UNIT – 5

❖ Model-driven Engineering

Model-driven software engineering provides an approach such that one can design complex software or system in the form of a model and use that model to generate a code for the system automatically. As the model is a preferred output of model-driven software engineering (MDSE) there is no need for engineers to be concerned with the actual specifics of the execution platform or which programming language has to be used, there is no need for programming language details. It allows engineers to think about software at a high level of abstraction, without being concerned about the implementation.

To overcome all these, Model-Driven Engineering (MDE) provides an approach that leads to platform-independent implementation and reduces the complexities and complications involved.

In MDE, the aim is to develop models rather than the program which will generate the code automatically. With the use of MDE level of abstraction is raised from program to model and the tiring process to write a code can be dropped.

Historical Overview

In the actual development process, abstraction plays an important role. From assembly language which was used to write machine code to a c programming language to object-oriented programming languages like java and python, there is a continuous rise in levels of abstraction. In the 80s prominent efforts began to raise abstraction levels and Computer-aided software engineering (CASE) was a result of it. It stressed on developing methods and tools that helped developers to represent their styles in terms of general-purpose graphical programming representations, including state machines, structure diagrams, and dataflow diagrams. But due to some reasons, it failed miserably. And now we are stepping forward to level up abstraction to the modeling. Lots of work has been done in this field.

Model-driven engineering in software

In simple language, MDE can be defined in two parts:

1. Defining the correct model for a problem statement that makes use of proper abstractions and provides an accurate solution so that people will be more

concerned about major key aspects of the problem statement rather than focusing on programming.

2. Providing an accurate solution as a code that will be generated from developed models automatically.

Models and abstraction levels

Models can be considered as a key in MDE as it plays an important role in developing complex systems. While creating of model one must be concerned with the abstraction levels.

Some of the abstraction types as described as follows:

- *Structure* — In this type of abstraction only structural components such as classes, subclasses, modules, interfaces which includes inputs and outputs, etc. are shown and all other details are hidden.
- *Behavior* — Main functionality of software to be developed i.e behavioral details are shown in this type.
- *Timing* — All the time-related constraints are addressed.
- *Resources* — Models may represent the environment of the software.
- *Metamodels* — Just like metadata which can be considered as data about the data, meta-models can be considered as models about the models. They describe how the model should be.

Generation of code

We can design and develop code generators in many different ways some of the important ones as explained below:

- *Templates and filtering*

This is one of the simplest ways of code generation. Specifications are given in the textual format. Initially, we have a source model which has different types of models, after filtering some of the specifications we get the subset of the source model, and code is embedded in the templates using a resultant subset of the source model. Thus the resultant code is generated.

- *Templates and metamodel*

This is an extension of templates and filtering. We don't directly apply patterns to the model instead of that we instantiate metamodel from the specification first. Templates are applied to this metamodel. Thus the resultant code is generated.

- *API based generators*

These generators provide an API against which code-generating programs are written. A target language or programming language is focused while developing the API.

- *Inline code generation*

In inline code generation, the final code is generated at the time of compilation of the non-generated program or using a precompiler.

Now as we know the basics of Model-driven engineering let's take a look at the pros and cons of MDE.

Advantages of Model-driven engineering

- *Higher Productivity* — As code is generated automatically by the developed model tiring process of programming is done autonomously by the model.
- *Abstraction* — Abstraction is raised to model level from program level with the help of Model-driven software engineering.
- *Consistency* — As code is auto-generated it's well maintained and consistent in nature.
- *Time* — More time can be given to the core problem statement and its key concepts as the time is reduced in coding.
- *Improved quality* — Handwritten code comes with variations and contrasts as work is done in a distributed manner. Hence these improvisations have to be made manually to make code more consistent but in the case of generators, incompetency is reduced significantly thus the overall quality of code is improved.

