CTIS359
Principles of Software Engineering

AGILE Software Development, Scrum and XP Frameworks

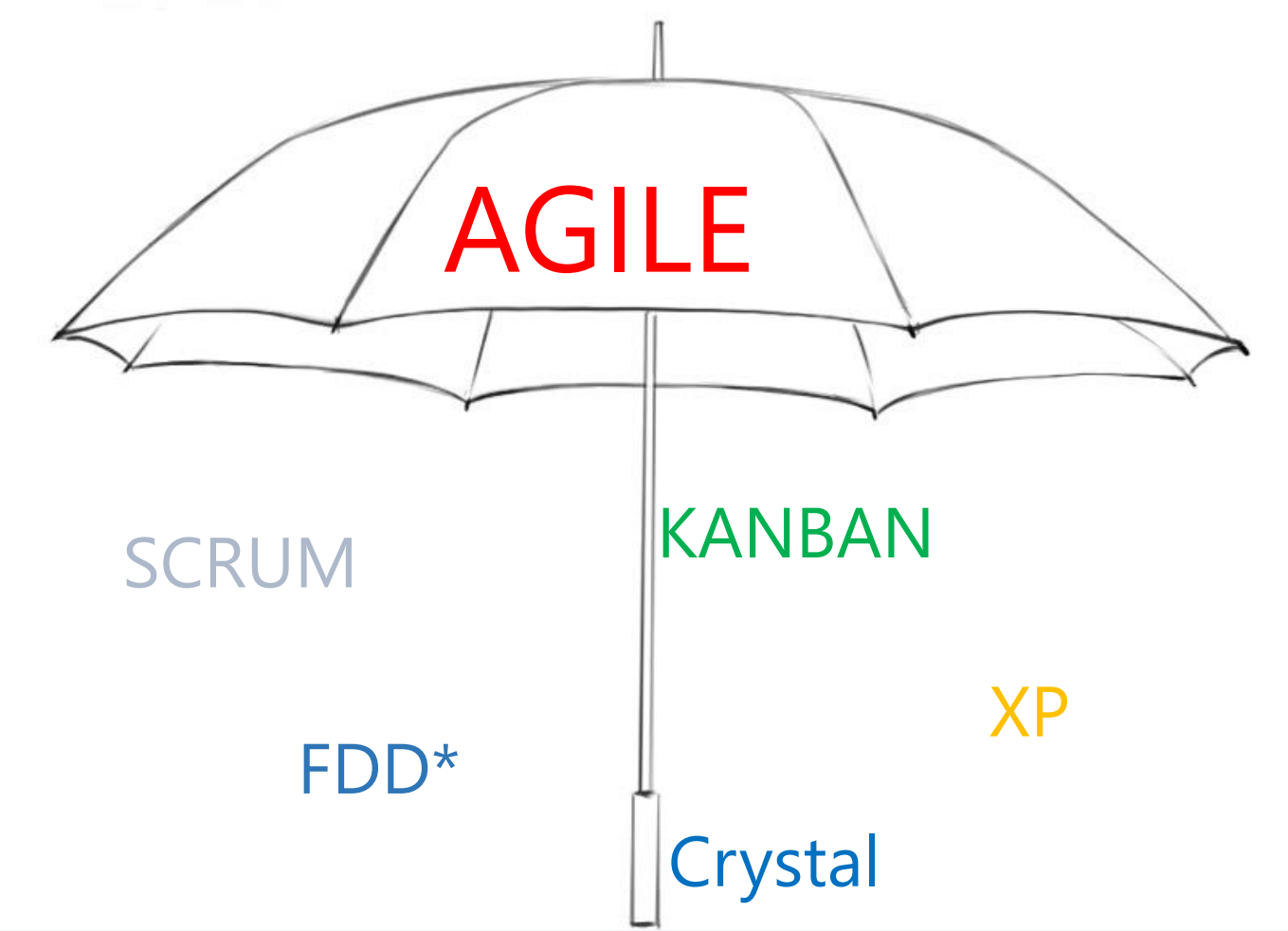
"I'm not a great programmer; I'm just a good programmer with great habits."

Kent Beck

Today

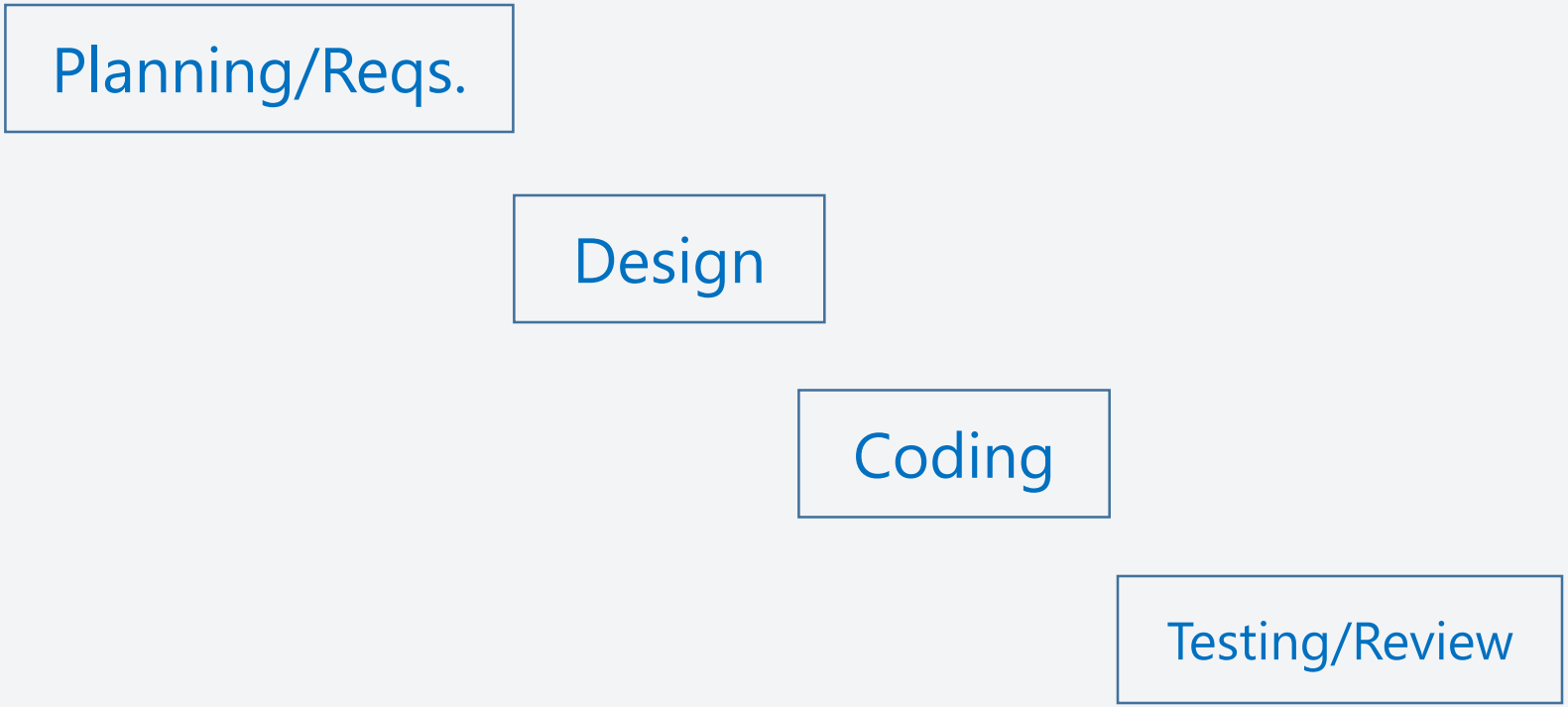
- Agile software development
  - Manifesto
  - Principles
- Well-known agile development frameworks
  - Scrum
  - XP
  - Kanban
  - FDD
  - Crystal

Agile Development & Its well-known frameworks



*Feature Driven Development

Planned software development approaches, or heavyweight approach



Traditional

Planning/Reqs.

Design

Coding

Testing/Review

You can move on the next phase/process ONLY AFTER you have completed the previous one.

Complete SRS

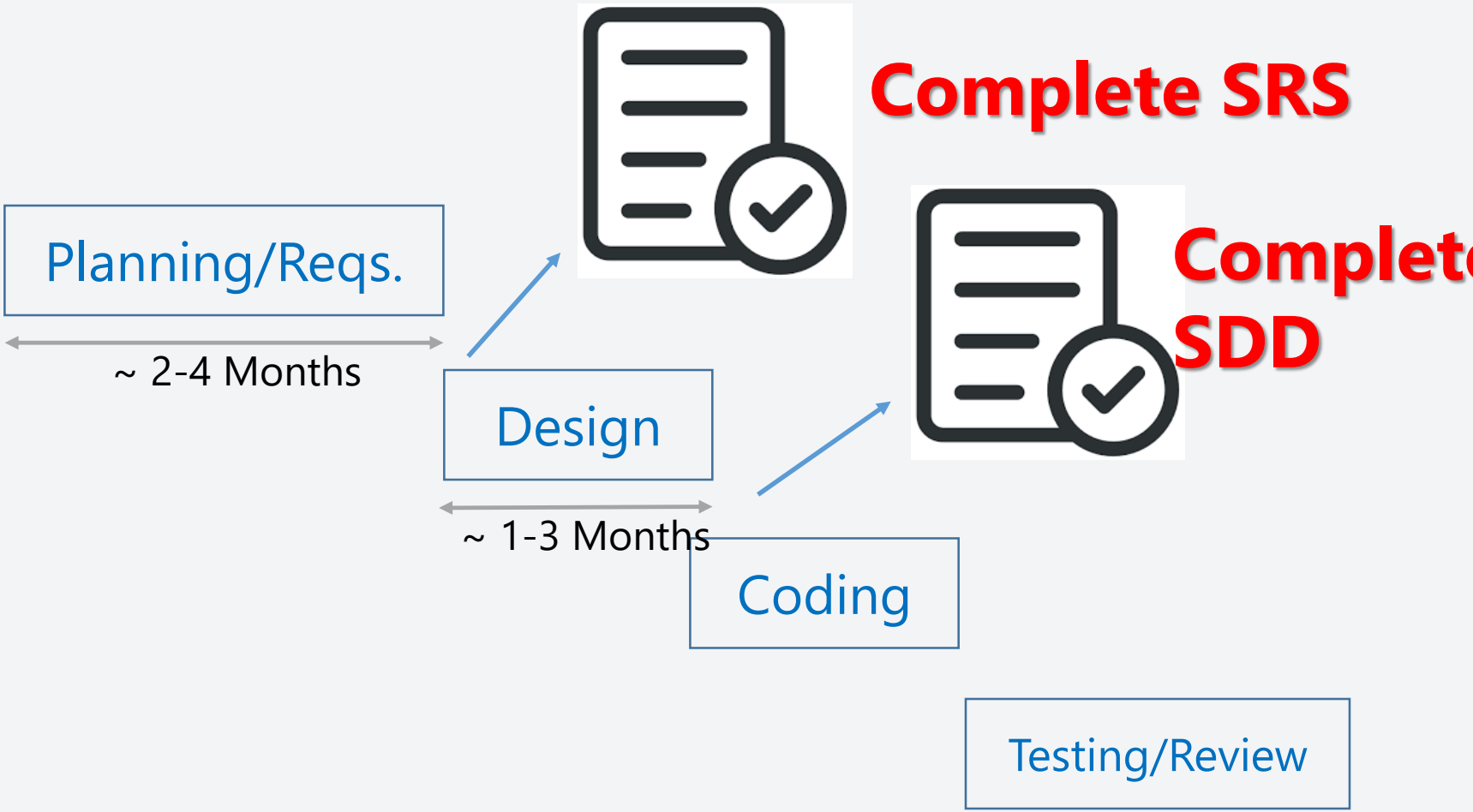
Planning/Reqs.

