the product to life without over specifying the backlog contents.



What is a User Story - 3 C's

A concise, written description of a piece of functionality that will be valuable to a user (or owner) of the software

CARD
(index
size)

Conversatio
n

A reminder to have a conversation that serves to explore and capture further information about the user story

Typical user story format:
: **“As a <role> I want to
<feature> so that
<value>”**

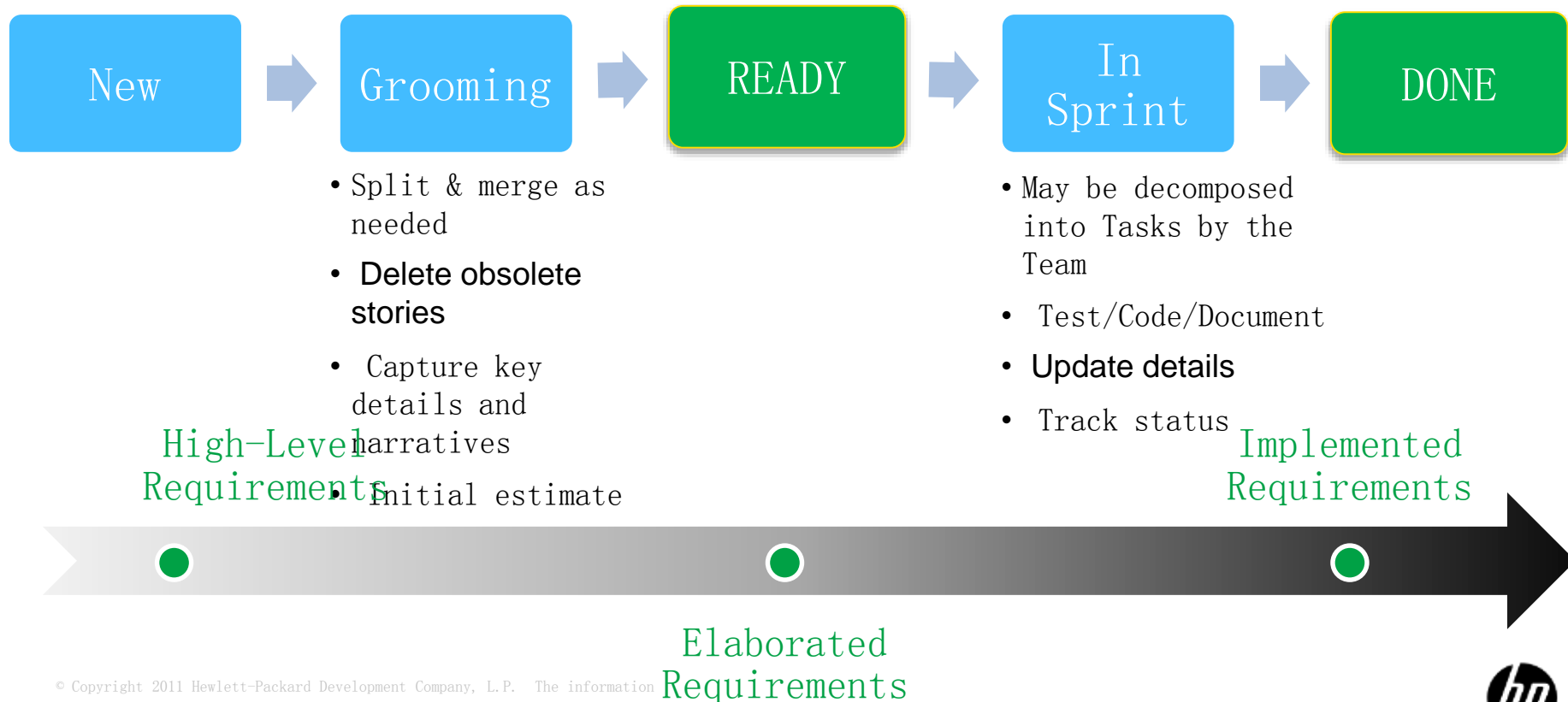
Acceptance Criteria to conform the story meets product owner expectations. (back side of the index card)

Confirmation

Special Stories (not “user”):
Research Spikes, Defects,
Nonfunctional, POCs

Lifecycle of a User Story

- Story written
- Added to the **Product Backlog** by the Product Owner
- Story meets INVEST criteria and Definition of Ready
- Completeness criteria defined
- Refined estimates
- Story meets the Team's Definition of Done
- Demo/Accept



INVEST in good user stories

Attributes of a good user story

	A good user story is...
I	Independent <i>Independent of other stories</i>
N	Negotiable <i>Understood through discussion, conversation, and negotiation between the owner and team at sprint level</i>
V	Valuable <i>Provide tangible benefit to the user or system owner, in the context of the Product Vision</i>
E	Estimatable <i>Can be estimated by the team with a reasonable degree of accuracy</i>
S	Small <i>Sized to fit within an iteration, or can be composed of smaller stories</i>
T	Testable <i>Specifies the criteria for completeness (“acceptance tests”)</i>

Acceptance Criteria Examples

Change password to valid password (accepted)

New password is same as previous password (not accepted)

New password is blank (not accepted)

New password is too short (not accepted)

New password has no number (not accepted)

- Acceptance Criteria Template
- **Given** the customer has just one transaction account
 - **When** they have completed logging in
 - **Then** the screen should show the name and number of the account.

Given some initial context,

When an event occurs,

Then ensure some outcomes

“SMART” Acceptance Criteria

Specific

- Explicitly defined and definite

Measureable

- Possible to observe and quantify

Achievable

- Capable of existing/taking place

Relevant

- Has a connection with the story

Timely

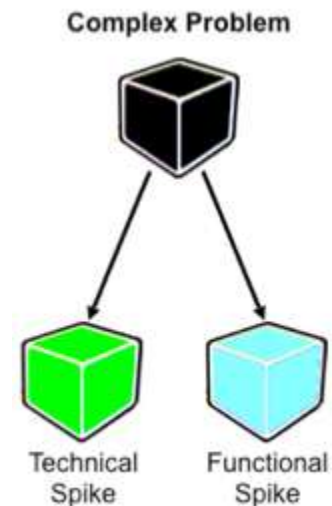
- When will the outcome be observed?

Spikes

A spike is a very focused research effort on an issue or technology that the team does not have prior knowledge on.

In some cases, a story may be too large or overly complex, or perhaps the implementation is poorly understood. In that case, build a technical or functional spike to figure it out:

1. The team may not have knowledge of a new domain, and spikes may be used for basic research to familiarize the team with a new technology or domain.
2. The story may be too big to be estimated appropriately, and the team may use a spike to analyze the implied behavior, so they can split the story into estimable pieces.
3. The story may contain significant technical risk, and the team may have to do some research or prototyping to gain confidence in a technological approach that will allow them to commit the user story to some future iteration.
4. The story may contain significant functional risk, in that while the intent of the story may be understood, it is not clear how to the system needs to interact with the user to achieve the benefit implied.



