Results of Document Analysis Using Thematic Analysis Approach[1]

Initial codes	Higher order codes/categories	Emerging themes
I'd really prefer to work with somebody else "to more of a" look	overloaded maintainer(s)	potential users
sharing more work so that you can scale back sometimes	overloaded maintainer(s)	potential users
For a busy subsystem, can often be more than one person can handle.	overloaded maintainer(s)	potential users
By this time, Linux is no longer a hobbyist project, and after 21 years, it is probably time to focus more on scaling the maintainer role.	overloaded maintainer(s)	potential users
Maintainers are not keeping up with the kernel growth overall.	overloaded maintainer(s)	potential users
Most subsystems have unsustainable maintainer ratios.	overloaded maintainer(s)	potential users
cult of busy	overloaded maintainer(s)	potential users
Being a leader of a much bigger team makes maintainers very busy.	overloaded maintainer(s)	potential users
Community keeps growing, or your maintainer becomes otherwise busy with work&life.	overloaded maintainer(s)	potential users
Those maintainers lack reviewers.	overloaded maintainer(s)	potential users
You have your standard-issue overloaded bottleneck.	overloaded maintainer(s)	potential users
I'd argue that having a group would be substantially more robust.	the risk of single point of failure	potential users
(A single maintainer) is hard to prepare for disaster.	the risk of single point of failure	potential users
For a busy subsystem, can often be more than one person can handle.	the risk of single point of failure	potential users
Group models are also more robust in the face of vacations, illness, or simply a day job that gets busy.	the risk of single point of failure	potential users
He and Jani were becoming a bottleneck in the process.	the risk of single point of failure	potential users

Most maintainers are just that, a single person, and often responsible for a bunch of different areas in the kernel with corresponding different git branches.	the risk of single point of failure	potential users
You have your standard-issue overloaded bottleneck.	the risk of single point of failure	potential users
The maintainer as bottleneck.	the risk of single point of failure	potential users
bottleneck	the risk of single point of failure	potential users
Has contributed at least 25 patches.	having capable candidate committers	basic requirements
(Committers) should have submitted non-trivial patches.	having capable candidate committers	basic requirements
(The patches should) being merged	having capable candidate committers	basic requirements
(Committers) should have reviewed at least 25 patches.	having capable candidate committers	basic requirements
Committers have enough experience.	having capable candidate committers	basic requirements
A subsystem clearly needs a team of developers, and non-maintainer reviews must be the norm.	having capable candidate committers	basic requirements
(Committers) should not abuse of commit rights.	sharing trust among maintainers and candidate committers	basic requirements
Maintainers trust contributors.	sharing trust among maintainers and candidate committers	basic requirements
(Committers) should be regular contributors.	sharing trust among maintainers and candidate committers	basic requirements
Maintainers trust each other.	sharing trust among maintainers and candidate committers	basic requirements
Trust is obviously key within the group, no matter background/ employment/ representation.	sharing trust among maintainers and candidate committers	basic requirements
Trust relationships have to be built first.	sharing trust among maintainers and candidate committers	basic requirements

People you would trust enough to do it.	sharing trust among maintainers and candidate committers	basic requirements
particular if due to lack of trust	sharing trust among maintainers and candidate committers	basic requirements
He (maintainer) trusts his committers.	sharing trust among maintainers and candidate committers	basic requirements
It is a "human nature thing."	sharing trust among maintainers and candidate committers	basic requirements
The group should be consistent, with developers who stay around.	sharing trust among maintainers and candidate committers	basic requirements
Hardware for testing	sufficient precommit testing	necessary guarantees
We have clearly documented merge criteria.	sufficient precommit testing	necessary guarantees
We have massive CI, available to all contributors automatically.	sufficient precommit testing	necessary guarantees
mandatory in-depth testing way before committing	sufficient precommit testing	necessary guarantees
Good testing is crucial to this model.	sufficient precommit testing	necessary guarantees
A multi-committer tree can never be rebased, so there is no way to remove embarrassing mistakes.	sufficient precommit testing	necessary guarantees
Testing Requirements for drm/i915 Features and Patches	sufficient precommit testing	necessary guarantees
Have confidence in the patches you push.	strict review process	necessary guarantees
The confidence must be explicitly documented with special tags (Reviewed-by, Acked-by, Tested-by, Bugzilla, etc.) in the commit message.	strict review process	necessary guarantees
The complexity and impact are properties of the patch that must be justified in the commit message.	strict review process	necessary guarantees
One of those is mandatory review, no one is allowed to do anything solo.	strict review process	necessary guarantees

Especially around purported quality enforcement tools like code reviews.	strict review process	necessary guarantees
dim: drm inglorious maintainer script	applying tools to simplify work and avoid errors.	necessary guarantees
advanced commands for committers and maintainers	applying tools to simplify work and avoid errors.	necessary guarantees
Pipes stdin into the fixup patch file for the current drm-Ip merge. A branch can be explicitly specified to fix up a non-conflicIng tree that fails to build.	applying tools to simplify work and avoid errors.	necessary guarantees
This command adds the Link: tag (for patches that failed to apply directly).	applying tools to simplify work and avoid errors.	necessary guarantees
Any duplicates by name or email will be removed automatically.	applying tools to simplify work and avoid errors.	necessary guarantees
Using patchwork to facilitate review.	applying tools to simplify work and avoid errors.	necessary guarantees
quilt git flow script facilitates review.	applying tools to simplify work and avoid errors.	necessary guarantees
qf is a workflow script to manage a quilt patch pile on top of a git baseline and track any changes in git itself.	applying tools to simplify work and avoid errors.	necessary guarantees
This automaEcally either creates a new, empty patch pile or checks out the state of an exisEng remote.	applying tools to simplify work and avoid errors.	necessary guarantees
tooling	applying tools to simplify work and avoid errors.	necessary guarantees
purported quality enforcement tools	applying tools to simplify work and avoid errors.	necessary guarantees
tools are necessary	applying tools to simplify work and avoid errors.	necessary guarantees

When somebody makes a mistake, if possible, a check should be put into the tools to keep it from happening again.	applying tools to simplify work and avoid errors.	necessary guarantees

1. Soares C D , Dybå Tore. Recommended Steps for Thematic Synthesis in Software Engineering[C]// International Symposium on Empirical Software Engineering & Measurement. IEEE, 2011.