GITHUB AT LINKEDIN

USABILITY RESEARCH FINDINGS AND HUMAN FACTORS RECOMMENDATIONS

FOUNDATION USER SUCCESS

WORK IN PROGRESS: LIVING DOCUMENT

This document enumerates some of our most recent findings (and respective recommendations) as we progress through our comprehensive usability evaluation of this tool. This is not a final deliverable, or a formal HFE work product.

PREPARED BY:

Elizabeth Frey
Staff Human Factors Engineer
Foundation User Success

WITH CONTRIBUTIONS FROM:

Branden R. Thompson
Senior Engineering Manager
User Success Manager PoC
Client Application Frameworks

REVISION MAY 2020

BOTTOM LINE UP FRONT (BLUF)

Topline Recommendations

The Foundation User Success team recommends that in the event of limited time and bandwidth, that the following recommendations/solutions are prioritized in order:

1. **CI-7** | BLOCKER | Failing CI jobs: jump directly to failure
2. **ACL-1** | BLOCKER | Add clarity around owner vs non-owner reviews
3. **ACL-3b** | BLOCKER | Improve clarity around "who to seek for a ship-it"
4. **ACL-3c** | BLOCKER | Ensure that files are listed in a "scannable" way
5. **CI-3** | BLOCKER | Post merge failed: add "failing" iconography
6. **ESP-2** | SHOULD HAVE | Option to surface custom checks to top level

7. **CI-5** | SHOULD HAVE | Post merge failed: post an additional comment...
8. **CI-6** | NICE TO HAVE | Auto-mege: PR controls vs. inert checkboxes

Scope of New Recommendations In This Version

For this study, Foundation User Success performed direct user interviews around the following GitHub integrations

- ACL App
- CI App
- Early Signal Platform (ESP)
- Policy App

In total, we worked with the GitHub migration team to define 29 hypotheses across these apps, the (in)validation of which will have a material consequence on important near-term design decisions.

ChangeLog (What we've reviewed)

- REVISION MAY 2020
  - Initial version

Hypotheses: What we are researching

LEGEND

Context
Human User/Actor Being Impacted
Thing being evaluated
Measurement and direction of movement

EXAMPLE:
When onboarding at the company, a new LinkedIn employee should feel more confident about where to look for acronym definitions.

Access Control Lists (ACL)

1. When looking at the pull request review screen (PR UI), users will be able to determine who to contact for a code review
2. Presenting "assignees" as required reviewers in addition to the required ACL approvals in the PR check status area, is not harmful in small projects, and is helpful in large projects.
3. Displaying the owners who have approved the PR in the check status area of the PR UI allows participants in the discussion to determine who has approved and whose approval is still needed to merge the PR.
4. When looking at the PR UI, users will be able to understand and articulate the conceptual difference between "assignees" and "reviewers
5. When looking at the PR UI, users will be able to disambiguate between reviews from ACL owners and non-owners
6. When looking at the PR UI, in the presence of a passing review from a non-owner and the absence of the requisite ship-its, users will understand why their code is not yet mergeable
7. When looking at the PR UI, users will find it self-explanatory that a *neutral* code review (neither an approval nor a request for changes) from a code owner is insufficient to meet ACL validation requirements
8. When looking at the PR UI, in the presence of all necessary ship-its from all required ACL owners, users will understand why their code is now mergeable
9. When looking at the PR UI, after receiving all necessary ship-its, upon pushing new unreviewed changes to the PR branch, users will find the feedback stating that a re-review is necessary sufficiently discoverable and actionable
10. Upon looking at the Check Details page for an individual ACL in a variety of scenarios, users will find the information presented there to be clear, valuable and actionable
11. Upon looking at the Check Details page for an individual ACL in a variety of scenarios, users will find the "Code Owners" section surfaces discoverable, intuitive and actionable information to the PR creator
    1. which ACLs are involved in a given PR
    2. who are all of the owners pertaining to a given ACL

12. In situations where multiple ACL approvals are required, users will find the feedback surfaced in the PR UI and checks pages clear, valuable and actionable
13. Users will find the summary text and additional details in the checks tab, sufficiently clear, detailed and actionable information about required ship-its
14. Users will find the "Required" indicator on the ACL check run to be a sufficiently clear indication that approval from owners is mandatory in order to merge the PR

Continuous Integration (CI)

15. When looking at the PR UI, users will be able to quickly and easily identify when a non-passing test execution is preventing their ability to merge
16. Users will largely favor and feel comfortable with having a critical-path wc-test-like PR validation check, providing they also have the ability to override it in a "break glass" situation
17. Users will understand the purpose and nature of the "branch validation" CI job
18. Users will be able to intuit how to find more detailed information about a branch validation CI status
19. Users will be able to discover the "post merge job started" comment posted to the discussion of their PR, and find it valuable
20. Users will be able to discover the "post merge job failed" update to the post-merge comment, and find it valuable and actionable
21. Users will find the level of detail to be "about right" in the "post merge" comments added to their PR discussion threads
22. Users will find the check box for controlling automatic merge of an approved, healthy and ready pull request to be sufficiently discoverable and intuitive
23. Users will find the concept of PR auto-merge intuitive
24. Users will find opt-in support for auto merge being allowed on a project and opt-in support on a per-PR basis to be a set of defaults that provides both the necessary safety and improved convenience relative to what we enjoy today
25. Users will not find the sole merge mechanism of "squash and merge" to be prohibitively limiting

Early Signal Platform (ESP)

26. Users will find the roll-up of all ESP checks into one signal on the PR conversation page (with details moved to a separate page) to be intuitive and sufficiently informative
27. For failed checks users will find it easy to see and understand what failed, and how to fix it
28. Users will find it easy to discover and understand which failing/pending ESP checks are blocking my merge (required) and which ones are informational

Policy App

29. Users will find the "unresolved comments" policy check, when blocking their merge, to be sufficiently discoverable and actionable

Methodology: How we Test

- Creating the Hypothesis
- Designing workflows to test the Hypothesis
- Identify/Validate User Cohorts associated with the hypothesis
- Creation of the interview script
- Conducting the User Interview
- Post-Interview Analysis
- Creation of Human Factors recommendations
- Hand-Off of Recommendations
- Post Hand-off Follow-up

Creating Hypotheses, Workflows and Stimulus

The GitHub migration team collaborated with Foundation User Success (FUS) in order to arrive at 29 hypotheses to test. FUS's guidance was to focus on areas where proving (or disproving) hypotheses would either surface new information about user priorities and preferences, or de-risk areas of the project by confirming alignment with the current work-in-progress solution.