Disadvantages

- To build a model itself is a tedious and quite time-consuming task.
- Even after building a code generator, there are chances that it might not be sufficient enough and work in all the cases.
- In each system, there might be a type of code that the model cant generate and has to be generated manually.

❖ Aspect-oriented programming(AOP)

Aspect-oriented programming (AOP) is a programming paradigm designed to improve and increase modularity by enabling the separation of cross-cutting concerns. It makes it easier to add code to pre-existing programmes – by extracting code into manageable sections known as 'aspects' – without changing the code itself.

Cross-cutting concerns refer to aspects of a programme affecting other concerns that – in both design and implementation – cannot be easily, or cleanly, decomposed from a system. Cross-cutting concerns can cause code duplication (known as 'scattering') and critical dependencies between systems (known as 'tangling'), as concerns can be found across multiple classes and components. Common cross-cutting concerns include transaction processing, security authentication, data validation, caching, format data, error handling, debugging, and logging.

Here is a simplified walk-through of how AOP works in practice:

1. **Identify cross-cutting concerns** (for example, logging)
2. **Define aspects** (the modules that encapsulate the concerns)
3. **Weaving** (use an AspectJ compiler or weaver to combine aspects with main business logic at compile-time, load-time, or run-time)
4. **Application of aspects** (apply aspects to specific 'join points', where functionality is integrated into the programme)
5. **Isolation of concerns** (which enables modularization).

The AOP framework is supported by a number of popular programming languages and platforms, including Java (and Java Spring Framework/Spring AOP), C#, Python, .NET, Perl, XML, and Ruby.

Aspect-oriented programming different from object-oriented programming:

Both AOP and object-oriented programming (OOP) are programming paradigms.

The goal of OOP is to organise code into 'objects' (instances of classes) that encapsulate data and behaviour. It uses four principles to model real-world entities: encapsulation (binding data), abstraction (using simple classes to represent complexity), inheritance (enabling classes to inherit features of others), and polymorphism (using different objects to reply to one form, and interact with the same interface).

While AOP and OOP focus on different aspects of software development and design – the former on separation of concerns and the latter on data encapsulation and behaviour – they can be used to complement one another.

Advantages of AOP

AOP's ability to provide declarative enterprise services – such as declarative transaction management/annotations – for a specific software system or organisation has contributed to its popularity and widespread use. It also allows users to implement custom elements and add additional functionalities and features that were not initially present in the software.

In addition, aspect-oriented software development provides a number of other strategic advantages:

- **Modularity**. Improvements to code modularity make everything simpler to understand. AOP modularises and separates cross-cutting concerns from core business logic, allowing programmers and developers to handle concerns separately.
- **Maintainability**. AOP makes it possible to modify or update specific code functionalities – without affecting the wider source code – making software changes more manageable for programmers and developers. It also helps to reduce undesirable or unintended side effects.
- **Code reusability**. The aspects that are inherent in AOP encapsulate common functionalities and, therefore, promote the reuse of code across different parts of the application.
- **Centralised management**. AOP makes it easier to implement modifications and changes uniformly across the entire application and, as a result, manage cross-cutting concerns such as transactions and security.
- **Readability**. Isolating and specifying non-functional requirements improves the readability of central business logic. As such, software engineers can focus on core functionality, free from the distractions of unrelated concerns.
- **Scalability**. As the codebase grows, the effective management of concerns becomes more important, as well as more complex. AOP supports a cleaner, more organised code structure – which, in turn, supports scalability.
- **Better testing functionality**. Concerns that have been separated/isolated make it much easier to conduct independent testing. This is useful in terms of promoting better overall software quality and implementing a more effective testing methodology and strategy.

Disadvantages of AOP

While there are many benefits to using the AOP framework, it's not perfect. As a result, computer programmers, engineers and developers should take time to understand whether AOP aligns well with the specific characteristics and requirements of a particular software program or project.

