EXAMPLES THAT MIGHT BE ON EXAM
Twitch - User is able to livestream
1-Preconditions
2-Actors
3-Instructions/Regular Flow
4-Alternative Flows
5-Postconditions
Preconditions
-Internet Availability
-An Account (created/logged in)
-Stream Key
-Streaming Software (on appropriate device)
Actors
Primary - Streamer
Secondary - Viewers/Twitch Users
Instructions
1. Provide stream key to your streaming software
2. Set up your streaming software "scene"
3. Click "Start Streaming" or equivalent in software
4. Stream is started
* 5. Update Meta information on twitch (game played/title/etc)
Alternative Flows
3a. Incorrect stream key provided/changed
3a1. Stream does not get "forwarded" to the appropriate channel/website
3b. Incorrect encoding/"old" handling of stream key/etc
* 3b1. Stream dashboard indicates invalid attempt to stream, provides details
4c. Connection lost/reconnect
4c1. Connection has been lost, timer is started
4c2. If user reconnects with same settings/key before timer elapses, stream resumes without ending
4c2a. User does not reconnect, and the stream ends, with the user able to stream again (maybe after a timer?)
Postconditions
-Their channel will be indicated as "live", with a link to the current livestream
-Followers will be notified that a new livestream has begun
-The server will begin recording the VoD
-Available for browsing under the provided game/activity
-Live viewer count is shown/made Available
-"Live Time" counter begins/is shown
The actors would be the viewer, streamer, the payment manager, and the livestream platform.
Preconditions: The streamer must be live and have a donation option available to viewers.
The viewer must be logged into their account associating with the stream.
The viewer must also use a valid payment method accepted by the streamer.
Sequence of required actions for the use case: The viewer selects the donation option.
The viewer types a message and inputs a donation amount to send.
The platform confirms and sends the donation request.
The payment manager validates if the payment information is valid or not.
Assuming the payment is successful, the message displays and reads aloud.
The Streamer sees the message and acknowledges the donation from the viewer.
Alternative flows: There could be technical difficulties with the platform and the message does not get read or sent.
Another situation would be a payment failure and the payment manager addresses the issue to the viewer to try again with a different payment option.
The Streamer might also have missed reading the message, so the platform might offer an option for the viewer to get a refund.
There might also be an option for the viewer to get their message viewed again by the streamer.
Postconditions: The streamer receives the donation. The message gets displayed and read aloud by either the streamer or a talk-to-speech option if available. The streamer acknowledges the donation from the viewer.
Sure! Let's consider a scenario where a company wants to develop a mobile application for task management. Here's a description of the software requirements:
Title: Taskify - Mobile Task Management Application
Objective:
Taskify aims to provide users with a convenient and intuitive mobile platform to manage their tasks efficiently. The application should help users organize, prioritize, and track their tasks on-the-go, enhancing productivity and time management.
Requirement examples
Functional Requirements:

1. User Authentication:
   - Users should be able to create an account or log in using existing credentials (email/password or social media accounts).
   - Password recovery/reset functionality should be available.
2. Task Management:
   - Users can create, edit, and delete tasks.
   - Each task should have a title, description, due date, priority level, and status (e.g., pending, in progress, completed).
   - Tasks can be categorized into different lists or projects.
3. Task Organization:
   - Users can organize tasks by due date, priority, or project.
   - Ability to set reminders and notifications for upcoming tasks.
   - Support for attaching files or notes to tasks.
4. Collaboration:
   - Users can share tasks or task lists with other users.
   - Collaborators should be able to view, edit (if permitted), and comment on shared tasks.
5. User Interface:
   - Intuitive and user-friendly interface with easy navigation.
   - Support for customization (e.g., themes, font sizes).
6. Offline Functionality:
   - Users should be able to access and modify tasks even when offline, with changes synced once online connectivity is restored.
7. Integration:
   - Integration with calendar applications to sync task due dates.
   - Ability to import tasks from other task management applications (e.g., Todoist, Trello).
8. Security:
   - Data encryption for sensitive information (e.g., passwords, task details).
   - Secure connection (HTTPS) for data transmission.
   - Regular data backups to prevent data loss.

Non-Functional Requirements:

1. Performance:
   - Fast response times, even with a large number of tasks.
   - Minimal loading times for various app functions.
2. Scalability:
   - The application should be scalable to accommodate a growing user base and increasing data volume.
3. Reliability:
   - High availability with minimal downtime.
   - Automated error detection and reporting.
4. Compatibility:
   - Support for both iOS and Android platforms.
   - Compatibility with a wide range of mobile devices and screen sizes.
5. Accessibility:
   - Compliance with accessibility standards (e.g., WCAG) to ensure usability for users with disabilities.
6. Localization:
   - Support for multiple languages and regional preferences.
7. Data Privacy:
   - Compliance with data protection regulations (e.g., GDPR).
   - Explicit user consent for data collection and processing.
8. Documentation:
   - Comprehensive user documentation and help resources.
   - Developer documentation for future maintenance and updates.

