Software Engineering: A Practitioner's Approach, 8/e

Chapter 1
Software and Software Engineering

Dr Tripty Singh

Proposed Evaluation Pattern

- Internal– 70 marks
  - Continuous assessment : 50 marks
    - 2 Quizzes : 20 marks
    - Project : 30 marks
      - Sprint 1 : 10 marks
      - Sprint 2 : 10 marks
      - Quiz : 10 marks
  - Periodical Test 1 : 30 marks reduced to 10 marks.
  - Periodical Test 2 : 30 marks reduced to 10 marks.
- External-30 marks
  - End Semester : 50 marks reduced to 30 marks.

Software's Dual Role

- Software is a product
  - Delivers computing potential
  - Produces, manages, acquires, modifies, displays, or transmits information
- Software is a vehicle for delivering a product
  - Supports or directly provides system functionality
  - Controls other programs (e.g., an operating system)
  - Effects communications (e.g., networking software)
  - Helps build other software (e.g., software tools)

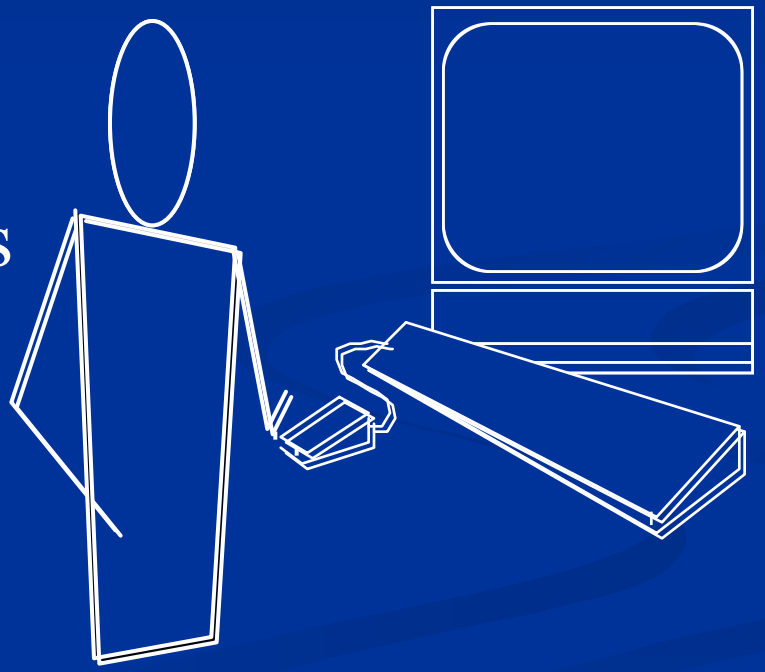
What is Software?

Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information and (3) documentation that describes the operation and use of the programs.

What is Software?

Software is a set of items or objects that form a "configuration" that includes

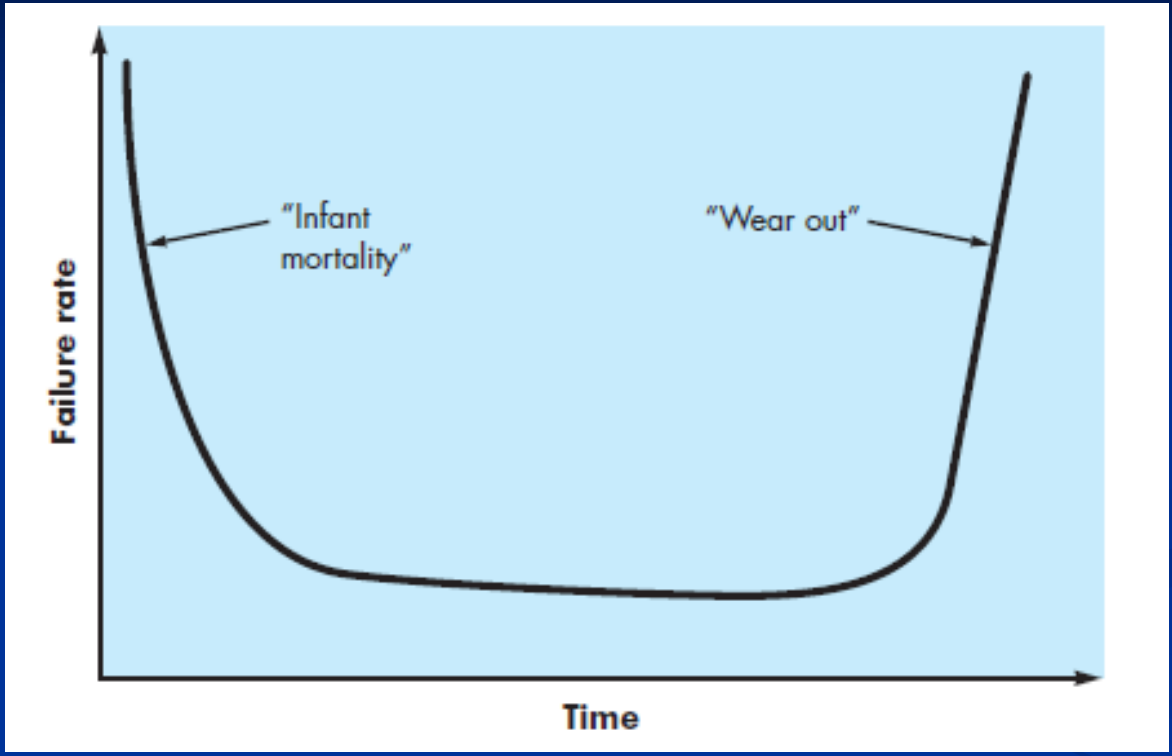
- programs
- documents
- data ...



What is Software?

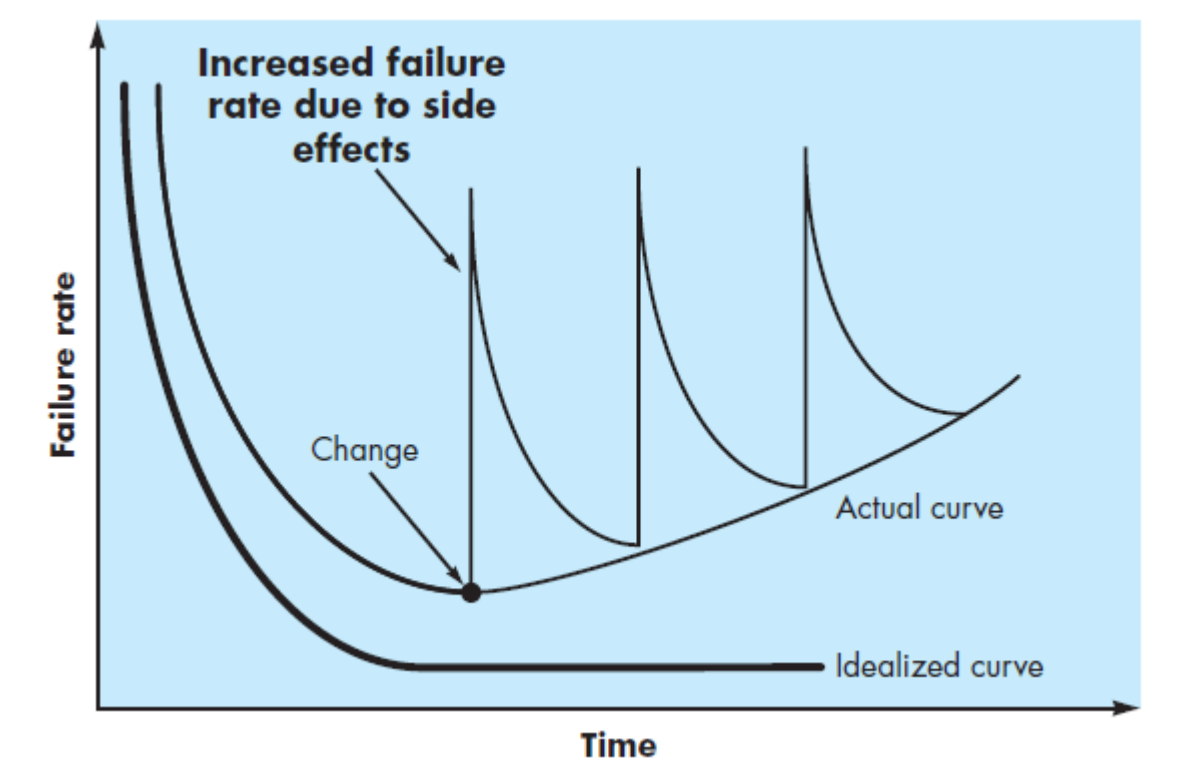
- software is engineered
- software doesn't wear out
- software is complex

- To accomplish that, it's important to examine the characteristics of software that make it different from other things that human beings build.
- Software is a logical rather than a physical system element.
- Therefore, software has one fundamental characteristic that makes it considerably different from hardware: *Software doesn't "wear out."*



- Fig. depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (hopefully, quite low) for some period of time.
- As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to *wear out.*

- Software is not susceptible to the environmental maladies that cause hardware to wear out.
- In theory, therefore, the failure rate curve for software should take the form of the "idealized curve"



- Undiscovered defects will cause high failure rates early in the life of a program.
- However, these are corrected and the curve flattens as shown.
- The idealized curve is a gross over simplification of actual failure models for software.
- However, the implication is clear—software doesn't wear out. But it does *deteriorate!*

- This contradiction can be explained by considering the actual curve in Figure . During its life, software will undergo change. As changes are made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the “actual curve”.
- Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

- Another aspect of wear illustrates the difference between hardware and software.
- When a hardware component wears out, it is replaced by a spare part.
- There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code.
- Therefore, the software maintenance tasks that accommodate requests for change involve considerably more complexity than hardware maintenance.

Software Applications

- system software
- application software
- engineering/scientific software
- embedded software
- product-line software
- WebApps (Web applications)
- AI software

- Millions of software engineers worldwide are hard at work on software projects in one or more of these categories.
- In some cases, new systems are being built, but in many others, existing applications are being corrected, adapted, and enhanced.
- It is not uncommon for a young software engineer to work on a program that is older than she is! Past generations of software people have left a legacy in each of the categories we have discussed.
- Hopefully, the legacy to be left behind by this generation will ease the burden on future software engineers.

Legacy Software:

- Hundreds of thousands of computer programs fall into one of the seven broad application domains discussed in the preceding subsection.
- Some of these are state-of-the-art software—just released to individuals, industry, and government.
- But other programs are older, in some cases *much* older.
- These older programs—often referred to as *legacy software* —have been the focus of continuous attention and concern since the 1960s.

- Dayani-Fard and his colleagues [Day99] describe legacy software in the following way:
- Legacy software systems . . . were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms. The proliferation of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve.
- Liu and his colleagues [Liu98] extend this description by noting that "many legacy systems remain supportive to core business functions and are 'indispensable' to the business."
- Hence, legacy software is characterized by longevity and business criticality.

- Unfortunately, there is sometimes one additional characteristic that is present in legacy software— ***poor quality*** .
- Legacy systems sometimes have inextensible designs, convoluted code, poor or nonexistent documentation, test cases and results that were never archived, a poorly managed change history—the list can be quite long.
- And yet, these systems support "core business functions and are indispensable to the business." What to do?

- The only reasonable answer may be: *Do nothing,* at least until the legacy system must undergo some signifi cant change.
- If the legacy software meets the needs of its users and runs reliably, it isn't broken and does not need to be fixed.

- However, as time passes, legacy systems often evolve for one or more of the following reasons:

  • The software must be adapted to meet the needs of new computing environments or technology.

  • The software must be enhanced to implement new business requirements.

  • The software must be extended to make it interoperable with other more modern systems or databases.

  • The software must be re-architected to make it viable within a evolving computing environment.

- Hence legacy system should be reengineered

–Devise methodologies that are founded on the notion of evolution

Software—New Categories

- Ubiquitous computing—wireless networks
- Netsourcing—the Web as a computing engine
- Open source—"free" source code open to the computing community (a blessing, but also a potential curse!)
- Also …
  - Data mining
  - Grid computing
  - Cognitive machines
  - Software for nanotechnologies

- 4 broad categories of software are evolving to dominate the industry:
  - –Web apps
  - –Mobile applications
  - –Cloud computing
  - –Product Line software

Key Concepts

- Realities behind the software development

–It follows that a concerted effort should be made to understand the problem before a software solution is developed.

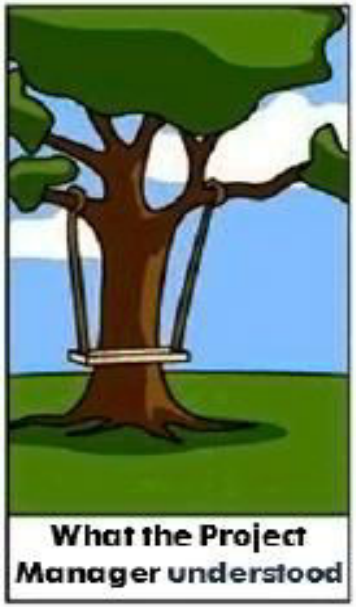
–It follows that design becomes a pivotal activity.

–It follows that software should exhibit high quality.

–It follows that software should be maintainable

•Software in all of its forms and across all of its application domains should be engineered.

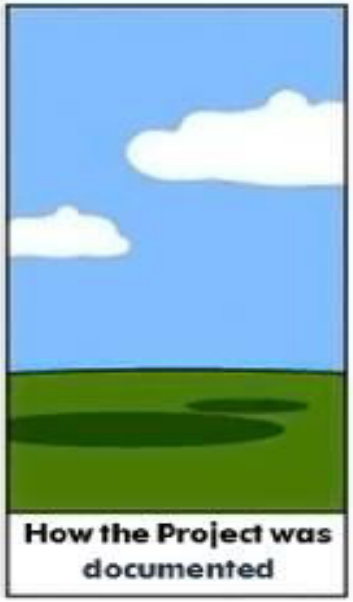

How the Customer
explained it

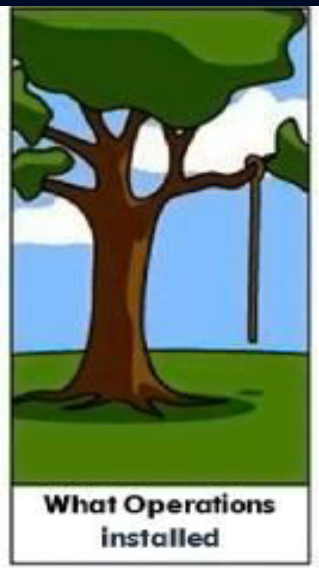

What the Project
Manager understood

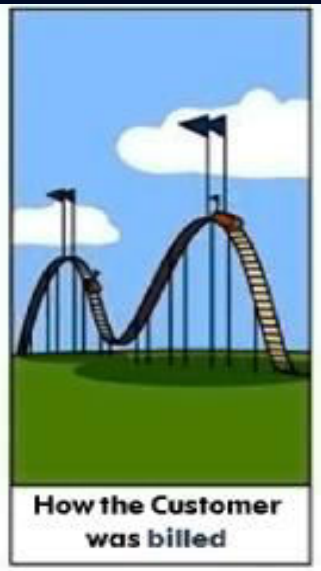

How the Analyst
designed it

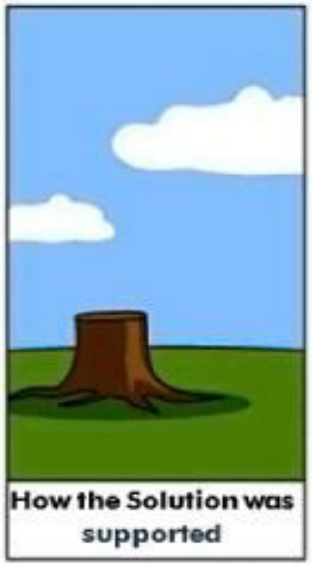

What the
Programmer wrote


What the Business
Consultant presented


How the Project was
documented


What Operations installed


How the Customer was billed


How the Solution was supported


What the Customer really needed

Defining Software Engineering as proposed by IEEE

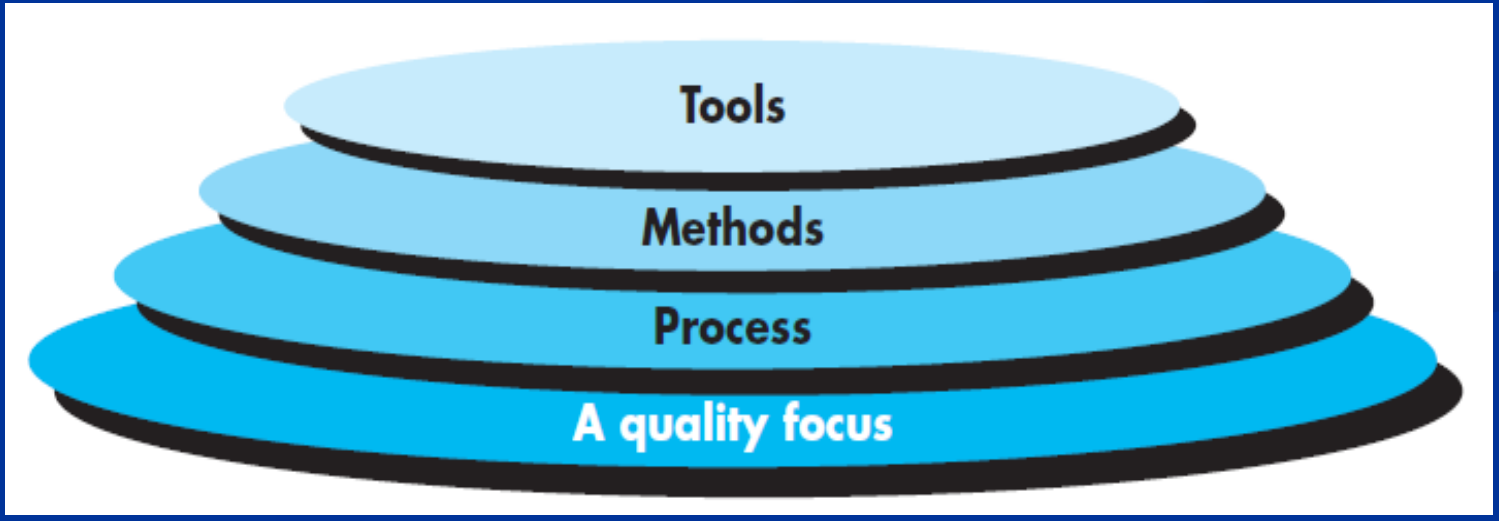
–***(1)The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.***

(2)The study of approaches as in(1).

–What about other approaches?

Adaptability and Agility

–Layered technology



- Any engineering approach must rest on an organizational commitment to quality.
- Process

–glue that holds the technology layers together & enables rational and timely development of computer software

–Forms the basis for management control of software projects and establishes the context in which

- technical methods are applied
- Work products are produced
- Milestones are establishes
- Quality is ensured &
- Change is properly managed

- **Methods** encompass a broad array of tasks that include
  - –Communication
  - –Requirements analysis
  - –Design modeling
  - –Program construction
  - –Testing &
  - –Support
- •Rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.
- •**Tools** provide automated or semi-automated support for the process and methods

The Software Process

- A *process is a collection of activities, actions, and tasks that are performed when* some work product is to be created
  - –Activity: strives to achieve broad objective
  - –Action: encompasses a set of tasks that produce a major work product
  - –Task: focuses on a small, but well-defined objective that produces a tangible outcome
- •It is an adaptable approach that enables the people doing the work to pick & choose the appropriate set of work actions and tasks

- The Process Framework

–Foundation for a software engineering process through a small number of framework activities

- Communication
- Planning
- Modeling
- Construction
- Deployment

–Encompasses a set of umbrella activities that are applicable across the entire software process.

Traditional Software Process Models

Generic Process Framework

- Communication
  - Involves communication among the customer and other stake holders; encompasses requirements gathering
- Planning
  - Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- Modelling (Analyse, Design)
  - Encompasses the creation of models to better under the requirements and the design

- Construction (Code, Test)
  - Combines code generation and testing to uncover errors
- Deployment
  - Involves delivery of software to the customer for evaluation and feedback

Modelling: Software Requirement Analysis

- Helps software engineers to better understand the problem they will work to solve
- Encompasses the set of tasks that lead to an understanding of what the business impact of the software will be, what the customer wants, and how end-users will interact with the software
- Uses a combination of text and diagrams to depict requirements for data, function, and behavior
  - Provides a relatively easy way to understand and review requirements for correctness, completeness and consistency

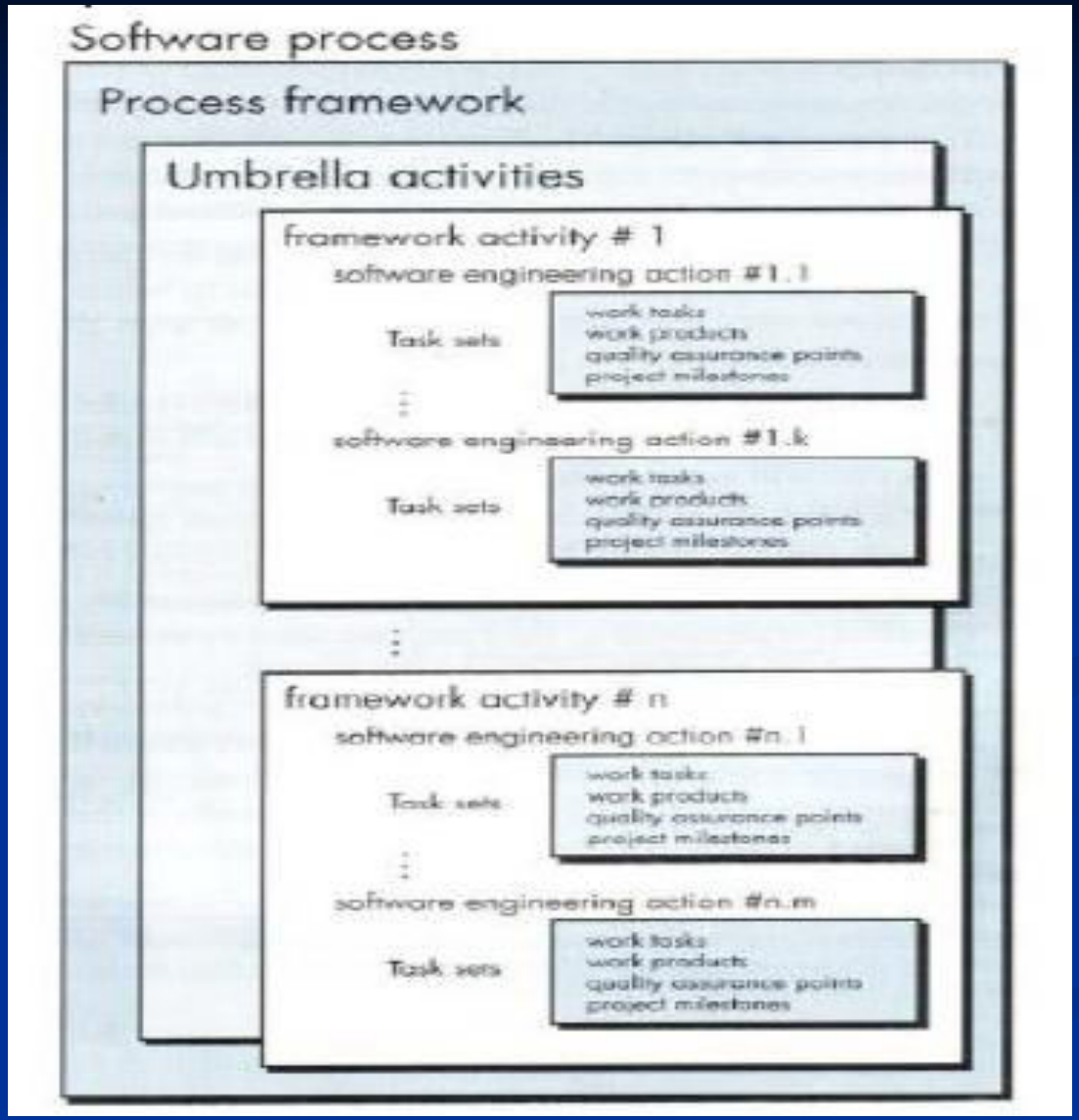
Modelling: Software Design

- Brings together customer requirements, business needs, and technical considerations to form the "blueprint" for a product
- Creates a model that provides detail about software data structures, software architecture, interfaces, and components that are necessary to implement the system
- Architectural design
  - Represents the structure of data and program components that are required to build the software
  - Considers the architectural style, the structure and properties of components that constitute the system, and interrelationships that occur among all architectural components

- User Interface Design
  - Creates an effective communication medium between a human and a computer
  - Identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype
- Component-level Design
  - Defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each software component

–Typical activities include,

- •Software Project Tracking and Control
- •Risk Management
- •Software Quality Assurance
- •Formal Technical Reviews
- • Measurement
- •Software Configuration Management
- •Reusability Management
- •Work Product preparation and production


Software process
Process framework
Umbrella activities
framework activity # 1
software engineering action #1.1
Task sets
work tasks
work products
quality assurance points
project milestones
software engineering action #1.k
Task sets
work tasks
work products
quality assurance points
project milestones
framework activity # n
software engineering action #n.1
Task sets
work tasks
work products
quality assurance points
project milestones
software engineering action #n.m
Task sets
work tasks
work products
quality assurance points
project milestones

Process Adaptation

It should be agile and adaptable

–Therefore, a process adopted for one project might be significantly different than a process adopted for another project

How do process models differ from one another?