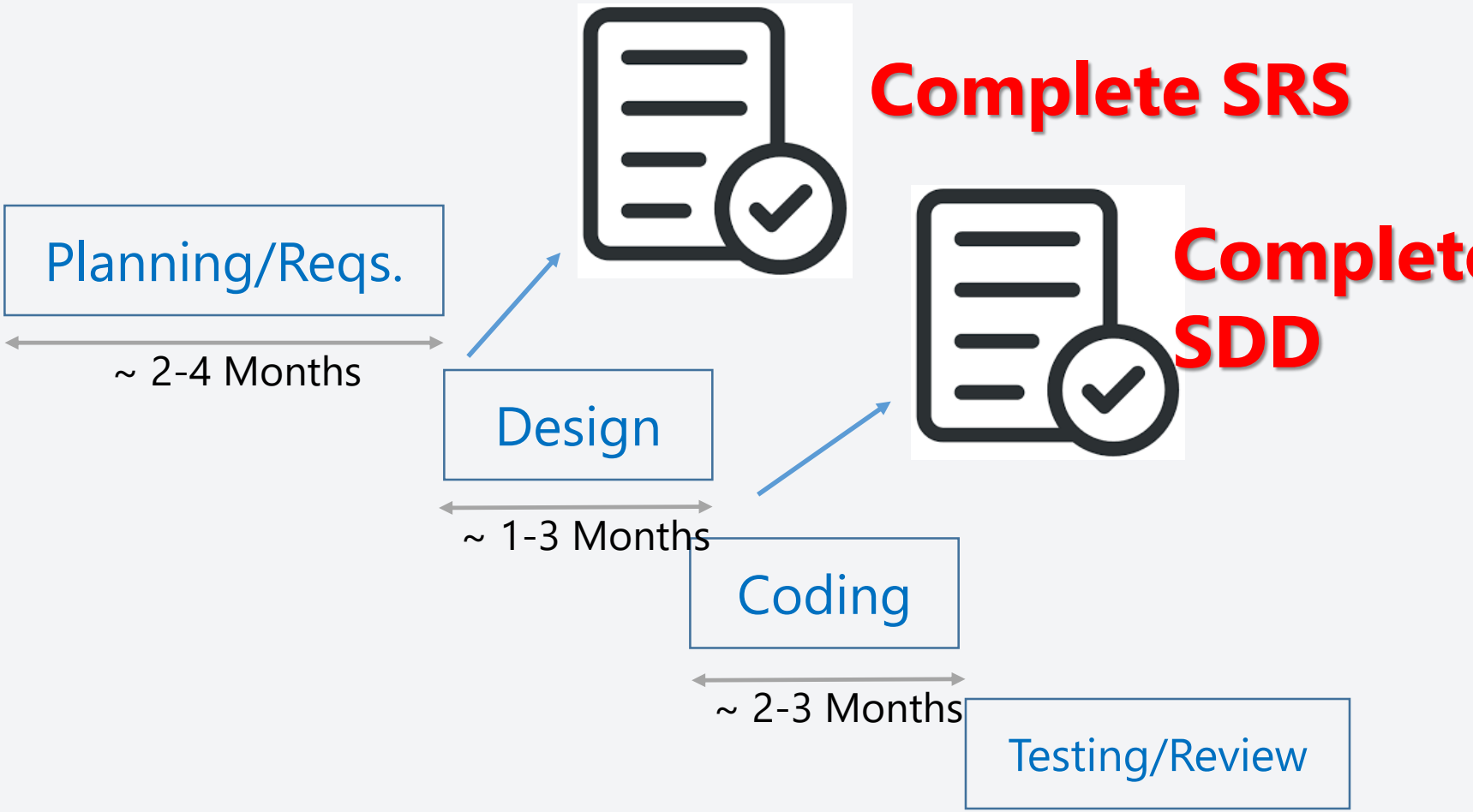
~ 2-4 Months

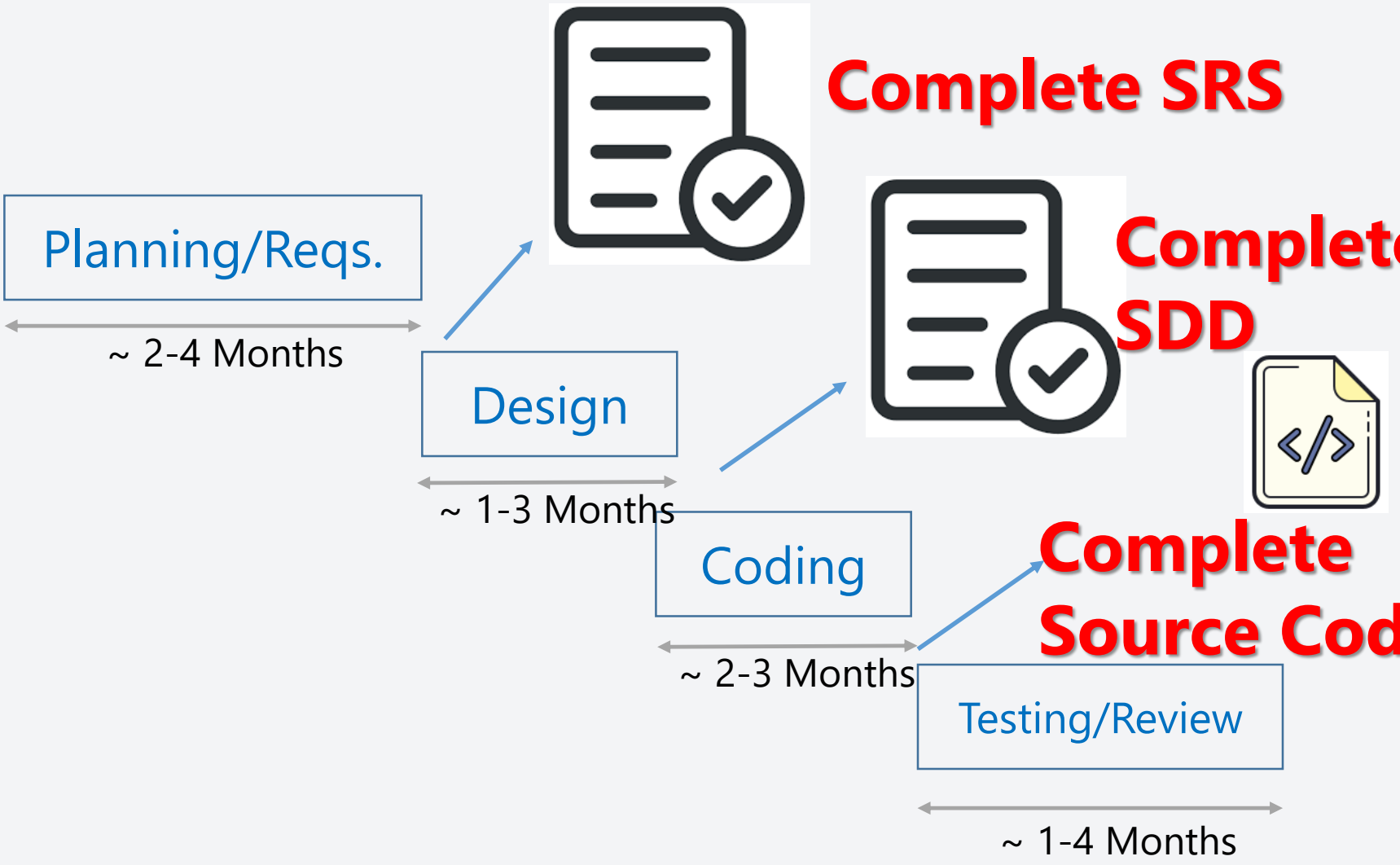
Design

Coding

Testing/Review


Complete SRS
Complete SDD
Planning/Reqs.
~ 2-4 Months
Design
~ 1-3 Months
Coding
Testing/Review


Complete SRS
Complete SDD
Planning/Reqs.
~ 2-4 Months
Design
~ 1-3 Months
Coding
~ 2-3 Months
Testing/Review


Complete SRS
Planning/Reqs.
~ 2-4 Months
Design
~ 1-3 Months
Complete
SDD
Coding
~ 2-3 Months
Complete
Source Cod
Testing/Review
~ 1-4 Months

Planned software development approaches, or heavyweight approach

Planning/Reqs.

