Agile Development

SWE mid prep. By **Efrat A.**

—

Has multiple models customer satisfaction: early and continuous delivery adaptive to change; continuous customer collab; no going back methods: XP(eXtreme Programming), scrum.

What is a Methodology?

•A methodology is a formalized process or set of practices for creating software
•A set of rules you have to follow.
•A systematical, engineering approach for organizing software projects
set of rules.

The Waterfall Process:

1. **System**
2. **Requirements Software**
3. **Requirements Analysis**
4. **Program Design**
5. **Coding**
6. **Testing**
7. **Operations**

Waterfall development Model: traditional process

cons: modify, change, update is difficult, high risk
needs to go back to change; not adaptive to change
use case: small projects

XP:

Most prominent agile`dev 12 key practices-

1. **Metaphor:**

   A shared story or analogy that helps the team understand and communicate about the system being developed. It provides a common understanding of how the software should work.

2. Release Planning:

Small releases in Extreme Programming (XP) involve dividing a project into bite-sized, frequent iterations of development and delivery. This approach ensures that working software is released regularly, delivering value incrementally, reducing risk, and maintaining close collaboration with customers throughout the project.

In Release Planning, the project requirements are taken using User Stories, which is basically a description in natural language of what the client wants from the project. After that the resources needed, and the risks that can happen are estimated by the developers. This planning is done after every increment, which is after a certain part of the project is complete and client feedback is taken.

3. (TDD) Test-Driven Development :

XP promotes the use of automated testing to ensure the quality of the code. Test-driven development (TDD) is a common XP practice where tests are written before the code. Tests are automated.

4. Pair Programming:

Pair Programming involves two developers working together on the same piece of code. One writes the code (the "driver"), and the other reviews, suggests improvements, and ensures code quality (the "navigator"). This practice enhances code quality and knowledge sharing.

5. Refactoring:

Refactoring is the practice of improving the internal structure and design of the code without changing its external behavior. It ensures that the code remains maintainable, readable, and adaptable as the project evolves. It helps maintain code quality and readability.

6. Simple Design:

XP encourages keeping the design of the software as simple as possible. Overly complex solutions are avoided.

7. Collective Code Ownership:

Team members are collectively responsible for the entire codebase and any developer can make changes to any part of the code.

8. Continuous Integration:

Code changes are frequently integrated into the main codebase, and automated builds and tests are performed to detect integration issues early.

9. On-Site Customer:

Ideally, a customer or a representative should be available on-site to answer questions and provide feedback during the development process.

10. Small Releases:

XP encourages frequent and small releases of working software. Instead of waiting for a large, monolithic release, small, incremental changes are delivered to users regularly to obtain early feedback and provide value sooner.

11. Forty-Hour Work/Week:

XP focused on a sustainable work pace. Team members are encouraged to work a standard 40-hour workweek, avoiding excessive overtime, to maintain productivity and prevent burnout.

12. Coding Standards:

Coding Standards are guidelines and conventions that ensure consistency and maintainability in the codebase. Adhering to coding standards makes it easier for team members to read and work on each other's code.

Tamims note:

1. **Metaphor**
   story of how the whole system works; a clear picture
   what are the roles for each employee
2. **Release Planning**

Requirements via User stories (card); what customer wants; customer priority.
cons: SRS is better. this is not a proper resource/requirement.
Resource & risk estimation; by devs; "the planning game"; repeat planning after each increment.
small releases have less risk

3. **Testing**

Test Driven Development: test before code; automated; must run 100% to proceede; not for simple any easy projects;

4. **Pair Programming**

Two soft. engineer for one task

The driver: who codes
The navigator: watches over driver; identify errors;
roles are rotated
pros: high quality code; faster task completion; great for complex, critical logics; when quality is priority;
cons: resource(HR) heavy, trivial task don't need this; Peer review is better.

5. **Refactoring**z

improve the code design (ie: make it faster), not functionality; relies on unit testing (to ensure code is ok)
remove BAD SMELL in code: long methods, duplicate codes, bad cohesion, too many dependencies (bad coupling), complex code
pros: faster delivery of working software.

6. **Simple Design**

No Big Design Up Front (BDUF): ship faster;
"Do the simplest thing that could possibly work"
Not too much formal documentation.
simple does not mean no design
establish priorities.

7. **Collective Code Ownership**

improves code quality; faster progress; fix and go.
this is a must practice. don't give a separate ownership of any module to any one employee. the code will be accessed by everyone in the project.