Handling Defects

“A defect is behaviour in a ‘Done’ story that violates valid expectations of the Product Owner (or customer)”

– Elizabeth Henderickson

Treat defects as “stories”

- They are estimated and prioritized in the backlog

Not all defects need to be logged

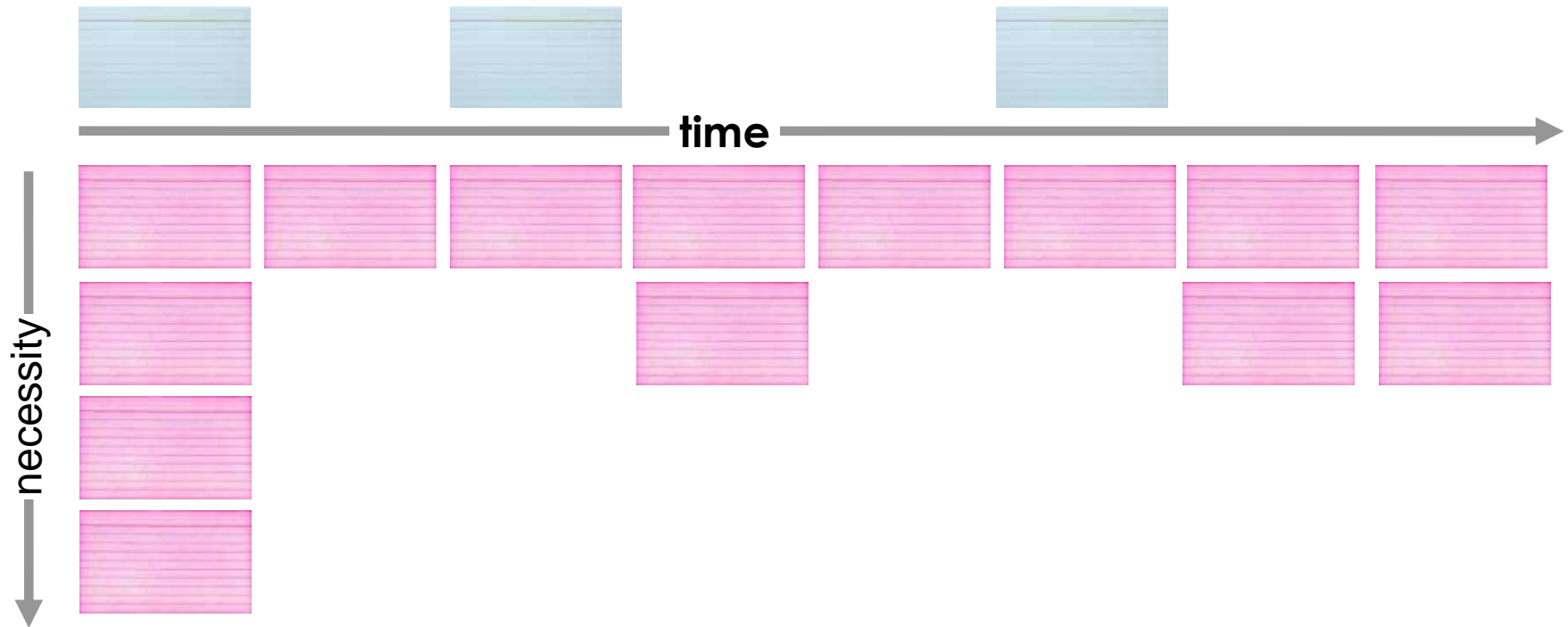
Only capture defects if they won't be fixed right away

You want developers to feel comfortable showing you something that is not complete to get some guidance



Organizing the Backlog: A Story Map

- We keep the bigger picture in mind using a Story Map

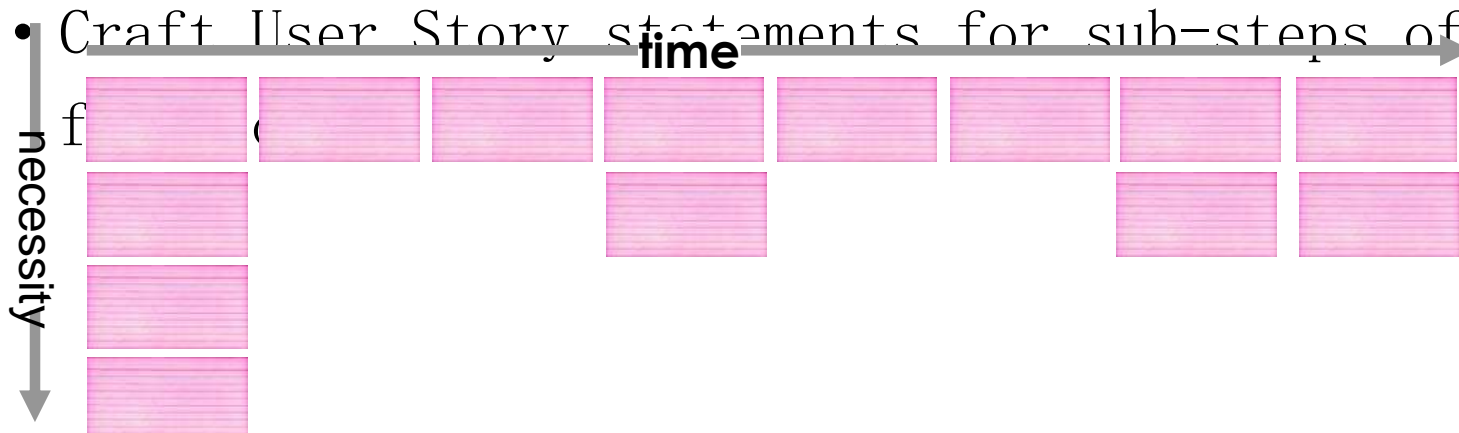


Steps to Writing Stories

- Identify users of the application
 - Refine roles by similarities / unique needs
 - Assign priority to each role, major user vs. minor user
- Create a list of major functions performed by each role



- Craft User Story statements for each major function of each role
- Craft User Story statements for sub-steps of major



Answer Questions

- Who performs this scenario?
- Is there more than one role that could do this?
- Are there different “permissions” for other roles?
- What will the user most likely want to do next?*
- What mistakes could the user make here?*
- What could confuse the user at this point?*
- What additional information could the user need?*

*User Stories Applied for Agile Software Development, Mike Cohn, Addison-Wesley, 2009



Guidelines

- Write from a business perspective
- Avoid implementation jargon
- Reveal value through the “what” not the “how”
- Keep stories intentionally vague so that it's obvious that more information is needed
- Too much detail gives a false sense of completeness



Exercise: Identify Users – For SD Project

- Review/Create a list of user roles for the application
- Think about: Who uses the application? What are the security roles in the system (e.g. admin, manager, general user)?
- Refine the roles by their similarities and unique needs
- Think about: What role will use the system most often? What role will use the system least often?