Some of the disadvantages of AOP include:

- **Issues with debugging**. Debugging can become more of a challenge. Aspects applied at different points in the programme can affect the flow of control and increase the level of complexity, making it more difficult to identify issues.
- **Complexity**. Codebases can suffer from greater levels of complexity as more aspects interact with the central business logic. This creates challenges for developers who may not be as comfortable, or familiar, working with AOP frameworks. There are also knock-on effects for testing; additional work may be required to ensure that isolated aspects behave as intended.
- **Portability**. Portability between different platforms, applications and programming languages may cause issues. As such, AOP can limit the capacity to reuse code in diverse environments.
- **Support systems**. Integrated development environments (IDE) and tooling may be limited in range when it comes to AOP frameworks. In contrast to OOP, which benefits from a more extensive range of tooling support, AOP can encounter issues working with certain aspects in certain environments.

❖ Key Application Areas of AOP

AOP is used across multiple application areas. Let’s look at how different fields use AOP programming.

1. Web applications

AOP can be used in web applications to separate concerns such as logging, security, and transaction management. For example, an AOP logging aspect can capture method execution times and stack traces, while a security aspect can enforce authentication and authorization policies.

2. Enterprise applications

Aspect-oriented programming can be used in enterprise applications to manage exception handling, caching, and performance monitoring. For instance, an AOP exception handling aspect can catch and handle exceptions in a uniform and consistent manner across multiple components. On the other hand, a caching aspect can cache frequently accessed data to improve performance.

3. Mobile applications

AOP is used in mobile applications to manage device compatibility, data synchronization, and user engagement. The device compatibility aspect ensures the application works seamlessly across different platforms and devices. Meanwhile, a data synchronization aspect can handle data conflicts and ensure data consistency across multiple devices.

4. Embedded systems

Aspect-oriented programming could be used in embedded systems to manage concerns such as memory management, power consumption, and fault tolerance. For example, an AOP memory management aspect can optimize memory usage and prevent memory leaks. Meanwhile, a fault tolerance aspect can handle hardware failures and ensure the system continues operating reliably.

5. IoT

In IoT applications, AOP addresses concerns such as security, fault tolerance, and data processing. This implies that the AOP security aspect can enforce authentication and authorization policies to protect against cyber-attacks. Meanwhile, a fault tolerance aspect can handle errors and failures gracefully to ensure that the system continues to operate even in unpredictable and unreliable environments. A data processing aspect can handle complex data processing tasks such as aggregation, filtering, and transformation, making it easier to write efficient and maintainable code for IoT applications.

As per February 2023 research by IoT Analytics, the IoT market size is estimated to grow at a CAGR of 19.4% between 2022 and 2027 to reach $483 billion. As the IoT market continues to accelerate, AOP is expected to play a crucial role in managing IoT applications.

6. Finance sector

In financial applications, AOP can be deployed to address concerns such as transaction management, auditing, and compliance. For instance, an AOP auditing aspect can capture audit logs to ensure compliance with regulatory requirements. A compliance aspect can enforce compliance policies and prevent unauthorized access to sensitive financial data.

7. Healthcare industry

In healthcare applications, aspect-oriented programming can address concerns such as privacy, security, and interoperability. An AOP privacy aspect can ensure that sensitive patient data is protected and handled in compliance with privacy regulations, while a security aspect can protect against cyber threats and data breaches.

An interoperability aspect can ensure that healthcare systems can communicate and exchange data seamlessly, making it easier to share patient information across different healthcare providers and systems.

8. Gaming industry

In gaming applications, AOP can be used to address issues such as performance optimization, debugging, and game logic. For example, an AOP performance optimization aspect can optimize game rendering and animation, improving the overall gameplay experience for players. An AOP debugging aspect can capture detailed debugging information to help developers identify and resolve issues quickly and efficiently.

