Why Quality Checking

Quality checking is why the work of a team becomes better than the skills of any one member of the team. It’s a critical part of software engineering (e.g. code reviews) and team work in general. It’s what takes the work of the team from being correct 80-90% of the time, to 99%+ of the time. And software won’t work without the parts being correct; for example, if there are 10 parts required for some functionality and 80% of them work, the software has a 10.7% chance of working (.8^10).

More generally, the software development process depends on many parts working from requirements to design, implementation, and testing; without a very high quality rate and correction process, errors accumulate very quickly with all the dependencies involved. This is because, as the number of components in a software project increases, the probability that all components work correctly (assuming each has a fixed probability of being correct) decreases exponentially. For example, if each component of a software project has a 90% chance of being correct, the probability that a project with 10 components is entirely correct is .9^10 = ~35%, for 20 components, .9^20=~12%, for 40, 1.5%, for 100, 0.0026%.This is a key root cause of most challenges faced in large-scale software development, where hundreds or thousands of components must interact seamlessly. And for each component, it must have had correct requirements, a correct specification, a correct design, and correct implementation; each piece of the software is the downstream result of all the steps of the software engineering process, each with many potential errors. This is why quality checking is critical. But how do we do it in a team at a larger scale?

How to do Quality Checking

Here is a Review / Quality checking policy you can directly adopt. Your policy needs to ensure 1) individual responsibility for each part of the quality checking, and 2) you can tell what has been quality checked or not by looking at the kanban. See the end of this document for concrete examples.

Review / Quality Check Policy:
All team members shall have their deliverable work reviewed by another team member. This quality checking process should be conducted by another team member familiar with the work (e.g. doing similar work on that part) for efficiency, and distributed across the team. This quality checking should be complete at least 24 hours before any final review of the whole deliverable done by the team (e.g. before a team deliverable review meeting). This includes reviewing code / pull requests.

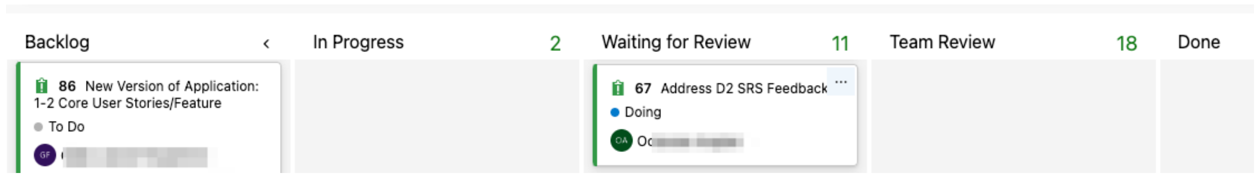
The person that did the task will assign a reviewer, by:
1) **Write a comment on the task @'ing a specific person to do the review** e.g. “@name for review:”

2) The comment also has either: a) a couple sentences/bullet points on what was done, or b) “see assigned comments” then add comments on the document on the relevant parts and assign the comments to the reviewer
3) **Check the task description in the kanban** has the rubric information and/or grading comments that were addressed, **so the reviewer knows what to check**. If it’s missing, add it.
4) **Move the kanban task to the Waiting for Review lane**

