



## **SOFTWARE QUALITY ASSESSMENT OF MINIMAL SPIRAL MODEL FOR AI FEATURED APPLICATION DEVELOPMENT**

**<sup>1</sup>Sadhana Pandey and <sup>2</sup>Dr.Abhay Kothari**

<sup>1</sup>Research Scholar, Department Of CSE, SAGE University Indore, India

<sup>2</sup>Professor , Department Of CSE, SAGE University Indore, India

### **Abstract**

Software development life cycle is the framework that helps team manage the entire software project. SDLC consists of various phases. Each phase has many activities. SDLC model improves software quality by ensuring all steps are systematically followed. Various models are widely used by software industries for example waterfall, agile, spiral etc. Some categories of software application may need some activities within a particular phase of SDLC model and other may not. It means activities can be minimized to save cost and effort. In this research work, we aim to understand the effect of minimizing Spiral model activities in context of inventory management software following AI and non- AI approaches. We wish to see the effect on quality attributes of the final product and for this we identified use of minimal a) UI/UX PROTOTYPING b) Architectural design within spiral model. For all kinds of estimation we have used COCOMO II model developed by Berry Bohem.

**Keywords:** Spiral Model, Cocomo II, Software Quality, Ui/Ux Prototyping, Minimization of Activities, AI Software Development.

### **Introduction:**

Software engineering involves defining, implementing, testing, managing, and maintaining software systems, as well as developing the software development process itself. The development of software engineering was seen as a struggle for long. Problems included software that was over budget, exceeded deadlines, required extensive debugging and maintenance, and unsuccessfully met the needs of consumers in many cases. 'Software engineering' encompasses not just the act of writing code, but all of the tools and processes an organization uses to build and maintain that code in its life cycle. 'Functional requirement deals with expected behaviors (i.e. outputs). Non-functional requirements deals with issues like portability, security, maintainability, reliability, scalability, performance, reusability, and flexibility. Software metrics includes all quantitative measures used to assess various aspects of software development processes, products, and projects. These metrics help in dealing with the quality, performance, and efficiency of software development efforts.

Software quality is a complex topic. Mainly, software quality refers to how well a software product conforms to its requirements and meets the needs of its users. From a process perspective, it refers to using disciplined methods or techniques and best practices that are likely to result in a higher quality product. Overall, software quality arises from the interplay of people, processes, and technology. Software quality assurance engineers try to build quality into the products by engaging in activities like requirements analysis, design, coding, testing, and maintenance. Processes like inspections, risk management, and configuration management create a framework for quality of the product. And tools like static analysis, test automation, and metrics provide data to assess and improve software quality on an ongoing basis as the product is being developed. Software quality parameter refers to functional suitability, reliability, performance, and usability, along with security, compatibility etc.

Each and every software is developed using some software development process model which is called SDLC models. The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model and is mostly suited for our research work. The model is mixture of iterative development process model and sequential linear development model. The spiral model consists of four phases. A software project iteratively passes through these phases in iterations called Spirals. The Spiral Model is widely used in the software industry as it is in sync with the natural



development process of any software product, i.e. learning with maturity which involves minimum risk for the customer and the development firms at the same time. COCOMO 81 is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity. It exists in three forms, each one offering greater detail and accuracy. Listed by increasing fidelity, these forms are called Basic, Intermediate, and Detailed COCOMO. In the COCOMO model, software projects are categorized into three types on the basis of their complexity, size, and the development environment. The COCOMO 2 model is based upon the non-linear reuse formula. This cost estimation model helps in providing estimates that represent one standard deviation near to the most likely estimate.

The exponent of the effort equation of the COCOMO 2 model is determined by 5 scale factors, and it has 17 cost drivers assigned to it. COCOMO 2 model is useful in non-sequential, rapid development, reengineering and reuse models of software development cycle and that's why we used it in our research work. AI-based software development using the spiral model includes integrating AI's capabilities for risk analysis, requirement refinement, and iterative development within the spiral model's framework and hence it is mostly suited in this case. The AI can automate tasks like risk identification and mitigation strategy generation, and the spiral model provide a structured, risk-driven process for large and complex projects where requirements are unclear. The combination uses AI to enhance the model's core principles like risk management and iterative improvement, moving beyond traditional methods and **applying the spiral model with AI is obvious.**