❖ Component-based software engineering (CBSE)

Component-based software engineering (CBSE) is a software engineering paradigm that emphasizes the design and construction of software systems using reusable components. This approach enables developers to assemble software systems from pre-existing components, which can significantly accelerate the development process while improving quality and reducing costs. Here are some key aspects of CBSE:

I. Concepts

1. **Components**:

- Components are self-contained units of functionality that can be independently developed, tested, and deployed. They typically encapsulate a specific business function and expose well-defined interfaces for interaction.
- Components can be software libraries, services (microservices), or even complete applications.

2. **Reusability**:

- One of the primary goals of CBSE is to promote reusability. Components are designed to be modular, allowing them to be reused across different applications and projects. This reduces duplication of effort and improves overall efficiency.

3. **Encapsulation**:

- Each component hides its internal implementation details and exposes only what is necessary through its interface. This promotes separation of concerns and allows for easier maintenance and updates.

4. **Interoperability**:

- Components can communicate with one another through defined interfaces and protocols, enabling interoperability across different systems. This is particularly important in heterogeneous environments where different technologies are in use.

5. **Component Composition**:

- CBSE focuses on composing applications from existing components. This involves defining how components interact and ensuring they work together to fulfill the desired functionality.

II. Benefits of CBSE

- **Accelerated Development**: By leveraging existing components, development time can be significantly reduced.
- **Increased Quality**: Frequently reused components are often better tested and more reliable, leading to higher quality software.
- **Easier Maintenance**: Since components are loosely coupled, changes to a component may not impact the rest of the system, making maintenance and updates simpler.
- **Cost Efficiency**: Shorter development cycles and reduced testing efforts can lead to cost savings.

III. Challenges of CBSE

- **Integration Complexity**: While components can be reused, integrating them into a cohesive system can be complex, especially if the components have different dependencies or compatibility issues.
- **Versioning**: Managing different versions of components can lead to conflicts and requires careful planning.

- **Dependency Management**: Ensuring that components work together requires careful tracking of dependencies and potential incompatibilities.

IV. Applications of CBSE

CBSE is widely used in various domains, including:

- Enterprise applications
- Web applications (using microservices architecture)
- Mobile applications
- System software

❖ service-oriented architecture (SOA)

Service-oriented architecture (SOA) is a method of software development that uses software components called services to create business applications. Each service provides a business capability, and services can also communicate with each other across platforms and languages. Developers use SOA to reuse services in different systems or combine several independent services to perform complex tasks.

For example, multiple business processes in an organization require the user authentication functionality. Instead of rewriting the authentication code for all business processes, you can create a single authentication service and reuse it for all applications. Similarly, almost all systems across a healthcare organization, such as patient management systems and electronic health record (EHR) systems, need to register patients. These systems can call a single, common service to perform the patient registration task.

I. Benefits of service-oriented architecture

Service-oriented architecture (SOA) has several benefits over the traditional monolithic architectures in which all processes run as a single unit. Some major benefits of SOA include the following:

Faster time to market

Developers reuse services across different business processes to save time and costs. They can assemble applications much faster with SOA than by writing code and performing integrations from scratch.

Efficient maintenance

It’s easier to create, update, and debug small services than large code blocks in monolithic applications. Modifying any service in SOA does not impact the overall functionality of the business process.

Greater adaptability

SOA is more adaptable to advances in technology. You can modernize your applications efficiently and cost effectively. For example, healthcare organizations can use the functionality of older electronic health record systems in newer cloud-based applications.

II. Basic principles of service-oriented architecture

There are no well-defined standard guidelines for implementing service-oriented architecture (SOA). However, some basic principles are common across all SOA implementations.

Interoperability

Each service in SOA includes description documents that specify the functionality of the service and the related terms and conditions. Any client system can run a service, regardless of the underlying platform or programming language. For instance, business processes can use services written in both C# and Python. Since there are no direct interactions, changes in one service do not affect other components using the service.

Loose coupling

Services in SOA should be loosely coupled, having as little dependency as possible on external resources such as data models or information systems. They should also be stateless without retaining any information from past sessions or transactions. This way, if you modify a service, it won’t significantly impact the client applications and other services using the service.

