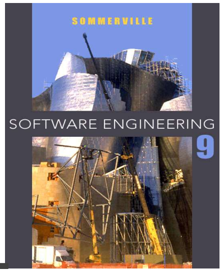
Agile Software Development

Agile

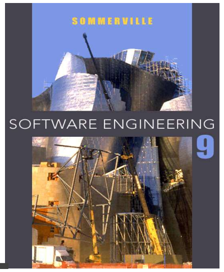


Agile Development techniques should be aligned with SE principles

But…

Are they?

Rapid software development



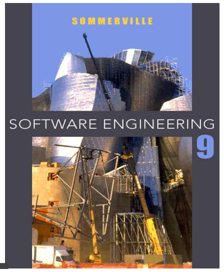
Rapid development and delivery is now often the most important requirement for software systems

- Businesses operate in a fast – **changing environment** and it is practically impossible to produce a set of stable software requirements
- Software has to evolve quickly to reflect changing business needs.

Rapid software development

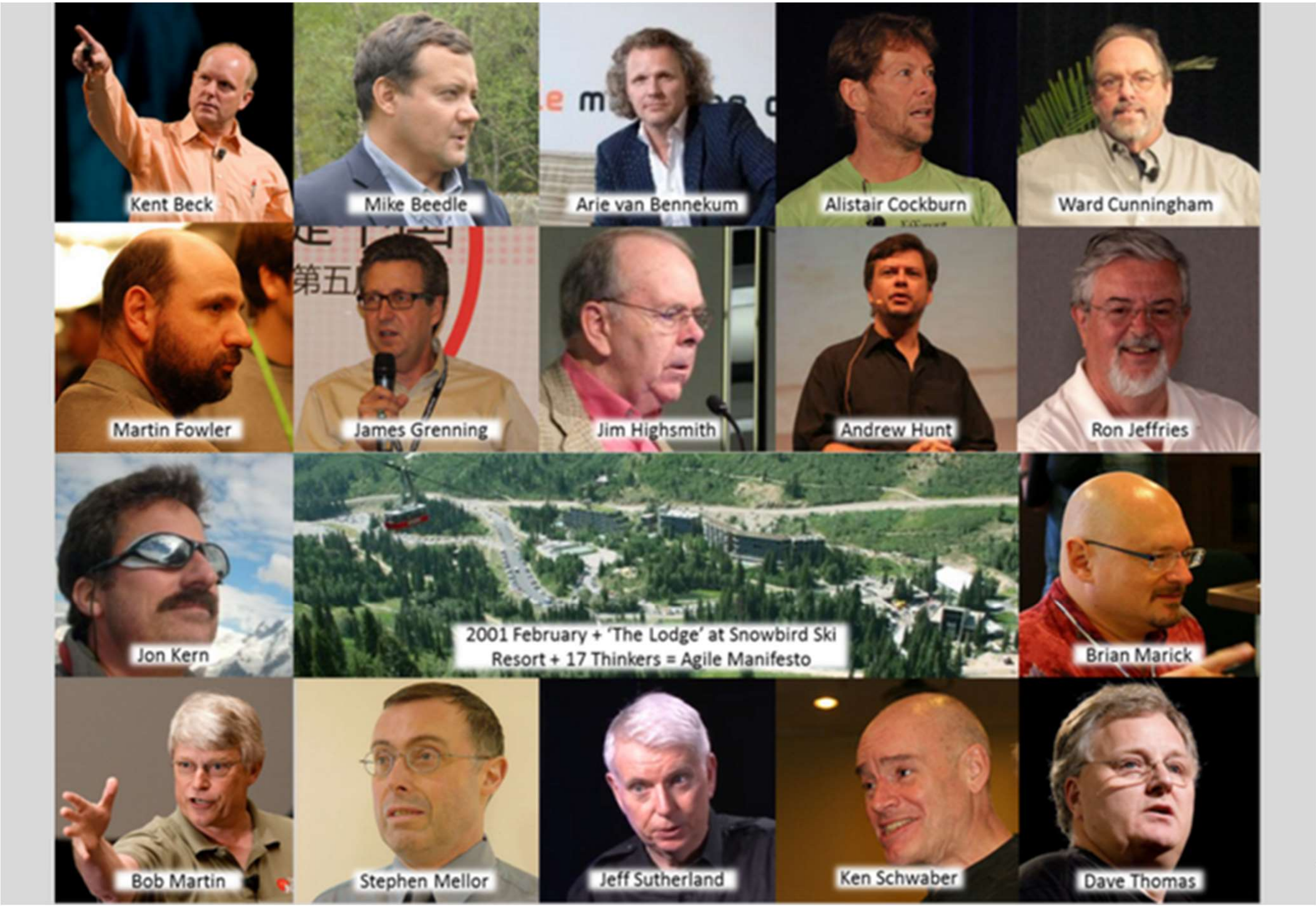
- Specification, design and implementation are **inter-leaved**
- System is developed as a series of **versions** with stakeholders involved in version evaluation
- User interfaces are often developed using an IDE (integrated development environment) and graphical toolset.

Agile methods



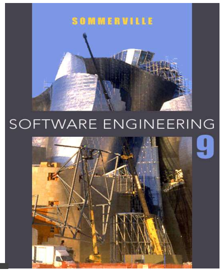
- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software 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



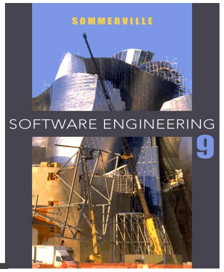
2001 February + 'The Lodge' at Snowbird Ski Resort + 17 Thinkers = Agile Manifesto

Agile manifesto



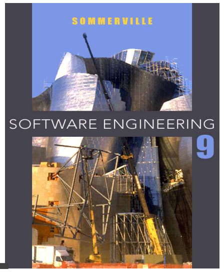
- *We are uncovering better ways of developing software by doing it and helping others do it.*
- *Through this work we have come to value:*
  - *Individuals and interactions over processes and tools*
    Working software over comprehensive documentation
    Customer collaboration over contract negotiation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

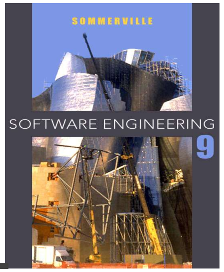


Product development where a software company is developing a small or medium-sized product for sale.

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 not a lot of external rules and regulations that affect the software.

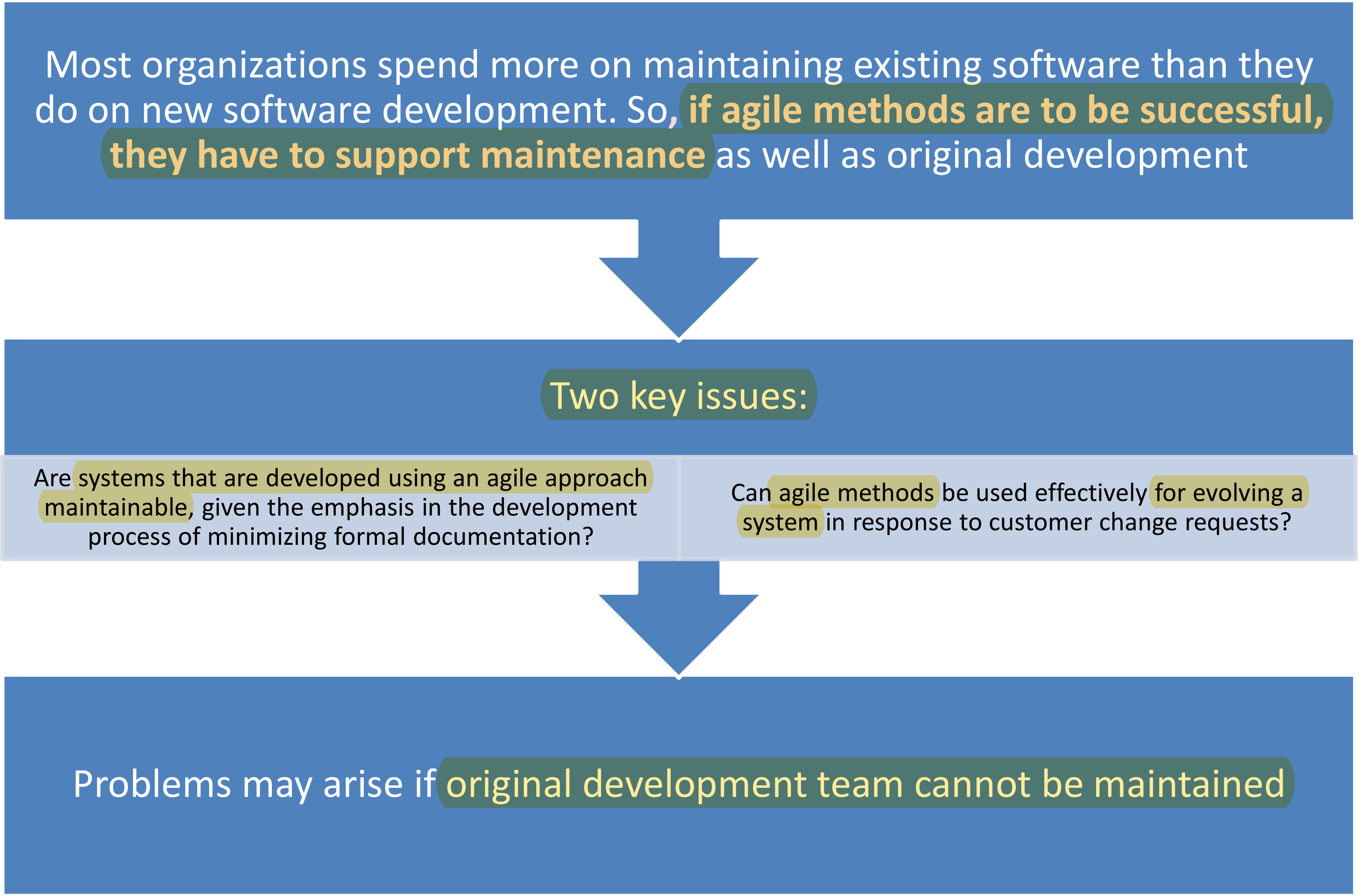
However, because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

Problems with agile methods



- Difficult to keep the interest of customers who are involved in the process
- Team members may be unsuited to the intense involvement that characterises agile methods
- Prioritising changes can be difficult where there are multiple stakeholders
- Maintaining simplicity requires extra work
- Contracts may be a problem as with other approaches to iterative development

Agile methods and software maintenance



Most organizations spend more on maintaining existing software than they do on new software development. So, **if agile methods are to be successful, they have to support maintenance** as well as original development

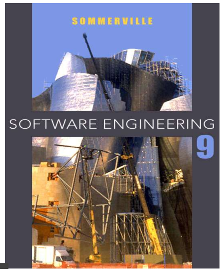
Two key issues:

Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?

Can agile methods be used effectively for evolving a system in response to customer change requests?

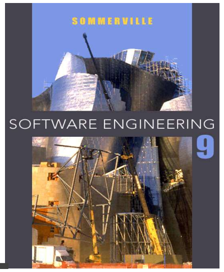
Problems may arise if original development team cannot be maintained

What is a User Story?



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

User Story Description
(team at Connextra, 2001)

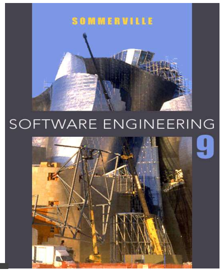


As a [user role] I want to [goal] so I can [reason]

For example:

- **As a** registered user **I want to** log in **so I can** access subscriber-only content

User Story Description



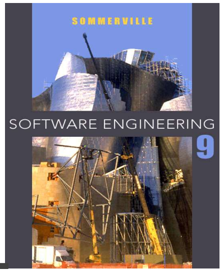
Who (user role)

What (goal)

Why (reason)

- gives clarity as to why a feature is useful
- can influence how a feature should function
- can give you ideas for other useful features that support the user's goals

Other formats



Mike Cohn, a well-known author on user stories, regards the "so that" clause as optional

- "As a <role>, I want <goal/desire>"

Chris Matts suggested that "hunting the value" proposed this alternative

- "In order to <receive benefit> as a <role>, I want <goal/desire>"

Another template based on the Five Ws specifies:

- "As <who> <when> <where>, I <what> because <why>."

A template developed at Capital One in 2004

- "As a <role>, I can <action with system> so that <external benefit>"

Examples

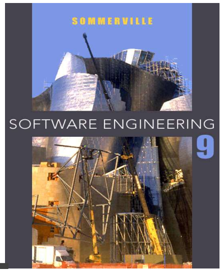
Quiz Recall

As a manager, **I want to** browse my existing quizzes **so I can** recall what I have in place and figure out if I can just reuse or update an existing quiz for the position I need now.

Limited Backup

As a user**, I can** indicate folders not to backup **so that** my backup drive isn't filled up with things I don't need saved

User Story Cards have 3 parts

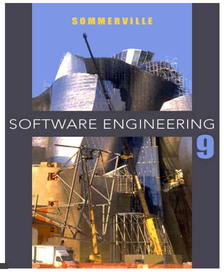


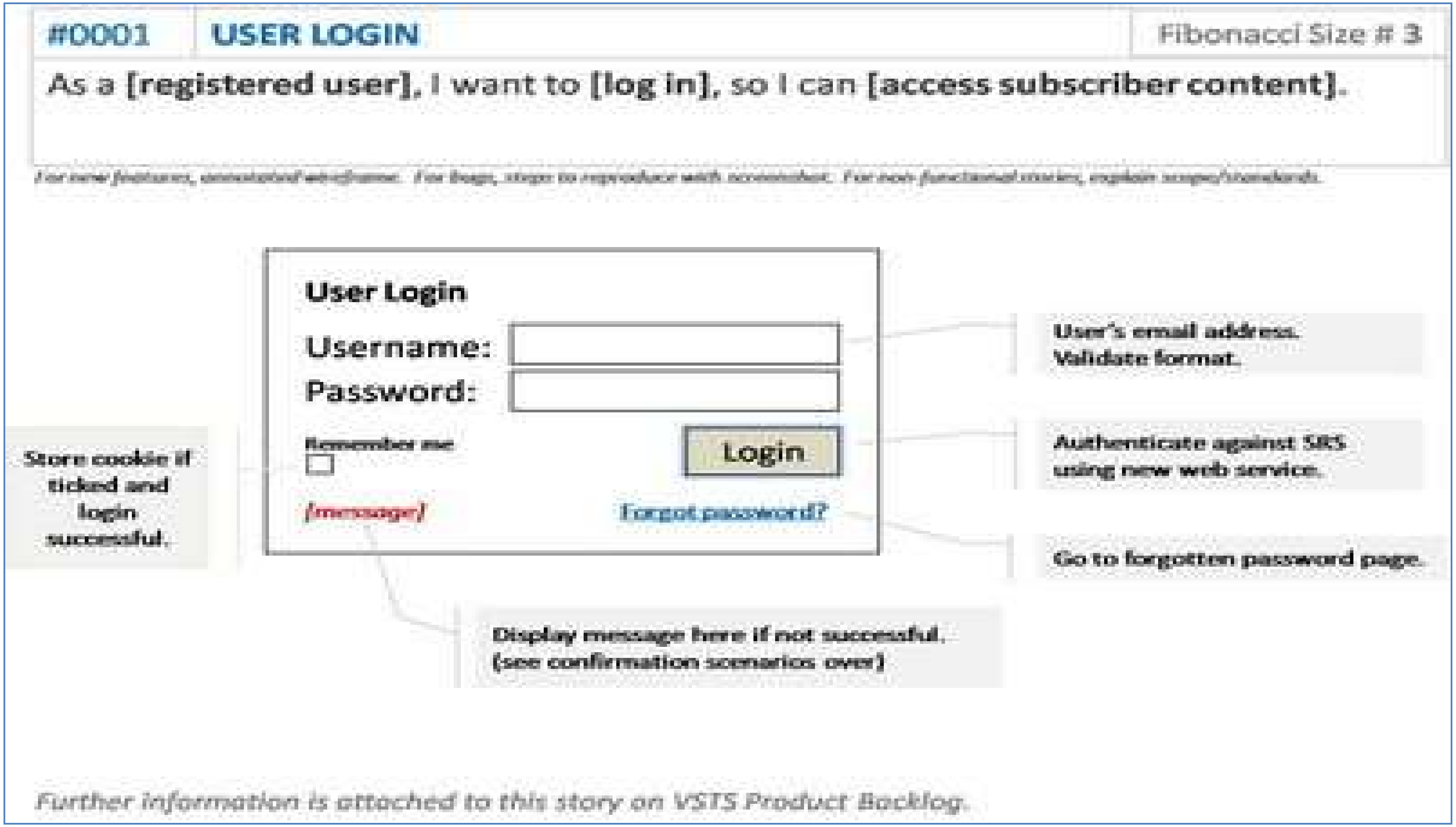
CARD - A WRITTEN DESCRIPTION OF THE USER STORY FOR PLANNING PURPOSES AND AS A REMINDER

CONVERSATION - A SECTION FOR CAPTURING FURTHER INFORMATION ABOUT THE USER STORY AND DETAILS OF ANY CONVERSATIONS

CONFIRMATION - A SECTION TO CONVEY WHAT TESTS WILL BE CARRIED OUT TO CONFIRM THE USER STORY IS COMPLETE AND WORKING AS EXPECTED

User Story Example: Front of Card





#0001	USER LOGIN	Fibonacci Size # 3

As a [registered user], I want to [log in], so I can [access subscriber content].

For new features, annotated wireframe. For bugs, steps to reproduce with screenshot. For non functional stories, explain scope/standards.

User's email address. Validate format.

Authenticate against SRS using new web service.

