**Software Development Lifecycle (SDLC) Analysis of Netflix**

***A comparative study of different models in relation to Netflix’s software development***

Delston Aaron Pereira

*Nitte Mahalinga Adyantaya Memorial Institute of Technology, Nitte, Karnataka, India*

*delston.aaron@gmail.com, nnm23is044@nmamit.in*

|  |  |
| --- | --- |
| Abstract: | Software development is an evolving discipline requiring structured approaches for building scalable, secure, and efficient systems. This report explores the **Software Development Life Cycle (SDLC)** models applicable to **Netflix**, a global leader in video streaming services. It provides a comparative analysis of SDLC methodologies, an overview of **requirements engineering**, and best practices in **version control management**.  The study aims to offer insights into the selection of an appropriate SDLC model for large-scale cloud-based platform (Netflix for this report), emphasizing **incremental development and spiral model approaches**. The report also discusses challenges and strategies involved in requirements validation and software deployment at Netflix.  The findings in this document are based on extensive research, industry best practices, and insights from Netflix’s technology stack. I hope this report serves as a valuable resource for software engineers, architects, and researchers interested in the intersection of **SDLC and requirements engineering methodologies and large-scale streaming platforms**. This are followed by the conclusion and considerations. |

**Table of Contents**

1. Preface
2. Introduction
3. Overview of Netflix
   * System Overview
   * Technologies Used
4. Comparative Analysis of SDLC Models
   * Waterfall Model
   * Incremental Development Model
   * Spiral Model
   * Summary of Comparison
5. Requirements Engineering for Netflix
   * Functional Requirements
   * Non-Functional Requirements
   * Requirements Validation Strategy
   * Challenges in Requirements Validation
6. Version Control Strategy for Netflix
   * GitHub Repository Strategy
   * Branching & Merging Workflow
7. Conclusion
8. References
9. Diagrams

**1. Introduction**

Netflix is one of the leading video streaming platforms globally, offering on-demand content to millions of users. The platform relies on a robust cloud-based infrastructure, primarily hosted on **Amazon Web Services (AWS)**. Given its large-scale nature, Netflix requires an efficient software development lifecycle (SDLC) to manage continuous updates, new features, and system stability.

This report conducts a comparative study of **Incremental Development, Spiral Model, and Waterfall Model** in relation to Netflix’s software development. It also explores **requirements engineering**, the challenges faced, and an optimal **version control strategy**.

**2. Overview of Netflix**

**2.1 System Overview**

Netflix provides subscription-based access to movies, TV shows, and original content. The system supports multiple devices, personalized recommendations, and high-quality video streaming with adaptive bitrate technology.It is an American subscription service which provides video on-demand over-the-top streaming. The service primarily distributes both original and acquired films and television shows from various genres, and it is available internationally in multiple languages.[[1]](#endnote-1)

**2.2 Technologies Used**

* **Cloud Platform**: AWS (Amazon Web Services)
* **Architecture**: Microservices-based
* **Database**: NoSQL (DynamoDB, Cassandra), MySQL
* **Content Delivery**: AWS CloudFront, Open Connect
* **Programming Languages**: Java, Python, Node.js
* **DevOps**: Continuous Integration & Continuous Deployment (CI/CD)

Netflix does use more technologies in every department. The ones mentioned above are just the major ones, powering the streaming platform in each individual component. Each of these component works seamlessly across the platform to provide users with a reliable stream.

