CSC 540 2.0
Software Engineering

M. K. A. Ariyaratne, PhD.

Department of Computer Science
Faculty of Applied Sciences
University of Sri Jayewardenepura
Sri Lanka

What is Software Engineering?

- Software engineering is the application of a systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software.
- It is both an engineering discipline and an art.
- **Goal:** Develop high-quality software efficiently and effectively.

Key Characteristics of Software:

- Reliability
- Maintainability
- Scalability
- Security

Differences Between Software Engineering and Computer Science (practical applications vs. theoretical knowledge)

Aspect	Software Engineering	Computer Science
Objective	Build reliable, maintainable, and usable software.	Develop new computational theories and methods.
Nature	Practical and application-oriented.	Theoretical and research-oriented.
Scope	Focuses on real-world applications of software in various domains.	Encompasses the study of hardware, software, and theory.
Skill Set	Requires skills in project management, system design, and team collaboration.	Requires strong mathematical and analytical skills.
Key Deliverables	Functional software systems.	New algorithms, computational models, or theories.
Examples	Developing an e-commerce website, creating banking software.	Designing a new sorting algorithm, studying computational models.

Table: Comparison between Software Engineering and Computer Science

Why is Software Engineering Important?

- Software is critical in almost every field: healthcare, finance, transportation, education, etc.
- Poor software design can lead to catastrophic failures:
  - Mars Climate Orbiter disaster (1999) - Unit conversion error.(`https://www.youtube.com/watch?v=u4r0yrF_Wa0`)
  - Ariane 5 rocket explosion (1996) - Software bug in guidance system (`https://www.youtube.com/watch?v=N6PWATvLQCY`).
- **Need:** A disciplined approach to building reliable and robust software.

Mars Climate Orbiter disaster (1999)

- Mission: Mars Climate Orbiter was a NASA mission designed to study the Martian climate, atmosphere, and surface changes over time.
- Launch Date: December 11, 1998.
- Failure Date: September 23, 1999.
- Cost: Approximately $327.6 million.
- Cause of Failure: A software error stemming from a unit conversion mismatch.

The mission failed because the spacecraft entered Mars' atmosphere at an excessively low altitude, causing it to either burn up in the atmosphere or crash into the planet. The issue was traced to a unit conversion error between two systems:

1. The navigation software used imperial units to calculate force.
2. The spacecraft expected metric units.

This discrepancy caused the spacecraft to deviate from its intended trajectory, leading to its destruction.

Mars Climate Orbiter disaster (1999)

- The disaster became a landmark case for the importance of software quality assurance in space missions.
- NASA implemented stricter protocols for unit consistency and interdisciplinary testing for future missions.

A Software Engineering Perspective

This failure highlights the importance of:

1. Requirements Engineering: Clearly defining and validating system requirements.
2. Risk Management: Identifying and mitigating potential integration risks.
3. Testing: Performing rigorous integration and system-level testing.
4. Standards: Adopting universal engineering standards (e.g., SI units).

Ariane 5 rocket explosion (1996)

- Mission: The Ariane 5 rocket was a European Space Agency (ESA) project designed to launch heavy payloads into orbit.
- Date of Failure: June 4, 1996.
- Cost: Approximately $370 million USD.
- Cause of Failure: Software error resulting in the destruction of the rocket 37 seconds after liftoff.

Reused Software:

The software for the inertial reference system (IRS) was carried over from the Ariane 4 rocket, which had a slower trajectory and different flight dynamics.
The software was not revalidated for Ariane 5's faster acceleration.

Ariane 5 rocket explosion (1996)

A Software Engineering Perspective

- Adapt Software for New Systems:
  Software reused from older systems must be revalidated for new environments and scenarios.
- Error Handling:
  Critical systems should include robust error handling to prevent cascading failures.
- Simulation and Testing:
  Comprehensive testing under real-world conditions is essential to identify unexpected behaviors.