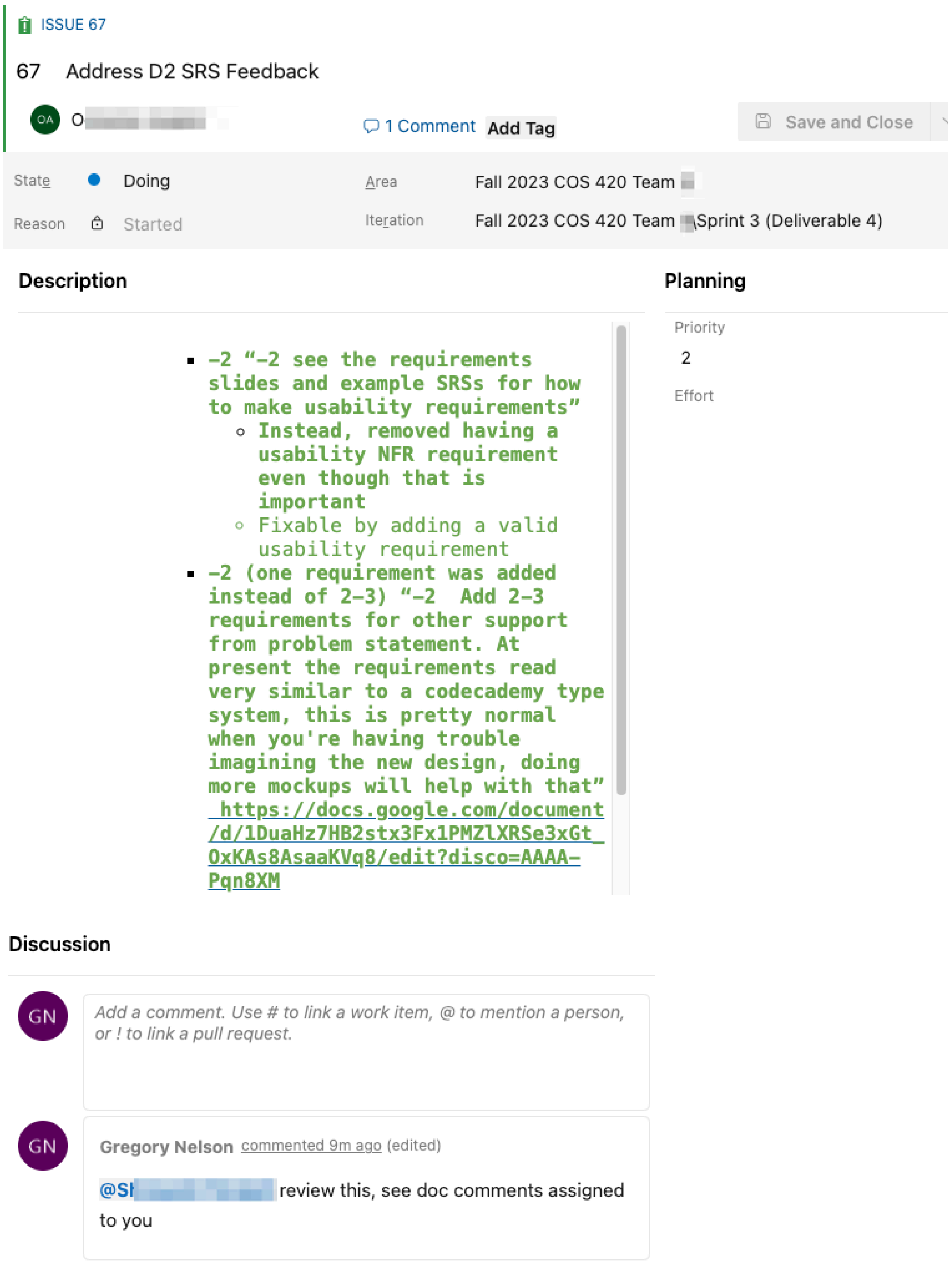
The reviewer will review / quality check by:
1) **Open task in the kanban to view the task guidelines and open relevant examples**
2) **Open the work and compare to the task description and examples for quality and completeness**
3) For revisions, looking at the edit history is very helpful to see what changed
4) Any questions about the reviewed work will be asked of the author in order to clarify any ambiguity or misconceptions, as comments on the kanban task or document
5) **Work that does not pass quality check will either be**: a) **fixed by the reviewer** if the work is understandable and fixes are minor, or b) **moved back to the In Progress lane and a comment added** with what is not done/needs to be fixed (or questions)
6) **Work that passes quality checks will be advanced on the kanban**; if the reviewer did work they should add a comment on the task with the rough percentage of the task they did/fixed

Be friendly to your team in writing your comments and assume they did they best they could with the knowledge they had at the time. If there is an issue with a person doing work that needs lots of revisions, figure out the root causes via discussion e.g. at the sprint retrospective (root causes might include: knowledge, not seeking clarification, not looking at the examples before doing the work, someone with better knowledge should have been assigned the task or provided mentorship, etc).

Here is an example:
The Kanban



Backlog	In Progress 2	Waiting for Review 11	Team Review 18	Done
86 New Version of Application: 1-2 Core User Stories/Feature ● To Do GF [redacted]		67 Address D2 SRS Feedback ● Doing OA Oc[redacted]		



ISSUE 67

67 Address D2 SRS Feedback

OA O[redacted]

1 Comment Add Tag

Save and Close

State ● Doing

Reason Started

Area Fall 2023 COS 420 Team [redacted]

Iteration Fall 2023 COS 420 Team [redacted]\Sprint 3 (Deliverable 4)

Description

- **-2 "-2 see the requirements slides and example SRSs for how to make usability requirements"**
  - **Instead, removed having a usability NFR requirement even though that is important**
  - Fixable by adding a valid usability requirement
- **-2 (one requirement was added instead of 2-3) "-2 Add 2-3 requirements for other support from problem statement. At present the requirements read very similar to a codecademy type system, this is pretty normal when you're having trouble imagining the new design, doing more mockups will help with that" https://docs.google.com/document/d/1DuaHz7HB2stx3Fx1PMZlXRSe3xGt_OxKAs8AsaaKVq8/edit?disco=AAAA-Pqn8XM**

