PROCESSES SURVEY CODEBOOK

Q7: How do you monitor for merge conflicts?

Code	**Rule**	**Example**
Proactive	The developer monitors the repository for commits that might lead to a merge conflict with their own changes	*“[...] a feed plugin on my desktop which notifies me about commits on branches that I'm monitoring (I look for commits that might be troublesome when we integrate branches)”*
Reactive	The developer does not monitor for merge conflicts, or uses a process that only alerts them once a merge conflict has occurred	*“Github lets us know if a PR will cause a merge conflict”*

Q8: How do you determine the urgency of a merge conflict?

Code	**Rule**	**Example**
Project Structure	What part of the code is affected by the conflict determines the merge conflict	*“depends on the tests that are breaking, but core modules take precedence [...]”*
Code Under Conflict	The code that is conflicting, as well with it’s intent (bug fix, feature, etc) is used to determine urgency	*“Reading the code allows me to know what went wrong [...]”*
External dependencies	The urgency is dependent on the feature, fix or story that is impacted by the conflict.	*“Based on the severity of the open issue associated with a particular patch or branch update.”*
No system	The develop has no system to differentiate between conflicts; all conflicts are equally urgent	*“All merge conflicts are considered to be urgent and to be resolved as soon as possible.”*

Q11: What is your first step in trying to understand code involved in a merge conflict?

Code	**Rule**	**Example**
About the conflict	The developer starts by looking at the history of the changes that generated the conflict.	*“Reviewing the most recent commits (comments and code) to see whether its a shallow conflict or not.”*
The code itself	The developer starts by analyzing the code that is conflicting	*“Reading code involved”*

Analyzing the code	Looking at the larger picture; starting by seeing what part of the system is affected	*“Checking out that branch and running the tests to see which parts of the code are breaking”*
Design Concerns	The developer first tries to understand the design and intent of the code, before attempting a resolution.	*“Pull up related design docs to know what the code *should* be doing”*
Project Organization	The developer looks at the work that is done on the system/module before attempting a resolution	*“Opening all associated issues in Lighthouse to see where things are at.”*
No-op	The developer that not have a solidified process.	*“Don't know”*

Q14: What effect did deferring your response to a merge conflict have on the resolution of the conflict?

Code	**Rule**	**Example**
Physical Manifestation	The developer reported physical discomfort	*“Gave me a headache!”*
External to company impact	The effects were visible by customers due to a disruption of service	*“Broke the app for customers [...]”*
Policy/cultural changes	Policy or cultural changes were required because of the consequences	“Weekly reviews were less efficient because we had to spend time discussing the conflict before resolving.”
The Nuclear Option	The developers have to scrap the changes, and start again, because resolving the conflict was too complicated	*“KABOOM! [...] Nothing to do but wipe it all back to clean and start trying to piece things back together.”*
Increased Complexity	The deferral resulted in increased merge conflict complexity	*“The resolution becomes a spaghetti nightmare if we try to move forward without addressing it”*
Stop the presses	The developer process is stopped or slowed down until the conflict can be resolved	*“Delayed merging development lines until after we could get the dev team together to design a solution to the conflict zones.”*
No-op	No effects were observed	*“Open source is volunteer [sic] and no consequences for having to wait for fixes to come in”*

Q19: If your first attempt at resolving a merge conflict fails, what backup strategies do you use?

Code	**Rule**	**Example**
Collaborating	The developer collaborated with the other authors of the conflicting code to resolve the conflict.	*“Working directly with the author or team that last modified the area in conflict”*
Redoing changes	The developer reimplements their changes	*“Throwing away the code and starting again.”*
Take it offline	The developer tries to reorder the commits, in order to avoid the conflict	*“Rebase and reorder to fix the little bugs in how git trying to merge.”*
Try again	The developer tries the same strategy they used the first time	*“first attempt more carefully”*
No clue/other	Invalid responses	*“Not sure.”*