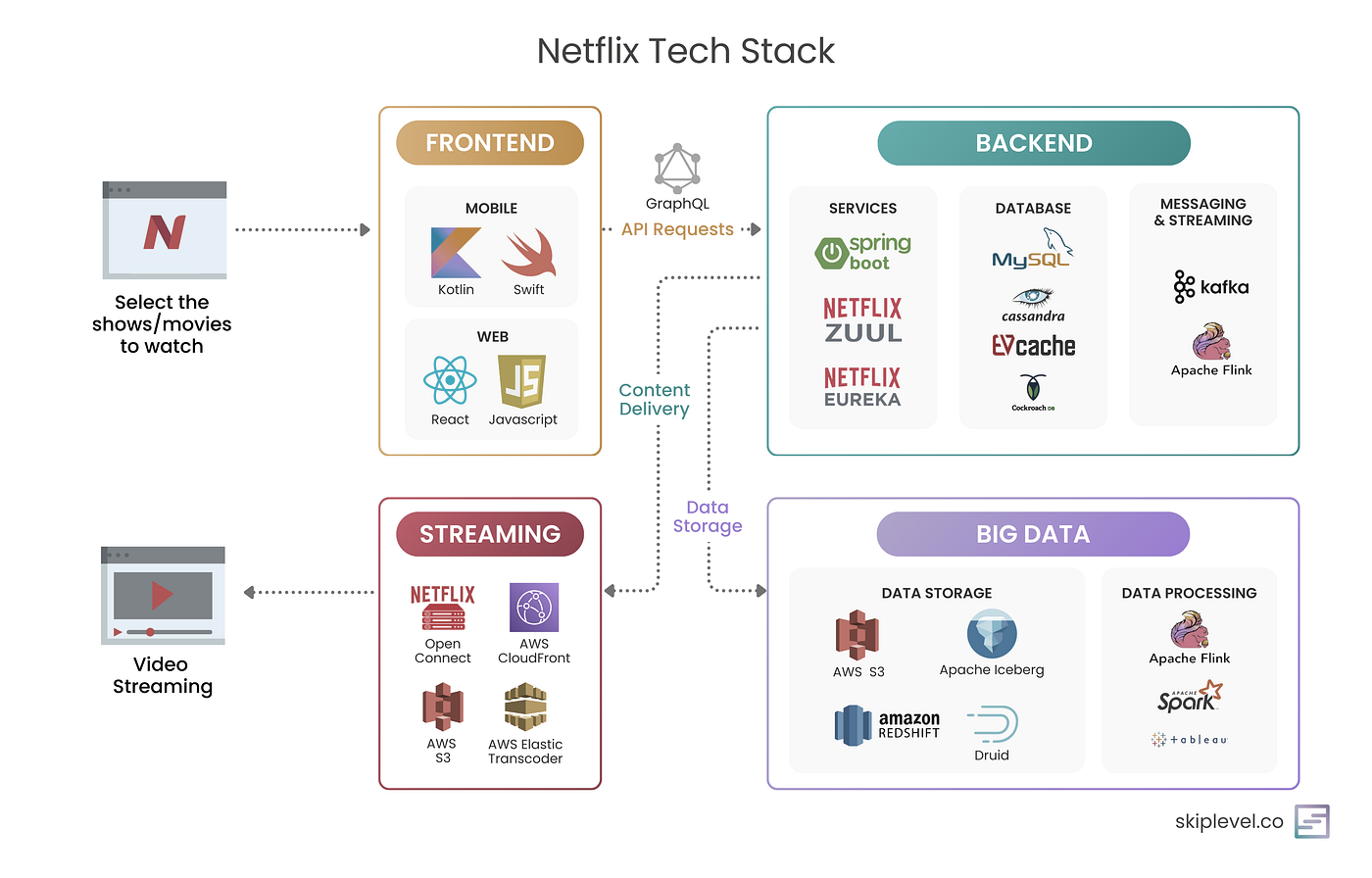


Figure 1- Represents the diverse tech stack[[2]](#endnote-2)