Abstraction

Clients or service users in SOA need not know the service's code logic or implementation details. To them, services should appear like a black box. Clients get the required information about what the service does and how to use it through service contracts and other service description documents.

III. components in service-oriented architecture

There are four main components in service-oriented architecture (SOA).

Service

Services are the basic building blocks of SOA. They can be private—available only to internal users of an organization—or public—accessible over the internet to all. Individually, each service has three main features.

Service implementation
The service implementation is the code that builds the logic for performing the specific service function, such as user authentication or bill calculation.

Service contract
The service contract defines the nature of the service and its associated terms and conditions, such as the prerequisites for using the service, service cost, and quality of service provided.

Service interface
In SOA, other services or systems communicate with a service through its service interface. The interface defines how you can invoke the service to perform activities or exchange data. It reduces dependencies between services and the service requester. For example, even users with little or no understanding of the underlying code logic can use a service through its interface.

Service provider

The service provider creates, maintains, and provides one or more services that others can use. Organizations can create their own services or purchase them from third-party service vendors.

Service consumer

The service consumer requests the service provider to run a specific service. It can be an entire system, application, or other service. The service contract specifies the rules that the service provider and consumer must follow when interacting with each other. Service providers and consumers can belong to different departments, organizations, and even industries.

Service registry

A service registry, or service repository, is a network-accessible directory of available services. It stores service description documents from service providers. The description documents contain information about the service and how to communicate with it. Service consumers can easily discover the services they need by using the service registry.

❖ Agile software development

Agile software development refers to software development methodologies centered around the idea of iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. The ultimate value in Agile development is that it enables teams to deliver value faster, with greater quality and predictability, and greater aptitude to respond to change. Scrum and Kanban are two of the most widely used Agile methodologies. Below are the most frequently asked questions around Agile and Scrum, answered by our experts.

DELIVER INCREMENTALLY
INSTEAD OF ALL AT ONCE

Agile software

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals. Agile development refers to any development process that is aligned with the concepts of the Agile Manifesto. The Manifesto was developed by a group of fourteen leading figures in the software industry, and reflects their experience of what approaches do and do not work for software development.

