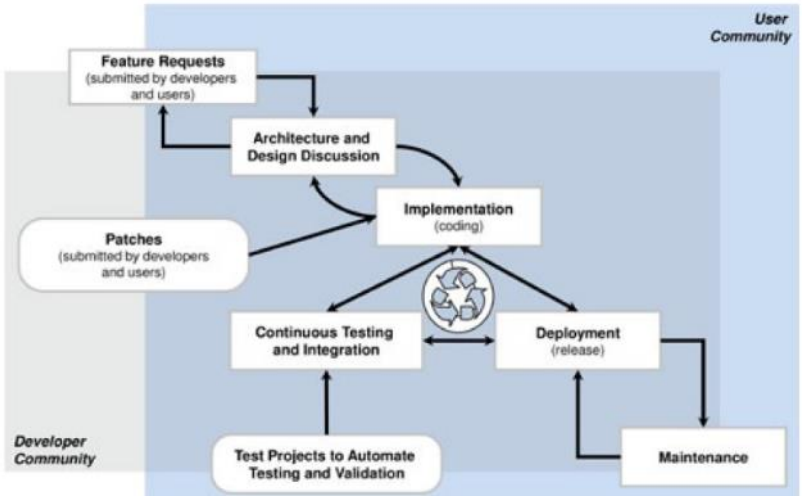
OPEN SOURCE SOFTWARE Q&A

1. What influences developers in open-source development?
    - **Intrinsic motivation:** Enjoyment and Amusement; Satisfaction and Fulfillment and Sense of Scientific Discovery, Creativity and Challenge
    - **Extrinsic motivation:** Reputation and Status; Signaling Incentives; Financial Incentives and Monetary Rewards
    - **Political / ideological motivation:** one's political, ideological or cultural beliefs
    - **Social motivation:** Sense of Belonging and Contributing to a Public Good; Collaboration and community
    - **Technological environment and working style:** Learning and Skills Development; Working with Bleeding-Edge Technology; Realization of Personal Ideas; Integration of Individual Fixes
    - **Process / Product**
    - **Openness**
2. Distinguish between free software and freeware?
    - **Free software:** refers to software that gives users the freedom to run, study, modify, and distribute the software. Examples of free software include Linux, GNU tools, and the Firefox web browser.
    - **Freeware:** refers to software that is available at no cost. It can be used without payment, but the source code may not be available, and users typically have limited rights to modify or distribute the software. Freeware is often proprietary software that is provided for free by the developer or company, such as Adobe Reader or Skype.
3. Why are open-source software considered more secure, reliable and robust?
    - **Transparency and peer review:** The source code of open-source software is open and accessible to anyone, allowing a large community of developers to review and audit the code. This transparency enhances security by identifying and fixing vulnerabilities more effectively.
    - **Rapid vulnerability identification and patching**: Users actively report vulnerabilities, leading to faster patches and updates to reduce the risk of exploitation.
    - **Customizability and flexibility:** Open-source software allows users to modify and customize the code to suit their specific needs, making it easier to adapt and enhance security features.
    - **Community support:** Open-source projects often have active communities that provide support and assistance. This community support fosters a proactive approach to addressing security concerns and maintaining reliability.
    - **Independent verification**: Security experts can independently examine the code, identify vulnerabilities, and propose improvements.
    - **Mitigation of hidden threats**: The open nature of the code makes it difficult for malicious actors to introduce hidden vulnerabilities.

4. With aid of diagram describe Feature life-cycle in OSS development model
   - **Feature Request Process:**
   - **Architecture and Design Discussion**
   - **Collaboration on Implementation:**
   - **Source Code Submission:** When the code is functional and applies cleanly against the main project, the project team submits the code to a project maintainer over the project mailing list. When the maintainer accepts submitted code, it will then be integrated into development tree.
   - **Continuous Testing and Integration:** The integrated code is tested further in the context of the complete project to ensure it works in conjunction with other features and components.
   - **Source Code Release:** After successful integration and testing, the new feature becomes part of a software release. This helps ensure that users can retrieve the most recent stable release, while developers can work from the most current code.



