**Part 1:**

The tool I chose to investigate for software inspections is Collaborator. This tool allows you to create and customize your own peer review template. Each template allows you to set rules, permissions and notifications which work in the background. [1] Planning and preparation for the inspection is done by customizing the review template, adding checklists or requirements for each step of the process, and then adding all documentation into the template to be reviewed. Collaborator also keeps a detailed audit log which shows whether a review is pending and the details of each review. [1]

The meeting or review is held through a threaded, contextual chat to comment asynchronously. [1] If you and another reviewer are in the tool at the same time, you may chat in real time as well. This enables software inspections to be done either in person or through a distributed team. Collaborator integrates with eleven of the most popular versioning systems including GitHub and allows you to save reviews to a zip file making documentation very flexible and easy. [1] All defects can be categorized and marked by severity and type. Reviewers may use the chat to mark and track these defects and with this tool, all defects must be fixed before the review can be completed. [1]

Overall, I would rate this tool as a 4.5 out of 5. Some businesses these days may also like to have the option to chat face-to-face rather than by typing. Voice-to-text could be implemented, to ensure that this is a viable option and that voice chats are recorded and documented as well. I would consider using this tool for software inspections.

**Part 2:**

The checklist I am reviewing contains 14 items that should be considered. These items are Readability/Understandability, Maintainability, Security, Speed and Performance, Documentation, Reinventing the Wheel, Reliability, Scalability, Reusability, Patterns, Test Coverage and Quality, Fit for Purpose, Notice What’s Missing, and Zoom Out. [2] Of these 14 items, the following paragraph describes which ones would be very difficult or impossible to assess using static analysis tools.

Readability is difficult for a static analysis tool to assess. It can be done be assessing how many times things like multiple inheritance or polymorphism are used which can make code difficult to understand. A tool however, may not accurately be able to assess the necessity of these code functions. Documentation is the determination of whether a new code requires additional information in the form of a Readme file or comment lines. [2] This is purely assessed based on the intent of the developer which a tool can’t determine. A static analysis tool is used mainly to detect bugs before running a program. This tool will only assess calculable items and patterns that can be recognized. Fit for Purpose is a checklist item that makes sure the code is accomplishing what it was intended to do. [2] A tool cannot analyze this because this item is purely based on the intent behind the code or feature and not whether it works or is optimized. There are other items that a static analysis tool might have difficulty assessing but, the last one we will discuss is Notice What Is Missing. The tool may only be able to assess the code that it is given so how could it possible assess for code that is not even there such as edge cases and error handling for unexpected inputs. [2]

**References:**

[1] Collaborator Team “Key Collaborator Features.” *SmartBear,* <https://smartbear.com/product/collaborator/features/>

[2] Postolovski, Tash. “Your Code Review Checklist: 14 Things to Include.” *Codementor,*

6 July. 2020, <https://www.codementor.io/blog/code-review-checklist-76q7ovkaqj>