**How the Impact Factor Works?**

* If **requirement changes = 0-2** → **Impact Factor = 1.0** (No change)
* If **requirement changes = 3-5** → **Impact Factor = 1.1** (Increase timeline by 10%)
* If **requirement changes = 6+** → **Impact Factor = 1.2+** (Increase timeline by 20%)

Example:

| **Requirement Changes** | **Predicted Timeline (Days)** | **Impact Factor** | **Final Prediction** |
| --- | --- | --- | --- |
| 2 | 100 | 1.0 | 100 |
| 4 | 100 | 1.1 | 110 |
| 7 | 100 | 1.2 | 120 |

"The impact factor is based on real-world project management principles and estimation models like COCOMO and Function Point Analysis. Since software projects experience different levels of requirement volatility, I categorized them into three tiers: low (0-2 changes), medium (3-5 changes), and high (6+ changes). The multipliers (1.1 and 1.2) are conservative estimates, ensuring that timeline extensions remain reasonable while reflecting the actual challenges teams face when adapting to requirement changes. Additionally, this method avoids retraining the model while still providing more accurate, real-world predictions."

| **Question** | **How to Respond** |
| --- | --- |

|  |  |
| --- | --- |
| *Why did you choose 3-5 changes for moderate and 6+ for high impact?* | "Based on research and industry observations, small projects tolerate 1-2 changes well. Beyond 3, teams experience noticeable delays. At 6+, projects often need significant rework, causing major delays." |

|  |  |
| --- | --- |
| *Why 1.1 and 1.2 as multipliers?* | "These are **pragmatic estimates** based on industry insights. Since major requirement changes typically cause **non-linear delays**, the impact factor scales up moderately (10-20%) to reflect this." |

|  |  |
| --- | --- |
| *Can we make the impact factor dynamic?* | "Yes! Instead of fixed values, we can adjust multipliers based on real-world data if available." |

**How Did I Determine the Impact Factor?**

1. **Industry Experience & Common Project Trends**
   * In real-world software projects, **minor requirement changes (0-2 times)** usually do not impact timelines significantly.
   * However, when requirements change **3-5 times**, there is typically **a noticeable delay** due to rework, additional testing, and modified implementation strategies.
   * When changes exceed **6 or more**, the project often undergoes **major restructuring**, leading to **significant timeline extensions**.
2. **Academic Literature & Research Insights**
   * Studies in **software engineering** (such as Agile and traditional project management) show that requirement changes increase project delays due to:
     + **Rework** (code modifications & additional testing)
     + **Team misalignment** (task dependencies & scope creep)
     + **Resource inefficiencies** (unexpected workload shifts)
   * **Example reference:** *Boehm's Cost of Change Curve* → As changes occur later in the development cycle, they become exponentially expensive in terms of time and resources.
3. **Empirical Justification Based on Estimation Models**
   * Traditional **Software Estimation Models** (e.g., COCOMO, Function Point Analysis) suggest that scope changes **increase effort exponentially** rather than linearly.
   * Since your model already **estimates project timelines**, applying a **simple multiplier (1.1 for moderate, 1.2 for high changes)** reflects a **practical approach** based on industry best practices.
4. **Practical Justification: Keeping It Simple Yet Effective**
   * Rather than retraining the model with **real-world historical change data (which we don’t have)**, this **adjustment method keeps the predictions dynamic and useful.**
   * **If needed, the multipliers can be fine-tuned** based on further research, real-world data, or supervisor feedback.