ROLE DESCRIPTION

DEVELOPER

WILDBIT

crewmojo

WELCOME TO

CREWMOJO

Enabling performance experiences to be designed and delivered in days not months.

Use these templates for inspiration or as a starting point for your own system. When you need to automate and scale the process, each template is ready to go in the Crewmojo platform.

Take a personal tour of Crewmojo.

crewmojo

Developer

This document describes the role by focusing on good and bad behaviours to clearly indicate what success looks like. It has been open-sourced from the Wildbit Good Bad Project

Role:	Developer
Level:	Individual Contributor
Skills & Behaviours	Codes Well
	Quality Focused
	User Focused
	Collaborative
	Connector
	Communicator
	Timely
	Performance focused
	Works within the architecture
Credit	Wildbit Good Bad Project

Codes Well

A good developer knows how to code well and is passionate about their craft. They care about the quality of the code and work hard to maintain it.

A bad developer creates technical debt by sacrificing quality to meet a delivery deadline. They don't keep other developers in mind when writing code and fail to make it easy to understand their work.

A good developer tests their own code instead of relying on QA to find every bug. Having said that, they value the safety and self-documenting properties of automated tests. They plan for ways in which their new work can introduce issues and have a plan B. A bad developer fails to document their work. They never ask for a code review for changes they make and have no interest in reviewing other people's code. A good developer is independent enough to know when the time is right to include others during a project. A bad developer sits on their hands after running out of tasks or hitting a roadblock. They fail to find consistency with existing code structure when making code updates or changes.

Quality Focused

A good developer tests their own code instead of relying on QA to find every bug. Having said that, they value the safety and self-documenting properties of automated tests. They plan for ways in which their new work can introduce issues and have a plan B. A bad developer fails to document their work. They never ask for a code review for changes they make and have no interest in reviewing other people's code. A good developer is independent enough to know when the time is right to include others during a project. A bad developer sits on their hands after running out of tasks or hitting a roadblock. They fail to find consistency with existing code structure when making code updates or changes.

User focused

A good developer is motivated by the success of their project. They keep the user in mind when building features. They are innovative in their thinking and think about the future to see the patterns of issues their systems and customers could encounter. They build long-term solutions for these issues. A bad developer puts technical correctness above user experience. They blame customers for not using the product "correctly.”

A bad developer fails to take ownership and responsibility for the entire product and bugs. They make sure everyone knows exactly who was responsible when a bug was created by someone else.

Collaborative

A good developer makes sure that the team is in tune with what is being released. They work closely with marketing and customer success to ensure that everyone is ready for each release.

A bad developer releases work before the marketing and success team knows and therefore causes issues for customers.

Connector

A good developer is eager to learn new things. They strive to understand how all the pieces of the architecture work together and what state they are in. They question the design and ideas behind features to solve for a solution. They understand what makes a good user experience.

A bad developer is attached to their favorite technology. They think a single method or process is the "ideal,” and that product history and situation should never drive decisions. They bring unnecessary dependencies into the project to suit their preferences.

Communicator

A good developer can communicate issues across the team as they happen to avoid internal confusion and customer frustration. They are glad to listen to advice and feedback. A bad developer is unwilling to shift their focus from whatever they're doing to a help a customer in need. They lack empathy for customers and users. A good developer works closely with other team members to make sure nothing gets lost in translation. They are a good teacher and share their expertise with the rest of their team. A good developer respects other people's time. They pull together all the background info before sending a task to another team member.

A bad developer fails to specify needs and expectations from other teams ahead of time. They are reluctant to help other team members. A bad developer blames other factors when their code causes an issue or fails to solve a problem, for customers, or their project.

Timely

A good developer knows how to estimate their time on a task and makes sure they finish their work in a reasonable amount of time with realistic deadlines. They prefer to use existing solutions, but aren't afraid to invent something new if it's necessary.

A bad developer creates stress for themselves and for the team by failing to prioritize their time. They will spend weeks on unnecessary work because they failed to ask insightful questions before getting started.

Performance Oriented

A good developer builds for project performance. They build within limits and always consider memory, storage, I/O, and network functions.

A bad developer complains about the limitation of their resources.

Works within the architecture

Good developers assume that what exists was designed with thought or prior reasoning in mind.

A bad developers assumes the previous author was unqualified and aims to change things instead of understanding history.

A good developer doesn’t strive for change to benefit their personal presence, and know the difference between standing on preference and principle.

Bad developers crave complexity to maintain control. They don’t recognize the difference between naive shortcuts and simplicity.

EXPLORE MORE

Building a world-class performance culture is made easy with our template library and pre-designed employee experiences.

Templates:

- One-on-one templates
- Performance review templates
- Role descriptions
- Goal templates
- Survey templates
- Engagement surveys
- and more

Experiences:

- Onboarding new employees
- Goal setting & alignment
- Growth plans & coaching
- Skill tracking & development
- Feedback & recognition
- Stay interviews
- Performance reviews
- Exit surveys & interviews
- and more

View Templates

Book a Demo