Exercise: Create the High-Level Story Map

Create the Themes for Releases

Scenario

For the: GSIP Project - Create or Review

1. Create Product Vision and Product Roadmap
2. Create an overall release plan
3. Create the High-Level Story Map
4. Create Release Plan and Themes
5. EPIC - 1- 5
6. User Story

Assumptions:

Record any assumptions you make along the way



Exercise: Create EPICS

Create User Stories

Scenario

For the: GSIP Project - Review

1. Create Product Vision and Product Roadmap
2. Create an overall release plan
3. Create the High-Level Story Map
4. Create Release Plan and Themes
5. EPIC - 1- 5
6. User Story

Assumptions:

Record any assumptions you make along the way

Agile Estimating



What is a Story Point

Probably the most commonly used estimating unit among agile teams today is the story point

Based on a combination of the *size* and *complexity* of the work, it serves to size one story against another (bigger or smaller relative size)

Unitless – but numerically relevant estimates

So if this is a
“5” ...→

Story
A

Story
B

← then this is a
“10”. We would
expect it to take twice
as long to build.

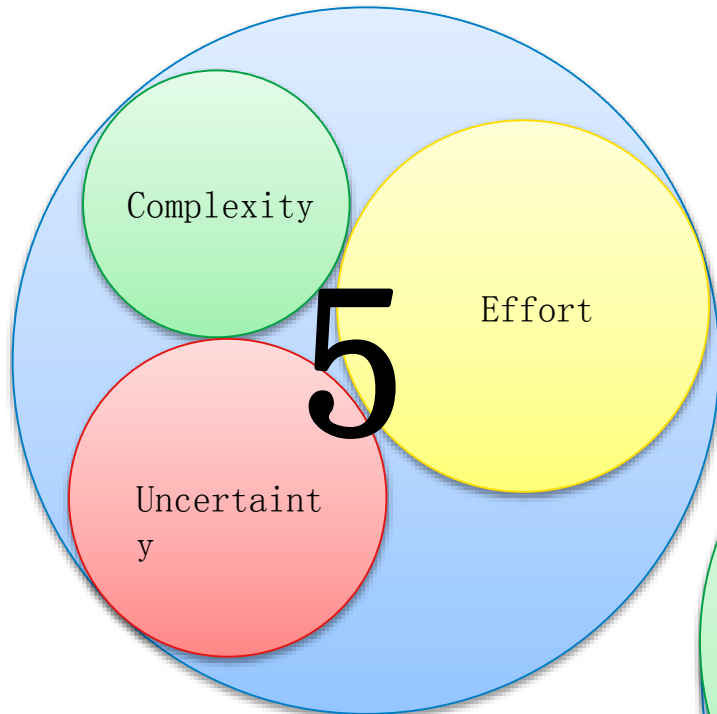
Generally teams use the Fibonacci Sequence (1, 2, 3, 5, 8, 13, 20, 40, 80, 100)

Humans ability to estimate with any degree of accuracy erodes the bigger the item gets.

Using the Fibonacci sequence avoids teams trying to “size” the difference between for example a 12 or 13.

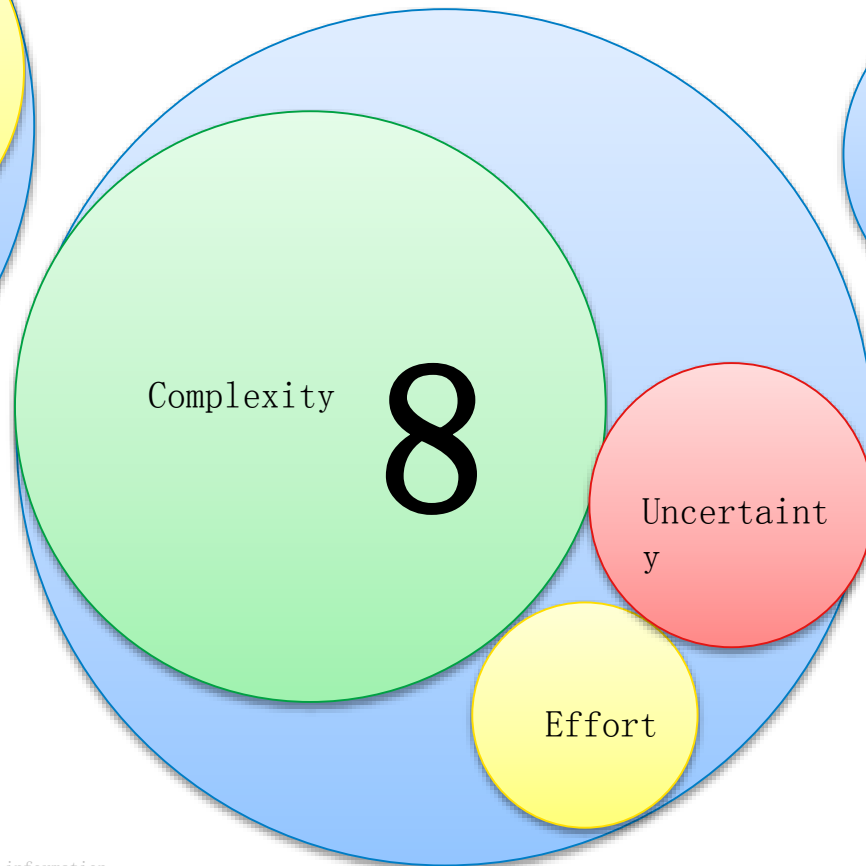
Sizing User Stories

Using Story Points

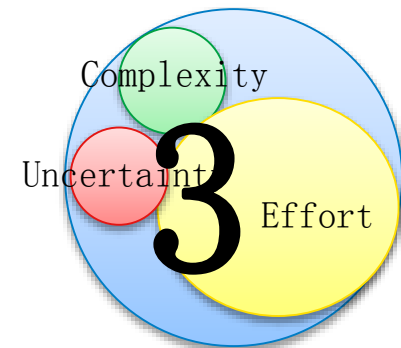


User Story A

Story points are a
measure of
relative size and
are unit-less



User Story B



User Story C

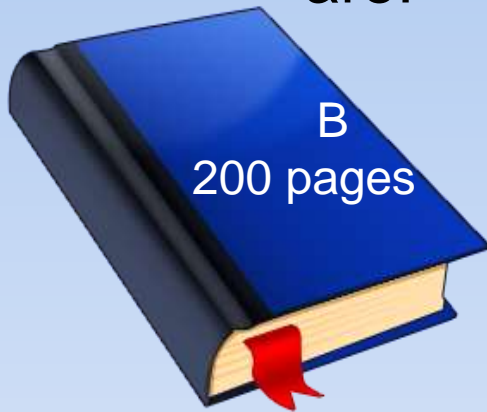


User Story D

How does relative size work?