Cost of software bugs

- Therac-25 Radiation Therapy Machine (1985–1987)
  - Cost: Loss of lives and lawsuits.
  - Cause: A software bug in the machine caused massive overdoses of radiation, leading to patient deaths.
  - Impact: Emphasized the critical need for safety in medical software systems.
- Windows Vista Development (2007)
  - Cost: Estimated $6 billion in development and lost revenue.
  - Cause: Poor project management and delayed bug fixes led to a flawed product release.
  - Impact: Damaged Microsoft's reputation and resulted in market share losses.

Cost of software bugs

- Toyota Prius Software Glitch (2005)
  - Cost: Over $3 billion in recalls.
  - Cause: A software bug in the anti-lock braking system (ABS) caused brakes to malfunction.
  - Impact: Harmed consumer trust and highlighted the importance of quality assurance in embedded systems.
- Samsung Galaxy Note 7 Recall (2016)
  - Cost: $5.3 billion.
  - Cause: A combination of hardware and software issues led to battery overheating and explosions.
  - Impact: Damaged Samsung's reputation and underscored the need for rigorous software-hardware integration testing.

Cost of software bugs

These examples underline the necessity of robust software development practices, including:

- Thorough Testing: Unit, integration, and system-level testing.
- Rigorous Validation: Especially in safety-critical and financial systems.
- Effective Project Management: To minimize delays and ensure quality.
- Continuous Monitoring and Maintenance: To address bugs post-deployment promptly.

Key Definitions

- **Software:** Programs, documentation, and configuration required for a system to function.
- **Software Development:** The process of creating and maintaining software applications.
- **Software Engineering:** Engineering discipline that uses principles, methods, and tools to produce software.

Software Development vs Software Engineering

- **Software Development:**
  - Focuses on writing code to meet specific requirements.
  - Primarily a programming activity.
- **Software Engineering:**
  - Encompasses a broader process: requirements, design, testing, maintenance, etc.
  - A systematic and disciplined approach.

Software Engineering Ethics

Responsibilities of Software Engineers:

- Build software that meets user needs.
- Ensure software is reliable and secure.
- Protect user data and privacy.
- Avoid harm caused by software errors.

Key Ethical Guidelines:

- ACM Code of Ethics (link)
- IEEE Software Engineering Code of Ethics

Characteristics of Good Software

Good software must satisfy the following characteristics:

- **Reliability:** Performs its intended functions without failure.
- **Efficiency:** Utilizes resources optimally.
- **Maintainability:** Easy to modify and update.
- **Usability:** Simple and intuitive to use.
- **Portability:** Runs on different platforms without modification.

Types of Software

- Embedded Systems:
  Examples: Smart appliances, automotive software.
- System Software:Software that manages hardware and provides a platform for application software.
  Examples: Operating systems, database management systems.
- Application Software:Software designed to perform specific tasks for the user.
  Examples: Productivity tools, games.
- Real-Time Systems:
  Examples: Airline reservation systems, financial trading platforms.

The Role of a Software Engineer

- Technical Skills:
  Programming, testing, and debugging.
- Soft Skills:
  Teamwork, communication, problem-solving.
- Career Paths in SE:
  Developer, Tester, Project Manager, DevOps Engineer, etc.

Future Trends in Software Engineering

- **Low-Code/No-Code Platforms**:
  allow users to create applications with minimal or no programming knowledge. They provide drag-and-drop interfaces, pre-built templates, and automation tools.
- Examples:
  Low-code: Mendix, OutSystems, Microsoft PowerApps. No-code: Wix, Bubble, Zapier.

Future Trends in Software Engineering

- AI tools, such as GitHub Copilot, Codex, GPTs, and TabNine, assist developers by generating code, offering suggestions, and automating repetitive tasks.
- Examples:
  GitHub Copilot: Suggests code snippets and even complete functions based on context.
  DeepCode: Reviews code and suggests improvements for performance and security.

Key Takeaways

- Software engineering applies engineering principles to software development.
- High-quality software requires a systematic approach, covering design, testing, and maintenance.
- Ethics and responsibility play a critical role in the profession.
- Understanding the principles of software engineering ensures reliability, maintainability, and scalability.
- To stay relevant, software engineers must embrace lifelong learning to adapt to new technologies.

Software Development Life Cycle (SDLC) Models

- SDLC stands for Software Development Life Cycle.
- It is a structured process used to design, develop, test, and deploy software.
- Ensures high-quality software is produced efficiently.