- Interdependencies among activities and tasks
- Degree to which work tasks are defined within each framework activity
- Degree to which work products are identified and required
- Manner in which Quality assurance, Project Tracking and Control activities are applied
- Degree of detail and severity of the process applied
- Degree to which customers are involved
- Level of autonomy to the team
- Degree to which team organization and roles are prescribed

Prescriptive Process Models exists for the past 30 years
–objectives not achieved
–Due to rigidity and without adaptation
–Increase the level of bureaucracy associated and unintentionally create difficulty for developers and customers.

Agile Process Models in recent years
–Project agility
–More informal but no less effective approach
–Emphasize maneuverability and adaptability

Process Models

- We examine a number of "prescriptive" software process models
- Why prescriptive?
- –Prescribe a set of process elements for each project
    - •Framework activities
    - •Software engineering actions
    - •Tasks
    - •Work products
    - •Quality assurance &
    - •Change control mechanisms
- –Each process model also prescribes a workflow that invoke each framework activity in a different manner

•WATERFALL MODEL

- –Classic life cycle; systematic; sequential approach to software development that begins with customer specifications of requirements & progresses through planning, modeling, construction and deployment, concluding in on-going support of the completed s/w.
- Oldest paradigms for s/w engg and sometimes fail; problems encountered are:

•Real projects rarely follow the sequential flow that the model proposes

•Often difficult for the customer to state all the requirements explicitly

•Customer must have patience

•Some project teams must wait for other team members to complete dependent tasks –"blocking states"

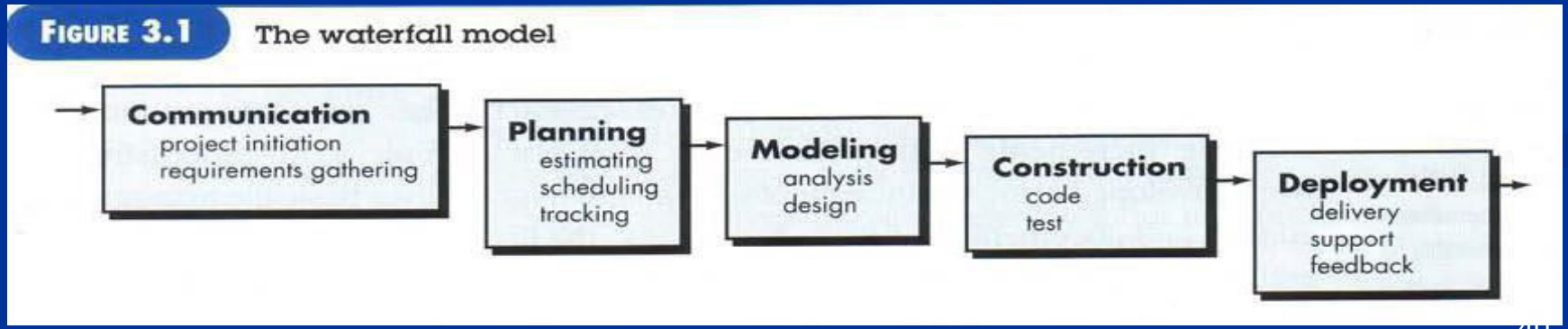


FIGURE 3.1 The waterfall model

Incremental Process Models

- –A compelling need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later software releases
- –A process model that is designed to produce the software in increments is chosen
- –Incremental Model

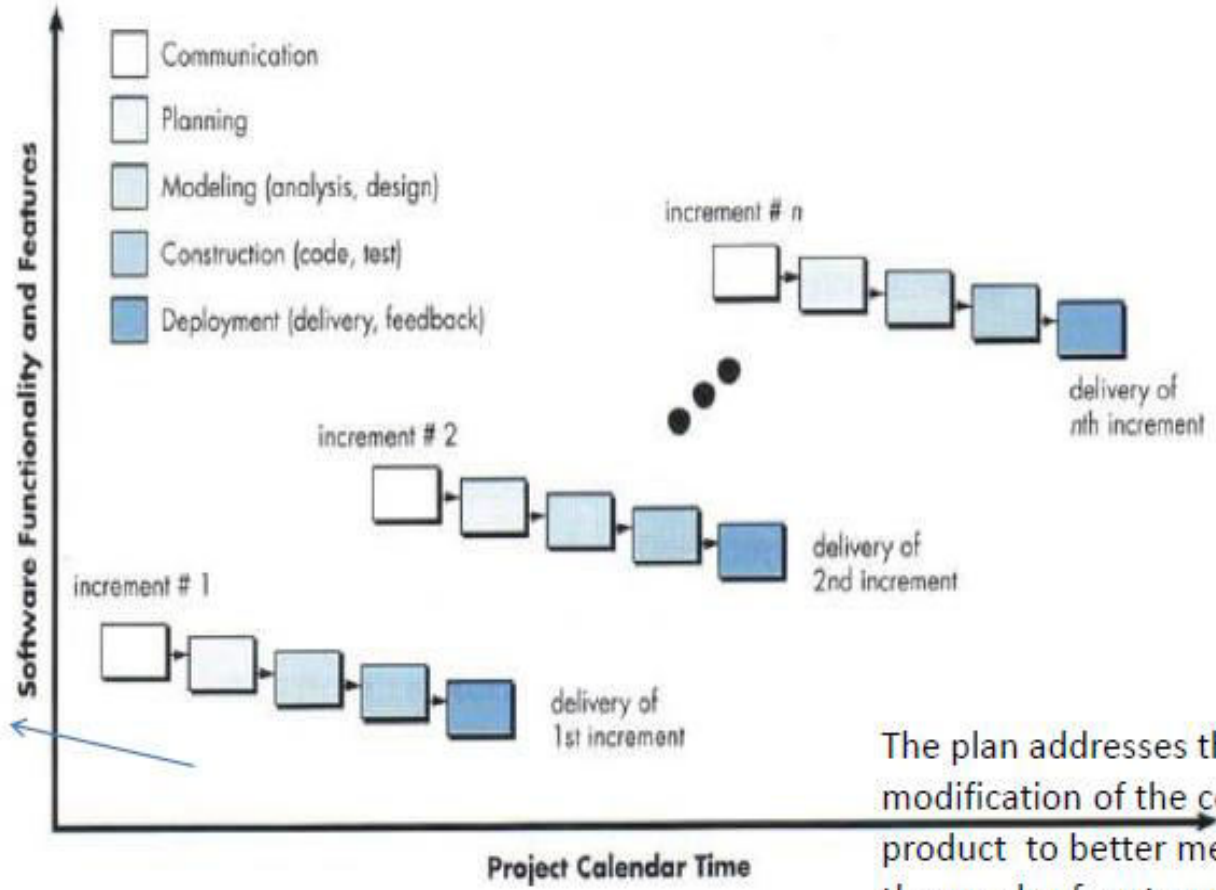
•Combines elements of the waterfall model applied in an iterative fashion

•Applies **linear sequences** in a staggered fashion as calendar time progresses –each linear sequences produces deliverable “increments” of the software

•Process flow for any increment may incorporate the prototype model

FIGURE 3.2

The incremental model



Often the "core product"

The plan addresses the modification of the core product to better meet the needs of customer and the delivery of additional features and functionality

Customer uses the product; as a result of use and evaluation, a plan is developed for the next increment

Rapid Application Development (RAD) model

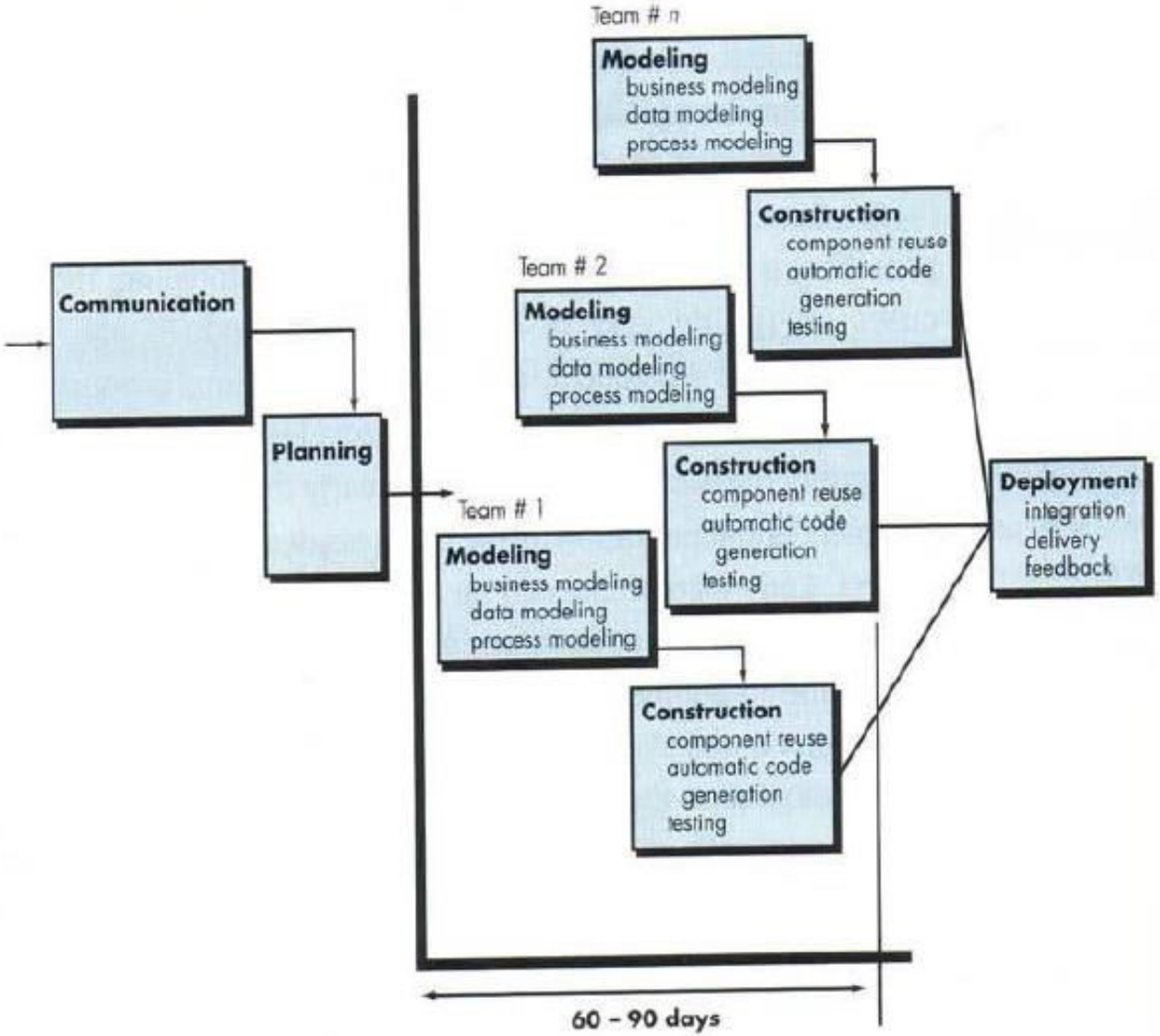
- –An incremental software process model that emphasizes a short development cycle
- –“high-speed” adaptation of the waterfall model
- –If requirements are well understood and project scope is constrained , the RAD process enables to create a “fully functional system” within a very short time period

Drawbacks

- Requires sufficient human resources to create the right number of RAD teams
- If developers and customers are not committed to the rapid-fire activities necessary to complete the system in a limited time, RAD projects fail
- Improper modularization will be problematic
- For high-performance, tuning the interfaces to system components will not work for RAD
- May not be appropriate for high technical risks

FIGURE 3.3

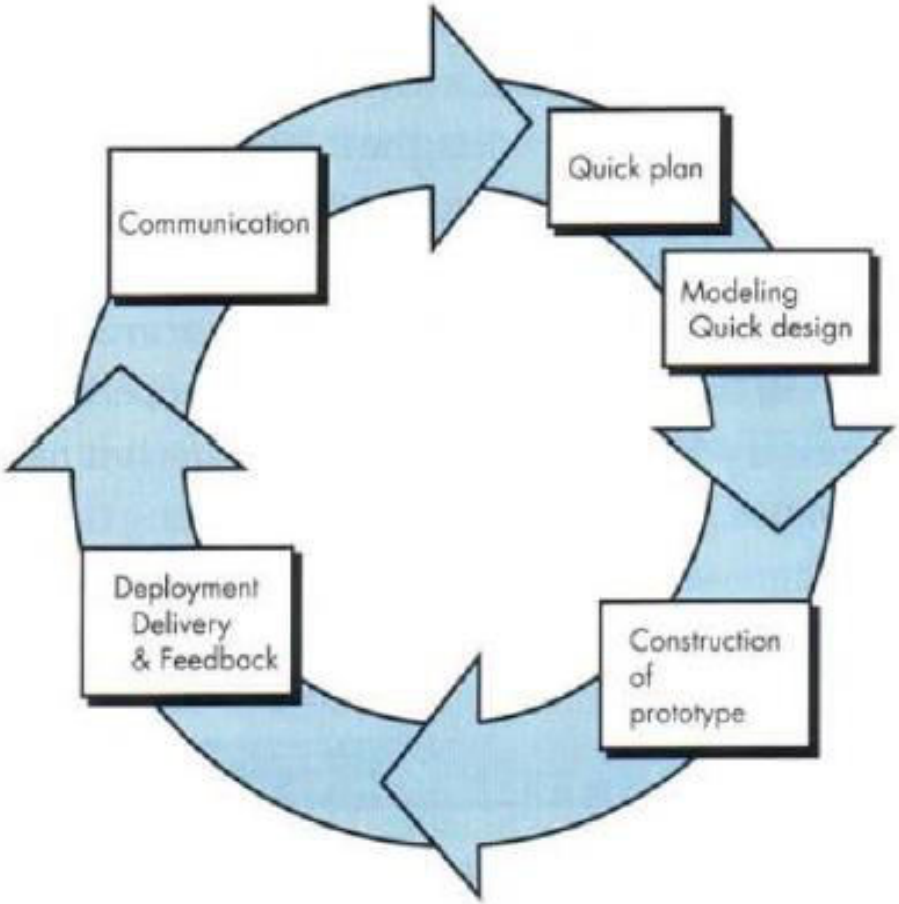
The RAD model



- **Evolutionary Process Models**
  - –Produces an increasingly more complete version of the software with each iteration
  - –Iterative
  - –Prototyping model
  - •Customer may not identify detailed input, processing or output requirements
  - •Developer may be unsure of the efficiency of an algorithm
  - •Adaptability of an OS
  - •For the above situations, Prototyping Paradigm is the best approach
  - •Quick Design

FIGURE 3.4

The proto-typing model



- The prototyping paradigm begins with communication. You meet with other stakeholders to define the overall objectives for the software.
- Identify whatever requirements are known, and outline areas where further definition is mandatory.
- A prototyping iteration is planned quickly, and modeling

(in the form of a “quick design”) occurs.

- A quick design focuses on a representation of those aspects of the software that will be visible to end users.

(e.g., human interface layout or output display formats).

- The quick design leads to the construction of a prototype.
- The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.
- Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done

- Ideally, the prototype serves as a mechanism for identifying software requirements.
- If a working prototype is to be built, you can make use of existing program fragments or apply tools that enable working programs to be generated quickly.
- But what do you do with the prototype when it has served the purpose described earlier? Brooks [Bro95] provides one answer:
- In most projects, the first system built is barely usable. It may be too slow, too big, awkward in use or all three.
- There is no alternative but to start again, smarting but smarter, and build a redesigned version in which these problems are solved.

- The prototype can serve as “the first system.” The one that Brooks recommends you throw away.
- But this may be an idealized view.
- Although some prototypes are built as “throwaways,” others are evolutionary in the sense that the prototype slowly evolves into the actual system.
- Both stakeholders and software engineers like the prototyping paradigm.
- Users get a feel for the actual system, and developers get to build something immediately.
- Yet, prototyping can be problematic for the following reasons:

- Stakeholders see what appears to be a working version of the software, unaware that the prototype is held together haphazardly, unaware that in the rush to get it working you haven't considered overall software quality or long-term maintainability.
- When informed that the product must be rebuilt so that high levels of quality can be maintained, stakeholders cry foul and demand that “a few fixes” be applied to make the prototype a working product.
- Too often, software development management relents.

- As a software engineer, you often make implementation compromises in order to get a prototype working quickly.
- An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability.
- After a time, you may become comfortable with these choices and forget all the reasons why they were inappropriate.
- The less-than-ideal choice has now become an integral part of the system.

- Although problems can occur, prototyping can be an effective paradigm for software engineering.
- The key is to define the rules of the game at the beginning; that is, all stakeholders should agree that the prototype is built to serve as a mechanism for defining requirements.
- It is then discarded (at least in part), and the actual software is engineered with an eye toward quality.

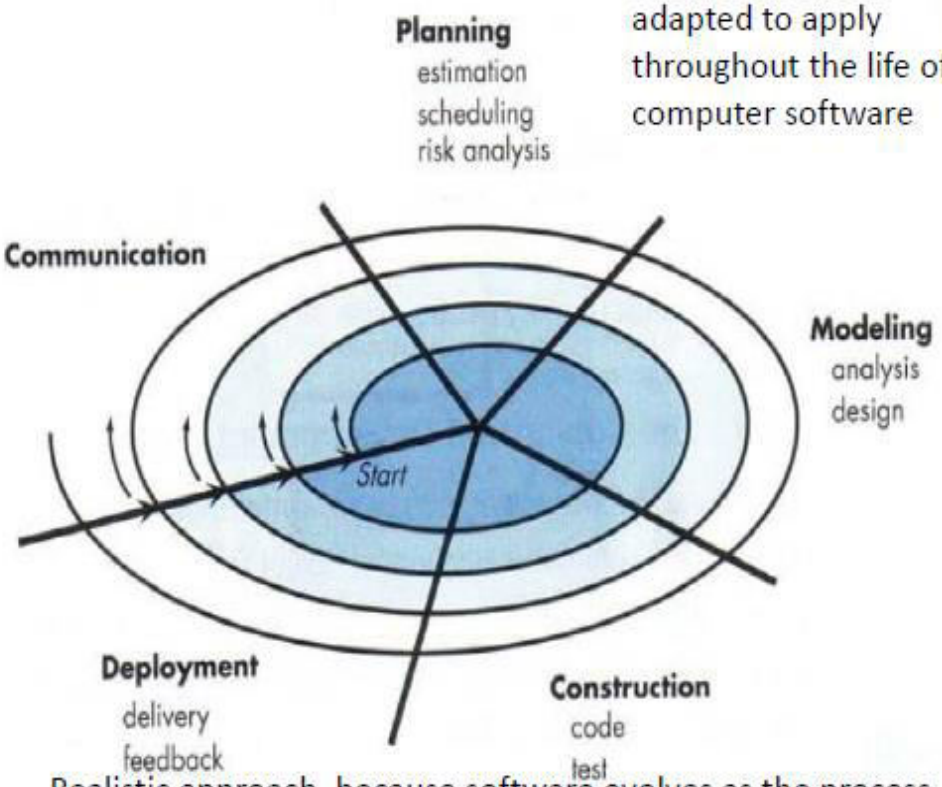
Spiral Model

- The classical models do not deal with the uncertainty with the software projects.
- A lot risk assessment and analysis form a part of the software development.
- This was first realized by Barry Boehm, who introduced the factor of “project risk” into the life cycle model which resulted in the spiral model in 1986
- Couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model

Concept Development Project in the first circuit;
New Product Development Project
Product Enhancement Project

Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of computer software

FIGURE 3.5
A typical spiral model



Risk is considered as each evolution is made

Anchor point mile stones - a combination of work products & conditions that are attained along the path of spiral are noted

Realistic approach, because software evolves as the process progresses, the developer and the customer better understand and reacts to risks at each evolutionary level

- Therefore, the first circuit around the spiral might represent a “concept development project” that starts at the core of the spiral and continues for multiple iterations until concept development is complete.
- If the concept is to be developed into an actual product, the process proceeds outward on the spiral and a “new product development project” commences.
- The new product will evolve through a number of iterations around the spiral.
- Later, a circuit around the spiral might be used to represent a “product enhancement project.”

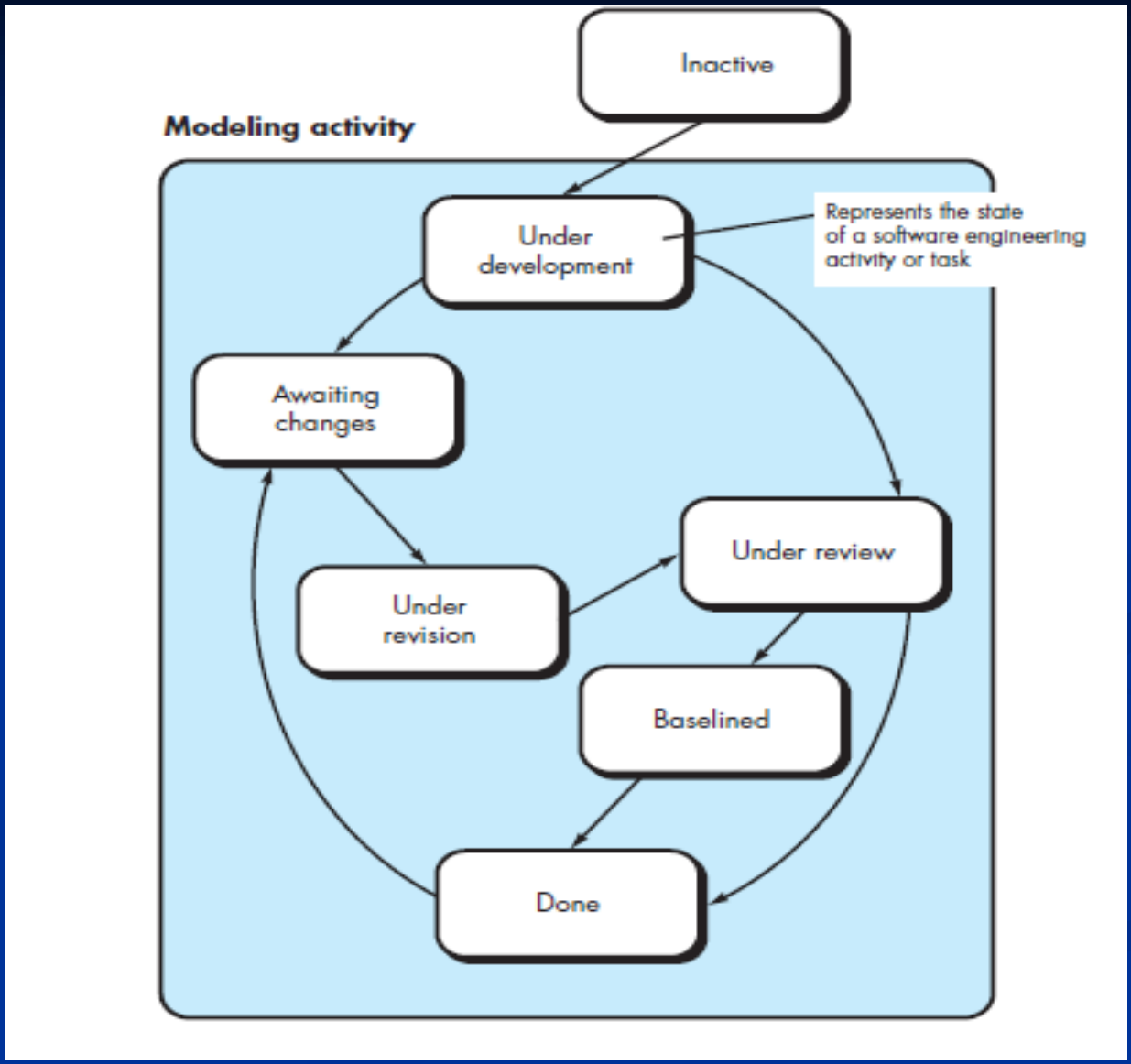
- In essence, the spiral, when characterized in this way, remains operative until the software is retired.
- There are times when the process is dormant, but whenever a change is initiated, the process starts at the appropriate entry point (e.g., product enhancement).

- The spiral model is a realistic approach to the development of large-scale systems and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level.
- The spiral model uses prototyping as a risk reduction mechanism but, more important, enables you to apply the prototyping approach at any stage in the evolution of the product.

- It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world.
- The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic.

CONCURRENT MODEL

- The *concurrent development model,* sometimes called *concurrent engineering,* allows a software team to represent iterative and concurrent elements of any of the process models described in this chapter.
- For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following software engineering actions: prototyping, analysis, and design.


Inactive
Modeling activity
Under development
Represents the state of a software engineering activity or task
Awaiting changes
Under review
Under revision
Baselined
Done

CONCURRENT MODEL

- An activity— **modeling** —may be in any one of the states 7 noted at any given time.
- Similarly, other activities, actions, or tasks (e.g., **communication** or **construction**) can be represented in an analogous manner.
- All software engineering activities exist concurrently but reside in different states.

CONCURRENT MODEL

- For example, early in a project the communication activity has completed its first iteration and exists in the **awaiting changes** state.
- The modeling activity (which existed in the **none** state while initial communication was completed) now makes a transition into the **under development** state.
- If, however, the customer indicates that changes in requirements must be made,the modeling activity moves from the **under development** state into the **awaiting changes** state.

CONCURRENT MODEL

- Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks.
- For example, during early stages of design (a major software engineering action that occurs during the modeling activity), an inconsistency in the requirements model is uncovered.
- This generates the event *analysis model correction* , which will trigger the requirements analysis action from the **done** state into the **awaiting changes** state.

CONCURRENT MODEL:

- Concurrent modeling is applicable to all types of software development and provides an accurate picture of the current state of a project.
- Rather than confining software engineering activities, actions, and tasks to a sequence of events, it defines a process network.
- Each activity, action, or task on the network exists simultaneously with other activities, actions, or tasks.
- Events generated at one point in the process network trigger transitions among the states associated with each activity.