Story points are the TEAMS assessment of the relative size of a piece of work

Relative size doesn't change but hours estimates depend on who is doing it and what their experiences are!



John reads 50 pages an hour

How long will it take John to read Book A? How about book B?



Jane reads 40 pages an hour

How long will it take Jane to read Book A? How about book B?

Exercise - Sizing

Assign “dog points” to each of the following types of dog to compare their “bigness” .

Labrador Retriever

Dachshund

Great Dane

Terrier

German Shepherd

Poodle

St. Bernard

Bulldog



Agile Testing



Agile Testing Success Factors

Organizational

- Testers are part of the Team
- Collaboration is Key Component

Cultural

- Maintain an Agile Testing Mindset (Focus on Value, not bug-killing)
- Don' t lose the “Big Picture”
- Continually Improve (personally and as a team)

Technical

- Automate wherever practical for immediate and future use
- Maintain a base foundation of Leveragable Collateral



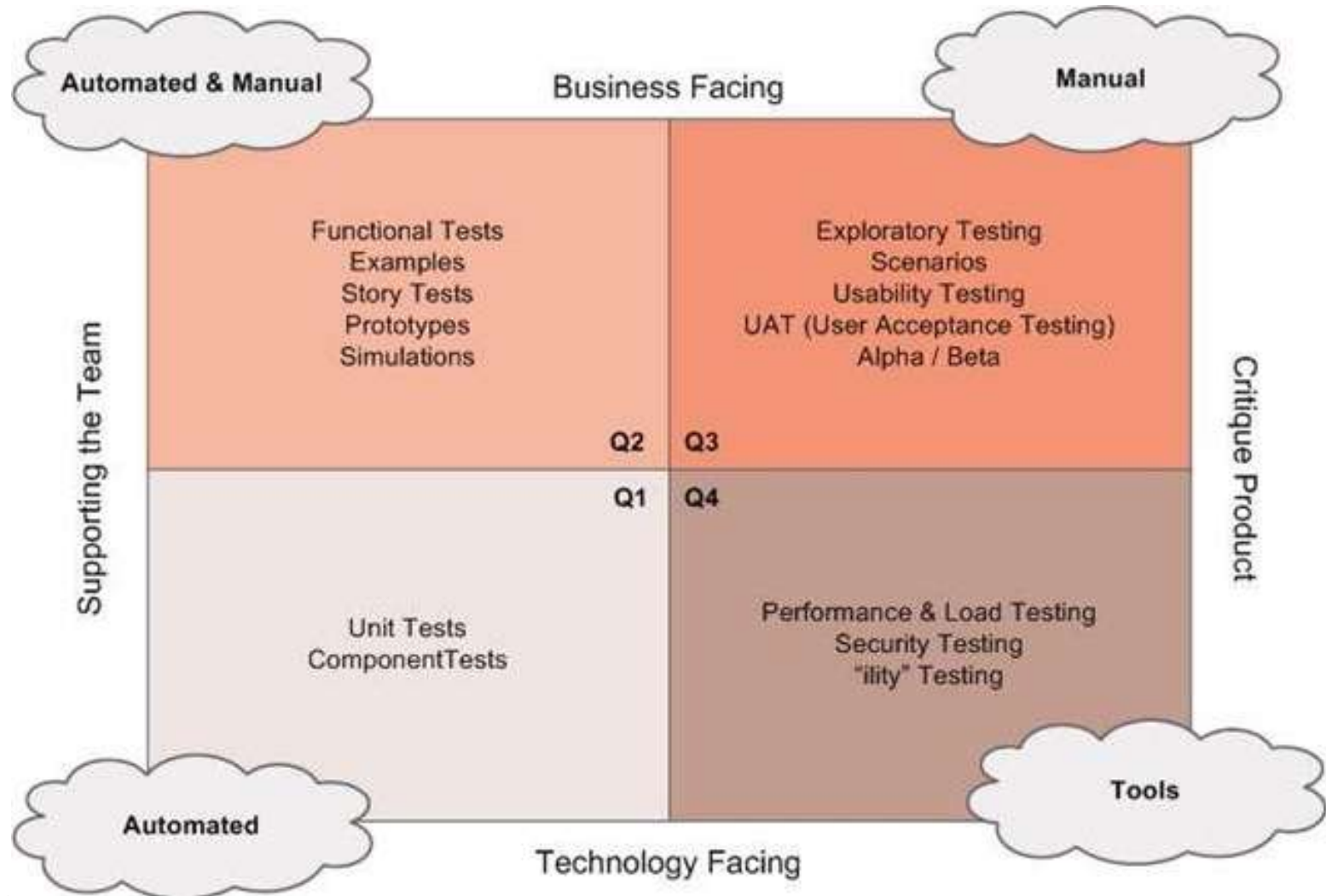
Agile Testing Principles

- Testers help with understanding and validating requirements
- Build quality in—QA's primary role is to prevent
- Move testing up front - prevents waste
- Favor testers in the Scrum teams vs. an outside "QA" or "Test Team" when possible
- Test early, test often & everyone tests
- Automate tests to the degree possible
- Find and fix defects as close to when they are introduced as possible



The Agile Testing Quadrants

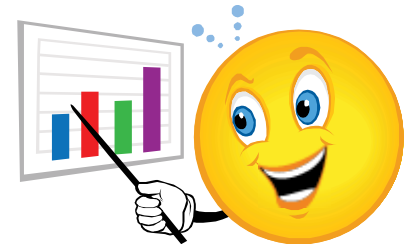
(Brian Marick)



Testing with Evolutionary Requirements and Solution

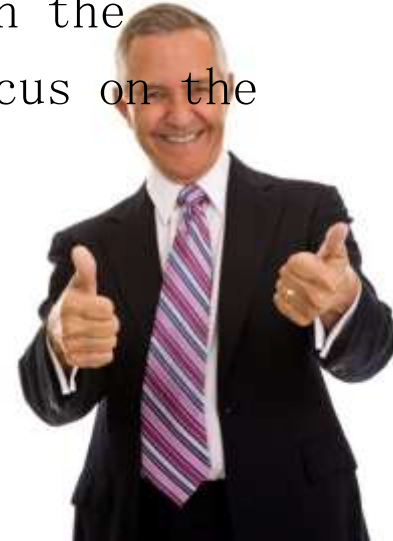
Agile Testers:

1. Testers can help the whole team work in a test-driven way by:
 - a. Assisting with defining requirements
 - b. Asking questions- and generating system-level tests from these stories.
2. Testers provide information about each delivered chunk
3. System-level regression testing during an iteration



