**The Code Behind Care: SDLC Models for Epic Systems**

***Empowering Healthcare with a Structured and Adaptive SDLC Approach***

Ananya Shetty

NNM23IS012

**Abstract:**

Epic Systems, one of the leading healthcare software, deals with complex issues of patient data, billing, and compliance, hence needs to have a well-laid Software Development Life Cycle (SDLC). This report aims at analysing the aptitude of the Waterfall, Incremental, and Spiral models for Epic Systems and comparing them in terms of flexibility, risk management, and adherence to regulation. The Waterfall is somewhat stable but very rigid for changes in the healthcare regulations. The Incremental Model will capture the continuous improvements but has little on risk management. Thus, the Spiral On comes out as the most promising, and provides for iterative development, continuous assessment for risk, and adaptability for needing changes. It takes care of security, compliance requirements, and user-driven enhancements that suit the evolution of a dynamic healthcare system. The better software development strategy allows Epic Systems to upgrade the level of health care delivery, mergence of hospital operations, and nurturing future drives in health technology.

**Publishing:**

This paper was submitted to *Dr. Jason Elroy Martis, Associate Professor, Department of Information Science and Technology, NMAM Institute of Technology. Nitte Karnataka, India.* This paper is also hosted on a GitHub repository, along with the material used for preparing this research. The link to the GitHub Repository is given in the endnote.

**Table of Contents**

1. **Introduction 3-4**
2. **Analysis of SDLC Models 4-13**
   * 2.1 Incremental Development Model 4-7
   * 2.2 Spiral Model 7-10
   * 2.3 Waterfall Model 11-14
3. **Requirements Engineering Process 14-15**
   * 3.1 Functional Requirements 14
   * 3.2 Non-functional Requirements 15
   * 3.3 Strategy for Requirements Validation 15-16
   * 3.4 Potential Challenges 16
4. **Conclusion 17**
5. **References 18**

**1.INTRODUCTION:**

The Epic healthcare community is a worldwide group collaborating to improve patient care and innovate healthcare delivery. Epic offers a complete set of tools for patient care management, billing, scheduling, and other critical functions making it a key component of today’s healthcare.

**What Epic Does:**

**1.Electronic Health Records (EHR):** Epic's main product is EHR software. This allows doctors, nurses, and other healthcare providers to maintain patient information electronically instead of on paper.

**2.Solutions for Healthcare:** Epic offers a variety of tools to support different parts of healthcare like EpicCare, MyChart, Tapestry, Revenue Cycle Management.

**3.Data sharing between healthcare providers:** Epic has the ability to share data between different hospitals and healthcare providers.

**4.Customization:** Epic’s software can be customized to meet the needs of different healthcare providers. It is also flexible for various healthcare settings.

**Epic Systems as an Ideal Case for SDLC Analysis**

Epic Systems is a leading provider of healthcare management software which is an ideal choice for studying the Software Development Life Cycle (SDLC) due to many key factors:

* Epic is deployed across major hospitals and healthcare systems, handling large amounts of patient data and complex workflows.
* Epic plays a key role in managing critical healthcare tasks, so it’s essential that the system is reliable, secure, and performs well in real-time making it a perfect example for studying SDLC models in high-stakes situations.

**Purpose of the Report:**

This report aims to explore and compare different Software Development Life Cycle (SDLC) models specifically the Waterfall, Incremental Development and Spiral Model within the context of Epic. By analyzing these models we will highlight their suitability for meeting the specific needs of Epic’s functional and non-functional requirements, as well as their ability to manage risk, handle change, and meet time and cost constraints.