Planning

Priority

2

Effort

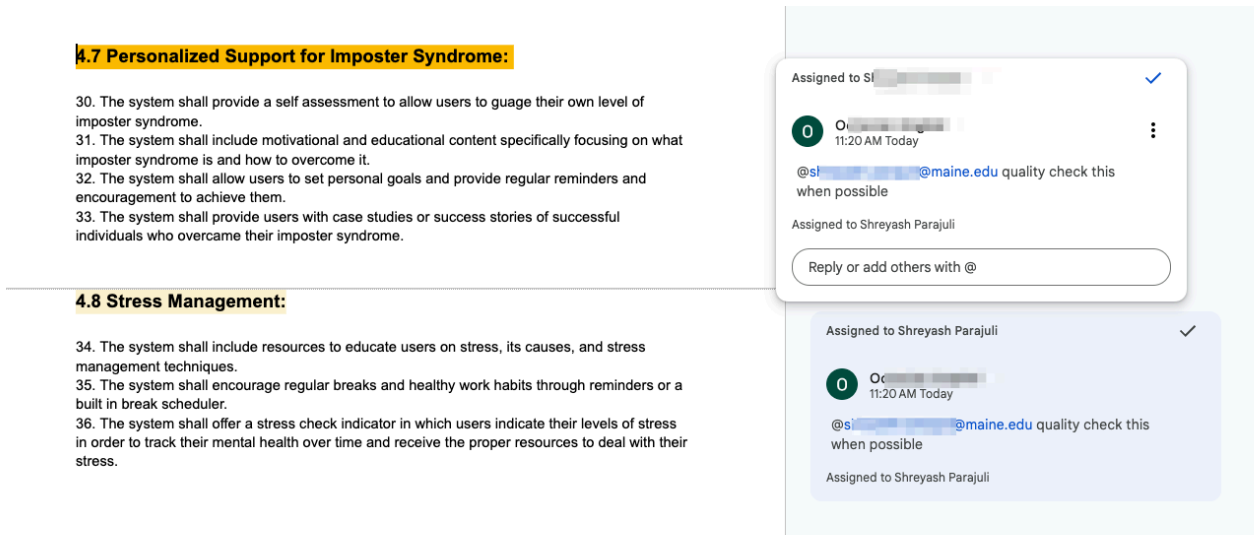
Discussion

GN *Add a comment. Use # to link a work item, @ to mention a person, or ! to link a pull request.*

GN **Gregory Nelson** commented 9m ago (edited)

@S[redacted] review this, see doc comments assigned to you

The SRS document:



4.7 Personalized Support for Imposter Syndrome:

30. The system shall provide a self assessment to allow users to guage their own level of imposter syndrome.
31. The system shall include motivational and educational content specifically focusing on what imposter syndrome is and how to overcome it.
32. The system shall allow users to set personal goals and provide regular reminders and encouragement to achieve them.
33. The system shall provide users with case studies or success stories of successful individuals who overcame their imposter syndrome.

4.8 Stress Management:

34. The system shall include resources to educate users on stress, its causes, and stress management techniques.
35. The system shall encourage regular breaks and healthy work habits through reminders or a built in break scheduler.
36. The system shall offer a stress check indicator in which users indicate their levels of stress in order to track their mental health over time and receive the proper resources to deal with their stress.

Assigned to Sh[redacted]

O[redacted]
11:20 AM Today

@sh[redacted]@maine.edu quality check this when possible

Assigned to Shreyash Parajuli

Reply or add others with @

Assigned to Shreyash Parajuli

O[redacted]
11:20 AM Today

@s[redacted]@maine.edu quality check this when possible

Assigned to Shreyash Parajuli

Why having people review other people's work is important

Fundamentally, quality checking trades some redundancy and loss in efficiency for a lower error rate.

If a person has a 90% chance of doing a task correctly and has no quality checking, there's a 10% chance the task has an error.

If a person has a 90% chance of doing a task correctly and the quality checker has a 90% chance of catching an error, there's only a 1% chance of an error getting through, i.e. 10% chance of an error, then 10% chance it gets missed. That's a huge improvement, a factor of 10x. And that's in the case when the quality checker has the same skill/chance of doing the task correctly. That's how the error rate of the team can become lower than that of any one member of the team.