- Last Lecture:
- What is Software?
- Software Process
- Process Models

Today’s Lecture

- Traditional Approach Vs Agile Approach
- Agile Manifesto
- Agile Methodology

What is Traditional Process models

- Traditional project management is an established methodology where projects are run in a sequential cycle.
- It follows a fixed sequence: Communication, planning, modelling, construction, and deployment.
- The traditional project management approach puts special emphasis on linear processes, documentation, upfront planning, and prioritization.
- As per the traditional method, time and budget are variable and requirements are fixed due to which it often faces budget and timeline issues.

What is Traditional Process models

- For every step, there are tools and techniques defined by the standard methodology PMBOK® which are followed by project managers.
- Interestingly, it also includes other methodologies such as PRINCE 2 which is followed by various organizations under UK government and private companies like Vodafone, Siemens and others.
- It is also called the Waterfall model

Benefits of traditional methodology

- Clearly defined objectives
- Controllable processes
- Clear documentation
- More accountability

Traditional Software Process Models

Generic Process Framework

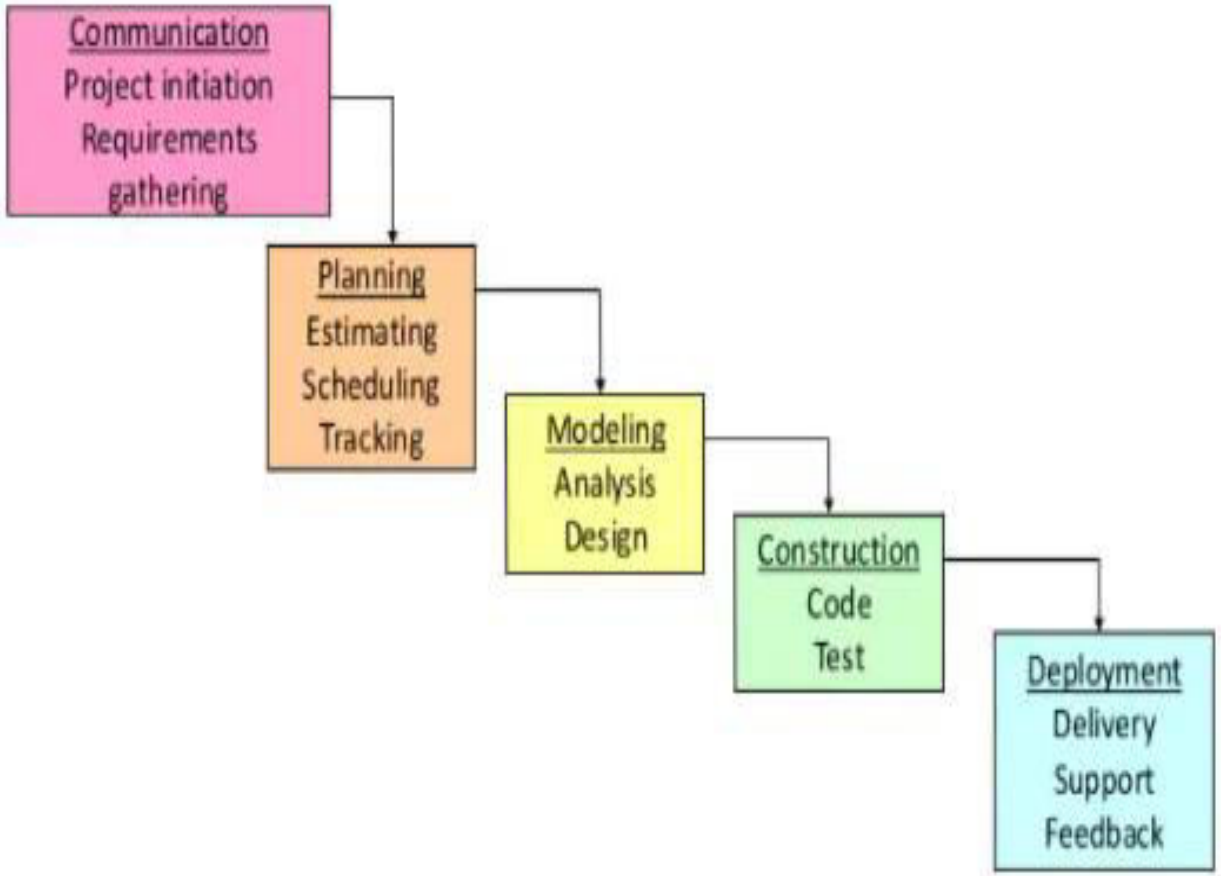
- Communication
  - Involves communication among the customer and other stake holders; encompasses requirements gathering
- Planning
  - Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- Modelling (Analyse, Design)
  - Encompasses the creation of models to better under the requirements and the design

- Construction (Code, Test)
  - Combines code generation and testing to uncover errors
- Deployment
  - Involves delivery of software to the customer for evaluation and feedback

Process Model

- Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software
- The activities may be linear, incremental, or evolutionary

Waterfall Model

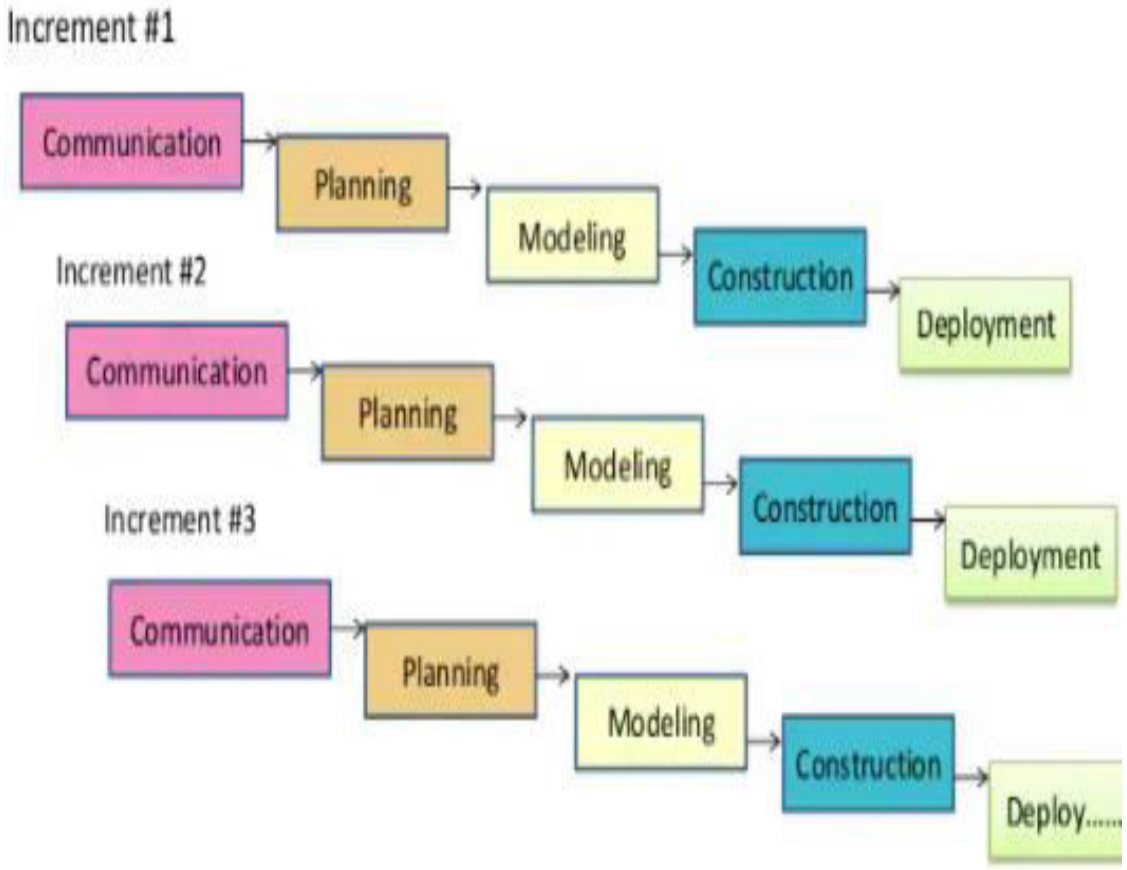


- Oldest software lifecycle model
- Used when requirements are well understood and risk is low
- Work flow is in a linear (i.e., sequential) fashion
- Used often with well-defined adaptations or enhancements to current software

Waterfall Model- Problems

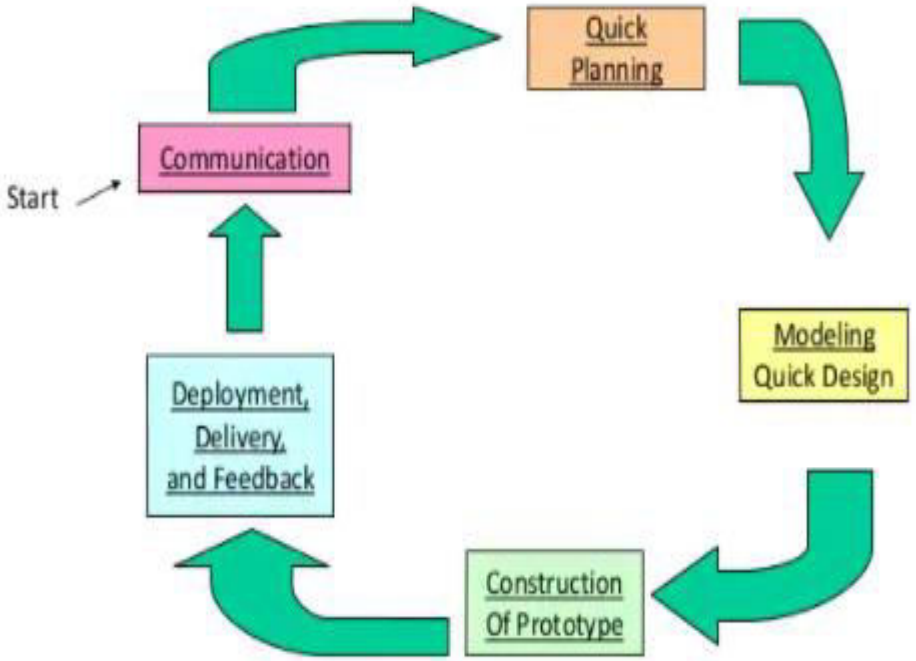
- Doesn't support iteration, so changes can cause confusion
- Difficult for customers to state all requirements explicitly and up front
- Requires customer patience because a working version of the program doesn't occur until the final phase

Incremental Model



- Used when requirements are well understood
- Multiple independent deliveries are identified
- Work flow is in a linear (i.e., sequential) fashion within an increment and is staggered between increments
- Iterative in nature; focuses on an operational product with each increment
- Provides a needed set of functionality sooner while delivering optional components later
- Useful also when staffing is too short for a full-scale development

Prototyping Model

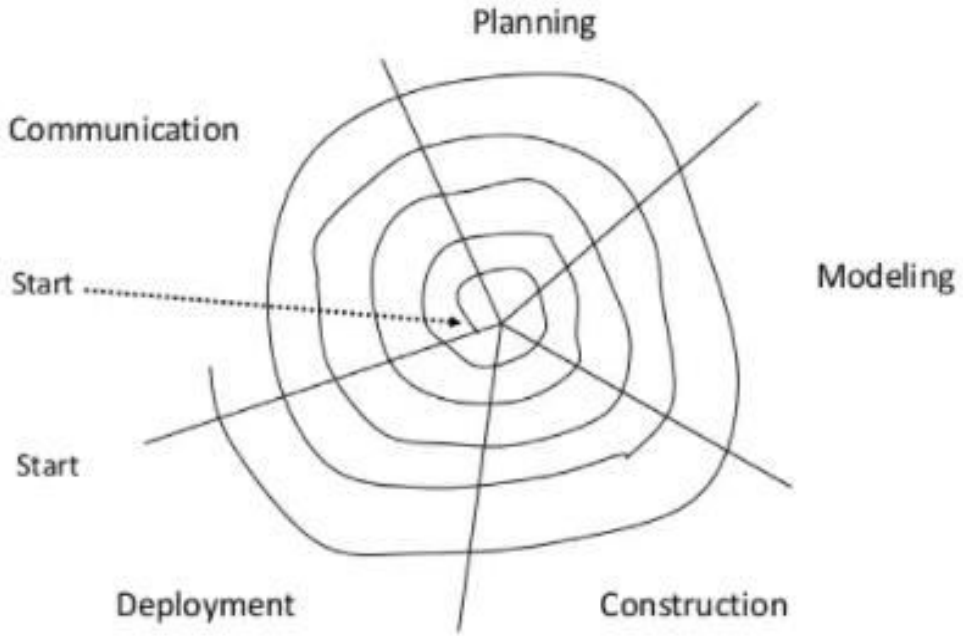


- Follows an evolutionary and iterative approach
- Used when requirements are not well understood
- Serves as a mechanism for identifying software requirements
- Focuses on those aspects of the software that are visible to the customer/user
- Feedback is used to refine the prototype

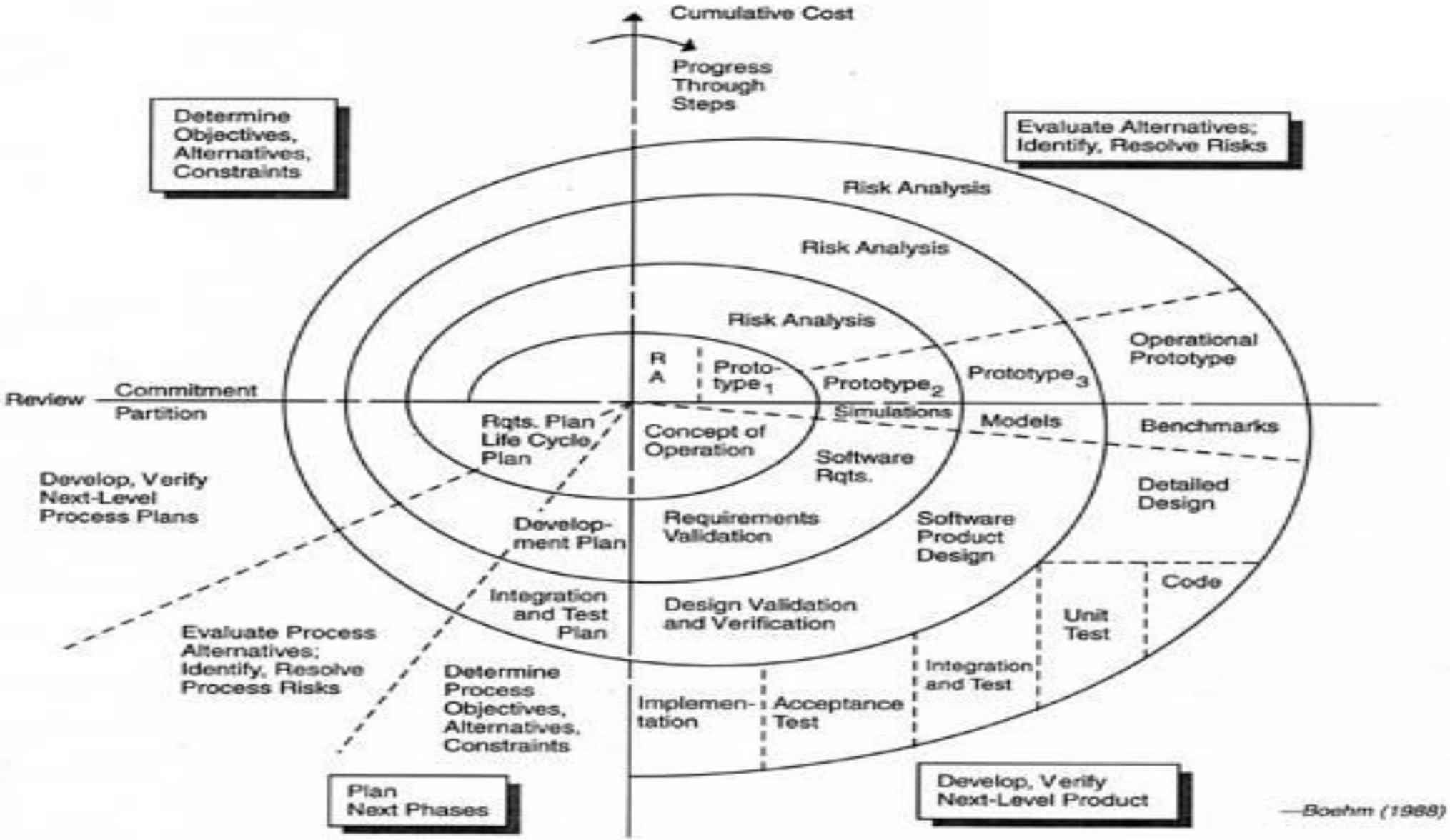
Prototyping Model- Problems

- The customer sees a "working version" of the software, wants to stop all development and then buy the prototype after a "few fixes" are made
- Developers often make implementation compromises to get the software running quickly (e.g., language choice, user interface, operating system choice, inefficient algorithms)
- Lesson learned
  - Define the rules up front on the final disposition of the prototype before it is built
  - In most circumstances, plan to discard the prototype and engineer the actual production software with a goal toward quality

Spiral Model



- Follows an evolutionary approach
- Used when requirements are not well understood and risks are high
- Inner spirals focus on identifying software requirements and project risks; may also incorporate prototyping
- Outer spirals take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software
- Operates as a risk-driven model…a go/no-go decision occurs after each complete spiral in order to react to risk determinations
- Requires considerable expertise in risk assessment
- Serves as a realistic model for large-scale software development


Cumulative Cost
Progress Through Steps
Determine Objectives, Alternatives, Constraints
Evaluate Alternatives; Identify, Resolve Risks
Risk Analysis
Risk Analysis
Risk Analysis
R A
Proto-type 1
Prototype 2
Prototype 3
Operational Prototype
Review
Commitment Partition
Rqts. Plan Life Cycle Plan
Concept of Operation
Simulations
Models
Benchmarks
Software Rqts.
Detailed Design
Develop, Verify Next-Level Process Plans
Develop-ment Plan
Requirements Validation
Software Product Design
Code
Integration and Test Plan
Design Validation and Verification
Unit Test
Evaluate Process Alternatives; Identify, Resolve Process Risks
Integration and Test
Determine Process Objectives, Alternatives, Constraints
Implemen-tation
Acceptance Test
Plan Next Phases
Develop, Verify Next-Level Product
—Boehm (1988)

General Weakness of Evolutionary Process model

1) Prototyping poses a problem to project planning because of the uncertain number of iterations required to construct the product
2) Evolutionary software processes do not establish the maximum speed of the evolution
   - If too fast, the process will fall into chaos
   - If too slow, productivity could be affected
3) Software processes should focus first on flexibility and extensibility, and second on high quality
   - We should prioritize the speed of the development over zero defects
   - Extending the development in order to reach higher quality could result in late delivery

What is Agile Process model

- While Agile is a general approach used for software development, it relies heavily on teamwork, collaboration, time boxing tasks, and the flexibility to respond to change as quickly as possible.
- Agile follows an iterative process where projects are divided into sprints of the shorter span.
- Unlike the traditional approach, less time is spent on upfront planning and prioritization as agile is more flexible in terms of changes and developments in the specification.

Benefits of agile process model

- Flexible prioritization
- Early and predictable delivery
- Predictable costs and schedules
- Improves quality
- More transparency

Difference between traditional and agile project methodology

Characteristics	Agile approach	Traditional approach
Organizational structure	Iterative	Linear
Scale of projects	Small and medium scale	Large-scale
User requirements	Interactive input	Clearly defined before implementation
Involvement of clients	High	Low
Development model	Evolutionary delivery	Life cycle
Customer involvement	Customers are involved from the time work is being performed	Customers get involved early in the project but not once the execution has started

Difference between traditional and agile project methodology

Characteristics	Agile approach	Traditional approach
Escalation management	When problems occur, the entire team works together to resolve it	Escalation to managers when problem arise
Model preference	Agile model favors adaption	Traditional model favors anticipation
Product or process	Less focus on formal and directive processes	More serious about processes than the product
Test documentation	Comprehensive test planning	Tests are planned one sprint at a time
Effort estimation	Scrum master facilitates and the team does the estimation	Project manager provides estimates and gets approval from PO for the entire project
Reviews and approvals	Reviews are done after each iteration	Excessive reviews and approvals by leaders

Why is Agile preferred not traditional PM approach

Many developers and project managers prefer to use the agile methodology for a variety of reasons. Some of them are discussed below:

More flexibility

- When it comes to making changes in the product or a process, agile methodology is much more flexible than the waterfall methodology.
- While working, if team members feel that there is a need to experiment and try something different than as planned, the agile methodology easily allows them to do so.
- The best thing about this methodology is that it focuses more on the product than following a rigid structure.
- Unlike the traditional approach, agile methodology isn't linear or follows a top-down approach. So, any last-minute changes can be accommodated in the process without affecting the end-result and disrupting the project schedule.

Transparency

- In agile methodology, everything is out there and transparent. The clients and decision makers are actively involved from the initiation, planning, review, and in the testing part of a product.
- Whereas in the traditional approach, the project manager is holding reins of the project, thus others don't get to make the major decisions.
- The agile methodology facilitates team members to view the progress right from the start to the end.
- This level of transparency plays a significant role to constitute a healthy work environment.

Ownership and accountability

- One of the striking differences in both project management approaches is the level of ownership and accountability that each provides to team members.
- In traditional project management, a project manager is the person of the ship which means that the entire ownership belongs to him/her. Customers are also involved during the planning phase but their involvement ends there and then as soon as the execution starts.
- In the agile methodology, every team member shares ownership of the project. Each one of them plays an active role to complete the sprint within the estimated time.
- Unlike traditional project management, everyone involved in the project can easily see view the progress from the beginning to an end.

Scope for feedback

- In the traditional approach, every single process is clearly defined and planned from the beginning of the project. The project has to be completed within the estimated time and budget. So, any big change or feedback that might push the deadline is skipped.
- Whereas agile management allows constant feedback that is helpful in providing better output.
- Due to high acceptance for feedback in agile methodology, it has become the first choice for many project managers and software developers.
- They can respond to customer requests as customers get to validate each iteration that enables them to deliver a high-quality product or service within the delivery time.

Project complexity

- Owing to its linear approach, the traditional project management methodology is majorly used for small or less complex projects. As discussed earlier, this methodology isn't a fan of sudden changes and avoids them strictly as it would take the team back to square one.
- Agile could be your best bet in terms of managing big and complex projects. Whether your project has multiple interconnected phases or one stage is dependent on many others, choose agile as it is a better fit for complex projects

How to choose the correct approach

- In reality, there is no ‘one-size-fits-all’ methodology suitable for every project or organization. The choice to implement a methodology largely depends on factors such as the nature of the project, size, resources involved among others.
- Most of the times, smart project managers decide which methodology to adopt during the beginning or initiation of the project. He takes the final call in agreement with other project sponsors and people involved in the project planning process.
- There are some factors you can take into consideration while choosing the right methodology for your project.

- Take a look at the project requirements. Are the requirements clear? If project requirements are unclear or tend to change, choose the agile methodology. And, the traditional methodology fits best to a situation where the requirements are clearly defined and well understood from the first go.
- Consider the technology involved in the project. The traditional project management methodology is more appropriate if no new technology or tools are involved. Agile methodology, being more flexible than the former allows more space for more experimentation with the new technology.

- Is the project prone to unwanted risks and threats? Considering the rigid nature of the traditional methodology, it's not advisable to go this methodology. However, risks can be addressed sooner in the agile approach, it seems like a better option in terms of risk management.
- Another important factor is the availability of resources. The traditional approach works best with big and complex teams and projects. Whereas an agile team usually consists of a limited number of experienced team members.
- The criticality of an end product depends a lot on the nature of the chosen project management methodology. As the traditional method involves a documentation, it is very much suitable for critical products as compared to the agile project management methodology.

Agile Manifesto

- The Agile Manifesto is a document that identifies four key values and 12 principles that its authors believe software developers should use to guide their work.
- Furthermore, the developers express a desire to find a balance between the existing ways of development and new the alternatives. They admit to accepting modeling and documentation, but only when it has a clear, beneficial use.
- The developers also explain that while planning is important, it is also necessary to accept that plans change and to allow flexibility for these modifications.
- Overall, the Manifesto focuses on valuing individuals and interactions over processes and tools.

Four values of Agile

The four core values of Agile software development as stated by the Agile Manifesto are:

- individuals and interactions over processes and tools;
- working software over comprehensive documentation;
- customer collaboration over contract negotiation; and
- responding to change over following a plan.

Agile Manifesto - 12 Principles


Watch later
Shar

The 12 principles articulated in the Agile Manifesto are:

- Satisfying customers through early and continuous delivery of valuable work.
- Breaking big work down into smaller tasks that can be completed quickly.
- Recognizing that the best work emerges from self-organized teams.
- Providing motivated individuals with the environment and support they need and trusting them to get the job done.
- Creating processes that promote sustainable efforts.
- Maintaining a constant pace for completed work.
- Welcoming changing requirements, even late in a project.
- Assembling the project team and business owners on a daily basis throughout the project.
- Having the team reflect at regular intervals on how to become more effective, then tuning and adjusting behavior accordingly.
- Measuring progress by the amount of completed work.
- Continually seeking excellence.
- Harnessing change for a competitive advantage

The Agile Manifesto's purpose

- Supporters of Agile methodologies say the four values outlined in the Agile Manifesto promote a software development process that focuses on quality by creating products that meet consumers' needs and expectations.
- The 12 principles are intended to create and support a work environment that is focused on the customer, that aligns to business objectives and that can respond and pivot quickly as user needs and market forces change.