8. **continuous intigration**:

not necessary for small projects.

9. **onsight customer**:

not realistic. no customer would actually do this. too much interfierance. clients ` don't always understand what is better. clients are not competent to understand code. frequent decision change.
meetings are better. use prototype.

10. **small releases**:

valuable; lowers the risk; helps customer to define better requirement;

11. **forty-hour-work-week**:

affects projects performance.
better planning needed.

12. **coding standards**:

follow coding conventions.
self documenting code (using comments)
don't comment bad code - remove or rewrite code!
code audit tools (FxCop, CheckStyle, TFS)
larger team needs strict maintainance of standards.

13th practice: **the stand up meeting**:

day start w/ 15 min meeting.
everyone stands up
everyone knows: past day tasks, todays plan, experience or problems

SCRUM: Agile framework

SCURUM PLANING :

Every Sprint starts with Sprint Planning where the Scrum Team determines what they plan to accomplish during the course of the Sprint. They make this transparent by creating a Sprint Backlog including the Sprint Goal, the selected Product Backlog Items and the Developers' plan for delivering the work

Roles:

Product Owner: Represents the customer and defines what needs to be built.
Scrum Master: Facilitates the Scrum process and ensures the team follows the principles.
Development Team: A self-organizing group responsible for delivering the product.
Artifacts:

Product Backlog:

A prioritized list of features, bug fixes, or other work that needs to be done.
Sprint Backlog: The subset of items from the Product Backlog that the team commits to complete during a sprint.
Increment: The product's potentially shippable product after each sprint.
Events:

Sprint:

A time-boxed period (typically 2-4 weeks) during which a set of work is completed.
Daily Scrum: A daily stand-up meeting for the team to synchronize and plan their work.
Sprint Review: A meeting at the end of the sprint to demonstrate completed work to stakeholders.
Sprint Retrospective: A meeting at the end of the sprint to reflect on the process and identify areas for improvement.

Principles:

Transparency: Everyone involved should have a common understanding of the work.
Inspection: Progress is continually monitored to identify variances from the plan.

What is Scrum?

Scrum is an agile project management framework for developing, delivering, and sustaining complex products. It emphasizes iterative and incremental progress, close collaboration, and adaptability to changing requirements. Scrum provides a structured way for teams to work together to deliver value to the customer.

2.1. What is a Scrum Master?

The Scrum Master is a facilitator and servant leader in the Scrum team. They help the team understand and apply Scrum principles, remove impediments, and ensure that Scrum events and roles are effectively utilized.

2.2. What is a Product Owner?

The Product Owner is responsible for defining and prioritizing the product backlog, representing the customer or stakeholder interests, and ensuring that the team is building the right features and delivering value. He controls the Product Backlog.

2.3. What is a Developer?

Developers are the members of the Scrum team responsible for designing, coding, testing, and delivering the product. They work collectively to turn product backlog items into a potentially shippable product increment.
The Scrum Events

3.1. What is a Sprint?

A Sprint is a time-boxed period (usually 2-4 weeks) during which a Scrum team works to deliver a potentially shippable product increment. Sprints provide a predictable cadence for development activities.

3.2. What is Sprint Planning?

Sprint Planning is a collaborative event at the beginning of each Sprint where the team selects work from the product backlog to accomplish during the Sprint and creates a plan for how to complete it.

3.3. What is a Daily Scrum?

The Daily Scrum is a brief, daily stand-up meeting where the team members share progress, plan the day's work, and discuss any impediments. It lasts around 15 minutes and helps keep the team aligned.

3.4. What is a Sprint Review?

A Sprint Review is held at the end of each Sprint to demonstrate the product increment to stakeholders and obtain their feedback. It helps validate the work done during the Sprint.

3.5. What is a Sprint Retrospective(পূর্ববর্তী)?

A Sprint Retrospective is a meeting at the end of each Sprint where the team reflects on their processes and identifies opportunities for improvement. It focuses on enhancing teamwork and the development process.
The Scrum Artifacts

4.1. What is a Product Backlog?

The Product Backlog is a prioritized list of features, user stories, and requirements for the product. It serves as the single source of requirements for the Scrum team to work on.

4.2. What is a Product Goal?

The Product Goal is an overarching objective for the product, aligning the work of the Scrum team with the desired outcome and providing direction for the product's development.

