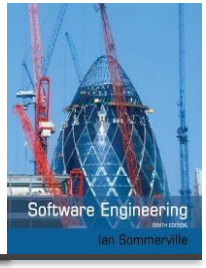
Chapter 3 – Agile Software Development

Topics covered



- ✧ Agile methods
- ✧ Agile development techniques
- ✧ Agile project management
- ✧ Scaling agile methods

Aspect	Plan-Driven Approach	Agile Approach
Development Philosophy	Predictive, plan-centric development	Adaptive, change-centric development
Requirements	Fully defined and frozen early in the project	Evolve continuously throughout the project
Change Handling	Changes are costly and formally controlled	Changes are welcomed, even late in development
Process Structure	Sequential and highly structured	Iterative and incremental
Documentation	Extensive and comprehensive documentation	Lightweight, just-enough documentation
Customer Involvement	Limited, mainly at requirement and acceptance stages	Continuous and active collaboration
Delivery Style	Single or few major releases	Frequent, small, working increments
Risk Management	Risks addressed early through detailed planning	Risks reduced through early and continuous delivery
Testing	Performed after implementation phase	Integrated continuously with development
Team Structure	Hierarchical, role-specialized teams	Self-organizing, cross-functional teams
Project Control	Milestone- and plan-based tracking	Time-boxed iterations (sprints) and reviews
Suitability	Stable, well-understood, large-scale or safety-critical systems	Dynamic, uncertain, rapidly changing environments

Rapid software development

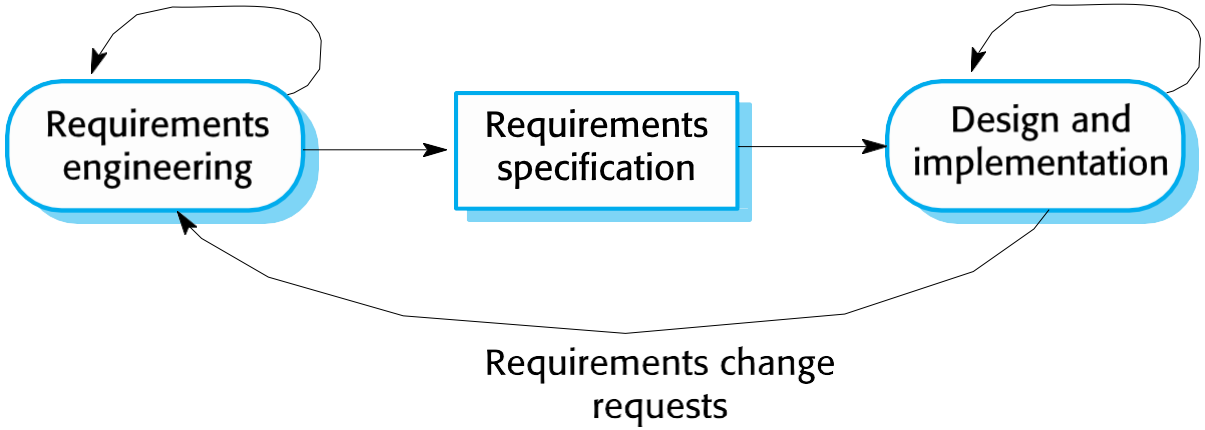
- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Plan-driven development is essential for some types of system but does not meet these business needs.
- Agile development methods emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems

Agile development

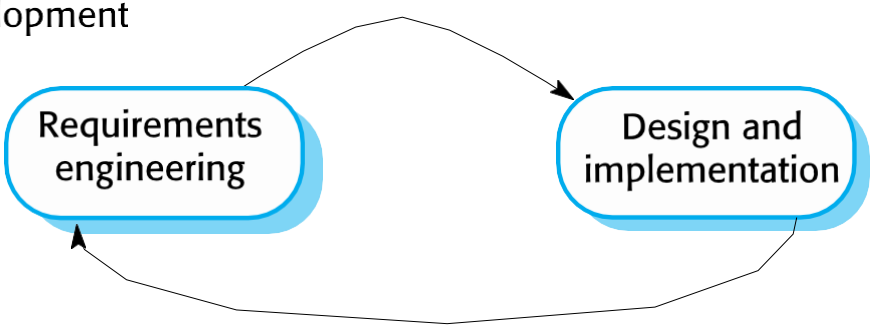
- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code

Plan-driven and agile development

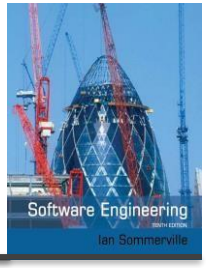
Plan-based development



Agile development



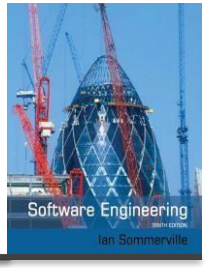
Plan-driven and agile development



- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities (going through several cycles of refinement or improvement within a single phase, but without stepping back to earlier phases in the overall process)
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

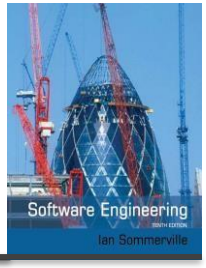
Agile methods

Agile methods



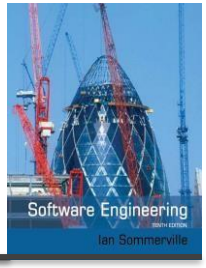
- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Agile manifesto



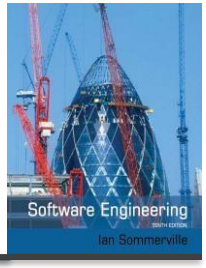
- *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
  - *Individuals and interactions over processes and tools*
  - *Working software over comprehensive documentation*
  - *Customer collaboration over contract negotiation*
  - *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.*

The principles of agile methods



Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

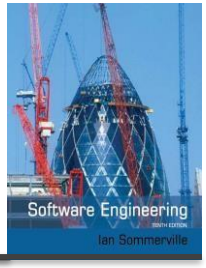
Agile method applicability



- Product development where a software company is developing a small or medium-sized product for sale.
  - Virtually all software products and apps are now developed using an agile approach
- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external rules and regulations that affect the software.