Phases of SDLC:

1. Requirements Gathering and Analysis
2. Design
3. Implementation
4. Testing
5. Deployment
6. Maintenance

What are Software Process Models?

- Software process models are strategies or methods used to implement the SDLC.
- They provide the "how"—the approach or workflow to carry out the SDLC phases.
- Examples include:
  1. Waterfall: Linear and sequential.
  2. Incremental: Small functional increments.
  3. Iterative: Cycles of improvement.
  4. Agile: Iterative with customer collaboration.
  5. Spiral: Risk-driven, iterative with prototypes.

How does SDLC relates with Process models

SDLC (Concept)	Process Models (Implementation)
Describes the phases of software development	Defines the workflow for how the phases are executed.
General framework (what to do).	Specific methodology (how to do it).
Applies to all types of software projects.	Tailored to project needs, team expertise, and goals.
Example: Testing is a phase in SDLC.	Example: In Agile, testing is integrated into each sprint; in Waterfall, it is a separate phase.

In summary, the SDLC provides the "what" (phases) of software development, while process models provide the "how" (workflow) to execute these phases. Together, they ensure a structured approach to creating high-quality software.

SDLC Process Categories

Methodologies or philosophies used to manage and execute the Software Development Life Cycle (SDLC).

1. Plan Driven
   - Plan-driven development is a traditional, structured, and methodical approach to software development.
   - All of the process activities are planned in advance and progress is measured against this plan.
2. Agile Development
   - Agile development is a flexible, adaptive, and iterative approach to software development.
   - planning is incremental and it is easier to change the process to reflect changing customer requirements.

Need a balance between plan-driven and agile processes.

Plan-Driven vs. Agile Development

Aspect	Plan-Driven Development	Agile Development
Planning	Detailed upfront planning for the whole project.	Minimal upfront planning; plans evolve iteratively.
Requirements	Fixed requirements upfront.	Flexible requirements, adapt as needed.
Documentation	Heavy documentation for all phases.	Lightweight documentation, just enough to support development.
Process	Rigid, sequential processes.	Iterative, incremental processes.
Customer Involvement	Limited to initial phases or milestones.	High involvement throughout the project.
Team Collaboration	Teams often work in silos.	Cross-functional teams work collaboratively.
Delivery	Delivered only at the end of the project.	Frequent deliveries of working software.
Adaptability	Difficult to adapt to chang	Highly adaptable to change.

Software Process Models

1. The waterfall Model
2. Prototyping
3. Incremental Development
4. Iterative Development
5. Agile
6. Reuse oriented software engineering

Waterfall Model

Description:

- A linear and sequential approach.
- Each phase must be completed before moving to the next.

Phases:

1. Requirements Gathering
2. System Design
3. Implementation
4. Testing
5. Deployment
6. Maintenance

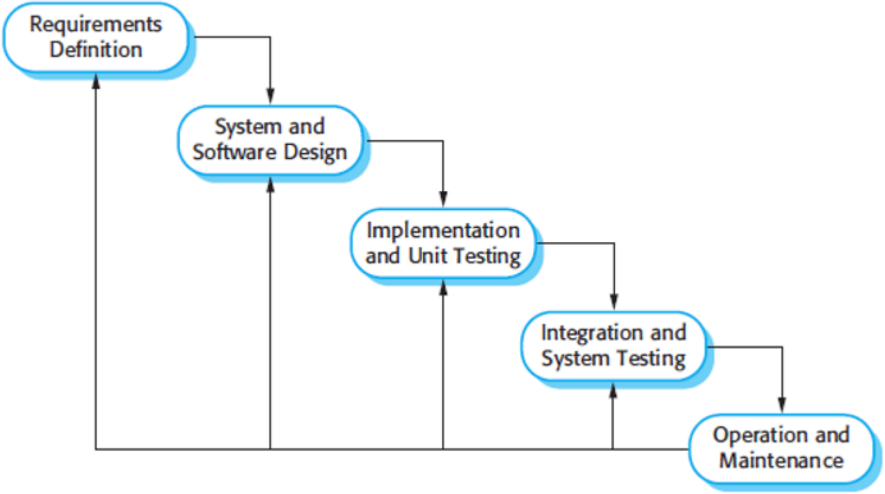
Advantages:

- Simple and easy to understand.
- Works well for small, well-defined projects.