4.3. What is a Sprint Backlog?

The Sprint Backlog is a dynamic document that contains the work items selected by the team for the current Sprint. It includes detailed tasks and commitments for the Sprint.

4.4. What is a Sprint Goal?

The Sprint Goal is a short description of the purpose and desired outcome of the Sprint. It provides focus and clarity to the team on what they aim to achieve in the Sprint.

4.5. What is an Increment?

An Increment is the sum of all the work completed during a Sprint, resulting in a potentially shippable product. It must be in a consistent and usable state, adhering to the Definition of Done.

4.6. What is a Definition of Done?

The Definition of Done is a shared understanding within the team of what "done" means for a product increment. It defines the quality criteria that must be met for an item to be considered complete. It ensures that the product is of high quality and can be potentially shipped at the end of each Sprint.

Adaptation: Changes are made as needed to achieve the goals.

*scrum team:

1. **scrum master**
    - accountable for establishing scrum. organizer.
    - client communication
    - discuss w/ product owner
2. **product owner**
    - handles backlog (list of requirements)
    - maximize product value
    -
3. **developers**
    - not necessarily a soft. dev.

scrum events

sprint: timeframe of updates

```
----------| ooooooooo (update)
sprint       | increment
| | | | | | | |
 2 3 3..   4 5 - parts of sprint
```

1. sprint
2. sprint planning
3. daily sprint (repeats)
4. sprint review
5. sprint retrospective

srum artifacts

1. product goal
    - long term target
2. product backlog
    - target: product goal
3. sprint goal
4. sprint backlog
    - target: sprint goal
5. increment

6. definition of Done

Uses of all models:

Waterfall, Agile, and Scrum are three different project management methodologies used in various types of projects based on their specific characteristics and requirements. Here's an overview of when each of these methodologies is typically used:

1. Waterfall:

The waterfall is a traditional and linear project management methodology where each phase must be completed before moving on to the next. It is best suited for projects that have well-defined requirements and where changes are expected to be minimal throughout the project. Waterfall is commonly used in industries like construction, manufacturing, and some software development projects with straightforward, stable, and well-understood requirements. It is less suitable for projects that require flexibility and frequent changes.

2. Agile:

Agile is an iterative and incremental approach to project management that is ideal for projects with rapidly changing requirements and where frequent feedback and adaptability are essential. Agile is commonly used in software development, but it can also be applied to various other projects, including product development and marketing. Agile methodologies, such as Scrum and Kanban, emphasize collaboration, continuous improvement, and delivering small increments of work in short cycles (sprints or iterations). Agile is particularly effective when the end goal of the project is not fully known or may evolve during the project.

3. Reason for Using Agile (Scrum):

Client Collaboration:

Agile methodologies, such as Scrum, emphasize close collaboration between the development team and the client or product owner. This is crucial when you need the client to be actively involved in meetings and provide clarifications to developers.

Iterative and Incremental:

Agile methodologies promote incremental development and iterative cycles, making it possible to release working increments of the business intelligence tool at regular intervals. This aligns well with your need for periodic releases.

Flexibility:

Agile allows for changing requirements and priorities, which is important in the business intelligence domain where client needs may evolve during the project.

Scrum is a specific framework within the Agile methodology that provides a structured approach to project management. It is particularly useful for software development projects, where small, cross-functional teams work in fixed-length time boxes called sprints to deliver increments of the product. Scrum promotes transparency, collaboration, and adaptability, making it suitable for projects with complex requirements and where customer feedback plays a crucial role in shaping the final product. Scrum is characterized by its roles (Product Owner, Scrum Master, and Development Team), events (Daily Standup, Sprint Planning, Sprint Review, and Sprint Retrospective), and artifacts (Product Backlog, Sprint Backlog, and Increment).

In summary, the choice of project management methodology depends on the nature of the project, its requirements, and the level of uncertainty and change involved. The waterfall is suitable for well-defined, stable projects, while Agile and Scrum are more adaptive and are used in projects where requirements may change frequently and customer feedback is critical. Scrum, as a subset of Agile, offers a structured approach with specific roles, events, and artifacts that can be especially beneficial in software development projects. It's essential to select the methodology that best aligns with the project's goals, constraints, and expected changes.

let's describe Waterfall, XP (Extreme Programming), and Scrum, which are three different software development methodologies:

Waterfall:

- Description: The Waterfall model is a linear and sequential approach to software development. It consists of distinct phases, and each phase must be completed before moving on to the next one. The phases typically include requirements, design, implementation, testing, deployment, and maintenance.
- Key Characteristics:
  - Progression through phases is strictly linear.
  - Emphasis on comprehensive documentation at each phase.
  - Changes in requirements are discouraged once the project has started.
  - Suited for projects with well-defined and stable requirements.
- Advantages:
  - Clear project milestones and documentation.
  - Easy to understand and manage.