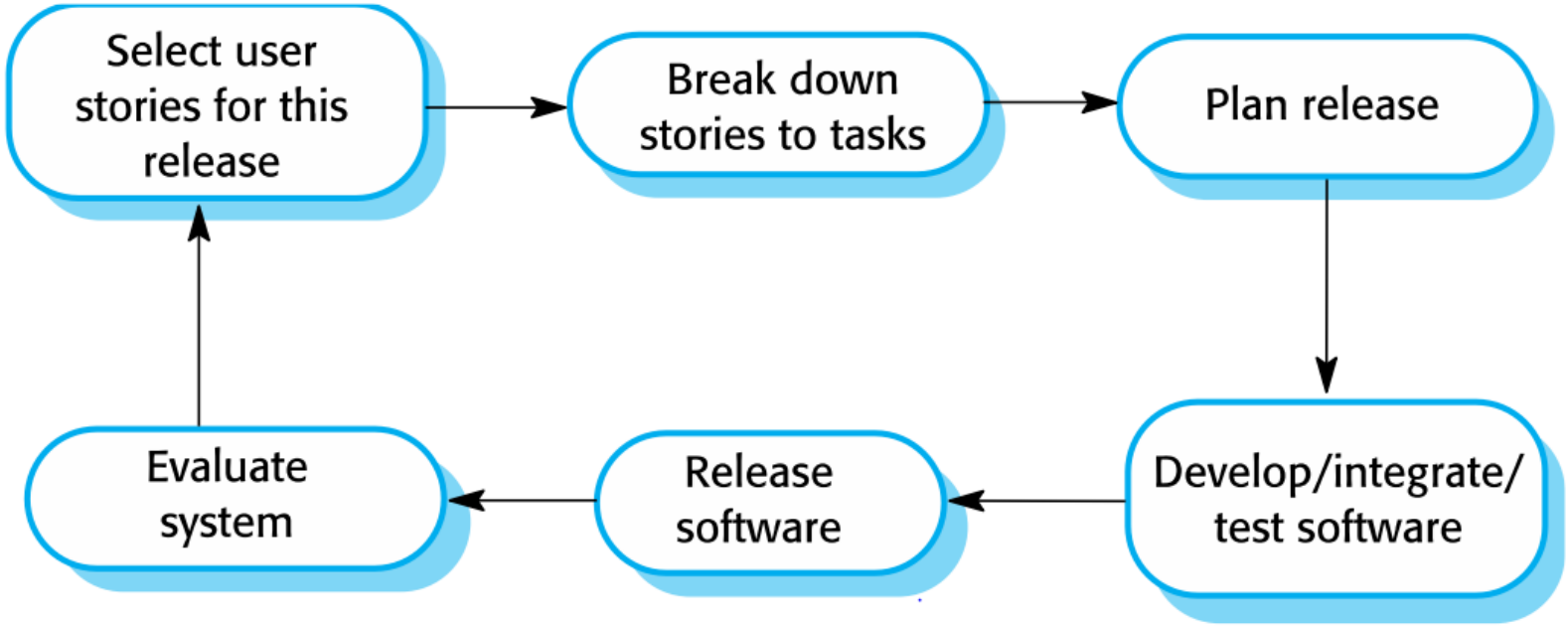
Agile development techniques

Extreme programming

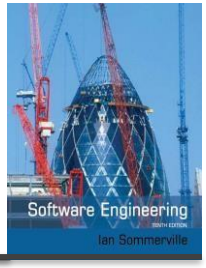


- A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.
- Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

The extreme programming release cycle

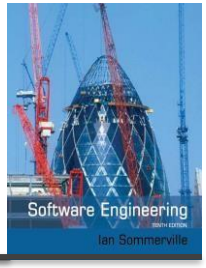


Extreme programming practices (a)



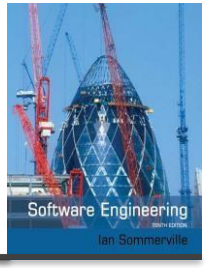
Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices (b)



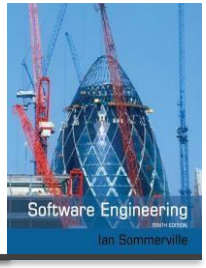
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and agile principles



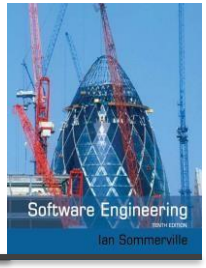
- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

Influential XP practices



- Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.
- Consequently, while agile development uses practices from XP, the method as originally defined is not widely used.
- Key practices
  - User stories for specification
  - Refactoring
  - Test-first development
  - Pair programming

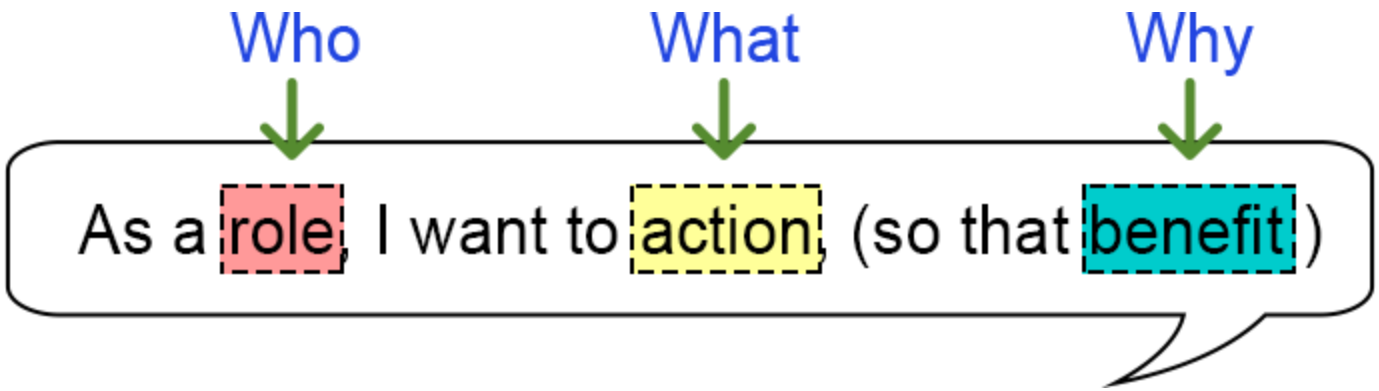
User stories for requirements



- In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.
- User requirements are expressed as user stories or scenarios.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

User Story

As a [**customer**], I want [**shopping cart feature**] so that [**I can easily purchase items online**].
As a [**manager**], I want to [**generate a report**] so that [**I can understand which departments need more resources**].
As a [**customer**], I want to [**receive an SMS when the item arrives**] so that [**I can go pick it up right away**]



A ‘prescribing medication’ story

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either ‘current medication’, ‘new medication’ or ‘formulary’.