FUS helped to ensure that all proposed hypotheses were conducive to an objectively-verifiable testing methodology. For example, investigating whether a given solution is "good" is hard to quantify,

but "simple" can be measured by avg. number of steps required to complete a task, "discoverable" can be measured by how long it takes a user to notice something, etc…

Hypotheses were then arranged into specific workflows that were used to form a repeatable testing procedure to be executed for each test subject. Finally, test stimuli in the form of static HTML were gathered for the purpose of repeating precise measurements on a fixed set of screens.

In order to preserve the state of the GitHub private instance for all interviews , Webrecorder was used to capture a 12.4MB archive of over 200 pages used in this study. Webrecorder player v1.8.0 was used while conducting user interviews, in order to provide a browser-like experience within a completely controlled environment.

Identifying User Cohorts

As hypotheses were formed, it became clear that behaviors, priorities and goals of users LinkedIn engineers vary depending on a number of factors. The GitHub migration team, in collaboration with FUS arrived at the following segments of users as being important to represent in the usability study panel:

- Contributors to apps, CLIs and other leaf-level dependencies
- Contributors to libraries
- Users new to LinkedIn infrastructure
- Users new to GitHub
- Multiproduct admins/owners (responsible for making decisions about MP-level options like auto-rollback)

Another cohort was identified, but omitted from the study due to the specific nature of the hypotheses to be tested

- Non-coding observers and contributors (product owner, manager, tech lead)

Screening Volunteer Participants

The GitHub migration team was responsible for ensuring that a sufficiently large pool of volunteer participants was made available for participation in the study. Each volunteer submitted a form with

a variety of questions, and their responses resulted in them being automatically "bucketed" into one or more cohorts.

Selecting Participants

Fourteen volunteers were selected such that we had no fewer than three participants inside and outside of each cohort. Additional adjustments were made to ensure that this group involved practitioners of a wide range of software engineering disciplines (i.e., iOS, web, CLI tools, APIs, etc…) and levels of experience.

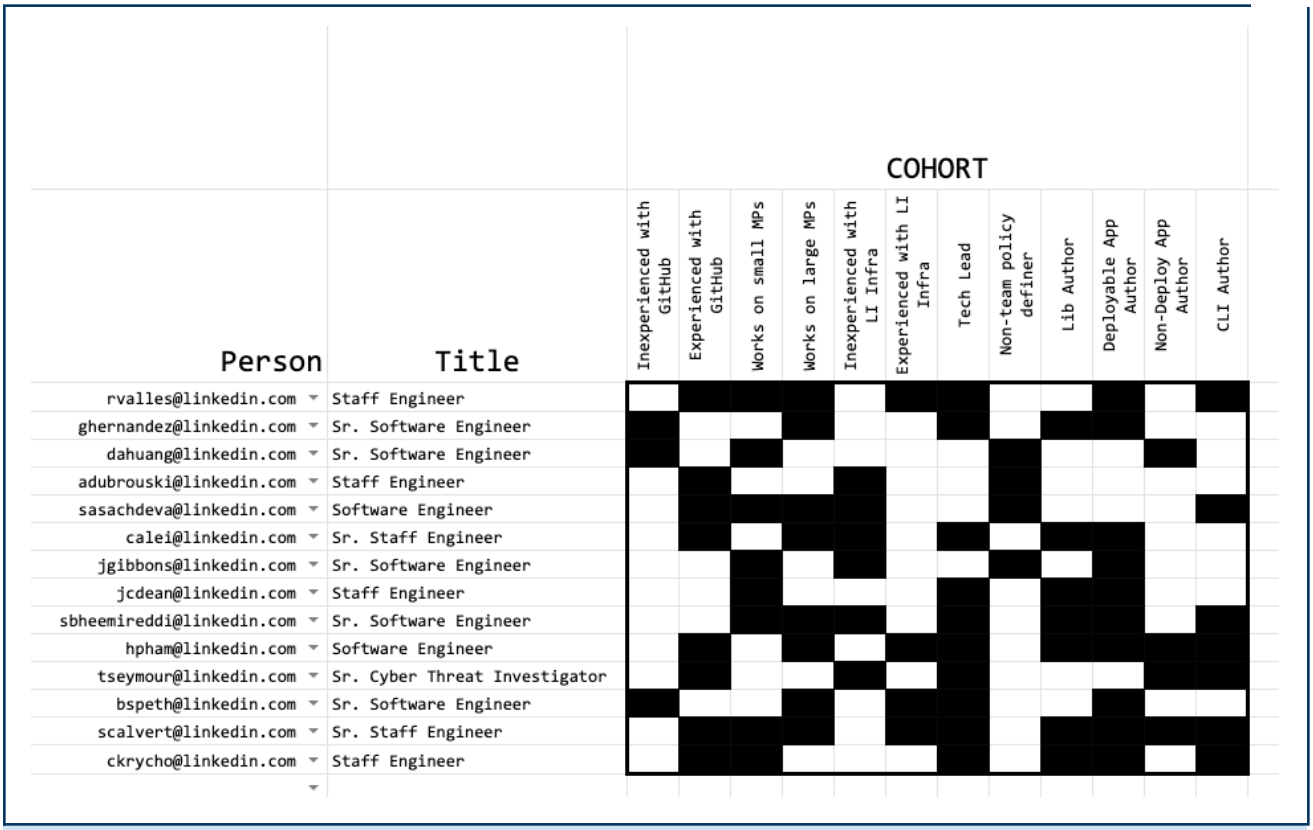

FIG 01: Panel participants. Black squares indicate person-to-cohort matches

Invitations were sent to this panel, and they set up their own appointments for a 60-minute one-on-one video call w/ an interviewer from Foundation User Success.

Conducting Interviews

Interviews began by thanking participants for their time, and explaining that they'd be shown a stimulus (webpage) and asked

questions. Interviewers asked permission to record the session, and made it clear the kind of feedback they should try to vocalize as much as possible ("please say what you're thinking and feeling").

For the remainder of the interview, users were presented with test stimuli (webpages) and questions were asked in order to evaluate each hypothesis. When possible, questions were formed using terminology used in the hypotheses themselves, so as to avoid affecting the experiment. When users asked questions of the interviewer, they were asked to make their best attempt to answer. In situations where a misunderstanding or misconception would interfere with the remainder of the interview, the interviewer provided the test subject with additional information.