Disadvantages:

- Inflexible to changes in requirements.
- High risk if errors are found late in the process.

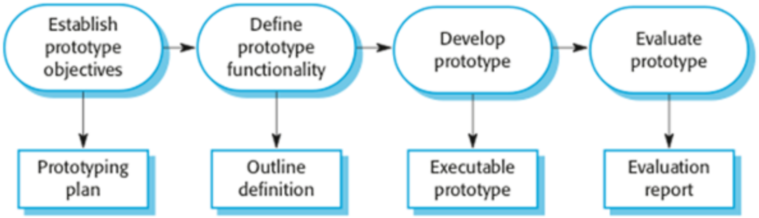
Waterfall Model


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

Prototyping

- A prototype is a version of a system or part of the system that's developed quickly to check the customer's requirements or feasibility of some design decisions.
- A prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc.
- The client is involved till the end, which increases the chance of accepting the final implementation.
- Some prototypes are developed with the expectation that they will be discarded, In some cases prototype can be evolved to the working system.

Prototyping



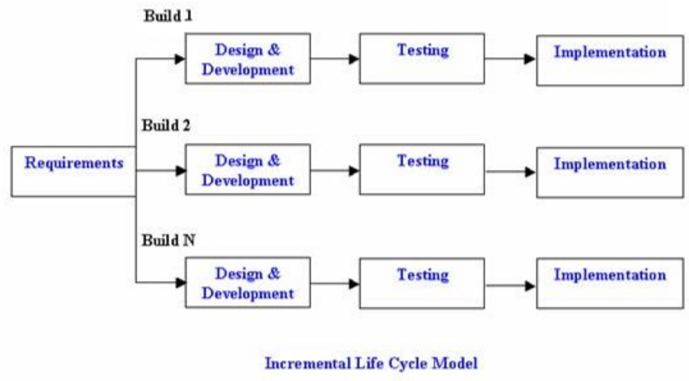
is not a standalone, complete development methodology, but an approach to be used in the context of a full methodology (such as incremental etc).

downsides are, good tools need to be acquired for quick development (like coding) in order to complete a prototype. In addition, the costs for training the development team on prototyping may be high.

Incremental development

- Product is designed, implemented and tested incrementally (a little more is added each time) until it is finished.
- Each increment builds a complete feature of the software.
- Multiple development cycles take place here, a “multi-waterfall” cycle.
- This approach can be either a plan-driven or agile,or both.
- In a plan-driven approach, the system increments are identified in advance, but, in the agile approach, only the early increments are identified and the development of later increments depends on the progress and customer priorities.

Incremental development



Incremental Life Cycle Model

System functionality is sliced into increments, in each increment, a slice of functionality is delivered.

Incremental development

Advantages

- Generates working software quickly and early during the software life cycle.
- Reduces cost of accommodating changing customer requirements.
- Easier to get customer feedback.
- Early delivery & deployment of software is possible.

Disadvantages

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

Iterative Model

Description:

- Develops the system in small, manageable iterations.
- Aims to develop a system through **building small portions of all the features**, across all components.
- Each iteration includes planning, design, implementation, and testing.

Advantages:

- Allows early feedback.
- Easier to identify and fix issues.

Disadvantages:

- Requires thorough planning.
- Can be resource-intensive.

Agile Development

Description:

- A flexible and adaptive approach to software development.
- Emphasizes collaboration, customer feedback, and small, rapid iterations.

Principles:

- Responding to change over following a plan.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.