Story Testing / Acceptance Testing / Customer Tests

- The term “acceptance testing” in an agile context means testing whether a user story meets the acceptance criteria specified for it.
- Agile acceptance testing should be done in the sprint/iteration in which the story is developed—part of “Done Done”
- Tests validate communication and understanding
- Automate as much as possible for future regression testing
- Good tests focus on the *essence* of your rules NOT on the *actions* (enter text, hit this button, ...) But do focus on the *things* (when X then Y)

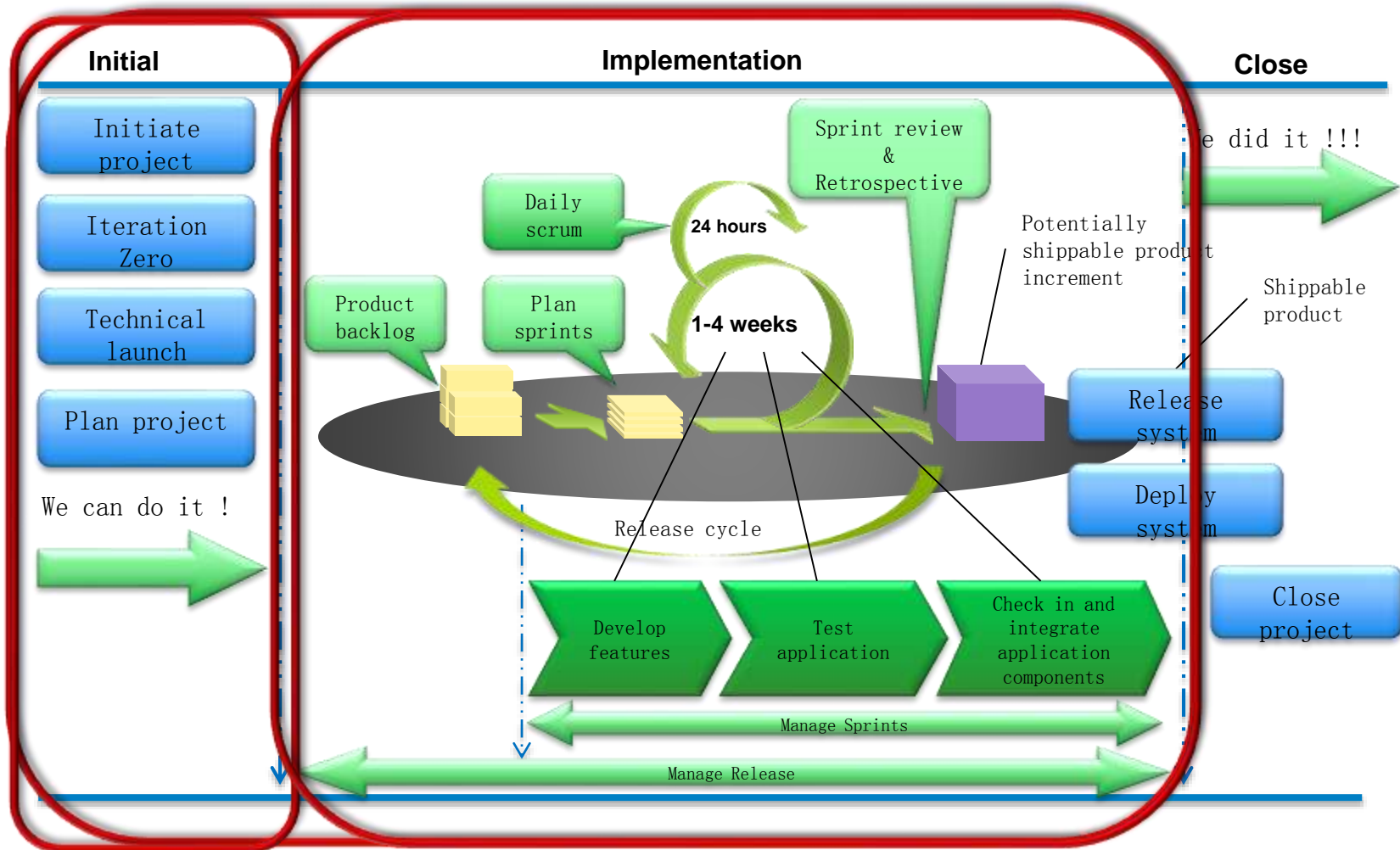


HP EDGE Agile Development



EDGE Agile lifecycle

Based on Scrum + XP



Edge – Demo

We will now switch to EDGE and briefly cover the process.

<http://edge.corp.hp.com/>

The EDGE process is a framework. It is not prescriptive as to flow and task details. Be sure to view agile specific guidance (must click on link to display it)

There are few required documents:

- Agile – Project Plan Template (single document)
- Data Strategy
- Test Strategy
- Team Norms*



Agenda – Technical Practices

- Technical Practices - from eXtreme programming
 - Test Driven Development/Behavior Driven Development and Code Reviews
 - Technical Debt/Refactoring
 - Collective Code Ownership
 - Continuous Integration/Automated Build
 - Code Metrics and Static Code Quality Analysis
 - Pair Programming
- Agile Design Principles

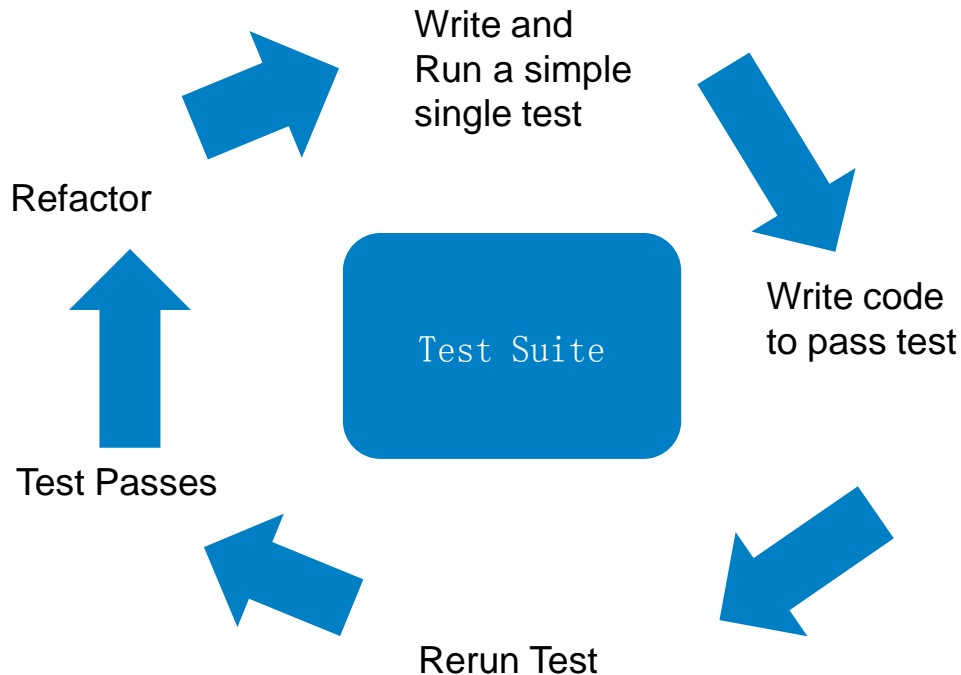


Test and Behavior Driven Development / Code Reviews



Test Driven Development (TDD) Flow

TDD is primarily a **DESIGN** practice (not testing or coding)



TDD Recipe Card

1. Write a test
2. Run the test (it fails)
3. Design/write code (just enough to pass the test)
4. Re-test
repeat steps 3 & 4 until test passes
5. Refactor
6. Repeat until the story meets all acceptance criteria

Move from Unknown-to-known in small, discrete steps
Focus on class *behavior* (does it do what it should?) over
implementation (how does it do it?)