Results: What We Found

Access Control Lists (ACL)

HYPOTHESIS (01)	CONDITIONALLY VALIDATED
When looking at the pull request review screen (PR UI), users will be able to determine who to contact for a code review	

During the course of the interview, test subjects were presented with several opportunities to indicate that they had clarity around, "who to ask for a ship-it."

- **CONDITIONS VALID**
  a. Users were highly likely to successfully identify required reviewers for their PR(s) when:
     i. A specific assignee was associated with it. (xx/yy)
     ii. H01-01b-i and the ACL validation failed. (xx/yy)
  b.
- **CONDITIONS INVALID**
  a. Users were highly unlikely to successfully identify required reviewers for their PR(s) when:
     i. {CONDITION / SCENARIO} (xx/yy)
     ii. {CONDITION / SCENARIO} (xx/yy)
- **ASSOCIATED RECOMMENDATIONS**
  a. ACL-4a | {PRIORITY} | {Title of Recommendation}
  b. ACL-4b | {PRIORITY} | {Title of Recommendation}

HYPOTHESIS (02)	VALIDATED

Presenting "assignees" as required reviewers in addition to the required ACL approvals in the PR check status area, is not harmful in small projects, and is helpful in large projects.

- When an assignee was present on a PR, users were significantly more likely (XX/YY) to be able to indicate who they'd have to ask for a ship-it.
- No users indicated that an assignee would be harmful, even conditionally.
- **ASSOCIATED RECOMMENDATIONS**
  a. ACL-## | {PRIORITY} | {Title of Recommendation}

HYPOTHESIS (03)	VALIDATED | IMPROVEMENT RECOMMENDED

Displaying the owners who have approved the PR in the check status area of the PR UI allows participants in the discussion to determine who has approved and whose approval is still needed to merge the PR.

This hypothesis was largely validated, but in discussing this with users, it's not the most interesting or important hypothesis in this area. Users who deal with large and more involved ACL situations (including 100% of participants who regularly contribute to a voyager MP) pointed out that knowing **which ship-its are still outstanding and how to obtain them** is the main problem to solve, and there is some significant opportunity for improvement in this area.

HYPOTHESIS (04)	INCONCLUSIVE | IMPROVEMENT RECOMMENDED

When looking at the PR UI, users will be able to understand and articulate the conceptual difference between "assignees" and "reviewers

HYPOTHESIS (05)	INCONCLUSIVE | IMPROVEMENT RECOMMENDED

When looking at the PR UI, users will be able to disambiguate between reviews from ACL owners and non-owners

Nearly all users correctly understood what "assignee" means in the context of a pull request.

In the general case, users less experienced with GitHub did not provide a consistent signal about what a "reviewer" means, compared to an "assignee". In the context where the reviewer *completed* their review, these users were largely able to reason their way to "someone who performed a code review" but could not speak to how that translated into ability to merge the PR and ACL approvals.

Users already experienced with GitHub universally pointed out that it was not clear whether a "reviewer" who is not an "assignee" is a non-assigned ACL owner, or a non-owner. It's important to be clear about this because in one situation a ship-it will affect the ACL validation requirement, and in the other it won't.

These hypotheses are marked as "inconclusive" because they were validated in some important scenarios and invalidated in others.

HYPOTHESIS (06)	**VALIDATED**
When looking at the PR UI, in the presence of a passing review from a non-owner and the absence of the requisite ship-its, users will understand why their code is not yet mergeable	

Irrespective of level of familiarity with GitHub, users seemed to recognize "a positive review that didn't make the ACLs happy" as a situation where the reviewer is not a listed ACL owner.

HYPOTHESIS (07)	**VALIDATED | IMPROVEMENT RECOMMENDED**
When looking at the PR UI, users will find it self-explanatory that a *neutral* code review (neither an approval nor a request for changes) from a code owner is insufficient to meet ACL validation requirements	

The way users responded to the situation that tests this hypothesis seemed to correlate with level of experience with GitHub. In RB, it's rare to see a "neutral" code review, although with a "general comment" it seems to be possible. The main opportunity for improvement seems to be a more informative CheckRun summary for the pertinent ACL CheckRun.

HYPOTHESIS (08)	VALIDATED
When looking at the PR UI, in the presence of all necessary ship-its from all required ACL owners, users will understand why their code is now mergeable	

Users were broadly able to figure this out. In general "no action required" feedback is unlikely to be a point of friction. Users were asked "why is your code able to be merged now" and they largely answered the "why" correctly. One exception was a Cyber Threat Investigator who is unfamiliar with our ACL system, who was not able to articulate much about ACL approvals in general.

HYPOTHESIS (09)	INCONCLUSIVE | NEEDS RE-TESTING
When looking at the PR UI, after receiving all necessary ship-its, upon pushing new unreviewed changes to the PR branch, users will find the feedback stating that a re-review is necessary sufficiently discoverable and actionable	

There was an error in the screen we needed for this testing scenario, and the problem was discovered upon using it in real interviews. Users had to be asked to "use their imagination" to make a specific correction before giving feedback, and this involved drawing their attention to specific areas where we were interested in learning about discoverability.

This should be re-tested in the next round of user interviews.

HYPOTHESIS (010)	INVALIDATED

Upon looking at the Check Details page for an individual ACL in a variety of scenarios, users will find the information presented there to be clear, valuable and actionable

While users universally felt that this was a substantial improvement relative to RB, it's clear that there's a lot of room for improvement as well. Users pointed out potentially conflicting, poorly-formatted and low-value information, and made it clear that they are looking for clear and simple answers to their biggest needs on a per-scenario basis.

Some specific opportunities for improvement:

- No user was happy seeing a full list of ACL owners in the "code owners" table, and a subset of these users in the 'please ask these people for a ship-it' notice near the top of the screen.
    - It's not clear how this subset of users was chosen
    - On two occasions when a user was probing into details about their ACL approval status, they had a ship-it but the information surfaced to them made them feel that it was a non-owner review. This actively misled them
- The "approval status" information starting in a collapsed state *almost guarantees* that an extra click will be required
- The formatting of the "approval status" table was not received well (although some of the information contained within it was appreciated)

HYPOTHESIS (011)	**INVALIDATED**

Upon looking at the Check Details page for an individual ACL in a variety of scenarios, users will find the "Code Owners" section surfaces discoverable, intuitive and actionable information to the PR creator

a. which ACLs are involved in a given PR
b. who are all of the owners pertaining to a given ACL

Users almost universally found the "Code Owners" table to be highly overlapping (in terms of purpose) with the "Approval Status" table.