5. Describe any three characteristics of open-source development model
   - **An interwoven development cycle**
   - **Release Early and Often**
   - **Peer Review**
6. What is the open-source software? Describe two advantages and three limitations of adopting open-source software?
   - **Open-source software** refers to software that is distributed with its source code openly available to users. It is licensed in a way that allows users to view, modify, and distribute the software freely.

   Advantages of adopting open-source software:
   - **Transparency and Flexibility:** Open-source software provides transparency as the source code is accessible. Users can examine and modify the code to meet their specific requirements, making it flexible and customizable to their needs.

- **Collaboration and Community Support:** Open-source projects foster collaboration among developers and users. The active community surrounding open-source software offers support, knowledge sharing, and collective problem-solving.

Limitations of adopting open-source software:

- **Support and Accountability:** it may not always be as comprehensive or readily available as dedicated commercial support for proprietary software. Users relying solely on community support might face challenges in obtaining timely assistance or specific expertise.
- **Integration and Compatibility:** Open-source software can vary in terms of compatibility and integration with other software and systems. It may require additional effort to ensure seamless integration with existing infrastructure or third-party tools.
- **Documentation and User-Friendliness:** Open-source software might have varying levels of documentation and user-friendly interfaces compared to commercially developed software. Some projects may lack extensive documentation or intuitive interfaces, requiring users to invest additional time and effort in understanding and effectively using the software.

7. Open source ecosystem is made up of different stakeholders who contribute in one way or another to open source software. Describe four kinds of stakeholder in open-source software
   - **Developers:** They contribute their time, expertise, and effort to creating, enhancing, and maintaining open source software. Developers may be volunteers or employed by organizations that support open source projects. They contribute code, documentation, testing, bug fixes, and other technical contributions.
   - **Users:** Users of open source software are crucial stakeholders as they provide feedback, bug reports, and feature requests.
   - **Organizations:** Some organizations develop and release open source software as part of their business strategies, while others adopt and utilize open source software in their operations.
   - **Community and Advocacy Groups:** Community and advocacy groups form an essential part of the open source ecosystem. They include user groups, forums, online communities, and nonprofit organizations dedicated to promoting open source software.
8. Is program open source simply because it is written in open-source reference implementation?
   - **No**, a program is not automatically considered open source just because it is written in an open-source reference implementation. While using an open-source reference implementation can provide a starting point or a framework for developing software, the open-source nature of a program depends on the licensing of the code and the permissions granted to users.

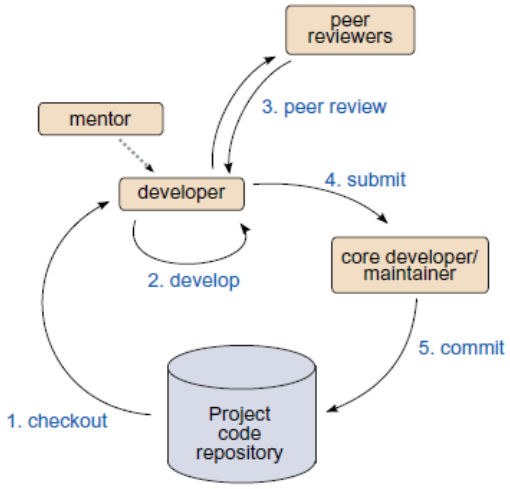
- To determine if a program is open source, it is necessary to examine the licensing terms applied to the program's code, regardless of whether it was based on an open-source reference implementation or not.