- Disadvantages:
    - Limited flexibility for changes during the development process.
    - Late detection of issues in the testing phase.

Extreme Programming (XP):

- Description: XP is an agile software development methodology that emphasizes adaptability, collaboration, and customer satisfaction. It is known for its practices, such as continuous integration, test-driven development, pair programming, and frequent releases.
- Key Characteristics:
    - Emphasis on customer involvement and feedback.
    - Iterative and incremental development.
    - Continuous testing and continuous integration.
    - Close collaboration between developers and customers.
- Advantages:
    - Flexibility to accommodate changing requirements.
    - Improved software quality through continuous testing.
    - High customer satisfaction due to frequent deliverables.
- Disadvantages:
    - Some practices may be challenging to adopt initially.
    - Requires a high level of collaboration and communication.

Scrum:

- Description: Scrum is an agile framework for managing and organizing complex work. It is based on the principles of transparency, inspection, and adaptation. Scrum divides work into fixed-length iterations called sprints and uses various roles (Scrum Master, Product Owner, and Development Team) and events (Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective) to structure the development process.
- Key Characteristics:
    - Iterative development with fixed-length sprints.
    - Emphasis on collaboration and self-organizing teams.
    - Regular inspection and adaptation through ceremonies.
    - Product backlog prioritization by the Product Owner.

- Advantages:
    - Flexibility to adapt to changing requirements.
    - Regular opportunities for feedback and improvement.
    - Increased transparency and visibility into project progress.
- Disadvantages:
    - May be challenging for large, complex projects.
    - Requires active involvement and commitment from all team members.

Each methodology has its strengths and weaknesses, and the choice of methodology often depends on the nature of the project, the level of uncertainty in requirements, and the team's preferences and expertise.

Version control with Git and GitHub

Initialize a Git repository:

1. Check if Gitis properly installed: `git--version`
2. Put information about you (author in the Git):
    - Sets the name you want attached to your commit transactions:
      `git config -global user.name "[name]"`
    - Sets the email you want attached to your commit transactions:
      `git config -global user.email "[email address]"`
    - Enable helpful colorization of command line output:
      `git config -global user.email "[email address]"`

Create a Repository:

1. Initialize a Git repository: `git init`
2. Specifies the remote repository for your local repository :

   ```
   git remote add origin <repository-url>
   ```

3. Clone a repository: `git clone <repository-url>`

Track changes using git:

1. Check the status of your repository: `git status`
2. Add files to the staging area: `git add <filename>`
3. Commit changes: `git commit -m "Your commit message"`

Useful Commands for Tracking:

1. View commit history: `git log`
2. Version history for a file, beyond renames (works only for a single file: `git log --follow [file]`
3. Show content differences:

   ```
   git diff[first-branch]...[second-branch]
   ```

4. Output metadata and content change of the specified commit: `git show [commit]`
5. Snapshots the file in preparation for versioning: `git add [file]`
6. Records file snapshots permanently: `git clone <repository-url>`

Useful Commands for Tracking:

1. Create a new branch: `git branch <branch-name>`
2. Switch to a branch: `git checkout <branch-name>`
3. Merge branches: `git merge <branch-name>`
4. Delete a branch: `git branch -d <branch-name>`

Syncing Remote Branches:

1. Download all history from the remote tracking branches:

   ```
   git fetch origin
   ```

2. Combines remote-tracking branches into the current local branch: `git merge`
3. Upload all local branch commits to GitHub:`git push`
4. Update current local working branch with all new commits:

   ```
   git pull
   ```

Question Solve- 2023 spring

Git question:

Write appropriate git commands to perform the following the tasks:–

1. - Initiate a git repository
2. - Create three branches named b1, b2, and b3
3. - Consider that, you have created a file named “SE.cpp” in the local directory, and commit in b1
4. - Consider that, you have created a file named “SE_newFeature.cpp” in the local directory, and commit in b2
5. - Update “SE.cpp”, and commit it again in b1
6. - Consider that, you have created a file named “SE_newFeature2.cpp” in the local directory, and commit in b3
7. - Update again “SE.cpp” and commit it again in b1
8. - Edit the file “SE_newFeature.cpp”, and commit it again in b2