If you select ‘current medication’, you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, ‘new medication’, the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose ‘formulary’, you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click ‘OK’ or ‘Change’. If you click ‘OK’, your prescription will be recorded on the audit database. If you click ‘Change’, you reenter the ‘Prescribing medication’ process.

Story ID: Story Title:

User Story:

As a: <role>

I want: <some goal>

So that: <some reason>

Importance:

Estimate:

Acceptance Criteria

And I know I am done when:

Type:

- [ ] Search
- [ ] Workflow
- [ ] Manage Data
- [ ] Payment
- [ ] Report/ View

Examples of task cards for prescribing medication

Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.
Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.
Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.
If within the range, enable the 'Confirm' button.

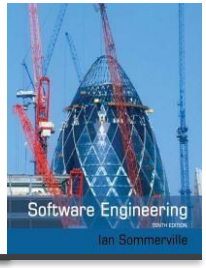
Refactoring

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

Examples of refactoring



- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

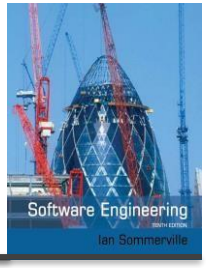
Test-first development

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- XP testing features:
  - Test-first development.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.

Test-driven development

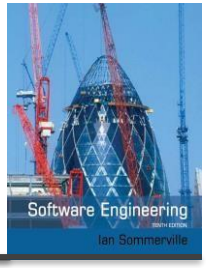
- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as Junit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

Customer involvement



- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

Test case description for dose checking



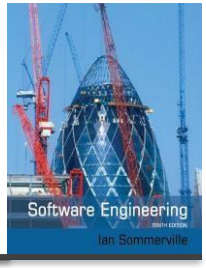
Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

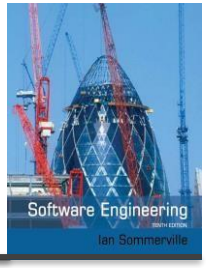
Output:
OK or error message indicating that the dose is outside the safe range.

Test automation



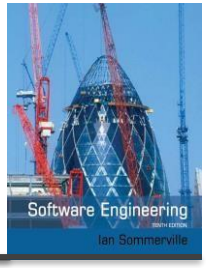
- Test automation means that tests are written as executable components before the task is implemented
  - These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification. An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- As testing is automated, there is always a set of tests that can be quickly and easily executed
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

Problems with test-first development



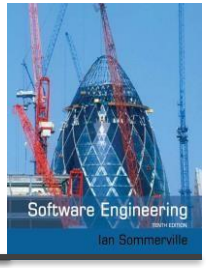
- Programmers prefer programming to testing and sometimes they take short cuts when writing tests. For example, they may write incomplete tests that do not check for all possible exceptions that may occur.
- Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the ‘display logic’ and workflow between screens.
- It difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

Pair programming



- Pair programming involves programmers working in pairs, developing code together.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from improving the system code.

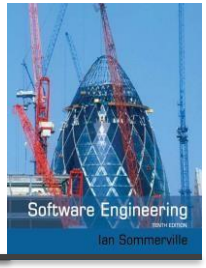
Pair programming



- In pair programming, programmers sit together at the same computer to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is some evidence that suggests that a pair working together is more efficient than 2 programmers working separately.

Agile project management

Agile project management

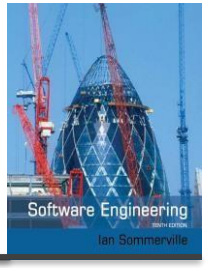


- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- Agile project management requires a different approach, which is adapted to incremental development and the practices used in agile methods.

Scrum

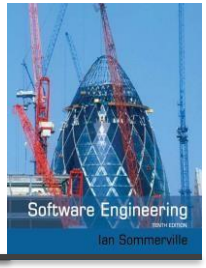
- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.
- There are three phases in Scrum.
  - The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
  - This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
  - The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.
-

Scrum terminology (a)



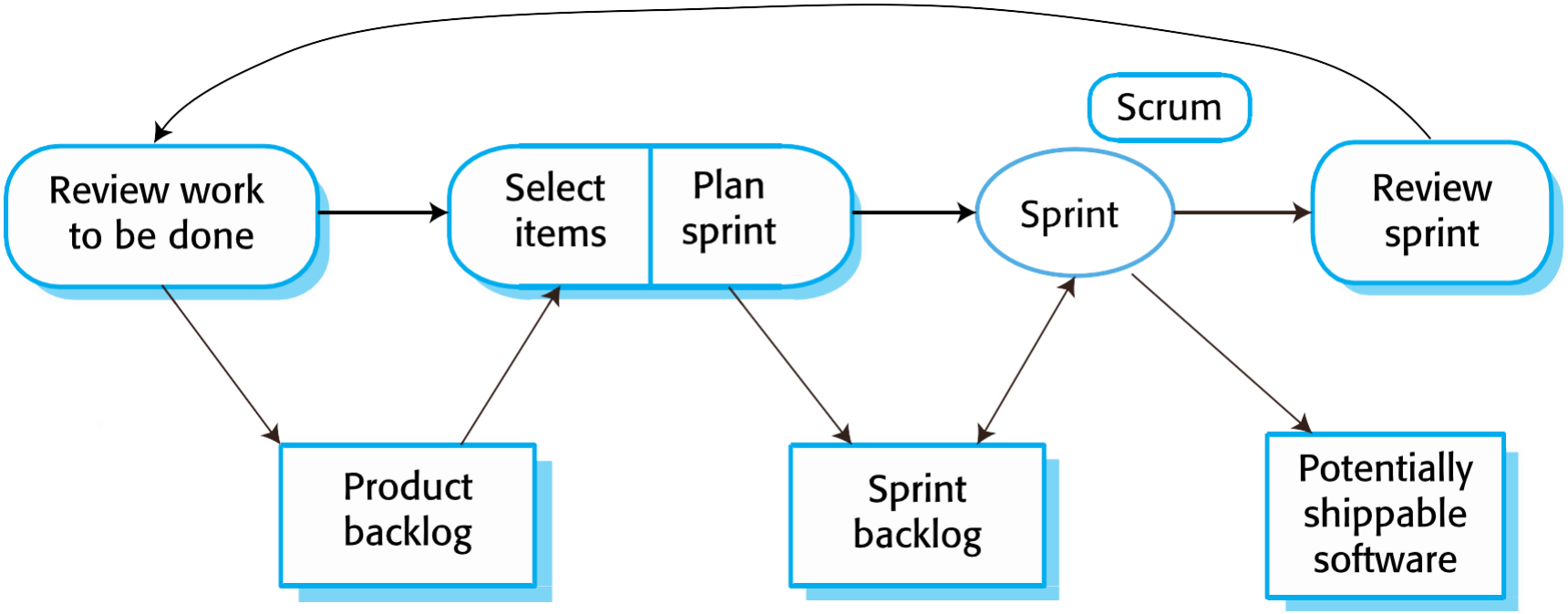
Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