The process integrates AI tools into each phase of the traditional spiral model which is explained below:

- **Planning:** AI helps in setting objectives, gathering requirements, and identifying potential solutions and constraints and finally helps in framing functional requirements.
- **Risk Analysis:** AI analyzes data and identifies, evaluates risks, with the potential to automatically generate and evaluate risk mitigation strategies.
- **Development and Validation:** AI mainly helps in development by automating code generation, performing tests, and creating prototypes based on the refined requirements.
- **Evaluation:** The AI also reviews the progress, evaluates the effectiveness of the mitigation strategies, and informs the plan for the next spiral.
- **Iteration:** Each cycle repeats, with the AI-enhanced processes guiding the development of increasingly complete versions of the software in the process.

### **SPIRAL MODEL good for AI application development**

Yes. The Spiral Model is a very good and suitable SDLC model for AI application development, particularly when the application involves machine learning, predictive analytics, intelligent decision-making, data processing, and continuous model improvement.

In fact, for AI-based software development, the Spiral Model can be more appropriate than the traditional Waterfall Model because AI applications involve uncertainty, experimentation, repeated training, testing, and refinement of models.

Why is the Spiral Model suitable for AI application development?

- The Spiral Model is a risk-driven, iterative software development model. It develops software through repeated cycles (spirals), with each cycle improving the previous version.
- AI application development requires several activities that naturally fit into this approach:

### **AI development requirement      How the Spiral Model supports it**

Uncertain requirements      Requirements can be refined in every iteration.



|                        |                                                                                       |
|------------------------|---------------------------------------------------------------------------------------|
| Data collection        | Data can be collected and evaluated incrementally.                                    |
| Model selection        | Different ML algorithms can be experimented with.                                     |
| Model training         | Models can be trained and improved in successive cycles.                              |
| Risk identification    | Data quality, model accuracy, bias, and security risks can be assessed in each cycle. |
| Model validation       | Performance can be evaluated repeatedly.                                              |
| User feedback          | Feedback can be incorporated into subsequent iterations.                              |
| Continuous improvement | AI models and application features can be enhanced incrementally.                     |

Example: Spiral Model for an AI-based Inventory Management System

Consider an inventory management application with AI capabilities.

| <b>Spiral iteration</b> | <b>Development activities</b>                                                | <b>Expected outcome</b>          |
|-------------------------|------------------------------------------------------------------------------|----------------------------------|
| Loop 1                  | Requirement analysis, database design, basic inventory modules               | Initial inventory prototype      |
| Loop 2                  | Data collection, preprocessing and ML feasibility using Decision Tree or SVM | ML proof of concept              |
| Loop 3                  | Demand forecasting using regression and clustering techniques                | Intelligent inventory prediction |
| Loop 4                  | AI integration, security, usability and performance testing                  | Integrated AI inventory system   |
| Loop 5                  | User feedback, model evaluation and optimization                             | Improved final application       |

This approach allows the development team to identify technical and functional risks before investing in complete system development.

The Spiral Model is a strong choice for AI application development, especially when the project involves uncertain requirements, experimental ML algorithms, predictive capabilities and continuous model improvement. However, it should not be claimed as universally superior to Agile or other iterative models.

## Literature Review

The accurate prediction of the software development is critical for the good management for decision and also for determining how much effort and time is required for the project to complete. This paper estimates the effort (person months), and the project schedule , the time required to complete is estimated and the average staffs per month is also calculated.[1].

A distinctive feature of the spiral model is the added risk analysis into the loops. Here risk means the conditions and events that may make it harder for the development team to achieve its goals during development. In each iteration, the prototype



of the project is evaluated and risk is assessed for the next iteration in the process. Sometimes the spiral model is categorized as a risk-driven model [2].

*The quality of a finished piece of software is affected by several factors during the whole process. The SDLC refers to a series of steps performed in to create usable software. A software architecture that meets a large class of needs results from careful planning throughout the software design process and is time taking activity in SDLC. According to the author, an abstract representation of the program's overall structure and a plan for building and organising software according to common standards in the field make up software architecture.[3].*

There are specific activities/phases in a single iteration (Spiral) where output is the small prototype of comprehensive software to be developed. Each interval around the spiral travels through the activities of spiral model. The feasibility of the project for development can be judged by keeping in mind the factors like cost, time and efficiency[4].

