On Jan 21 2016 the San Francisco Selenium Meetup Group hosted Brian Jordan from Code.org to discuss their continuous automated testing process in a meeting titled “Continuous Testing meets the Classroom”. Code.org is working to make the concept of software coding more broadly approachable to children and he specifically focuses on the “Hour of Code” program they make available to teachers and students.

Brian himself seems to be quite affable, but has a tendency to ramble a bit. Perhaps he is nervous in the presentation or perhaps he has competing ideas he wants to talk about all at once. Regardless, he feels quite knowledgeable in the Code.org process, even if he is a bit under prepared for publicly speaking about it.

The presentation starts with a general articulation of the challenges that the application encounters with the many browsers supported, the large array of languages included (including left to right rendering), and how in these browser environments occasionally have one-off conflicts with the technologies that the application employs. The technical discussion of the various moving pieces is at a high level and easily absorbed if you don’t get caught up in feeling as if you need to understand the modules. These elements are presented just to illustrate the complexity of the application and what needs to be considered when designing tests.

About 20 minutes into the presentation, we get to the meat of how on-going testing is achieved, starting with Selenium. In conjunction with Sauce Labs, Selenium is used to test functionality against the wide array of web browsers to ensure that new code changes have not broken previous (and new) features. Additionally, the problem of lengthy test cycles is addressed using the parallelization feature that Sauce Labs provides. Running many tests in parallel allows for more testing in a reduced time frame, and that is critical if they want to be able to publish updates on a daily basis.

However, Selenium is a somewhat limited testing tool, and Code.org had a desire to ensure that the visual continuity was included in their testing. Selenium isn’t able to readily measure if text has shifted or any other visual differences are rendered. For this reason, Applitools was included in the testing process. Applitools inspects the visual aspects of your rendered web page, and reports any variance form the established norm. It even goes so far as to allow excluded areas for elements that are known to change, like counters or a message of the day, so false failure reports can be reduced.

Overall, the discussion around the tools used for active pre-release testing is quite interesting and the fully rounded testing approach seems sound. I appreciate the details that Brian goes into to illustrate the problems and how the tools catch them.