Some of the biggest opportunities for improvement:

- File paths for most realistic use cases will be long enough to cause text to wrap within the table cells they're currently rendered in. Wrapped lines are hard to scan.
- Centering the text alignment in "Approval Status" and "code owners" tables makes file paths even harder to scan
- A flat list of file paths contains a lot of redundant information. Alternate formats should be explored

HYPOTHESIS (012)	VALIDATED
In situations where multiple ACL approvals are required, users will find the feedback surfaced in the PR UI and checks pages clear, valuable and actionable	

All users who were familiar with the idea of multiple ACLs found this to be an intuitive way to present the right level of information.

HYPOTHESIS (013)	INCONCLUSIVE | IMPROVEMENT RECOMMENDED
Users will find the summary text and additional details in the checks tab, sufficiently clear, detailed and actionable information about required ship-its	

Users found the summary strings in simple situations to be intuitive, but are looking for improvement around the following scenarios

- Re-review
- No review yet
- Neutral review
- Non-owner review

HYPOTHESIS (014)	VALIDATED | NEEDS RE-TESTING
Users will find the "Required" indicator on the ACL check run to be a sufficiently clear indication that approval from owners is mandatory in order to merge the PR	

Nobody was confused by ACLs being a mandatory requirement, but this should be re-tested in a situation involving a truly optional (and

likely failing) PR check of some sort. This additional scenario will help determine whether users understand "Required", or whether they think *all checks are required.* The data we have today does not shed any light on the difference.

Continuous Integration (CI)

HYPOTHESIS (015)	VALIDATED
When looking at the PR UI, users will be able to quickly and easily identify when a non-passing test execution is preventing their ability to merge	

Aside from those who are unfamiliar with MP development, all users were able to discover and understand "pending" and "failing" CI job states. One user did recommend surfacing "elapsed vs estimated time" being surfaced in the summary string for the Branch Validation check.

HYPOTHESIS (016)	BORDERLINE
Users will largely favor and feel comfortable with having a critical-path wc-test-like PR validation check, providing they also have the ability to override it in a "break glass" situation	

From a philosophical standpoint, all users were comfortable with automated testing as a mandatory step on the critical path to bringing a proposed code change into master.

Several (3) users who work on voyager MPs raised concerns about the reliability of kibitzer-initiated wc-tests, and "flaky" test suites. Today, one possible workaround in the face of a problem like this is to run `mint wc-test` locally, and post a URL to the successful execution in the RB discussion. Of these three users, two of them felt that mandatory automated tests would motivate *solving the root cause* of this flakiness, and one felt that requiring tests to pass would be harmful to productivity.

Because there's a split between the two parts of this hypothesis (all favor it in an ideal world, but fewer "feel comfortable" with it

based on their feelings about infra stability) this is marked as "borderline".

HYPOTHESIS (017)	BORDERLINE
Users will understand the purpose and nature of the "branch validation" CI job	

~60% of users interviewed correctly articulated the purpose of the branch validation job, and the commit being tested. Other users were confused about how "branch validation" might be related to

- Checking whether the branch could be merged into master
- `mint validate`

Because this hypothesis does not have a specific threshold for "success", and because a non-trivial number of users were confused, it is marked as "borderline".

HYPOTHESIS (018)	VALIDATED
Users will be able to intuit how to find more detailed information about a branch validation CI status	

All users were able to successfully drill into details about the CI job

HYPOTHESIS (019)	VALIDATED
Users will be able to discover the "post merge job started" comment posted to the discussion of their PR, and find it valuable	

None of the users in the panel had trouble finding the comment in the discussion. Most found the comment to be valuable, but additional probing should be performed in this area to involve the notifications experience (i.e., is it right to send everyone an email when this job begins?) and *relative* value of this message, compared to waiting for a post-merge outcome of some sort before reporting back with information.

HYPOTHESIS (020)	INVALIDATED
Users will be able to discover the "post merge job failed" update to the post-merge comment, and find it valuable and actionable	

While users were able to discover this update, its discoverability was called into question by those most knowledgeable about GitHub, given that this is an *update* to a previous message and thus *no notification will be sent*. This signal is vastly more important relative to "the post-merge job has started".

User feedback revealed at least two more opportunities for improvement:

- The only emoji on this message is 🎉 , which suggests "success" in spite of this message being an "action required" situation
- Several users asked if information about the nature of the failure could be surfaced into the message itself

HYPOTHESIS (021)	INVALIDATED
Users will find the level of detail to be "about right" in the "post merge" comments added to their PR discussion threads	

Most users found the level of detail to be "about right" for "the job has started" and "the job succeeded", but the majority of users felt that detail was insufficient for the "the job failed" case.

HYPOTHESIS (022)	BORDERLINE
Users will find the check box for controlling automatic merge of an approved, healthy and ready pull request to be sufficiently discoverable and intuitive	

A narrow majority of users were able to spot the checkbox. Those who did felt it to be an adequate mechanism for controlling automatic merge of pull requests.

Users who needed help (hints from the interviewer) to find the checkbox had a mix of feedback

- Some of them did not realize that the checkboxes were interactive
- Some of them had seen checkboxes earlier in the study, and did not intuit that *this* checkbox had some side effect of consequence
- Some of them hadn't noticed it at all

There seems to be a correlation between familiarity with GitHub and being able to figure out how to operate this UI.

HYPOTHESIS (023)	**VALIDATED**
Users will find the concept of PR auto-merge intuitive	

The way this was explained to users was the following statement

> *We are considering a feature called "Pull Request Auto-Merge" whereby a healthy PR, once it receives all necessary approvals and passes all automated tests may be automatically merged into master by a bot. We're interested in your opinions around how often you can see yourself using this, and how you feel about opt-in vs opt-out on a per-PR or per-Multiproduct basis.*

All user responses indicated that they were able to understand the concept correctly, although their opinions around preferred defaults and safety varied.

HYPOTHESIS (024)	**VALIDATED**
Users will find opt-in support for auto merge being allowed on a project and opt-in support on a per-PR basis to be a set of defaults that provides both the necessary safety and improved convenience relative to what we enjoy today	

About 30% of users felt uncomfortable at the idea of auto-merging being allowed at all for certain Multiproducts. Concerns around control and safety seemed to primarily come from the *maintainer (ACL*

owner) standpoint, and they appreciated the ability to disable this entirely for certain projects and/or situations.

FUS advises that this question be posed again in a future study, with the added context of automated dependency upgrades.