The author also included a case study to empirically investigate quality of the product developed using XP while comparing with that of the product developed using Spiral model. Since both XP and Spiral are iterative in nature, it is possible to map phases of XP equivalent to phases of Spiral model easily. Phases of XP seem to have built-in similarities with the phases of Spiral model in terms of activities defined in each model. The first phase of Spiral starts with Identification of objectives, alternative means of development, and constraints applied on such alternatives [7].

In this paper the study primarily aims to define the different dimensions of SQ, identify the requirements for enhancing SQ, and present the challenges when SQ is restricted. The study also provides a review on the impact of quality and its measurement in the life cycle of software development thoroughly. It evaluates the need for a quality standard to measure the increasing quality requirements and size of software in a particular case [8].

Software metrics is directly linked with measurement in software engineering. Correct measurement is the prior condition in any engineering fields, and not only in software engineering, but also as the size and complexity of software increases, manual inspection of software becomes a harder task. Most Software Engineers worry about the quality of software, how to measure and improve its quality [9].

Efforts should be made to use a combination of the estimation techniques to arrive at a better cost and quality estimate. To produce reliable estimates it is needed to have proper knowledge of each technique and the relationship between the software attributes of each kind [10].

Since, all the activities are not required, a study for actual requirement in various types of application software is done by the author. Such study would be very useful for faster software development thereby eliminating unnecessary activities in each phase of software development process while eliminating some activities may reduce cost and time. The results of studies will be a set of minimum activities for each type of application software that are necessary for the development of good quality software [5]. In this paper, the author concluded that some activities may be mandatory in one kind of application and may be optional for other kind of application development. So the study of process model for various application development leads to minimized version of process model [6].

## Methodology

WE HAVE Performed estimations of size-effort-time and staff using COCOMOII post architecture model. For this purpose 2(two) loops of spiral model were used. We again performed estimation with elimination of UI/UX prototyping and looked at the results. We again performed estimations after eliminating architectural designs. We generated the graphs for quality attributes to see the effects of these eliminations. Performed correlation and regression analysis on these quality attributes.



Cover Page



## Results and Discussion:

To estimate **Size, Cost, Time, and Staff** for a **desktop-based inventory management system** using the **Spiral Model** and **COCOMO II**

We assume the following about the system:

### Project Scope:

- A desktop-based inventory management system with:
  - Product catalog management
  - Purchase and sales tracking
  - Supplier/customer management
  - Basic reporting & analytics
  - User authentication
- **Platform:** Windows Desktop, built in C#, JavaFX, or Electron (for example)

### COCOMO II model: Post-Architecture Model

Reuse and maintenance are not yet considered.

### Estimates

**Size = 144 FP × 53 LOC = ~7,632 LOC ≈ 7.6 KSLOC**

We'll use a range: **7.5 KSLOC – 10 KSLOC**.

### COCOMO II Estimation (Post-Architecture)

Key Inputs:

| Parameter              | Value                                    |
|------------------------|------------------------------------------|
| Size (KSLOC)           | 7.6                                      |
| Model                  | Post-Architecture                        |
| Development Mode       | Semi-Detached (medium size & experience) |
| Effort Multiplier (EM) | Based on typical values                  |
| Scale Factors (SF)     | ~3.0                                     |

Let's plug into the **COCOMO II Post-Architecture formula:**



$$\text{Effort (PM)} = A \times (\text{Size})^E \times \Pi(\text{EM}_i)$$

- $A = 2.94$  (constant for COCOMO II)
- $E = 0.91 + 0.01 \times \Sigma(\text{SF}_i) \rightarrow \text{Assume } \Sigma(\text{SF}_i) = 16 \rightarrow E = 0.91 + 0.16 = 1.07$
- **Size** = 7.6 KSLOC
- **EM** = Product of effort multipliers = Assume typical EM product  $\approx 1.2$

### **Effort:**

$$\text{Effort} = 2.94 \times (7.6)^{1.07} \times 1.2 \approx 2.94 \times 8.64 \times 1.2 \approx 30.5 \text{ Person-Months}$$