9. - Revert back to the first commit of b1 in such a way that only contents of 1st commit will be preserved (Let hash code of 1st commit of b1 is “a11111”
10. - Merge b1, b2, and b3 with the master branch

1. Initiate a Git Repository:

bash Copy code

```
git init
```

2. Create Three Branches:

bash Copy code

```
git branch b1
git branch b2
git branch b3
```

3. Commit in Branch b1:

bash Copy code

```
git checkout b1
echo "Content for SE.cpp" > SE.cpp
git add SE.cpp
git commit -m "Commit in b1"
```

4. Commit in Branch b2:

bash Copy code

```
git checkout b2
echo "Content for SE_newFeature.cpp" > SE_newFeature.cpp
git add SE_newFeature.cpp
git commit -m "Commit in b2"
```

5. Update and Commit in Branch b1:

bash Copy code

```
git checkout b1
echo "Updated content for SE.cpp" >> SE.cpp
git add SE.cpp
git commit -m "Update in b1"
```

6. Commit in Branch b3:

bash Copy code

```
git checkout b3
echo "Content for SE_newFeature2.cpp" > SE_newFeature2.cpp
git add SE_newFeature2.cpp
git commit -m "Commit in b3"
```

7. Update and Commit Again in Branch b1:

bash Copy code

```
git checkout b1
echo "Another update for SE.cpp in b1" >> SE.cpp
git add SE.cpp
git commit -m "Another update in b1"
```

8. Edit and Commit Twice in Branch b2:

bash Copy code

```
git checkout b2
# Edit SE_newFeature.cpp
echo "Updated content for SE_newFeature.cpp" >> SE_newFeature.cpp
git add SE_newFeature.cpp
git commit -m "Edit 1 in b2"

# Edit SE_newFeature.cpp again
echo "More updates for SE_newFeature.cpp" >> SE_newFeature.cpp
git add SE_newFeature.cpp
git commit -m "Edit 2 in b2"
```

9. Revert to the First Commit of b1:

bash Copy code

```
git checkout b1
git revert --no-commit a11111^..HEAD
git commit -m "Revert to the first commit of b1"
```

Note: Replace "a11111" with the actual hash code of the first commit of b1.

10. Merge b1, b2, and b3 with the Master Branch:

bash Copy code

```
git checkout master

# Merge b1
git merge b1

# Merge b2
git merge b2

# Merge b3
git merge b3
```

Scrum question

1. (i) Explain the roles of people involved in scrum activity. [CO1] 2+4
(ii) Consider that, you are working on a powerful business intelligence tool, and you need to release it
periodically. As a project manager, you want someone from the client who will be present as a part
of the development process in the meetings, and clear the developer's doubts. [CO1]
- Which model should be used in this scenario and why?
- Also describe the processes involved in that model considering the above scenario.

1. **Product Owner:**
Defines the product backlog, which is a prioritized list of features, enhancements, and fixes.
Represents the voice of the customer and stakeholders.
Makes decisions on what features to include in each sprint based on business value.

2. **Scrum Master:**
Ensures the Scrum framework is followed.
Facilitates communication and collaboration within the team.
Removes impediments or obstacles that hinder the team's progress.
Helps the team improve its processes and practices.

3. **Development Team:**
Cross-functional and self-organizing group responsible for delivering the product increment.
Determines how to accomplish the work in the sprint backlog.
Collaborates with the Product Owner to understand and deliver the highest-priority items.
4. **Stakeholders -**
Responsibilities:
Individuals or groups with an interest in the project's outcome.
Provide input and feedback during sprint reviews.
May attend Scrum events, such as sprint reviews, to gain insights into the project's progress.
5. Scrum Team:
Responsibilities:
Collective term for the Product Owner, Scrum Master, and Development Team.
Collaborates to achieve the goals of the project.
Works together during the sprint to deliver a potentially shippable product increment.

Requirements Engineering Process in Software Engineering:

1. Feasibility Study:
    - Objective: Determine whether the proposed software project is technically and economically feasible.
    - Activities: Assess project scope, analyze risks, estimate costs, and evaluate potential benefits.
    - Output: Feasibility study report with recommendations for project initiation or cancellation.
2. Requirements Elicitation:
    - Objective: Gather information from stakeholders to identify and understand their needs and expectations for the software system.