9. Differentiate between open-source development method and agile development methods
   - **Open-Source Development Method:** is a collaborative approach to software development where the source code is made available to the public. It emphasizes source code availability, transparency, community involvement, and the ability to modify and redistribute the software.
   - **Agile Development Methods:** refers to a set of iterative and incremental software development methodologies that prioritize flexibility, adaptability, and customer collaboration. It also has iterative and incremental development.
   - ➢ **In short:** Open-source development primarily revolves around the openness of source code and community collaboration, while agile development focuses on iterative delivery, customer collaboration, and adaptability to changing requirements.
10. Free software vs freeware vs open-source software
    - **Free software:** refers to software that grants users the freedom to run, study, modify, and distribute the software and its source code. The "free" in free software refers to freedom, not necessarily price. Free software is typically distributed under licenses that ensure the preservation of these freedoms, such as the GNU General Public License (GPL).
    - **Freeware:** refers to software that is made available for use at no cost. It is often distributed as proprietary software, meaning the source code is not accessible or modifiable by users. Freeware does not necessarily grant users the freedom to modify or distribute the software. Users may have limited rights and restrictions on how they can use or distribute the software.
    - **Open-source software:** refers to software that provides users with the freedom to view, modify, and distribute the source code. It is typically developed in a collaborative manner with a community of contributors. Open-source software emphasizes transparency, community participation, and the availability of the source code for inspection and modification.
    - Open-source software can be distributed under a variety of licenses, including permissive licenses like the MIT License or copyleft licenses like the GPL.
    - ➢ In summary, **free software** emphasizes freedom and the ability to modify and distribute software, while **freeware** refers to software that is available at no cost but may have limitations on usage and distribution. **Open-source software** combines the principles of freedom and open collaboration, allowing users to access, modify, and distribute the source code while fostering community involvement.
11. All free source software qualifies as open-source software, but not all open-source software can be free.
    - ➢ Some open source may have restrictions on the use or distribution of the software, or may require payment for certain types of usage or support.

12. Free software does not mean non-commercial.
    - ➢ Free in free software means freedom to users not price. Freedom without being constrained by proprietary restrictions
13. Differentiate between copyright and license? What is the relationship between copyright and license in open-source software?
    - **Copyright:** Copyright is a legal protection granted to the original creators of creative works, including software. It gives the author exclusive rights to reproduce, distribute, display, and modify their work. Copyright is automatically granted upon the creation of the work and provides a legal framework to prevent unauthorized copying or use of the work by others.
    - **License:** A license, in the context of software, is a legal instrument that grants permissions and sets conditions for the use, modification, and distribution of copyrighted software. It is a legal agreement between the copyright holder (licensor) and the user (licensee). The license specifies the rights and obligations of both parties, detailing what can and cannot be done with the software.
    - ➢ Relationship between copyright and license in open-source software: In the case of open-source software, the copyright holder retains the copyright of the software but grants permissions to users through an open-source license. The license defines the terms under which the software can be used, modified, and distributed. It ensures that the software remains open and accessible, allowing users to view, modify, and redistribute the source code. The license complements the copyright by enabling the software to be shared and developed collaboratively within the framework of open-source principles.
14. Explain two cons of modular development method?
    - Definition: The modular development method involves breaking down a software system into smaller, self-contained modules that can be developed independently and then integrated together.

    Cons:
    - **Complexity of integration:** One challenge with modular development is the complexity of integrating the different modules. As modules are developed independently, ensuring smooth integration and compatibility between modules can be a non-trivial task.
    - **Overhead of communication and coordination:** With modular development, communication and coordination become crucial among the development team. Since each module can be developed by different developers or teams, effective communication channels and collaboration practices need to be established. This can introduce overhead in terms of coordination efforts and increased communication requirements.