### **Schedule Estimation (Time)**

$$\text{Development Time (TDEV)} = C \times (\text{Effort})^F$$

- $C = 3.67$  (standard)
- $F = 0.28 + 0.2 \times (E - 0.91) = 0.28 + 0.032 = 0.312$

$$\text{TDEV} = 3.67 \times (30.5)^{0.312} \approx 3.67 \times 2.45 \approx 9.0 \text{ months}$$

### **Staff Estimation**

$$\text{Average Staff} = \text{Effort} / \text{Time} = 30.5 / 9 \approx 3.4 \approx 3-4 \text{ people}$$

### **Spiral Model Considerations**

The Spiral Model involves **iterations**, risk analysis, and prototyping. So the phases are:

1. **Planning & Requirements**
2. **Risk Analysis & Prototyping**
3. **Development & Testing**
4. **Evaluation**

Each spiral loop might take **2–3 months**, and there could be **3–4 loops**, aligning well with the 9-month schedule.

#### **Loop 1 (3.6 months):**

- Requirements gathering
- Feasibility & risk analysis
- UI/UX prototyping
- Architecture decision

#### **Loop 2 (5.4 months):**

- Full implementation
- Testing and QA
- Packaging & deployment
- **Final Estimation Summary (Updated with Loop 1 Activity Weighting)**

| Category   | Value               |
|------------|---------------------|
| Total Size | 7.6 KSLOC (~144 FP) |



Cover Page



| Category             | Value                |
|----------------------|----------------------|
| <b>Total Effort</b>  | 30.5 Person-Months   |
| <b>Schedule</b>      | 9 Months             |
| <b>Loop 1 Effort</b> | 12.2 PM (3.6 months) |
| <b>Loop 2 Effort</b> | 18.3 PM (5.4 months) |
| <b>Loop 1 Team</b>   | ~3 people            |
| <b>Loop 2 Team</b>   | ~4 people            |

Now we shall present the results of no UI/UX prototyping and using basic UI/UX prototyping in spiral model.

### Summary of Impact (No UI/UX Prototyping)

| Factor            | With UI/UX | Without UI/UX                                                                                                                      |
|-------------------|------------|------------------------------------------------------------------------------------------------------------------------------------|
| Total Effort (PM) | 30.5       | 33.25 (+9%)                                                                                                                        |
| Total Duration    | 9 months   | ~9.6 months (+6%)                                                                                                                  |
| Cost(Mid Region)  | ~\$213,500 | ~\$232,750 ( The cost would be divide by 90 in Indian Rs. And is usually 1/8 if developed offshore in INDIA as salaries are 1/8th) |
| Usability         | High       | Medium to Low                                                                                                                      |
| Rework in Loop 2  | Low        | Moderate to High                                                                                                                   |
| StakeholderBuy-In | High       | Medium                                                                                                                             |

### Recommendation

Even **basic UI/UX prototyping** (wireframes, click-through mockups) costing just **2 PM effort** can:

- Reduce rework
- Improve clarity in requirements
- Raise user satisfaction
- Avoid delays in Loop 2

**Conclusion:** Eliminating UI/UX prototyping **saves ~2.3 PM up front**, but **adds 2.75+ PM later** and increases risk — **not cost-effective**.

Removed:

- No user interface prototypes
- No wireframes/mockups before implementation
- No early user feedback on usability or workflows



## Negative Impacts

| Quality Attribute   | Impact Description                                                             |
|---------------------|--------------------------------------------------------------------------------|
| Usability           | High risk of poor navigation, confusing layouts, and low user satisfaction     |
| Rework              | Increased likelihood of design rework in Loop 2 (more expensive late changes)  |
| Requirements Fit    | Users may not fully visualize what they're asking for — ambiguous requirements |
| Adoption & Training | Higher training/support burden due to less intuitive interfaces                |
| Stakeholder Buy-in  | Lower confidence in the early stages without visual progress                   |

## Results of Spiral Model Breakdown – Minimal Loop 1

We are only doing:

- Basic planning
- Basic risk identification

This means:

- **No detailed requirements**
- **No UI/UX prototyping**
- **No architectural design**
- **No technical feasibility**
- **No stakeholder validation**

All major design/planning work is pushed to Loop 2.