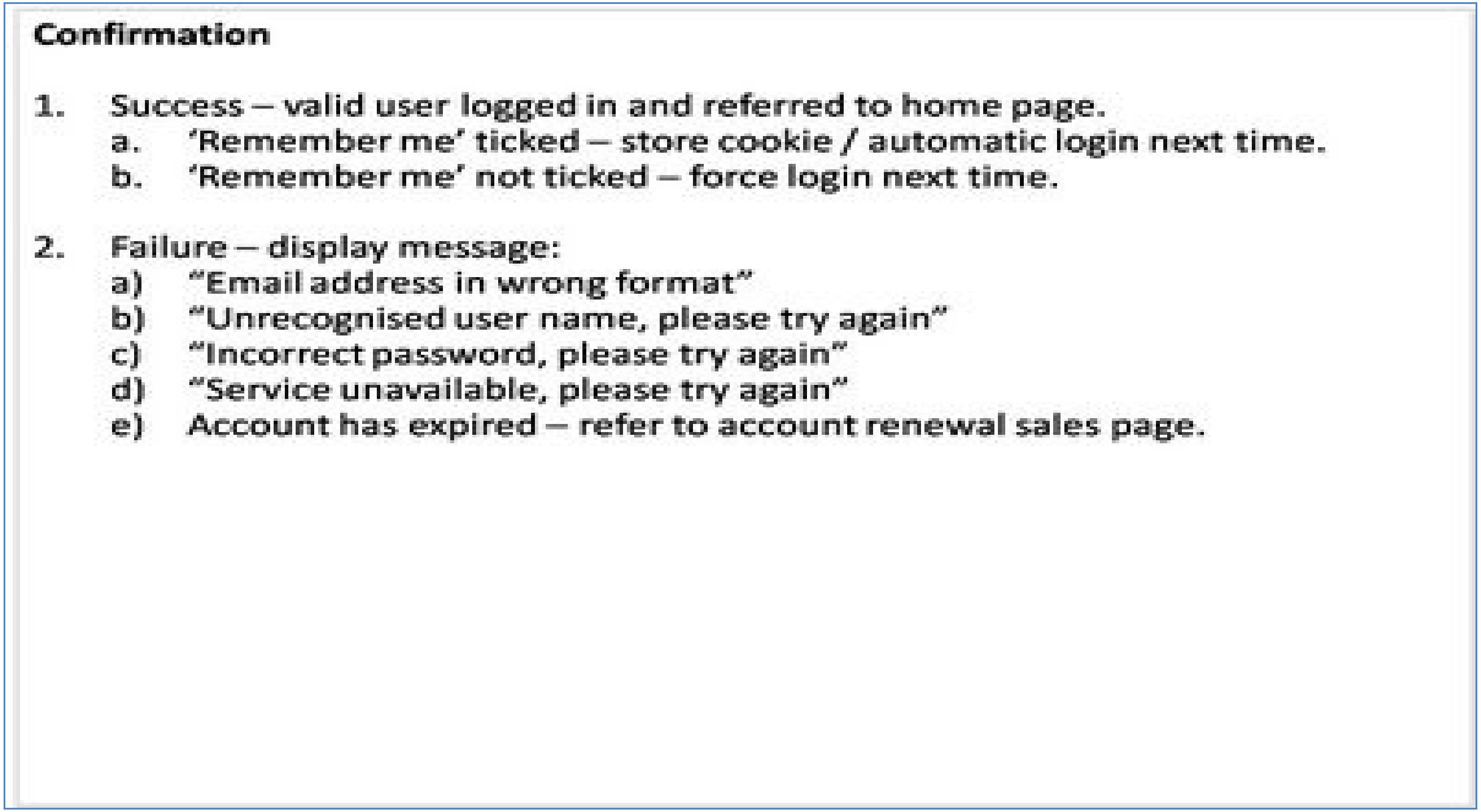
Go to forgotten password page.

Store cookie if ticked and login successful.

Display message here if not successful. (see confirmation scenarios over)

Further information is attached to this story on VSTS Product Backlog.

User Story Example: Back of Card

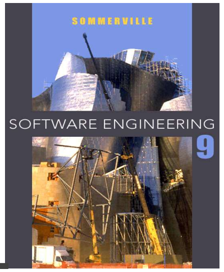


Confirmation

1. Success – valid user logged in and referred to home page.
   a. 'Remember me' ticked – store cookie / automatic login next time.
   b. 'Remember me' not ticked – force login next time.

2. Failure – display message:
   a) "Email address in wrong format"
   b) "Unrecognised user name, please try again"
   c) "Incorrect password, please try again"
   d) "Service unavailable, please try again"
   e) Account has expired – refer to account renewal sales page.

Limitations



Scale-up problem

- User stories written on small physical (story) cards are hard to maintain, difficult to scale to large projects and for geographically distributed teams.

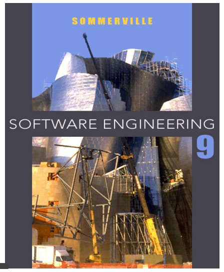
Vague, informal and incomplete

- User story cards are regarded as conversation starters. Being informal, they are open to many interpretations.
- Being brief, they do not state all of the details necessary to implement a feature. Stories are inappropriate for formal agreements or writing legal contracts

Lack of non-functional requirements

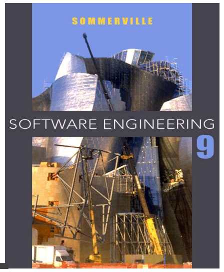
- User stories rarely include performance or NFR details, so non-functional tests (e.g. response time) may be overlooked

Epics and User stories



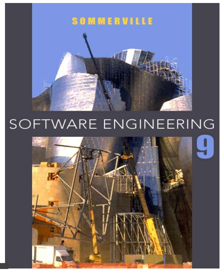
- **Epic**
  - Large stories or multiple user stories that are very closely related are summarized as epics. A common explanation of epics is also: a user story that is too big for a sprint.
  - Ex. “Registration & Authentication”
- User stories for the epic “Registration & Authentication”
  - “Sign Up with Email”, “Sign Up with Facebook”, “Log In with Email”, “Log In with Facebook” “Forgot Password”, and “Log out”

Issues



- HOW MANY USER STORIES SHOULD BE IN AN EPIC?
  - There is no exact number because every project is different. But we would recommend adding no more than 10 user stories to an epic.
  - This will allow to complete it within 3 months and proceed with further development.
- WHAT INFORMATION SHOULD BE INCLUDED IN EPICS?
  - Epics should contain project, technical and design requirements.
  - Also, there should be an introduction explaining what and why should be added to your project, what metrics of your business you want to improve.

Initiative and Themes

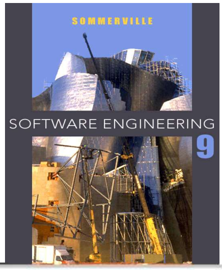


✧ Initiative

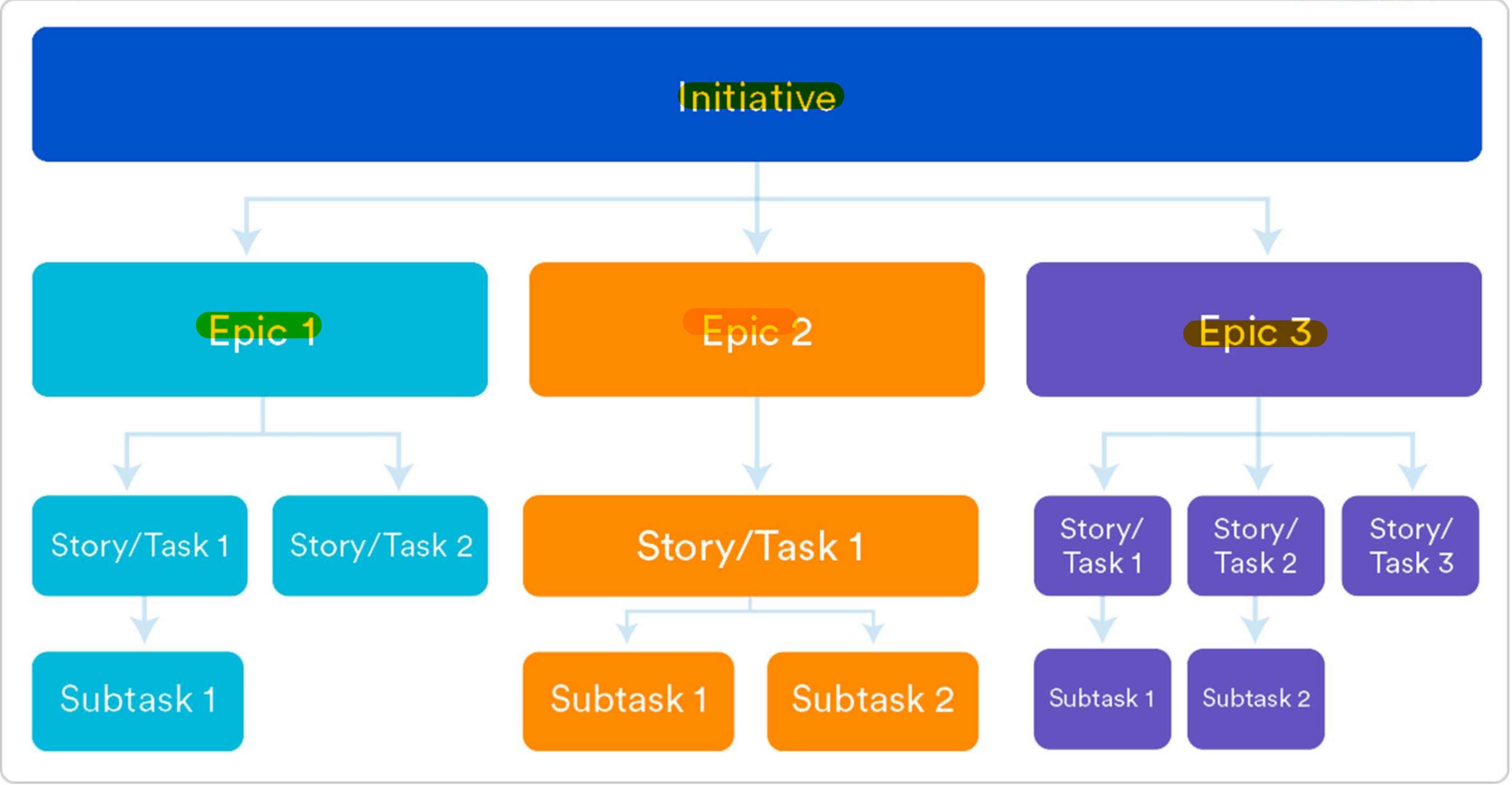
- Multiple epics or stories grouped together hierarchically
- Ex. Your rocket ship company wants to decrease the cost per launch by 5% this year.
- Epics:
  - "Decrease launch-phase fuel consumption by 1%,"
  - "Increase launches per quarter from 3 to 4,"
  - "Turn all thermostats down from 71 to 69 degrees

✧ Theme

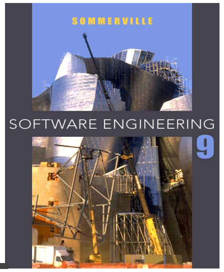
- Multiple epics grouped together by a common theme or semantic relationship.
- A theme for a rocket ship company would be something like "Safety First."

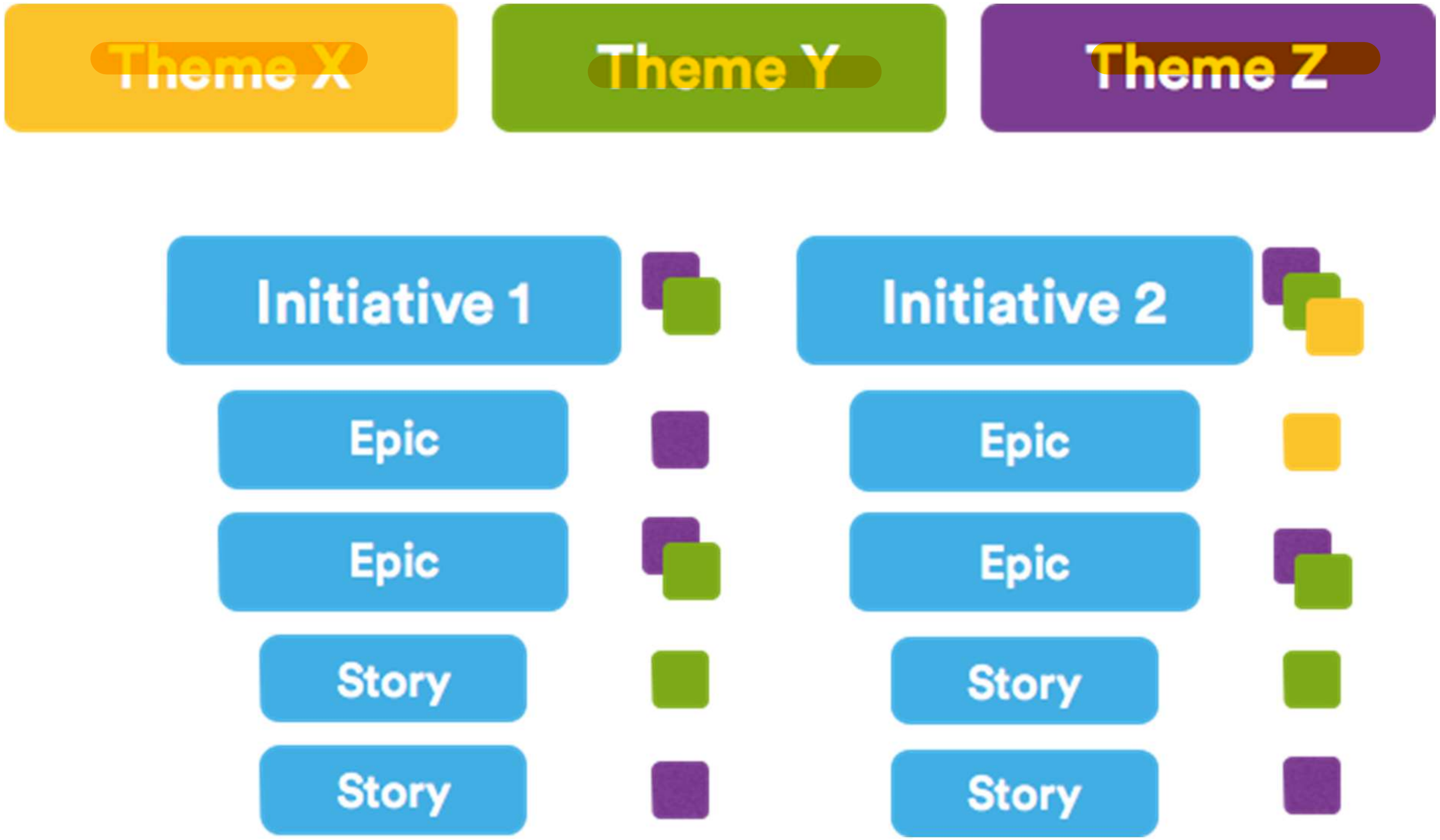


Initiative, Epics and User Stories



Themes, Initiatives, Epics, and User stories



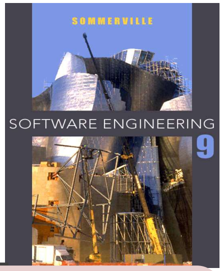


Theme X 🟨 Theme Y 🟩 Theme Z 🟪

Initiative 1 🟪🟩
- Epic 🟪
- Epic 🟪🟩
- Story 🟩
- Story 🟪

Initiative 2 🟪🟩🟨
- Epic 🟨
- Epic 🟪🟩
- Story 🟩
- Story 🟪

Technical, human, organizational issues



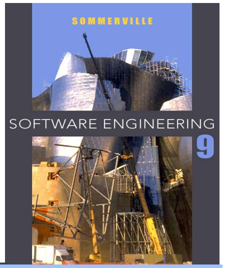
Most projects include elements of plan-driven and agile processes. Deciding on the balance raises some issues

Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.

Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.

How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

Technical, human, organizational issues



What type of system is being developed?

- Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements).

What is the expected system lifetime?

- Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.

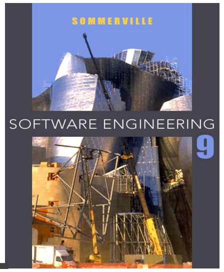
What technologies are available to support system development?

- Agile methods rely on good tools to keep track of an evolving design

How is the development team organized?

- If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.

Technical, human, organizational issues



Are there cultural or organizational issues that may affect the system development?

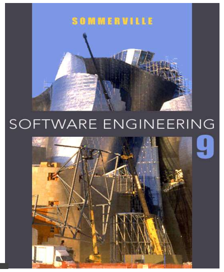
- Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.

How good are the designers and programmers in the development team?

- It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code

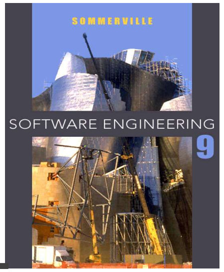
Is the system subject to external regulation?

- If a system has to be approved by an external regulator (e.g. the FAA (Federal Aviation Administration) approve software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.