15. With aid of diagram describe how new feature code are integrated in the open source project



16. Give a short description of release categories in open-source software projects
    - **pre-alpha (not feature complete),**
    - **alpha (for testing purposes),**
    - **beta (further testing before release), and**
    - **release-candidate (ready to release unless fatal bugs emerge)**
17. With examples, describe any three software tools which are utilized in open-source software development.
    - **Version Control Systems (VCS):** Version control systems are crucial for managing source code changes and facilitating collaboration among developers. Git, a distributed version control system, is widely used in open-source projects. It allows developers to track changes, merge code from multiple contributors, and maintain a complete history of revisions.
    - **Issue Tracking Systems:** Issue tracking systems help in managing and tracking software issues, bugs, feature requests, and other tasks. Projects commonly use tools like Jira, GitHub Issues, or Bugzilla to centralize issue reporting, assign tasks, track progress, and facilitate communication among contributors.
    - **Continuous Integration (CI) Tools:** CI tools automate the process of building, testing, and deploying software. Popular CI platforms such as Jenkins, Travis CI, or CircleCI are frequently used in open-source development. These tools integrate with version control systems, run automated tests, and provide feedback on the code quality, ensuring that new changes don't introduce regressions.

18. Giving two examples of usage, differentiate between lesser general public license and general public license

- Usage Example: **Library vs. Application** The LGPL is often used for software libraries or components that are intended to be linked or used by other programs. It allows developers to use the library in proprietary or closed-source applications without requiring the entire application to be open source. For example, if a developer creates a software library for image processing and releases it under the LGPL, other developers can use that library in their commercial applications without having to open source their entire application.
    - On the other hand, the GPL is commonly used for complete applications or software projects. If a developer chooses the GPL license for their application, it means that the entire application, including its source code, must be made available under the GPL. This ensures that any derivative works or modifications made to the application are also open source. For example, if a developer creates a video editing application and licenses it under the GPL, anyone who distributes or modifies that application must also provide the source code under the GPL.
- **Copyleft and License Compatibility** Both the LGPL and GPL are copyleft licenses, meaning they require derivative works to be distributed under the same license. However, there are differences in terms of license compatibility and linking with other software.
    - The LGPL allows for more flexibility when it comes to linking with proprietary software. If a library is licensed under the LGPL, it can be linked with proprietary applications without forcing the proprietary application to adopt the LGPL license. This is known as "LGPL linking exception." For example, a developer can create an LGPL-licensed database library that can be linked with a closed-source application, allowing the application to remain proprietary while still utilizing the library.
    - In contrast, the GPL has strict requirements for license compatibility. If a program is licensed under the GPL, any software that links to it or includes it as a component must also be distributed under the GPL. This ensures that all the code in the project remains open source. The GPL does not have a linking exception like the LGPL.

19. Describe five categories that distinguish open-source software licenses to other software licenses:
    - **Compatibility with proprietary license**
    - **Compatibility with other types of free licenses**
    - **Enforcement of crediting**
    - **Protection of trademark**
    - **Patent snapback**
20. Describe forking as applied to open-source projects?
    - **Forking refers to the process of creating a new project or software branch that diverges from the original project. It occurs when a group or individual takes a copy of the source code of an open-source project and starts developing it independently, resulting in two separate and distinct projects.**

21. Describe four challenges which may lead to forking
    - **Governance and Leadership Disputes:** Differences in opinions and conflicts over project governance and leadership can create tensions within the project community. Disagreements regarding decision-making processes, project direction, or the influence of individual contributors may result in a faction of the community deciding to fork the project to pursue an alternative vision or approach.
    - **Licensing and Legal Issues:** Disputes or concerns related to licensing and legal aspects of the project can lead to forking. If there are conflicts or disagreements over the license choice or compliance issues, it may result in a group deciding to fork the project to address these concerns or explore different licensing options.
    - **Technical Direction and Design Decisions:** Disagreements over technical direction, design choices, or implementation strategies can also lead to forking. When different factions within the community have conflicting views on how the project should evolve, and consensus cannot be reached, individuals or groups may choose to fork the project to pursue their preferred technical approach or vision.
    - **Community Fragmentation and Communication Issues:** Community fragmentation and breakdowns in communication can contribute to forking. If the project community becomes divided or communication channels become ineffective, it can lead to a lack of collaboration, trust, and shared vision.
22. Describe testing process in open-source projects, describe two challenges which arise in open-source software testing process?