**3. Comparative Analysis of SDLC Models**

**3.1 Waterfall Model**

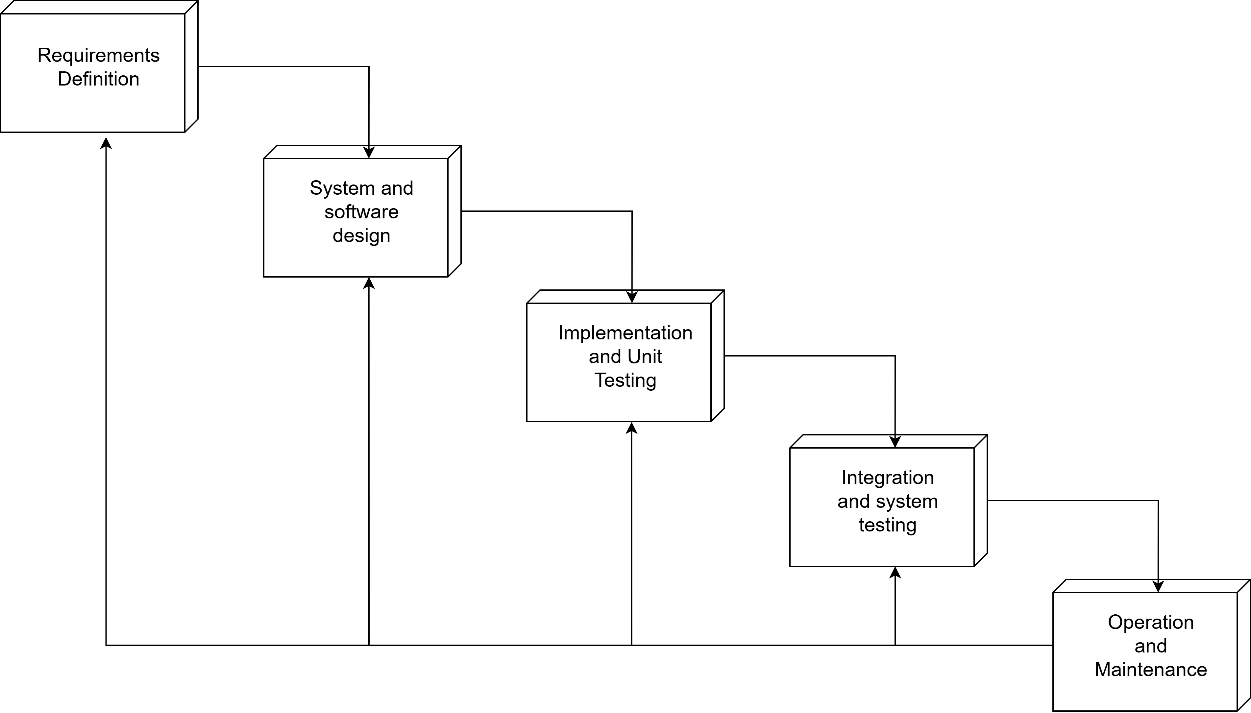
****

Figure 2- Depicts the Waterfall Model

**How Netflix Would Be Developed Using Waterfall:**

* **Phase 1: Requirements Definition** – In this phase, all system requirements are gathered upfront. This includes defining user authentication, content streaming capabilities, personalized recommendations, and offline viewing. Since changes are difficult to implement later, exhaustive documentation is required at this stage.
* **Phase 2: System & Software Design** – A complete architecture is developed, including database structures, API endpoints, and server infrastructure on AWS. Every aspect of the system is carefully mapped out before development begins.
* **Phase 3: Implementation** **& Unit testing**– Developers begin coding the entire system in one go, following the previously defined architecture. No changes to the requirements are permitted, and development follows a linear path.
* **Phase 4: Integration and System Testing** – After development is completed, the entire system undergoes integration and rigorous testing. This includes functional testing, performance testing, and security testing. Since all components are built at once, identifying and fixing bugs can be time-consuming.
* **Phase 5: Operation and Maintenance** – The fully developed streaming platform is deployed to production. This marks the system’s release, and users can now access Netflix. Any bugs or issues discovered post-launch are addressed during the maintenance. However, because new changes require extensive planning and reimplementation, updates take significant time to roll out.

**Suitability for Netflix:**

* **Pros:**
  + Well-documented and structured approach ensures clarity in development.
  + Defined phases simplify project management.
  + Suitable for smaller, well-defined projects with minimal expected changes.
* **Cons:**
  + Lacks flexibility for rapidly changing user needs.
  + Late-stage issue discovery can cause significant delays.
  + Long development cycles make it unsuitable for continuous updates.
* **Verdict:** Not suitable, as Netflix requires frequent updates, rapid feature deployment, and adaptability to evolving user preferences.