HYPOTHESIS (025)	VALIDATED
Users will not find the sole merge mechanism of "squash and merge" to be prohibitively limiting	

Less than 30% of users had any opinion on this. Those who did correctly identified "squash and merge" as the mechanism most conducive to a clean and linear history on trunk, and the closest equivalent to LinkedIn's existing code review workflow.

Early Signal Platform (ESP)

HYPOTHESIS (026)	BORDERLINE
Users will find the roll-up of all ESP checks into one signal on the PR conversation page (with details moved to a separate page) to be intuitive and sufficiently informative	

Many of the users in this study had no experience working in GitHub repos with a large number of independent Checks applied to each PR. Upon asking "how many checks is too many to list at the bottom of the conversation tab", we got answers varying from 6 to 20. It seems that tastes range widely here.

While in some cases, users were comfortable with many validations being rolled up into one Check, some users spoke about cases that they preferred to be "promoted" to their own top-level status.

For example, one of the voyager-web engineers we spoke with felt that there are certain tools (some existing, some in progress) whose signals around performance and craftsmanship were on-par, in terms of importance, with "you have passed your tests". Moving a signal this

important into a collection of other statuses felt (to this user) like "burying the lede"

Another use case that two users pointed out was tooling that measured code quality or documentation coverage. While these measurements may never *fail*, in some contexts they're sufficiently important to surface along-side the other top-level checks

Almost universally, the label "custom validations" was not sufficiently informative -- particularly given that our test stimulus included `mint validate`, which should be a required (non-custom) validation of some sort.

HYPOTHESIS (027)	**VALIDATED**
For failed checks users will find it easy to see and understand what failed, and how to fix it	

All users were able to figure out how to drill in for more information, and they *really* appreciated having direct links to the specific console output. Upon seeing how this was presented, several users expressed an interest in seeing this same kind of "drill right into the problem area" treatment applied to CI (i.e., "jump to failing child execution log")

HYPOTHESIS (028)	**VALIDATED**
Users will find it easy to discover and understand which failing/pending ESP checks are blocking my merge (required) and which ones are informational	

No user had problems with this.

Policy Bot

HYPOTHESIS (029)	**VALIDATED**

> Users will find the "unresolved comments" policy check, when blocking their merge, to be sufficiently discoverable and actionable

No users had major problems with this, but interviews did draw attention to some potentially confusing areas.

Recommendations

General

Access Control Lists (ACL)

1. **Improvements around 'owner vs. reviewer' ergonomics**

   Results from testing hypothesis 04 and 05 revealed some potential ambiguities between roles and responsibilities of discussion participants listed as "reviewers" on the right sidebar of the PR UI.

   In particular, a "reviewer" not listed as an "assignee" could be an ACL owner or not. This is very important information to surface in a reasonable way, because "ownership" determines whether an approving review counts as part of the ACL validation requirement.

   RECOMMENDATION
   FUS recommends...
   - ACL owners who provide a review signal of any kind be *added as assignees* to the PR in question. Effectively, an additional owner providing a review can be thought of as *sharing or taking over responsibility* to review the code in question.
   - ACL check details should include *all review signals*, so the user is not left trying to deduce the meaning of a review signal by looking at the ACL validation check. This could be represented as some sort of table like the one shown below

Reviewer	Own	Approval	Commit	Most Significant Signal
@user1		✅	`fcb1c43`	
@user2		❌	`fcb1c43`	
@user3	✓	✅	`fcb1c43`	✓

FIG 02: Code review signal table, including non-owner reviews

2. **Improved ACL check summary for "neutral" reviews**
   Some users less familiar with GitHub were confused by a "neutral" review (found while testing Hypothesis 07). The current implementation treats this scenario as identical to having no review feedback at all.