Test Driven Development Benefits

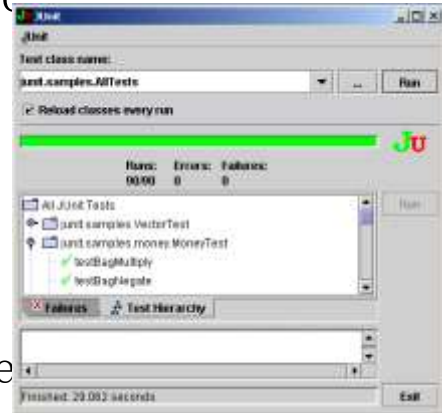
Leads the developer to primarily think about “how will the component be used?” (why do we need the component, what’s it for?) and only secondarily about “how to implement it?”

This is known as PROGRAMMING BY INTENTION

- It results into components that are easy to test.
- It results into components that are easy to enhance and adapt

You get a suite of automated tests that can (and should) be run going forward to prove the code works

Programmers know when they are done coding (all the tests pass), which results in less ‘gold-plating’



Test Driven Development – Example (1)

Fibonacci Numbers – The sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21...

1. First Test shows fib (0) = 0

```
public void testFibonacci ( ) {  
    assertEquals (0, fib (0));  
}
```

```
int fib (int n) {  
    return 0;  
}
```

Run All Tests → Pass (Green)

Refactor → Pass

2. Second Test shows fib (1) = 1

```
public void testFibonacci ( ) {  
    assertEquals (0, fib (0));  
    assertEquals (1, fib (1))  
}
```

```
int fib (int n) {  
    if ( n == 0) return 0;  
    return 1;  
}
```

New
Functionality

Quick Fix

Run All Tests → Pass (Green)

Refactor → Pass

Test Driven Development – Example (2)

Fibonacci Numbers – The sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21...

3. Refactor

```
public void testFibonacci ( ) {
    int cases [ ] [ ] = { { (0,
0), (1, 1) } } };

    for (int i = 0; i < cases.length;
i++)
        assertEquals (cases [i][1], fib
(cases [i] [0] ));
}

*****
int fib (int n) {
if ( n == 0) return 0;
return 1;
}
```

Run All Tests → Pass (Green)


Refactor → Pass

4. Third Test shows fib (1) = 1

```
public void testFibonacci ( ) {
    int cases [ ] [ ] = { { (0, 0),
(1, 1), (2, 1) } } };

    for (int i = 0; i < cases.length;
i++)
        assertEquals (cases [i][1], fib
(cases [i] [0] ));
}

*****
int fib (int n) {
if ( n == 0) return 0;
return 1;
}
```



Run All Tests → Pass (Green)

Refactor → Pass

Behavior-Driven Development (BDD)

BDD Practices

- Using examples to describe the behavior of the application from the user perspective
- Automating those examples for quick feedback and regression testing
- Using Given/ When/Then for describing the behavior of software (acceptance test)
- Using mocks to stand-in for collaborating modules of code which have not yet been written

Can serve to communicate more of the story details as well as acceptance criteria

For examples and more technical details see: <http://dannorth.net/introducing-bdd/>



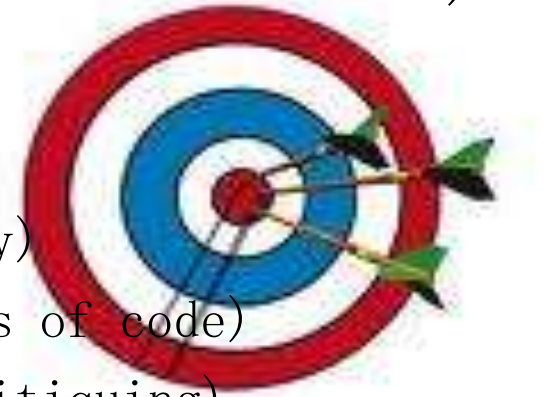
Code Reviews

Agile design/code reviews should be as much (or more) about **education** than defect discovery

Involve experts early (reviewer as *film editor*) rather than at the end of a step (reviewer as *film critic*)

Agile reviews should be:

- **Early** (don' t wait until the end)
- **Effective** (focus on value-added discovery)
- **Efficient** (short, looking at small chunks of code)
- **Educational** (more like mentoring than critiquing)



Build Quality In vs. *Inspect Quality In*

Static code quality tools can help you pinpoint areas for further exploration

A look at Jupiter plug-in for eclipse

<http://code.google.com/p/jupiter-eclipse-plugin/wiki/UserGuide>

Start in section 3.0

Look at simplified EDGE WPR



Refactoring & Technical Debt



Technical Debt

Avoid accruing technical debt!

- It slows the team down
- It lowers morale and increases fear

Forms of technical debt:

- Code without tests (legacy code)
- (Unnecessarily) complex code and designs
- Duplicated code or otherwise poorly-written code
- Tightly-coupled interfaces or code in need of abstraction
- Poorly-implemented development (build) and testing platforms



Broken Window Theory

How do you pay down your debt?

- Refactoring, inter-sprint breaks, code reviews
- Don' t create additional debt!

Additional Study ([Grow@HP 1019757](#)) Emergent Design: Evolving Systems With High Efficiency and Low Risk

Refactoring

“Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior” Martin Fowler

When to Refactor

- Incorporating Evolving Design
- Overly-complex or bulky code
- Unclear class responsibilities, architectural layering
- Easier to implement current story by doing some re-design

Entropy wins over time if not constantly refactoring

When not to refactor

- Refactoring is not changing working code just to make it look neat or simpler It is not the same thing as “re-writing” or “re-formatting”
- Avoid using “this is not how I like it” criteria
- Avoid premature or speculative refactoring
- Refactoring requires a PURPOSE and a TECHNIQUE



Refactoring Tips

Some signs you should consider refactoring can be found at:

<http://www.industriallogic.com/papers/smellstorefactorings.pdf>

Pay down Technical debt with refactoring!

Refactor in steps (your IDE can do some for you) - NOT all-at-once!