Definitions of models

9. Waterfall Model: This is a linear and sequential approach to software development. It progresses through distinct phases such as requirements gathering, design, implementation, testing, deployment, and maintenance. Each phase must be completed before moving on to the next one.
10. V-Model: This model is an extension of the waterfall model. It emphasizes the relationship between each phase of development and its corresponding testing phase. The left side of the "V" represents the development phases, while the right side represents the testing phases. It ensures that testing is integrated throughout the development process.
11. Incremental Model: In this model, the development process is divided into small increments or iterations. Each iteration produces a partial but working product. With each increment, new features are added or existing ones are enhanced, leading to the gradual development of the final product.
12. Evolutionary Model: Also known as Prototyping Model, it involves building a basic version of the software quickly, known as a prototype, which is then refined through iterations based on feedback. It's useful when requirements are not well understood initially or when stakeholders need to visualize the product early in the development process.
13. Agile Models: Agile methodologies are a group of software development approaches based on iterative and incremental development. Agile emphasizes flexibility, customer collaboration, and rapid response to change. Examples include Scrum and Extreme Programming.
14. Scrum: Scrum is an agile framework that divides the development process into short iterations called sprints. It involves cross-functional teams working collaboratively to deliver potentially shippable increments of the product at the end of each sprint. Scrum roles include Product Owner, Scrum Master, and Development Team.
15. Extreme Programming (XP): XP is another agile methodology focused on improving software quality and responsiveness to changing customer requirements. It emphasizes practices such as continuous integration, test-driven development, pair programming, and frequent releases.
16. Framework Activities: These are general activities that are applicable across various software development methodologies. They include Communication (within the team and with stakeholders), Planning (for project scope, schedule, and resources), Modeling (creating visual representations of the system), Construction (actual coding and development), and Deployment (delivering the product to users).
17. Umbrella Activities: These activities are often applied across different phases of development and involve collaboration and coordination among team members. Examples include Pair Programming (two programmers working together on the same task), Code Review (peer review of code to identify defects and improve quality), and Testing (ensuring the software meets requirements and functions correctly).
18. Project Management Activities: These activities are essential for managing the overall software development project. They include Quantifying progress and quality (measuring project metrics and assessing the quality of deliverables), Estimation and Scheduling (predicting project timelines and allocating resources), and Risk Management (identifying and mitigating potential risks to the project's success).

1. Requirements: In software engineering, requirements refer to descriptions of the system's functionalities and constraints that the system should satisfy. These requirements can be categorized as functional requirements (what the system should do) and non-functional requirements (qualities or constraints the system must adhere to, like performance, security, etc.). Requirements gathering is the process of collecting, analyzing, documenting, and managing these requirements throughout the software development lifecycle.
2. Use Cases: Use cases describe interactions between an actor (user or external system) and a system to accomplish a specific goal. They provide a way to capture and communicate functional requirements from the user's perspective. Use cases typically consist of a description of the interaction (scenario), preconditions, postconditions, and any alternative paths.
3. Activity Diagrams: Activity diagrams are a type of UML (Unified Modeling Language) diagram used to model the flow of control or the sequence of activities in a system. They are particularly useful for visualizing business processes, workflows, and the logic of complex algorithms. Activity diagrams consist of activities (actions or tasks), control flow (arrows indicating the sequence of activities), and decision points (diamond shapes indicating branching logic).
4. Swimlane Diagrams: Swimlane diagrams, also known as cross-functional flowcharts or deployment flowcharts, visually depict the flow of activities across different organizational units or actors. Each "swimlane" represents a specific role, department, or system component, and activities are mapped onto these lanes to show who is responsible for each task and the sequence of their execution.
5. Use Case Diagrams: Use case diagrams are a type of UML diagram used to visualize the relationships between actors and use cases in a system. They provide a high-level overview of the system's functionality and the interactions between users and the system. Actors are represented as stick figures, and use cases are represented as ovals. Relationships between actors and use cases are depicted using lines.
6. UML Class Diagrams: UML class diagrams are used to model the structure of a system by visualizing classes, their attributes, methods, and relationships. Classes represent

objects in the system, attributes represent the properties of those objects, methods represent the behaviors, and relationships represent how classes are connected to each other (e.g., associations, aggregations, compositions, inheritance).

7. Risk Table: A risk table, also known as a risk matrix, is a tool used in risk management to assess and prioritize risks based on their likelihood and impact. Risks are typically listed in rows, and their likelihood and impact are assessed and assigned numerical values. These values are then used to determine the level of risk (low, medium, high) and prioritize risk mitigation efforts.