- Activities: Conduct interviews, surveys, workshops, and observations to capture requirements. Use prototypes and mockups to facilitate discussions.
- Output: Requirement documents, user stories, use cases, and other artifacts that describe the system's functionality, constraints, and quality attributes.

3. Requirements Analysis and Negotiation:
   - Objective: Analyze, prioritize, and resolve conflicts in the gathered requirements. Ensure that the requirements are clear, complete, and feasible.
   - Activities: Prioritize requirements, identify dependencies, and resolve conflicting requirements. Engage stakeholders in negotiations to reach a consensus.
   - Output: Refined and clarified requirement documents, updated prioritization, and resolution of conflicts.
4. Requirements Specification:
   - Objective: Document the requirements in a clear, unambiguous, and comprehensive manner to serve as a basis for design and implementation.
   - Activities: Create formal requirement documents, including functional requirements, non-functional requirements, and any necessary supporting documentation (e.g., data models, system architecture diagrams).
   - Output: Requirement specifications that serve as a contract between the development team and stakeholders.
5. Requirements Validation:
   - Objective: Ensure that the specified requirements accurately represent the needs of the stakeholders and are suitable for further development.
   - Activities: Perform reviews, inspections, and walkthroughs of the requirements documentation. Use prototypes and simulations to validate user interfaces and workflows.
   - Output: Validated requirement documents with identified issues addressed.
6. Requirements Management:
   - Objective: Control changes to the requirements and ensure that the evolving project remains aligned with the stakeholders' needs.
   - Activities: Establish a change control process, track and manage changes, and update documentation accordingly. Communicate changes to stakeholders.
   - Output: Updated requirement documents reflecting approved changes.

The Requirements Engineering Process is not a linear process but often involves iteration and feedback loops, especially as the understanding of the system evolves and new information becomes available. Effective communication and collaboration with stakeholders are crucial throughout the process to ensure that the software system meets the expectations of end-users and other relevant parties.

Security:

Trojan Horse:

- Description: A Trojan horse, or Trojan, disguises itself as legitimate software but contains malicious code. Once the user installs the seemingly harmless program, the hidden malicious code is executed, allowing unauthorized access or causing harm to the system.
- Example: Zeus (Zbot) is a Trojan horse that primarily targets financial information. It often infects computers through phishing attacks and is known for stealing online banking credentials.

Worm:

- Description: Worms are self-replicating malware that can spread across networks without user interaction. They often exploit vulnerabilities in operating systems or network protocols to propagate.
- Example: Conficker is a notorious worm that spread through Windows-based systems, taking advantage of unpatched systems and weak passwords. It could download additional malware, steal sensitive information, and create a botnet.

Ransomware:

- Description: Ransomware encrypts a user's files, rendering them inaccessible, and demands payment (usually in cryptocurrency) in exchange for the decryption key.
- Example: CryptoLocker is a well-known ransomware variant that emerged in 2013. It encrypted files on infected systems and demanded payment in Bitcoin. The success of CryptoLocker spawned numerous copycat ransomware attacks.

Spyware:

- Description: Spyware is designed to spy on a user's activities without their knowledge. It can monitor keystrokes, capture screen shots, and gather sensitive information.

- Example: FinFisher, also known as FinSpy, is spyware that has been used for targeted surveillance. It can record conversations, capture screen activity, and log keystrokes.

Adware:

- Description: Adware displays unwanted advertisements on a user's device, often in the form of pop-up ads or banners. While not always malicious, some adware can be intrusive and compromise user privacy.
- Example: Superfish was pre-installed adware on some Lenovo laptops. It injected third-party ads into web pages and even compromised SSL/TLS security by installing its own certificate authority.

Rootkit:

- Description: Rootkits are designed to hide the presence of other malicious software on a system. They often manipulate the operating system to conceal their activities.
- Example: Sony BMG rootkit, used in some Sony music CDs, caused controversy in 2005. It was designed to prevent unauthorized copying but posed significant security risks by hiding its presence and creating vulnerabilities that could be exploited by other malware.

Password Attack:

Brute Force Attacks:

- Description: In a brute force attack, an attacker systematically tries all possible combinations of passwords until the correct one is found. This method can be time-consuming, but it's effective if the password is weak or short.
- Prevention: Use strong, complex passwords with a combination of uppercase and lowercase letters, numbers, and special characters. Additionally, implement account lockout policies that temporarily lock an account after a certain number of failed login attempts.