## Revised Effort Distribution

| Loop         | Description                  | % Effort | Person-Months  |
|--------------|------------------------------|----------|----------------|
| Loop 1       | Basic Planning & Risk Only   | 15%      | 4.6 PM         |
| Loop 2       | All actual design + dev work | 85%      | 25.9 PM        |
| <b>Total</b> |                              | 100%     | <b>30.5 PM</b> |

## STEP 3: Time Estimation

$$\text{Schedule(TDEV)} = 3.67 \times (\text{Effort})^F$$

$$F = 0.312$$

$$\text{TDEV} = 3.67 \times (30.5)^{0.312} \approx 9.0 \text{ months}$$



Cover Page



## Loop % Time Duration

Loop 1 15% 1.35 mo

Loop 2 85% 7.65 mo

**Total 100% 9.0 mo**

## STEP 4: Team Size

Avg Team Size = Total Effort / Duration = 30.5 / 9 ≈ 3.4 → ~3–4 people

## STEP 5: Cost Estimate

### Region Cost/PM Total Cost

Low-Cost \$4,000 \$122,000

Mid-Cost \$7,000 \$213,500

High-Cost \$10,000 \$305,000

The cost would be divide by 90 in Indian Rs. And is usually 1/8 if developed offshore in INDIA as salaries are 1/8th

## STEP 6: Impact on Final Product Quality

Now let's evaluate **how skipping most of Loop 1 affects product quality.**

### Missed Activities

We've skipped:

- Requirements elaboration
- UI/UX prototyping
- Architectural planning
- Technical feasibility
- Stakeholder feedback

### Effects on Quality

| Quality Attribute      | Impact                                                                        |
|------------------------|-------------------------------------------------------------------------------|
| <b>Usability</b>       | Poor. No prototypes → unintuitive UI, unhappy users                           |
| <b>Correctness</b>     | Risk of building the wrong thing due to vague/unstated requirements           |
| <b>Maintainability</b> | Poor architecture → difficult to scale, fix, or extend                        |
| <b>Performance</b>     | Potentially poor choices (e.g., DataBase, caching) without feasibility checks |



## Quality Attribute Impact

**Rework Cost** Higher in Loop 2 → late changes are 3–5× more expensive

**Stakeholder Satisfaction** Low. No early validation or feedback

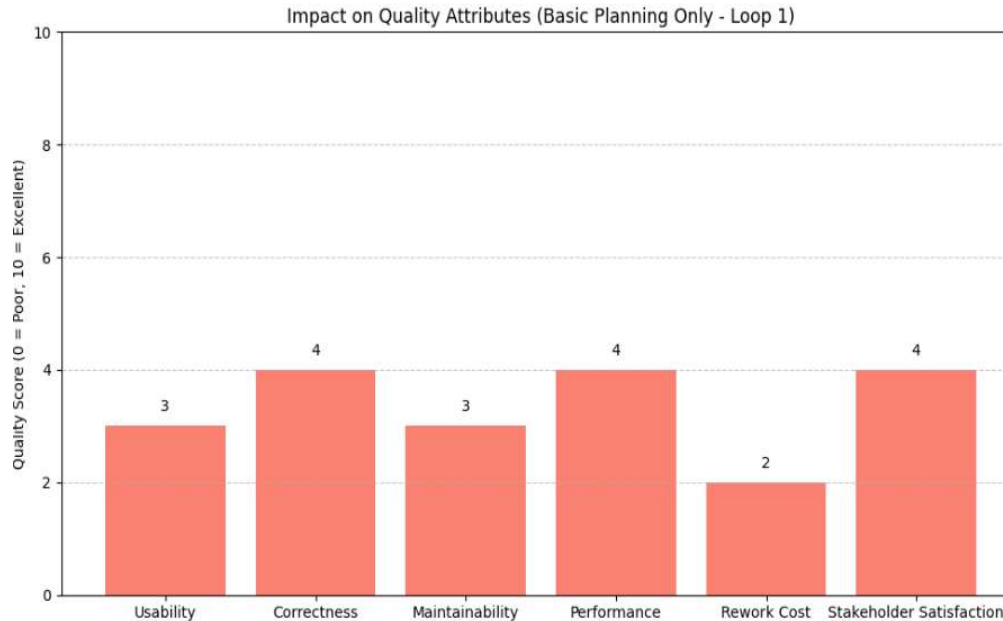


Figure 1 (impact of quality attributes – Basic planning only Loop 1)

### Interpretation of correlation analysis

- **Pearson Correlation Coefficient ( $r$ )  $\approx 0.99$**
- This indicates a **very strong positive correlation** between **Cost** and **Bugs & Fixes**.
- In this case: **as bugs increase, cost increases** significantly (mainly due to rework, testing, debugging time).