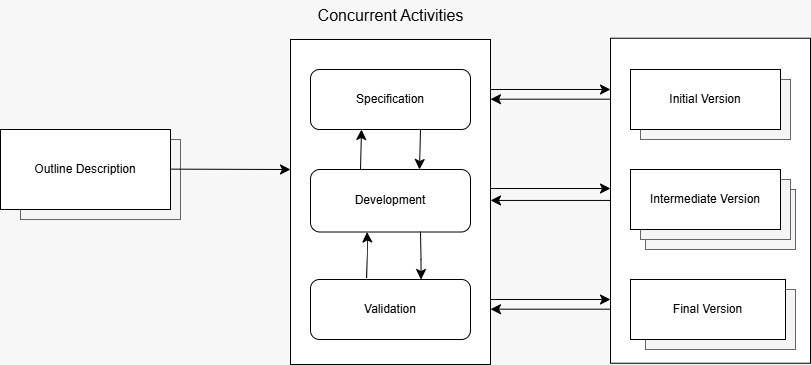
**2.Software Development Lifecycle Model (SDLC):**

The SDLC (Software Development Life Cycle) is a process that guides the development of software.

For a big healthcare system like Epic Systems which helps manage patient data, appointments, and medical records, the SDLC process is really important. In the Incremental Model, Epic Systems develops and releases the system in small parts or increments. For Epic Systems, the waterfall model creates a step by step approach where each phase happens one after the other. The spiral model of the Epic Systems combines the benefits of both waterfall and incremental models.

**CASE STUDY**: A large hospital network decides to implement Epic Systems to improve patient care, automate medical records, and streamline operations.

**2.1 Incremental Development Model:**

*****Fig1: Incremental Development Model*

**1. Outline:**  
The hospital focuses on critical demands like electronic health records (EHR), appointment scheduling, billing, and compliance in order to create a development plan.

**2. Concurrent Activities:**

* **Requirement Specification**: Ongoing, adjusted functional and non-functional requirements assure that a system fits with real demands the hospitals face.
* **Development:** Engineers develop versions from the minimal features to rich, sophisticated functionalities.
* **Validation:** Each version is subject to stringent testing for security, usability, and performance measures to ensure regulatory compliance and operational acceptability.

**3. Incremental Releases:**

* **Initial version:** Minimal features such as patient registration and appointment management are released for testing.
* **Intermediate version:** Enhancement by the billing system, integration with insurance, and sharing functions between departments.
* **Final version:** Other sophisticated features like e- health and AI diagnostics and predictive models are released and made operational.

**Functional and Non-Functional Requirements**

**Functional Requirements**:

* The nucleus elements (EHR, patient management) should be developed first, ensuring that the hospitals can start using the system fairly early.
* Billing, lab integration and telemedicine can be added progressively without any disturbance in the flow of work.
* Doctors and hospital administrators will test early versions and suggest changes, assuring a high level of usability.

**Non-Functional Requirements:**

* Each increment developed will accommodate a growing number of patient records and healthcare facilities.
* Compliance with HIPAA and GDPR will be established from the very beginning, thus helping in protecting patient data since the very first minute.
* System bottlenecks may be detected and subsequently repaired to avoid large failures in the system.

**Risk and Change Management  
Risk Management**

* Problems that may arise are brought to professional notice at parts of the journey before they evolve into major hindrances to project advancement.
* Problems in certain modules (consider billing) can be cured without interfering with the working of basic functions, such as the management of patient records.
* Vulnerabilities when security-related can be addressed in every increment rather than being put off and addressed all at once at the end.

**Change Management**

* Considering the highly dynamic nature of compliance laws, the adoption of an incremental approach allows for faster changes.
* Further improvements based on user feedback can be made open to hospitals and medical practitioners to request after seeing increments early.
* Facilities such as added dependencies based on BI analytics driven by AI or enhancements in the area of telemedicine can be integrated very smoothly.

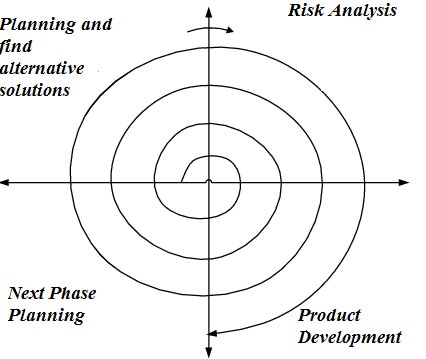
**Time and Cost Constraints**  
**Time Suitability**

* Development of an MVP would enable hospitals to roll it off swiftly with basic functionalities usable almost the first minute after launch.
* Multiple teams would work on multiple increments at relatively similar times, which further lessens the delay in development.
* Continuous testing and feedback will ensure that fewer resources are wasted at the end of the project rectifying major problems.

