Nexus Sprint Retrospective

Taken from Nexus Guide (January 2021)

The purpose of the Nexus Sprint Retrospective is to plan ways to increase quality and effectiveness across the whole Nexus. The Nexus inspects how the last Sprint went with regards to individuals, teams, interactions, processes, tools, and its Definition of Done. In addition to individual team improvements, the Scrum Teams' Sprint Retrospectives complement the Nexus Sprint Retrospective by using bottom-up intelligence to focus on issues that affect the Nexus as a whole.

The Nexus Sprint Retrospective concludes the Sprint.

Taken from Nexus Guide (January 2018)

The Nexus Sprint Retrospective is a formal opportunity for a Nexus to inspect and adapt itself and create a plan for improvements to be enacted during the next Sprint to ensure continuous improvement. The Nexus Sprint Retrospective occurs after the Nexus Sprint Review and prior to the next Nexus Sprint Planning.

It consists of three parts:

1. The first part is an opportunity for appropriate representatives from across a Nexus to meet and identify issues that have impacted more than a single team. The purpose is to make shared issues transparent to all Scrum Teams.

2. The second part consists of each Scrum Team holding their own Sprint Retrospective as described in the Scrum framework. They can use issues raised from the first part of the Nexus Retrospective as input to their team discussions. The individual Scrum Teams should form actions to address these issues during their individual Scrum Team Sprint Retrospectives.

3. The final, third part is an opportunity for appropriate representatives from the Scrum Teams to meet again and agree on how to visualize and track the identified actions. This allows the Nexus as a whole to adapt.

Because they are common scaling dysfunctions, every Retrospective should address the following subjects:

- Was any work left undone? Did the Nexus generate technical debt?
- Were all artifacts, particularly code, frequently (as often as every day) successfully integrated?
- Was the software successfully built, tested, and deployed often enough to prevent the overwhelming accumulation of unresolved dependencies?

For the questions above, address if necessary:

- Why did this happen?
- How can technical debt be undone?
- How can the recurrence be prevented?