To perform a **multivariate regression analysis** on the **Quality Implication of Skipping UI/UX Wireframes in Loop 1**, we need to **quantify** the qualitative data and define:

1. **Dependent Variable (Y):** Typically, a measurable impact — in this case, we'll use **Cost** (as it's directly measurable and affected by other variables).
2. **Independent Variables (X):** Other qualitative aspects affecting cost:
  - Rework
  - End-User Satisfaction
  - Development Time
  - Bugs & Fixes

Estimated Regression Equation:

$$\text{Cost} = 236800 + 7200(\text{Rework}) + 7200(\text{Satisfaction}) + 8000(\text{DevTime}) + 8000(\text{Bugs}) \text{ } \text{Cost}$$



Cover Page



The cost would be divide by 90 in Indian Rs. And is usually 1/8 if developed offshore in INDIA as salaries are 1/8th

## Conclusion

- **Skipping UI/UX early on leads to more bugs**, which then **increases cost**.
- Investing in **good design up front** (even simple wireframes) can **greatly reduce bugs and cost**.

## Discussions:

With elimination of UI/UX and prototyping we come out with unintuitive UIs that means low in quality; correctness also gets affected negatively. With elimination of architectures maintainability goes low; as well the performance and the stakeholder satisfaction goes lower. Rework cost goes higher.

## Inventory System with AI capabilities

In this section we re adding AI functions with earlier inventory system and working to estimate the **size, cost, and time** for developing an **Inventory System with AI capabilities** using **COCOMO II** and the **Spiral Model**, we'll break the problem down step by step. This includes:

### Project Overview

- **Project Type:** Inventory Management System
- **AI Abilities:** Prediction (e.g., demand forecasting) and classification (e.g., item categorization)
- **ML Algorithms:** Decision Tree, SVM, K-Means, etc.
- **Development Model:** Spiral Model (2 Loops)
- **Estimation Model:** COCOMO II (Post-Architecture Model)

### *Process model with AI:*

#### Loop 1: Prototype + Requirements + ML Feasibility

- Develop ML proof-of-concept (Decision Tree, SVM)
- Validate classification accuracy (item category, vendor)
- Risk: ML data availability, model performance

#### Loop 2: Final Architecture + Integration

- Full inventory logic
- AI modules for demand prediction (e.g., K-means clustering + regression)
- Final testing, security, UX improvements

The results for Inventory system with AI- with and without SPIRAL are:

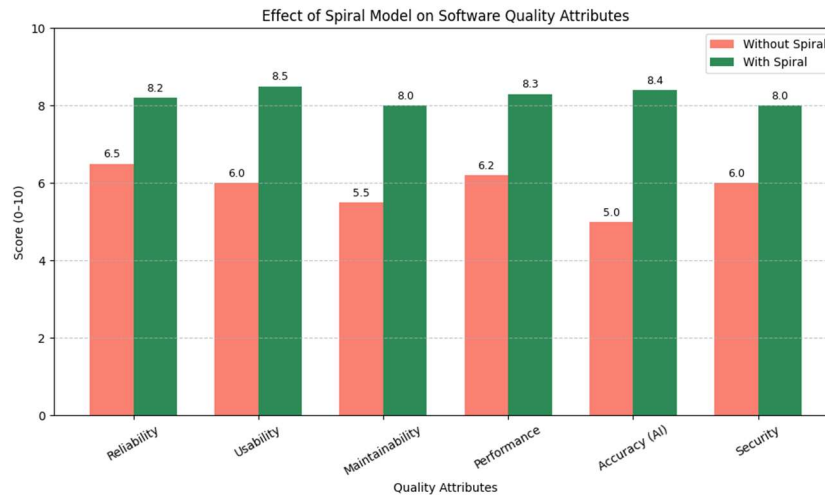


Figure 2 effect of spiral model on software quality attributes

## Discussion:

The bar chart compares software quality attribute scores **with** and **without** the use of the **Spiral Model** during development. Across all six evaluated attributes—Reliability, Usability, Maintainability, Performance, Accuracy (AI), and Security—the Spiral Model consistently produces higher scores. This pattern highlights the model's strengths, particularly its iterative and risk-driven nature.