Scrum

Scrum



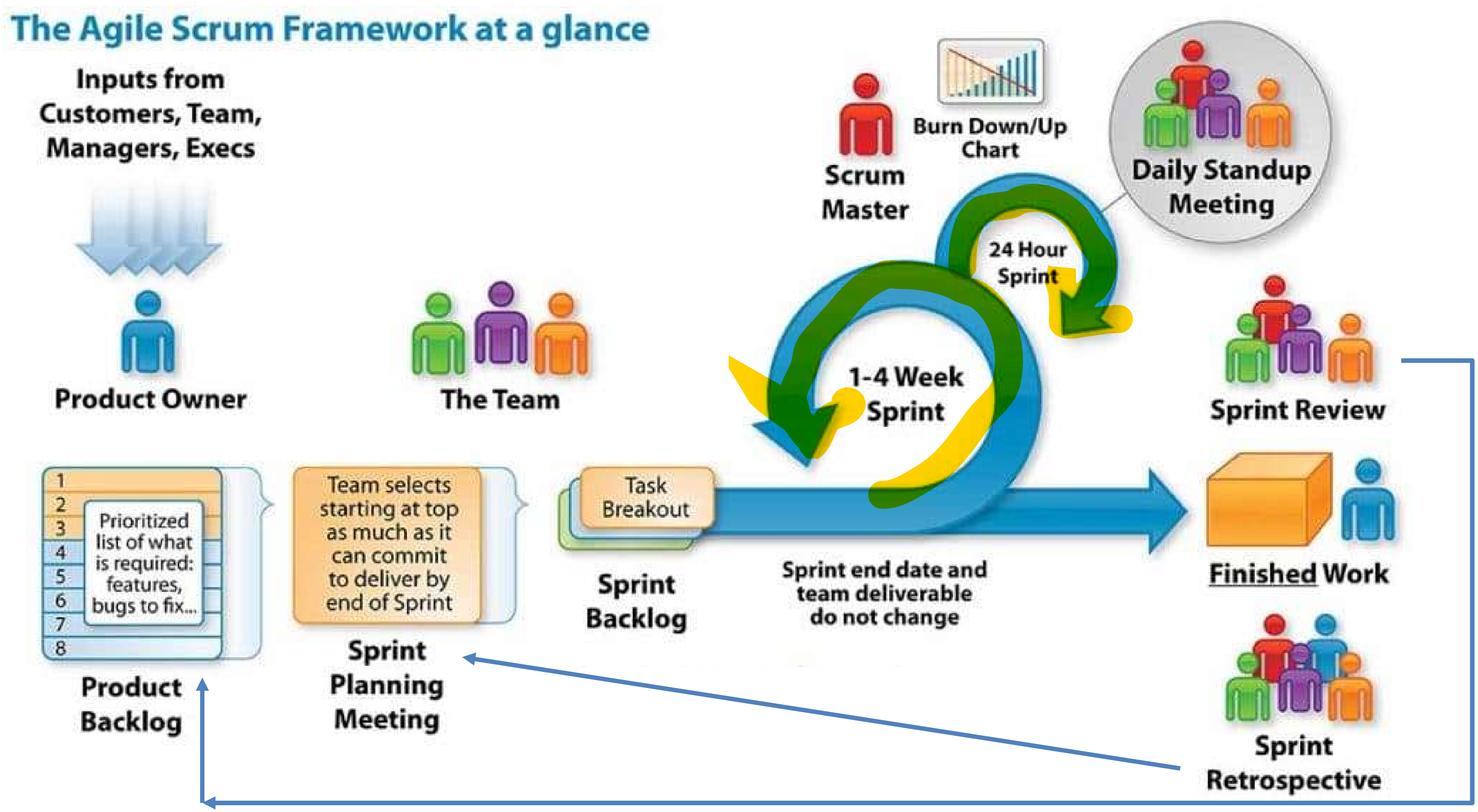
The Scrum approach is a general agile method but its focus is on managing iterative development rather than specific agile practices. **There are three phases in Scrum.**

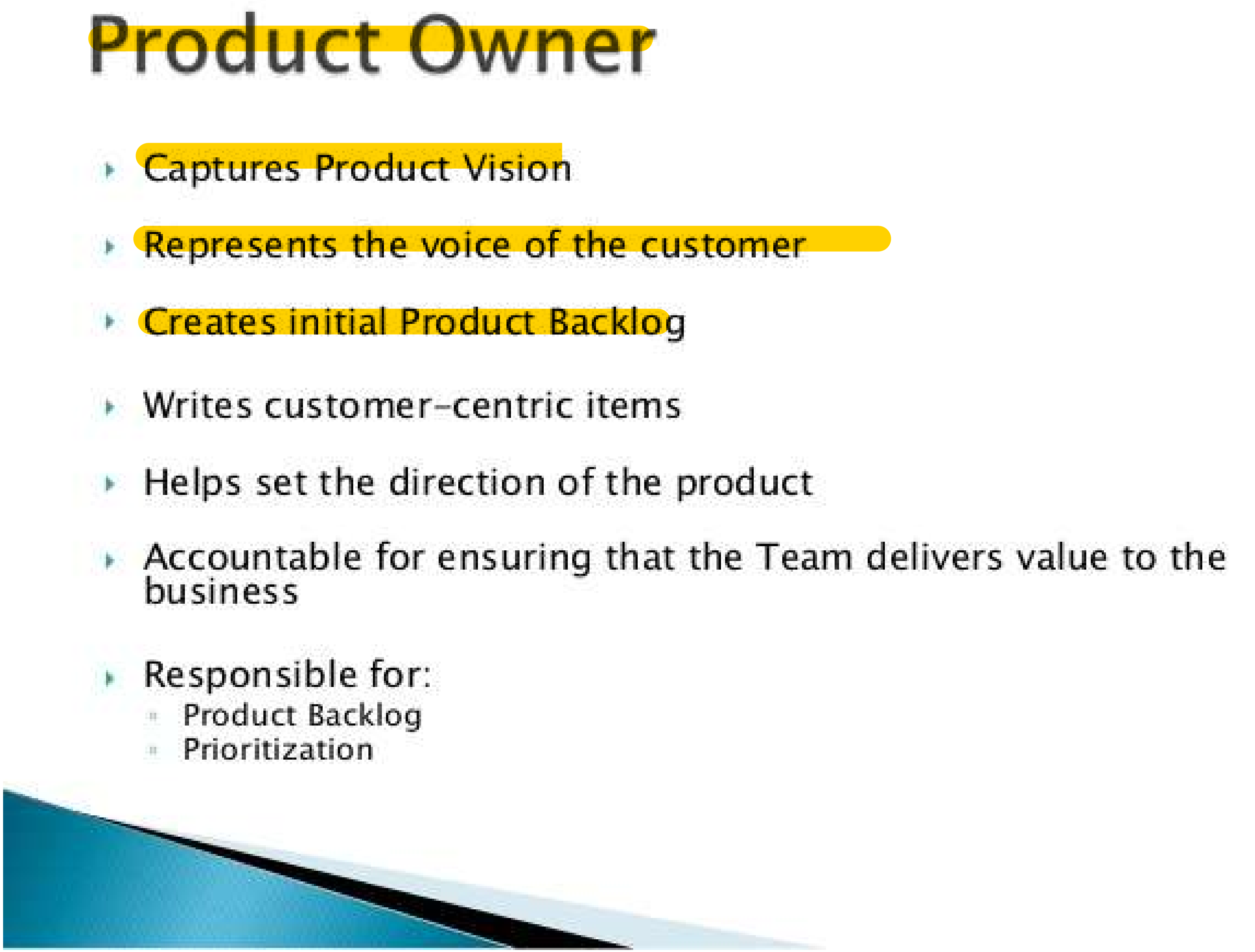
The initial phase is an **outline planning phase** where you establish the general objectives for the project and design the software architecture.

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.

The Scrum process

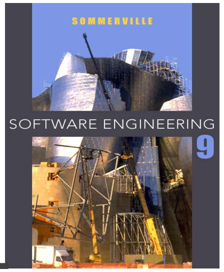

The Agile Scrum Framework at a glance
Inputs from Customers, Team, Managers, Execs
Product Owner
The Team
Scrum Master
Burn Down/Up Chart
Daily Standup Meeting
24 Hour Sprint
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Prioritized list of what is required: features, bugs to fix...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective

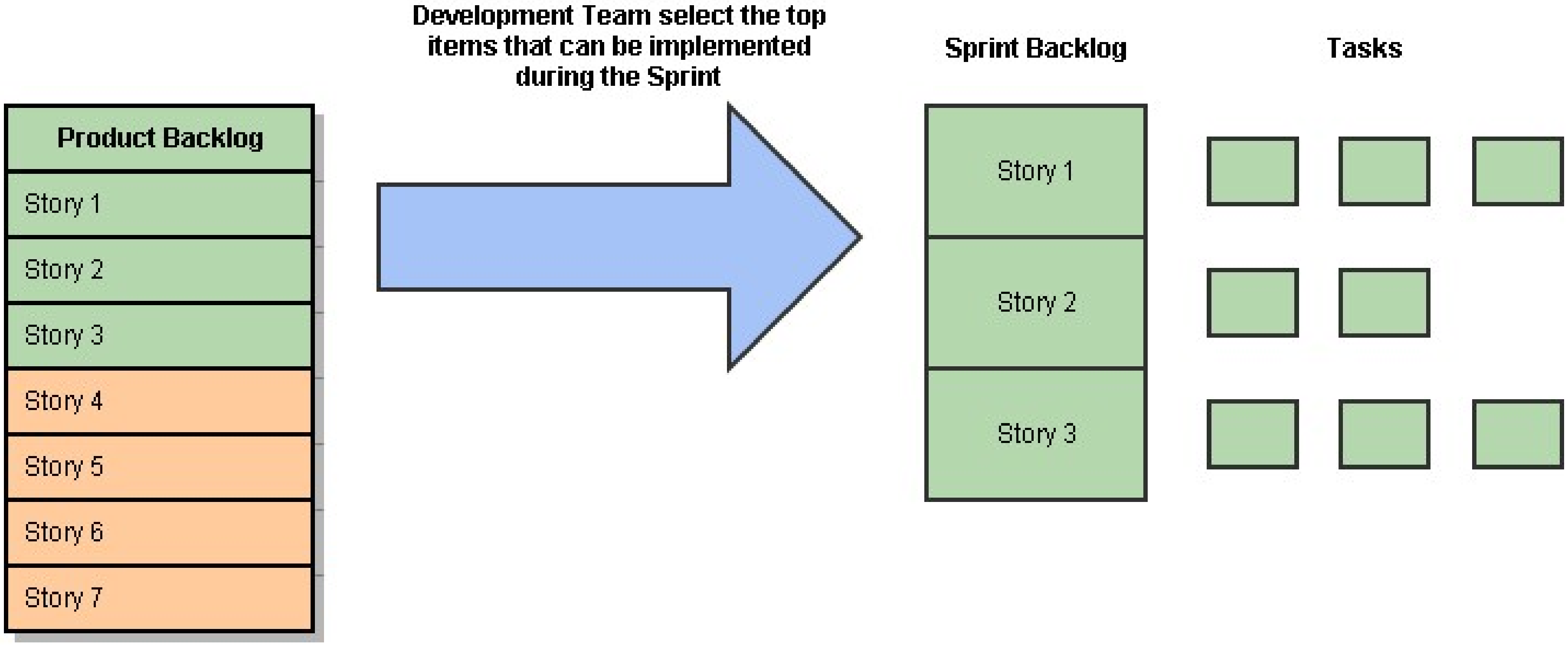


Product Owner

- Captures Product Vision
- Represents the voice of the customer
- Creates initial Product Backlog
- Writes customer-centric items
- Helps set the direction of the product
- Accountable for ensuring that the Team delivers value to the business
- Responsible for:
  - Product Backlog
  - Prioritization

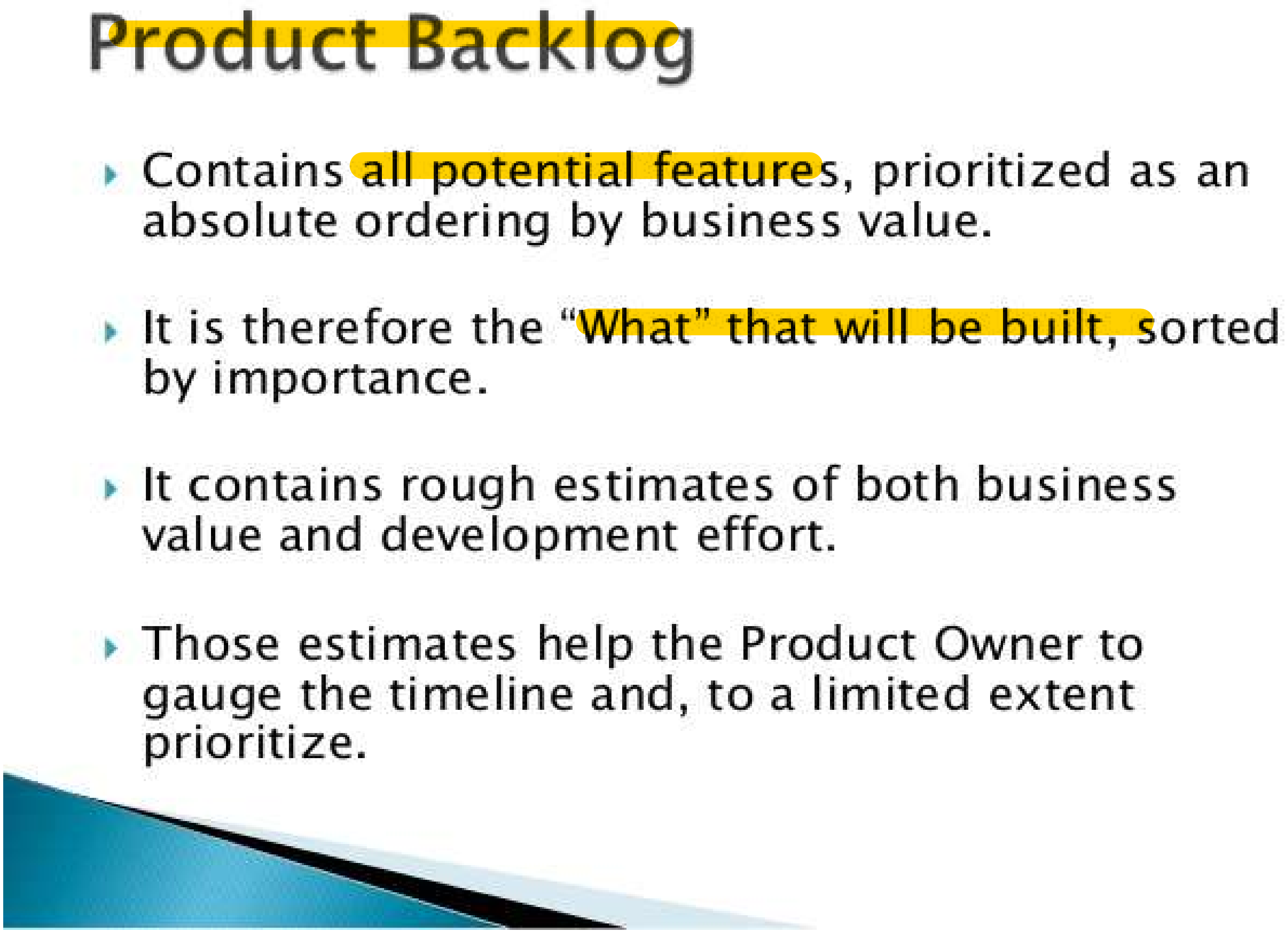
Product and Sprint Backlogs





Scrum Characteristics

- Simple and scaleable
- Empirical process
- Short-term detailed planning with constant feedback provides simple inspect and adapt cycle
- Simple techniques and work artifacts
- Requirements are captured as user stories in a list of product backlog
- Progress is made in Sprints
- Teams collaborating with the Product Owner
- Optimises working environment
- Reduces organisational overhead
- Detects everything that gets in the way of delivery
- Fosters openness and demands visibility



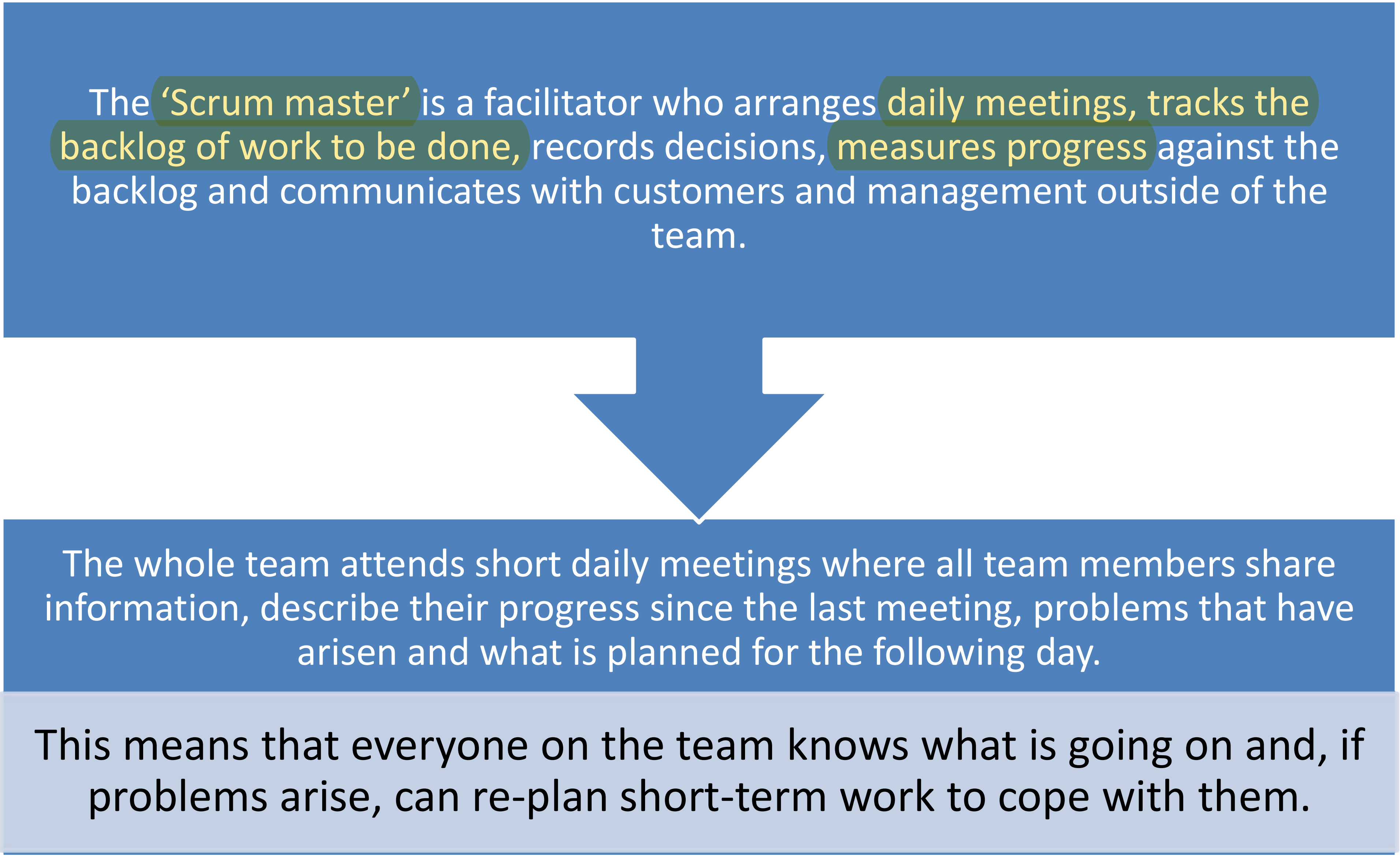
Product Backlog

- Contains all potential features, prioritized as an absolute ordering by business value.
- It is therefore the “What” that will be built, sorted by importance.
- It contains rough estimates of both business value and development effort.
- Those estimates help the Product Owner to gauge the timeline and, to a limited extent prioritize.