    Testing process: Informal, Peer reviewing, Communication and user participation, Strong community involvement after release
    - **Diverse Testing Environments:** Open-source projects often have a wide range of users and deployment environments. Ensuring comprehensive testing across different operating systems, platforms, and configurations can be challenging. Testers need to consider the diverse environments and ensure that the software functions correctly and reliably in each scenario.
    - **Limited Testing Resources:** Open-source projects typically rely on volunteer contributions and may have limited resources dedicated to testing. This can pose challenges in terms of the availability of skilled testers, access to various hardware and software configurations for testing, and time constraints. The lack of dedicated testing resources can result in inadequate test coverage and potential issues going unnoticed.
    - **Coordination and Communication:** Open-source projects often involve a distributed and diverse community of contributors, including testers. Coordinating testing efforts, ensuring effective communication, and maintaining consistency in testing methodologies can be challenging. The asynchronous nature of open-source development and the reliance on remote collaboration platforms can make it difficult to synchronize testing activities and share timely feedback.
    - **Rapid Development Pace:** Open-source projects often have an agile and fast-paced development environment, with frequent releases and updates. This can pose challenges for testing, as testers need to keep up with the rapid development pace and

ensure that comprehensive testing is conducted within limited timeframes. The need for quick feedback and iteration cycles can add pressure to the testing process.

23. Differentiate between backbox and white box reuse, give out two pros and cos of each

- **Black box reuse** refers to reusing software components or modules without any knowledge of their internal implementation details. Use as it is with minor modification.

  Pros:
  - **Zero purchasing and royalty cost**
  - **Lower risk of vendor lock-in**
  - **Ability to migrate to other more intrusive reuse model**
  - **Encapsulation**
  - **Independence**

  Cons:
  - **Limited Customization**
  - **Limited Understanding**

- **White box reuse** refers to reusing software components or modules while having access to their internal implementation details.

  Pros:
  - **Customization:** With access to the internal implementation, white box reuse allows for greater customization and fine-tuning of the component to suit specific requirements.
  - **Debugging and Optimization:** White box reuse facilitates easier debugging and optimization since developers have a deep understanding of the internal workings of the reused component.

  Cons:
  - **Dependency on Implementation Details:** White box reuse tightly couples the consumer with the internal implementation details of the component. This can create dependencies and increase the risk of breaking changes when the component is updated or modified.
  - **Increased Complexity:** With access to the internal implementation, developers may be tempted to rely on undocumented or unstable features, leading to increased complexity and potential maintenance challenges.
  - **Higher risk**

24. How can open-source project diminish problematic code?
    - **Establishing a standard for code submissions**
    - **Accepting of a common license, and**
    - **Implementing peer review**
    - **Testing and Quality Assurance**
    - **Documentation and Guidelines**
    - **Community Engagement**
    - **Regular Maintenance and Updates**

25. What can be patent in open source software and why patent is important issue in open source software development
What can be patented:

- **Software Algorithms:** Patents can be obtained for innovative algorithms or methods that provide a novel solution to a technical problem. This includes unique algorithms for data processing, machine learning models, cryptographic techniques, etc.
- **Hardware-Software Integration:** If the open-source software involves integration with hardware components, patents can be obtained for the hardware aspects, such as innovative electronic circuits, chip designs, or hardware architectures.
- **User Interfaces:** Unique and inventive user interfaces, graphical designs, or user interaction techniques can potentially be patented, ensuring protection for the user experience innovations.