### 1. Reliability

WithSpiral:8.2

Without Spiral: 6.5

The substantial improvement in reliability suggests that the Spiral Model's repeated validation and risk analysis cycles help identify and mitigate potential system failures early. Continuous refinement supports more robust, dependable software.

### 2. Usability

WithSpiral:8.5

Without Spiral: 6.0

Usability sees one of the largest boosts. Because the Spiral Model incorporates user feedback in each iteration, design flaws and usability issues are addressed progressively. This leads to more user-centric interfaces and smoother interactions.

### 3. Maintainability

WithSpiral:8.0

Without Spiral: 5.5

Maintainability shows a significant improvement, likely due to the model's structured documentation, iterative refinement, and evolving prototypes. These practices help create clearer, modular designs that are easier to update and debug.



## 4. Performance

WithSpiral:**8.3**

Without Spiral: **6.2**

Performance gains reflect the advantage of iterative testing. Bottlenecks and inefficiencies can be identified in earlier cycles and optimized before full-scale deployment.

## 5. Accuracy (AI)

WithSpiral:**8.4**

Without Spiral: **5.0**

Accuracy, especially for AI-driven systems, benefits greatly from the Spiral Model. AI solutions often require repeated tuning, model evaluation, and error analysis—all of which align well with the model's risk-driven, iterative structure.

## 6. Security

WithSpiral:**8.0**

Without Spiral: **6.0**

Security improvements indicate that the Spiral Model's emphasis on risk assessment directly contributes to identifying vulnerabilities early. Iterative reviews allow proactive integration of security measures rather than reactive patching.

## Overall Interpretation

The following heat maps show the results of correlation analysis on quality attributes in two situations: one where Spiral model is applied and second where spiral model is not applied.