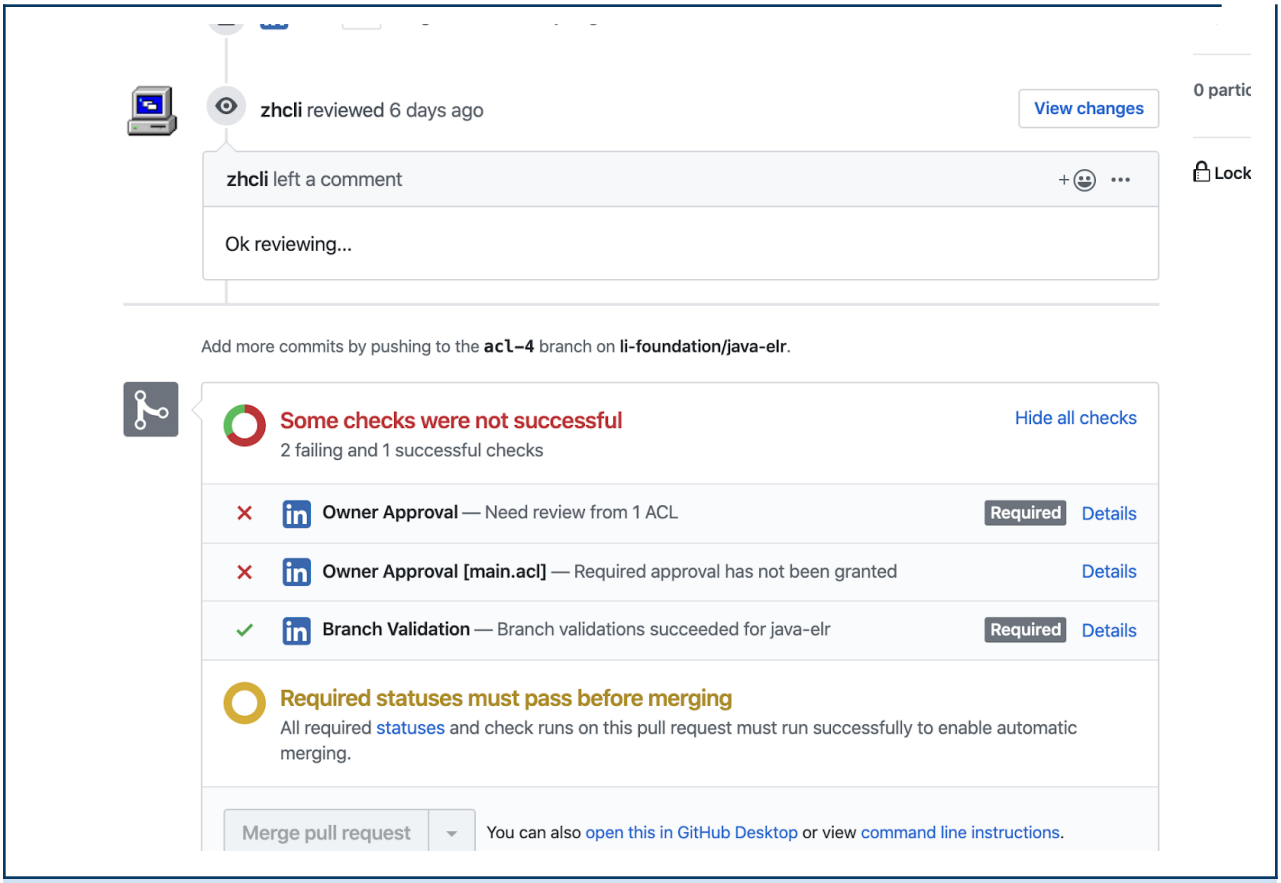

FIG 03: Owner Approval summaries for a "neutral" review

RECOMMENDATION

FUS recommends more complete and explicit ACL validation feedback in general. Specifically in the "neutral" case, there should be clear feedback indicating that the review exists, but does not include an approval to merge

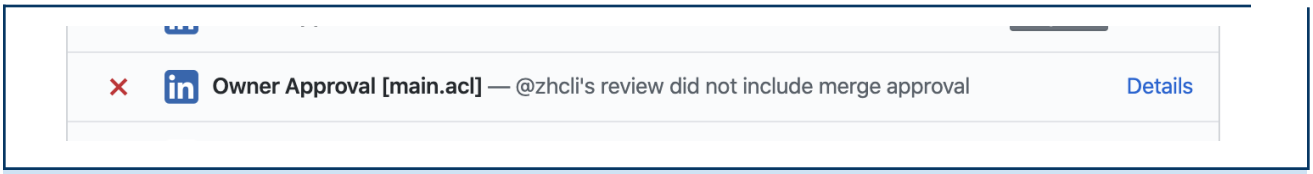

FIG 04: A specific "neutral" review summary string

3. ACL check details

Results from testing hypotheses 10 revealed several opportunities for improvement to the ACL check details page.

a. Improve clarity around "who to seek for a ship-it"

Several users, upon seeing the "Here are some users that can approve this ACL file" (marked with "A" in the figure below) were confused about why a *subset of owners* were listed in one place, and the full list of owners in another (marked as "E").

RECOMMENDATION

FUS recommends that either all owners be listed (preferred), or an *extremely high degree of clarity* be provided around any subset presented to users in place of the full list. Users unanimously agreed that the most important purpose of this page is answering the question: "who do I need approval from in order to merge this?" -- any increased alignment with that goal would likely be appreciated

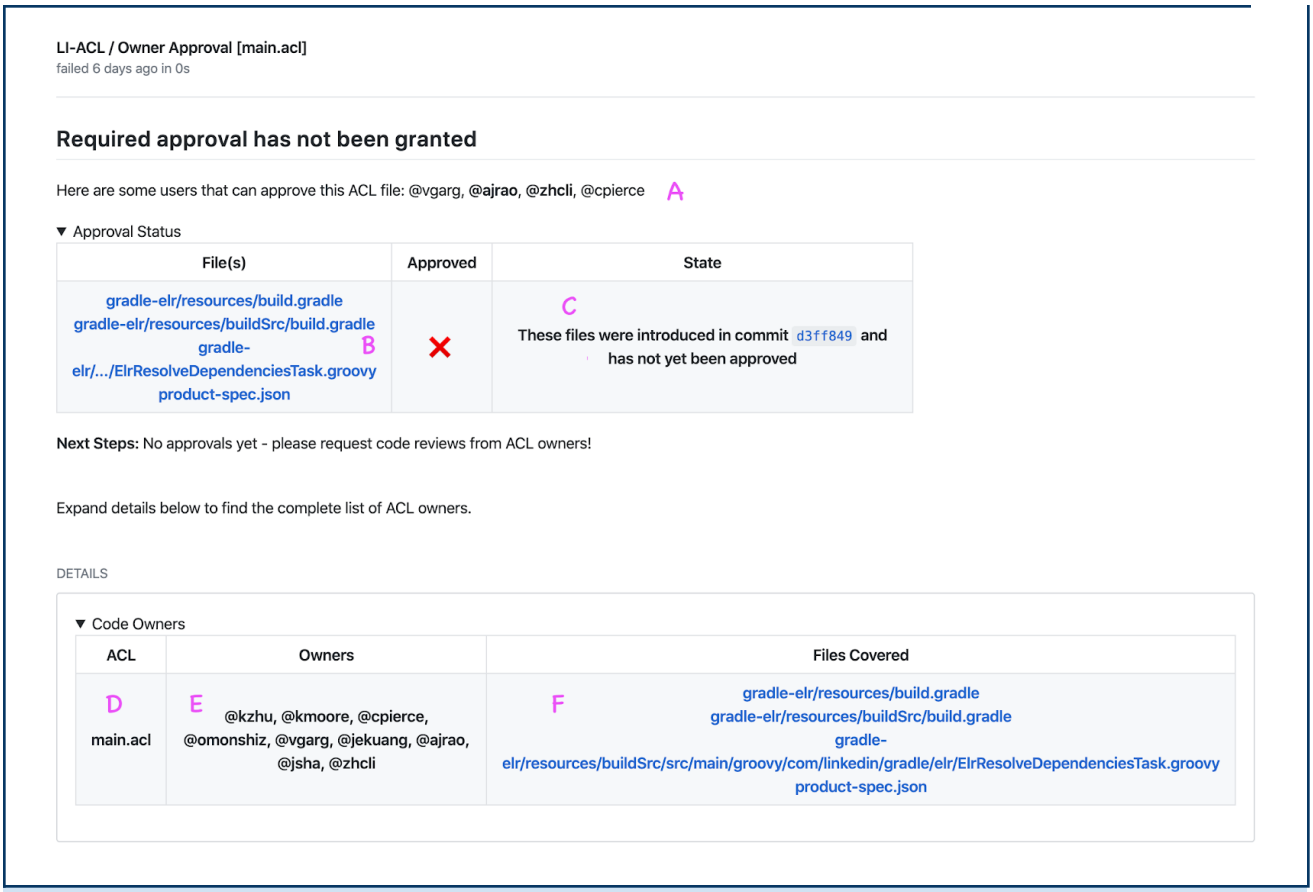

FIG 05: ACL check details when no approval has been given yet

b. Resolve redundancy between "approval status" and "code owners" tables

All users who probed into details enough to discover the "Code Owners" table felt that the information it provides is highly redundant with the "Approval Status" table above. This effectively invalidated hypothesis 11.