**3.2 Incremental Development Model**

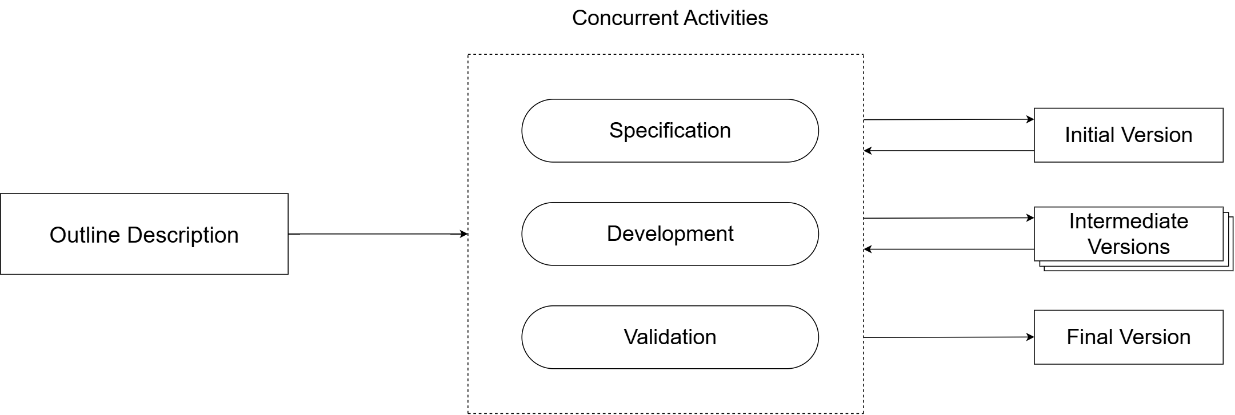
****

Figure 3- represents the Incremental Development Model

**How Netflix Uses Incremental Development:**

* **Phase 1: Planning and Initial Requirements** – The development team identifies high-priority features, such as user authentication, video playback, and recommendation algorithms. Only essential requirements for the first iteration are finalized, leaving room for future updates.
* **Phase 2: Feature-wise Development** – Instead of developing the entire system at once, Netflix continuously releases new features (e.g., improved recommendation algorithms, new content categories, enhanced offline viewing). Each feature undergoes design, development, testing, and deployment in separate cycles.
* **Phase 3: Continuous User Feedback Integration** – Each increment is deployed to users who provide feedback through usage patterns and explicit ratings. Developers analyse this feedback and make necessary adjustments before launching the next increment.
* **Phase 4: System Testing and Refinement** – Every new feature or update undergoes extensive testing, including performance analysis, bug fixes, and security assessments. Automated pipelines ensure continuous integration and testing.
* **Phase 5: Deployment and Monitoring** – Features are released to production in batches, ensuring minimal downtime. A/B testing is often used to compare different versions of a feature before full-scale deployment.
* **Phase 6: Iteration and Maintenance** – The cycle repeats as developers refine existing features, resolve user-reported issues, and introduce new functionalities based on emerging trends.

**Suitability for Netflix:**

* **Pros:**
  + Faster time-to-market with incremental feature releases.
  + Allows for quick adaptation to changing user needs.
  + Continuous testing ensures high reliability and performance.
  + Easily scalable and supports cloud-based microservices architecture.
* **Cons:**
  + Requires effective integration strategies to avoid system conflicts.
  + Needs strong version control mechanisms to manage multiple development branches.
  + Can become complex if too many features are being worked on simultaneously.
* **Verdict:** Highly suitable for a large-scale, evolving platform like Netflix, as it supports continuous updates and user-driven enhancements.