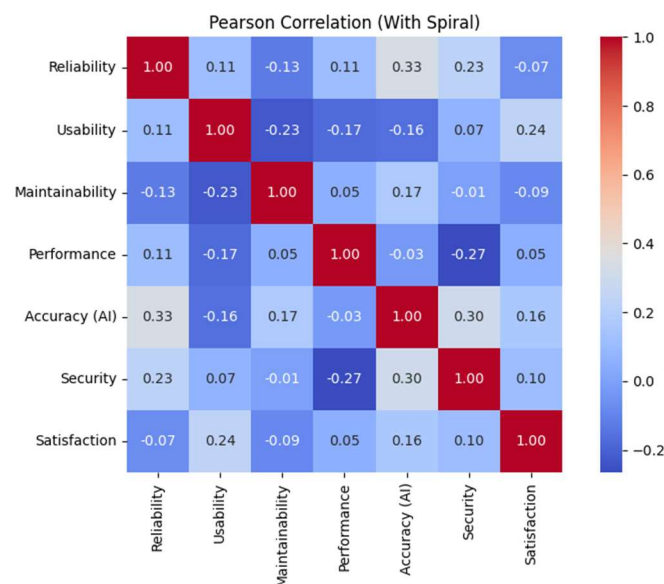


Fig 3: Inventory system with AI-ML using spiral model correlation study plot

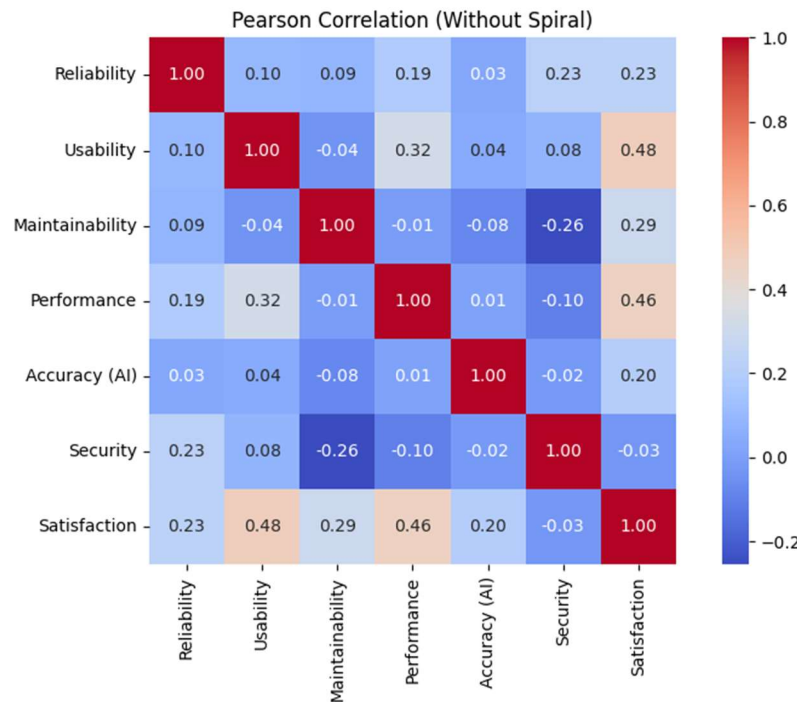


Fig 6: Inventory system with AI-ML using no spiral model correlation study plot

## Conclusion:

We term minimal loop1 as executing loop1 of SPIRAL model with elimination of UI/UX prototyping activity and the activity of architectural designs. We performed all estimations with COCOMOII post architectural MODEL applying it on Inventory management software with and without AI features. After applying the method mentioned and looking at the effects of this method on quality attributes we conclude following:-

While a **minimal Loop 1 may save time and effort initially**, it increases:

- Total cost
- Rework
- Risk
- Time
- And **significantly lowers quality** of the final product

For Inventory system with AI features the results are presented and that the results collectively demonstrate that the Spiral Model significantly enhances software quality across all key attributes. Its iterative nature supports continuous improvement, and its focus on risk management ensures early detection of issues.

The largest improvements are observed in:

- Accuracy (AI)
- Usability
- Maintainability



Cover Page



## References

1. K. Htay, M. M. Aung, Y. Y. Cho, and M. M. Thein, "Software engineering cost estimation using COCOMO II model," *International Journal of Trend in Scientific Research and Development (IJTSRD)*, vol. 3, no. 5, pp. 2326–2329, Aug. 2019, doi: 10.31142/ijtsrd28019.
2. R. Haraty and G. Hu, "Software process models: A review and analysis," *International Journal of Engineering & Technology*, 2018, doi: 10.14419/ijet.v7i2.29.13206.
3. D. Kumari and S. Suman, "Study of software development process and its impact on software features: A review," *Journal of Emerging Technologies and Innovative Research (JETIR)*, vol. 11, no. 5, May 2024.
4. S. A. Ghanghro, M. A. Sawand, W. A. Channa, U. A. Kashif, M. H. Tunio, K. Kumar, and N. Nooruddin, "Comparative analysis of software process models in software development," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 10, no. 3, May–Jun. 2021, ISSN: 2278-3091.
5. S. Pandey and A. Kothari, "Identification of minimum software process development activities for development of categories of software: An efficient and effective approach," in *Data Science and Intelligent Computing Techniques*, S. J. Nanda and R. P. Yadav, Eds. SCRS, India, 2023, pp. 329–336, doi: 10.56155/978-81-955020-2-8-29.
6. S. Pandey and A. Kothari, "Process models and estimation methods in cloud, artificial intelligence and mobile software development scenarios: A systematic review," *International Journal of Computer Applications*, vol. 187, no. 18, pp. 33–38, Jul. 2025, doi: 10.5120/ijca2025925249.
7. F. M. Al Obisat, Z. T. Alhalhouli, T. I. Alrawashdeh, and T. E. Alshabat, "Review of literature on software quality," *World of Computer Science and Information Technology Journal (WCSIT)*, vol. 8, no. 5, pp. 32–42, 2018.
8. Y. I. Hrytsiuk and T. O. Mukha, "Methods of determination of quality of software."  
 Note: Journal/conference name, publication year, volume, issue, page numbers, and DOI are required to complete this reference.
9. S. Shekhar and U. Kumar, "Review of various software cost estimation techniques," *International Journal of Computer Applications*, vol. 141, no. 11, pp. 31–34, May 2016, doi: 10.5120/ijca2016909867.