Learn how to implement Design Patterns

You cannot do refactoring efficiently without also institutionalizing collective code ownership and continuous integration

There are also techniques for database refactoring

NOTE: You cannot reliably refactor your code if it does not have unit tests! They are your safety net.

Repeat: Refactoring is dangerous without tests - write tests first
Tips on what and how to refactor: <http://www.refactoring.com/catalog/index.html>
if necessary



Collective Code Ownership

We are all responsible for high-quality code

Avoid appointing “owners” for code, but leave room for

“experts” who will guide others through dangerous territory...

Fix problems no matter where you find them

Always leave the code a little better than you found it

Code must be available to everyone

- for updates
- for refactoring
- when the ‘expert’ is on vacation

Embrace continuous integration!

“Collective ownership increases your feeling of personal power on a project. You are never stuck with someone else’s stupidity. You see something in the way, you get it out of the way. “

– Kent Beck, Extreme Programming Explained

Automated Build Continuous Integration and Code Metrics



Automate the Build

The build should:

- Be automated
- Be comprehensive but not complex
- Be flexible (add/move/delete step)
- Be fast (10 minutes or less)
- Include automated tests (regression testing)
- Result in either success or failure
- Be run at least daily/nightly (preferably at each check-in)



Automating your build is one of the easiest ways to improve morale and increase productivity

Slow tests are the most common cause of slow builds

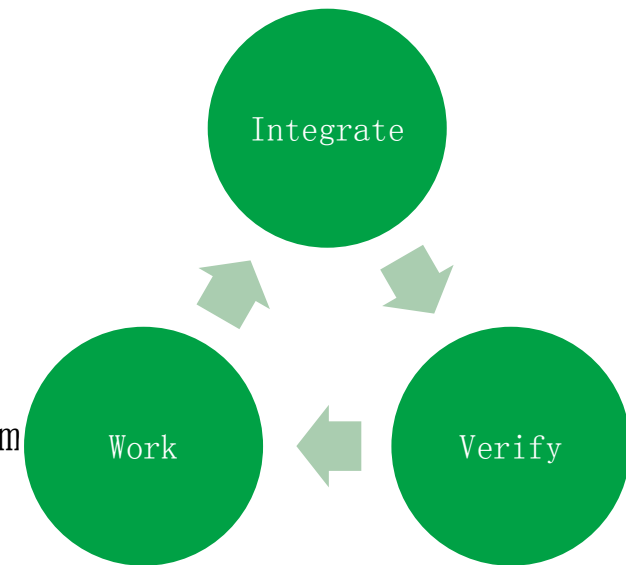
Continuous Integration (CI)

“Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates daily at minimum - leading to multiple integrations per day. Each integration is verified by an automated build (including a set of tests) to detect integration errors as quickly as possible”

– Martin Fowler

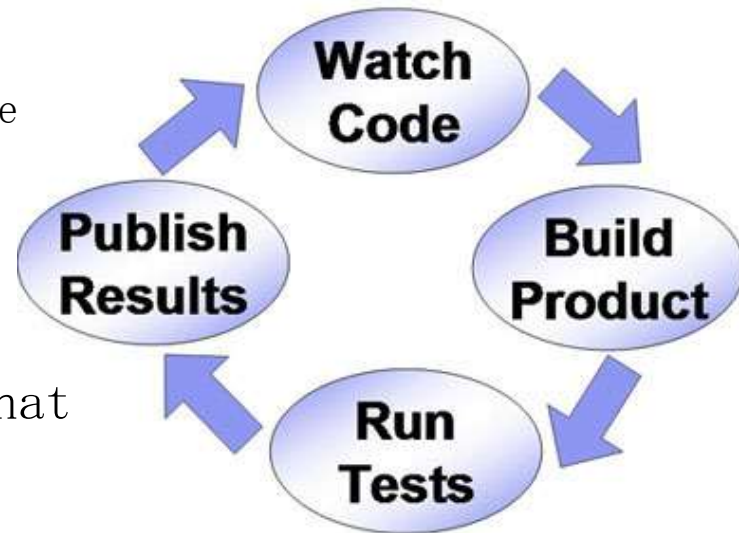
Continuous Integration is...

- A developer practice
- To keep a working system
- Growing the system by small changes
- By integrating at least daily - preferably multiple times a day
- As few code branches as possible (goal - 1)
- Supported by an automated CI system
- With lots of automated tests



CI Benefits

1. Improved feedback and project visibility
2. Improved error detection:
 - Effective continuous integration requires automated unit testing with appropriate code coverage (minimum 80%)
3. Improved collaboration
4. Improved system integration
5. Reduced number of parallel changes that need to be merged and tested
6. Reduced number of errors found during system testing
7. Reduced technical risk
8. Reduced management risk



Continuous Integration Tools



Automated Build Tools

Ant

Groovy (Gant)

Maven 1

Maven 2

Nant

Rake

Build Scheduler Tools

AnthillPro

Apache Continuum

Atlassian Bamboo

CruiseControl (Java)

CruiseControl.NET

ElectricCommander

Hudson

Continuous Integration Reports

Continuous integration can add report for any type of report tool:

- Compiler warnings
- Checkstyle
- PMD
- Cobertura
- Metric Tools: JavaNCSS, JDepend



Code Metrics and Static Code Quality Analysis

What is “good” agile code?

- Meets standards
- Low complexity
- Has automated tests
- Modular, loosely-coupled, high-cohesion
- Bug-free
- Appropriate abstraction (no duplication)

Verify with tooling, preferably tied to the automated build and test cycle

- Checkstyle, FindBugs, PMD, FxCop. CAST (commercial), Parasoft (commercial), Jdepend, Ndepend, JavaNCSS, Sonar, Panopticode

For more information:
Neal Ford (ThoughtWorks)



Sonar demo

<http://nemo.sonarsource.org/>

Nemo is the online demo site for Sonar
Sonar has Eclipse plug-in and works with .NET



Pair Programming



Pair Programming / Pairing

Driver and navigator
produce code through
conversation

Small, frequent design
steps

Ask questions and discuss

Switch partners and roles
frequently

Collaborate, don't
critique—we help each
other succeed

Real-time code review



A pair programming session...

How do we make this test pass?

Does this code “smell”? Or is it Bill?

I saw this same code in another class—we need to refactor!

I need some coffee!

Does this code follow our team standards?
This code is confusing—is there a simpler way to do this?

What else should we test?

I have an idea—can I ‘drive’ for awhile?



Pairing Tips

Establish Team Norms on pairing

Pair on everything you' ll need to maintain

Allow pairs to form fluidly rather than assigning partners

Switch partners when you need a fresh perspective (“promiscuous pairing”)

Avoid paring with the same person for more than one day at a time

Sit comfortably, side by side

Produce code through conversation. Collaborate, don' t critique

Switch driver and navigator roles frequently

When a pair “goes dark” —talks less lowers their voices, or doesn' t switch off with other pairs—it' s often a sign of technical difficulty

If you' re bored while pairing, consider how you can make your design less repetitive.