**Cost Suitability**

* Instead of making full payments at the start for a Waterfall model, in the incremental mode, payments are distributed over time.
* Funds set aside by the hospitals can be spent on incremental improvements as opposed to one major release that demands a pre-determined cost.
* Early detection of bugs reduces the potential for expensive downtime of the system or any fines arising from compliance failure.

**2.2 Spiral Model:**

****

*Fig2: Spiral Model*

**1. Planning and Exploring Alternative Solutions:**  
First, the hospital recognizes its essential requirements, i.e. EHR, billing systems, and telemedicine capabilities. Next, it researches a number of technological options to satisfy these requirements.  
  
**2. Risk Assessment:**  
Once the development starts, it focuses on areas such as data security, patient privacy compliance with HIPAA regulations, and system downtime.  
  
**3. Development of Products:**  
Development takes place in a series of iterations; iterations begin with primitive features like patient registration and build up on with insurance integration and analytics through artificial intelligence.  
  
**4. Next Phase Planning:**  
At the end of each iteration, the healthcare practitioners give feedback to improve the system into the next phases of development.  
  
The Spiral Model provides a systematic guide in developing Epic Systems characterized by continuous refinement and risk management coupled with the involvement of stakeholders, thereby ensuring a safe, efficient, and flexible management system.

**Functional and Non-Functional Requirements:**

**Functional Requirements:**

* Iterative feature development ensures developments in progressive refinements of the system with functionality improvements such as EHR, billing and telemedicine in every iteration.
* Integrating feedback allows doctors, nurses, and administrators to really test the real-world use of these features and cause more user-centric features to emerge from the iteration process.
* Prototyping for the complex features allows for validation and refinement of these advanced modules, such as AI-driven analytics and automated diagnosis tools, before their full deployment.

**Non-Functional Requirements:**

* The system always follows HIPAA and GDPR rules. These rules are checked and updated regularly to make sure they are always up to date.
* The system is built to grow and perform well. We find any slow parts early on and fix them to improve the system over time.
* Keeps improving the system’s design based on feedback from medical professionals, making it easier and more efficient for them to use.

**Risk and Change Management:  
Risk Management:**

* Regularly checks for risks in security, data privacy and system stability in every cycle to catch problems early.
* The system is built step by step with regular checks, which lowers the chance of a big failure.
* Keeps reassessing risks related to patient data and compliance with regulations at every stage to ensure better protection.

**Change Management:**

* Great adaptability to regulatory changes ensures that frequent modifications in healthcare laws can be deployed in the system unattended.
* Integration of new technologies allows advancements such as AI-driven medical-enrolment predictions or blockchain-based record storage to be tested in prototypes before being fully deployed.
* Flexibility for stakeholder collaboration facilitates each iteration sense feedback and risk assessment, allowing hospitals,  
  doctors, and IT teams to request changes at various stages.

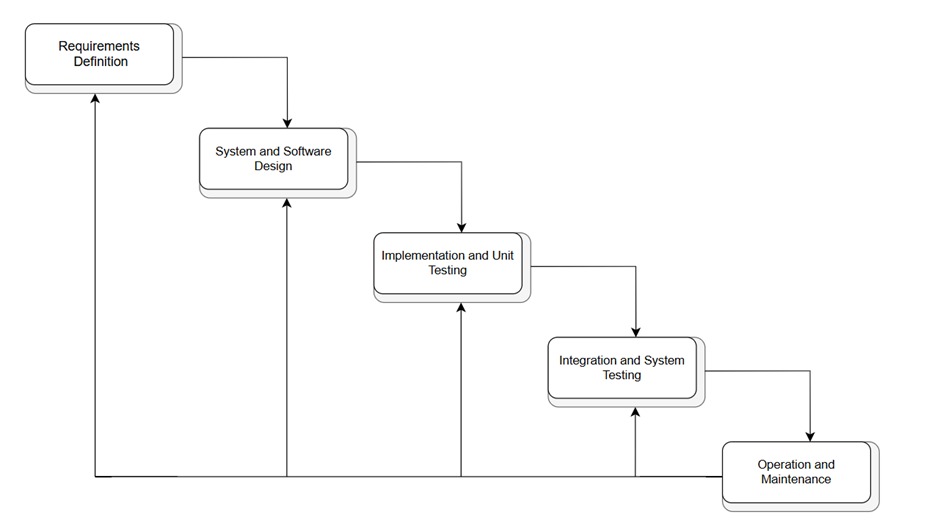
**Time and Cost Constraints:**  
**Time Suitability:**