Agile methodology

Any Agile Methodology that is suitable for any enterprise, can be depending on various factors;

- ***Team types:*** In relevance to the means of Agile processes, an individual is willing to deploy.
- ***Organization size*** **and** *the condition on which an individual looks to scale agile from lower to top*
- ***Organizational culture:*** To determine whether an organization is ready, or would be interested, for a highly- configurated agile strategy, or looking for more compliant approaches.

Various Agile Methodologies

- There are various types of agile methodology available in the market to suit every project's wants. Although there are different agile methodologies, everything is based on the main principles in the agile manifesto.
- Therefore, every framework or behavior that adapts these principles is named Agile, and, the agile methodology benefits can be copiously apprehended only with the collaboration of all the involved parties.
- The below agile methodologies list comprises of famous types of agile methodology that one can opt from:

Agile Scrum Methodology

- Scrum is a lightweight framework of Agile Project Management, it can be adopted to conduct iterative and all types of incremental projects.

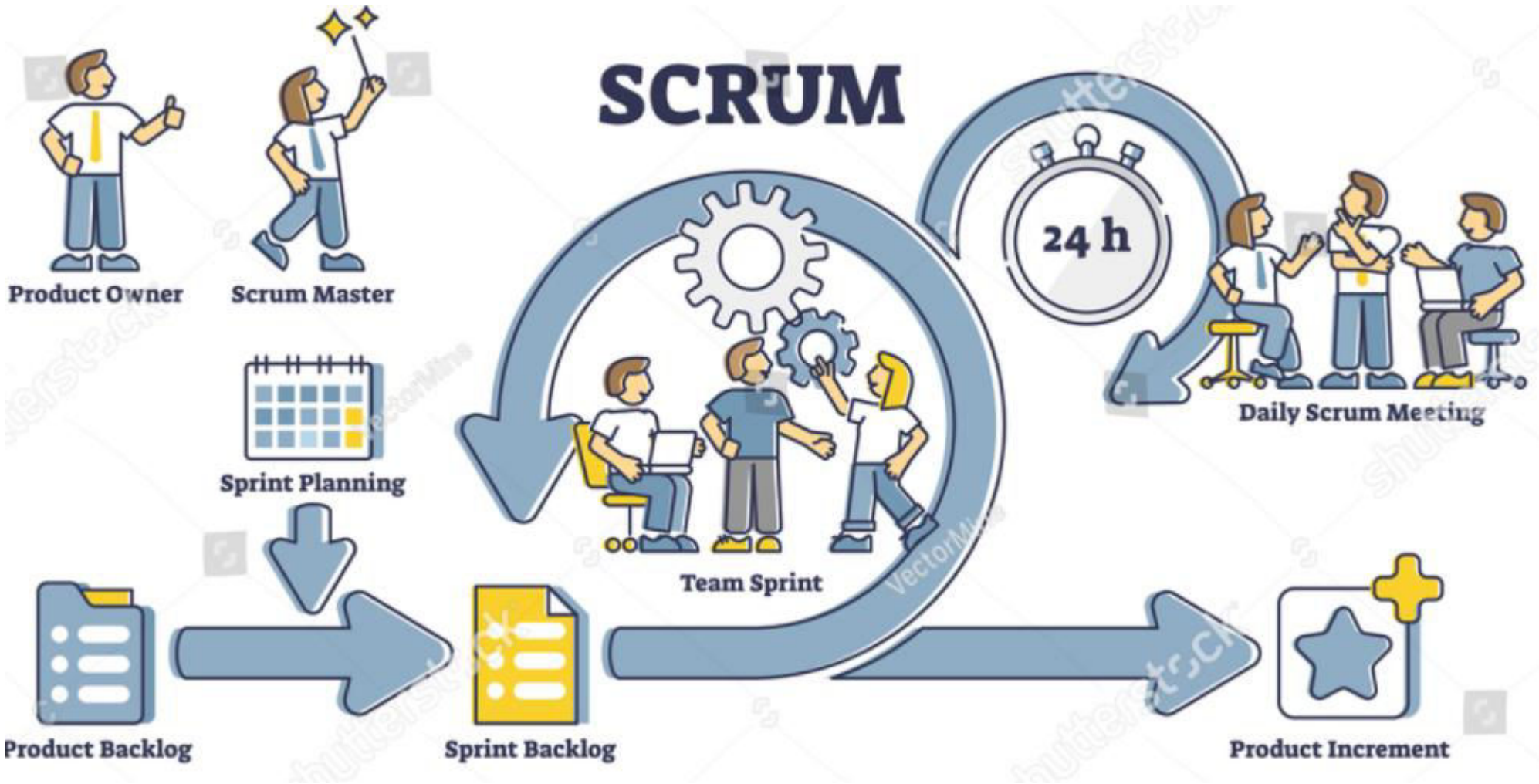
- Due to its specific characteristics like simplicity, sustained productivity, and strength for blending several underlying approaches adopted by other agile methods, Scrum has obtained popularity over the years.

- The hands-on system under Scrum includes easy steps and elements, that are the following;

Agile Scrum Methodology

- ***Product owner,*** who creates an estimated wish list that is identified as a product backlog.
- ***Scrum team,*** that takes one little part of the top wish list, termed as ***Sprint Backlog*** and work out in order to implement it.
- After that scrum team concludes their sprint backlog task in a ***Sprint***, i.e., a period of 2-4 weeks. In addition to that, the progress of their work can be accessed through a meeting that is called ***Daily Scrum***.
- The ***Scrum Master*** maintains the team focused toward their targets.
- At the end of a sprint, the task is able to represent or transmit, and team finishes that particular sprint with a review and feedback and initiates with a new one.

Agile Scrum Methodology



Agile Scrum Methodology

Advantages:	Disadvantages:
• There is a lot of motivation in teams, because programmers want to meet the deadline of every sprint; • The transparency allows for the project to be followed by all members of a team or even an organisation; • The focus on quality is a constant in the scrum method, resulting in fewer mistakes. • The dynamics of this method allow developers to reorganise priorities, ensuring that sprints that have not yet been completed get more attention	• The segmentation of the project and the search for agility of the development can sometimes lead the team to lose track of the project as a whole, focusing only on one part; • Each developer's role may not be well defined, resulting in some confusion amongst team members.

Lean

- It is the iterative, agile methodology that directs the team on addressing customer values by compelling value stream mapping, although, it is a deeply adaptable, emerging methodology with the absence of solid guidelines, laws, or methods.

The fundamental *principles of Lean* including;

- *Uninterrupted advancement,*
- *Respect for other people,*
- *Eradicate waste,*
- *Rapid delivery,*
- *Knowledge-making and*
- *Defer commitment.*

Lean

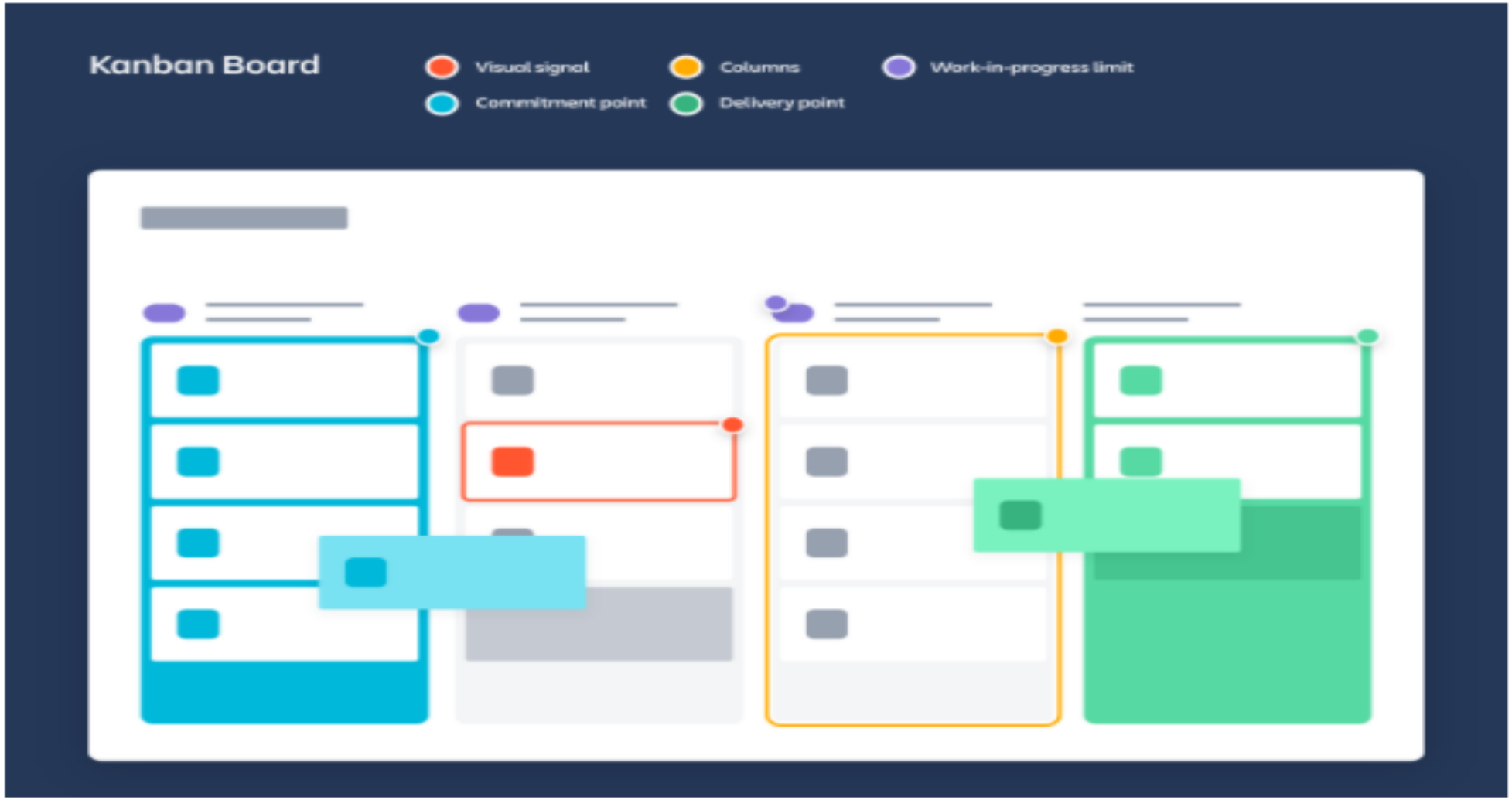
- Lean gives the authority of decision-making to every individual and small teams as it is considered as the most faster and effective method in comparison to the hierarchical flow of control.
- It focuses on the proficient implementation of team resources and assures to make everyone as productive as possible for maximum time

Kanban

- Kanban is an eminently visual workflow management approach, famous amidst Lean teams, that can be employed for visualizing and thoroughly maintaining the making of products, it focuses on continual delivery of the product, but is not making stress to the entire software development life cycle.
- Similar to Scrum, Kanban is the process developed for supporting collaborative teamwork more effectively. It works adequately on three principles;
- ***For visualizing what to perform today,*** i.e, workflow automation, that specifies all the elements, under the context of each other, could be very informative**.**
- ***For bounding the quantity of work in progress*** to maintain harmony in the flow-based approach, so that teams can‘t begin and commit extra work at once.
- ***For boosting flow,*** like, when some task is about to complete, the next priority would be item into play from the backlog.
- However, Kanban encourages steadily collaboration and maintains active, continuous learning and enhancement through describing the best plausible team workflow.

Elements of a Kanban board

Kanban boards can be broken down into five components: Visual signals, columns, work-in-progress limits, a commitment point, and a delivery point.



- **Visual Signals** — One of the first things you'll notice about a kanban board are the visual cards (stickies, tickets, or otherwise).
- Kanban teams write all of their projects and work items onto cards, usually one per card. For agile teams, each card could encapsulate one user story.
- Once on the board, these visual signals help teammates and stakeholders quickly understand what the team is working on.

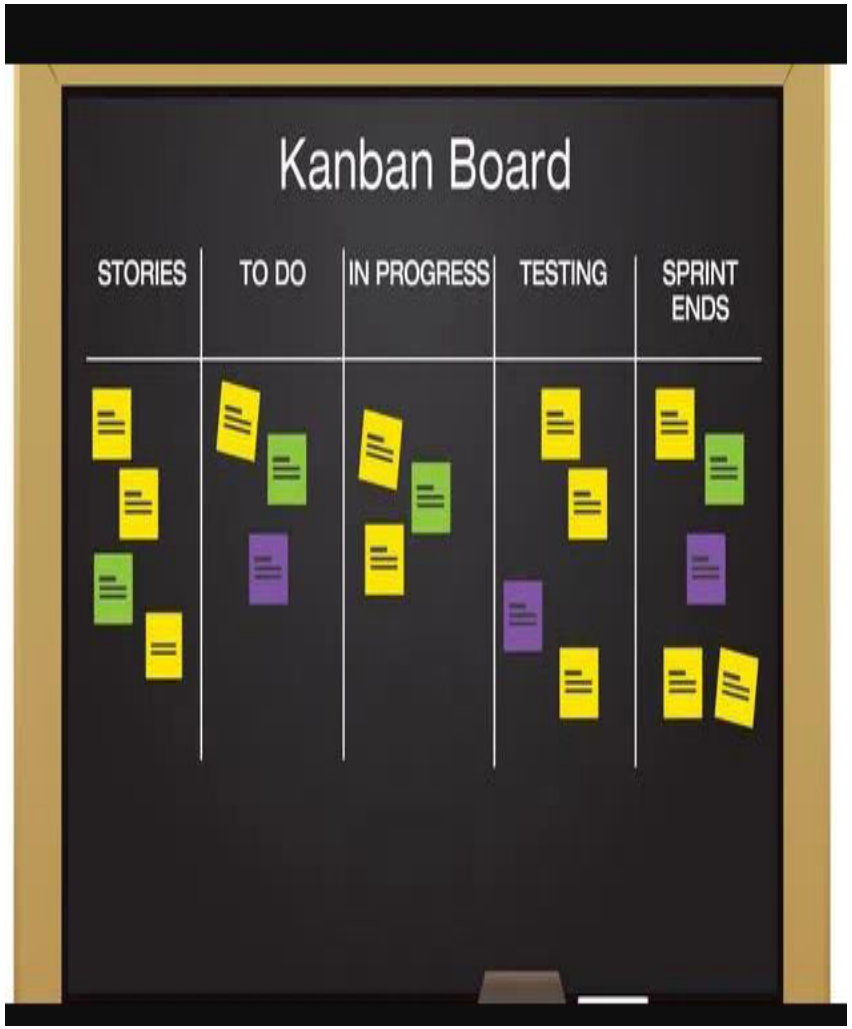
- **Columns** — Another hallmark of the kanban board are the columns. Each column represents a specific activity that together compose a “workflow”.
- Cards flow through the workflow until completion. Workflows can be as simple as “To Do,” “In Progress,” “Complete,” or much more complex.

- **Work In Progress (WIP) Limits** — WIP limits are the maximum number of cards that can be in one column at any given time.
- A column with a WIP limit of three, cannot have more than three cards in it.
- When the column is “maxed-out” the team needs to swarm on those cards and move them forward before new cards can move into that stage of the workflow.
- These WIP limits are critical for exposing bottlenecks in the workflow and maximizing flow.
- WIP limits give you an early warning sign that you committed to too much work.

- **Commitment point** — Kanban teams often have a backlog for their board.
- This is where customers and teammates put ideas for projects that the team can pick up when they are ready.
- The commitment point is the moment when an idea is picked up by the team and work starts on the project.

- **Delivery point** — The delivery point is the end of a kanban team's workflow.
- For most teams, the delivery point is when the product or service is in the hands of the customer.
- The team's goal is to take cards from the commitment point to the delivery point as fast as possible.
- The elapsed time between the two is the called Lead Time.
- Kanban teams are continuously improving to decrease their lead time as much as possible.

To Do	In Progress	Done
Try a Kanban tool	Learn about Kanban	Get a whiteboard
Plan a project using Kanban		Find sticky notes



Kanban vs. scrum board

The difference between kanban and scrum is actually quite subtle. By most interpretations, scrum teams use a kanban board, just with scrum processes, artifacts, and roles along with it. There are, however, some key differences.

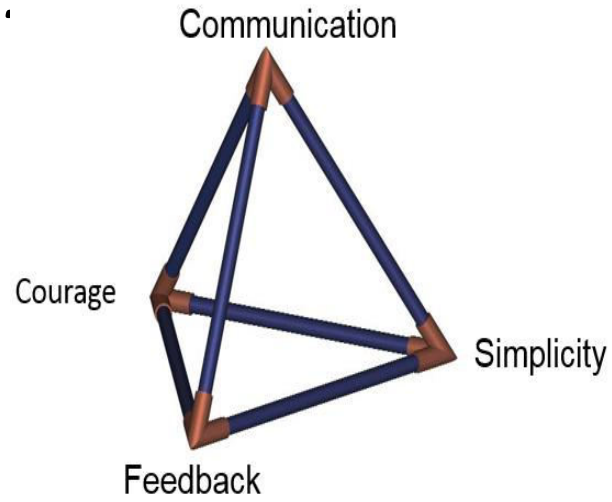
- Scrum sprints have start and stop dates whereas kanban is an ongoing process.
- Team roles are clearly defined in scrum (product owner, dev team, and scrum master), while kanban has no formal roles. Both teams are self-organized.
- A kanban board is used throughout the lifecycle of a project whereas a scrum board is cleared and recycled after each sprint.
- A scrum board has a set number of tasks and strict deadline to complete them.
- Kanban boards are more flexible with regards to tasks and timing. Tasks can be reprioritized, reassigned, or updated as needed.

Extreme Programming(XP)

- Generally being used with Scrum, it can focus on how Agile can increase customer satisfaction, instead of delivering at the entirety, the customer seeks for the near future, it provides them what they demand at present.
- XP is concentrated on regular propaganda and precise development cycles. In addition to that, it implements code review, pair programming and regular communication with customers.

XP method is basically based on the *four simple values*.

- *Uniformity,*
- *Simplicity*
- *Communication,*
- *Feedback and Endurance*.



Why do we need XP?

Common problems of software development:

- Schedule slips
- Business misunderstood
- Defect rate
- ***Management***
- ***Motivation of developers***

XP solutions:

- Short iterations, fast delivery
- Whole team
- Test driven development
- ***Shared understanding***
- ***Humanity and productivity***

XP features

- XP is the most suitable for:
  - Small and medium size projects
  - New technologies
  - Projects with unclear requirements
  - Risky projects
- XP improves skills by cross training
- No more than 20 developers in a team
- Using of XP in life-critical projects is questionable

Extreme programming practices 1

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 ÔTasksÕ
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 2

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 own all the code. Anyone can change anything.
Continuous Integration	As soon as 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 over-time 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.

Pair Programming

- Perhaps the most radical principle of extreme programming is pair programming. Code is developed at the work station by pairs of developers.
- One (the ‘driver’) actually types the code while the other (the ‘navigator’) watches, asks questions, and make suggestions.
- From time to time the pair will change roles.
- In one way this is taking peer review of products a step further.
- Pair programming promotes the production of code that is well structured, easy to understand and relatively free of errors.

Benefits of Agile Methodology

As defined, Agile is a mindset that directly benefits faster, lighter, and more efficient development processes. The process delivers products and services that customers look for, and the entire product development process is more quicker in response to changes;

- **Faster:** One of the major benefits of Agile methodology is faster/speedy development and response. A quicker software development process significantly reduces times between paying and getting paid and leads to a profitable business.
- **Upgrade customer satisfaction:** There is no requirement for a longer queue to get exactly what customers want with agile development. Infact, a swift set of iterations are done very closely for what they look for, very quickly. The system adjusts rapidly to reshape successful customer solutions and adapt it as it would alter the overall environment of product development.

Benefits of Agile Methodology

- **Values executives:** Employees are highly valued when providing productive ideas rather than following a fixed set of rules. Agile methodologies enable employees to set their goals and achieve them appropriately. With these methodologies, employees are in the best position to respond to challenges, resolve obstacles and meet the goals and objectives at hand.
- **Eradicate rework:** Implicating more customers into each phase of the requirements and delivery helps in aligning project on-task and in-tune with customer requirements at each step that lead to less backtracking and save time amid development cycle and customer suggested revisions.

Next Lecture

- Agile Scrum framework -in Detail

Software Enginnering

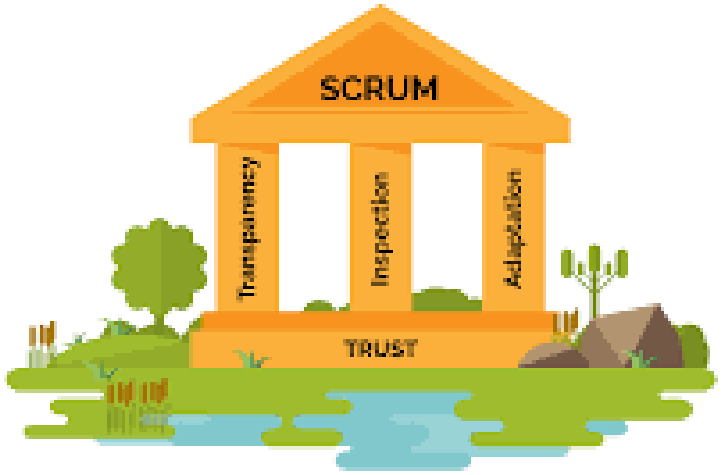
Dr Tripty Singh

- Last Lecture:
- Traditional Approach Vs Agile Approach
- Agile Manifesto
- Agile Methodology

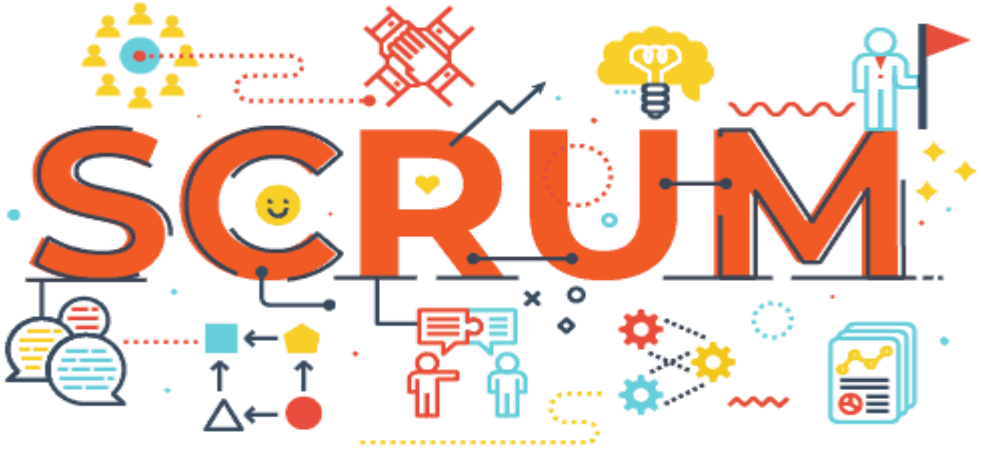
Today’s Lecture

- Agile Scrum Framework in detail

- Scrum is **a management framework that teams use to self-organize and work towards a common goal**.
- It describes a set of meetings, tools, and roles for efficient project delivery.
- Much like a sports team practicing for a big match, Scrum practices allow teams to self-manage, learn from experience, and adapt
- **The three pillars of empiricism at the base of the Scrum framework are:**
- transparency,
- inspection,
- Adaptation to change.
- Trust



Scrum –



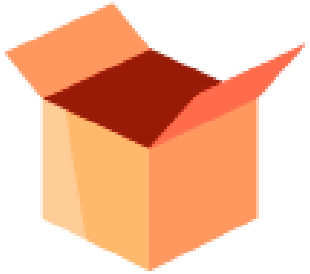
- The scrum framework is heuristic; it's based on **continuous learning** and adjustment to fluctuating factors. It acknowledges that the team doesn't know everything at the start of a project and will evolve through experience.
- Scrum is **structured to help teams naturally adapt to changing conditions and user requirements**, with re-prioritization built into the process and short release cycles so your team can constantly learn and improve.
- While **scrum is structured, it is not entirely rigid**. Its execution can be tailored to the needs of any organization. There are many theories about how exactly scrum teams must work in order to be successful.