EXAMPLES OF RISKS

1. Key team member leaving the project (R1):
    - Mitigation Strategy: Cross-training team members to ensure knowledge sharing and reduce dependency on individual team members.
    - Contingency Plan: Maintain updated documentation and procedures to facilitate the transition if a key member leaves unexpectedly.
    - Preventive Action: Implement retention strategies such as professional development opportunities and recognition programs to reduce the likelihood of team members leaving.
2. Technology stack not meeting requirements (R2):
    - Mitigation Strategy: Conduct thorough technology evaluation and proof of concept before finalizing the technology stack.
    - Contingency Plan: Identify alternative technologies that could meet the project requirements and keep them as backup options.
    - Preventive Action: Involve stakeholders and subject matter experts in the technology selection process to ensure alignment with project goals and requirements.
3. Scope creep (R3):
    - Mitigation Strategy: Regularly review and prioritize project requirements with stakeholders to prevent scope creep.
    - Contingency Plan: Have a change management process in place to evaluate and approve changes to the project scope.
    - Preventive Action: Clearly define project scope and objectives at the outset, and document any changes through formal change requests.
4. Tight project deadline (R4):
    - Mitigation Strategy: Implement Agile methodologies to break down the project into manageable iterations and deliver incremental value.
    - Contingency Plan: Identify critical path activities and allocate additional resources if necessary to meet the deadline.
    - Preventive Action: Conduct thorough project planning and risk assessment to identify potential schedule risks early in the project lifecycle.
5. Security vulnerabilities discovered post-release (R5):
    - Mitigation Strategy: Implement secure coding practices and conduct regular security reviews throughout the development process.
    - Contingency Plan: Have a response plan in place to address security incidents quickly and effectively.
    - Preventive Action: Include security considerations as part of the initial project requirements and conduct thorough security testing before release.
6. Inadequate testing coverage (R6):
    - Mitigation Strategy: Implement a comprehensive testing strategy that includes unit testing, integration testing, and system testing.
    - Contingency Plan: Allocate additional time and resources for testing if necessary to ensure adequate coverage.
    - Preventive Action: Involve QA professionals early in the project lifecycle to define testing requirements and scenarios.

<u>Examples of models and their implementation</u>

7. Scenario: Developing a mobile application for a startup company with rapidly changing requirements and a tight deadline.
    - Selected Model/Framework: Agile (specifically Scrum)
    - Argument for Suitability:
        i. Agile/Scrum is well-suited for this scenario due to its iterative and incremental approach. The project can be broken down into smaller increments called sprints, allowing for frequent releases and adaptation to changing requirements.
        ii. The Scrum framework provides clear roles and responsibilities (Product Owner, Scrum Master, Development Team), facilitating effective communication and collaboration.
        iii. Regular sprint planning sessions ensure that the team focuses on high-priority features and adapts to changing market conditions.
        iv. Umbrella activities like pair programming and code review can be integrated within each sprint to ensure code quality and knowledge sharing.
        v. Testing is an integral part of each sprint, with emphasis on automated testing and continuous integration to maintain product quality.
        vi. Project management activities such as quantifying progress and quality, estimation and scheduling, and risk management are inherent in the Scrum framework, with tools like burndown charts and retrospectives aiding in project monitoring and improvement.
8. Scenario: Developing a safety-critical software system for a nuclear power plant.
    - Selected Model/Framework: V-Model
    - Argument for Suitability:
        i. The V-Model is suitable for safety-critical systems where thorough testing and verification are essential.
        ii. It emphasizes a systematic and disciplined approach, with each stage of development corresponding to a stage of testing.
        iii. Requirements analysis and system design are done upfront, ensuring clear understanding and documentation of system requirements and architecture.
        iv. Each stage of development is followed by corresponding testing activities, such as unit testing, integration testing, and system testing, ensuring that the system meets its requirements and safety standards.
        v. Umbrella activities such as testing, pair programming, and code review are integrated into each stage, with a focus on ensuring reliability and minimizing errors.
        vi. Project management activities like quantifying progress and quality and risk management are crucial for ensuring compliance with safety regulations and standards.
9. Scenario: Developing a large-scale enterprise software system for a multinational corporation with distributed development teams.
    - Selected Model/Framework: Incremental Model
    - Argument for Suitability:
        i. The Incremental Model is suitable for large-scale projects where the requirements are not fully understood upfront and can evolve over time.
        ii. It allows for the delivery of a working product incrementally, providing early value to stakeholders and facilitating feedback-driven development.
        iii. The project can be divided into multiple increments, each delivering a subset of functionality, allowing for parallel development and integration.
        iv. Communication and collaboration are crucial in this scenario, with regular interactions between distributed development teams and stakeholders to ensure alignment and address any issues or changes.
        v. Umbrella activities such as pair programming, code review, and testing are integrated into each increment, ensuring quality and reliability.
        vi. Project management activities like quantifying progress and quality, estimation and scheduling, and risk management are essential for managing the complexity of the project and ensuring timely delivery.