Adrien Protzel

# Heuristic evaluation debrief

## GitHub

1. How do you feel about the software, now that you have evaluated it?

GitHub is a great platform for organizing, archiving, and collaborating on projects. It offers a wide range of features and visibility into other developers' work. However, it can feel overwhelming due to the sheer amount of information and technical terminology. For new users, learning how to navigate and use GitHub effectively requires external research or trial-and-error exploration.

2. Were there any heuristics that stood out as being particularly important (from either set of heuristics)?

Yes, the heuristic of “recognition rather than recall” stood out as especially important. GitHub uses a lot of domain-specific language like “repository,” “commit,” and “pull request,” which can be confusing for beginners. Once users become familiar with these terms and workflows, they can recall the steps more easily but initially, the lack of intuitive guidance makes the learning curve steep.

3. Which heuristics did you struggle the most with? Why?

I struggled most with applying the “aesthetic and minimalist design” heuristic. GitHub’s interface is dense with information, menus, and options, which can be useful for experienced users but overwhelming for newcomers. It was difficult to evaluate whether this complexity was necessary or if it could be simplified without sacrificing functionality. Additionally, applying the “help and documentation” heuristic was challenging because GitHub does offer help resources, but they are often buried or assume prior knowledge, making them less accessible to beginners.

4. How did the two sets of heuristics compare in terms of advantages and disadvantages? (e.g., what did Nielsen’s Heuristics catch that the Inclusivity Heuristics didn’t, and vice-versa?)

Nielsen’s heuristics are comprehensive and effective at identifying usability issues from a general perspective. They focus on core principles like error prevention, user control, and consistency. Inclusivity heuristics, on the other hand, are valuable for highlighting issues that might not affect experienced users but could significantly impact those with less technical background or different cognitive styles. They help reduce bias by encouraging evaluators to consider diverse user experiences.

5. In what ways was evaluating with Nielsens’s similar to the Inclusivity Heuristics? In what ways was it different?

Both sets of heuristics aim to improve usability and user experience, and they often overlap in areas like user control and feedback. However, Nielsen’s heuristics are more focused on general usability principles, while Inclusivity Heuristics push evaluators to consider how different

users, especially those with less experience or different needs, might interact with the software. Inclusivity Heuristics often involve using personas, which shifts the focus from “can this be done?” to “how difficult is this for someone unfamiliar?”

6. Any other takeaways or reflections?

Evaluating software through both sets of heuristics helped me realize how easy it is to overlook usability issues when you already know how a system works. Using personas and thinking from a beginner’s perspective revealed pain points I might have missed otherwise. It reinforced the importance of designing not just for power users, but for a broader, more inclusive audience.

7. How do your findings compare to those of your classmates?

My classmates and I had similar observations. Many of us noted that GitHub, like other developer tools, tends to cater to experienced users. It often uses specialized language and symbols that can alienate newcomers. This highlights a common trade-off in software design: balancing advanced functionality with accessibility.