**3.3 Spiral Model**

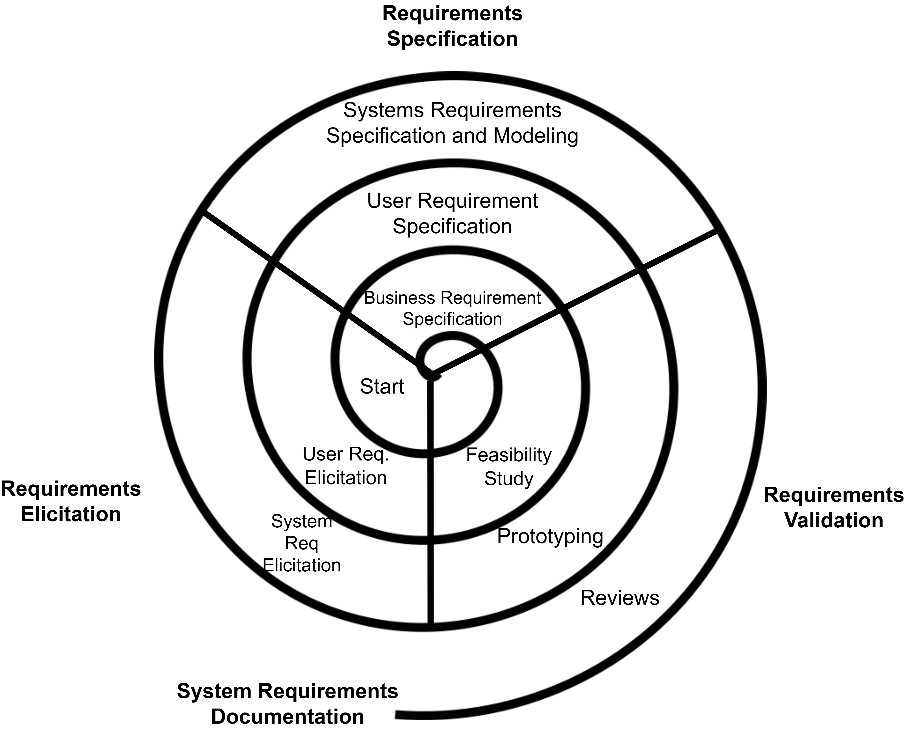
****

Figure 4- Represents the Spiral Model

**How Netflix Would Be Developed Using Spiral:**

* **Phase 1: Risk Analysis and Prototyping** – Before implementing a major feature (e.g., AI-powered recommendations), Netflix evaluates potential risks such as algorithm biases, data security concerns, and performance overhead. Small-scale prototypes are developed to test feasibility before committing to full-scale development.
* **Phase 2: Concept Validation and Refinement** – After testing the prototype, Netflix collects initial feedback and refines the feature’s design. This iterative process ensures the feature aligns with technical constraints and business goals.
* **Phase 3: Iterative Development and Testing** – Features are built and tested in cycles. Developers follow an iterative approach, incorporating changes based on risk assessment and early feedback before the feature reaches full-scale deployment.
* **Phase 4: Extensive Validation and Security Testing** – Since each iteration undergoes rigorous risk assessment, Netflix ensures new features meet strict security and performance standards before a full-scale rollout. This is particularly important for features involving **machine learning models, personalization, and data privacy regulations**.
* **Phase 5: Gradual Deployment and Performance Monitoring** – The developed feature is initially deployed to a limited user base. Real-world performance is monitored, and adjustments are made before scaling the deployment to a larger audience.
* **Phase 6: Refinement and Continuous Improvement** – Developers use real-time analytics and user insights to refine the feature further. The cycle repeats for future improvements, ensuring ongoing innovation while minimizing risks.

**Suitability for Netflix:**

* **Pros:**
  + Strong risk management ensures robust system development.
  + Iterative improvements help refine complex features like AI-driven personalization.
  + Suitable for large-scale, high-risk functionalities such as **security upgrades and algorithm-driven recommendations**.
* **Cons:**
  + Expensive and time-consuming for simple features.
  + Requires highly skilled teams to assess risks accurately.
  + Can be inefficient for routine updates that do not involve high-risk factors.
* **Verdict:** Suitable for **high-risk features** (e.g., security updates, AI-based recommendations, cloud infrastructure upgrades), but may not be necessary for routine feature development.