~ 2-4 Months

Design

~ 1-3 Months

Coding

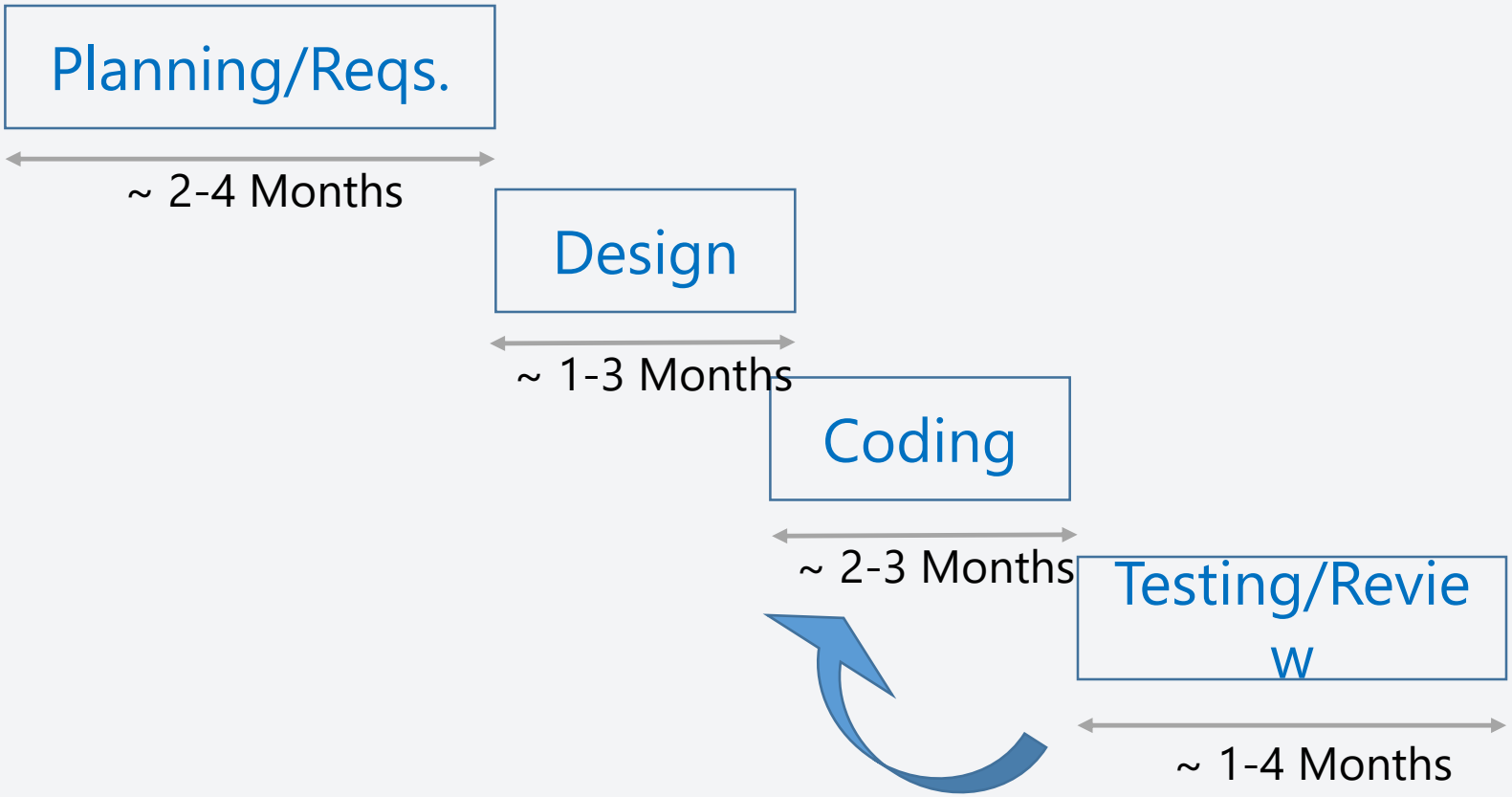
~ 2-3 Months

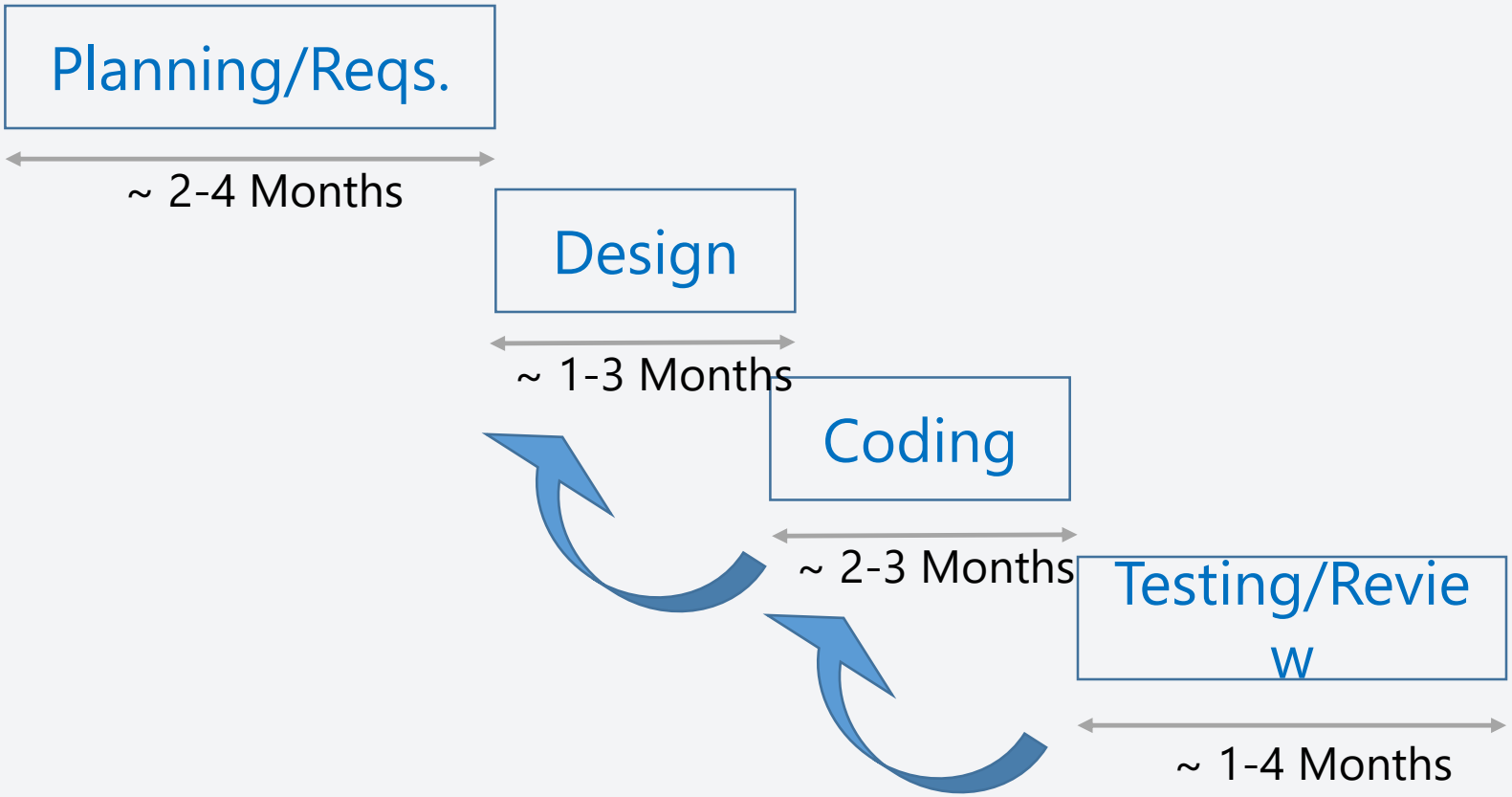
Testing/Review

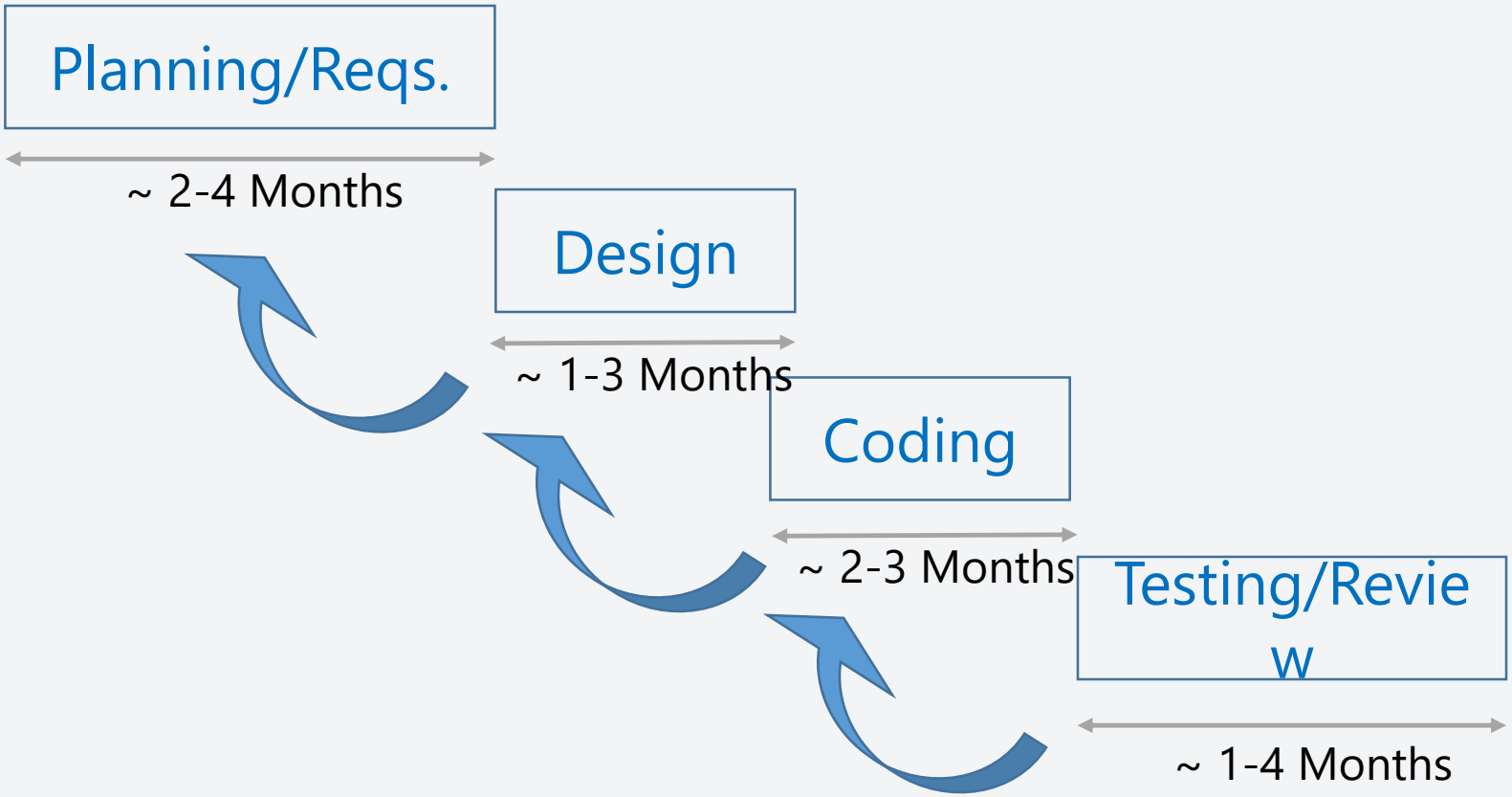
~ 1-4 Months

It is difficult to bring changes to software.

Risky


Planning/Reqs.
~ 2-4 Months
Design
~ 1-3 Months
Coding
~ 2-3 Months
Testing/Review
~ 1-4 Months


Planning/Reqs.
~ 2-4 Months
Design
~ 1-3 Months
Coding
~ 2-3 Months
Testing/Review
~ 1-4 Months


Planning/Reqs.
~ 2-4 Months
Design
~ 1-3 Months
Coding
~ 2-3 Months
Testing/Review
~ 1-4 Months

Why Agile?

- Businesses now operate in a global, rapidly changing environment. They have to respond to new opportunities and markets, changing economic conditions and the emergence of competing products and services. Software is part of almost all business operations, so new software has to be developed quickly to take advantage of new opportunities and to respond to competitive pressure. Rapid software development and delivery is therefore the most critical requirement for most business systems. In fact, businesses may be willing to trade off software quality and compromise on requirements if they can deploy essential new software quickly.
- Because these businesses are operating in a changing environment, it is practically impossible to derive a complete set of stable software requirements. Requirements change because customers find it impossible to predict how a system will affect working practices, how it will interact with other systems, and what user operations should be automated. It may only be after a system has been delivered and users gain experience with it that the real requirements become clear. Even then, external factors drive requirements change.

Source: Software Engineering, I. Sommerville, 10th Ed. Pearson, 2016.

Why Agile?

- Plan-driven software development processes that completely specify the requirements and then design, build, and test a system are not geared to rapid software development. As the requirements change or as requirements problems are discovered, the system design or implementation has to be reworked and retested. As a consequence, a conventional waterfall or specification-based process is usually a lengthy one, and the final software is delivered to the customer long after it was originally specified.
- For some types of software, such as safety-critical control systems, where a complete analysis of the system is essential, this plan-driven approach is the right one. However, in a fast-moving business environment, it can cause real problems. By the time the software is available for use, the original reason for its procurement may have changed so radically that the software is effectively useless. Therefore, for business systems in particular, development processes that focus on rapid software development and delivery are essential.

Source: Software Engineering, I. Sommerville, 10th Ed. Pearson, 2016.

Why Agile?

- The need for rapid software development and processes that can handle changing requirements has been recognized for many years (Larman and Basili 2003).
- However, faster software development really took off in the late 1990s with the development of the idea of "agile methods" such as Extreme Programming (Beck 1999), Scrum (Schwaber and Beedle 2001), and DSDM (Stapleton 2003).
- Rapid software development became known as agile development or agile methods. These agile methods are designed to produce useful software quickly.

Source: Software Engineering, I. Sommerville, 10th Ed. Pearson, 2016.

Why Agile?

- All of the agile methods that have been proposed share a number of common characteristics:

Source: Software Engineering, I. Sommerville, 10th Ed. Pearson, 2016.

Why Agile?

- All of the agile methods that have been proposed share a number of common characteristics:

> The processes of specification, design and implementation are interleaved. There is no detailed system specification, and design documentation is minimized or generated automatically by the programming environment used to implement the system. The user requirements document is an outline definition of the most important characteristics of the system.

Source: Software Engineering, I. Sommerville, 10th Ed. Pearson, 2016.

Why Agile?

- All of the agile methods that have been proposed share a number of common characteristics:

> The system is developed in a series of increments. End-users and other system stakeholders are involved in specifying and evaluating each increment. They may propose changes to the software and new requirements that should be implemented in a later version of the system.

Source: Software Engineering, I. Sommerville, 10th Ed. Pearson, 2016.

Why Agile?

- All of the agile methods that have been proposed share a number of common characteristics:

> Extensive tool support is used to support the development process. Tools that may be used include automated testing tools, tools to support configuration management, and system integration and tools to automate user interface production.

Source: Software Engineering, I. Sommerville, 10th Ed. Pearson, 2016.

Agile (Çevik) Lifecycle With Short Cycles

- The ***agile software development*** process, proposed in 2001 by Agile Alliance.
- In the Manifesto of Agile Alliance, the spirit of the agile development is captured in four recommendations:
    1. Individuals and interactions over processes and tools
    2. Working software over comprehensive documentation
    3. Customer collaboration over contract negotiation
    4. Responding to change over following a plan

Source: Practical Software Engineering: A Case Study Approach, L. Maciaszek, B. Lee Liong, S. Bills, Pearson/Addison-Wesley, 2005.

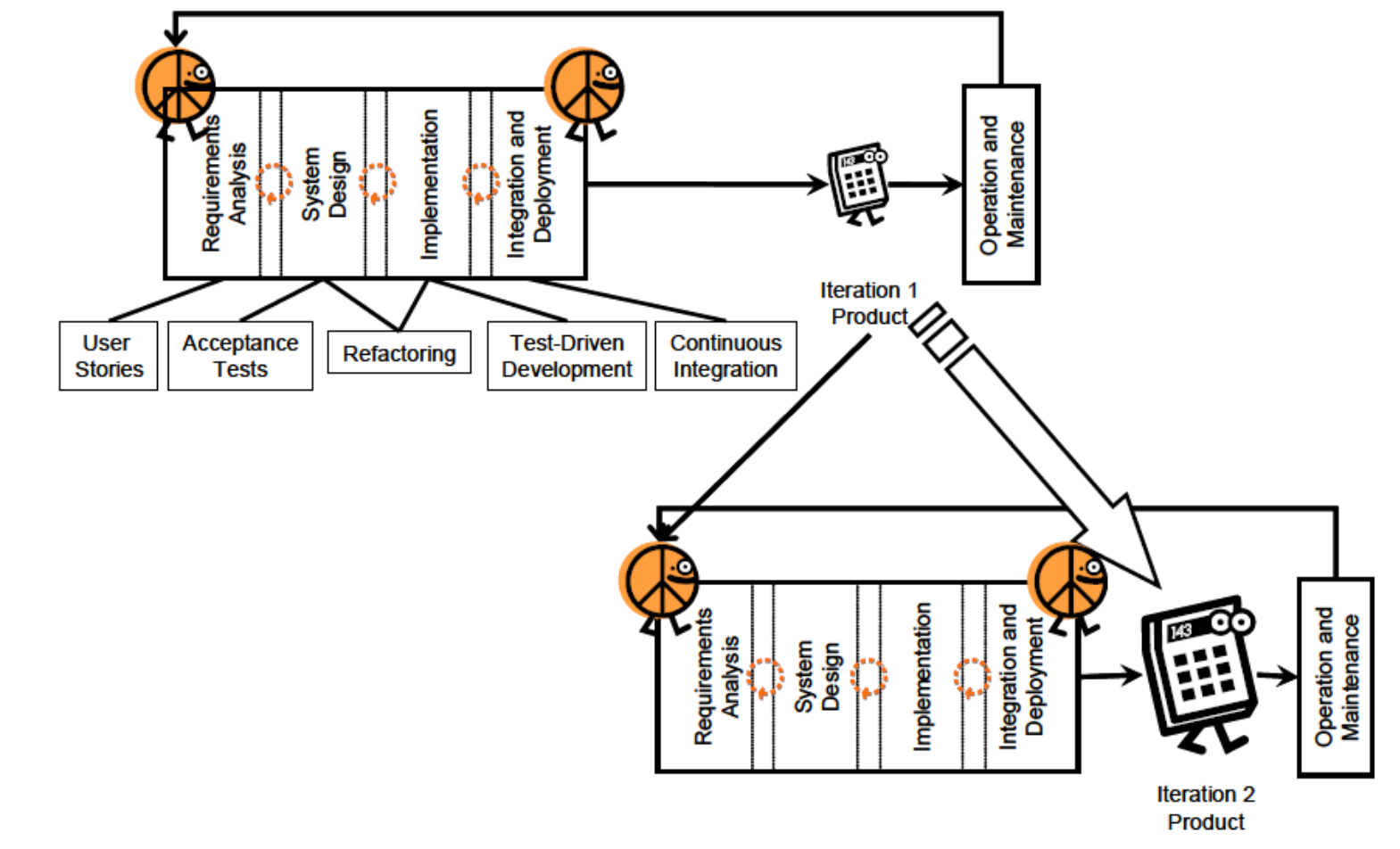

Agile Manifesto
February 11-13, 2001
Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas
In Commemoration of The Manifesto
for Agile Software Development
("Agile
esto") cre d in the
Aspen Ro The Lodge at Snowbird
February -13, 2001. Agile values and
principles continue to influence
millions worldwide.

Agile Lifecycle With Short Cycles

- "great software comes from great people", everything else is secondary.
  - "People" includes ALL project stakeholders – developers and customers.
- Despite all these "revolutionary" propositions, agile development sits well among other iterative lifecycles.

Source: Practical Software Engineering: A Case Study Approach, L. Maciaszek, B. Lee Liong, S. Bills, Pearson/Addison-Wesley, 2005.

Agile Lifecycle With Short Cycles



Source: Practical Software Engineering: A Case Study Approach, L. Maciaszek, B. Lee Liong, S. Bills, Pearson/Addison-Wesley, 2005.

Agile Lifecycle With Short Cycles

- "Conventional" requirements analysis is replaced in agile development by *user stories* – features that the customer would like the system to support.
- Acceptance tests, refactoring, and test-driven development.
  - *Acceptance tests* are programs that an application program must pass to be accepted by customers. This process is called ***test-driven development*** or ***intentional programming*** – the developer programs his/her intent in an acceptance test before s/he implements it.
  - The whole approach is facilitated by frequent ***refactorings*** – improvements to the structure of the system without changing its behavior.

Source: Practical Software Engineering: A Case Study Approach, L. Maciaszek, B. Lee Liong, S. Bills, Pearson/Addison-Wesley, 2005.

Agile Lifecycle With Short Cycles

- Agile development encourages other practices, such as ***pair programming*** and *collective ownership*.
  - All programming is done by pairs of programmers – two programmers working together at a single workstation.
- "Conventional" integration and deployment is replaced in agile development by *continuous integration* and *short cycles*.

Source: Practical Software Engineering: A Case Study Approach, L. Maciaszek, B. Lee Liong, S. Bills, Pearson/Addison-Wesley, 2005.

Agile Lifecycle With Short Cycles

- Agile development does NOT mean lack of planning. In fact, the deployment dates are carefully planned.
- Each iteration is normally planned to complete in <u>short cycles of two-week duration</u>.
  - The product at the end of two-week cycle is a minor delivery for customer evaluation.
  - A major delivery, a product put into production, is a result of about six two-week cycles.

Source: Practical Software Engineering: A Case Study Approach, L. Maciaszek, B. Lee Liong, S. Bills, Pearson/Addison-Wesley, 2005.

Agile Development

- All agile development methodologies are based on the agile manifesto and a set of 12 principles.
- The emphasis of the manifesto is to focus the developers on
  - the working conditions of the developers
  - the working software
  - the customers, and addressing changing requirements instead of focusing on detailed systems development processes, tools, all-inclusive documentation, legal contracts, and detailed plans.
- These programming-centric methodologies have few rules and practices, all of which are fairly easy to follow.

Source: System Analysis & Design, An Object-Oriented Approach with UML 5th Edt, A. Dennis, B. H. Wixom, D. Tegarden, 2015.