SCRUM PROCESS

SCRUM MASTER

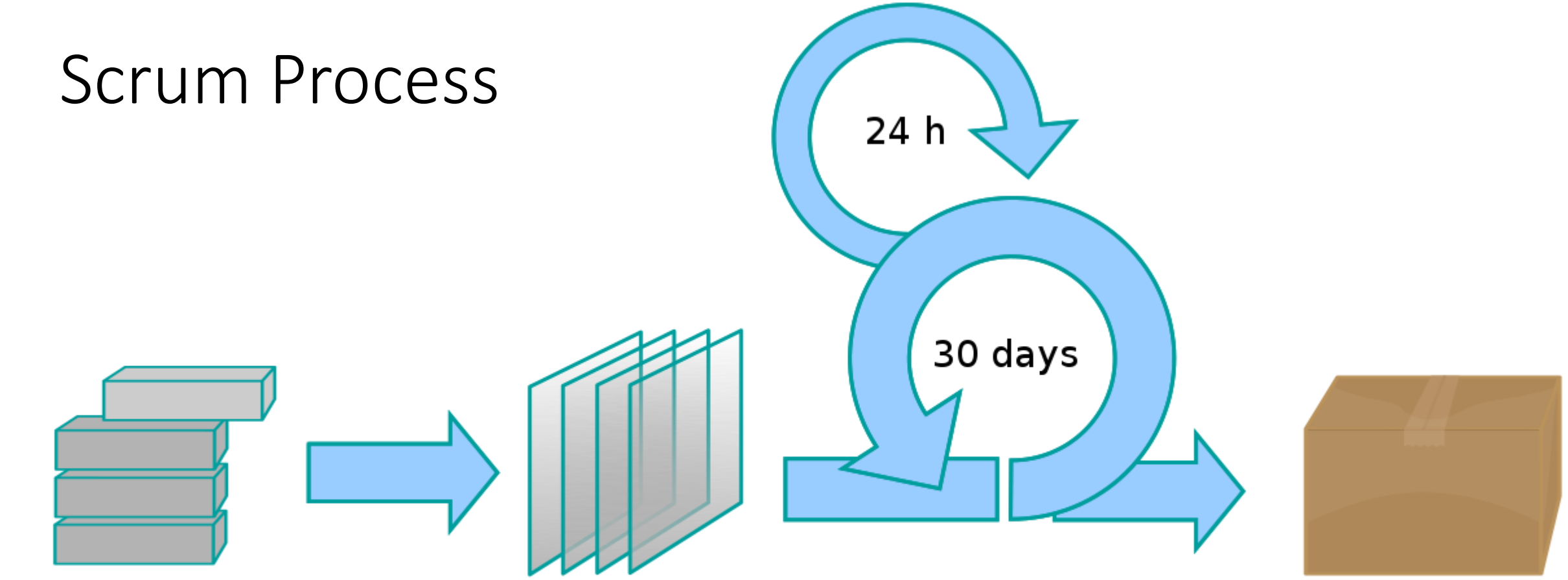
PRODUCT OWNER

USER STORIES

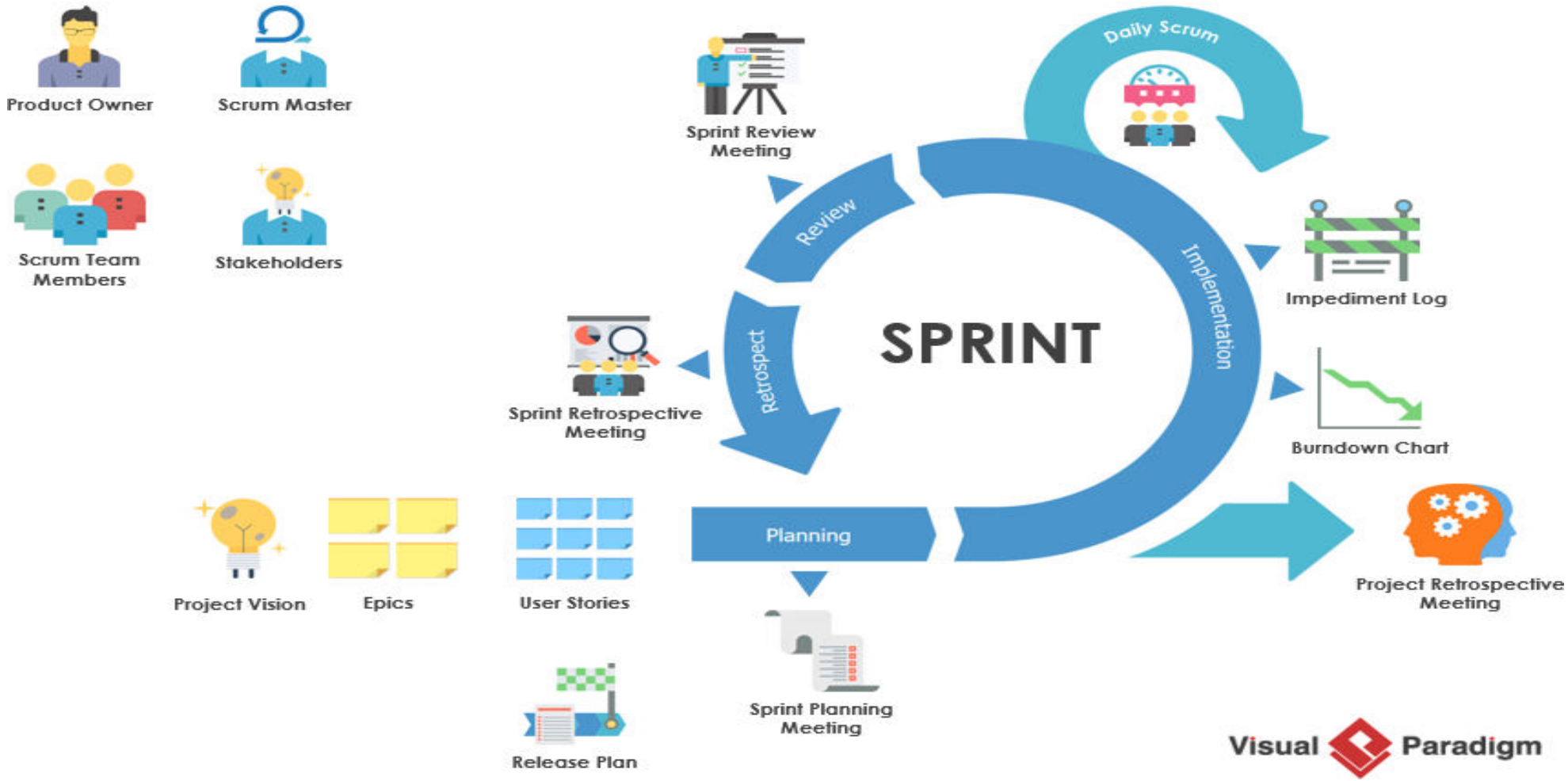


The distinction between Sprint and Scrum is that they are two linked but different terms. Scrum is a framework often used in Agile methodology, and a Sprint is part of Scrum’s framework structure. Scrum gives meetings, tools, and roles, while a Sprint is a defined period for creating a feature.

Scrum Process



The Agile – Scrum Framework



Roles

- Product owner
- Development team
- Scrum master

Artifacts

- Increment
- Product backlog
- Sprint backlog

Ceremonies

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

SCRUM ROLES

PRODUCT OWNER

Represents the client and the business in general for the product on which they're working.

SCRUM MASTER

Responsible for ensuring the team has everything they need to deliver value.

DEVELOPMENT TEAM

A group of cross-functional team members all focused on the delivery of working software.

Three essential roles for scrum success

- A scrum team needs three specific roles:
- product owner,
- scrum master, and
- the development team. [Scrum Team]

And because scrum teams are cross-functional, the development team includes testers, designers, UX specialists, and ops engineers in addition to developers.

The scrum product owner-Setting clear direction

- Product owners are the champions for their product. They are focused on understanding business, customer, and market requirements, then prioritizing the work to be done by the engineering team accordingly.

The scrum product owner

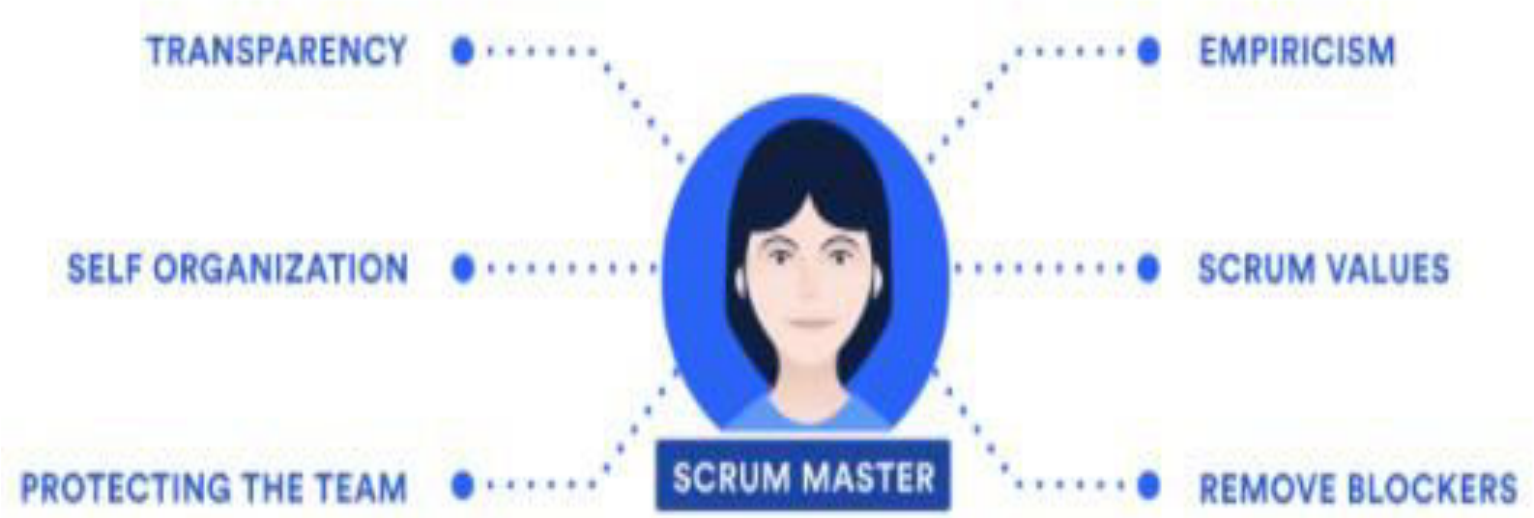
Effective product owners:

- Build and manage the product backlog.
- Closely partner with the business and the team to ensure everyone understands the work items in the product backlog.
- Give the team clear guidance on which features to deliver next.
- Decide when to ship the product with a predisposition towards more frequent delivery.
- The product owner is not always the product manager. Product owners focus on ensuring the development team delivers the most value to the business. Also, it's important that the product owner be an individual. No development team wants mixed guidance from multiple product owners.

The scrum master-Holding it all together

- Scrum masters are the champions for scrum within their teams. They coach teams, product owners, and the business on the scrum process, and look for ways to fine-tune their practice of it.
- An effective scrum master deeply understands the work being done by the team and can help the team optimize their transparency and delivery flow.
- As the facilitator-in-chief, he/she schedules the needed resources (both human and logistical) for sprint planning, stand-up, sprint review, and the sprint retrospective.

The scrum master-Holding it all together



The scrum development team-Redefining “developer”

- They are the champions for sustainable development practices. The most effective scrum teams are tight-knit, co-located, and usually five to seven members.
- Team members have differing skill sets, and cross-train each other so no one person becomes a bottleneck in the delivery of work. Strong scrum teams are self-organising and approach their projects with a clear ‘we’ attitude.
- All members of the team help one another to ensure a successful sprint completion.

- The scrum team drives the plan for each sprint.
- They forecast how much work they believe they can complete over the iteration using their historical velocity as a guide.
- Keeping the iteration length fixed gives the development team important feedback on their estimation and delivery process, which in turn makes their forecasts increasingly accurate over time.

EVENTS/CEREMONIES IN AGILE METHODOLOGY (SCRUM)

1

SPRINT PLANNING

To craft a Sprint Goal and a plan to achieve it

2

DAILY SCRUM

To review team's progress on Sprint Goal and adjust the plan to get there

3

SPRINT REVIEW

To review the work completed in the sprint with stakeholders

4

SPRINT RETROSPECTIVE

To review the product development process and identify areas for continuous improvements

SPRINT BACKLOG

1.REGISTER

2. LOG IN

Scrum ceremonies or events

- Some of the more well-known components of the scrum framework are the set of sequential events, ceremonies, or meetings that scrum teams perform on a regular basis. The ceremonies are where we see the most variations for teams. For example, some teams find doing all of these ceremonies cumbersome and repetitive, while others use them as a necessary check-in.

key events a scrum team might partake in

Organize the backlog:

- Sometimes known as backlog grooming, this event is the responsibility of the product owner.
- The product owner’s main jobs are to drive the product towards its product vision and have a constant pulse on the market and the customer.
- Therefore, he/she maintains this list using feedback from users and the development team to help prioritize and keep the list clean and ready to be worked on at any given time

key events a scrum team might partake in

Sprint planning:

- The work to be performed (scope) during the current sprint is planned during this meeting by the entire development team.
- This meeting is led by the scrum master and is where the team decides on the sprint goal.
- Specific use stories are then added to the sprint from the product backlog.
- These stories always align with the goal and are also agreed upon by the scrum team to be feasible to implement during the sprint.

At the end of the planning meeting, every scrum member needs to be clear on what can be delivered in the sprint and how the increment can be delivered.

key events a scrum team might partake in

Sprint :

- A sprint is the actual time period when the scrum team works together to finish an increment.
- Two weeks is a pretty typical length for a sprint, though some teams find a week to be easier to scope or a month to be easier to deliver a valuable increment.
- Dave West, from Scrum.org advises that the “more complex the work and the more unknowns, the shorter the sprint should be. But it’s really up to your team, and you shouldn’t be afraid to change it if it’s not working! “
- During this period, the scope can be re-negotiated between the product owner and the development team if necessary. This forms the crux of the empirical nature of scrum.

key events a scrum team might partake in

Daily scrum or stand up:

This is a daily super-short meeting that happens at the same time (usually mornings) and place to keep it simple.

Many teams try to complete the meeting in 15 minutes, but that's just a guideline. This meeting is also called a '**daily stand-up**' emphasizing that it needs to be a quick one.

The goal of the daily scrum is for everyone on the team to be on the same page, aligned with the sprint goal, and to get a plan out for the next 24 hours.

The stand up is the time to voice any concerns you have with meeting the sprint goal or any blockers.

A common way to conduct a stand up is for every team member to answers three questions in the context of achieving the sprint goal:

- What did I do yesterday?
- What do I plan to do today?
- Are there any obstacles?

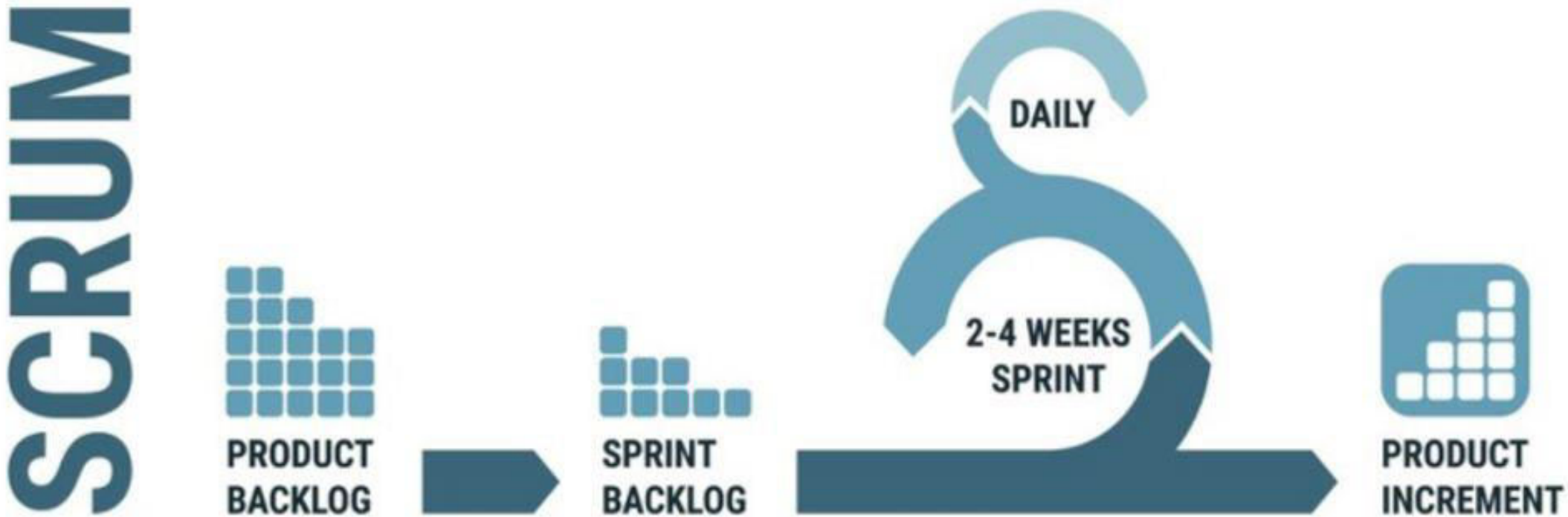
key events a scrum team might partake in

- **Sprint review**:
- At the end of the sprint, the team gets together for an informal session to view a demo of, or inspect, the increment.
- The development team showcases the backlog items that are now ‘Done’ to stakeholders and teammates for feedback.
- The product owner can decide whether or not to release the increment, although in most cases the increment is released.

  This review meeting is also when the product owner reworks the product backlog based on the current sprint, which can feed into the next sprint planning session.
- For a one-month sprint, consider time-boxing your sprint review to a maximum of four hours.

key events a scrum team might partake in

- **Sprint retrospective**:
- The retrospective is where the team comes together to document and discuss what worked and what didn't work in a sprint, a project, people or relationships, tools, or even for certain ceremonies.
- The idea is to create a place where the team can focus on what went well and what needs to be improved for the next time, and less about what went wrong.



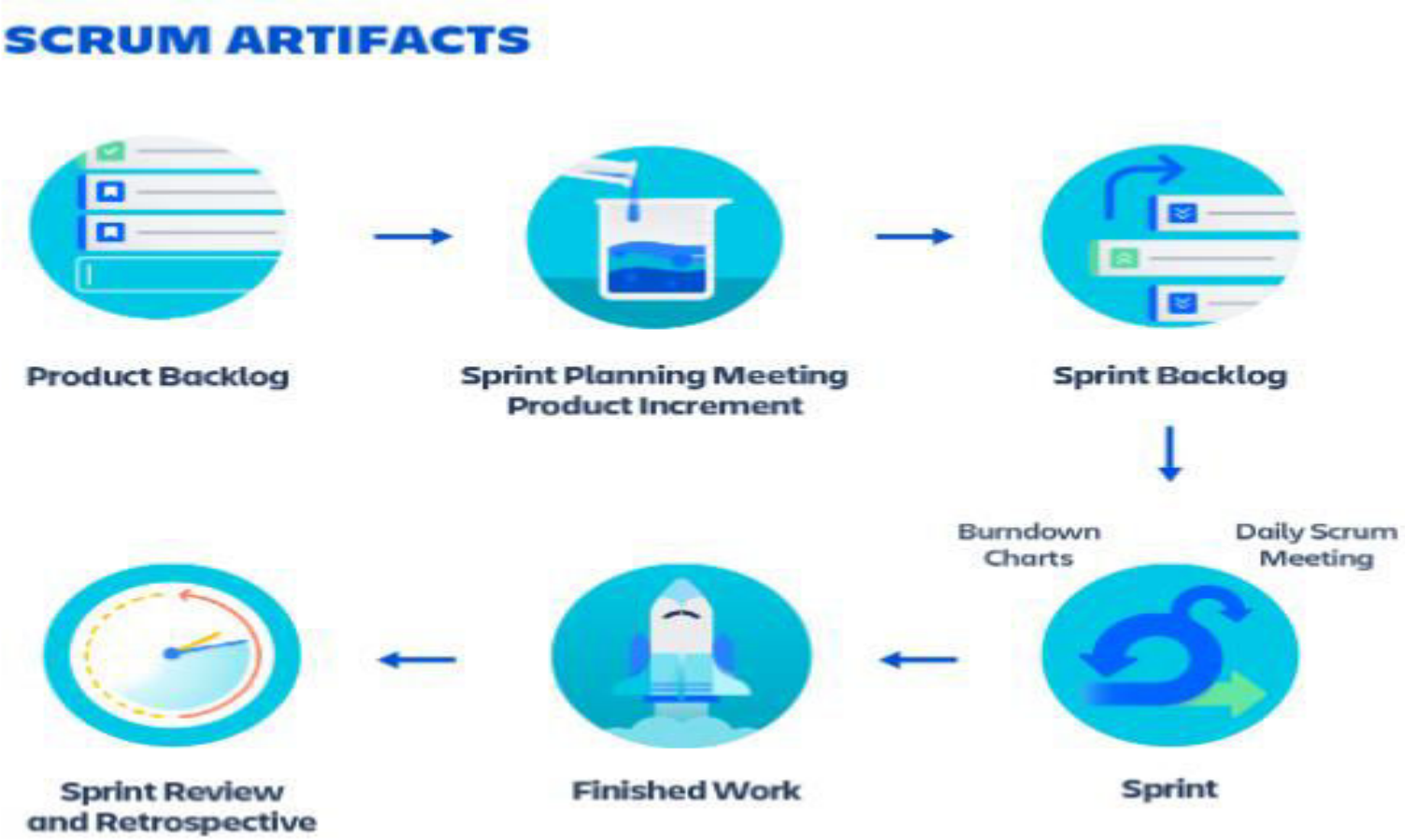
Artifacts in Scrum Framework - agile

Agile scrum artifacts

- Agile scrum artifacts are information that a scrum team and stakeholders use to detail the product being developed, actions to produce it, and the actions performed during the project.
- These artifacts provide metadata points that give insight into the performance of a sprint.
- They are essential tools for every scrum team since they enable core scrum attributes of transparency, inspection, and adaption.

- Artifacts are created during the main activities of a scrum sprint:
- Plan work and future goals
- Create tasks to achieve these goals
- Organize tasks into sprints based on dependency and priority
- Execute the tasks
- Review and analyze results in order to compare to the goals
- Repeat these steps

The main artifacts of agile scrum



Product backlog

- The product backlog is a list of new features, enhancements, bug fixes, tasks, or work requirements needed to build a product. It's compiled from input sources like customer support, competitor analysis, market demands, and general business analysis.

Sprint backlog

- The sprint backlog is a set of product backlog tasks that have been promoted to be developed during the next product increment.
- Sprint backlogs are created by the development teams to plan deliverables for future increments and detail the work required to create the increment.
- Sprint backlogs are created by selecting a task from the product backlog and breaking that task into smaller, actionable sprint items.

- Consider an example task like “build a shopping cart page”, which requires many design and development subtasks.
- The product backlog is home to the primary task while the supporting tasks like “create a shopping cart visual design mockup” or “program the shopping cart sessions” are housed in the sprint backlog.

- The sprint backlog is updated during the sprint planning phase of scrum.
- The smaller sprint tasks are assigned to the relevant teams like design and development.
- If a team does not have the capacity to deliver all the sprint tasks, the remaining sprint tasks will standby in the sprint backlog for a future sprint.

Product increment

- A product increment is the customer deliverables that were produced by completing product backlog tasks during a sprint.
- It also includes the increments of all previous sprints. There is always one increment for each sprint and an increment is decided during the scrum planning phase.
- An increment happens whether the team decides to release to the customer.
- Product increments are incredibly useful and complementary to CI/CD in version tracking and, if needed, version rollback.

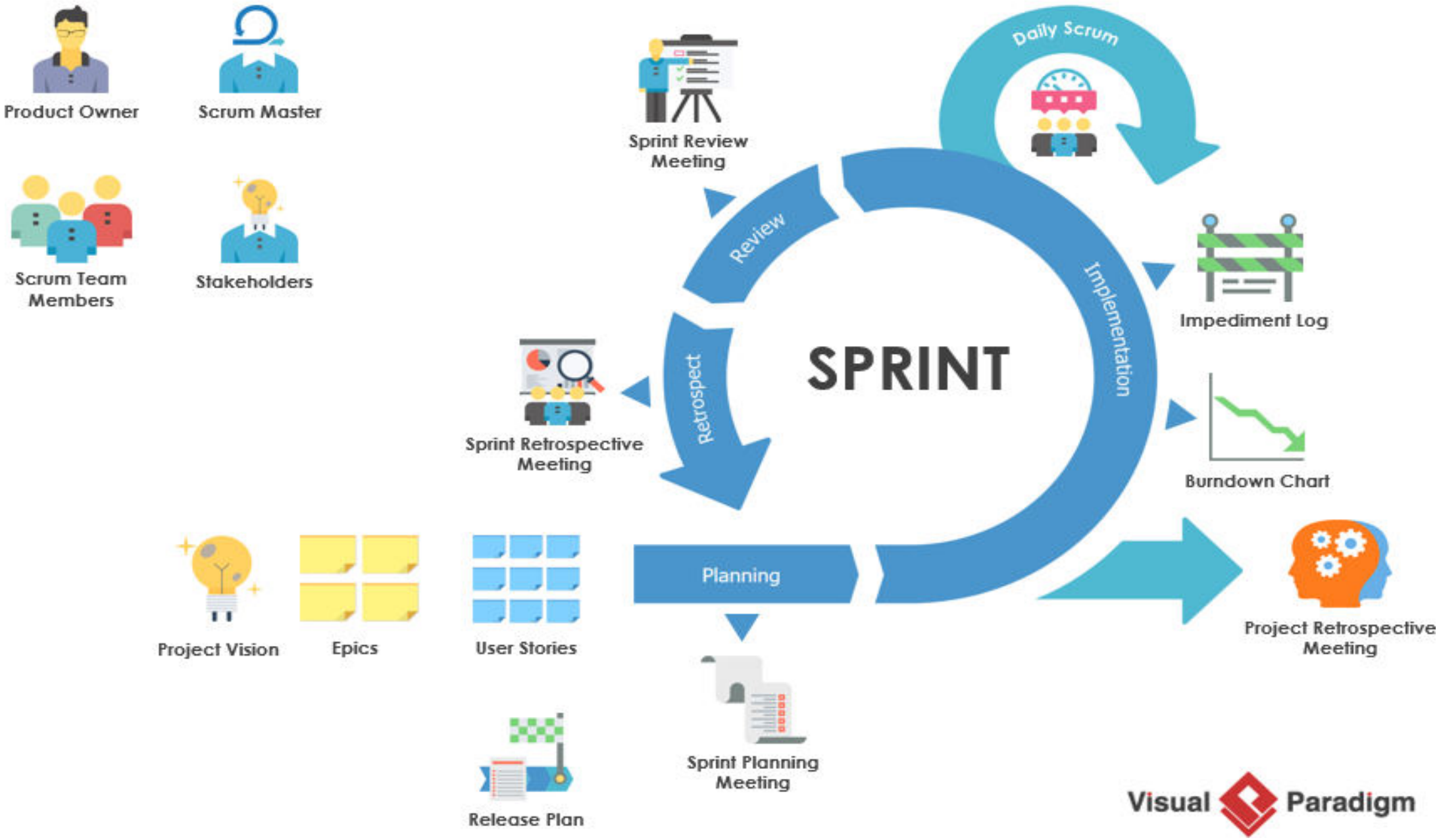
Artifact transparency

- Scrum artifacts are powerful aids that help teams operate more efficiently. Therefore, it's important all teams have access and visibility into the artifacts.
- Product owners and scrum masters need to make it a regular practice to review and discuss artifacts with development teams.
- This will help teams stay aware of operational inefficiencies and produce creative ways to improve velocity.