Dictionary Attacks:

- Description: In a dictionary attack, an attacker uses a precompiled list of common passwords, known as a "dictionary," to attempt to gain access. This is more efficient than a brute force attack and targets the likelihood of users using easily guessable passwords.
- Prevention: Use complex passwords that are not easily found in dictionaries. Additionally, implement account lockout policies and monitor for multiple failed login attempts.

Credential Stuffing:

- Description: In a credential stuffing attack, attackers use username and password combinations obtained from previous data breaches on other websites to gain unauthorized access to user accounts on a target system. Many users reuse passwords across multiple accounts, making this type of attack effective.
- Prevention: Encourage users to use unique passwords for each account and employ multi-factor authentication (MFA) to add an extra layer of security.

Phishing Attacks:

- Description: Phishing attacks involve tricking users into revealing their passwords by posing as a trustworthy entity. This can be done through deceptive emails, fake websites, or other social engineering techniques.
- Prevention: Educate users about recognizing phishing attempts, and encourage them to verify the legitimacy of emails and websites before providing any login credentials.

➢ **Man-in-the-middle: Sniffing the communication packets**
➢ **Credential Stuffing: Signed in from another device**

Man-in-the-Middle (MitM) Attack - Sniffing Communication Packets:

- Description: In a Man-in-the-Middle attack, an attacker intercepts and potentially alters the communication between two parties without their knowledge. Sniffing communication packets involves capturing and analyzing data packets exchanged between the communicating parties, which can include sensitive information such as login credentials.
- Prevention: Use secure communication channels such as HTTPS, employ encryption, and use virtual private networks (VPNs) to protect against eavesdropping.

Credential Stuffing - Signed in from Another Device:

- Description: Credential stuffing is a type of attack where attackers use previously leaked or stolen username and password combinations to gain unauthorized access to user accounts on various online services. The mention of "signed in from another device" suggests that attackers may use compromised credentials to log in from a new or different device.

Prevention: **Encourage users to use unique passwords for each account, implement multi-factor authentication (MFA), and monitor for suspicious login activity**.

SQL INJECTION:

SQL injection is a type of cyber attack that targets the vulnerabilities in a website's or application's database layer. It occurs when an attacker is able to manipulate an SQL query by injecting malicious SQL code into the input fields of a web form or URL parameters. This can lead to unauthorized access, manipulation, or retrieval of data from the database.

Here's a more detailed explanation of SQL injection:

Vulnerability Exploitation:

- Injection Points: SQL injection typically occurs when an application does not properly validate or sanitize user inputs before incorporating them into SQL queries. Common injection points include input fields in web forms, URL parameters, or any user-controlled data that interacts with the database.

Attack Techniques:

- Union-Based SQL Injection: In a union-based attack, an attacker exploits the SQL UNION operator to combine the results of the original query with those of a malicious query.
- Time-Based Blind SQL Injection: Attackers exploit time delays in the database's response to infer information about the database structure.
- Error-Based SQL Injection: Attackers deliberately cause SQL errors to extract information about the database structure or to gain insights into the application's inner workings.

Impact:

- Unauthorized Access: Successful SQL injection attacks can allow attackers to bypass login mechanisms, gaining unauthorized access to sensitive data or even administrative functions.
- Data Manipulation: Attackers can modify, delete, or insert data into the database, potentially causing data corruption or loss.

- Data Extraction: Attackers can extract sensitive information from the database, such as usernames, passwords, or other confidential data.

Prevention:

- ➢ **Parameterized Statements (Prepared Statements): Use parameterized queries or prepared statements, which separate SQL code from user input, making it more difficult for attackers to inject malicious code.**
- ➢ **Input Validation and Sanitization: Validate and sanitize user inputs to ensure that they conform to expected formats and do not contain malicious characters.**
- ➢ **Least Privilege Principle: Limit the permissions of database accounts to the minimum necessary for the application to function, reducing the potential impact of a successful SQL injection attack.**

Attacks:

DoS (Denial of Service) and DDoS (Distributed Denial of Service):

- DoS: Denial of Service attacks aim to disrupt or disable a system or network by overwhelming it with a flood of traffic. This can be achieved by sending a large volume of requests, consuming all available resources and making the system or network unavailable to legitimate users.
- DDoS: Distributed Denial of Service attacks involve multiple systems, often forming a botnet, coordinating to flood the target with traffic. DDoS attacks are more difficult to mitigate than traditional DoS attacks due to the distributed nature of the attack.