Scrum terminology (b)

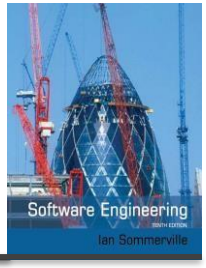


Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

Scrum sprint cycle



The Scrum sprint cycle

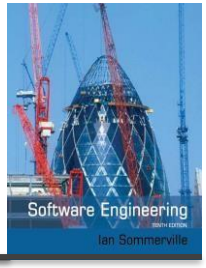


- Sprints are fixed length, normally 2–4 weeks.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog to be developed during the sprint.

The Sprint cycle

- Once these are agreed, the team organize themselves to develop the software.
- During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called ‘Scrum master’.
- The role of the Scrum master is to protect the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

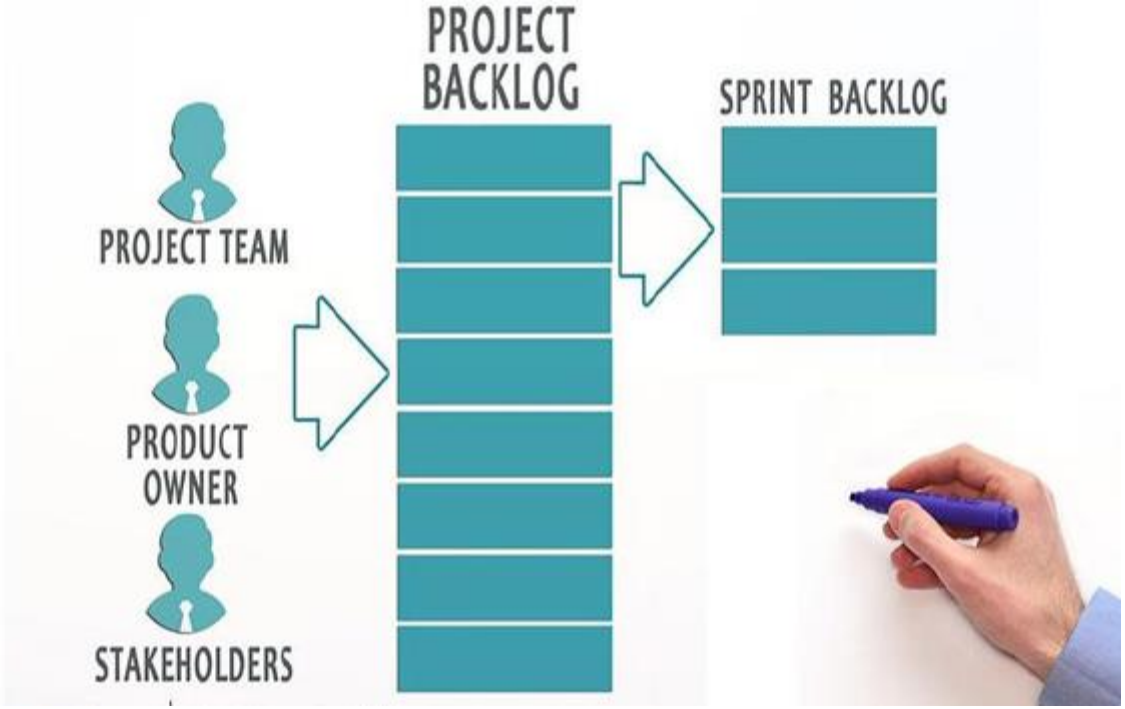
Teamwork in Scrum



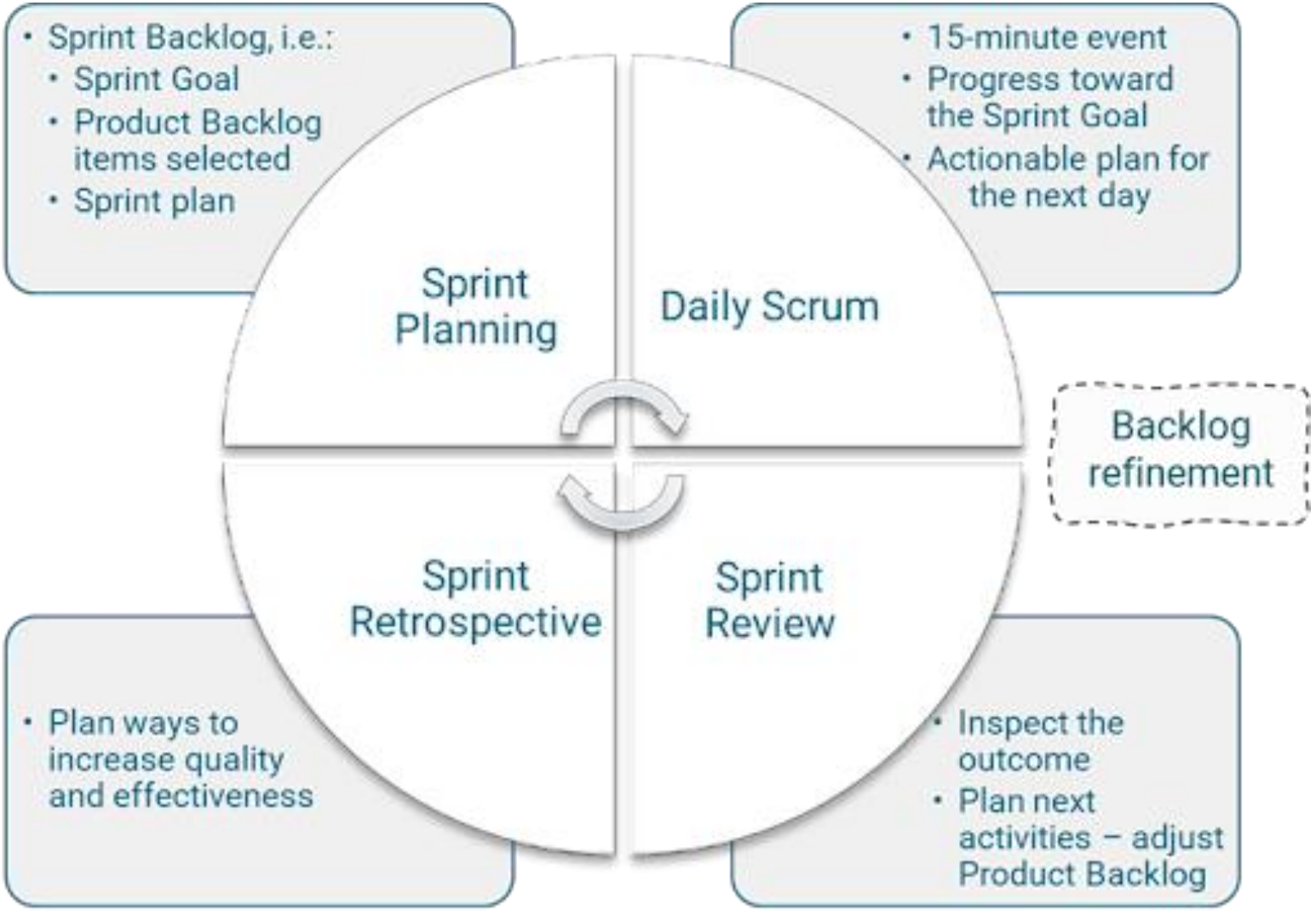
- The ‘Scrum master’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings (Scrums) where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
  - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