Patents are an important issue in open-source software development due to the following reasons:

- **Protection of Innovations:** Patents provide legal protection for innovative ideas and inventions. By obtaining patents, open-source developers can safeguard their novel contributions and prevent others from using, distributing, or selling their inventions without permission.
- **Collaboration and Licensing:** Patents can play a role in licensing agreements within open source projects. Developers can choose to license their patented technologies under open source licenses or use patent licenses to establish specific terms and conditions for the use of their inventions by others.
- **Encouraging Commercial Adoption:** Patents can incentivize commercial adoption of open source software. Companies may be more willing to invest in and contribute to open source projects that have patented technologies, as it provides them with a level of exclusivity and potential commercial advantage.
- **Avoiding Patent Litigation:** By obtaining patents, open source developers can proactively protect their software from potential patent infringement claims. It allows them to assert their own patent rights and potentially deter others from asserting patents against their open source projects.

26. Describe four way which are used by open-source project to generate revenue

- **Value Added Packaging**
- **Services and Support**
- **Dual Licensing**
- **Brand Licensing**
- **Accessorizing**
- **Crowdfunding and Donations**
- **Partnerships and Sponsorships**

27. Can someone write proprietary codes that link to a shared library that is open source? Explain
    - **Yes**, it is possible for someone to write proprietary code that links to a shared library that is open source. The open-source license of the library governs its terms of use and distribution.
    - When someone writes proprietary code that links to an open-source shared library, the licensing of the two components may coexist. The key consideration is ensuring that the licensing terms of the open-source library are compatible with the proprietary code's intended use.
    - Open-source licenses vary in their requirements and restrictions. Some open-source licenses, such as the GNU General Public License (GPL), impose certain obligations on software that uses or links to GPL-licensed code. In the case of the GPL, if the proprietary code is distributed together with the open-source library or is derived from the library, it may need to be made available under the GPL or a compatible license.
    - However, other open-source licenses, such as the Apache License or the MIT License, may not impose such restrictions. They typically allow for greater flexibility, enabling the use of the open-source library in proprietary code without imposing reciprocal licensing requirements on the proprietary code.
28. Differentiate between centralized version control system and distribution version control system

    Centralized Version Control System (CVCS):
    - In a centralized version control system, there is a single, central repository that stores the entire history and versions of the project's source code.
    - Developers interact with the central repository by checking out files to make changes and checking them back in when completed.
    - The central repository acts as the authoritative source, and developers must have access to it to perform most version control operations.

    Examples of CVCS include Apache Subversion (SVN) and Team Foundation Version Control (TFVC).

    Key characteristics of CVCS include:
    - Single, central repository
    - Requires network connectivity to access the central repository
    - Generally, relies on a client-server architecture
    - Synchronization of code changes occurs between the central repository and local working copies

    Distributed Version Control System (DVCS):
    - In a distributed version control system, each developer maintains a complete local copy of the repository, including the full history and version information.
    - Developers can perform version control operations locally, including committing changes, creating branches, and merging branches, without requiring constant network connectivity.

- Multiple repositories can exist, and changes can be synchronized between them as needed.

Examples of DVCS include Git and Mercurial.
Key characteristics of DVCS include:

- Each developer has a full local copy of the repository
- Enables offline work and independent branching and merging
- Supports multiple remote repositories and decentralized collaboration
- Changes can be synchronized between repositories through push and pull operations

29. Differentiate public domain software from OSS
    - Public Domain Software are not under the copyright act. They are free programs & can be used w/o restrictions. It means the user of the software can copy, distribute & even modify the software.
30. Open-source software refers to computer software that is made available with its source code, allowing anyone to view, modify, and distribute the software. Key characteristics of open-source software include:
    - **Source code availability:** Open-source software provides access to its underlying source code, allowing users to study, modify, and enhance it.
    - **Free redistribution:** Open-source software can be freely shared, distributed, and used by anyone, without any licensing fees or restrictions.
    - **Modification and customization:** Users have the freedom to modify the source code to suit their specific needs, enabling them to customize the software and contribute improvements to the community.
    - **Collaboration and community-driven development:** Open-source projects often have a community of developers who contribute to the software's development, sharing their expertise, knowledge, and enhancements.
    - **Transparency and openness:** Open-source software promotes transparency, ensuring that the development process and decision-making are accessible to the community. This allows for peer review and helps build trust and accountability.
    - **Licensing:** Open-source software is typically distributed under licenses approved by the Open-Source Initiative (OSI), such as the GNU General Public License (GPL), Apache License, or MIT License.