* The timeliness in launching the product is achieved through a  
  well-thought-out critical path whereby the critical modules TT  
  developed first, paving the way for testing, retesting, and fixing the more complicated modules.
* Risk management pursued in parallel saves time since it is built into development and prevents large delays from redesigns or regulatory penalties.
* Prototyping minimizes rework since significant features are tested  
  in early prototype phases so that expensive, band-aid solutions are avoided later.

**Cost Suitability**:

* This significantly reduced financial risk, as budgeting can be incremental. It significantly lowers the chance for any larger-scale financial fiascos.
* The fixing of bugs is done in a cost-efficient stage since these bugs are detected in earlier spiral cycles before they turn into costly, system-wide calamities.
* Resource optimization assures resource allocation according to risk analysis hence avoiding situation of over-commitment on any low-involvement features.

**2.3 Waterfall Model:**

****

*Fig3: Waterfall Model*

**1. Requirements Definition:**   
At the time of development, hospital's requirements are collected on the granular level of Electronic Health Records (EHR), billing, patient management and compliance needs.  
 **2. System and Software Design:**   
System architecture is proposed, including DBs, security protocols, and user interfaces for physicians, nurses, and administrators.  
  
**3. Implementation and Unit Testing:**   
Programmers write and test the basic modules (e.g., scheduling of appointments, history of the patient, insurance processing) before integration.  
  
**4. Integration and System Testing:**   
All modules are integrated and validated for smooth working in combination, compliance with the regulations, and for proper functioning in the different areas of the hospitals.  
  
**5. Operation and Maintenance:**   
The system is situated at an organizational level across the whole of the hospital network and therefore is supported with regular maintenance, providing updates, bug fixes, and security upgrades with time.  
  
The Waterfall Model guarantees that Epic Systems is built according to a precisely defined process, with an extensive testing process, and a formalized release, and therefore it would be a good option for large hospital networks looking for stability and adherence.

**Functional and Non-Functional Requirements:**

**Functional Requirements:**

* This is very convenient when the requirements are reduced and stiff from the beginning.
* A structured approach to developing key functions for healthcare, including Electronic Health Records (EHR), patient management, billing, and scheduling.
* Supplying all features and then moving to the next stage. This is very well suited to healthcare institutions with standard workflows that are not frequently changed.

**Non-Functional Requirements:**

* Security and compliance with HIPAA and GDPR are planned early on, which will reduce the chances of any legal troubles later.
* Arrival at specifications on system performance and scalability at the design phase ensures smooth operation even under very heavy loads.
* The nature of overall reliability and data integrity is well incorporated since all requirements are analysed well before any implementation.

**Risk and Change Management:**

**Risk Management:**

* Early identification of risks in the planning phases minimizes the chances of unexpected issues later on in the project cycle.
* Well-documented plans allow for risk to be tracked and therefore comply with healthcare processes.
* Since tests happen later, critical issues may be revealed too late, which will make repairs expensive.

**Change Management:**

* Difficult to change anything once the development stage is started.
* Every change must take the process back to the earlier stages, leading to additional costs and delays in the process time.
* Does not meet several health care needs since compliance updates and medical innovations may call for new cycles of development.
* Best suited for projects where issues are relatively stable; in other words, they are not going to change very frequently.