12 principles of Agile Development

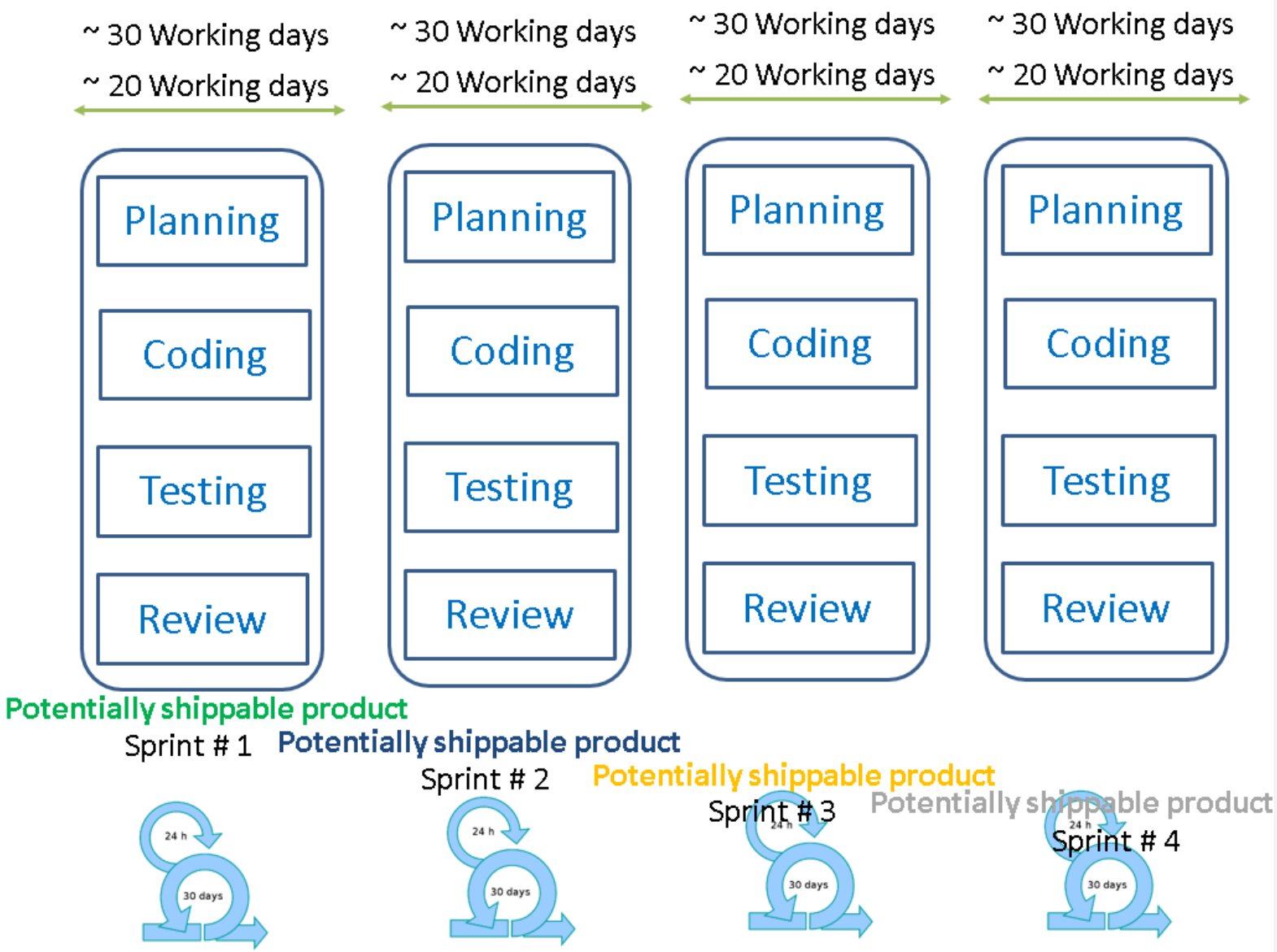
1. Software is delivered early and continuously through the development process, satisfying the customer.
2. Changing requirements are embraced regardless of when they occur in the development process.
3. Working software is delivered frequently to the customer.
4. Customers and developers work together to solve the business problem.
5. Motivated individuals create solutions; provide them the tools and environment they need, and trust them to deliver.
6. Face-to-face communication within the development team is the most efficient and effective method of gathering requirements.

Source: System Analysis & Design, An Object-Oriented Approach with UML 5th Edt, A. Dennis, B. H. Wixom, D. Tegarden, 2015.

12 principles of Agile Development

7. The primary measure of progress is working, executing software.
8. Both customers and developers should work at a pace that is sustainable. That is, the level of work could be maintained indefinitely without any worker burnout.
9. Agility is heightened through attention to both technical excellence and good design.
10. Simplicity, the avoidance of unnecessary work, is essential.
11. Self-organizing teams develop the best architectures, requirements, and designs.
12. Development teams regularly reflect on how to improve their development processes.

Source: System Analysis & Design, An Object-Oriented Approach with UML 5th Edt, A. Dennis, B. H. Wixom, D. Tegarden, 2015.


~ 30 Working days
~ 20 Working days
~ 30 Working days
~ 20 Working days
~ 30 Working days
~ 20 Working days
~ 30 Working days
~ 20 Working days
Planning
Coding
Testing
Review
Planning
Coding
Testing
Review
Planning
Coding
Testing
Review
Planning
Coding
Testing
Review
Potentially shippable product
Sprint # 1
Potentially shippable product
Sprint # 2
Potentially shippable product
Sprint # 3
Potentially shippable product
Sprint # 4
24 h
30 days
24 h
30 days
24 h
30 days
24 h
30 days

Scrum

Scrum

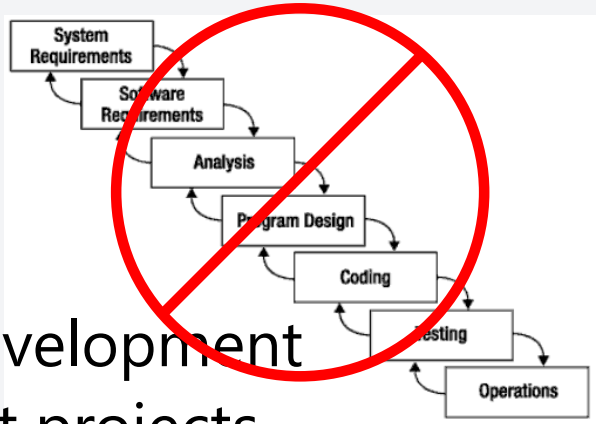
- *Scrum* is a term that is well known to rugby fans.
- In rugby, a scrum is used to restart a game.
- "*No matter how much you plan, as soon as the software begins to be developed, chaos breaks out and the plans go out the window*"
  - The best you can do is to react to where the proverbial rugby ball squirts out. You then sprint with the ball until the next scrum.
- In the case of the Scrum methodology, a sprint lasts 30 working days.
  - At the end of the sprint, a system is delivered to the customer.

Source: System Analysis & Design, An Object-Oriented Approach with UML 5th Edt, A. Dennis, B. H. Wixom, D. Tegarden, 2015.

What is Scrum?

It's about common sense

- **Scrum**:
  - Is an agile, lightweight process
  - Can manage and control software and product development
  - Uses iterative, incremental practices
  - Has a simple implementation
  - Increases productivity
  - Reduces time to benefits
  - Embraces adaptive, empirical systems development
  - Is not restricted to software development projects
  - Embraces the opposite of the waterfall approach…

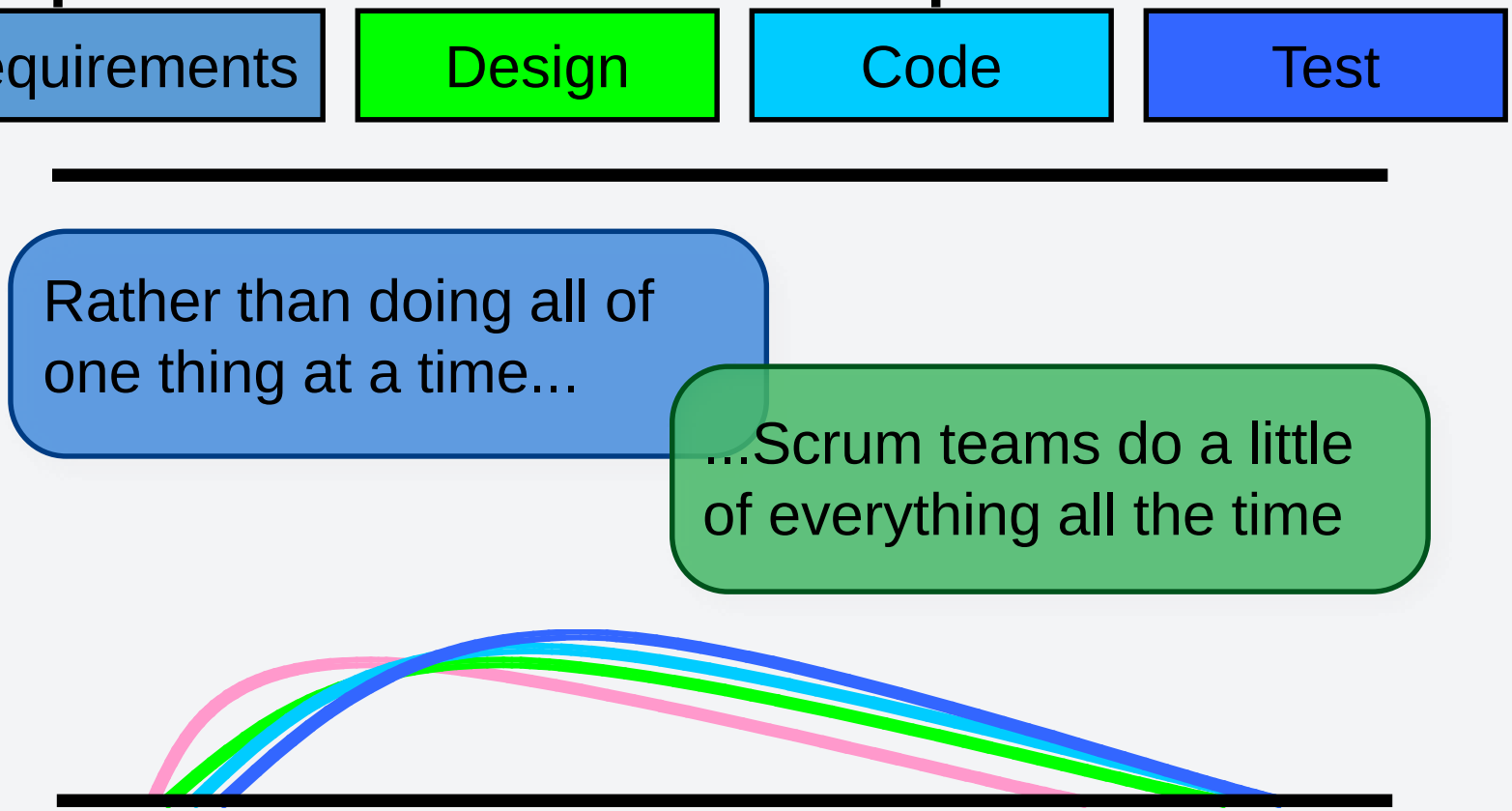


Agile Manifesto

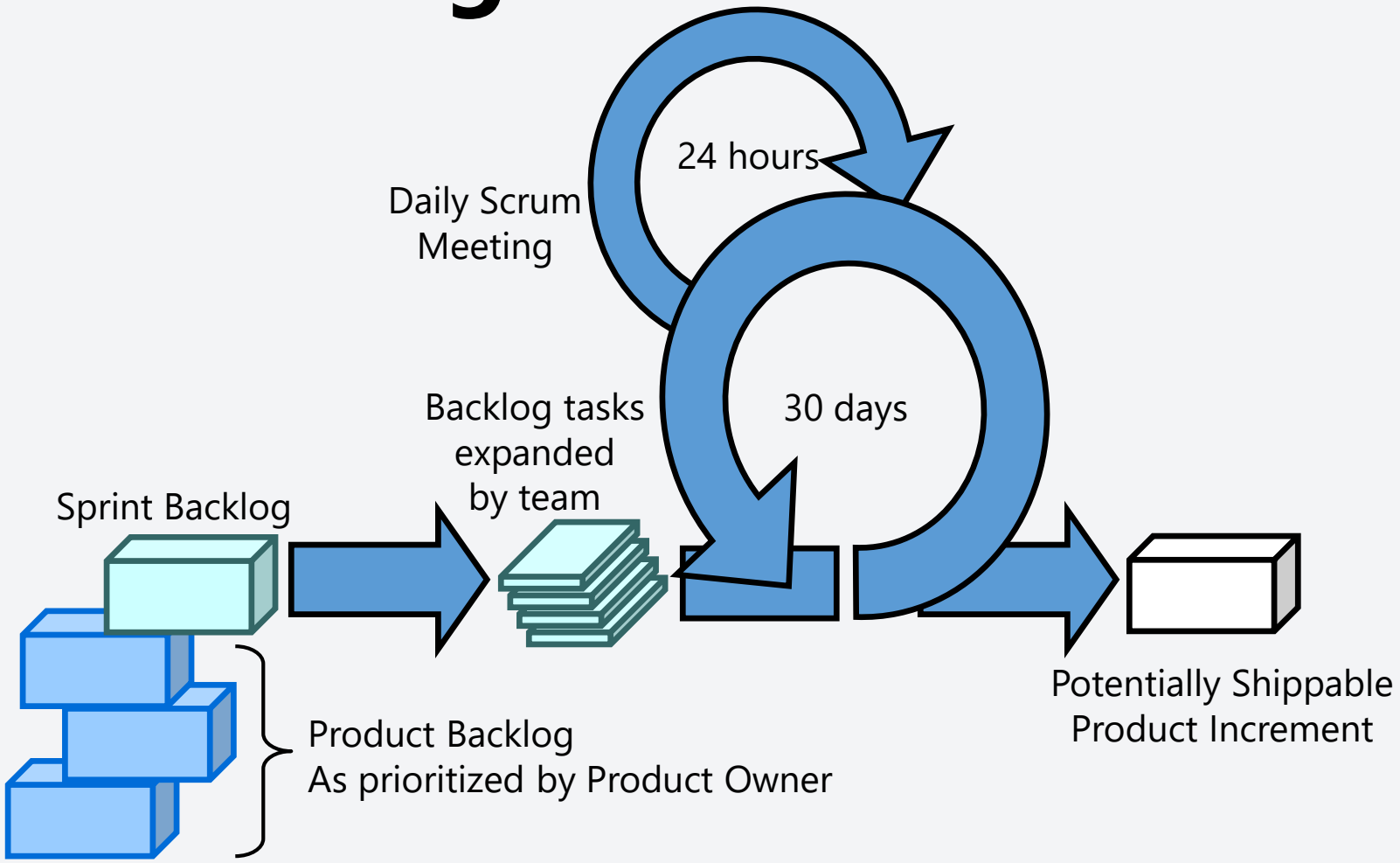
Individuals and interactions	over	Process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