Sample Product Backlog

ToDo List		
Story	**Estimation**	**Priority**
As a user I want to be able to reset my password	1	1
As a user I want to edit items	3	2
As a user I want to export data	2	3
As an administrator I want to define KPI's for my sales team	4	4
As a user I want to view my data on mobile	5	5
As an administrator I want to send alerts when new leads come in	2	6
As a user I want to create a report of my data	5	7
As a user I want to update my reminder settings when a date is added	3	8
As a user I want filtering enhancements	4	9
As an administrator I want to configure views of data	5	10
Total	**34**	

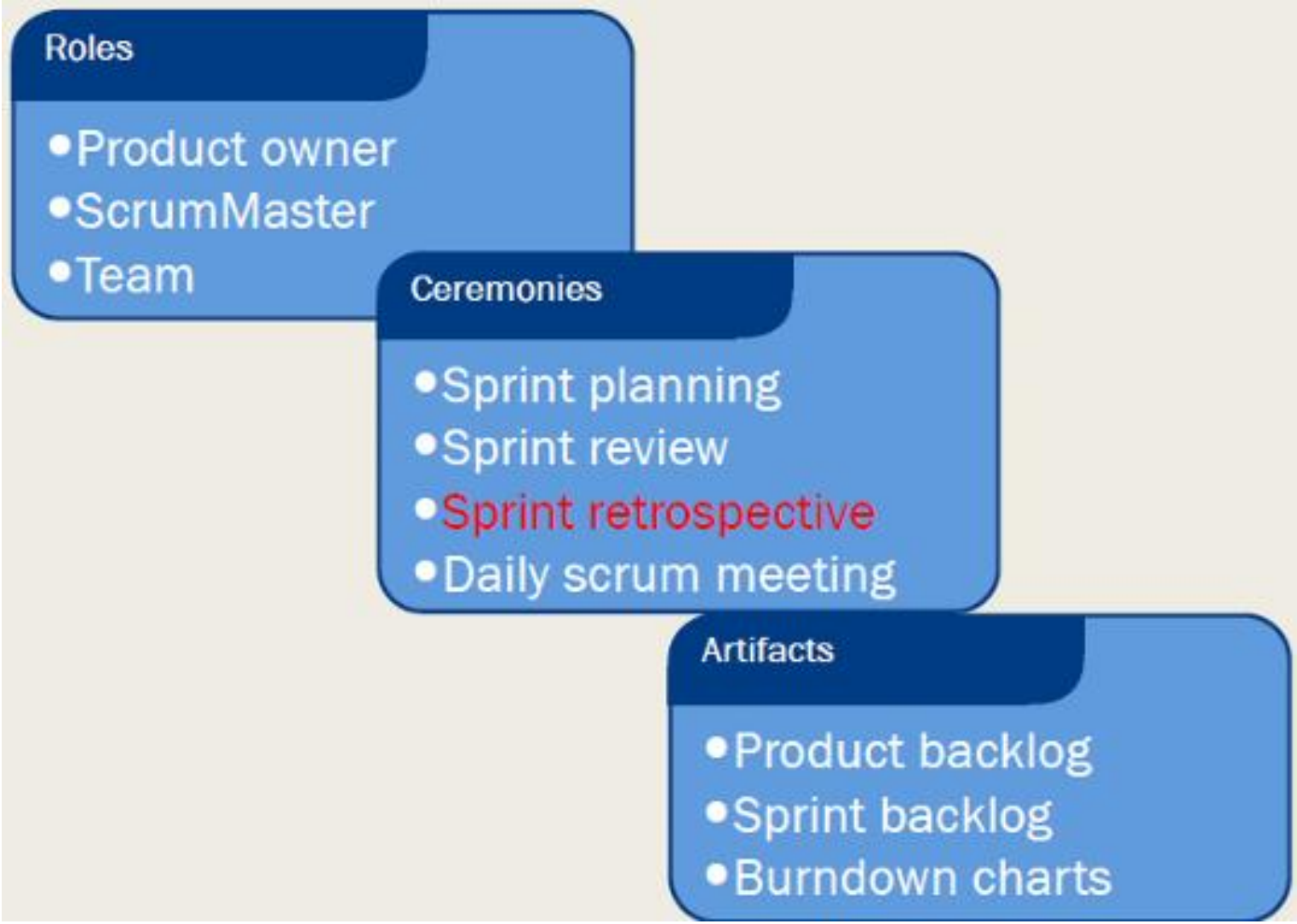

PROJECT BACKLOG
SPRINT BACKLOG
PROJECT TEAM
PRODUCT OWNER
STAKEHOLDERS

Scrum Framework

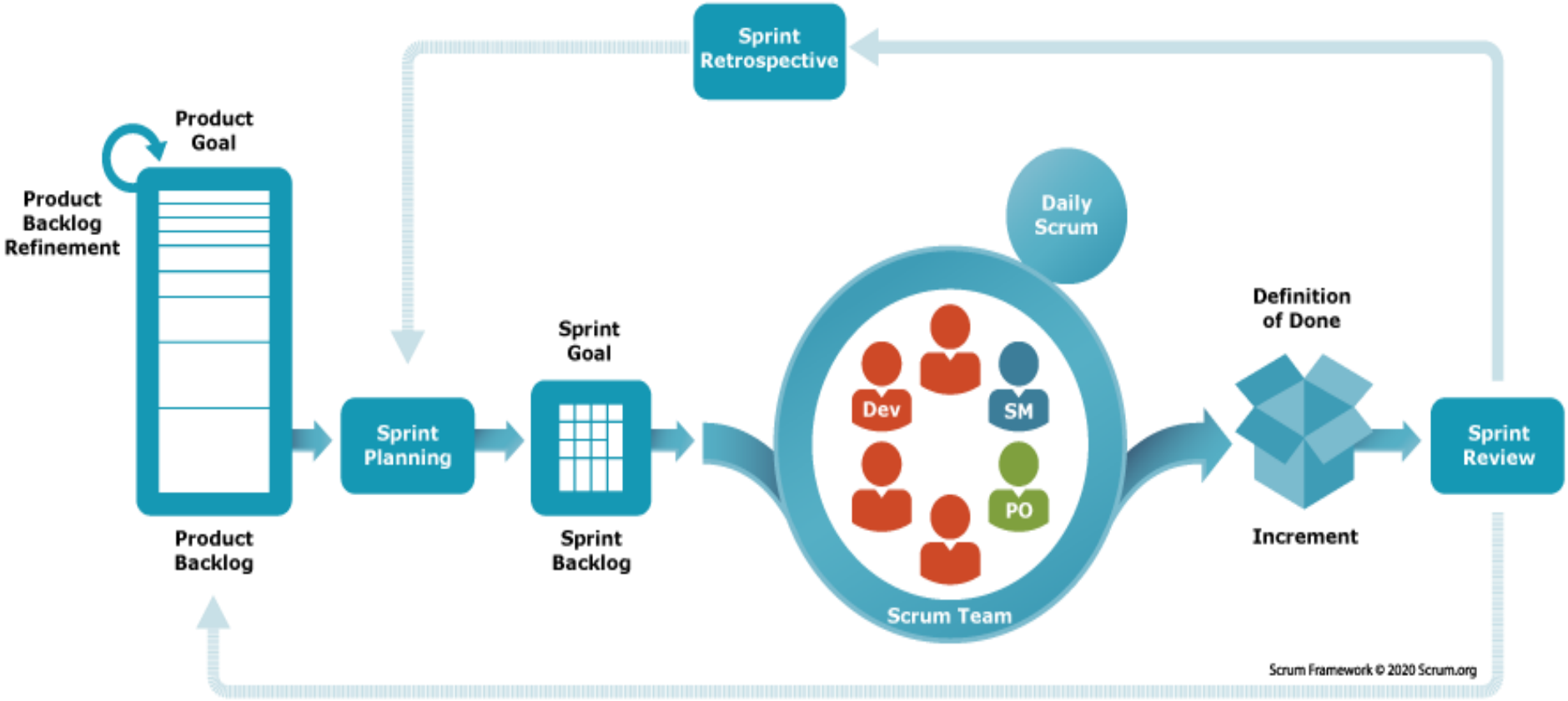


No Changes During Sprint

Scrum Framework



Putting it all together



Sprint Planning

Sprint Velocity

By looking at the amount of work your team completed in previous sprints, you should be able to estimate how much work they can do in future sprints. In Agile development, this estimate is known as sprint velocity