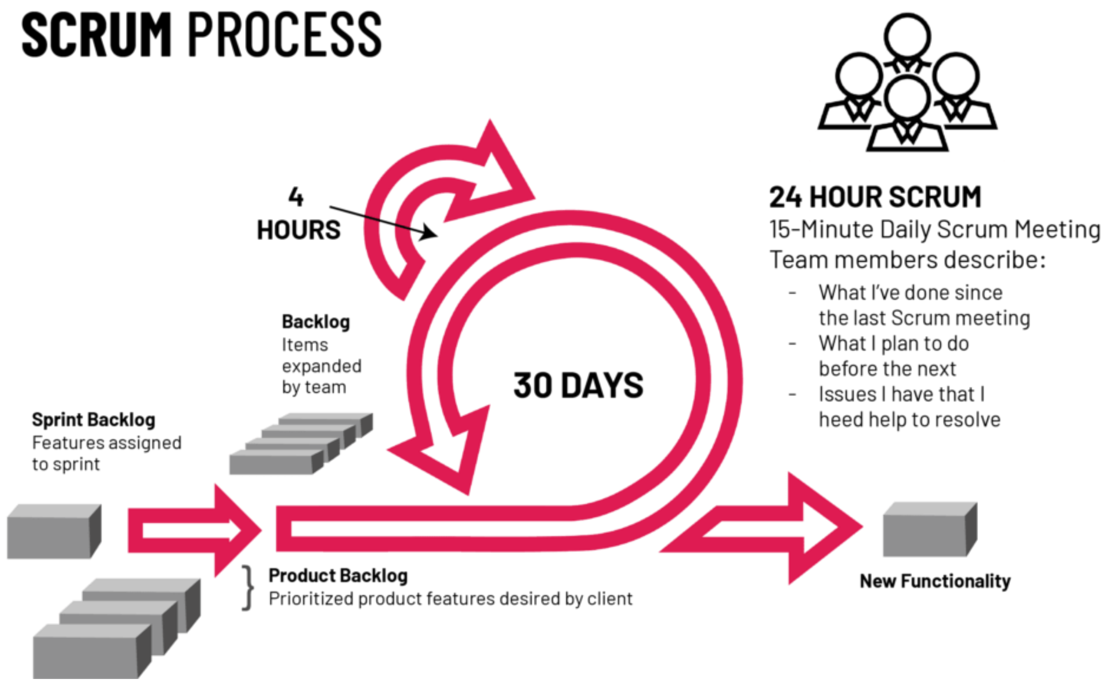
Scrum

Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one.

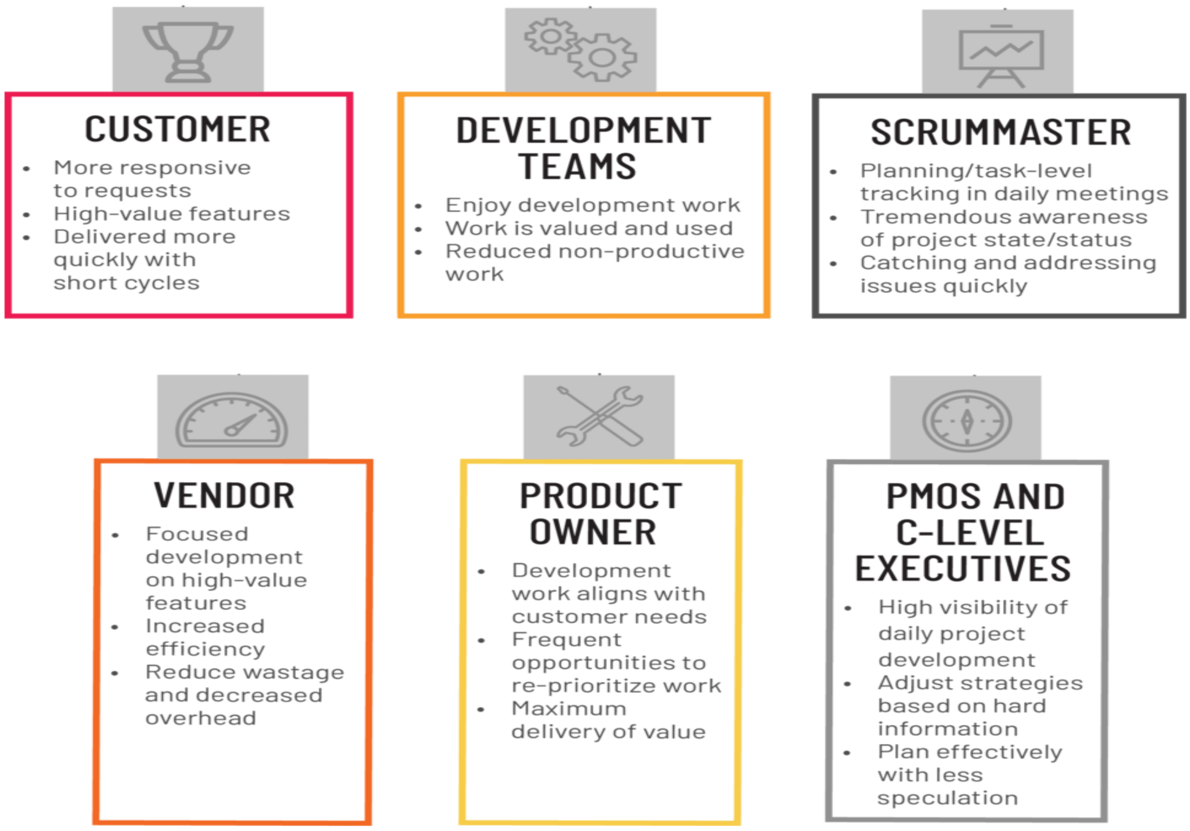
- A "process framework" is a particular set of practices that must be followed in order for a process to be consistent with the framework. (For example, the Scrum process framework requires the use of development cycles called Sprints, the XP framework requires pair programming, and so forth.)
- "Lightweight" means that the overhead of the process is kept as small as possible, to maximize the amount of productive time available for getting useful work done.