Popular Frameworks:

- Scrum
- Kanban
- Extreme programming

Principles of Agile Methods

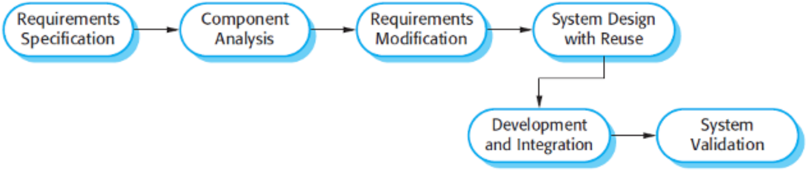
Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer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

Increment Vs Iterative Vs Agile

- What is the difference between incremental, iterative and agile models.
- Each increment in the incremental approach builds a complete feature of the software, while in iterative, it builds small portions of all the features.
- An agile approach combines the incremental and iterative approach by building **a small portion of each feature, one by one,** and then both gradually adding features and increasing their completeness.

Reuse-oriented development

- Attempting to reuse an existing design or code (probably also tested) that is similar to what's required. It's then modified, and incorporated to the new system.


Requirements Specification
Component Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

Reuse-oriented development

Advantages :

- It can reduce total cost of software development.
- The risk factor is very low.
- It can save lots of time and effort.
- It is very efficient in nature.

Disadvantages :

- Reuse-oriented model is not always worked as a practice in its true form.
- Sometimes using old system component, that is not compatible with new version of component, this may lead to an impact on system evolution.

Comparison of SDLC Models

Feature	Waterfall	Iterative	Agile	Reuse-Oriented
Flexibility	Low	Medium	High	Medium
Risk Management	High	Medium	Low	Medium
Customer Involvement	Low	Medium	High	Low
Delivery Time	Long	Medium	Short	Medium
Suitable for Complex Projects	No	Yes	Yes	Yes
Reuse of Components	No	Minimal	Minimal	High
Development Speed	Slow	Moderate	Fast	Moderate
Documentation Required	High	Medium	Low	Medium
Adaptability to Changes	Low	Medium	High	Low
Cost Efficiency	Low	Medium	High	High (when reusable components are available)

Types of Software

- Software can be broadly classified into:
    1. **Generic Software Products:**
        - Pre-built software designed for a broad market.
        - Examples: Microsoft Office, Adobe Photoshop.
    2. **Bespoke Software Products:**
        - Custom-built software developed for a specific client.
        - Examples: Custom enterprise systems, tailored e-commerce platforms.

Characteristics of Generic Software

- **Target Audience:** Multiple customers with common needs.
- **Features:**
  - Pre-defined and standardized functionality.
  - Limited customization options.
- **Development Approach:** Focus on reuse and scalability.
- **Examples:**
  - Operating systems, productivity tools (e.g., Word processors).
  - Commercial software packages.

Characteristics of Bespoke Software

- **Target Audience:** Specific client or organization.
- **Features:**
  - Highly tailored to client requirements.
  - Flexibility to adapt to changing needs during development.
- **Development Approach:** Close collaboration with the client.
- **Examples:**
  - Custom CRM systems.
  - Tailored banking or healthcare systems.

Comparison of Generic vs. Bespoke Software

Feature	**Generic Software**	**Bespoke Software**
Target Audience	Broad market	Specific client
Customization	Limited	High
Development Cost	Shared among customers	Borne by client
Development Speed	Faster due to reuse	Slower due to custom design
Examples	Word processors, Antivirus	Custom CRM, Enterprise Systems

Types of Software with Software Process Models

- **Software Process Models:**
  - Generic software often uses reuse-oriented models.
  - Bespoke software aligns with plan-driven or agile models.

Scenario-Based Questions

1. Scenario 1: Your team is tasked with developing a mission-critical software system for a spacecraft. The system requires a high degree of reliability, and the requirements are well-documented and unlikely to change.

   Question: Which process model would you choose and why?

2. Scenario 2: A startup is building an e-commerce website but is unsure about all the features their customers might need. They want to deploy a basic version quickly and then add features based on user feedback.

   Question: Which process model is best suited for this scenario? Explain your choice.

Scenario-Based Questions

3. Scenario 3: Your client has a vague idea of what they want but insists on seeing a working prototype before finalizing the requirements.

   Question: Which process model should you use, and why?

4. Scenario 4: You're working on a government project where strict compliance with legal and procedural standards is required, and the project must follow a pre-defined plan.

   Question: Which process model fits best, and why?