31. Difference between The GNU Lesser General Public License (LGPL) and the GNU General Public License (GPL)

Scope of Application:

- GPL: The GPL is a copyleft license that aims to ensure the software remains free and open source. It applies to both the original software and any derivative works based on it.
- LGPL: The LGPL is also a copyleft license but with a more permissive scope. It applies to the original software as well as any changes made to the licensed software itself, but it allows for greater flexibility when used as a library.

Licensing of Derivative Works:

- GPL: If you create a derivative work based on GPL-licensed code, the derivative work must also be licensed under the GPL. This includes software that dynamically links or statically links to GPL-licensed code.
- LGPL: When using LGPL-licensed code, you have the option to dynamically link to it without the requirement of your application being licensed under the LGPL. This allows for proprietary applications to use LGPL-licensed libraries without having to release their own source code.

Use as a Library:

- GPL: If you use GPL-licensed code as a library in your application, the entire application must be licensed under the GPL.
- LGPL: The LGPL allows for greater flexibility when using LGPL-licensed code as a library. You can use the LGPL-licensed library in a proprietary application without being required to release the source code of the application.

Code Distribution:

- GPL: If you distribute software under the GPL, you must make the corresponding source code available to recipients and grant them the same rights to modify and distribute the code.
- LGPL: The LGPL has similar requirements for distributing modifications to the LGPL-licensed code itself, but it allows for more flexibility when it comes to distributing the application that uses the LGPL-licensed library.

32. Difference between Permissive and Copyleft
    - **Permissive Licenses:** Permissive open-source licenses, such as the MIT License and the Apache License, provide maximum freedom to users. They allow users to modify, distribute, and use the software, including incorporating it into proprietary projects, without imposing significant restrictions on how the software is licensed or distributed.
    - **Copyleft Licenses:** Copyleft licenses, such as the GNU General Public License (GPL), are designed to ensure that derivative works and modifications of the software remain open source. They require that any modifications or derivative works based on the original software must be licensed under the same terms as the original, thus preserving the openness of the code.
33. How can developer monetize their contribution?
    - **Bug Hunting programs**
    - **Provide Premium Support**
    - **Sell Associated content**
    - **Double licensing**
    - **SaaS**

34. Factors to consider when deciding to reuse OSS within an organization:
    - **Calculate cost of reusing the code**
    - **Complexity and sheer number of different oss licenses**
    - **Adoption and reuse procedures are not standardized**
35. The process of reusing a piece of OSS within a proprietary software product consists of the following four stages:
    - **Decision**
    - **Selection**
    - **Integration**
    - **Assessment.**
36. The following should be considered when deciding whether to open source a product.
    - **Evaluate the Market for the Target Product (Market interest)**
    - **Determine Development Community Interest (How likely it is for a community of developers to form around the product)**
    - **Decide what Parts of Product to Open Source (Trade secrets or algorithms that are better not publicized)**
    - **Select Appropriate License**
    - **Sanitize Code**
37. Conflicts resolution:
    - **Rules and Institutions**
    - **Monitoring and Reputation**
38. Cooperation challenges:
    - **Free Riding**
    - **Heterogeneity in opinions and aims and risk of forking**
    - **Geographical distributed development**
    - **Modularized code production**
    - **Handling code complexity**
39. Governance process in OSS:
    - **Management of membership**
    - **Allocation of tasks**
    - **Decision making**
40. OSS Community challenges:
    - **More contribution more risk**
    - **Mitigate risk of problematic code**
    - **More contributors mean less progress**
41. OSS Development Model Visible Phases
    - **Problem discovery**
    - **Finding volunteers**
    - **Solution identification**
    - **Code development and testing**
    - **Code change review**
    - **Code commit and documentation**
    - **Release management**