How to estimate sprint velocity

Step 1: Count how many user story points are completed in each sprint

At the end of a sprint, add up how many story points the team completed.

For example, if:

In sprint one:

The team committed to completing five user stories. Each user story had eight story points for a total of 40 story points. The team completed three of the five user stories.

In sprint two:

The team committed to seven user stories (including the two that were not completed in sprint 1). Each user story had eight story points for a total of 56 story points. The team completed four of the seven user stories.

In sprint three:

The team committed to nine user stories. Each user story had eight story points for a total of 72 story points.The team completed five of the nine user stories.

How to estimate sprint velocity

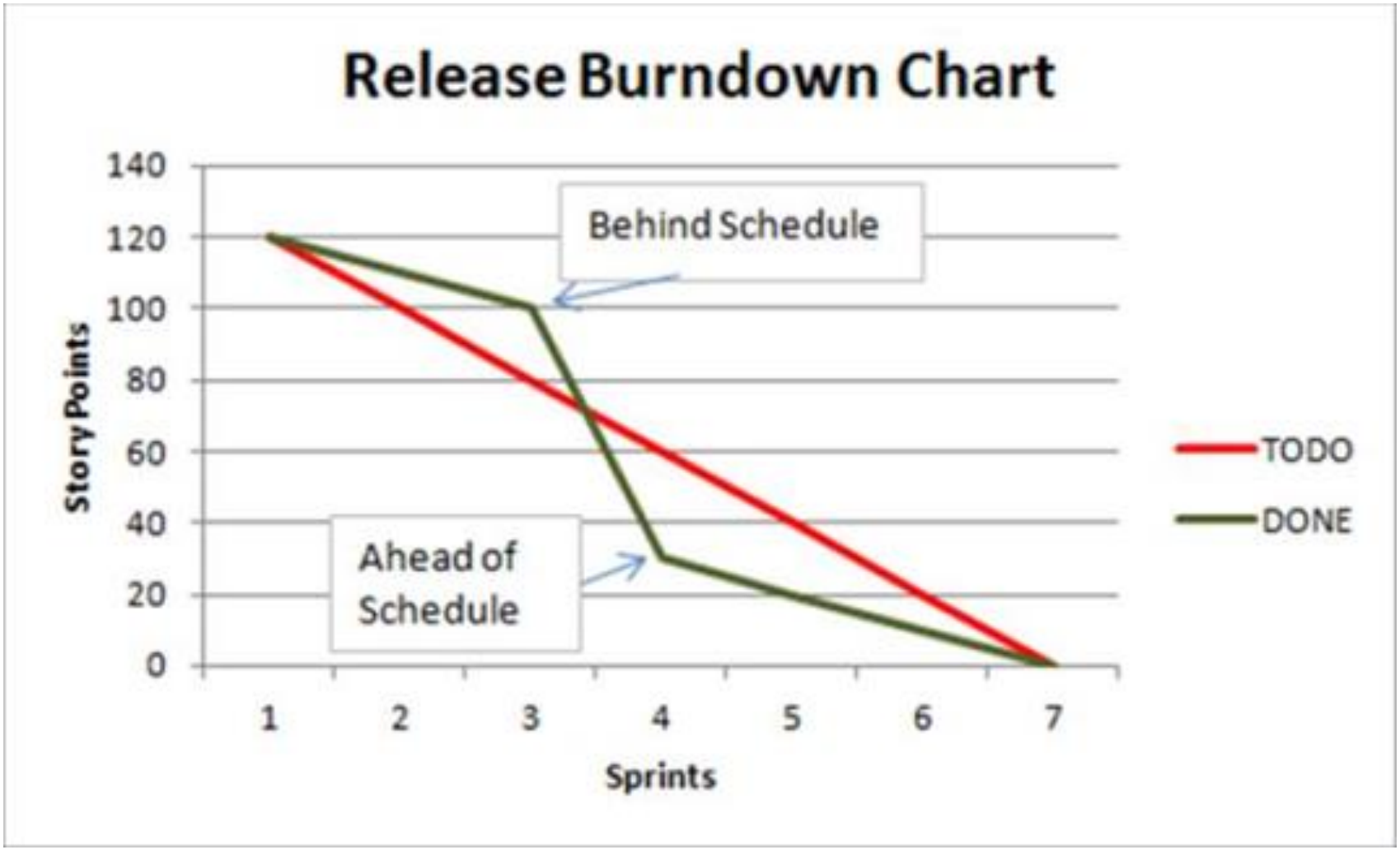
Step 2: Calculate the average of completed story points
Simply add up the total of story points completed from each sprint, then divide by the number of sprints.