RECOMMENDATION

Combine these two UI elements into a single area. Further recommendations TBD.

c. Ensure that files are listed in a "scannable" way

All users who probed deeply enough into the UI to discover the "Approval Status" or "Code Owners" tables expressed displeasure at how the files were formatted. FUS recommends rendering this list of files in a way that can be easily scanned vertically.

One option is to left-align the text. As the figure below shows, this becomes easier to vertically scan, but still includes a large amount of redundancy (creating a signal-to-noise ratio problem)

```
01 gradle-elr/resources/buildSrc/src/main/groovy/com/linkedin/gradle/elr/ElrReso
   lveDependenciesTask.groovy
02 gradle-elr/resources/buildSrc/src/main/groovy/com/linkedin/gradle/elr/Publish
03 ArtifactsPlugin.groovy
04 gradle-elr/resources/buildSrc/src/main/groovy/com/linkedin/gradle/elr/helper/
050 utils.groovy
6  gradle-elr/resources/buildSrc/src/main/groovy/com/linkedin/gradle/elr/model/E
   lrData.groovy
07 gradle-elr/resources/buildSrc/src/main/groovy/com/linkedin/gradle/elr/model/E
08 lrDependency.groovy
09 gradle-elr/resources/buildSrc/src/test/groovy/com/linkedin/gradle/elr/Publish
   ArtifactsPluginTest.groovy
   gradle-elr/resources/buildSrc/src/test/resources/scala/expected_elr-details.j
   son
   gradle-elr/resources/buildSrc/src/test/resources/structure-base/com/google/co
   de/findbugs/jsr305/1.3.9/jsr305-1.3.9.ivy
   gradle-elr/resources/buildSrc/src/test/resources/structure-base/com/google/co
   llections/google-collections/1.0/google-collections-1.0.ivy
```

FIG 06: Left-aligned file paths

This noise can be reduced through organizing these paths into a tree, where single-child paths are collapsed into a single node

```
gradle-elr/resources/buildSrc/src/
├─ main/groovy/com/linkedin/gradle/elr/
│  ├─ ElrResolveDependenciesTask.groovy
│  ├─ PublishArtifactsPlugin.groovy
│  ├─ helper/utils.groovy
│  └─ model/
│     ├─ ElrData.groovy
│     └─ ElrDependency.groovy
└─ test
   ├─ groovy/com/linkedin/gradle/elr/PublishArtifactsPluginTest.groovy
   └─ resources/
      ├─ scala/expected_elr-details.json
      └─ structure-base/
         └─ com/google/
            ├─ code/findbugs/jsr305/1.3.9/jsr305-1.3.9.ivy
            └─
collections/google-collections/1.0/google-collections-1.0.ivy
```

FIG 07: File paths arranged into a tree

Rendering file path information in general should be considered carefully (several users mentioned that they only see value in this information *when it's actionable*) but if it must be done, the ability to identify where in the project changes are made "at a glance" should be a high priority.

d. Simpler signals in some scenarios

RECOMMENDATION

When no review signal has been received yet, we can provide some much simpler feedback to users such as

No review from an owner listed in `main.acl` has been completed yet. Please reach out to one of them for a code review

`main.acl` owners:
@user1
@user2
....

Users felt that everything else displayed on the page had low value in the "no reviews yet" state

RECOMMENDATION
When a re-review is required, the most important things to surface to the user are

- which commits (and files within those commits) invalidated ship-its
- who to reach out to for a re-review

This information should not require any clicking in order to reveal (expandable sections should begin OPEN)

A few users also asked for some easy ability to ping someone on slack (either via copy/paste of some pre-generated message, or a button that would automatically ping a reviewer. This should be considered carefully, as over-calibrating toward automated begging for ship-its may not be the right way to solve this problem.

4. **Surface more actionable information in Check summary strings**
   Detailed guidance TBD. Related hypothesis: 013.

5. **Provide more useful information in the "parent" ACL check details**

   The current ACL "parent" status that represents the overall ACL validation state is shown in FIG 08.

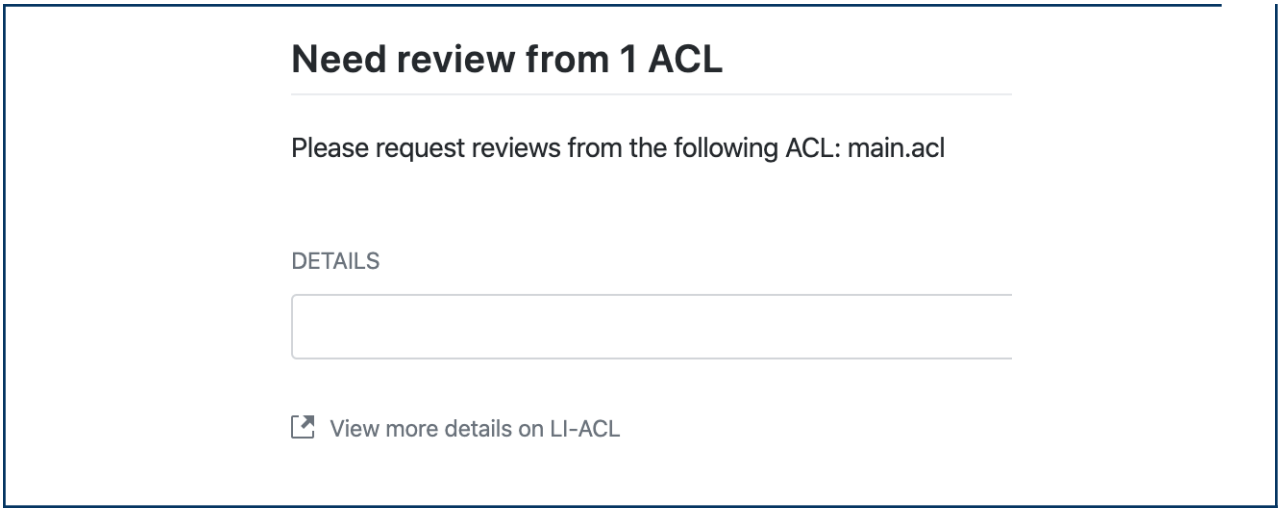

FIG 08: The current top-level ACL check details page (example)

Many users reached this page, when faced with notification that they hadn't received their ship-its, and expressed a "50/50 guess" at which details link was best to try first