Scenario-Based Questions

5. Scenario 5: A software company is developing a mobile game app with tight deadlines and a focus on delivering the core functionality first. The team anticipates major design changes based on user feedback during development.

   Question: What process model would you recommend, and why?

6. Scenario 6: A team is building a software system for a bank where security and precision are the top priorities. The client has provided detailed requirements but doesn't want to see incremental releases.

   Question: Which process model would you choose, and why?

7. Scenario 7: You are part of a team developing a new AI-based recommendation engine. The requirements are not fully understood yet, and there's a need for exploratory development and experimentation.

   Question: What process model would you recommend and why?

Matching Questions

Match the following scenarios with the most suitable process model:

1. Scenario A: Developing a large-scale enterprise software system with a clearly defined scope.
2. Scenario B: Building a web application where features will evolve based on market trends.
3. Scenario C: Developing a mobile app where customer feedback will determine the next features.
4. Scenario D: Designing a simulation system for a military training program where requirements are strict and predefined.

Options:

1. Waterfall
2. Agile
3. Incremental Development
4. Prototyping

Scenario-Based Questions-Answers

1. Scenario 1: Your team is tasked with developing a mission-critical software system for a spacecraft. The system requires a high degree of reliability, and the requirements are well-documented and unlikely to change.

   Question: Which process model would you choose and why?

   The Waterfall Model

   - High Reliability and Predictability
   - Well-Documented Requirements
   - Thorough Verification and Validation (Testing)
   - Compliance with Standards (as heavy documentation is used)

Scenario-Based Questions-Answers

1. Scenario 2: A startup is building an e-commerce website but is unsure about all the features their customers might need. They want to deploy a basic version quickly and then add features based on user feedback.

   Question: Which process model is best suited for this scenario? Explain your choice.

   Agile Process Model/Incremental

   - Gradual Feature Development/Incremental Delivery
   - Feedback-Driven Improvements/Continuous Customer Feedback
   - Focus on Refinement

   Limitations Compared to Agile:

   Longer Feedback Loop:

   In Iterative, feedback is incorporated after an iteration, which might delay response time compared to Agile's continuous feedback.

   Lack of Continuous Customer Collaboration: While Iterative involves some customer input, it does not prioritize constant collaboration as Agile does.

Scenario-Based Questions-Answers

1. Scenario 3: Your client has a vague idea of what they want but insists on seeing a working prototype before finalizing the requirements.

   Question: Which process model should you use, and why?

   Prototype Model

   - Vague Requirements
   - Focus on Early Validation
   - Iterative Refinement
   - Minimizing Misunderstandings

Scenario-Based Questions-Answers

1. Scenario 4: You're working on a government project where strict compliance with legal and procedural standards is required, and the project must follow a pre-defined plan.

   Question: Which process model fits best, and why?
   Waterfall Model

   - Strict Compliance with Standards
   - Pre-Defined Plan
   - High-Level Documentation
   - Low Risk of Scope Creep

Scenario-Based Questions-Answers

1. Scenario 5: A software company is developing a mobile game app with tight deadlines and a focus on delivering the core functionality first. The team anticipates major design changes based on user feedback during development.

   Question: What process model would you recommend, and why?
   Agile

   - frequent iterations and delivers core functionality quickly.
   - The process allows rapid adaptation to user feedback, enabling the team to refine the game during development.

Scenario-Based Questions-Answers

1. Scenario 6: A team is building a software system for a bank where security and precision are the top priorities. The client has provided detailed requirements but doesn't want to see incremental releases.

   Question: Which process model would you choose, and why?

   Waterfall

   - Banks require strict compliance with security and precision standards, making Waterfall a suitable choice for its focus on comprehensive requirements and design phases before coding.

Scenario-Based Questions-Answers

1. Scenario 7: You are part of a team developing a new AI-based recommendation engine. The requirements are not fully understood yet, and there's a need for exploratory development and experimentation.

   Question: What process model would you recommend and why?
   Iterative Development or Agile

   - focuses on adaptive planning, flexibility, and incremental delivery, which are ideal for exploratory development.
   - incremental and iterative nature of Agile allows the team to experiment and refine the engine over multiple sprints.