**3.4 Summary of Comparison**

| **SDLC Model** | **Flexibility** | **Risk Management** | **Time-to-Market** | **Cost** | **Suitability for Netflix** |
| --- | --- | --- | --- | --- | --- |
| Waterfall | Low | Low | Slow | Medium | Not Suitable |
| Incremental | High | Medium | Fast | High | Highly Suitable |
| Spiral | High | High | Moderate | High | Suitable for High-Risk Features |

**4. Requirements Engineering for Netflix**

Functional requirements define the core capabilities of Netflix’s system. These include:

* **User Authentication & Account Management**: Secure login, account creation, multi-profile support.
* **Content Streaming & Playback**: Adaptive bitrate streaming, offline downloads, and multi-device support.
* **Recommendation System**: AI-based content recommendations based on viewing history and preferences.
* **Search & Navigation**: Efficient search filters and personalized UI/UX.
* **Payment & Subscription Management**: Secure payment gateway integration with multiple subscription plans.
* **Content Management System**: Backend for managing and updating content metadata, genres, and subtitles.
* **Parental Controls**: Age-restricted content management and customizable parental controls.
* **User Analytics & Insights**: Collection of user engagement data for personalization and marketing strategies.

**4.2 Non-Functional Requirements**

Netflix’s non-functional requirements ensure high availability, security, and performance. Key aspects include:

* **Scalability**: The system must handle millions of concurrent users without performance degradation.
* **Availability**: 99.99% uptime is essential for a global audience.
* **Security**: End-to-end encryption, secure user authentication, and fraud detection.
* **Performance**: Low latency for streaming and quick content buffering.
* **Compliance & Legal Considerations**: Adherence to GDPR, DMCA, and copyright laws.
* **Maintainability & Upgradability**: Support for continuous deployment and infrastructure upgrades.

**4.3 Requirements Validation Strategy**

Netflix employs various techniques to validate software requirements:

* **Stakeholder Reviews**: Collaboration with business teams, developers, and end-users to refine requirements.
* **Prototyping & A/B Testing**: Early-stage UI/UX testing and data-driven decisions based on user behavior.
* **Automated Testing & CI/CD Pipelines**: Integration of unit, functional, and regression testing to ensure compliance.
* **Security Audits**: Continuous vulnerability assessments and penetration testing to meet security standards.
* **User Feedback Loops**: Collection of real-time analytics to adapt to evolving customer needs.

**4.4 Challenges in Requirements Validation**

Despite robust processes, Netflix faces challenges in requirements validation:

* **Dynamic User Expectations**: Constantly changing trends in entertainment consumption.
* **Global Compliance Issues**: Adapting to different regional content laws and licensing agreements.
* **Scalability & Performance Bottlenecks**: Maintaining seamless service across diverse network conditions.
* **AI Bias in Personalization**: Ensuring fairness and inclusivity in recommendations.
* **Content Piracy & Security Threats**: Constantly evolving risks in digital rights management.

**5. Conclusion**

Netflix, as a continuously evolving platform, benefits most from **Incremental Development** for rapid feature deployment and **Spiral Model** for high-risk features. Waterfall is not suitable due to its rigid structure.

Effective **requirements engineering** ensures that Netflix meets both **functional** and **non-functional** demands.

**7. References**

1. Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice.
2. Sommerville, I. (2015). Software Engineering (10th Edition).
3. Netflix Technology Blog - <https://netflixtechblog.com/>

**Diagrams to Include:**

1. **Netflix Microservices Architecture**

1. Wikipedia – Netflix; as edited on 3rd February 2025

   <https://en.wikipedia.org/wiki/Netflix> [↑](#endnote-ref-1)
2. Medium – The Tech Behind Netflix’s Unstoppable Streaming by Irene Yu

   [https://iamireneyu.medium.com/the-tech-behind...](https://iamireneyu.medium.com/the-tech-behind-netflixs-unstoppable-streaming-74b30d630b25) [↑](#endnote-ref-2)