Source: www.agilemanifesto.org

Sequential vs. Overlap

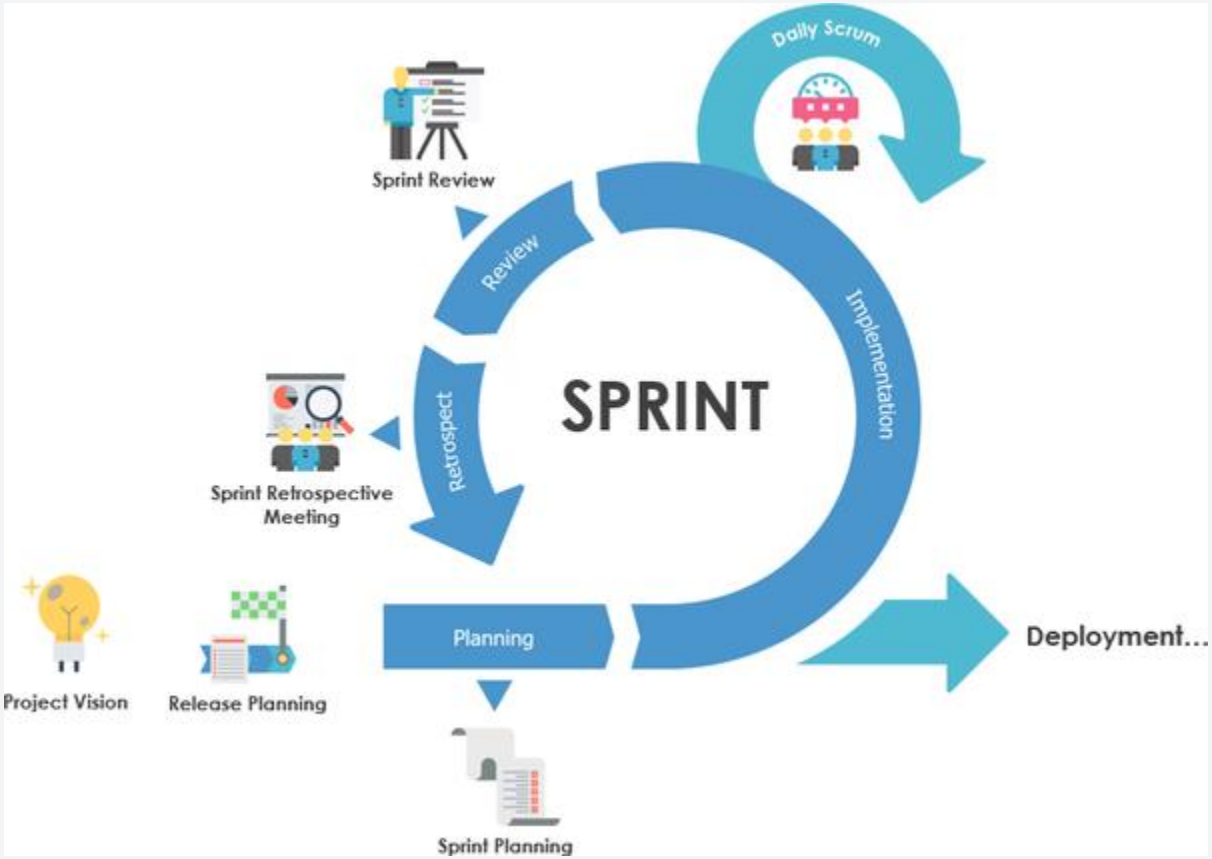


Scrum at a glance

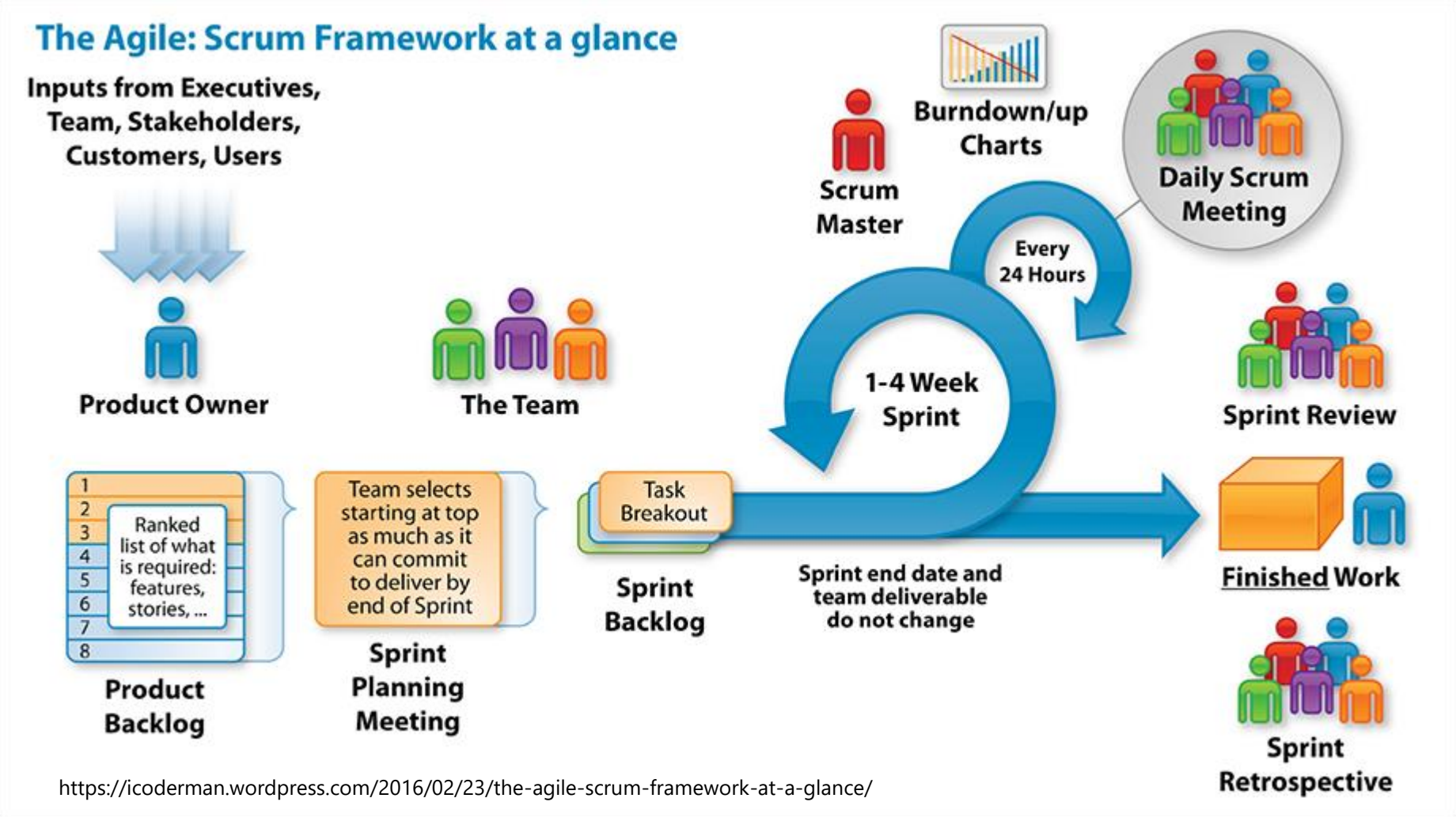


Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

Scrum at a glance



Scrum at a glance



https://icoderman.wordpress.com/2016/02/23/the-agile-scrum-framework-at-a-glance/

Scrum at a glance

3 Artifacts

A) Product Backlog

B) Sprint Backlog

C)Burndown charts

3 Roles

A) Product Owner

B) Scrum Master

C) Team

3 Ceremonies

A) Sprint Planning

B) Daily Scrum Meeting

C) Sprint Review

Scrum Framework

Roles

- Product owner
- Scrum Master
- Team

Ceremonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

Scrum at a glance

3 Artifacts

A) Product Backlog

B) Sprint Backlog

C) Burndown charts

- User stories are defined by Product owner
- Product owner prioritized the list of user stories.
- User stories are evolved and changed every sprint

Scrum at a glance

3 Artifacts

A) Product Backlog

B) Sprint Backlog

C) Burndown charts

- Is composed of highest priority user stories
- Team estimates the sizes of user stories

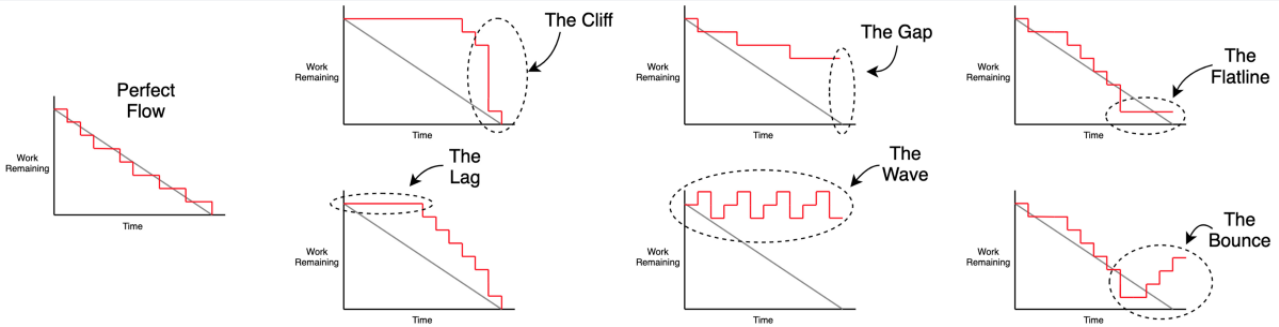
Scrum at a glance

3 Artifacts

A) Product Backlog

B) Sprint Backlog

C) Burndown charts

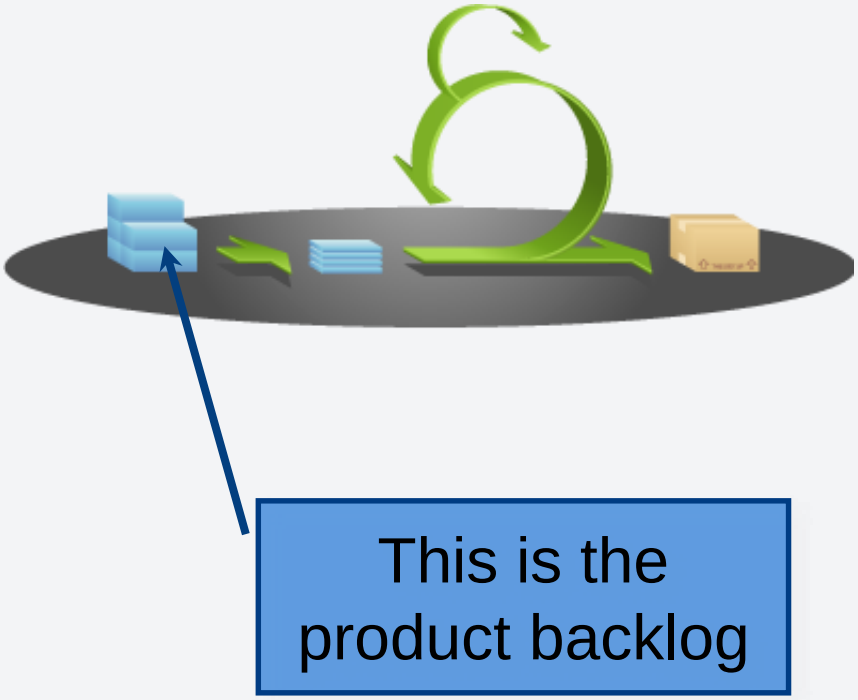


- Should approach to zero point as the work is being done.

Scrum's Artifacts

- Scrum has remarkably few artifacts
  - Product Backlog
  - Sprint Backlog
  - Burndown Charts
- Can be managed using just an Excel spreadsheet
  - More advanced / complicated tools exist:
    - Expensive
    - Web-based – no good for Scrum Master/project manager who travels
    - Still under development

Product Backlog



- The requirements
- A list of all desired work on project
- Ideally expressed as a list of user stories along with "story points", such that each item has value to users or customers of the product
- Prioritized by the product owner
- Reprioritized at start of each sprint

User Stories

- Instead of UCs, Agile project owners do "user stories"
  - **Who** (user role) – Is this a customer, employee, admin, etc.?
  - **What** (goal) – What functionality must be achieved/developed?
  - **Why** (reason) – Why does user want to accomplish this goal?

As a [user role], I want to [goal], so I can [reason].

- **Ex:** "As a user, I want to log in, so I can access subscriber content."
  - **Story points**: Rating of effort needed to implement this story
    - common scales:
      - 1-10,
      - shirt sizes (XS, S, M, L, XL),
      - 1, 2, 3, 5, 8, 13, 21, 34
      - etc.

Sample Product Backlog

Backlog item	Estimate
Allow a guest to make a reservation	3 (story points)
As a guest, I want to cancel a reservation.	5
As a guest, I want to change the dates of a reservation.	3
As a hotel employee, I can run RevPAR reports (revenue-per-available-room)	8
Improve exception handling	8
…	30
…	50

Sample Product Backlog 2

Product Backlog
Estimating System Upgrade

Sprint	ID		Backlog Item	Owner	Estimate (days)	Remaining (days)
1	1	Minor	Remove user kludge in .dpr file	BC	1	1
1	2	Minor	Remove cMap/cMenu/cMenuSize from disciplines.pas	BC	1	1
1	3	Minor	Create "Legacy" discipline node with old civils and E&I content	BC	1	1
1	4	Major	Augment each tbl operation to support network operation	BC	10	10
1	5	Major	Extend Engineering Design estimate items to include summaries	BC	2	2
1	6	Super	Supervision/Guidance	CAM	4	4
	7	Minor	Remove Custodian property from AppConfig class in globals.pas	BC	1	
	8	Minor	Remove LOC_ constants in globals.pas and main.pas	BC	1	
	9	Minor	New E&I section doesn't have lblCaption set	BC	1	
	10	Minor	Delay in main.releaseform doesn't appear to be required	BC	1	
	11	Minor	Undo modifications to Other Major Equipment in formExcel.pas	BC	1	
	12	Minor	AJACS form to be centred on the screen	BC	1	
	13	Major	Extend DUnit tests to all 40 disciplines	BC	6	

Sprint Backlog

- Individuals sign up for work of their own choosing
  - Work is never assigned
- Estimated work remaining is updated daily
- Any team member can add, delete change sprint backlog
- Work for the sprint emerges
- If work is unclear, define a sprint backlog item with a larger amount of time and break it down later
- Update work remaining as more becomes known

Sample Sprint backlog

Tasks	Mon	Tue	Wed	Thu	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	4	
Test the middle tier	8	16	16	11	8
Write online help	12				
Write the Foo class	8	8	8	8	8
Add error logging			8	4	

Sample Sprint Backlog

Sprint 1

01/11/2004

19 days work in this sprint

	Backlog Item	Backlog Item	Owner	Estimate								
				Sprint Day	1	2	3	4	5	6	7
					Mo	Tu	We	Th	Fr	Sa	Su
				Hours remaining	152	152	152	152	152	152	152
1	Minor	Remove user kludge in .dpr file	BC	8	8	8	8	8	8	8	8
2	Minor	Remove cMap/cMenu/cMenuSize from disciplines.pas	BC	8	8	8	8	8	8	8	8
3	Minor	Create "Legacy" discipline node with old civils and E&I content	BC	8	8	8	8	8	8	8	8
4	Major	Augment each tbl operation to support network operation	BC	80	80	80	80	80	80	80	80
5	Major	Extend Engineering Design estimate items to include summaries	BC	16	16	16	16	16	16	16	16
6	Super	Supervision/Guidance	CAM	32	32	32	32	32	32	32	32