Sprint Backlog

- Break Release backlog items into several tasks
- each sprint should deliver a fully tested product with all the features of that sprint backlog 100% complete
- late finish of the sprint is a great indicator that the project is not on schedule

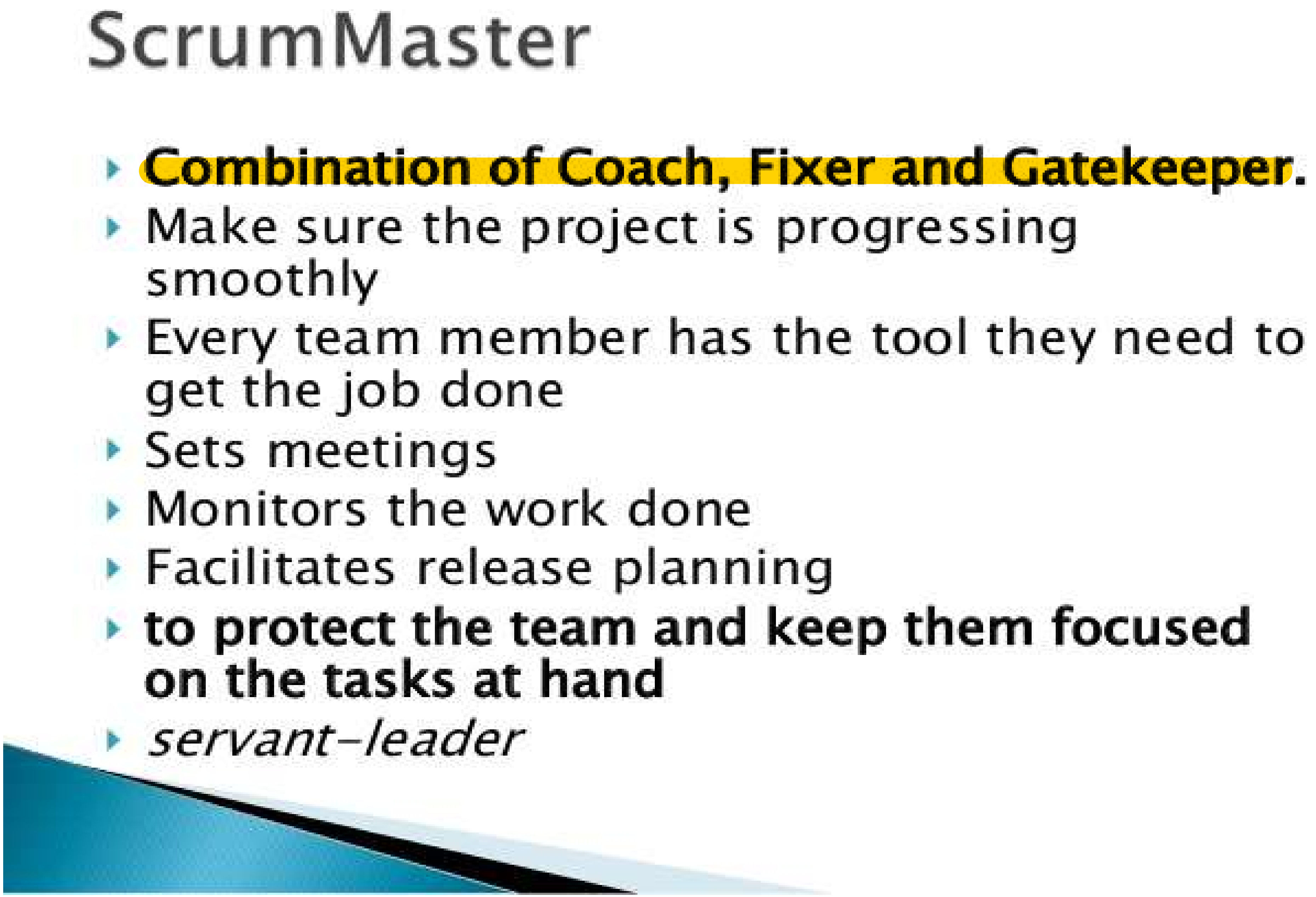
Teamwork in Scrum



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 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.

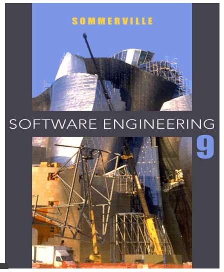
This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.



ScrumMaster

- **Combination of Coach, Fixer and Gatekeeper.**
- Make sure the project is progressing smoothly
- Every team member has the tool they need to get the job done
- Sets meetings
- Monitors the work done
- Facilitates release planning
- **to protect the team and keep them focused on the tasks at hand**
- *servant-leader*

The Sprint cycle

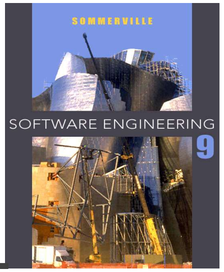


Sprints are fixed length, normally 1–4 weeks. They correspond to the development of a release of the system in XP.

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 to be developed during the sprint.

The Sprint cycle



Once these are agreed, the team organize themselves to develop the software.

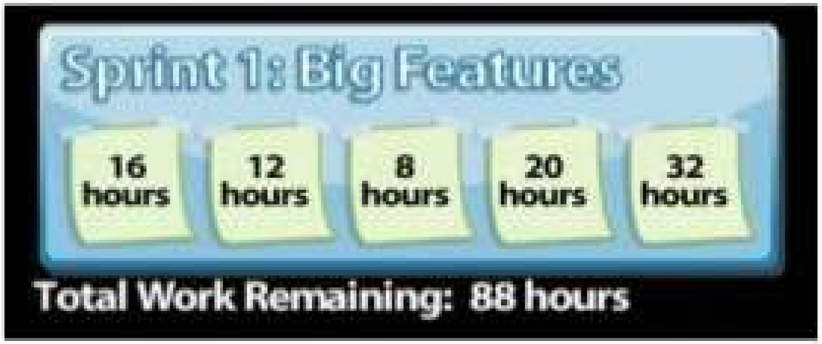
During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called ‘Scrum master’.

The role of the Scrum master is to protect the development team from external distractions.

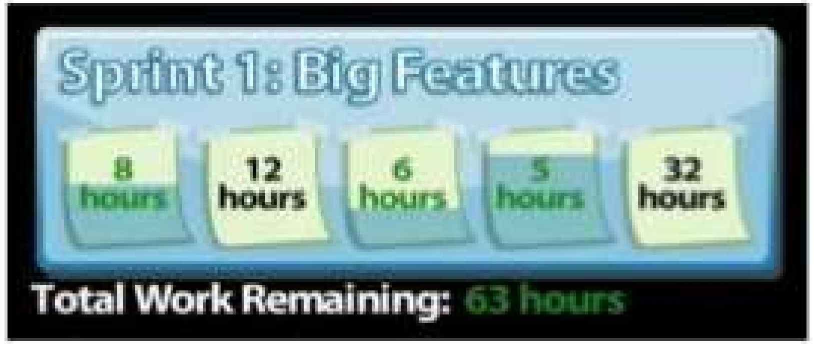
At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

Burndown Chart (Contd.)

-

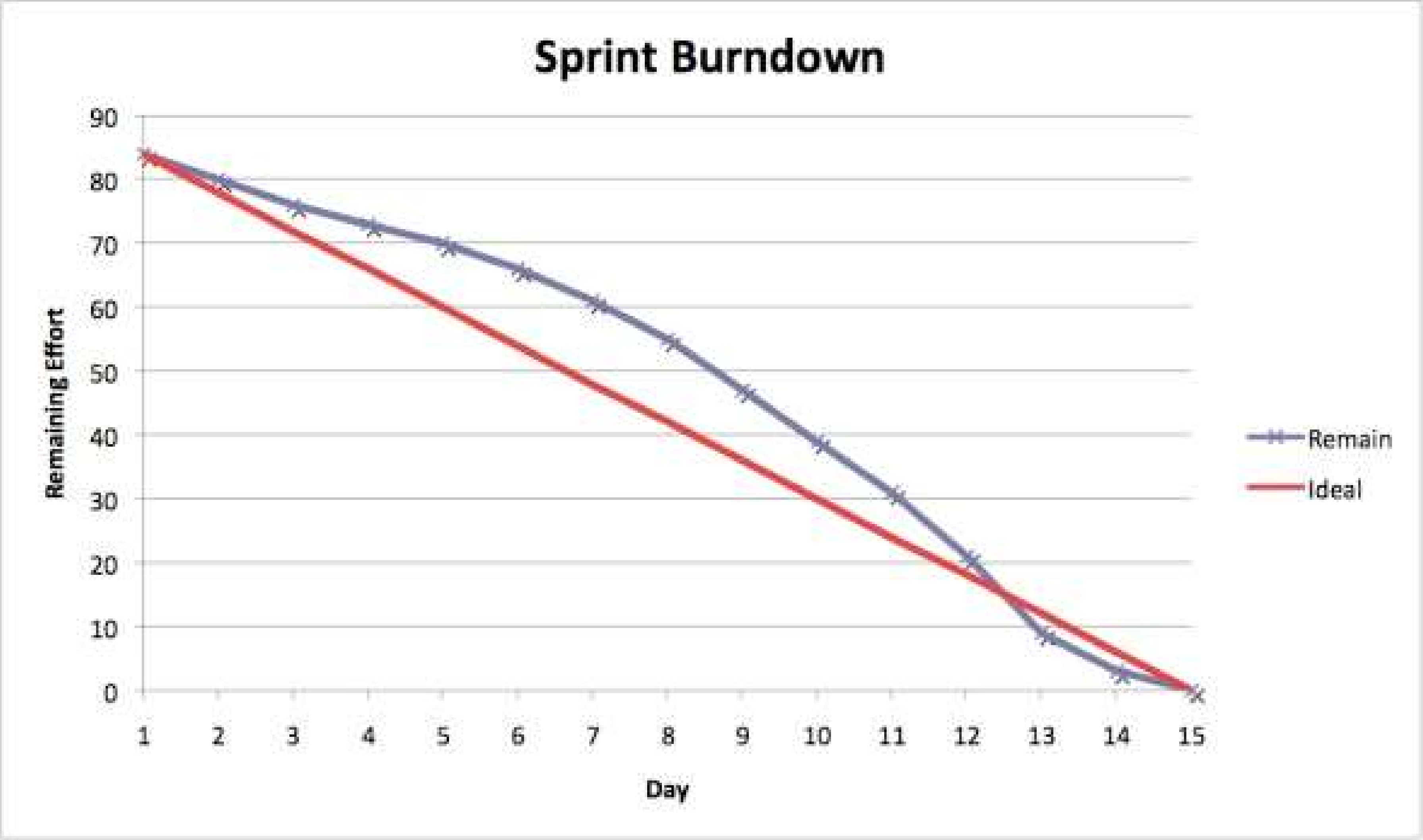


Actual estimates for each feature in the backlog during the initial planning process

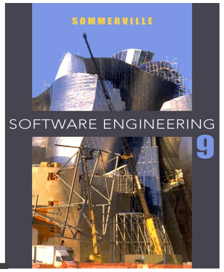


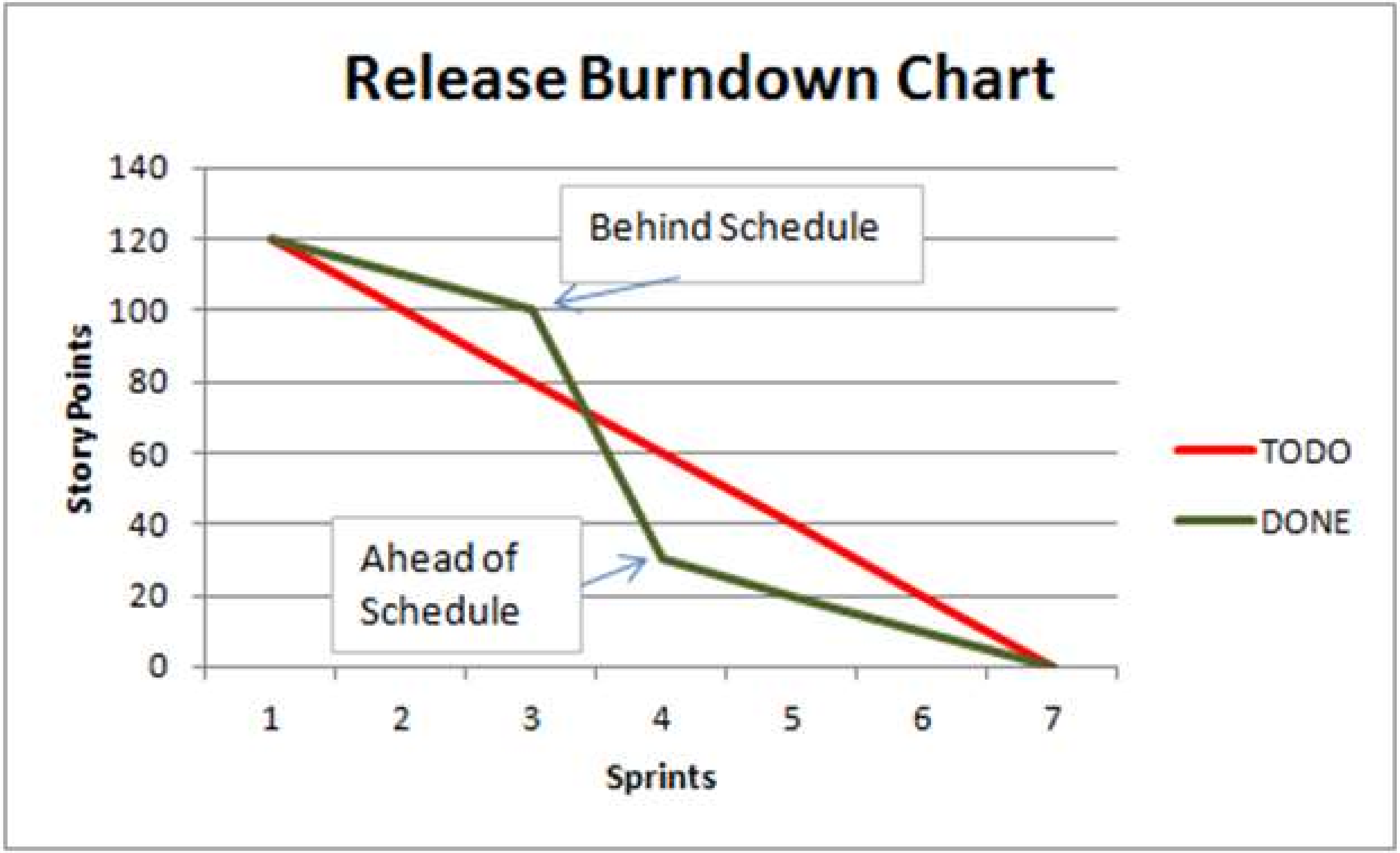
Daily progress on one ore more features is updated by team member by updating the amount of time remaining for each of their items