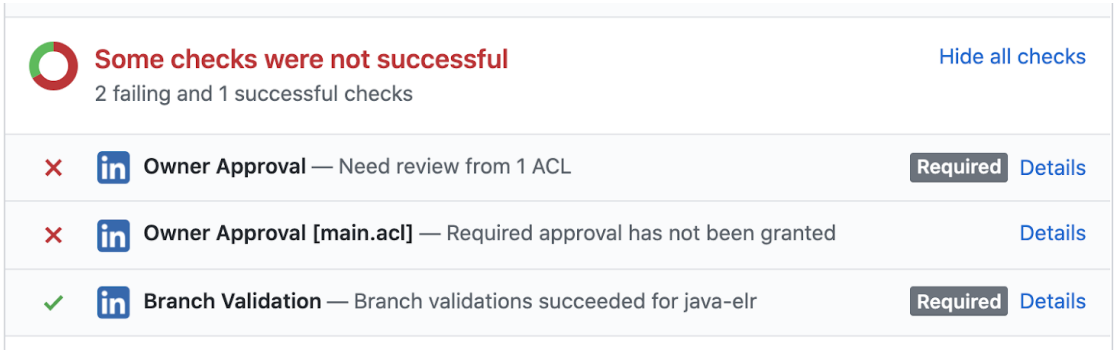

FIG 09: Multiple Owner Approval "details" links to choose from

RECOMMENDATION
We should present a summary of ACL validation across all ACLs on this page, rather than having users drill in without helping them out at all.

For example, something like the following might work well:

```
ACL         STATE
--------    -----------------
main.acl    approved by @user (link to review)
docs.acl    no review yet (link to details)
```

If users have to "roll the dice" and pick one of the two links to click on, let's set them up to never lose, by providing them with actionable information in both places (but possibly at different levels of detail)

Continuous Integration (CI)

1. **Provide an OVERRIDE mechanism to bypass branch validation requirement**

   While there seems to be near universal support for branch validation being placed on the critical path to being able to

merge code, users who primarily work on large MPs expressed concerns that this would almost certainly impede velocity in the short term. It's for this reason alone that hypothesis 16 is marked as "borderline".

RECOMMENDATION
To address concerns about being able to get around flaky tests and infrastructure issues by posting a link to a locally-run `mint wc-test`, a "break glass" tool should be provided, similar to PCXVALIDATIONOVERRIDE.

2. **Rename branch validation check to improve clarity**

While about half of users understood "branch validation" to be equivalent to the kibitzer-initiated wc-test present in LinkedIn's current development workflow, a significant number of users were confused. This is why hypothesis 17 was marked as "borderline".

RECOMMENDATION
Clarity is important, particularly while users are being asked to make some non-trivial adjustments to their mental model. Please consider using a simple and self-descriptive label like:

- Build & Test PR
- PR tests
- `mint test`

To avoid the most common misinterpretations ("checking for whether there will be a git conflict", "mint validate")

3. **Post merge failed: add "failing" iconography**

Hypothesis 020 was invalidated, in part due to the "post-merge failed" message not being presented as a clear failure.

RECOMMENDATION
Add some 🚨 or ❌ or 🛑 that attracts the user's attention

4. **Post merge failed: surface more information in the comment**

Hypothesis 021 was invalidated, because users wanted to see more information directly embedded in the comment added to the PR discussion when a post-merge job fails.

RECOMMENDATION
Add the following information to the "post merge failed" content

- How many executions and sub-executions failed
- Direct links to jump straight to any failed executions
- Failure classification

5. **Post merge failed: post an additional comment instead of editing an existing one**

   On GitHub, edits to comments do not result in notifications being sent to participants in the PR discussion, but creation of a new comment does. This is part of why Hypothesis 020 was invalidated.

   RECOMMENDATION
   Since a post-merge failure is an *exceptional event* that is *highly likely to be important enough to demand the attention of the author*, a new comment should be added to the discussion with the failure information, rather than performing an edit to the existing "post-merge job has started" comment.
6. **Auto-mege: clear delineation between PR controls, and inert checkboxes**

   It's too easy to get confused between checkboxes in a PR description that represent the author's TODO list, and checkboxes that control merge behavior. This is part of why Hypothesis 022 was marked as "borderline".

   RECOMMENDATION
   Use ASCII line drawings to create a clear "UI panel", and place controls that are machine-parsed in that designated area

FIG 09: A UI panel created with ASCII box drawing characters

7. Failing CI jobs: jump directly to failure

Although not directly part of a hypothesis, when faced with the "failed branch validation" screen, users almost universally described the next steps they would take

- Click the link to open CRT for the failing execution
- Scroll past all of the passing top-level tasks
- Upon reaching a failing task, open up the list of child tasks
- Scroll past all of the passing child tasks
- Upon reaching the failing child task, read the failure classification
- Depending on the information visible on the failing child task, decide whether to open the corresponding log file

RECOMMENDATION
Bring users much closer to where they'll almost certainly be interested in going by surfacing information about granular (child execution) failures, and providing the user with a link to jump directly to the log file

Early Signal Platform (ESP)

1. Custom Validation: consider alternate check name

The term "custom validations" felt inappropriately broad to a non-trivial portion of users, which is part of why Hypothesis 026 was marked as "borderline".

RECOMMENDATION

TBD

2. **Provide the option to surface custom checks to top level**

   Particularly among web developers who are involved in MP stewardship (i.e., Flagship Infra), there is an ask to allow certain custom checks to be surfaced to the top-level (directly in the PR tab).

   RECOMMENDATION

   This flexibility should be allowed, even if it does potentially allow the possibility of "too many checks". Users have a wide range of opinions around how many checks is too many (ranging from 6 to 20) so we can have confidence that this aspect of the issue is largely *subjective* within this range.

Policy App

1. **Surface clear info about "comments resolved" policy**

   None of the hypotheses around the policy app were disproved, but some follow-up questions from users shed light on a potential clash between the RB and GitHub mental model, relating to "resolved comments".

   Four users (~30%, all of whom were already familiar with github) expected that an "approving" review should not *also* require comments to be resolved as an independent task. To be clear, this is not a point of confusion -- users were able to understand what was being asked of them -- this is "something that felt strange" when placed in the GitHub world.

   RECOMMENDATION

   In order to help users understand why things are validated in a particular way, they should be provided with clear information that helps them understand how this works *in the context of GitHub.* A link to some wiki page within the "comments resolved" check details should be sufficient.