Sprint 1

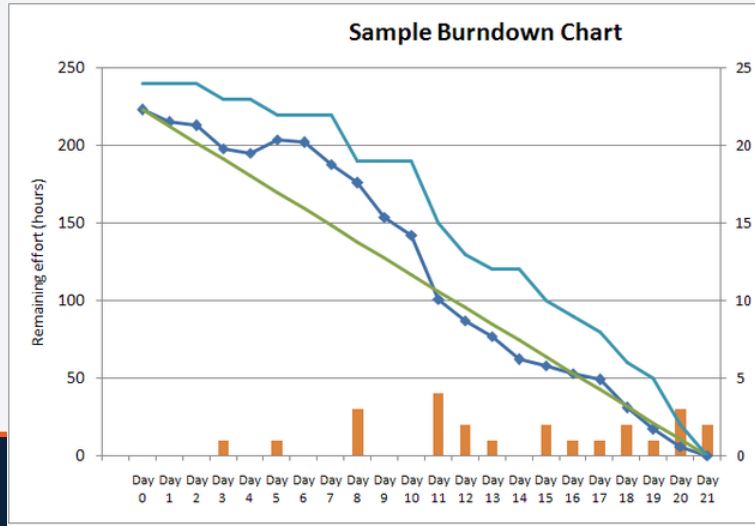
01/11/2004

19 days work in this sprint

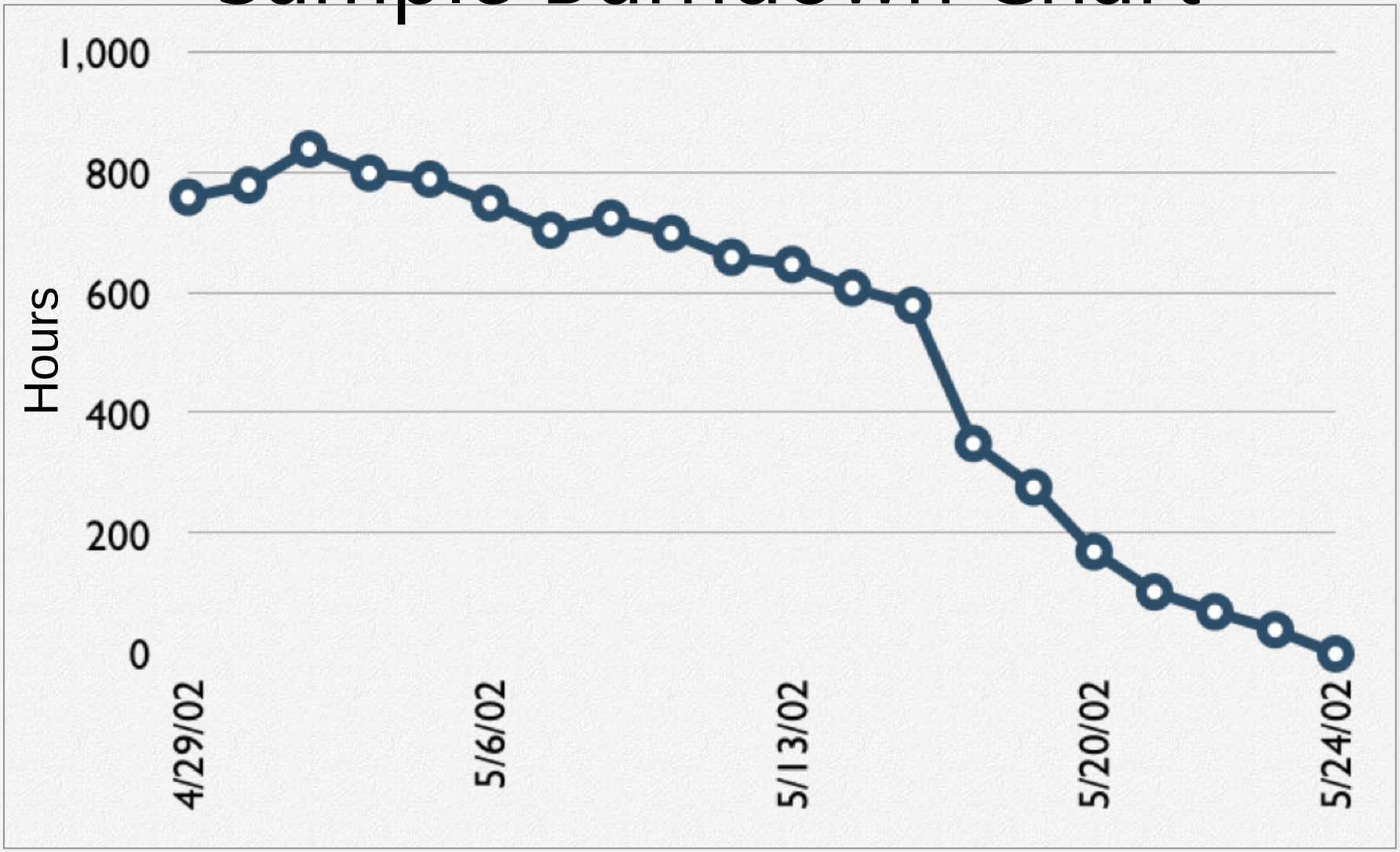
	Backlog Item	Backlog Item	Owner	Estimate								
				Sprint Day	1	2	3	4	5	6	7
					Mo	Tu	We	Th	Fr	Sa	Su
				Hours remaining	152	150	140	130	118	118	118
1	Minor	Remove user kludge in .dpr file	BC	8	8	8	4	2	0		
2	Minor	Remove cMap/cMenu/cMenuSize from disciplines.pas	BC	8	8	8	4	0			
3	Minor	Create "Legacy" discipline node with old civils and E&I content	BC	8	8	8	8	6	0		
4	Major	Augment each tbl operation to support network operation	BC	80	80	80	80	80	78	78	78
5	Major	Extend Engineering Design estimate items to include summaries	BC	16	16	16	16	16	16	16	16
6	Super	Supervision/Guidance	CAM	32	32	30	28	26	24	24	24

Sprint Burndown Chart

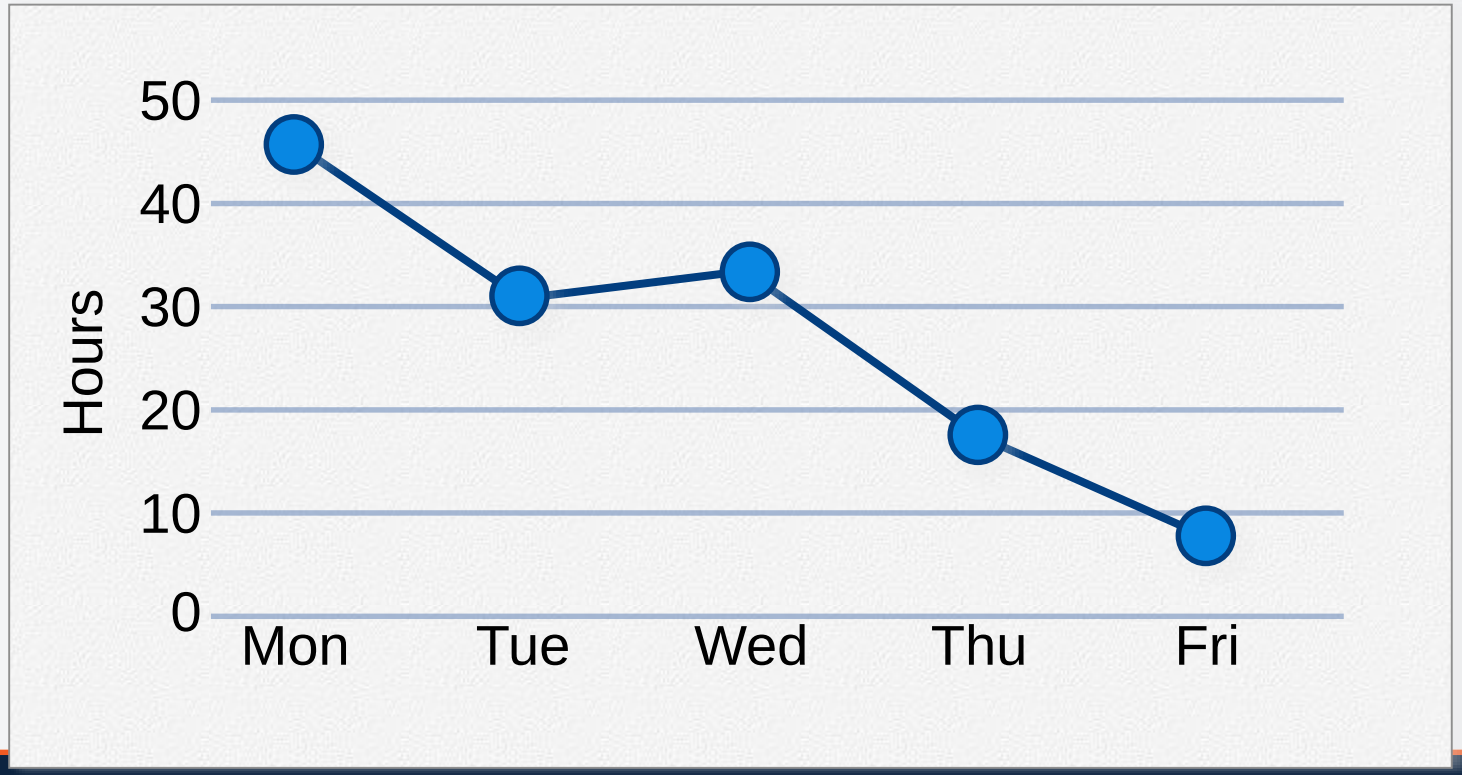
- A display of what work has been completed and what is left to complete
  - one for each developer or work item
  - updated every day
    - make best guess about hours/points completed each day
- *variation:* Release burndown chart
  - shows overall progress
  - updated at end of each sprint



Sample Burndown Chart

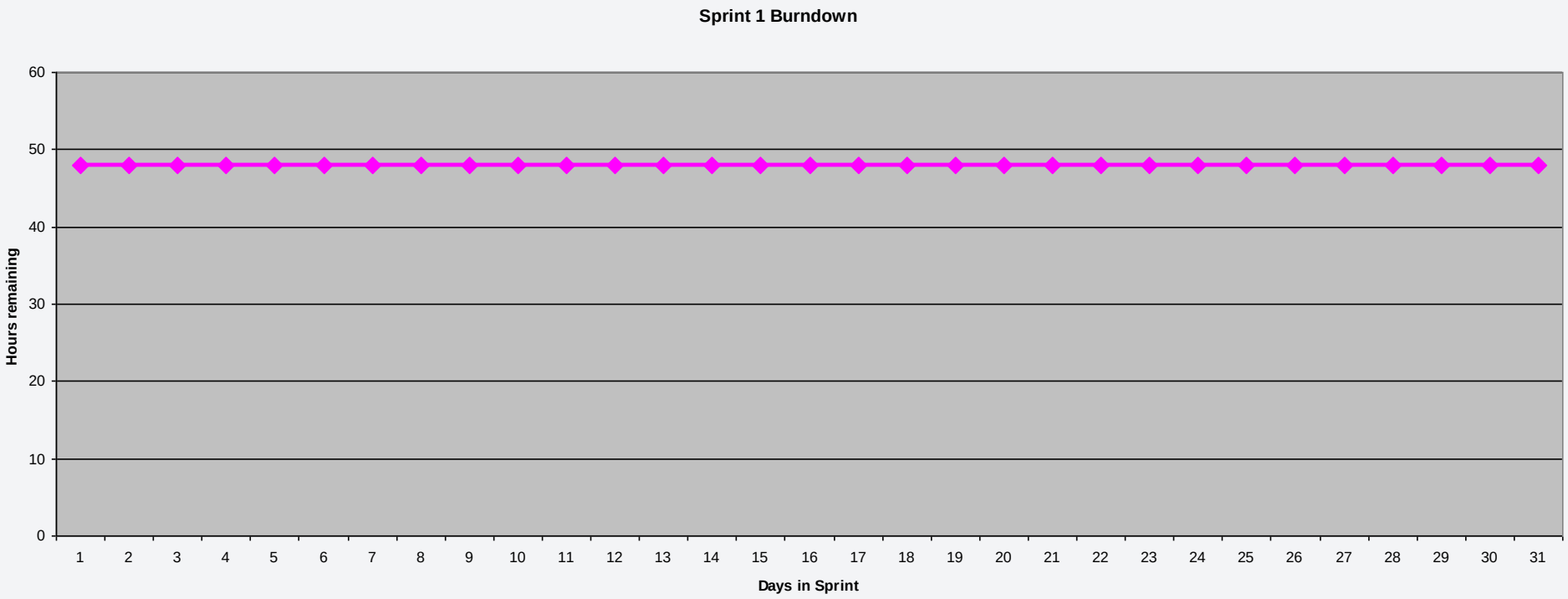


Tasks	Mon	Tue	Wed	Thu	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	7	
Test the middle tier	8	16	16	11	8
Write online help	12				