Sprint burndown chart

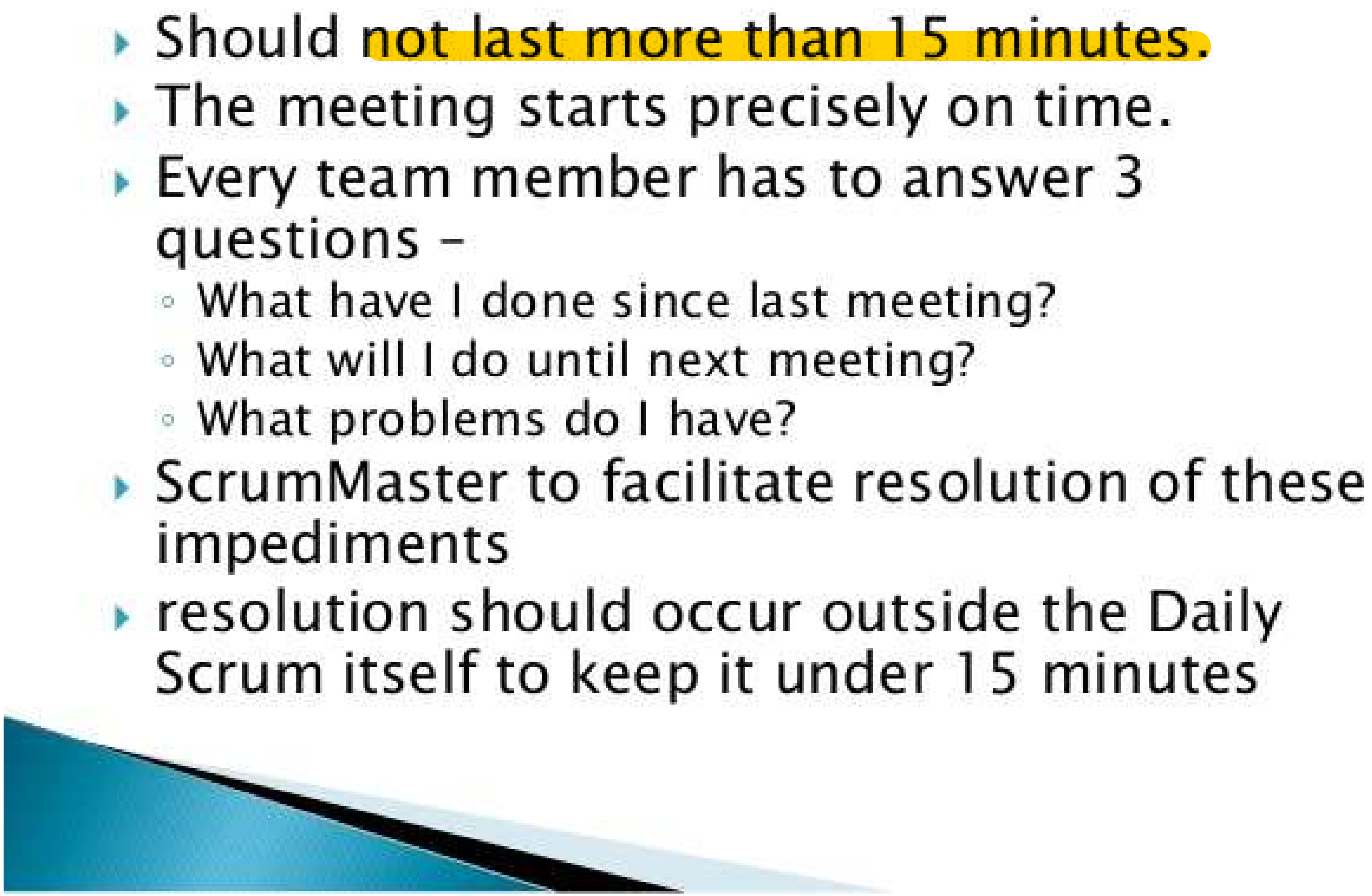


Burndown chart (Release)



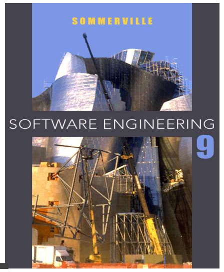


Stand-up meetings:
Daily Scrum



- Should not last more than 15 minutes.
- The meeting starts precisely on time.
- Every team member has to answer 3 questions –
  - What have I done since last meeting?
  - What will I do until next meeting?
  - What problems do I have?
- ScrumMaster to facilitate resolution of these impediments
- resolution should occur outside the Daily Scrum itself to keep it under 15 minutes

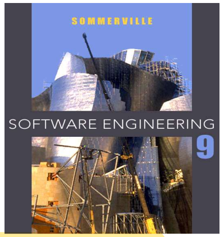
Sprint Review x Sprint Retrospective

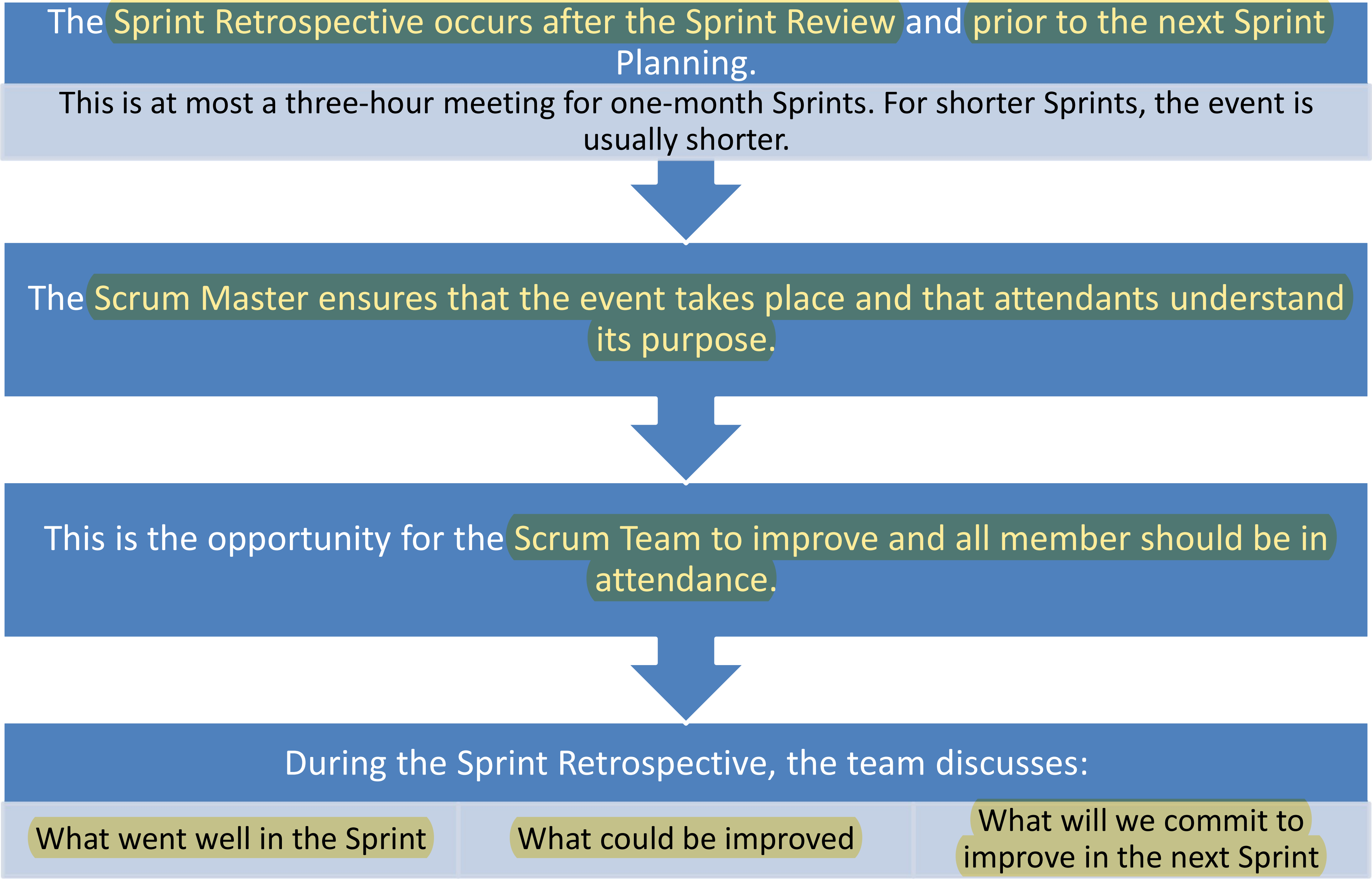


The **Sprint Review** is equivalent to a user acceptance test. It is where the project team demonstrates the results of the work that they have done in the sprint and the Product Owner and any required stakeholders accept the work or not.

A **Sprint Retrospective** is equivalent to a project post-mortem except that it is done at the end of a sprint. The purpose of the meeting is to reflect on what went well and what didn't go well in the previous sprint and determine how it can be improved in the next sprint.

Sprint retrospective





The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning.

This is at most a three-hour meeting for one-month Sprints. For shorter Sprints, the event is usually shorter.

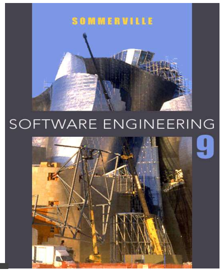
The Scrum Master ensures that the event takes place and that attendants understand its purpose.

This is the opportunity for the Scrum Team to improve and all member should be in attendance.

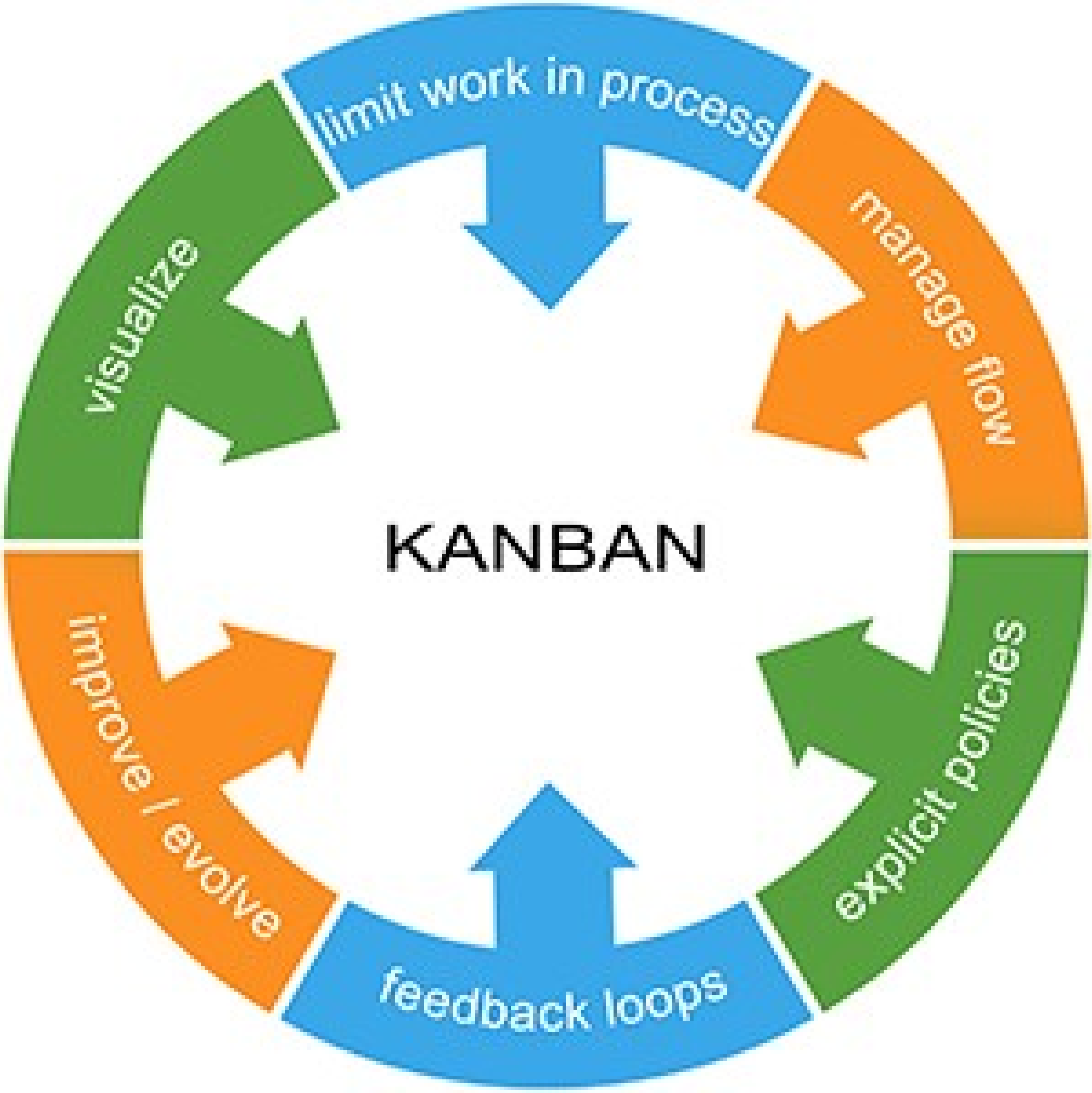
During the Sprint Retrospective, the team discusses:

- What went well in the Sprint
- What could be improved
- What will we commit to improve in the next Sprint

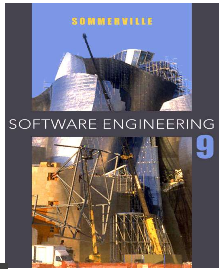
Scrum benefits



- 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.


limit work in process
manage flow
explicit policies
feedback loops
improve / evolve
visualize
KANBAN

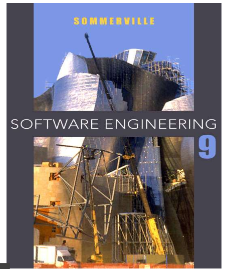
Scrum x Kanban

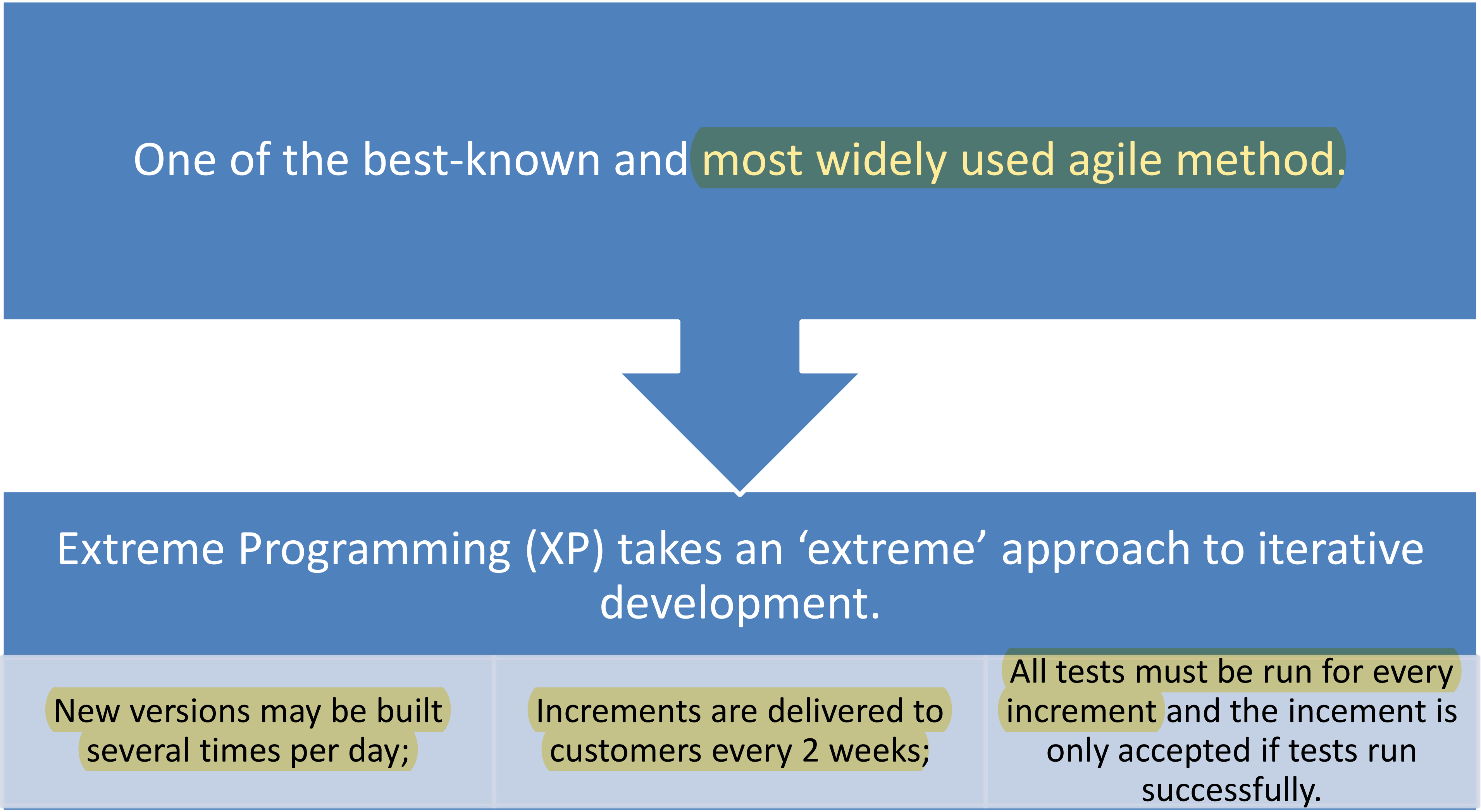


✧ https://www.youtube.com/watch?v=HNd1_irOL5k

Extreme Programming

Extreme programming





One of the best-known and most widely used agile method.

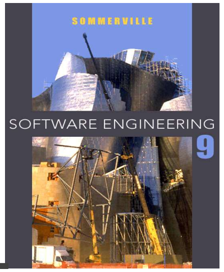
Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.

New versions may be built several times per day;

Increments are delivered to customers every 2 weeks;

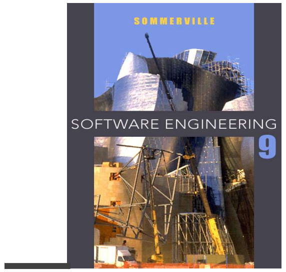
All tests must be run for every increment and the incement is only accepted if tests run successfully.