Last Lecture

- **Scrum**:
- Scrum process,
- Scrum roles - Product Owner, ScrumMaster, Team, Project Manager, product manager,
- Scrum Events,
- Scrum artifacts

The Agile – Scrum Framework



Roles

- Product owner
- Development team
- Scrum master

Artifacts

- Increment
- Product backlog
- Sprint backlog

Ceremonies

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

SCRUM ROLES

PRODUCT OWNER

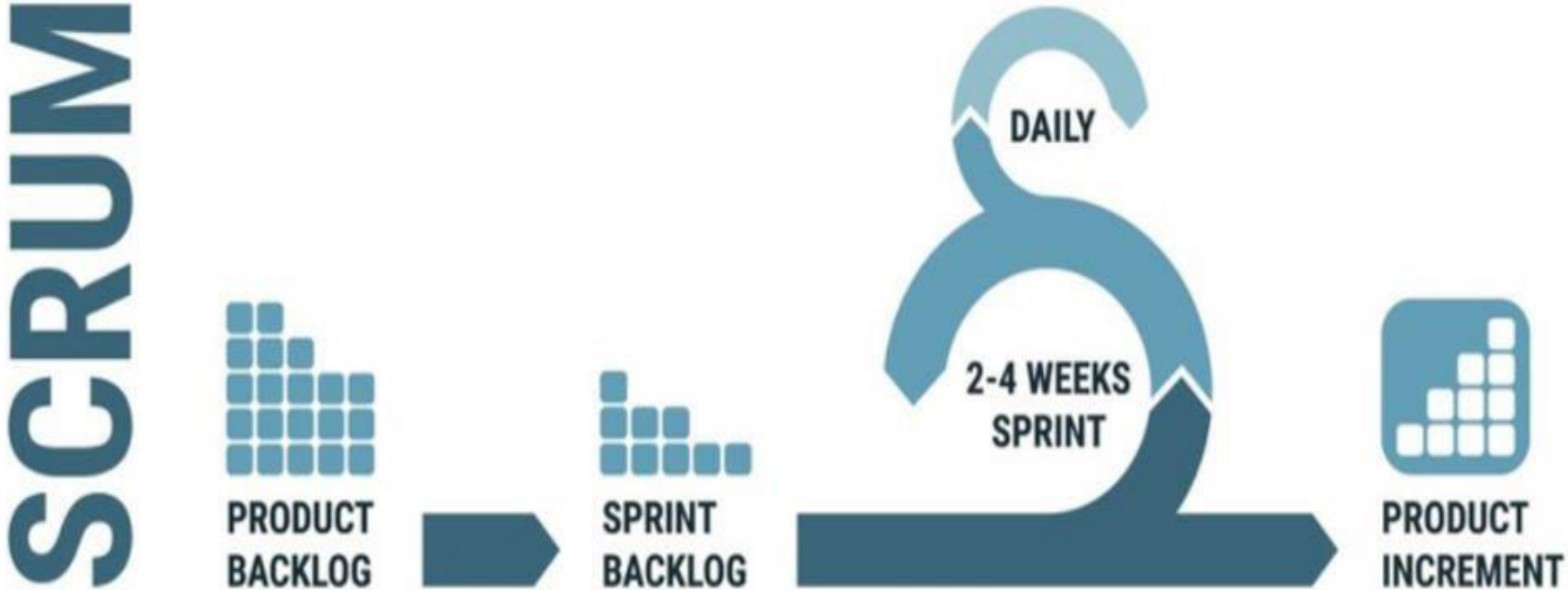
Represents the client and the business in general for the product on which they're working.

SCRUM MASTER

Responsible for ensuring the team has everything they need to deliver value.

DEVELOPMENT TEAM

A group of cross-functional team members all focused on the delivery of working software.



Artifacts in Scrum Framework - agile

EVENTS/CEREMONIES IN AGILE METHODOLOGY (SCRUM)

1

SPRINT PLANNING

To craft a Sprint Goal and a plan to achieve it

2

DAILY SCRUM

To review team's progress on Sprint Goal and adjust the plan to get there

3

SPRINT REVIEW

To review the work completed in the sprint with stakeholders

4

SPRINT RETROSPECTIVE

To review the product development process and identify areas for continuous improvements

SPRINT BACKLOG

1.REGISTER

2. LOG IN

Today’s Lecture

- **Product Inception:**
- Product vision,
- stakeholders,
- initial backlog creation

Product Inception

- Every product should begin with a problem that needs to be solved.
- Product Inception is the process by which we pin down the problem in concrete terms and determine how we are going to solve it—all before writing a single line of code.
- Representatives from the design and development team as well as stakeholders must be present throughout the process in order for Product Inception to be effective.
- By the end of a Product Inception session, everyone should be on the same page about where we're going, why we're going there, and how we are going to arrive at our destination.

Why do we believe in Product Inception?

- Product Inception is crucial to developing a product that truly solves the problem we intended to solve and helps users meet their goals.
- Even if a problem seems obvious, there are always multiple perspectives to consider; there may be aspects of a problem we can't see until a stakeholder brings it to our attention.
- In addition, everyone may have a different idea about what an effective solution looks like.

Why do we believe in Product Inception?

- Product Inception, then, brings everyone to the table so we can come up with a unified vision for the product and agree on a road map to take us there.
- The collaborative nature of Product Inception ensures that we include everyone's point of view, thus allowing us to produce ideas that are even better than what each of us could have thought of individually.

How It Works

- Product Inception involves a series of exercises that helps us define
  - why we are building the product,
  - how to best fulfill users' needs,
  - what features the product will include,
  - and which parts of the product are high priorities.

The different phases of Product Inception that we'll explore in this series include:

- Defining the main problem and solution with a **Product Vision Board**. This exercise enables us to articulate what exactly we want to achieve with this product.

Product Vision Board

THE PRODUCT VISION BOARD

romanpichler

VISION

What is your purpose for creating the product?

Which positive change should it bring about?

- Making the long-term home rental process for both tenants and landlords more simple and global
- Will save plenty of time for both tenants and landlords while contributing to build trust between them

TARGET GROUP

Which market or market segment does the product address?

Who are the target customers and users?

- Tenants looking for a new accommodation
- Tenants seeking to save time in unnecessary visits
- Tenants looking for a new accommodation in a foreign country
- Landlords seeking to manage digitally the whole rental process
- Landlords who desire to secure the payment of the rent
- Landlords who hesitate to rent to a foreigner

NEEDS

What problem does the product solve?

Which benefit does it provide?

- It helps landlords to rent their property at the right price
- It helps landlords to handle to whole rental process
- It helps landlords to assess tenants' solvency
- It prevents tenants from visiting unnecessary accommodations

PRODUCT

What product is it?

What makes it stand out?

Is it feasible to develop the product?

- The product is a long-term home rental platform and price negotiation platform that connects tenants with landlords
- It automates all the standard rental tasks (submitting personal information, customer offers, eSigning rental agreement, sending maintenance requests, etc.)
- The MVP can be developed in approximately 3 months

BUSINESS GOALS

How is the product going to benefit the company?

What are the business goals?

- The product will complete the existing real estate marketplace
- The platform will create a huge traffic increase across all our products
- It will offer a one-stop-shop for real estate asset managers and property managers

THE PRODUCT VISION BOARD EXTENDED

romanpichler

VISION

What is your motivation for creating the product?

Which positive change should it bring about?

TARGET GROUP

Which market or market segment does the product address?

Who are the target customers and users?

NEEDS

Which problem does the product solve?

What benefit does it provide?

PRODUCT

What product is it?

What makes it stand out?

Is it feasible to develop the product?

BUSINESS GOALS

How is the product going to benefit the company?

What are the business goals?

COMPETITORS

Who are your main competitors?

What are their strengths and weaknesses?

REVENUE STREAMS

How can you monetise your product and generate revenues?

COST FACTORS

What are the main cost factors to develop, market, sell, and service the product?

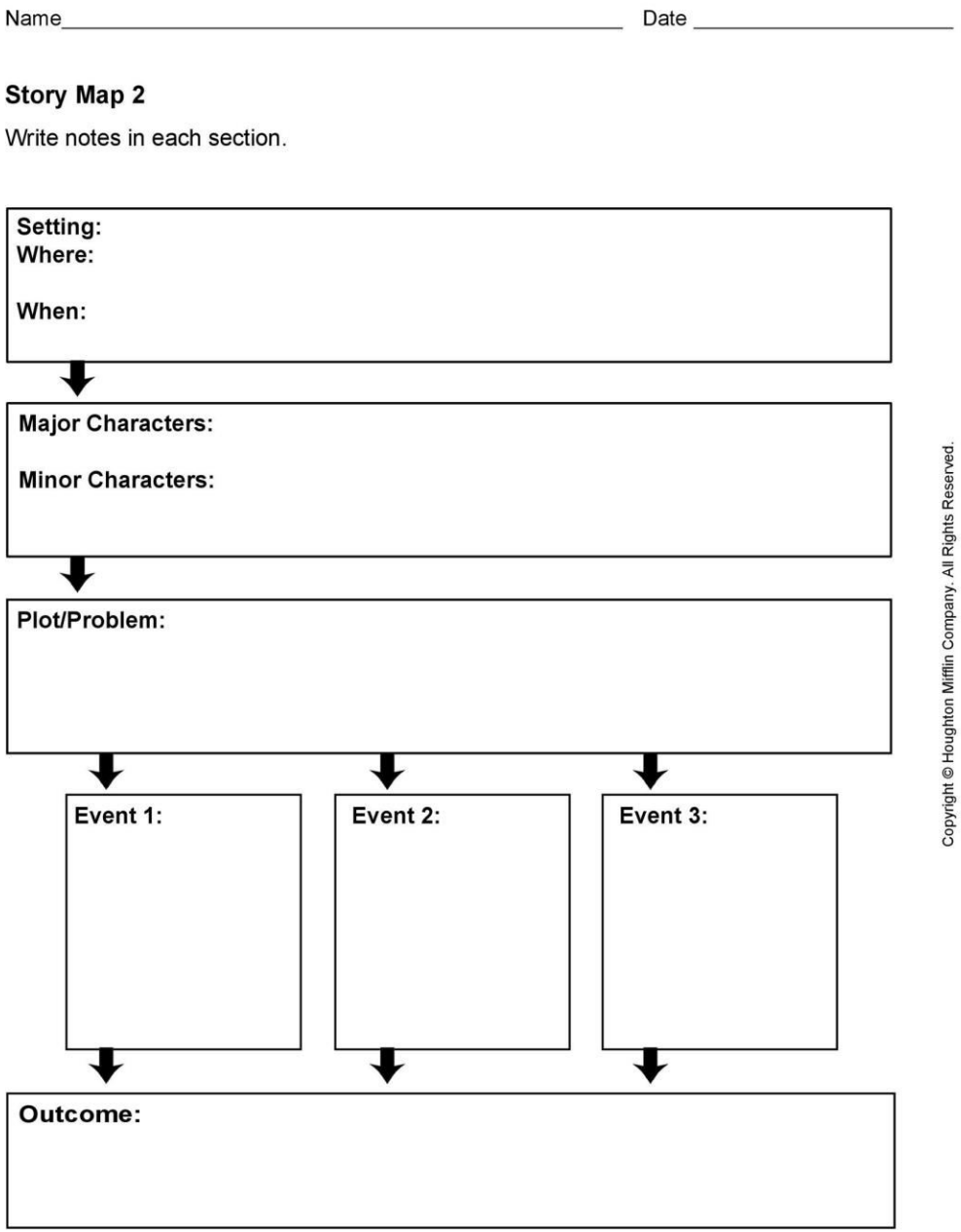
CHANNELS

How will you market and sell your product?

Do the channels exist today?

How It Works

- Ensuring the product is relevant and identifying which product attributes to prioritize through **Stakeholder Analysis**.
- Understanding users' goals and related needs through **Persona Analysis**.
- Visualizing the functionality of the product, identifying necessary features, and planning the first release with a **Story Map**.



Name_____ Date _____

Story Map 2

Write notes in each section.

Setting:
Where:

When:

Major Characters:

Minor Characters:

Plot/Problem:

Event 1:

Event 2:

Event 3:

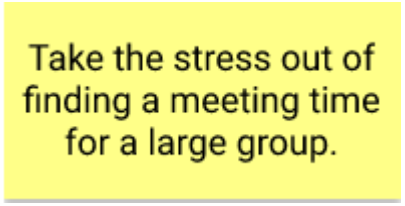
Outcome:

The Product Vision Board: The First Step to Discovering a Successful Software Product

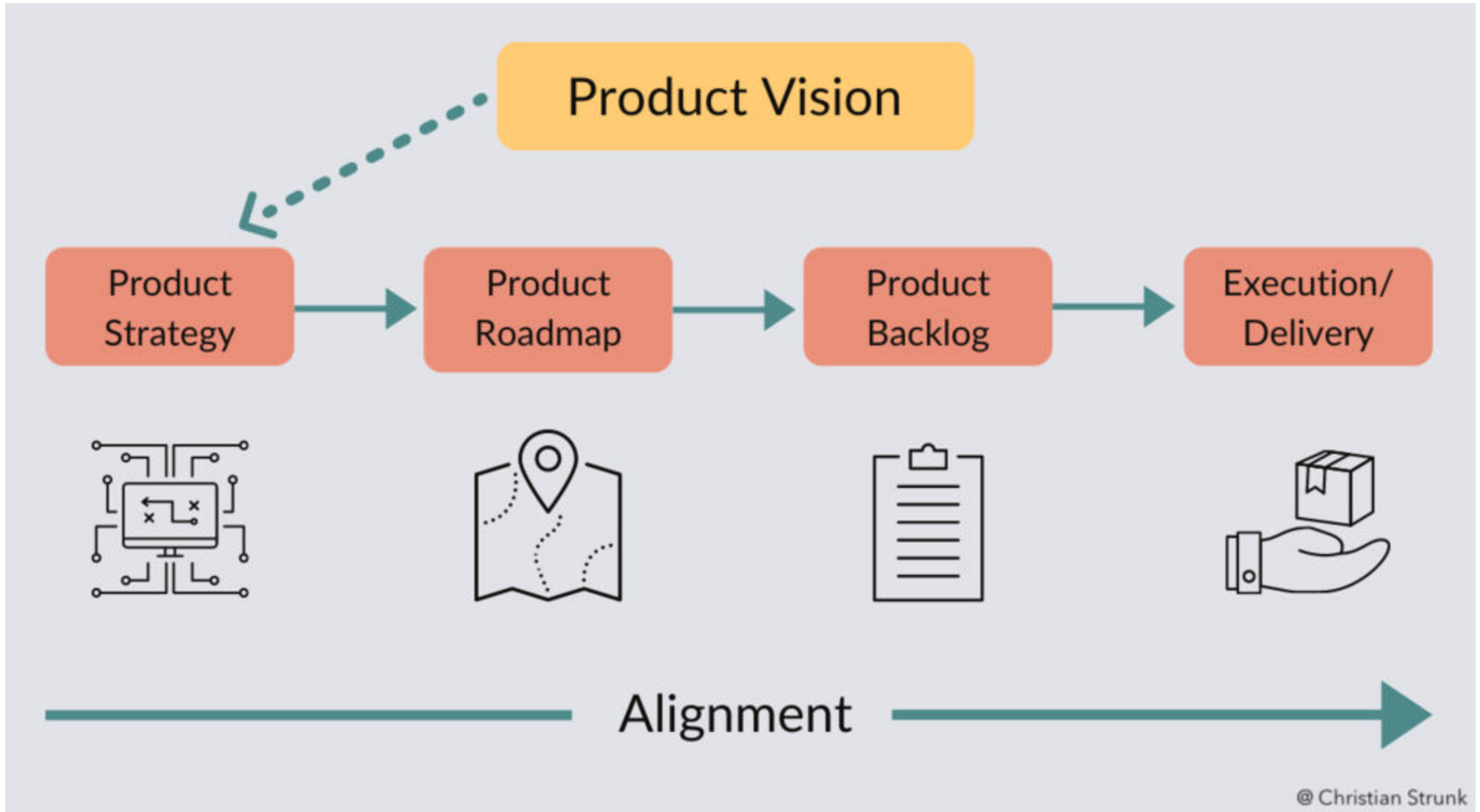
- A concrete product vision is crucial for any software product. It tells you where you are going, why you are going there, and who you are going there for.
- A well-defined product vision is the foundation to a successful development process, while an unclear product vision will often leave your development team confused, making it difficult to prioritize user stories and features.
- In the end, without product vision, your users may not even understand what your product does.

What is Product Vision?

- The product vision is a short statement that encapsulates what you are trying to achieve with a product.
- It sounds simple, but in reality it can be quite challenging.
- Say for example, we want to create a calendar application for finding meeting times. That sounds great, but that's not a clear product vision.
- Product vision isn't actually about the product at all, it's about what the product will do and what problem it will solve. A much better product vision would be:



- This tells us exactly what we want to achieve, which will keep us focused on track throughout the development process.


Product Vision
Product Strategy
Product Roadmap
Product Backlog
Execution/ Delivery
Alignment
@ Christian Strunk

Source: www.romanpichler.com

PRODUCT VISION CANVAS

The Product Vision Board Extended

Vision :

- What is your motivation for creating the product?
- Which positive change should it bring about?

Target Group

- Which market or market segment does the product address?
- Who are the target customers & users?

Needs

- Which problem does the product solve?
- What benefit does it provide?

Product

- What product is it?
- What makes it stand out?

Business Goals

- How's the product going to benefit the company?
- What are the business goals?

Competitors

- Who are your main competitors?
- What are their strengths and weaknesses?

Revenue Streams

- How can you monetize your product and generate revenues?

Cost Factors

- What are the main cost factors in developing, market, sell and service the product?

Channels

- How will you market and sell your product?
- Do the channels exist today?

Product Vision Board

The Product Vision Board

SOPHiLABS

Vision

What is your purpose for creating the product?

What positive change should it bring about?

Target Group

Which market or market segment does the product address?

Who are the target customers and users?

Needs

What problem does the product solve?

What benefit does it provide?

Product

What product is it?

What makes it stand out?

Is it feasible to develop the product?

Business Goals

How is the product going to benefit the company?

What are the business goals?

The Product Vision Board

SOPHiLABS

Vision

What is your purpose for creating the product?

What positive change should it bring about?

Take the stress out of finding a meeting time for a large group.

Target Group

Which market or market segment does the product address?

Who are the target customers and users?

Our Company

Employees who plan meetngs

Employees who attend meetngs

Needs

What problem does the product solve?

What benefit does it provide?

Easy to use

Simple

Enable employees to quickly find a meeting time that will work for all participants

Product

What product is it?

What makes it stand out?

Is it feasible to develop the product?

Web and mobile app that automatically generates time that everyone invited to a meeting is free.

Sync with major calendar applications

Clean and simple design, no frills

Easy to use, does work for you

Business Goals

How is the product going to benefit the company?

What are the business goals?

Decrease time spent planning meetings and emailing back and forth about schedules

Increase Productivity

Boost employee morale

Maximize the number of people in important meetings

Completed Product Vision Board, Product Vision Board example

Stakeholder Analysis: Another Key Piece of Product Inception

- Stakeholders are a vital part of any software project, which makes Stakeholder Analysis a critical part of the Product Inception phase.
- The only way to make sure that we are building a product that is relevant and valuable is to engage people and understand why they may be interested in the product and bring them into the process.
- If we don't involve stakeholders from the very beginning, we can easily build an excellent product with great features, but it's possible that no one will want or need to use it.

What is a stakeholder?

- A stakeholder is any person or group of people that have a specific interest in the product we are going to build.
- This will of course include end users and customers but can also include, for example, a leadership team, marketing department, investors, or any number of other people who have an interest or "stake" in the product and its developments.
- We've already begun the process of identifying stakeholders on the Product Vision Board, when we listed Target Groups who can also be considered stakeholders.

- Say for example, we want to develop a flower delivery app where users can log in and order flowers from a local florist to be delivered anywhere in the city. Our stakeholders might include:
- **Customers/End Users**: These are the people who will sign up and use your app to order flowers.
- **Flower Shops**: These are the people who you will need to convince to affiliate with your app and be vendors.
- **Marketing Staff**: They have to market the product and make it desirable for users
- **Investors**: They have a financial interest in the success of the product.

What is Stakeholder Analysis?

- Stakeholder Analysis is a process of understanding your stakeholders' background and context in order to pinpoint what they want and contrast that with what they actually need.
- Once we've identified our stakeholders, we want to ask questions like:
  - Who are the stakeholders?
  - Where do they come from?
  - What is their common background?
  - What are they looking for, as a group, from the project/product?
  - What is important to them?
  - Do they have any specific goals as a group?

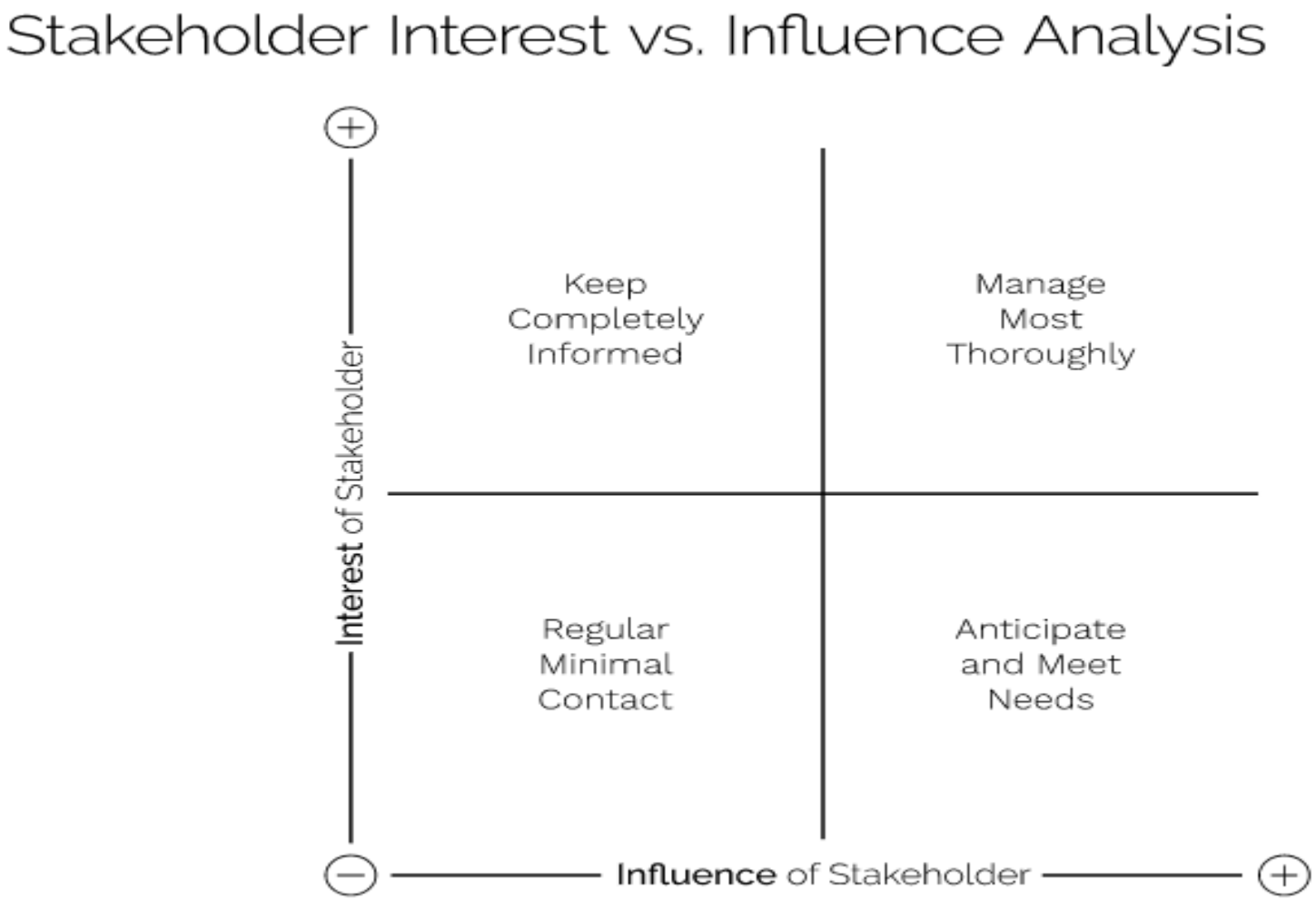
What is Stakeholder Analysis?

- During Stakeholder Analysis, we need to determine not only who our stakeholders are but also what is their relationship to the project as well as figure out which stakeholders' concerns we will need to prioritize throughout the project.