Sprint one: 3 user stories x 8 story points = 24
Sprint two: 4 user stories x 8 story points = 32
Sprint three: 5 user stories x 8 story points = 40
Total = 96

So, your average sprint velocity is 96 ÷ 3 = 32.

How to estimate sprint velocity

- You can now base the amount of work to be done in future sprints on an average of 32 story points. If you have 160 story points remaining to be completed in the project, you can assume that your team will need another five sprints to complete the project.
- If your team is new to Agile development, you won't have any previous sprints to look at. As part of your sprint velocity estimation to-do list, you'll have to complete a couple of sprints while tracking how many story points are completed in each. Then you will have some useful data that will help calculate an average.


Release Burndown Chart
140
120
100
80
60
40
20
0
Story Points
Behind Schedule
Ahead of Schedule
1
2
3
4
5
6
7
Sprints
TODO
DONE

Scrum on a Page

Roles

Product Owner
Set Priorities
Manage Product Backlog

Scrum Master
Teach Scrum
Manage Process
Protect Team
Enforce Rules
Remove Blocks

Team
Develop Product
Organize Work
Report Progress

Stakeholders
Observe & advise

Artifacts

Product Backlog
List of requirements
Owned by product owner
Anybody can add to it
Prioritized by business value
Can change without affecting the active sprint

Sprint Goal
One sentence summary
Defined by Product Owner
Accepted by Team

Sprint Backlog
Decomposed task list
Driven by a portion of Product Backlog
Owned by Team
Only Team modifies it

Blocks List
List of blocks & pending decisions
Owned by Scrum Master
Blocks stay on list until resolved

Increment
Version of the product
Potentially shippable
Working functionality
Tested & documented according to project definition of "DONE"

Meetings

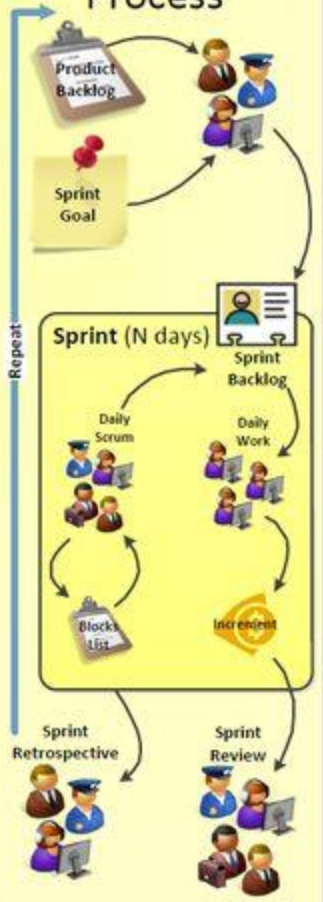
Sprint Planning
Part A
Time-boxed to 4 hours
Run by Scrum Master
Declare Sprint Goal
Top of Product Backlog presented by Product Owner to Team
Team asks questions & selects topmost features
Part B
Time-boxed to 4 hours
Run by Scrum Master
Team decomposes selected features into a Sprint Backlog
Team adjusts +/- features by estimates against sprint capacity

Daily Scrum
Time-boxed to 15 minutes
Run by Scrum Master
Attended by all
Stakeholders do not speak
Same time/place every day
Answer 3 questions:
1) What I did yesterday?
2) What I'll do today?
3) What's in my way?
Team updates the Sprint Backlog
Scrum Master updates the Blocks List

Sprint Review
Time-boxed to 2 or 4 hours
Run by Scrum Master
Attended by all
Informal, informational, Discussion
Team demonstrates increment
All discuss

Sprint Retrospective
Time-boxed to 1 or 2 hours
Run by Scrum Master
Attended by Team and Product Owner
Discuss process improvements, successes and failures
Adjust process

Process



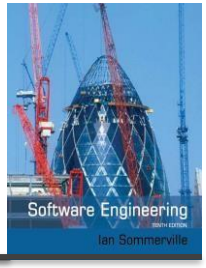
www.andrewsiemer.com / (661) 600-2355 / asiemer@hotmail.com