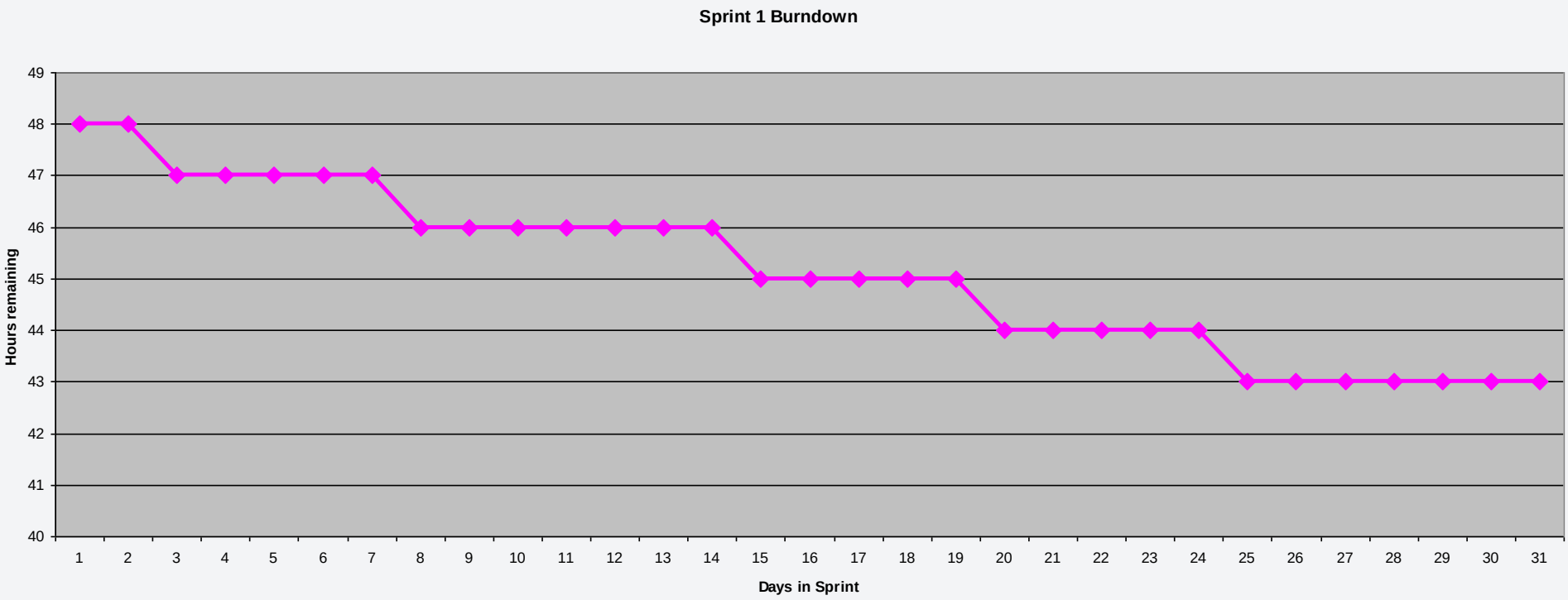
Burndown Example 1

No work being performed



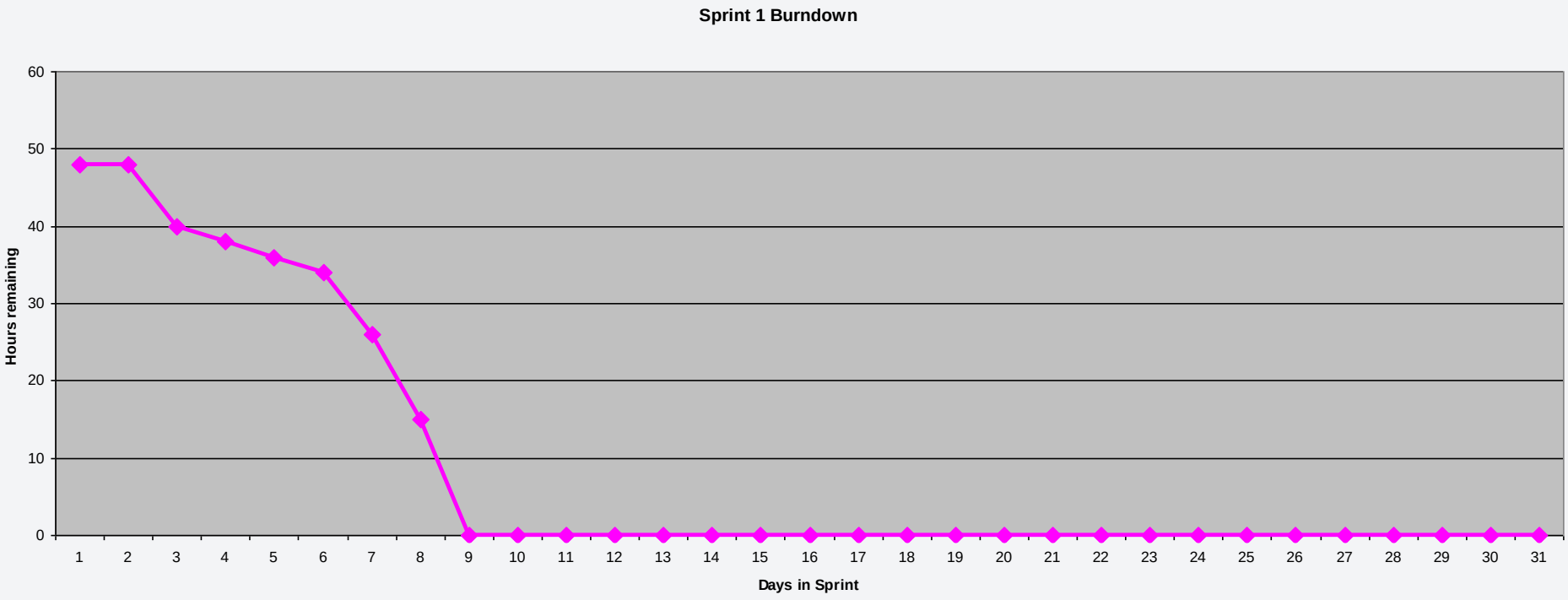
Burndown Example 2

Work being performed, but NOT fast enough



Burndown Example 3

Work being performed, but too fast!



Scrum at a glance

3 Roles

A) Product Owner

B) Scrum Master

C) Team

- Responsible for defining the features that are needed in the product.

Scrum at a glance

3 Roles

A) Product Owner

B) Scrum Master

C) Team

- Is a **servant leader** to the team.
- Protects the team and process

Scrum at a glance

3 Roles

A) Product Owner

B) Scrum Master

C) Team

They may change their roles at different sprints or projects

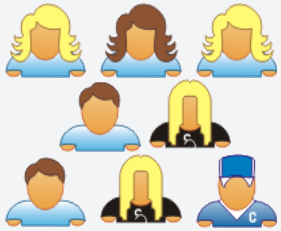
- UI Designer
- Tester
- Coder
- Etc.

Scrum Roles

- Product Owner
  - Possibly a Product Manager or Project Sponsor
  - Decides features, release date, prioritization, $$$

- Scrum Master
  - Typically a Project Manager or Team Leader
  - Responsible for enacting Scrum values and practices
  - Remove impediments / politics, keeps everyone productive

- Project Team
  - 5-10 members; Teams are self-organizing
  - Cross-functional: QA, Programmers, UI Designers, etc.
  - Membership should change only between sprints



Scrum at a glance

- Owner, Master, and Team
- Meet to discuss the users stories
  - Estimate their relative sizes.

3 Ceremonies

A) Sprint Planning

B) Daily Scrum Meeting

C) Sprint Review

Scrum at a glance

- What completed since the previous meeting?
- What are you working on?
- Do we anticipate anything that might blocked the process? Need help?

3 Ceremonies

A) Sprint Planning

B) Daily Scrum Meeting

C) Sprint Review

Scrum at a glance

- Team only → Retrospective: *Identify how to improve teamwork by reflecting on what worked, what didn't, and why.*
- All → Discuss the issues that will improve the process going forward

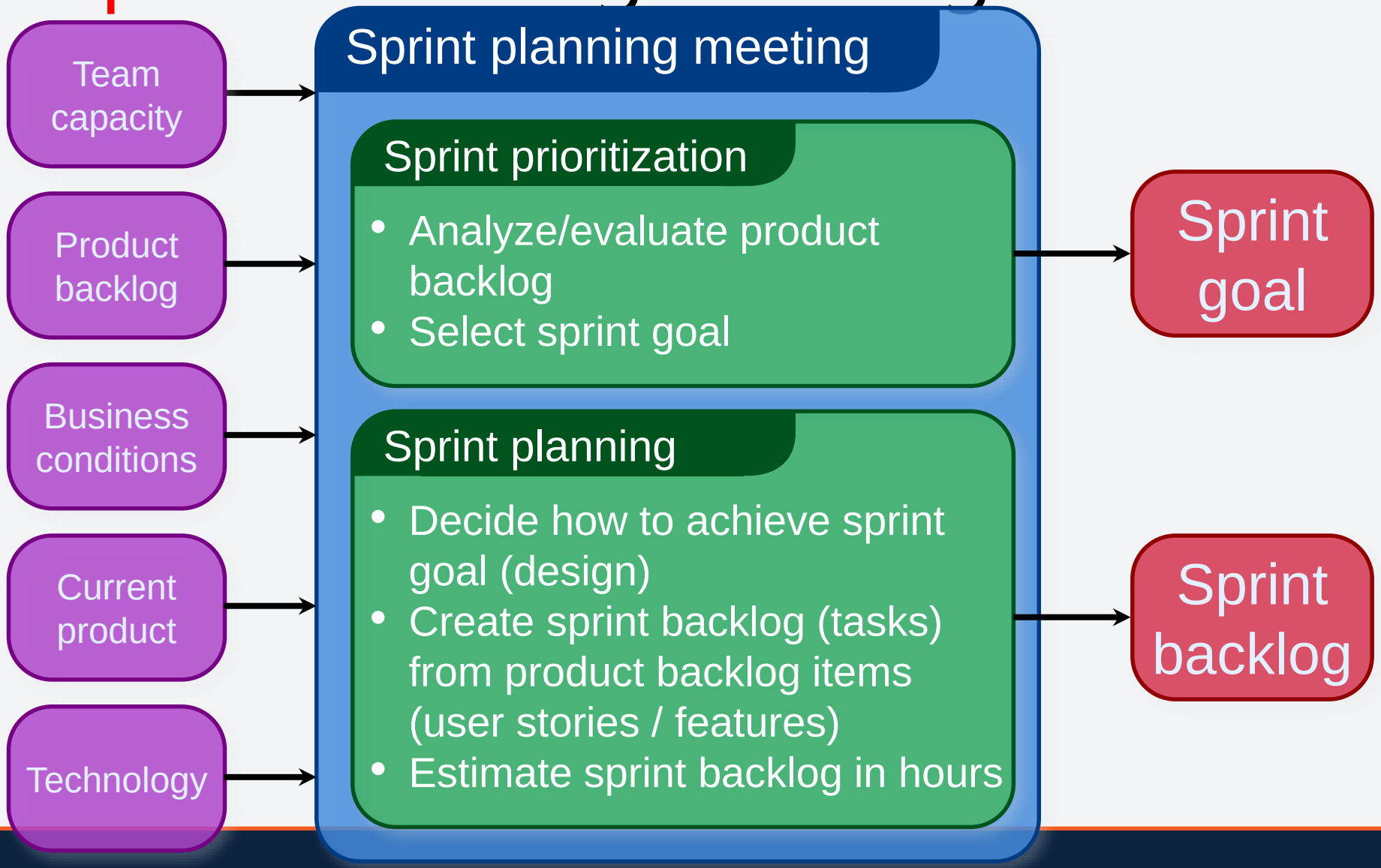
3 Ceremonies

A) Sprint Planning

B) Daily Scrum Meeting

C) Sprint Review

Sprint Planning Meeting

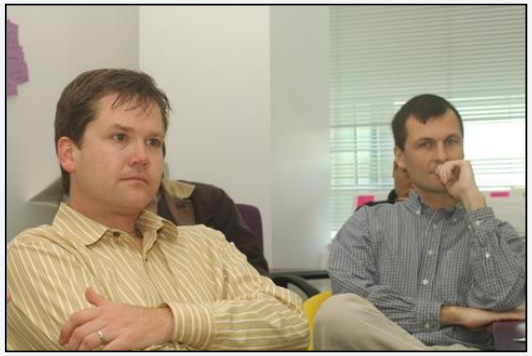


Daily Scrum Meeting

- Parameters
  - Daily, ~15 minutes, Stand-up
  - Anyone late pays a $1 fee
- Not for problem solving
  - Whole world is invited
  - Only team members, Scrum Master, product owner, can talk
  - Helps avoid other unnecessary meetings
- 3 questions answered by each team member:
  1. What did you do yesterday?
  2. What will you do today?
  3. What obstacles are in your way?

The Sprint Review

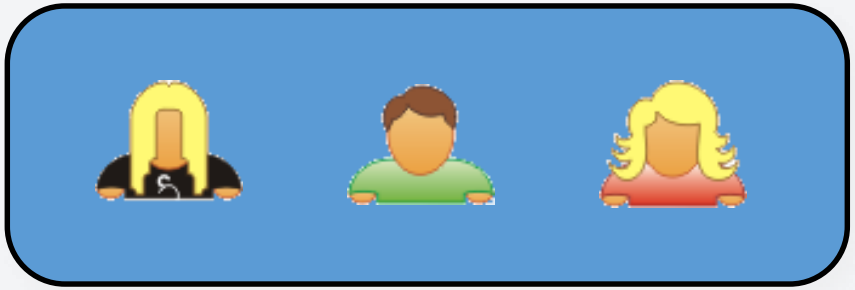
- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features or underlying architecture
- Informal
  - 2-hour prep time rule
  - No slides
- Whole team participates
- Invite the world

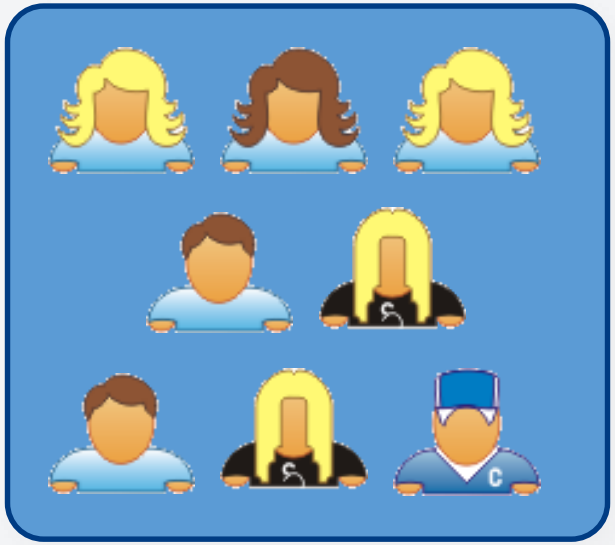


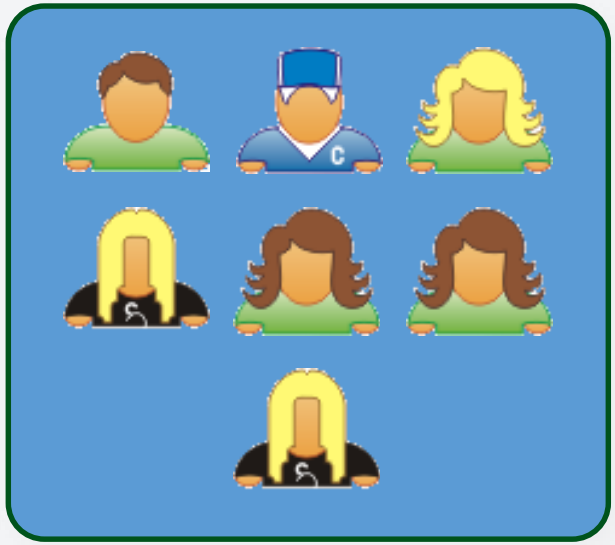
Scalability

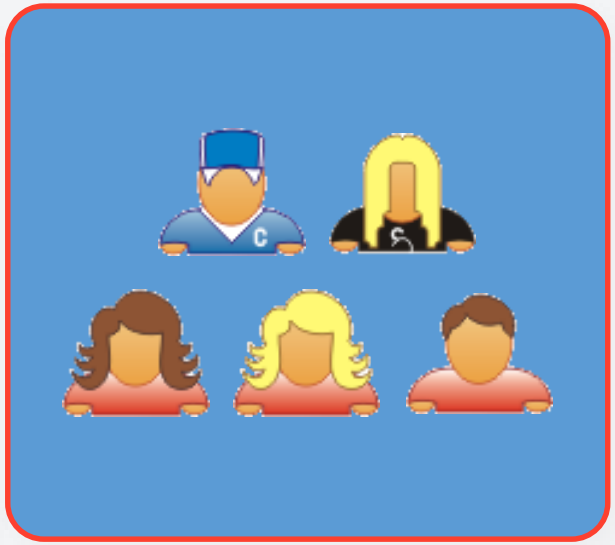
- Typical individual team is 7 ± 2 people
  - Scalability comes from teams of teams
- Factors in scaling
  - Type of application
  - Team size
  - Team dispersion
  - Project duration
- Scrum has been used on multiple 500+ person projects