Stakeholder Analysis Techniques

- Product Vision Board
- During the process of creating a **product vision board**, we identified broad target groups, who are ultimately stakeholders
- Stakeholder Mapping: Influence vs. Interest Chart
- Every software development project will have multiple stakeholders and at some point, their needs are going to be in conflict.
- Stakeholder Mapping on an Influence vs. Interest Chart help us to determine each stakeholder's relationship to the product and how we will need to engage them throughout the process.

Stakeholder Mapping: Influence vs. Interest Chart



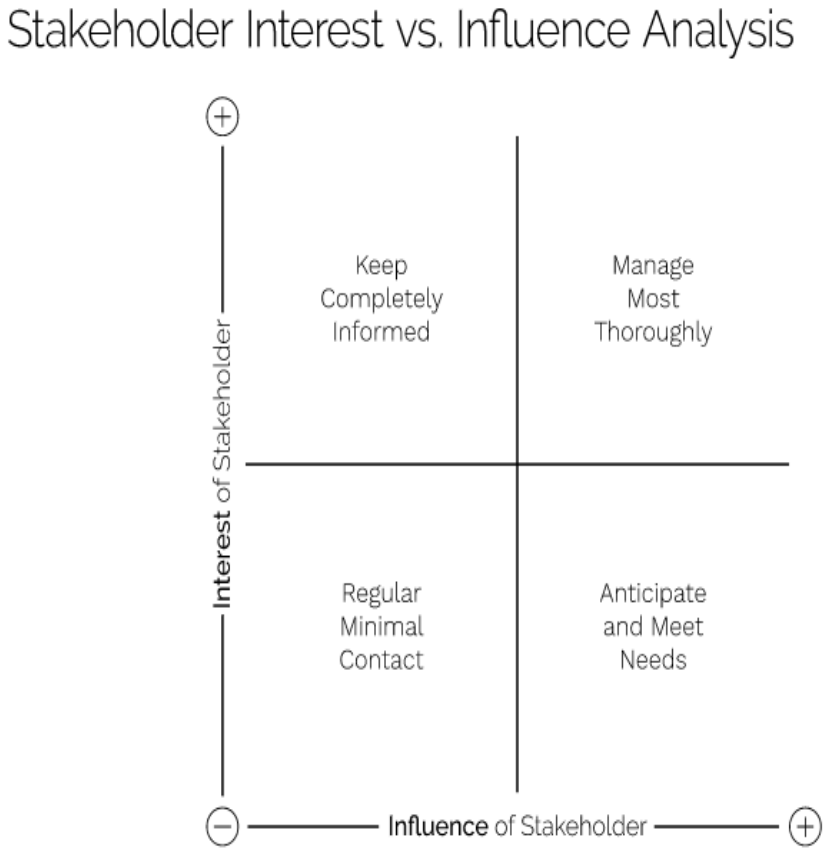
Flower App Delivery example revisited

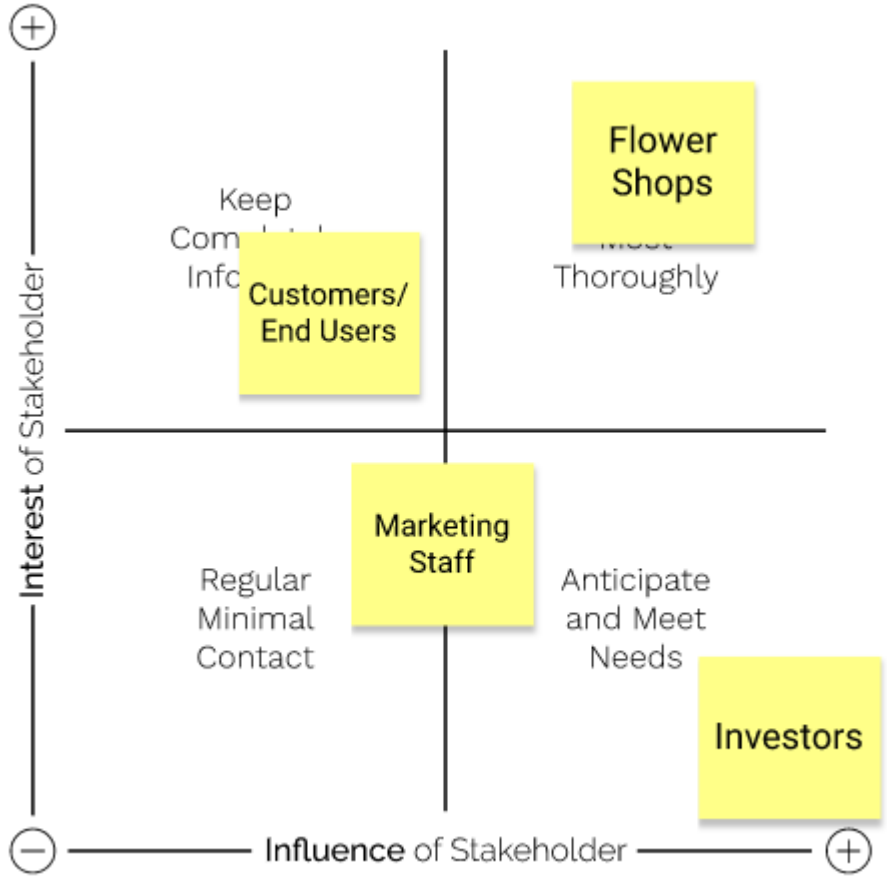
- The chart maps Influence on the x-axis and Interest on the y-axis.
- The more Influence a stakeholder has, the more necessary it will be to meet their needs.
- And the more Interest that they have, the more we will need to keep them informed about the product.
- When mapping the stakeholders, we need to keep the product at the center at all times.

Flower App Delivery example revisited

- Our investors will have a lot of influence over the product, because they are the ones financing it. However, their primary interest in this case is in the revenue received from a successful product, rather than the product itself. For **this reason, they are High Influence, Low Interest.**
- In contrast, our flower shops have a lot of influence as well, because their buy-in is necessary for the app's success. However, they also have a lot of interest in the product, as it is a tool for them to increase their sales and improve their business.
- This makes them a **relatively High Influence, High Interest stakeholder that we will have to manage thoroughly.**

In our example, we might map our stakeholders in this way:





Initial backlog creation

- Product Backlog is a dynamic prioritized source for all the work that needs to be done to deliver a product. This artifact is product focused not system focused.
- A product backlog consists of:
- Features
- Enhancements
- Defects
- Non IT Deliverables

- Example:
- As the mail room I must scan mortgage applications from the customer so that they can be processed
- As the system I must load scanned images so they are available to the indexing user
- As the indexer I want to be able to view a list of all applications requiring indexing
- What information do we capture in the product backlog?

- The details below provide a guideline to what information you may want to capture in a product backlog

Item	Description
Reference	Unique identifier for quick reference during collaboration
Description	A high level description of the item
Type	Feature, New Feature, Enhancement, Defect
Group	This can also be known as a feature set but is simply a grouping of functionality for example: Customer Management Contact Management
Relative Value	A factor that identifies the business value (Scale of 1 to 10, 10 been the highest business value)
Initial Estimate	This can be in story point or ideal days
Adjustment Factor	A summation of factors that may increase the initial estimate
Actual Estimate	Initial Estimate X (1 + Adjustment Factor)
Release	The release number that the PBI will be included in
Sprint (Iteration)	The sprint that the PBI has been allocated to
Notes	Additional comments, references or points of interest, be careful not to get into too much detail, some people prefer this to be separate
Status	Done or Not Done
Committed	Has this item been committed by the product owner i.e it has been decided that it will be incorporated into the product

Example

Ref	Description	Type	Group	Relative Value	Initial Estimate	Adjust Factor	Estimate	Release	Sprint	Status
01	As the mail room I must scan mortgage applications from the customer so that they can be processed	Feature	Scanning and Imaging	10	3	0	3	1	1	Not Done

Next Session

- **Agile Requirements** –
- User personas,
- story mapping,
- user stories,
- 3Cs, INVEST,
- Acceptance criteria, sprints, requirements,
- product backlog and backlog grooming;

Last Lecture

- **Product Inception:**
- Product vision,
- stakeholders,
- initial backlog creation

The different phases of Product Inception that we'll explore in this series include:

- Defining the main problem and solution with a **Product Vision Board**. This exercise enables us to articulate what exactly we want to achieve with this product
- Ensuring the product is relevant and identifying which product attributes to prioritize through **Stakeholder Analysis**.

- **Initial Product Backlog Creation**.

Today’s Session

- **Agile Requirements** –
- User personas,
- story mapping,
- user stories,
- 3Cs, INVEST,
- Acceptance criteria, sprints, requirements,
- product backlog and backlog grooming;

How It Works

- Ensuring the product is relevant and identifying which product attributes to prioritize through **Stakeholder Analysis**.
- Understanding users' goals and related needs through **Persona Analysis**.
- Visualizing the functionality of the product, identifying necessary features, and planning the first release with a **Story Map**.

- A good requirement should tell each audience member exactly what the expected functionality is.

Good Requirements:

- User Stories
- User Acceptance Tests
- Workflow
- Requirements (Details)
- Wireframes

User Stories

What is a User Story?

- A User Story is a requirement expressed from the perspective of an end-user goal. User Stories may also be referred to as Epics, Themes or features but all follow the same format.

- A User Story is really just a well-expressed requirement. The User Story format has become the most popular way of expressing requirements in Agile for a number of reasons:
- It focuses on the viewpoint of a role who will use or be impacted by the solution
- It defines the requirement in language that has meaning for that role
- It helps to clarify the true reason for the requirement
- It helps to define high level requirements without necessarily going into low level detail too early
- User goals are identified and the business value of each requirement is immediately considered within the user story.

User Stories

User Stories are often deemed to comprise three elements - **the 3C's**

- **C**ard
- **C**onversation
- **C**onfirmation

User Acceptance Tests

- These should include all scenarios outlined in the user stories. These should not be too detailed (they don't need to mention specific screens or a complete list of actions to execute the steps). These should read:
- GIVEN that condition 1 and condition 2….
  WHEN I do step 1, and step 2…
  THEN, desired result 1, desired result 2….
- These define a set of actual scenarios a tester could walk through to assert that the feature is complete. These are not detailed test scripts that you find in UAT. They are meant to convey a set of tests that all involved can walk through to understand how the feature will work.

User Stories

- This states all of the scenarios of the users involved. These should read:
- As a SOME ROLE,
  I want to DO SOMETHING,
  So that I CAN GET SOME BENEFIT
- The user stories are critical to lay out exactly who is going to do what, and for what reason(s).

- The format of the User Story is as follows:

  As a < role>

  I need

  So that

  These two examples demonstrate User Stories at different levels, but using the same format:

- At a project level

  As a Marketing Director,

  I need to improve customer service

  So that we retain our customers.

- At a detailed level

  As an Investor,

  I need to see a summary of my investment accounts,

  So that I can decide where to focus my attention.

- User Stories provide another powerful message.
- Choosing User Stories to define requirements demonstrates an intention to work collaboratively with the users to discover what they really need.
- The User Story is brief and intended to be a placeholder for a more detailed discussion later – the Conversation.
- Much of the detail of User Stories emerges during Timeboxes as part of evolutionary development.
- High-level User Stories (Epics) are broken down by the Solution Development Team into more detailed User Stories just before development commences on that group of stories.
- Even then, the User Stories are not intended to be full specifications of the requirements.
- Fine detail may not need to be written down at all, but may simply be incorporated directly into the solution as part of the work within a Timebox.

  The user story format helps to ensure that each requirement is captured in a feature-oriented, value oriented way, rather than a solution oriented way.

User Story – the Card

- User Stories are often printed onto physical cards, for planning purposes and to help the Solution Development Team monitor progress.
  The Front of the Card

- On the front of the card, the following information is typically displayed:

- A unique “Story Identifier”, usually a number or reference
- A clear, explicit, short name or title
- “As a I need , so that ”; this section states:
  - who is the primary stakeholder (the role that derives business benefit from the story)
  - what effect the stakeholder wants the story to have
  - what business value the stakeholder will derive from this effect.

User Story – the Card

- **The Back of the Card**

- On the back, the Confirmation area contains:

- Acceptance criteria (the test criteria)
- These acceptance criteria define, at a high level, the test criteria which will confirm that this user story is working as required. These are not intended to be the full test scripts, but will be used to expand into the appropriate test scenarios and test scripts during Timeboxes, as necessary.

  For User Stories at the highest level (sometimes called a project Epic), the acceptance criteria may be used to define the aims of the project using criteria that may be measured after the project has completed (as part of the Benefits Assessment).

- Project acceptance criteria example:
- Is customer retention improved by 20% within two years?
- Is product range increased by 10% within 5 years?
- Has speed of dispatch improved to within 24 hours of time of order for 99% of in-stock items within 6 months?

User Story Example:

- **Story Identifier:** STK001

  Story Name: Customer Order

  Description: As a Customer, I need to place an order so that I can have food delivered to my house.

  Confirmation: Acceptance Criteria examples:

  Functional:

  - Can I save my order and come back to it later?

  - Can I change my order before I pay for it?

  - Can I see a running total of the cost of what I have chosen so far?

- Non-functional: availability:

  - Can I place an order at any time (24 hours per day or 24/7/365)?

  - Can I view the order at any time (24 hours per day or 24/7/365) up to and including delivery?
- Non-functional: security:

  - Are unauthorised persons and other customers prevented from viewing my order?

:

Well constructed User Stories

Bill Wake's INVEST model provides guidance on creating effective User Stories

Independent	Stories should be as independent as possible from other stories, to allow them to be moved around with minimal impact and potentially to be implemented independently. If stories are tightly dependent, consider combining them into a single user story.
Negotiable	Stories are not a contract. They are "placeholders" for features which the team will discuss and clarify near to the time of development.
Valuable	Stories should represent features providing clear business value to the user/owner of the solution and should be written in appropriate language. They should be features, not tasks.
Estimable	Stories need to be clear enough to estimate (for the appropriate timeframe), without being too detailed.
Small	Stories should be small enough to be estimated. Larger "Epic" stories should be broken down into smaller User Stories as the project progresses. The stories after splitting still follow the INVEST criteria.
Testable	Stories need to be worded clearly and specifically enough to be testable.

- A well-written user story is clear, concise and complete. Some simple checks are:
- It does not combine with, overlap nor conflict with other User Stories
- It conforms to organisational and project standards and policies where applicable
- It is traceable back to the business needs expressed in the business case and project objectives
- Where several User Stories relate to the same feature, but for different users, they are cross-referenced to each other

What is backlog grooming?

- Backlog grooming refers to the practice of refining the backlog by selecting the important work items, prioritizing them to the top of the backlog and cutting out the unimportant stories and tasks.
- This practice is important because it helps product owners keep the backlog stable, while making it easy to pick and choose which items to work on when the team is ready to start a sprint.

Who should groom the backlog?

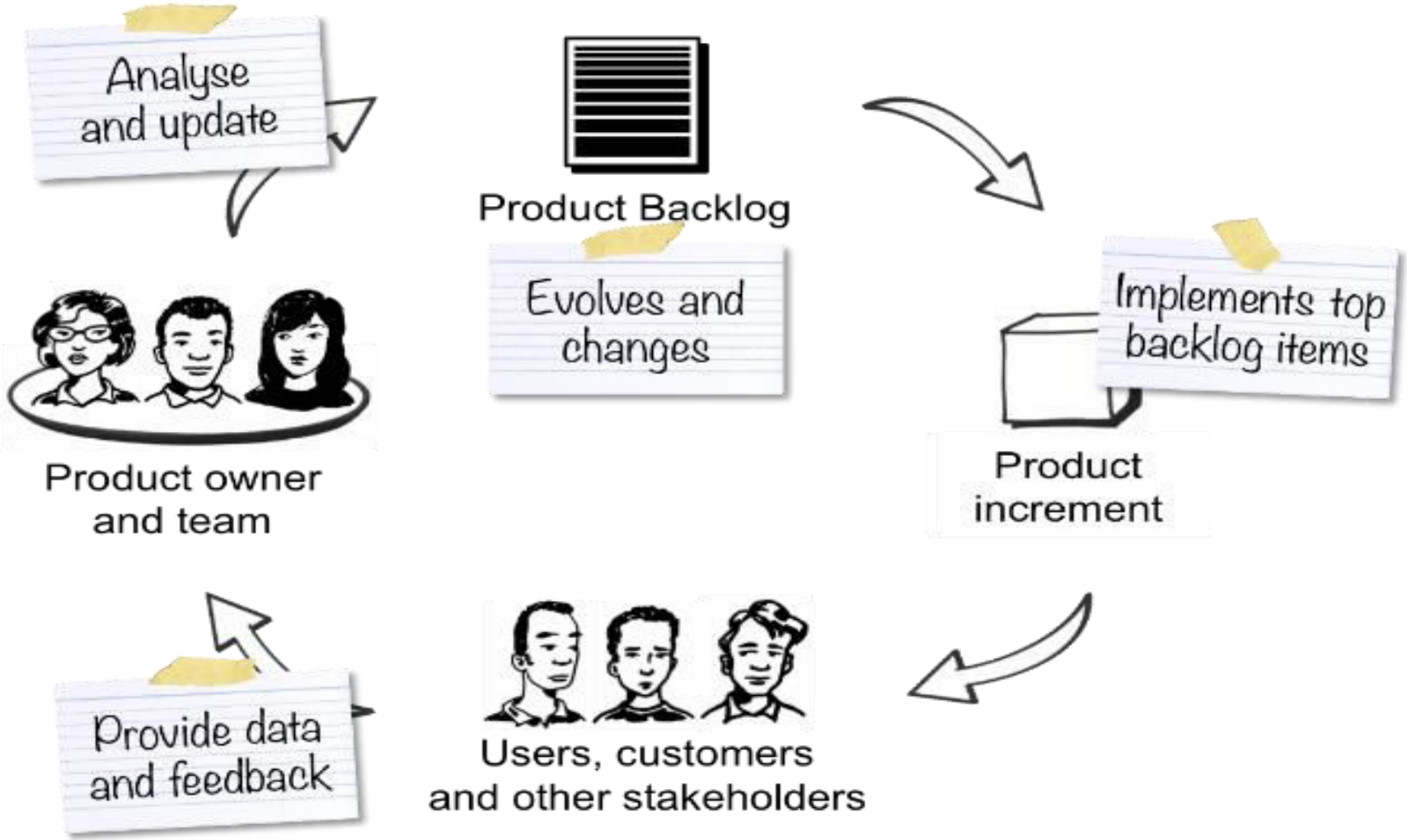
- While the sprint planning meeting is an official scrum meeting, there is no compulsion to have a separate meeting to groom the backlog.
- The backlog is usually groomed by the person responsible for owning and maintaining the backlog.
- The product owner is generally the best person to groom the backlog as he/she has the best understanding of the product that is being built.
- This makes sure that the right items are trimmed out and the most important ones are moved up the backlog.

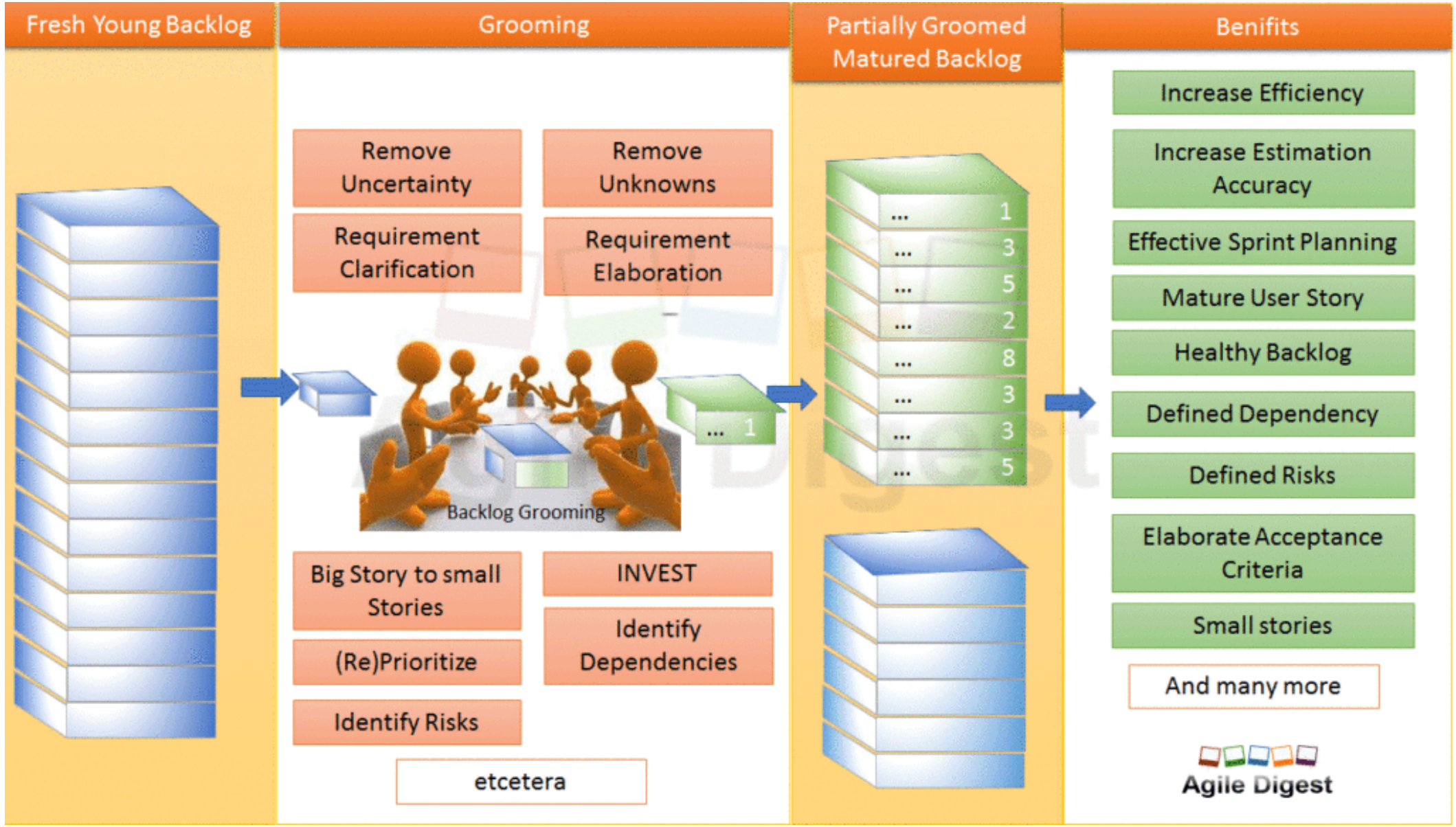
Who should groom the backlog?

- In addition to the product owner, an engineering leader or manager could be present for a backlog grooming meeting as they add a much needed technical perspective towards estimating and prioritizing items or cutting items from the backlog.
- In this type of grooming session, the product owner acts as a facilitator of the grooming meeting.

Back log Grooming

Importance of Backlog Grooming




Fresh Young Backlog
Grooming
Partially Groomed Matured Backlog
Benifits
Remove Uncertainty
Remove Unknowns
Requirement Clarification
Requirement Elaboration
Backlog Grooming
Big Story to small Stories
INVEST
(Re)Prioritize
Identify Dependencies
Identify Risks
etcetera
Increase Efficiency
Increase Estimation Accuracy
Effective Sprint Planning
Mature User Story
Healthy Backlog
Defined Dependency
Defined Risks
Elaborate Acceptance Criteria
Small stories
And many more
Agile Digest

The backlog grooming process

- In the course of the development process, a sprint backlog grows ever bigger. Backlog grooming keeps it organized. Before a sprint planning meeting, product owners should review the backlog to ensure:
- The set priorities are correct.
- The prioritized items have all the correct information.
- The feedback from previous meetings has been incorporated.

Sometimes, The backlog can quickly become overcrowded and out of control. To keep it relevant, grooming refines the backlog by cutting out unimportant stories and tasks and maintaining only priority ones.

Key backlog grooming activities include:

- Eliminating unwanted user stories that don’t fit the current product direction.
- Reprioritizing stories to move lower priority items to the bottom of the backlog.
- Breaking down large work items into smaller ones.
- Updating estimates.
- Adding new work items.

The benefits of frequent backlog grooming quickly become apparent as development teams can readily access all the information they need to put together a sprint action plan.

Benefits of backlog grooming

There are several important reasons to adopt backlog grooming:

- **Keeps the backlog clean**
- **Keeps the backlog relevant**
- **Keeps the whole product team up to date**
- **Increases work velocity**

Backlog grooming best practices

Make your product backlog DEEP

Roman Pichler, author of “Agile Product Management with Scrum,” used the acronym DEEP to sum up the essential traits of a good product backlog:

- **Detailed Appropriately** — User stories and other items in the product backlog that will be done soon need to be sufficiently well understood by cross-functional teams. Items and initiatives that will not be delivered for a while should be described with less detail.
- **Estimated** — Backlog items at the top should include an accurate estimation of the work needed to deliver them. Conversely, items down the backlog should only be roughly estimated as they are not that well understood yet.

Backlog grooming best practices

Make your product backlog DEEP

- **Emergent** — A product backlog changes over time. As you capture new user insights, the backlog will be changed to adapt to customer needs.
- **Prioritized** — The product backlog should be ordered with the most valuable items at the top and the least valuable at the bottom. Every single backlog item is ranked in relation to its business value and alignment with the strategic goal of the company.

Sprint planning

- The purpose of sprint planning is to agree on a goal for the next sprint and the set of backlog items (aka sprint backlog) to achieve it. This involves more than just communicating requirements.
- It's about **learning, considering options and making decisions together**.
- Sprint planning consists of two main components: (1) prioritizing backlog items and (2) agreeing on the number of backlog items in the sprint based on team capacity.