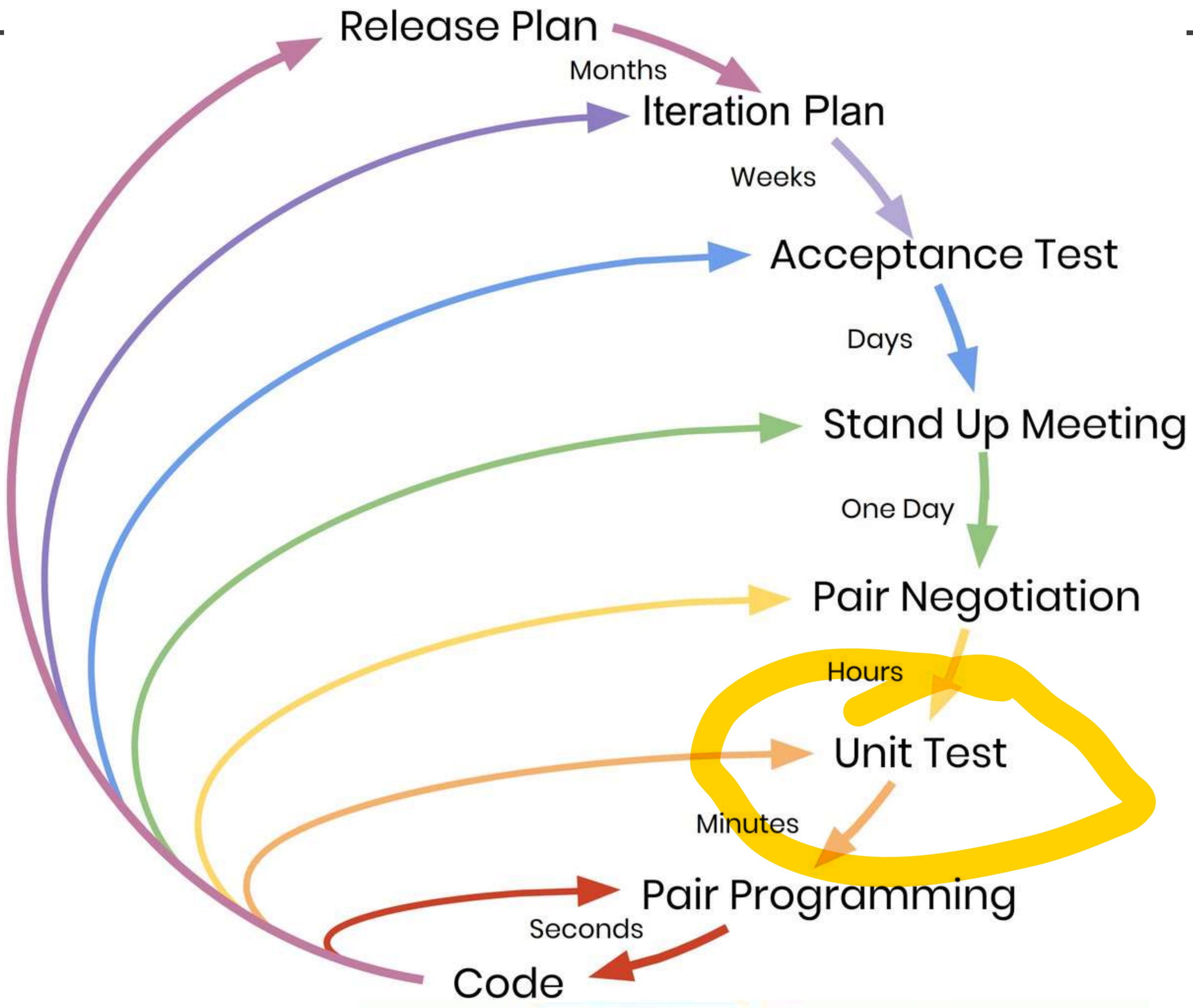
XP and agile principles



- 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.

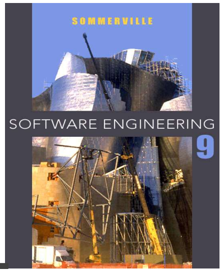
XP Process

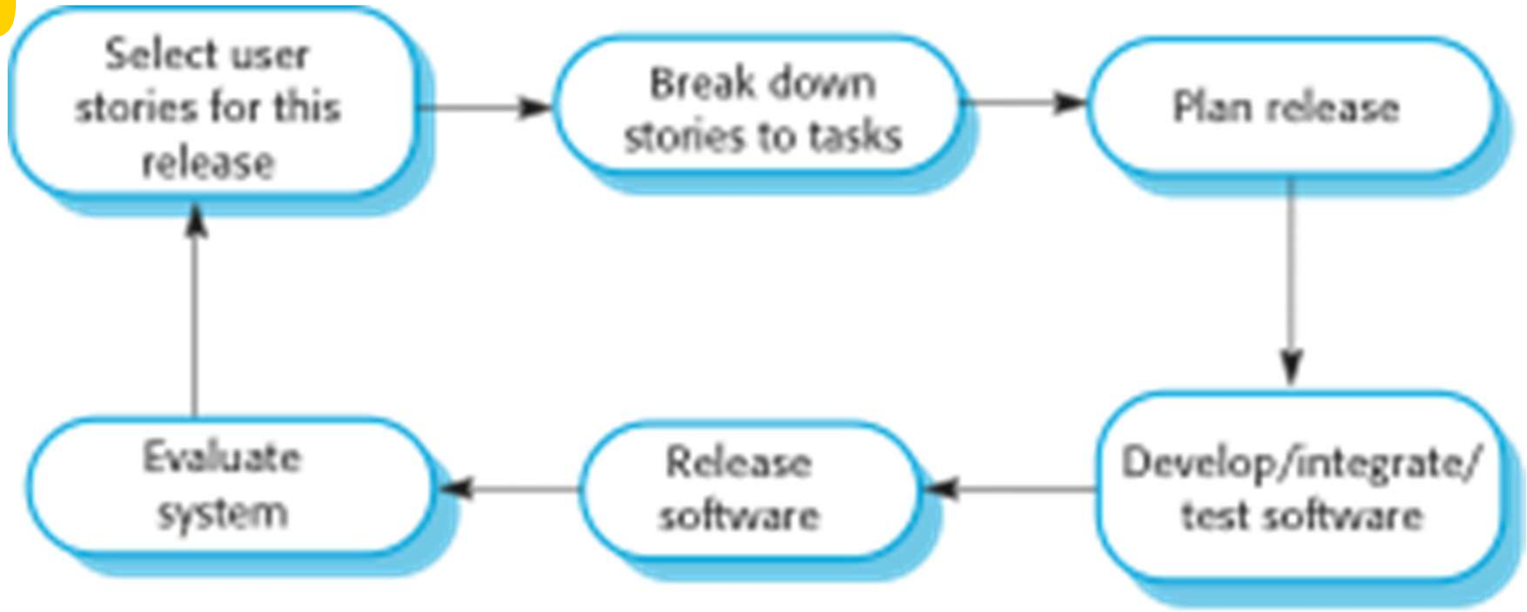




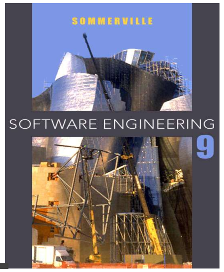
© J. Donovan Wells

The extreme programming release cycle



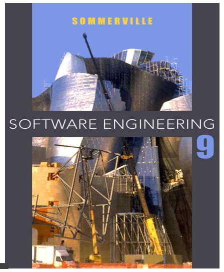

Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

Extreme programming practices (a)



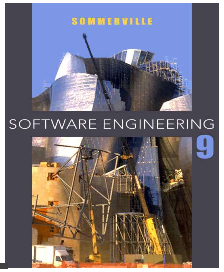
Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.
Small releases	The minimal useful 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.

Requirements scenarios



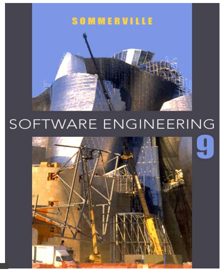
In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.

User requirements are expressed as scenarios or user stories.

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.

XP and change

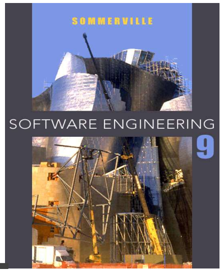


Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.

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

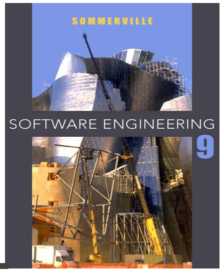


Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.

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.

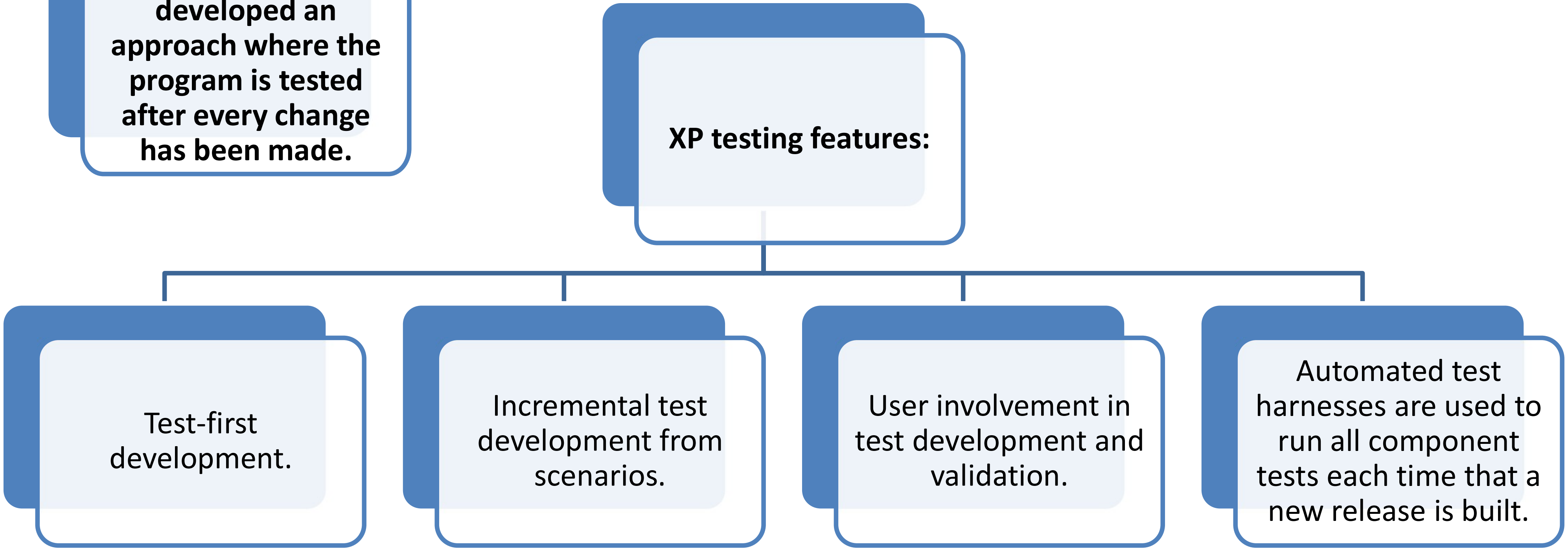


Examples of refactoring

- 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.

Testing in XP

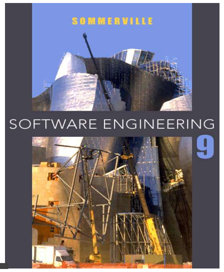
Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.



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-first development



Writing tests before code clarifies the requirements to be implemented.

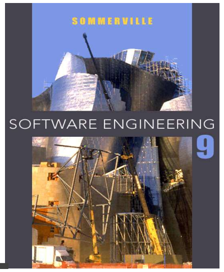
Tests are written as programs rather than data so that they can be executed automatically.

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



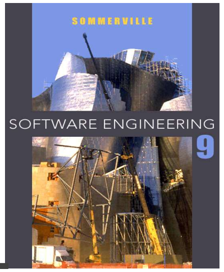
The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.

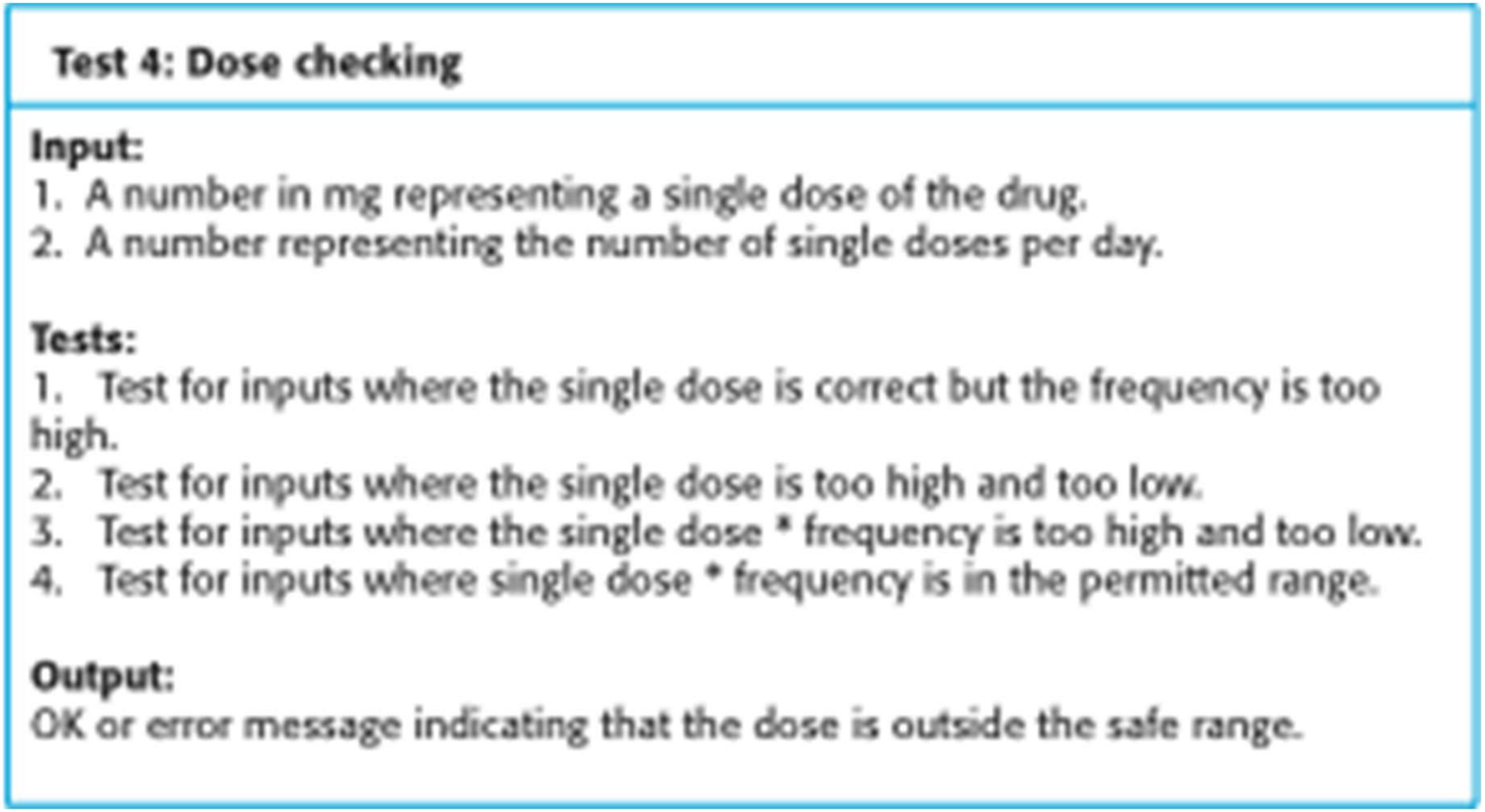
The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.

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.

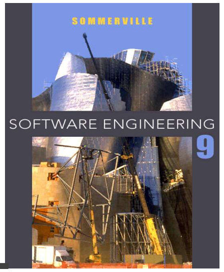
Tests:

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



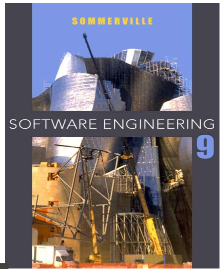
Test automation means that tests are written as executable components before the task is implemented

- 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.

XP testing difficulties

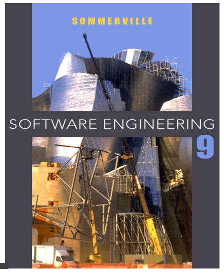


Programmers prefer programming to testing and sometimes they take short cuts when writing tests. For example, they may write **incomplete tests that do not check for all possible exceptions** that may occur.

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 is 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



IN XP, PROGRAMMERS WORK IN PAIRS, SITTING TOGETHER TO DEVELOP CODE.

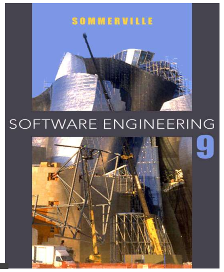
THIS HELPS DEVELOP COMMON OWNERSHIP OF CODE AND SPREADS KNOWLEDGE ACROSS THE TEAM.

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 THIS.

MEASUREMENTS SUGGEST THAT DEVELOPMENT PRODUCTIVITY WITH PAIR PROGRAMMING IS SIMILAR TO THAT OF TWO PEOPLE WORKING INDEPENDENTLY.

Pair programming



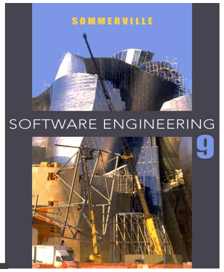
In pair programming, programmers sit together at the same workstation to develop the software.