Kanban Board

Visual signal
Increase visibility

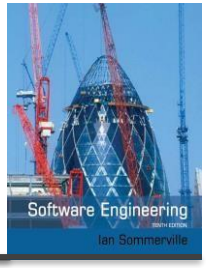
SYDLE

Scrum benefits



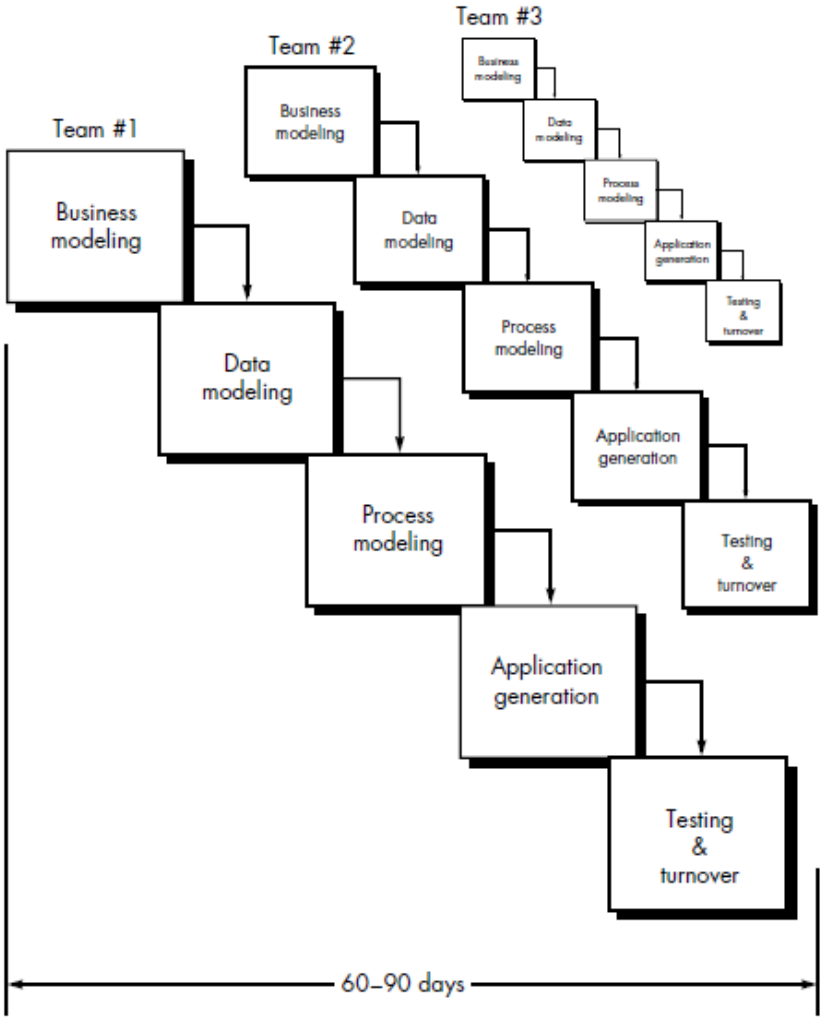
- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

Summary



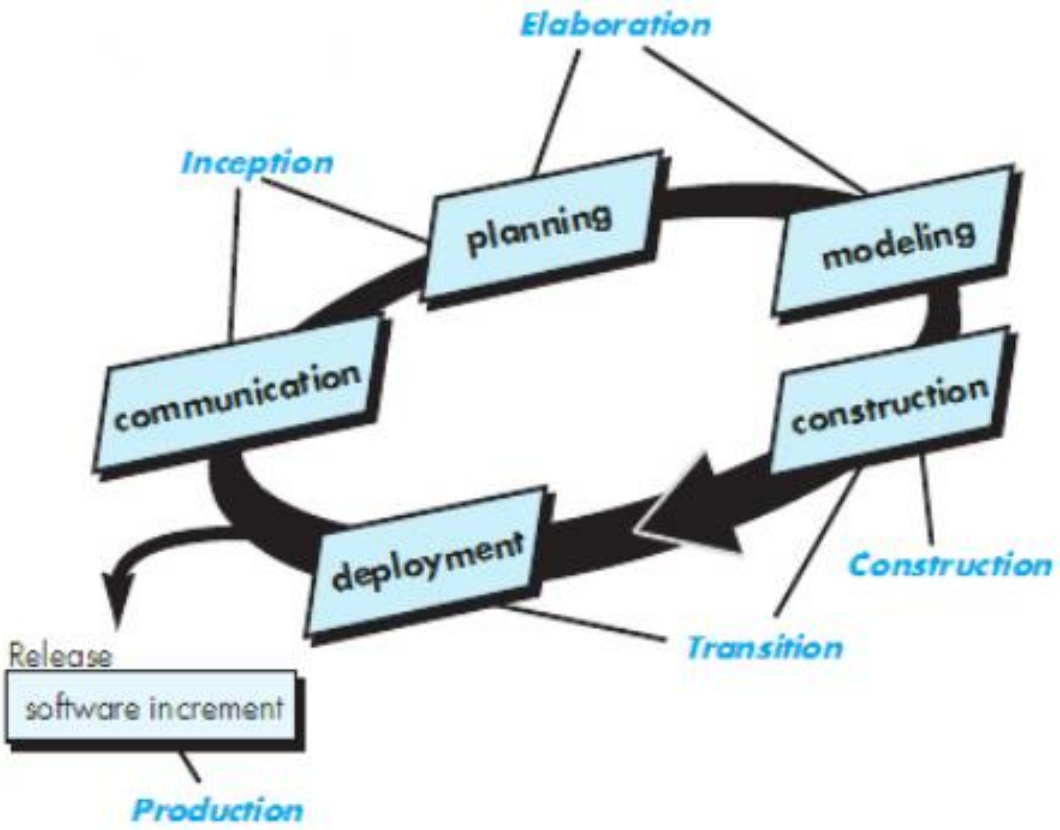
- Scrum is an agile method that provides a project management framework.
  - It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Many practical development methods are a mixture of plan-based and agile development.
- Scaling agile methods for large systems is difficult.
  - Large systems need up-front design and some documentation and organizational practice may conflict with the informality of agile approaches.

Rapid Application Development

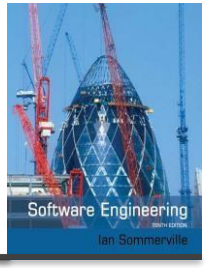


RAD works only when time is more important than perfection.

Unified Process Model



Core Characteristics of Unified Process



1.Iterative and Incremental
Development proceeds through multiple iterations, delivering increments.
2.Use-Case Driven
Functional requirements are captured using **use cases**.
3.Architecture-Centric
System architecture is defined early and refined continuously.
4.Risk-Focused
High-risk elements are addressed early.

Unified Process Model

UP phase	Objective
Inception	Develop an approximate vision of the system, make the business case, define the scope, and produce rough estimates for cost and schedule.
Elaboration	Refine the vision, identify and describe all requirements, finalize the scope, design and implement the core architecture and functions, resolve high risks, and produce realistic estimates for cost and schedule.
Construction	Iteratively implement the remaining lower-risk, predictable, and easier elements and prepare for deployment.
Transition	Complete the beta test and deployment so users have a working system and are ready to benefit as expected.

Process Models (Recap)

- Waterfall model
  - Classical
  - V model
- Incremental Models
  - Phased development in increment
  - Rapid Application Development (RAD)
- Evolutionary Models
  - Prototyping model
  - Spiral model
- Unified process
- Agile methods
  - XP
  - Scrum

Reference

How to Estimate Sprint Velocity | Lucidchart Blog