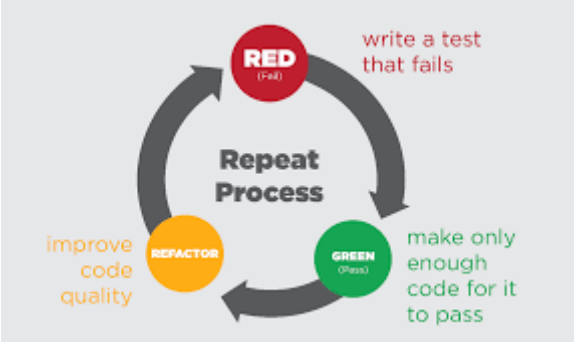
Assignment 2: Produce a comparative infographic of TDD, BDD, and FDD methodologies. Illustrate their unique approaches, benefits, and suitability for different software development contexts. Use visuals to enhance understanding.

a) Test-Driven Development (TDD)



Test-Driven Development (TDD) is a software development approach that emphasizes writing tests before writing the actual code. The core idea behind TDD is to ensure that every piece of code is thoroughly tested, leading to higher code quality and reliability. TDD follows a simple and iterative cycle, often referred to as the "red-green-refactor" cycle.

The TDD Cycle: Red, Green and Refactor

- **Red**: In the initial "Red" phase, the developer writes a failing test case that highlights the desired behavior not yet implemented.
- **Green**: In the "Green" phase, the developer writes the minimum amount of code necessary to pass the test.
- **Refactor**: In the "Refactor" phase, the developer improves the code without changing its behavior. This step enhances the design, removes duplication and improves maintainability.

Benefits of TDD

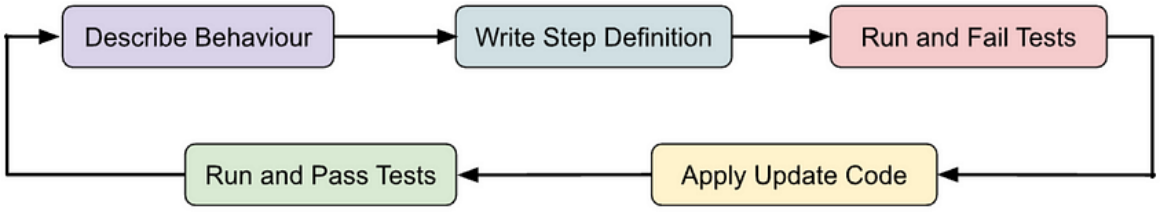
1. **Improved Code Quality**:
    - Writing tests first leads to better design decisions and cleaner, more modular code.

2. **Early Bug Detection**:
    - Since tests are written before the code, many bugs are caught early in the development process, reducing debugging time.
3. **Better Requirements Understanding**:
    - Writing tests requires a clear understanding of the requirements, leading to fewer misunderstandings and miscommunications.
4. **Facilitates Refactoring**:
    - With a comprehensive suite of tests, developers can refactor code with confidence, knowing that any regression will be caught.
5. **Documentation**:
    - Tests serve as documentation, providing examples of how the code is supposed to behave, which is particularly useful for new developers joining the project.
6. **Reduced Debugging Time**:
    - Since code is continually tested, the time spent debugging and fixing issues is significantly reduced.

Suitability for Different Software Development Contexts

- Agile Development
- Complex and Critical Systems
- Legacy Systems
- Startups and Rapid Prototyping
- Large and Distributed Teams
- Open Source Projects

b) Behavioral-Driven Development (BDD)



Behavioral-Driven Development (BDD) is a testing approach derived from the Test-Driven Development (TDD) methodology. In BDD, tests are mainly based on systems behavior. This approach defines various ways to develop a feature based on its behavior. In most cases, the *Given-When-Then* approach is used for writing test cases.

Example

- Given the user has entered valid login credentials
- When a user clicks on the login button
- Then display the successful validation message

As shown above, the behavior is illustrated in a very simple English language, a shared language. This helps everyone in the team responsible for development to understand the feature behavior.

Benefits of BDD

1. **Improved Communication**:
   - Enhances collaboration and communication between technical and non-technical team members, ensuring everyone has a clear understanding of requirements.
2. **Better Requirements Understanding**:
   - Focus on user stories helps in understanding the actual needs and expected behaviors, reducing misunderstandings.
3. **High-Quality Software**:
   - Ensures that software meets the user's needs and behaves as expected, leading to higher quality and user satisfaction.
4. **Reduced Development Cycle Time**:
   - Early detection of discrepancies between requirements and implementation reduces rework and speeds up the development cycle.
5. **Living Documentation**:
   - Tests act as up-to-date documentation that reflects the current state of the system.

Suitability:

- **Highly Suitable**: Customer-facing applications, Agile development, complex business logic, regulated industries, distributed teams.
- **Moderately Suitable**: Startups aiming for high quality, with consideration of initial overhead.

c) Features-Driven Development (FDD)

- It is a itertative incremental software developement methology.

- In FDD, the software product is developed feature by feature in an incremental approach.

- Focuses on developing the working software with the feature that satisfy the client needs.

Flow diagram of FDD with example:-

---

1. Develope Overall model
   | [step 1] > over all requiremnt
   | > define the objective of the system.
   | > gather all the client requirement.
   | > customer can able to register and login
   | search the product,filter the product,
   | place the order,order cancel,.....
   |
2. Build the feature list
   | [step 2] > build the feature list from
   | the requirement.
   |
   | feature list:
   | login,signup,reset password,
   | profile,social media logic,
   | search, filter,cart, order, payment,
   |
3. plan by feature [step 3]
   | profile feature development
   | > assign the team
   | > duration of the feature
4. design the feature [step 4]
   |
   | >before staring the implementation
   | create a design specification
   | HLD & LLD
   |
5. build the feature [step 5]
   >develope the feature based on design specification
   > coading and testing happens here.
   > feature integration.

6. test the feature

7. review the feature.

Benefits of FDD

1. **Clear Project Scope**:
    - By organizing development around features, FDD provides a clear understanding of the project scope and progress.
2. **Early Detection of Issues**:
    - Issues are detected early in the development cycle through feature-specific design and planning, reducing the risk of larger problems later on.
3. **Improved Team Collaboration**:
    - Fosters collaboration among team members by assigning clear responsibilities for each feature and promoting frequent communication.
4. **Predictable Progress Tracking**:
    - Progress can be easily tracked at the feature level, providing stakeholders with predictable timelines and milestones.
5. **Quality Assurance**:
    - Each feature undergoes its own design and testing phases, ensuring higher quality and reducing the likelihood of bugs slipping through.

Suitability:

- **Highly Suitable**: Large-scale projects, enterprise applications, team-based development, long-term projects, client-focused projects.
- **Moderately Suitable**: Projects in regulated industries, where thorough planning and documentation are necessary.