Scaling: Scrum of Scrums





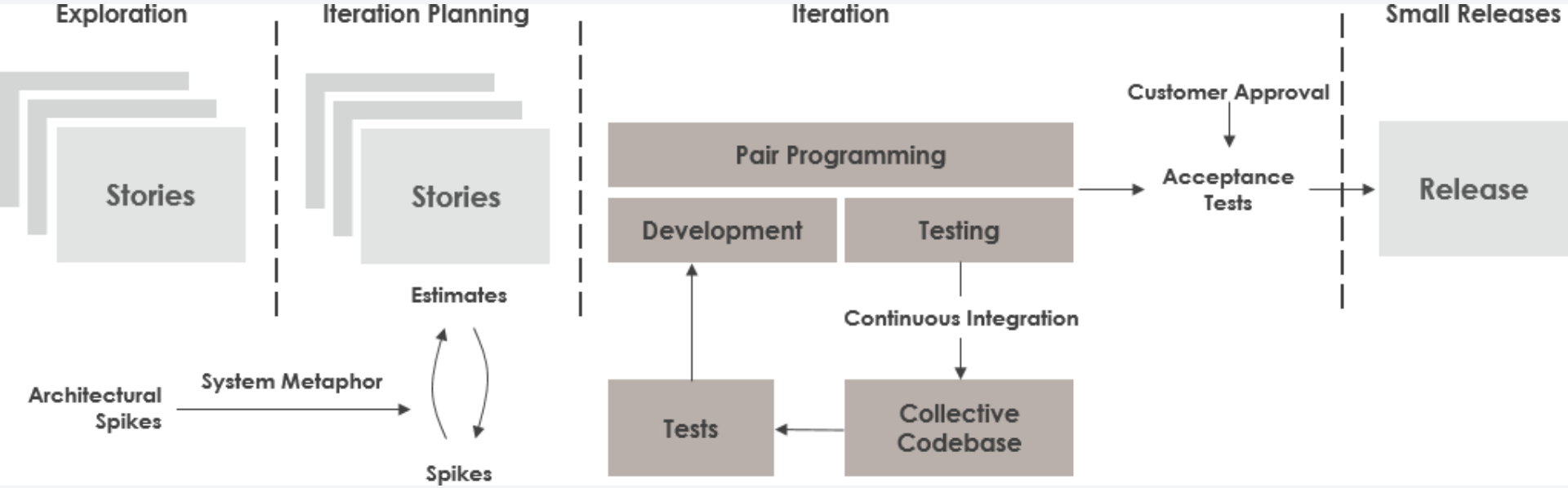




Scrum Dictionary

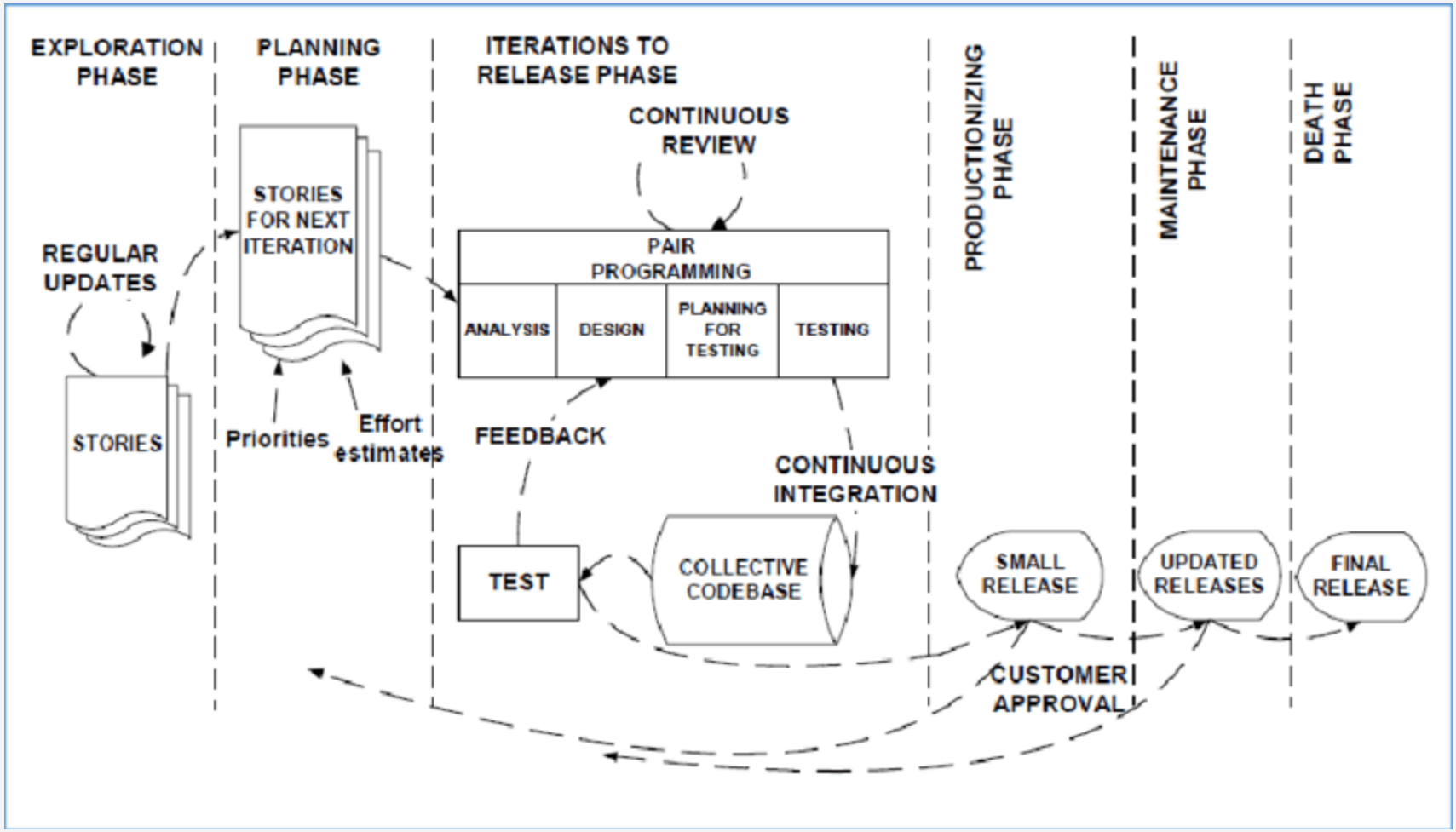
- **scrum**
- 1. iterative project management framework used in agile development, in which a team agrees on development items from a requirements backlog and produces them within a short duration of a few weeks
- [ISO/IEC/IEEE 26515: 2011 Systems and software engineering: Developing user documentation in an agile environment, 4.9]
- **scrum master**
- 1. person who facilitates the scrum process within a team or project
- [ISO/IEC/IEEE 26515: 2011 Systems and software engineering: Developing user documentation in an agile environment, 4.10]
- **scrum meeting**
- 1. brief daily project status meeting or other planning meeting in agile development methodologies
- [ISO/IEC/IEEE 26515: 2011 Systems and software engineering: Developing user documentation in an agile environment, 4.11]
  - Note 1 to entry: The scrum meeting is usually chaired by the scrum master.
- **scrum report**
- 1. report that documents the daily activities of a scrum team, recording any problems or issues to be dealt with
- [ISO/IEC/IEEE 26515: 2011 Systems and software engineering: Developing user documentation in an agile environment, 4.12]

XP at a glance



An architectural spike is a technical risk-reduction technique popularized by Extreme Programming (XP) where you write just enough code to explore the use of a technology or technique that you're unfamiliar with.

XP at a glance



Source: Source: Agile Software Development: Methodologies and Trends July 2020 International Journal of Interactive Mobile Technologies (iJIM) 14(11):246

eXtreme Programming

- Test-driven development (TDD) originated as one of the core XP practices and consists of writing unit tests prior to writing the code to be tested.
- In this way, TDD develops the test cases as a surrogate for a software requirements specification document rather than as an independent check that the software has correctly implemented the requirements.
  - Rather than a testing strategy, TDD is a practice that requires software developers to define and maintain unit tests; it thus can also have a positive impact on elaborating user needs and software requirements specifications.

Source: Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK® 2014

XP by Kent Beck

- *XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements.*

eXtreme Programming

- Get the traditional software development activities to extreme level.
  - Specific planning
  - On-site customer
  - Continuous testing

Source: https://www.youtube.com/watch?v=PRYmsmMdlko

eXtreme Programming

- Simple design
- Pair programming
  - two people simultaneous work together on all production code
- constant testing on going integration
  - integrate the systems several times per day every time a task is completed by a developer
- refactoring
  - practice of restructuring a program or implementing a feature without changing the behavior of the system
- coding standards
- small releases

Source: https://www.youtube.com/watch?v=PRYmsmMdlko

eXtreme Programming

- Release Planning Phase --> Iteration --> Release Phase
- **Release Planning Phase**
  - The Customer writes stories based on the requirements
  - The developer estimates them.
  - The customer chooses the order in which stories will be developed
- **Iteration Phase**
  - The customer writes test and answer questions while the developers develop software according to the stories
  - Iteration Phase provides ready –to-go software.
- **Release Phase**
  - Developers install the software and customer approves the result

Source: https://www.youtube.com/watch?v=PRYmsmMdlko

eXtreme Programming

- XP works best for small to mid-sized teams developing software working in the midst of vague and fast-changing requirements.

Source: https://www.youtube.com/watch?v=PRYmsmMdlko

Scrum vs. XP

- **Similarities**
  - divide the development process into sprints
  - planning meetings
    - before the development starts
      - to pinpoint the user stories
    - @ each sprint as well

Source: https://www.youtube.com/watch?v=PRYmsmMdlko

Scrum vs. XP

- **Differences**
- **primary focus**
  - Scrum → management focus → deals all activities besides coding
    - Not give much technical and engineering emphasis
    - Do NOT mention how the work is actually done & how the product is built
  - XP concentrates on programming & testing
- **Sprint duration**
  - Scrum 2-4 weeks → length is flexible --> produce a release/potentially shippable
  - XP --> shorter iterations → 1-2 weeks to develop a working system
- **Prioritizing the tasks**
  - Scrum → developers determine the order of actions themselves
  - XP → The team follows strict order according to priority and requirement

Source: https://www.youtube.com/watch?v=PRYmsmMdlko

Wrap-Up

Agile Methods - SWEBOK

- Agile methods are considered **lightweight methods** in that they are characterized by short, iterative development cycles, self-organizing teams, simpler designs, code refactoring, test-driven development, frequent customer involvement, and an emphasis on creating a demonstrable working product with each development cycle.

Source: Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK® 2014

Agile Methods - SWEBOK

- Many agile methods are available in the literature; some of the more popular approaches, which are discussed here in brief, include
  - Rapid Application Development (RAD),
  - eXtreme Programming (XP),
  - Scrum, and
  - Feature-Driven Development (FDD)

Source: Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK® 2014

Agile Methods - RAD

- Rapid software development methods are used primarily in **data-intensive**, **business systems** application development.
- The RAD method is enabled with special-purpose database development tools used by software engineers to quickly develop, test, and deploy new or modified business applications.

Source: Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK® 2014

Agile Methods - XP

- This approach uses stories or scenarios for requirements, develops tests first, has direct customer involvement on the team (typically defining acceptance tests), uses pair programming, and provides for continuous code refactoring and integration.
- Stories are decomposed into tasks, prioritized, estimated, developed, and tested. Each increment of software is tested with automated and manual tests; an increment may be released frequently, such as every couple of weeks or so.

Source: Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK® 2014

Agile Methods - Scrum

- Scrum project management-friendly than the others.
- The scrum master manages the activities within the project increment; each increment is called a sprint and lasts no more than 30 days.
- A Product Backlog Item (PBI) list is developed from which tasks are identified, defined, prioritized, and estimated.
- A working version of the software is tested and released in each increment.
- Daily scrum meetings ensure work is managed to plan.

Source: Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK® 2014

Agile Methods - Scrum

- Scrum project management-friendly than the others.
- The scrum master manages the activities within the project increment; each increment is called a sprint and lasts no more than 30 days.
- A Product Backlog Item (PBI) list is developed from which tasks are identified, defined, prioritized, and estimated.
- A working version of the software is tested and released in each increment.
- Daily scrum meetings ensure work is managed to plan.

Source: Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK® 2014

Agile Methods - FDD

- FDD: This is a model-driven, short, iterative software development approach using a five-phase process:
  - (1) develop a product model to scope the breadth of the domain,
  - (2) create the list of needs or features,
  - (3) build the feature development plan,
  - (4) develop designs for iteration-specific features, and
  - (5) code, test, and then integrate the features.
- FDD is similar to an incremental software development approach; it is also similar to XP, except that code ownership is assigned to individuals rather than the team.
- FDD emphasizes an overall architectural approach to the software, which promotes building the feature correctly the first time rather than emphasizing continual refactoring.

Source: Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK® 2014

References

1. CSE 403, Lecture 24, Scrum and Agile Software Development, Reading: Scrum Primer, by Deemer/Benefield/Larman/Vodde, slides created by Marty Stepp http://www.cs.washington.edu/403/
2. System Analysis & Design, An Object-Oriented Approach with UML 5th Edt, A. Dennis, B. H. Wixom, D. Tegarden, 2015.