From “The Art of Agile Development”
by James Shore and Shane Warden



Pair Programming video

<http://www.videosurf.com/video/bizness-on-rails-part-2e-pair-programming-1255476475>

What were some of the things they pointed out as to why pairing works?

Should there be some team norms around pairing?



Agile Design Principles



Developing an Evolutionary Architecture

1. Envision the Architecture

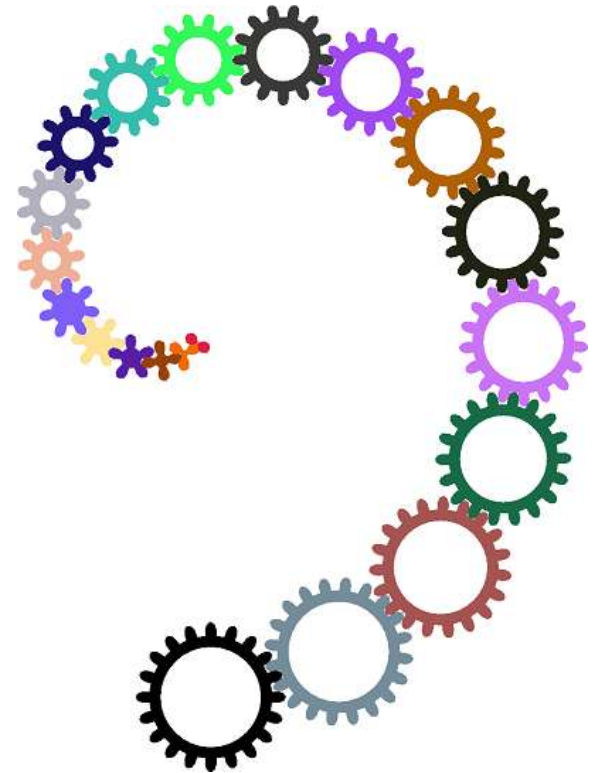
- Develop the initial “vision” for the architecture through analysis of the architecturally significant requirements and identification of architectural constraints, decisions and objectives

2. Evolve the Architecture

- Refine the architecture to an appropriate level of detail to support development. This is an ongoing activity.

Additional Reading:

<http://www.agilemodeling.com/essays/initialArchitectureModeling.htm>



For more information:

“Is Design Dead?” by Martin Fowler

Simple Design

Simple solutions

Do enough design for the moment (iteration)

“What’s the simplest thing that could possibly work?”

Simple Design is still Good Design

Fail fast (use assertions to validate design)

Simple design is not an excuse for poor design.

“Simple” does not mean “easiest” or “fastest”.

Refactor, Refactor, Refactor!



Simplicity is the
ultimate
sophistication.
– Leonardo da
Vinci

“Simplicity--
the art of
maximizing the
amount of work
not done--is
essential.”

*Agile
Manifesto,
Principle #10*



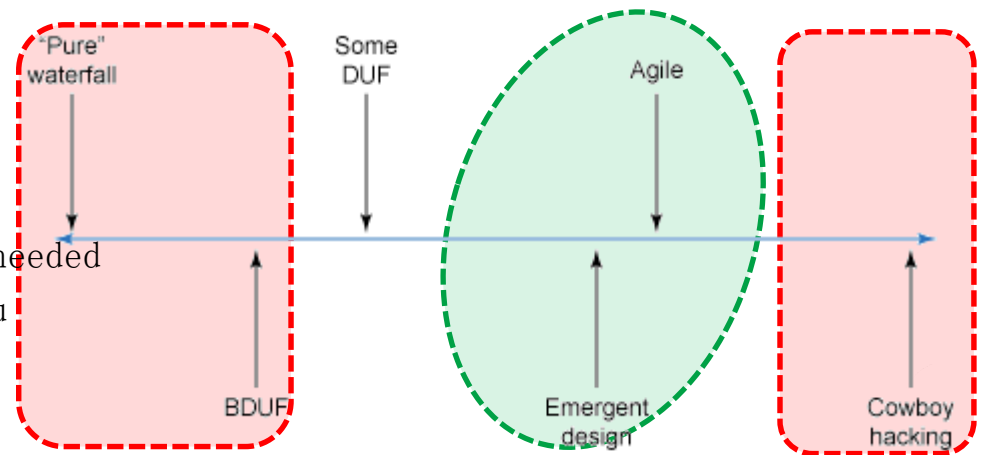
Emergent Design / Evolutionary Architecture / Simple Design

What is the problem with Big Design Up Front? (BDUF)

- Rarely “correct”. There are “known unknowns” and “unknown unknowns”
- Often makes assumptions that prove to be false, and assumes static requirements
- Often created by people who are not familiar with current technology or practices
- Often (always?) results in significant waste and re-work (or non-use)
- Does not allow for feedback (working code) to inform the design
- Leaves little room for improvement or creativity (what if we discover a better way?)

So is there a better way?

- Evolve the architecture as you go by doing minimal up-front design and only make big decisions just before you need to, informed by the work to-date (use spikes as necessary)
- Allow the design to emerge from iteration-to-iteration as the team learns and adapts
- Allow everyone to design, everyday
- Make barely good-enough models and prove them with working code
- Prove design assumptions with code
- Design with simplicity for what is needed today; refactor tomorrow if/when you need it



Simple Design tips

- Write self-documenting code that reveals intention
- Encapsulate components or logic that may need to change in the future
- Limit *published* interfaces (expose API' s gradually)
- Avoid introducing complexity or abstraction that is not yet needed
- Design Patterns can be helpful, but they can also be overdone
 - Do not jump too quickly to design patterns.
 - When you do utilize a design pattern, prefer a simpler implementation of it (there are multiple ways to implement a given design pattern)
 - It is usually better to refactor to a pattern rather than starting with one (i.e., start concrete, generalize later)
- Test-Driven Development is a good way to introduce simple design
- Alan Shalloway' s rules of simple design:
 - Run all the tests
 - Follows once and only once rule (no duplicate code)
 - Has high cohesion (clarity)
 - Has loose coupling



References



Additional References

- Agile Architecture Method – <http://shapingsoftware.com/2009/03/02/agile-architecture-method/>
- Is Design Dead – <http://martinfowler.com/articles/designDead.html>
- “Refactoring: Improving the Design of Existing Code” by Martin Fowler
- “Refactoring to Patterns” by Joshua Kerievsky
- “Refactoring Databases: Evolutionary Database Design” by Ambler & Sadalage
- “Design Patterns: Elements of Reusable Object-Oriented Software” (GoF)
- Refactoring catalog
- <http://www.refactoring.com/catalog/index.html>
- Smells-to-Refactoring catalog
- <http://www.industriallogic.com/papers/smellstorefactorings.pdf>

Questions