**Time and Cost constraints:**

**Time Suitability:**

* Predictable times as every stage is planned and executed sequentially.
* Good for large projects on a strict time schedule where any delay must be avoided.
* Time-intensive initial planning that allows all requirements to be documented before in-development begins.
* Testing late in development can slow down completion if serious issues are unmasked at that time.

**Cost Suitability:**

* Long projects are easier to get control of from a cost perspective due to fixed budgets.
* Proactive identification of risk avoids larger problems becoming expensive later in development.
* High level of rework if alterations are requested after completion of phase 2. It works poorly for continuous improvements as every new addition would require a new lost cycle of development.

**3. Requirement Engineering Process**

**3.1Functional Requirements:**

1.Patient Management

* Registration of patients, appointments, and their management, as well as keeping records.
* Search and retrieve the following: patient history, test results, and prescriptions.
* Updating the patient records in real-time.

2.Electronic Health Records (EHR)

* To create and manage medical records, prescriptions, and diagnoses.
* Interoperate with other hospital systems and outside health databases.
* Provide authorized personnel with secure access.

3.Billing and Insurance Management

* To create and track medical bills, invoices, and payment records.
* Handle medical billing with health insurance companies and validate patient coverage.
* Provide an automated estimate of treatment costs.

**3.2 Non-Functional Requirements**:  
  
1.Performance and Scalability

* Pass large quantities of patient data quickly.
* Provide for scalability with the expanding hospital network.

2.Security and Data Privacy

* Implement multi-factor authentication for system access.
* Regular automatic data back-up and recovery. Usability and Accessibility.
* Intuitive user interfaces for doctors, nurses, and administrators. From mobile and web accessibility to its remote usage.

3.Reliability and Availability

* Ensuring the reliability and availability of the software such that there is a 99.9% guarantee that hospital activities will proceed without interruption.
* Enable failover application instances in case of crashes.
* Support the integration with third-party software applications, such as a lab system and various pharmacies.

**3.3 Strategy for Requirement Validation:**  **1. Interviews with Stakeholders**: Meet with physicians, nurses, and administrators to gather insights into requirements.  
  
**2. Prototyping:** A prototype or wire frame is developed to provide visual demonstration of system features.  
  
**3. Use Case Scenarios**: Real-life workflows carried out in hospitals ascertain that the prototype features meet real-life requirements.  
  
**4. Requirement Reviews:** Hold formal meetings with Hospital IT Departments and relevant healthcare regulatory bodies.  
  
**5. Testing and User Acceptance**: To make sure that the system will be easy to use, beta test with actual hospital workers.  
  
**3.4 Potential Challenges**

* The modifications to the regulatory requirements lead to updates in compliance requirements; making it a tall order to deal with compliance requirements.
* Conflicting requirements from the stakeholders make it difficult to strike a balance from the huge number of potential future features.
* Interoperability should provide realistic expectations for integration into the other hospital systems.
* The lengthiness and costliness of the whole process could delay the deployment or further development of the project.

**4. Conclusion**

It is clear that the development of Epic Systems as a healthcare management solution requires a planned and standardized approach to the development of software so that the combination of functional, non-functional, and regulatory requirements is fulfilled. Through the comparison of Waterfall, Incremental, and Spiral models, it becomes evident that each model has its advantages and limitations, differently affecting risk management, flexibility, cost, and time constraints.

Out of them all, the Spiral Model appears to be the most appropriate as it truly employs an iterative approach, involves good risk management patterns, and enables changes in health regulations and an evolution in the needs of the system. The Waterfall Model may provide stability and the Incremental Model allows gradual development, whereas the Spiral Model ensures continuous improvement, close involvement of stakeholders, and security compliance, which involves Epic in a very complex environment.

In summary, an effectively coupled requirements engineering process and model of delivery ensure scalability, security, and alignment with the users in health facilities and the public that uses Epic Systems. With the right approach, Epic is hence able to keep doing things that gain better efficiency in healthcare, better hands-on care for the patients, and hence push the innovation of medical technology.

**5.References**

1. Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf -book

2. <https://www.epic.com/software/?searchText=>

3.