Sprint planning

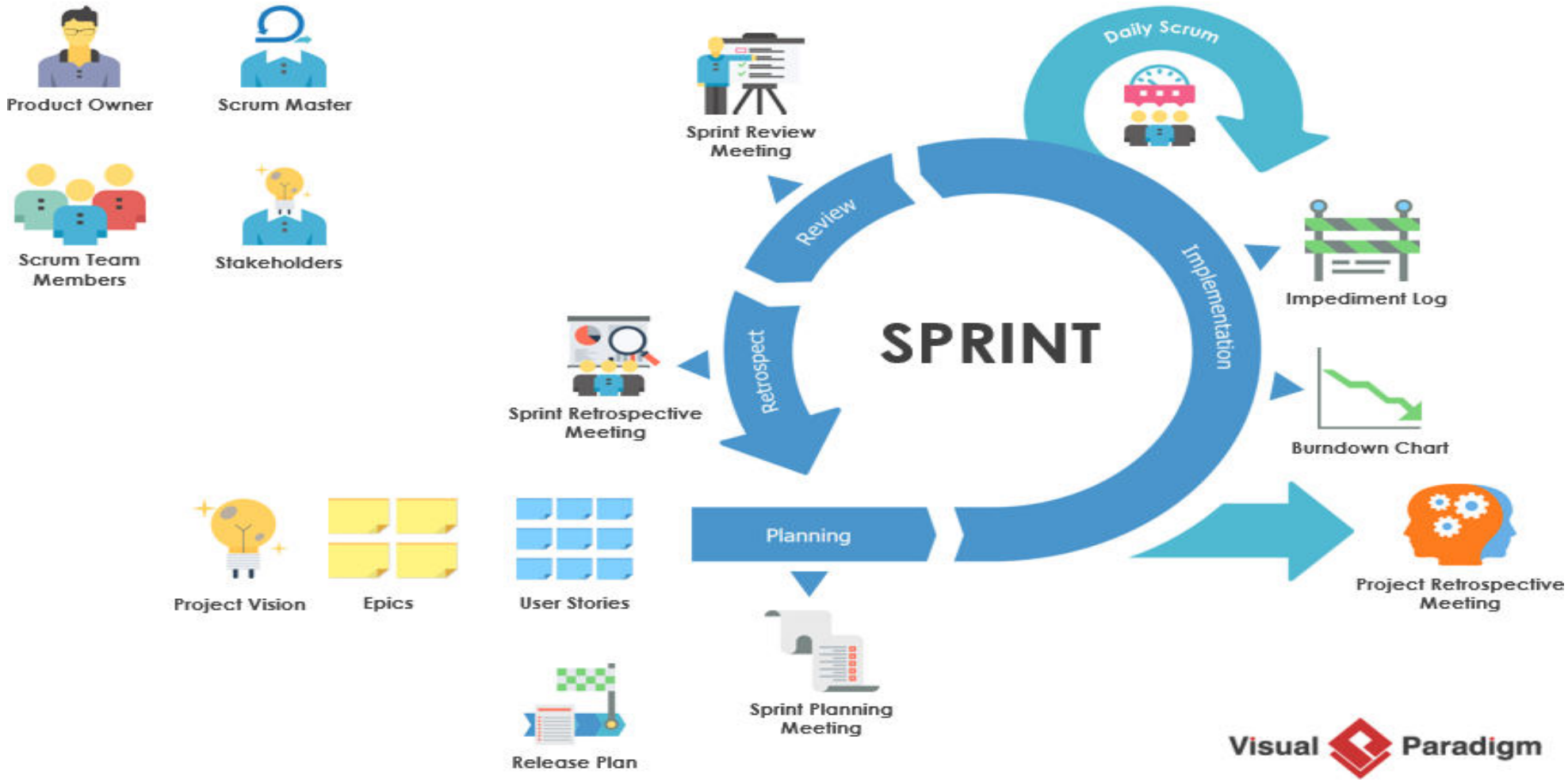
- Traditionally, (1) prioritizing backlog items is the responsibility of the product owner, whereas (2) deciding how much to pull in the sprint is the development team's decision. Both should be a discussion with the entire team, while the responsibilities (and the final say) stay with the respective roles.
- The input for a sprint planning session is the product backlog. In order to make the meeting as effective as possible, the top of the backlog — the most important backlog items that should be tackled next — should be well "groomed" or rather "refined" ahead of time.

Last Lecture

Scrum:

- Scrum process,
- Scrum roles - Product Owner, Scrum Master, Team, Project Manager, product manager,
- Scrum Events,
- Scrum artifacts

The Agile – Scrum Framework



Roles

- Product owner
- Development team
- Scrum master

Artifacts

- Increment
- Product backlog
- Sprint backlog

Ceremonies

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

SCRUM ROLES

PRODUCT OWNER

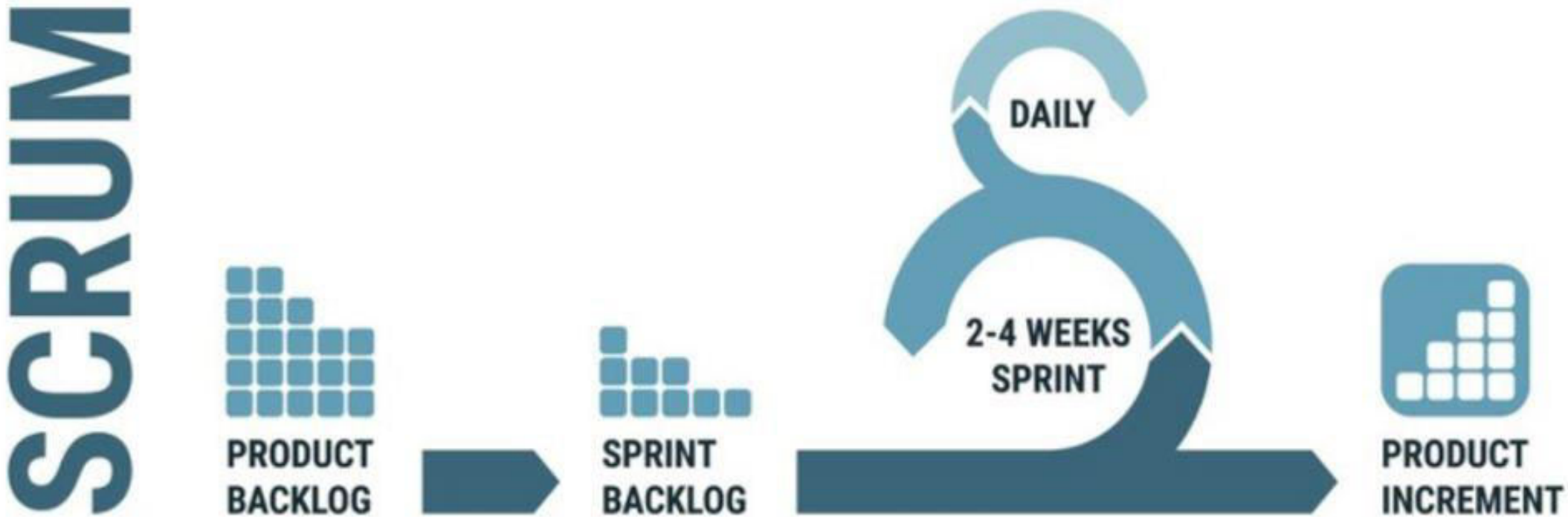
Represents the client and the business in general for the product on which they're working.

SCRUM MASTER

Responsible for ensuring the team has everything they need to deliver value.

DEVELOPMENT TEAM

A group of cross-functional team members all focused on the delivery of working software.



Artifacts in Scrum Framework - agile

EVENTS/CEREMONIES IN AGILE METHODOLOGY (SCRUM)

1

SPRINT PLANNING

To craft a Sprint Goal and a plan to achieve it

2

DAILY SCRUM

To review team's progress on Sprint Goal and adjust the plan to get there

3

SPRINT REVIEW

To review the work completed in the sprint with stakeholders

4

SPRINT RETROSPECTIVE

To review the product development process and identify areas for continuous improvements

SPRINT BACKLOG

1.REGISTER

2. LOG IN

Today’s Lecture

- **Product Inception:**
- Product vision,
- stakeholders,
- initial backlog creation

Product Inception

- Every product should begin with a problem that needs to be solved.
- Product Inception is the process by which we pin down the problem in concrete terms and determine how we are going to solve it—all before writing a single line of code.
- Representatives from the design and development team as well as stakeholders must be present throughout the process in order for Product Inception to be effective.
- By the end of a Product Inception session, everyone should be on the same page about where we're going, why we're going there, and how we are going to arrive at our destination.

Why do we believe in Product Inception?

- Product Inception is crucial to developing a product that truly solves the problem we intended to solve and helps users meet their goals.
- Even if a problem seems obvious, there are always multiple perspectives to consider; there may be aspects of a problem we can't see until a stakeholder brings it to our attention.
- In addition, everyone may have a different idea about what an effective solution looks like.

Why do we believe in Product Inception?

- Product Inception, then, brings everyone to the table so we can come up with a unified vision for the product and agree on a road map to take us there.
- The collaborative nature of Product Inception ensures that we include everyone's point of view, thus allowing us to produce ideas that are even better than what each of us could have thought of individually.

How It Works

- Product Inception involves a series of exercises that helps us define why we are building the product, how to best fulfill users' needs, what features the product will include, and which parts of the product are high priorities.

The different phases of Product Inception that we'll explore in this series include:

- Defining the main problem and solution with a **Product Vision Board**. This exercise enables us to articulate what exactly we want to achieve with this product.

How It Works

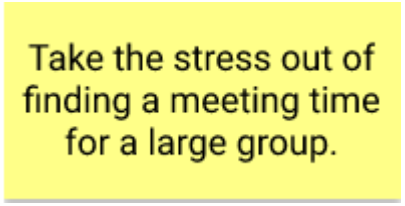
- Ensuring the product is relevant and identifying which product attributes to prioritize through **Stakeholder Analysis**.
- Understanding users' goals and related needs through **Persona Analysis**.
- Visualizing the functionality of the product, identifying necessary features, and planning the first release with a **Story Map**.

The Product Vision Board: The First Step to Discovering a Successful Software Product

- A concrete product vision is crucial for any software product. It tells you where you are going, why you are going there, and who you are going there for.
- A well-defined product vision is the foundation to a successful development process, while an unclear product vision will often leave your development team confused, making it difficult to prioritize user stories and features.
- In the end, without product vision, your users may not even understand what your product does.

What is Product Vision?

- The product vision is a short statement that encapsulates what you are trying to achieve with a product.
- It sounds simple, but in reality it can be quite challenging.
- Say for example, we want to create a calendar application for finding meeting times. That sounds great, but that's not a clear product vision.
- Product vision isn't actually about the product at all, it's about what the product will do and what problem it will solve. A much better product vision would be:



- This tells us exactly what we want to achieve, which will keep us focused on track throughout the development process.

Product Vision Board

The Product Vision Board

SOPHiLABS

Vision

What is your purpose for creating the product?

What positive change should it bring about?

Target Group

Which market or market segment does the product address?

Who are the target customers and users?

Needs

What problem does the product solve?

What benefit does it provide?

Product

What product is it?

What makes it stand out?

Is it feasible to develop the product?

Business Goals

How is the product going to benefit the company?

What are the business goals?

The Product Vision Board

SOPHiLABS

Vision

What is your purpose for creating the product?

What positive change should it bring about?

Take the stress out of finding a meeting time for a large group.

Target Group

Which market or market segment does the product address?

Who are the target customers and users?

Our Company

Employees who plan meetngs

Employees who attend meetngs

Needs

What problem does the product solve?

What benefit does it provide?

Easy to use

Simple

Enable employees to quickly find a meeting time that will work for all participants

Product

What product is it?

What makes it stand out?

Is it feasible to develop the product?

Web and mobile app that automatically generates time that everyone invited to a meeting is free.

Sync with major calendar applications

Clean and simple design, no frills

Easy to use, does work for you

Business Goals

How is the product going to benefit the company?

What are the business goals?

Decrease time spent planning meetings and emailing back and forth about schedules

Increase Productivity

Boost employee morale

Maximize the number of people in important meetings

Completed Product Vision Board, Product Vision Board example

Stakeholder Analysis: Another Key Piece of Product Inception

- Stakeholders are a vital part of any software project, which makes Stakeholder Analysis a critical part of the Product Inception phase.
- The only way to make sure that we are building a product that is relevant and valuable is to engage people and understand why they may be interested in the product and bring them into the process.
- If we don't involve stakeholders from the very beginning, we can easily build an excellent product with great features, but it's possible that no one will want or need to use it.

What is a stakeholder?

- A stakeholder is any person or group of people that have a specific interest in the product we are going to build.
- This will of course include end users and customers but can also include, for example, a leadership team, marketing department, investors, or any number of other people who have an interest or "stake" in the product and its developments.
- We've already begun the process of identifying stakeholders on the Product Vision Board, when we listed Target Groups who can also be considered stakeholders.

- Say for example, we want to develop a flower delivery app where users can log in and order flowers from a local florist to be delivered anywhere in the city. Our stakeholders might include:
- **Customers/End Users**: These are the people who will sign up and use your app to order flowers.
- **Flower Shops**: These are the people who you will need to convince to affiliate with your app and be vendors.
- **Marketing Staff**: They have to market the product and make it desirable for users
- **Investors**: They have a financial interest in the success of the product.

What is Stakeholder Analysis?

- Stakeholder Analysis is a process of understanding your stakeholders' background and context in order to pinpoint what they want and contrast that with what they actually need.
- Once we've identified our stakeholders, we want to ask questions like:
  - Who are the stakeholders?
  - Where do they come from?
  - What is their common background?
  - What are they looking for, as a group, from the project/product?
  - What is important to them?
  - Do they have any specific goals as a group?

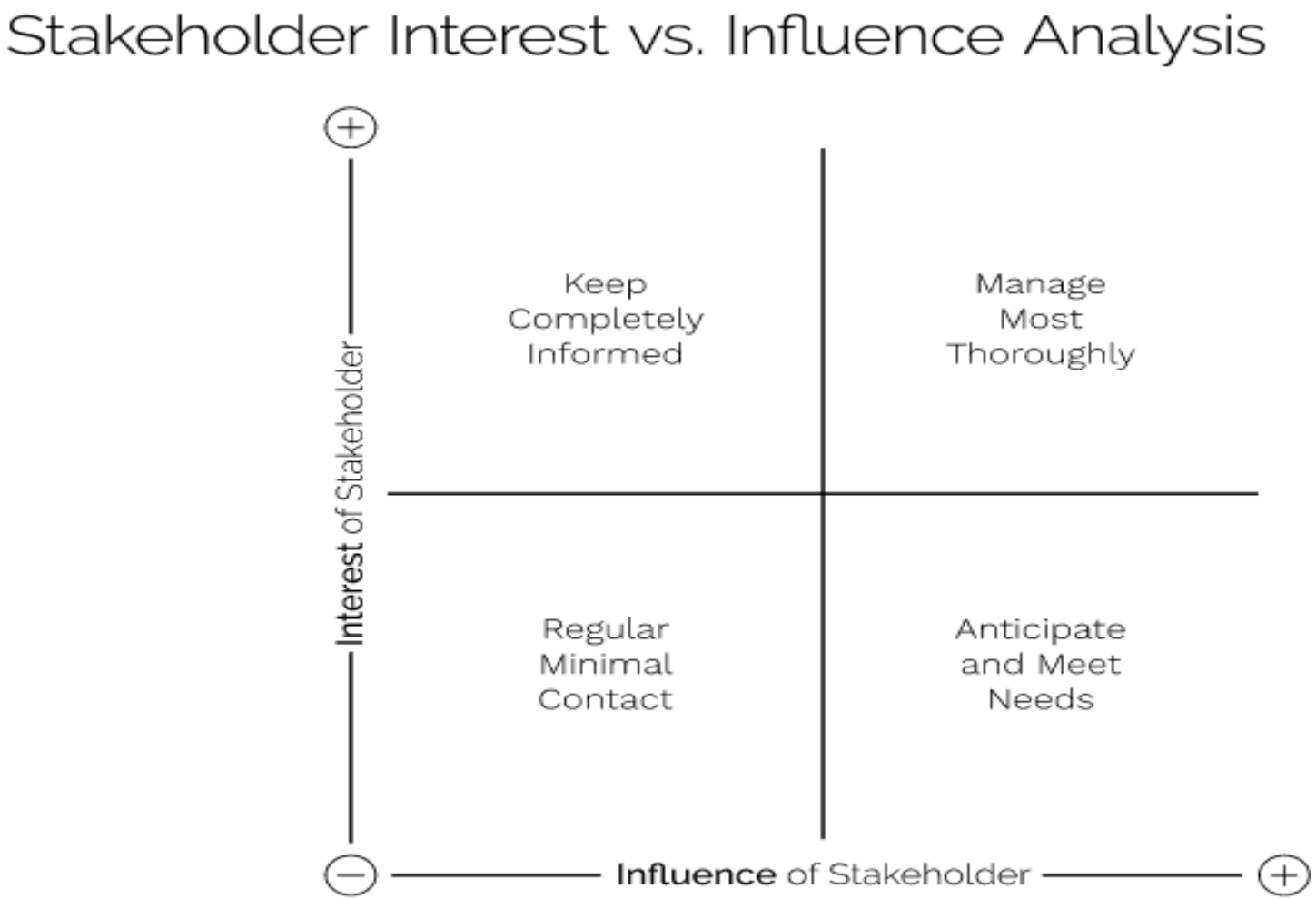
What is Stakeholder Analysis?

- During Stakeholder Analysis, we need to determine not only who our stakeholders are but also what is their relationship to the project as well as figure out which stakeholders' concerns we will need to prioritize throughout the project.

Stakeholder Analysis Techniques

- Product Vision Board
- During the process of creating a **product vision board**, we identified broad target groups, who are ultimately stakeholders
- Stakeholder Mapping: Influence vs. Interest Chart
- Every software development project will have multiple stakeholders and at some point, their needs are going to be in conflict.
- Stakeholder Mapping on an Influence vs. Interest Chart help us to determine each stakeholder's relationship to the product and how we will need to engage them throughout the process.

Stakeholder Mapping: Influence vs. Interest Chart



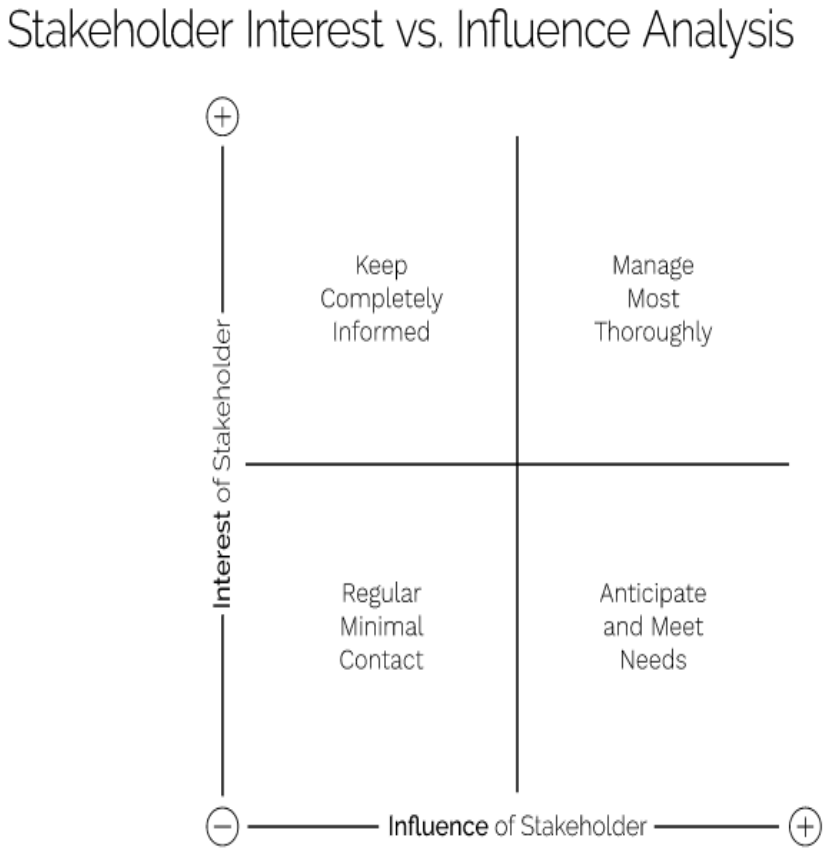
Flower App Delivery example revisited

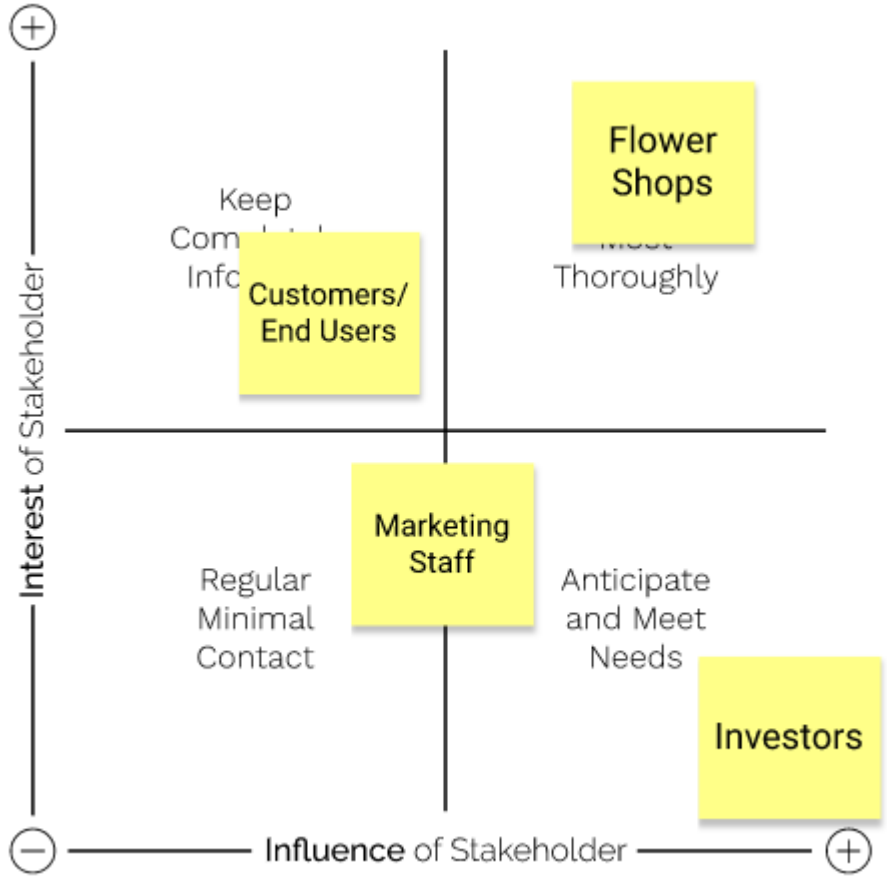
- The chart maps Influence on the x-axis and Interest on the y-axis.
- The more Influence a stakeholder has, the more necessary it will be to meet their needs.
- And the more Interest that they have, the more we will need to keep them informed about the product.
- When mapping the stakeholders, we need to keep the product at the center at all times.

Flower App Delivery example revisited

- Our investors will have a lot of influence over the product, because they are the ones financing it. However, their primary interest in this case is in the revenue received from a successful product, rather than the product itself. For **this reason, they are High Influence, Low Interest.**
- In contrast, our flower shops have a lot of influence as well, because their buy-in is necessary for the app's success. However, they also have a lot of interest in the product, as it is a tool for them to increase their sales and improve their business.
- This makes them a **relatively High Influence, High Interest stakeholder that we will have to manage thoroughly.**

In our example, we might map our stakeholders in this way:





Initial backlog creation

- Product Backlog is a dynamic prioritized source for all the work that needs to be done to deliver a product. This artifact is product focused not system focused.
- A product backlog consists of:
- Features
- Enhancements
- Defects
- Non IT Deliverables

- Example:
- As the mail room I must scan mortgage applications from the customer so that they can be processed
- As the system I must load scanned images so they are available to the indexing user
- As the indexer I want to be able to view a list of all applications requiring indexing
- What information do we capture in the product backlog?

- The details below provide a guideline to what information you may want to capture in a product backlog

Item	Description
Reference	Unique identifier for quick reference during collaboration
Description	A high level description of the item
Type	Feature, New Feature, Enhancement, Defect
Group	This can also be known as a feature set but is simply a grouping of functionality for example: Customer Management Contact Management
Relative Value	A factor that identifies the business value (Scale of 1 to 10, 10 been the highest business value)
Initial Estimate	This can be in story point or ideal days
Adjustment Factor	A summation of factors that may increase the initial estimate
Actual Estimate	Initial Estimate X (1 + Adjustment Factor)
Release	The release number that the PBI will be included in
Sprint (Iteration)	The sprint that the PBI has been allocated to
Notes	Additional comments, references or points of interest, be careful not to get into too much detail, some people prefer this to be separate
Status	Done or Not Done
Committed	Has this item been committed by the product owner i.e it has been decided that it will be incorporated into the product

Example

Ref	Description	Type	Group	Relative Value	Initial Estimate	Adjust Factor	Estimate	Release	Sprint	Status
01	As the mail room I must scan mortgage applications from the customer so that they can be processed	Feature	Scanning and Imaging	10	3	0	3	1	1	Not Done

Next Session

- **Agile Requirements** –
- User personas,
- story mapping,
- user stories,
- 3Cs, INVEST,
- Acceptance criteria, sprints, requirements,
- product backlog and backlog grooming;