A Scrum process is distinguished from other agile processes by specific concepts and practices, divided into the three categories of Roles, Artifacts, and Time Boxes. These and other terms used in Scrum are defined below. Scrum is most often used to manage complex software and product development, using iterative and incremental practices. Scrum significantly increases productivity and reduces time to benefits relative to classic "waterfall" processes. Scrum processes enable organizations to adjust smoothly to rapidly-changing requirements, and produce a product that meets evolving business goals. An agile Scrum process benefits the organization by helping it to

- Increase the quality of the deliverables
- Cope better with change (and expect the changes)
- Provide better estimates while spending less time creating them
- Be more in control of the project schedule and state


SCRUM PROCESS
4 HOURS
24 HOUR SCRUM
15-Minute Daily Scrum Meeting
Team members describe:
- What I've done since the last Scrum meeting
- What I plan to do before the next
- Issues I have that I heed help to resolve
Backlog Items expanded by team
30 DAYS
Sprint Backlog Features assigned to sprint
Product Backlog Prioritized product features desired by client
New Functionality

Benefits of Agile



CUSTOMER

- More responsive to requests
- High-value features
- Delivered more quickly with short cycles

DEVELOPMENT TEAMS

- Enjoy development work
- Work is valued and used
- Reduced non-productive work

SCRUMMASTER

- Planning/task-level tracking in daily meetings
- Tremendous awareness of project state/status
- Catching and addressing issues quickly

VENDOR

- Focused development on high-value features
- Increased efficiency
- Reduce wastage and decreased overhead

PRODUCT OWNER

- Development work aligns with customer needs
- Frequent opportunities to re-prioritize work
- Maximum delivery of value

PMOS AND C-LEVEL EXECUTIVES

- High visibility of daily project development
- Adjust strategies based on hard information
- Plan effectively with less speculation

- Benefits to Customer

Customers find that the vendor is more responsive to development requests. High-value features are developed and delivered more quickly with short cycles, than with the longer cycles favored by classic "waterfall" processes.

- Benefits to Vendors

Vendors reduce wastage by focusing development effort on high-value features, and reduce time-to-market relative to waterfall processes due to decreased overhead and increased efficiency. Improved customer satisfaction translates to better customer retention and more positive customer references.

- Benefits to Development Teams

Team members enjoy development work, and like to see their work used and valued. Scrum benefits Team members by reducing non-productive work (e.g., writing specifications or other artifacts that no one uses), and giving them more time to do the work they enjoy. Team members also know their work is valued, because requirements are chosen to maximize value to customers.

- Benefits to Product Managers

Product Managers, who typically fill the Product Owner role, are responsible for making customers happy by ensuring that development work is aligned with customer needs. Scrum makes this alignment easier by providing frequent opportunities to re-prioritize work, to ensure maximum delivery of value.

- Benefits to Project Managers

Project Managers (and others) who fill the ScrumMaster role find that planning and tracking are easier and more concrete, compared to waterfall processes. The focus on task-level tracking, the use of Burndown Charts to display daily progress, and the Daily Scrum meetings, all together give the Project Manager tremendous awareness about the state of the project at all times. This awareness is key to monitoring the project, and to catching and addressing issues quickly.

- Benefits to PMOs and C-Level Executives

Scrum provides high visibility into the state of a development project, on a daily basis. External stakeholders, such as C-Level executives and personnel in the Project Management Office, can use this visibility to plan more effectively, and adjust their strategies based on more hard information and less speculation