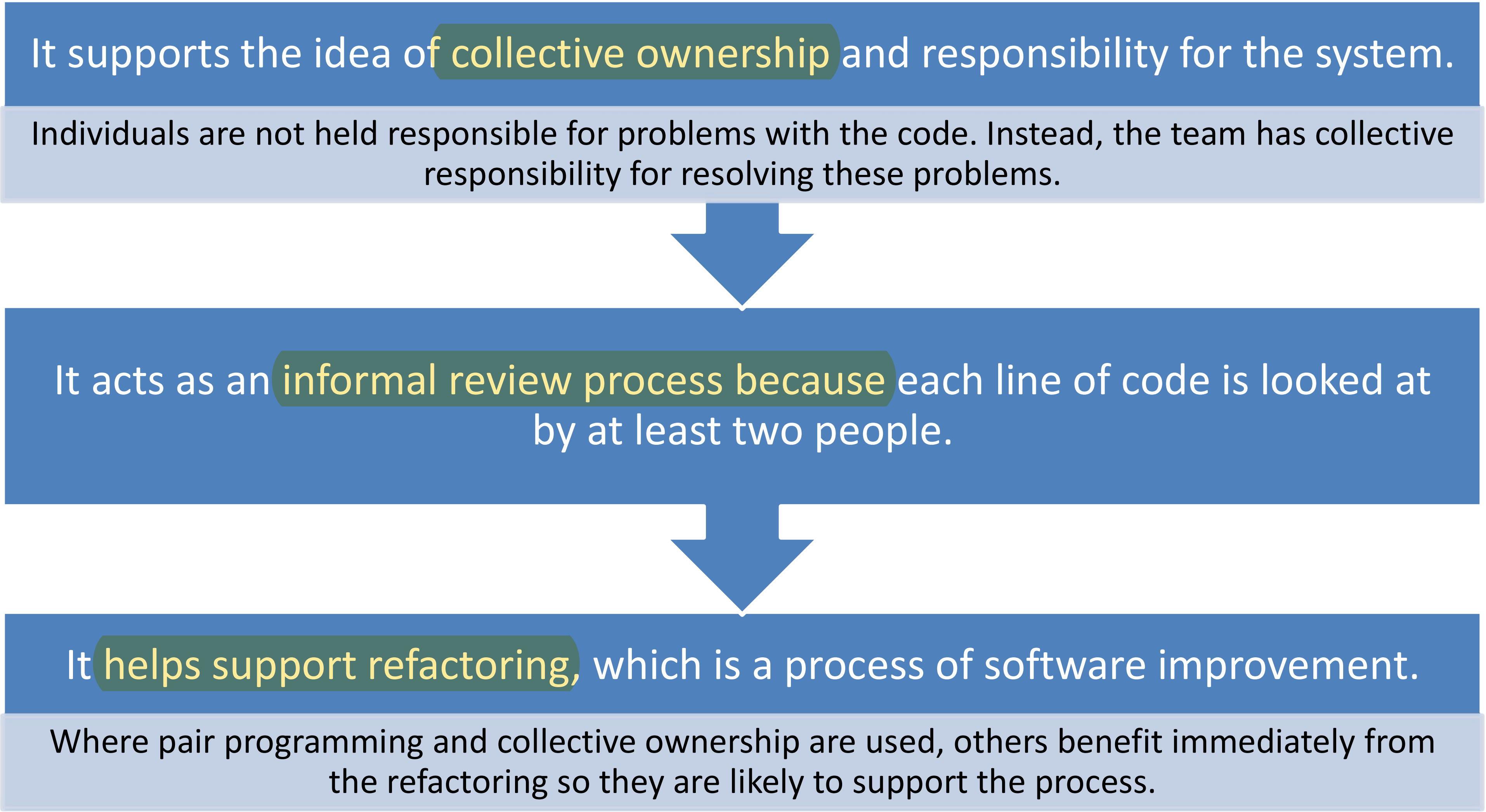
Pairs are created dynamically so that all team members work with each other during the development process.

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 evidence that a pair working together is more efficient than 2 programmers working separately.

Advantages of pair programming





It supports the idea of collective ownership and responsibility for the system.

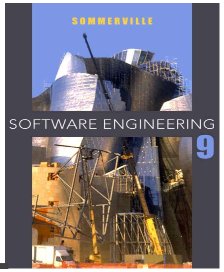
Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.

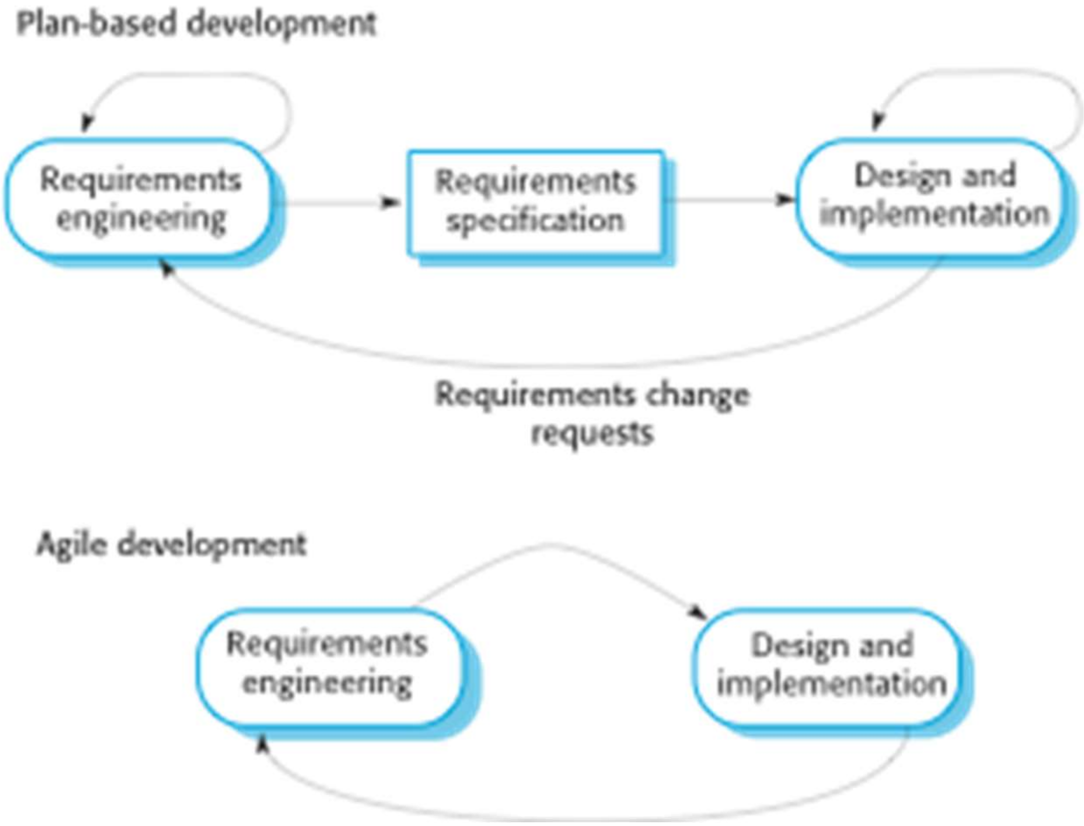
It acts as an informal review process because each line of code is looked at by at least two people.

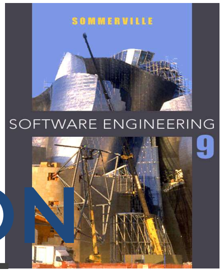
It helps support refactoring, which is a process of software improvement.

Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

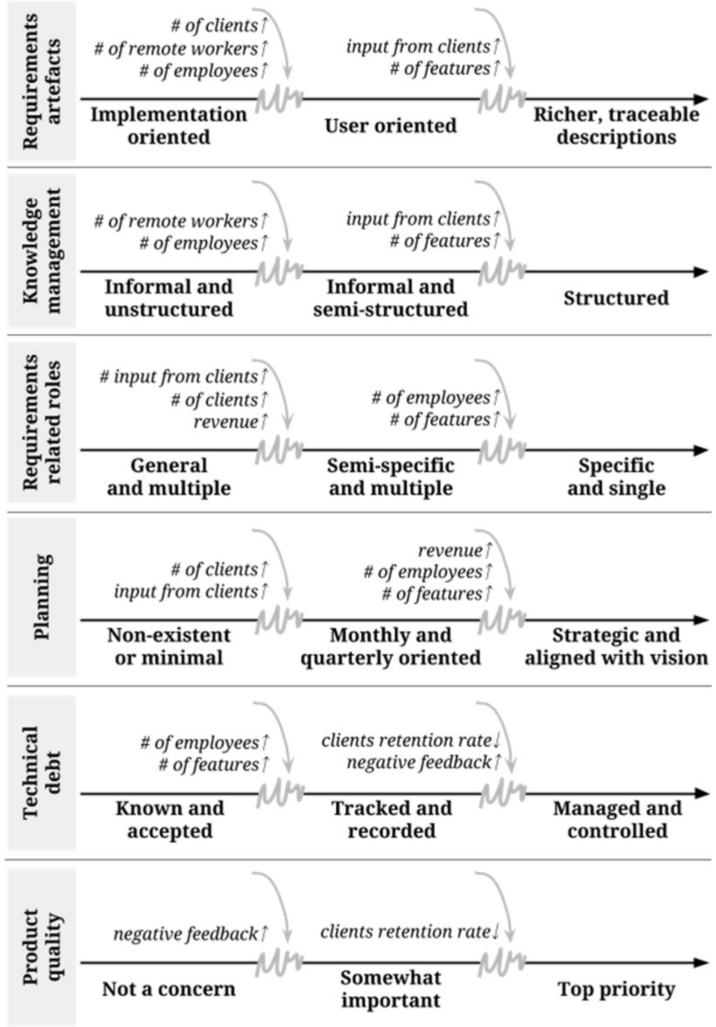
Plan-driven and agile specification




Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

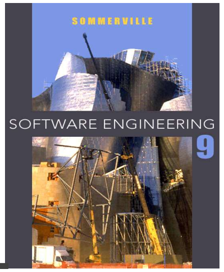


REQUIREMENTS PRACTICE EVOLUTION



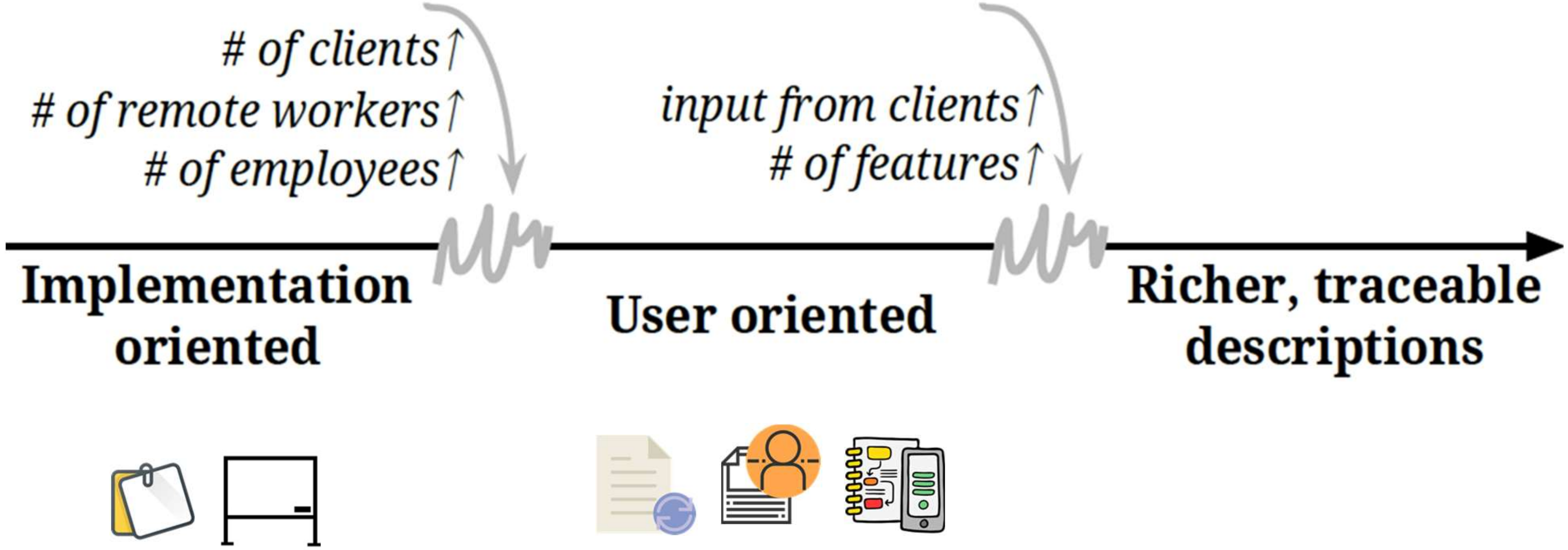
Requirements artefacts	Implementation oriented	# of clients↑ # of remote workers↑ # of employees↑	User oriented	input from clients↑ # of features↑	Richer, traceable descriptions
Knowledge management	Informal and unstructured	# of remote workers↑ # of employees↑	Informal and semi-structured	input from clients↑ # of features↑	Structured
Requirements related roles	General and multiple	# input from clients↑ # of clients↑ revenue↑	Semi-specific and multiple	# of employees↑ # of features↑	Specific and single
Planning	Non-existent or minimal	# of clients↑ input from clients↑	Monthly and quarterly oriented	revenue↑ # of employees↑ # of features↑	Strategic and aligned with vision
Technical debt	Known and accepted	# of employees↑ # of features↑	Tracked and recorded	clients retention rate↓ negative feedback↑	Managed and controlled
Product quality	Not a concern	negative feedback↑	Somewhat important	clients retention rate↓	Top priority

Catarina Gralha, Daniela E. Damian, Anthony I. Wasserman, Miguel Goulão, João Araújo:
The evolution of requirements practices in software startups. ICSE 2018: 823-833



REQUIREMENTS ARTEFACTS:

content of information and **user** orientation matters



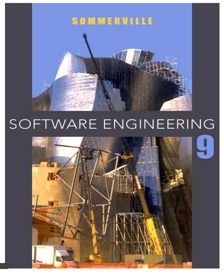
# of clients ↑
# of remote workers ↑
# of employees ↑

input from clients ↑
# of features ↑

Implementation oriented

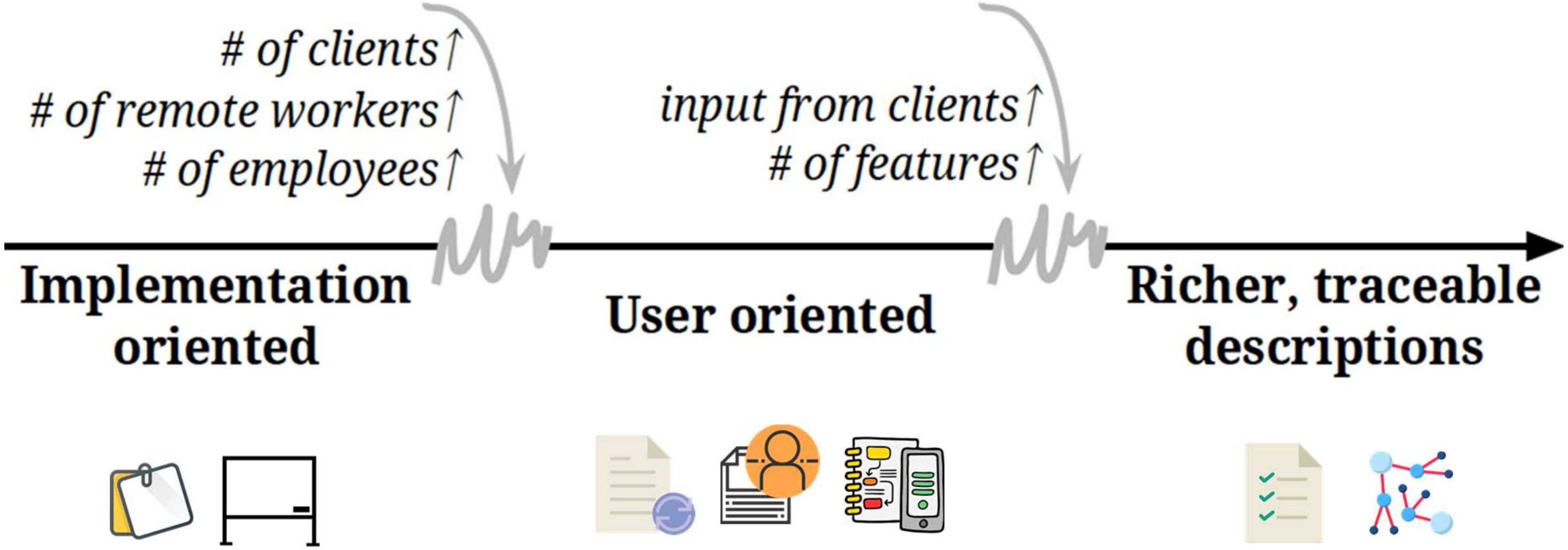
User oriented

Richer, traceable descriptions



REQUIREMENTS ARTEFACTS:

content of information and user orientation matters



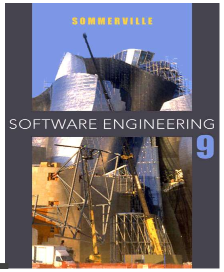
# of clients ↑
# of remote workers ↑
# of employees ↑

input from clients ↑
# of features ↑

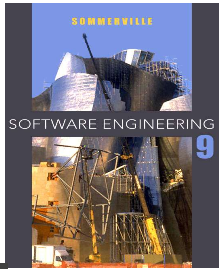
Implementation oriented

User oriented

Richer, traceable descriptions

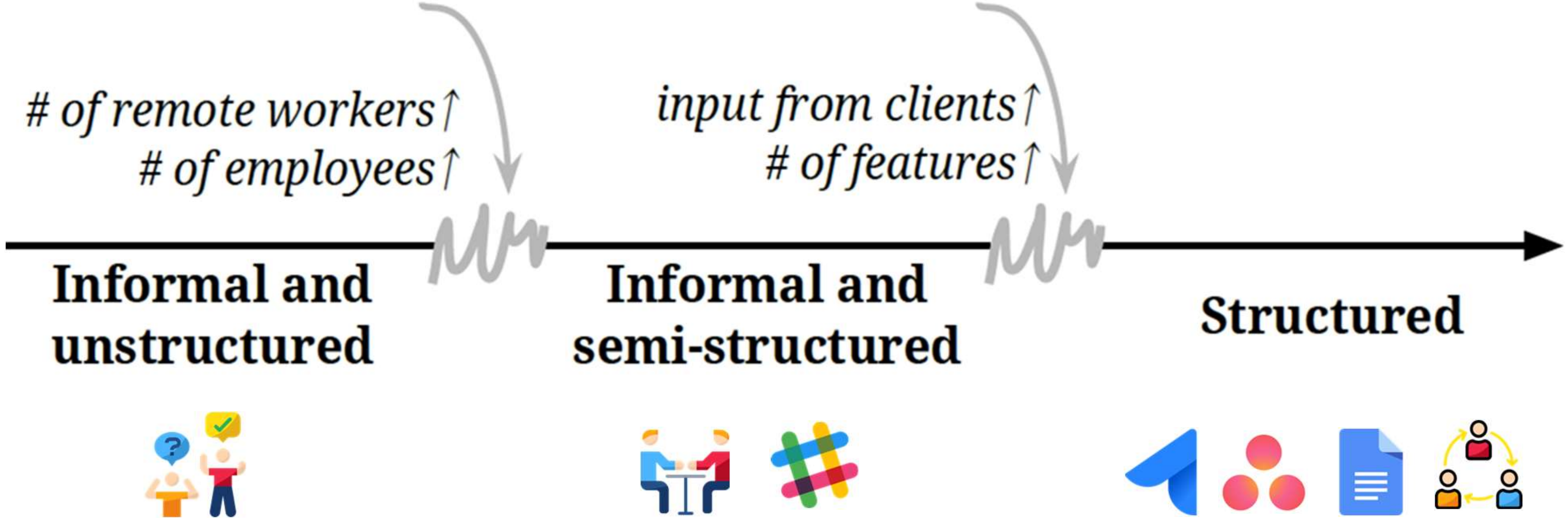


“It was mainly because we had more and more ***clients*** *(…) We need to know their* ***needs*** *when we are writing code, so (…)* ***user stories*** *are important”*

