Pair Compatibility (PC): refers to the compatibility between a pair of developers and is based on the quality of collaboration between two developers who have previously worked together on past projects.

PC	
Value	Description
VL	**POOR COLLABORATION** **Characteristics:** No Communication: The two developers rarely, if ever, communicate. They work in complete isolation from each other, even if their tasks are interdependent. Lack of Trust: There is no trust between the developers. They avoid asking each other for help or feedback, leading to missed opportunities for improvement. Conflicting Work Styles: Their approaches to coding, problem-solving, and task management are incompatible, leading to misunderstandings and conflicts. No Collaboration: They refuse to work together, even when collaboration would resolve issues faster. Each developer prefers to handle tasks alone, resulting in duplicated efforts or broken integrations. Institutional: Both developers do not follow technical standards, they work without using the institution's tools, thus making it difficult to monitor their activities and consequently the project. **Example**: Two developers working on different modules of the same application frequently ignore each other's progress, resulting in code conflicts and integration issues. They do not seek or offer feedback, and when problems arise, they each try to fix them alone, often exacerbating the issues.
L	**BELOW AVERAGE COLLABORATION** **Characteristics**: Minimal Communication: The developers only communicate when absolutely necessary, such as when a blocker arises or during mandated team meetings. Weak Trust: They show little trust in each other's abilities and are reluctant to delegate tasks or ask for help. Inconsistent Collaboration: Collaboration occurs occasionally but is not sustained. They may work together on urgent tasks, but otherwise, they avoid working as a pair. Limited Feedback: They rarely exchange feedback, and when they do, it is usually superficial. Neither developer is particularly interested in improving the other's work. **Institutional**: They partially adhere to technical standards but may skip institutional tools or processes, making monitoring and consistency challenging. **Example**: Two developers working on a shared microservice occasionally communicate during sprint meetings, but outside of those formal settings, they rarely collaborate. One

	developer often reworks the other's code rather than discussing improvements, leading to frustration and wasted time.
M	**AVERAGE COLLABORATION** **Characteristics:** Functional but Infrequent Communication: The developers communicate well during specific tasks, but their collaboration is primarily task-driven and doesn't extend beyond immediate needs. Moderate Trust: They trust each other enough to ask for help when stuck, but only after trying to solve issues alone first. Occasional Pair Programming: The developers sometimes work together on complex tasks or during bug fixing, but pair programming or teamwork is not a regular part of their routine. Basic Feedback Exchange: They give each other feedback, but this exchange tends to focus on immediate technical issues rather than long-term growth or quality improvements. **Institutional**: They generally follow institutional standards and tools but may occasionally skip documentation or updates, leading to minor inconsistencies in project tracking. **Example**: Two developers working on a shared feature coordinate well during the planning phase and when blockers arise. They occasionally pair program when debugging complex issues but often return to working independently once the problem is resolved. Feedback is limited to technical fixes rather than constructive discussions.
H	**ABOVE AVERAGE COLLABORATION** **Characteristics:** Frequent and Clear Communication: The developers communicate regularly, share ideas, and update each other on their progress. They naturally check in with each other without waiting for structured meetings. High Trust: They trust each other's technical skills and judgment, often asking for advice or help without hesitation. Regular Pair Programming: They frequently work together on tasks, either through pair programming or brainstorming sessions, and collaborate to solve problems efficiently. Constructive Feedback: They give each other constructive feedback regularly, which improves the quality of the code and helps them grow as developers. They are open to each other's suggestions and actively incorporate them. **Institutional**: They fully adhere to institutional standards, using recommended tools and processes, which facilitates project tracking and ensures high-quality work. **Example**: Two developers working on an API service regularly review each other's code, pair program when tackling complex refactors, and frequently check in on each other's progress. They collaborate in daily discussions and offer suggestions that improve the overall quality of their work.

VH	**PÈRFECT COLLABORATION** **Characteristics:** Seamless Communication: The developers are in constant communication, discussing tasks, sharing insights, and keeping each other informed about progress without it feeling forced. Complete Trust: They have a high level of trust in each other's abilities. They divide tasks effortlessly, knowing the other will handle their responsibilities with minimal oversight. Constant Collaboration: Whether they're solving bugs, implementing new features, or refactoring code, these developers prefer to work together and often anticipate each other's needs without explicit communication. Continuous Feedback and Learning: Feedback is continuous and reciprocal. They mentor each other, share knowledge regularly, and consistently help each other improve. This results in both technical and personal growth. Shared Ownership: They feel equally responsible for the success of shared tasks and often contribute beyond their immediate areas to ensure the overall project succeeds. **Institutional**: They fully embrace and utilize all institutional standards, tools, and best practices. This adherence enhances transparency, quality, and project monitoring, setting a benchmark for institutional compliance. **Example**: Two developers working on a complex AI feature work seamlessly together. They alternate between writing code and reviewing each other's contributions in real time. They pair program daily, tackle complex issues together, and offer mutual feedback that drives continuous improvement. Their collaboration is natural, proactive, and results in consistently high-quality outcomes.