Cross-Site Scripting (XSS):

- Description: XSS is a type of web security vulnerability where attackers inject malicious scripts into web pages that are then viewed by other users. These scripts can steal sensitive information, manipulate web content, or perform actions on behalf of the user without their consent.
- Types: There are different types of XSS attacks, including Stored XSS, Reflected XSS, and DOM-based XSS, each targeting different aspects of web application security.
- Prevention: Developers can prevent XSS by validating and sanitizing user inputs, using secure coding practices, and implementing Content Security Policy (CSP) headers.

Birthday Attack:

- Description: The birthday attack is a cryptographic attack that exploits the mathematics of probability related to the birthday problem. It

demonstrates the likelihood of two people sharing the same birthday in a group and applies the same concept to hash functions.

- Hash Collisions: In the context of a birthday attack on hash functions, the goal is to find two different inputs that produce the same hash output. This could lead to security vulnerabilities, especially in situations where unique hash outputs are expected.
- Mitigation: To mitigate the risk of birthday attacks, cryptographic hash functions should use longer hash lengths (larger bit sizes) to make it computationally infeasible for attackers to find collisions.

Eavesdropping:

- Description: Eavesdropping, or sniffing, is the unauthorized interception of electronic communications, such as network traffic or phone conversations. Attackers can capture and monitor data transmitted over a network, potentially gaining access to sensitive information.
- Mitigation: Encryption plays a crucial role in mitigating the risks of eavesdropping. Implementing protocols like HTTPS for web traffic and using virtual private networks (VPNs) for secure communication can help protect against eavesdropping attacks.

CT 1 Question & Solve:

Q1. Explain the following XP principles stating an advantage and a disadvantage for each:

i) Continuous Integration
ii) Code Refactoring
iii) Collective Ownership
iv) On-Site Customer

Q2. Define the following Scrum terminologies:
i) Sprint Retrospective
ii) Sprint Backlog
iii) Product Owner
iv) Daily Scrum

SOLVE:

Q1: XP (Extreme Programming) Principles:

i) **Continuous Integration:**

- ***Advantage:*** Ensures that changes made by different team members integrate smoothly and frequently, reducing the risk of integration issues later in the development process.
- ***Disadvantage:*** Requires a robust automated testing environment. Without proper testing, continuous integration can lead to the integration of faulty code, causing issues in the software.

ii) Code Refactoring:

- ***Advantage:*** Improves code maintainability and readability, making it easier for developers to understand and modify. This helps in the long-term evolution of the software.
- ***Disadvantage:*** If not done carefully or if not supported by comprehensive test suites, refactoring can introduce new bugs or issues into the code.

iii) Collective Ownership:

- ***Advantage:*** Fosters a sense of shared responsibility and accountability within the development team. Any team member can make changes to any part of the codebase, promoting collaboration and flexibility.
- ***Disadvantage:*** Requires effective communication and coordination to avoid conflicts or misunderstandings when multiple team members are working on the same code simultaneously.

iv) On-Site Customer:

- ***Advantage:*** Facilitates real-time communication and feedback between the development team and the customer, leading to a better understanding of customer needs and priorities.
- ***Disadvantage:*** May not be practical or cost-effective in some situations, especially for distributed teams or projects with customers located in different geographical regions.

Q2: Scrum Terminologies:

i) Sprint Retrospective:

- **Definition:** A Sprint Retrospective is a meeting held at the end of each sprint, where the Scrum team reflects on the just-concluded sprint to identify what went well, what could be improved, and what actions can be taken to make improvements.

ii) Sprint Backlog:

- **Definition:** The Sprint Backlog is a subset of the product backlog selected for a particular sprint. It contains the user stories and tasks that the development team plans to work on during the sprint.

iii) Product Owner:

- **Definition:** The Product Owner is a role in Scrum responsible for maximizing the value of the product by managing and prioritizing the product backlog. The Product Owner represents the voice of the customer and works closely with the development team.

iv) Daily Scrum:

- **Definition:** The Daily Scrum, also known as the Daily Standup, is a daily meeting where the Scrum team discusses progress, plans for the day, and any impediments. It is a short, time-boxed event to synchronize activities and ensure that the team is aligned with the sprint goals.