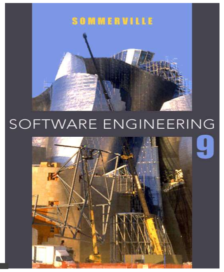


KNOWLEDGE MANAGEMENT:

project communication and documentation matters

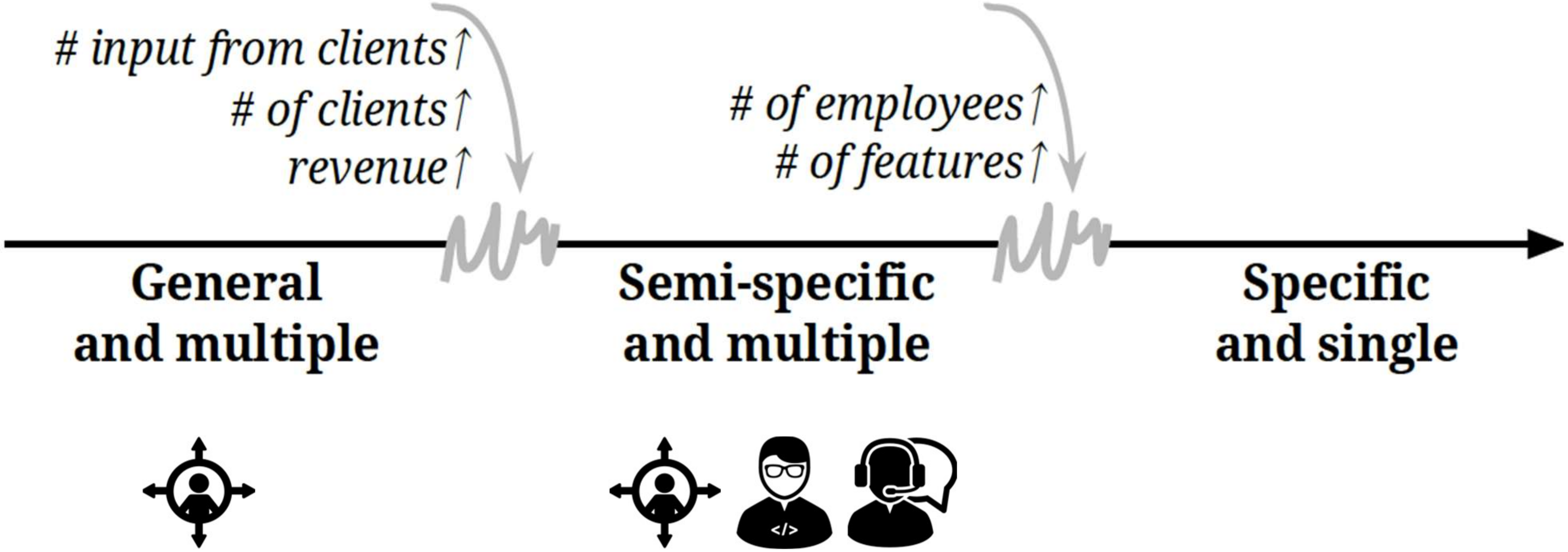


“It’s possible to have good practices and improve the ***knowledge dissemination*** *earlier because the tools are there (…) but there were more important things to do.”*



REQUIREMENTS-RELATED ROLES:

focusing on **customer-facing** roles matters



# input from clients↑
# of clients↑
revenue↑

# of employees↑
# of features↑

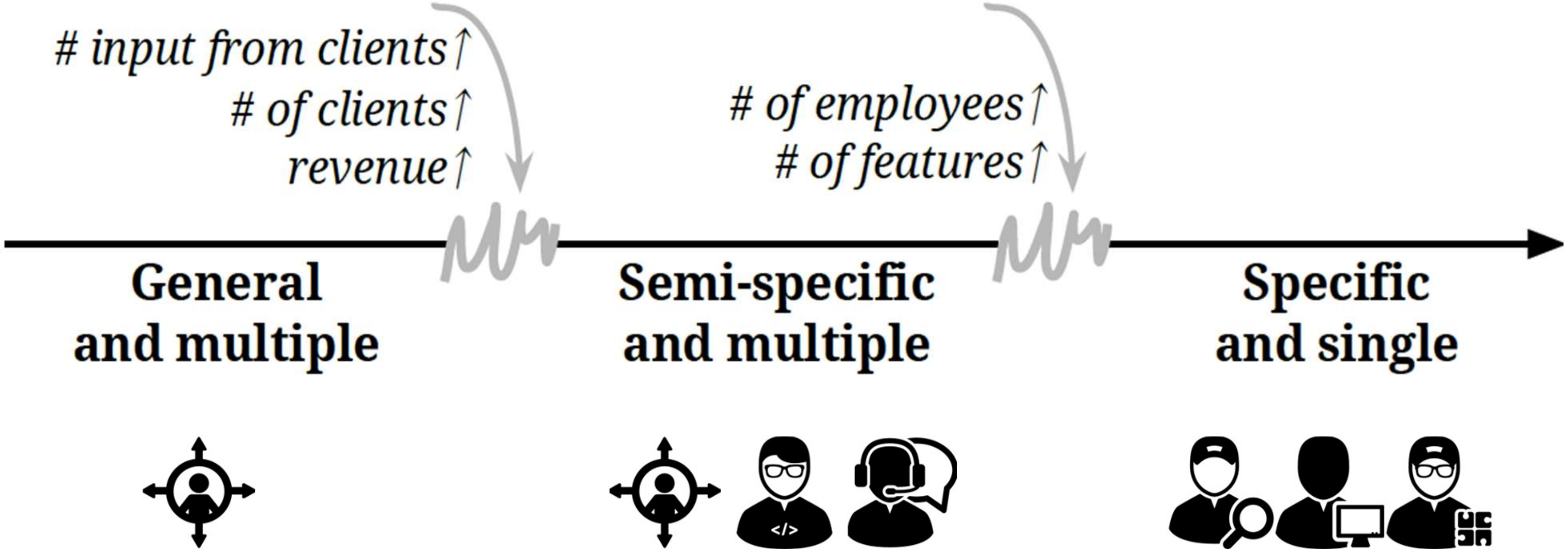
General and multiple

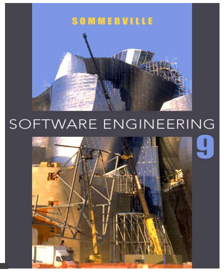
Semi-specific and multiple

Specific and single

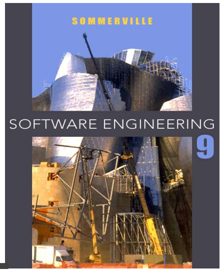
REQUIREMENTS-RELATED ROLES:

focusing on **customer-facing** roles matters



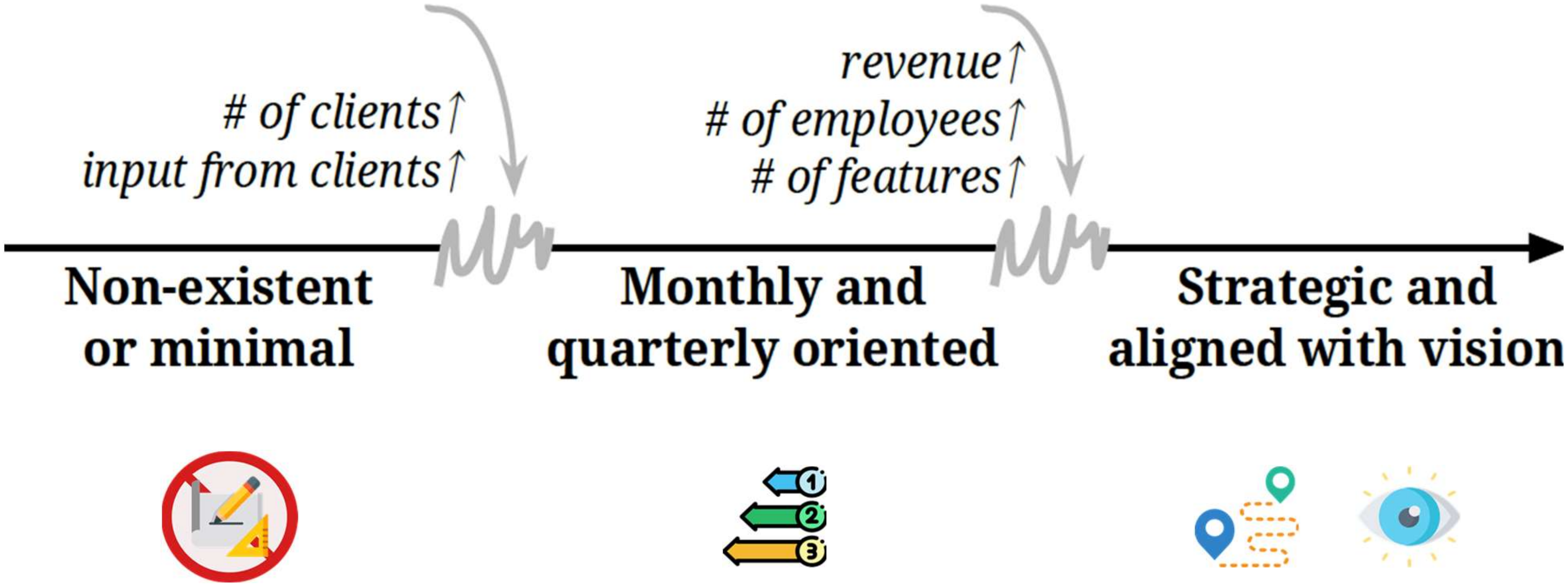


"We started hiring more people for specific roles. We had ***developers*** *(...) we hired a* ***client success manager*** *to stay on track of all of our clients. We still need to be more specialised."*



PLANNING:

company / product vision and alignment with it matters



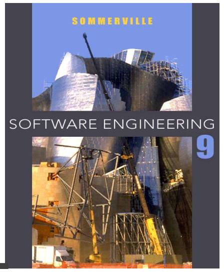
# of clients↑
input from clients↑

revenue↑
# of employees↑
# of features↑

Non-existent or minimal

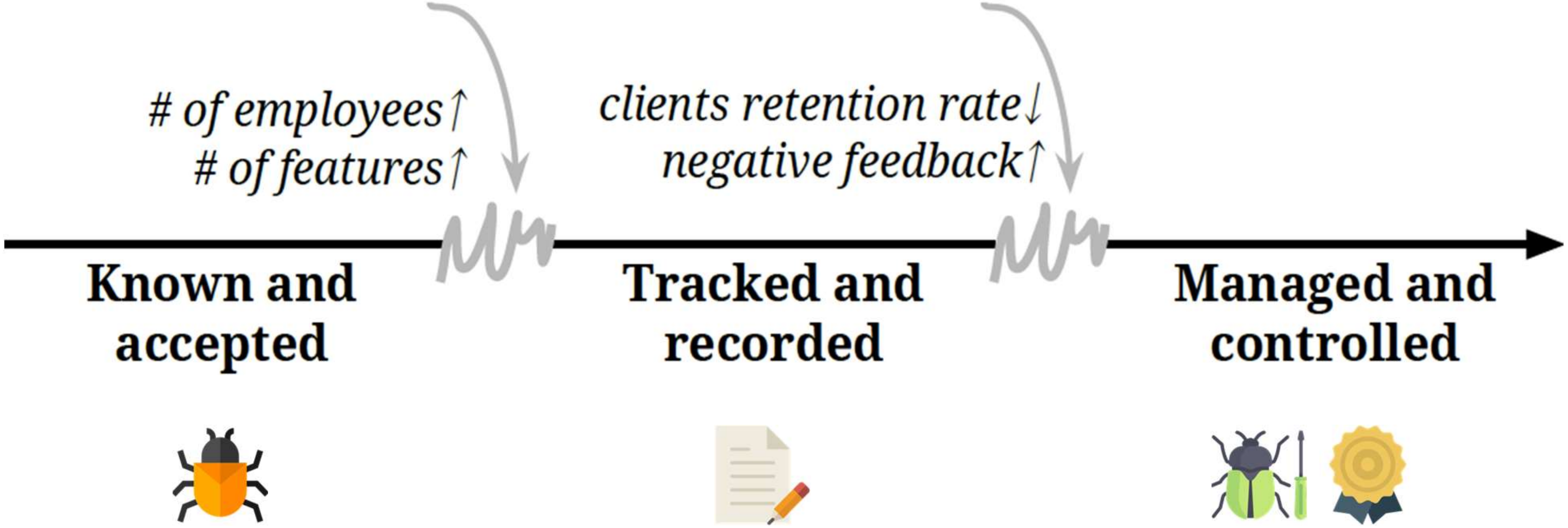
Monthly and quarterly oriented

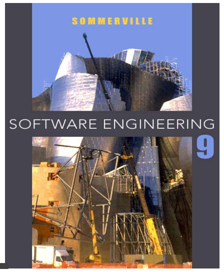
Strategic and aligned with vision



TECHNICAL DEBT:

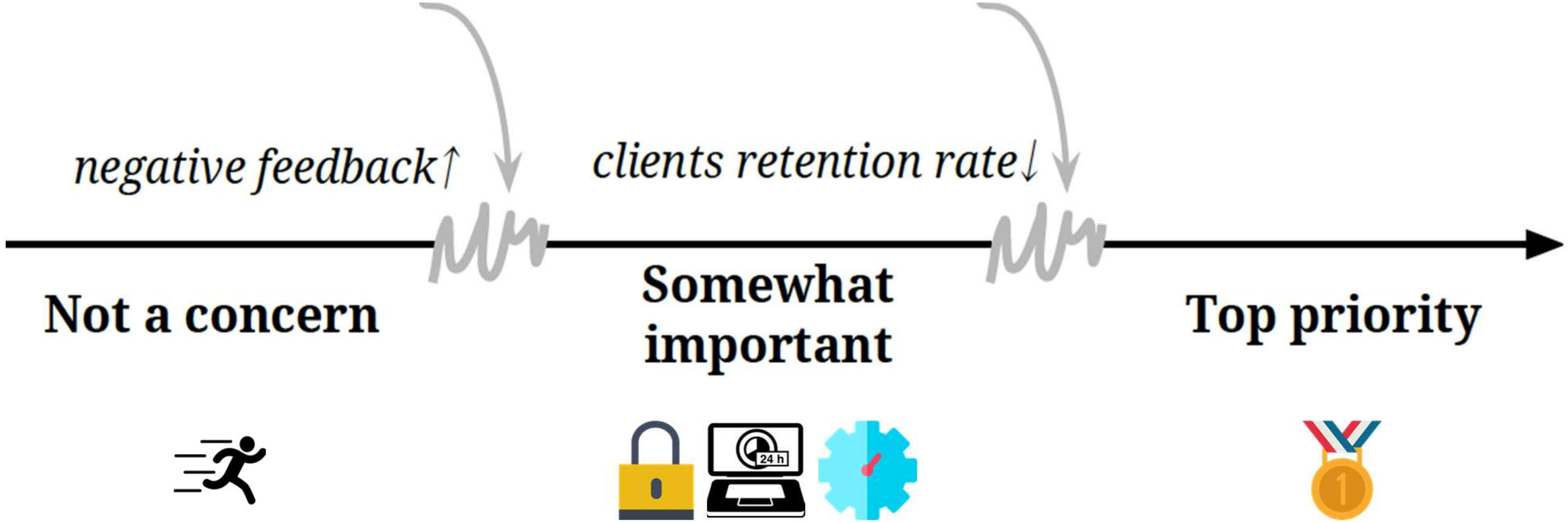
understanding the impact of technical debt in product / service matters





PRODUCT QUALITY:

tradeoffs between quality and speed start to matter



negative feedback↑

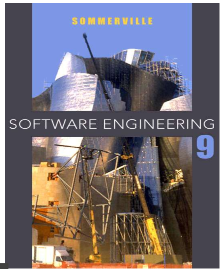
clients retention rate↓

Not a concern

Somewhat important

Top priority

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 particular strengths of agile methods.