HW 6b - EECS 481

Josiah Bruner <jsbruner@umich.edu>

Project

Selected Project

The project I have chosen to work on for HW 6 is Mozilla Thunderbird (website: https://www.mozilla.org/en-US/thunderbird/ GitHub mirror: https://github.com/mozilla/releases-comm-central) Thunderbird is a cross-platform, client-side, email client which uses Mozilla's Gecko web rendering engine to control most of the UI and HTML email rendering. It is a large open-source project, serving over 25 million users [1].

Project Context

Mozilla Thunderbird can be thought of as a sub-project of the much larger Mozilla project (which contains other products such as Firefox). Originally it was created as part of Mozilla's internet strategy and was maintained both by volunteers and Mozilla Corporation employees. Eventually though, Mozilla Corporation transitioned Thunderbird to "support only". Presently Mozilla only provides infrastructure for Thunderbird, but all development, maintenance, and QA is done by volunteers. This is described in more detail in the next section ("Project Governance").

Thunderbird has two primary competing markets. The first are other local email clients, such as Microsoft Outlook, Polymail, and other open source clients (e.g. Evolution Mail). Another competing market are web email clients, such as Gmail. Increasingly users are moving to do all their email activities on a website rather than a local application, but Thunderbird exists to provide a reliable solution for those you don't want this. It is especially useful if you have many email accounts, as they can all be "merged" together.

Project Governance

As briefly mentioned above, Thunderbird used to be a full "product" of Mozilla, and so it therefore has a lot of formal processes and governance surrounding it. However, after Mozilla stepped away, it is now considered its own project, with its own government (https://wiki.mozilla.org/Modules/Thunderbird). However, since the infrastructure is still shared, much of the processes are still in place.

As can be seen from the below workflow graphic, the contribution process at Mozilla is quite detailed. The main steps relevant to my contribution are described below:

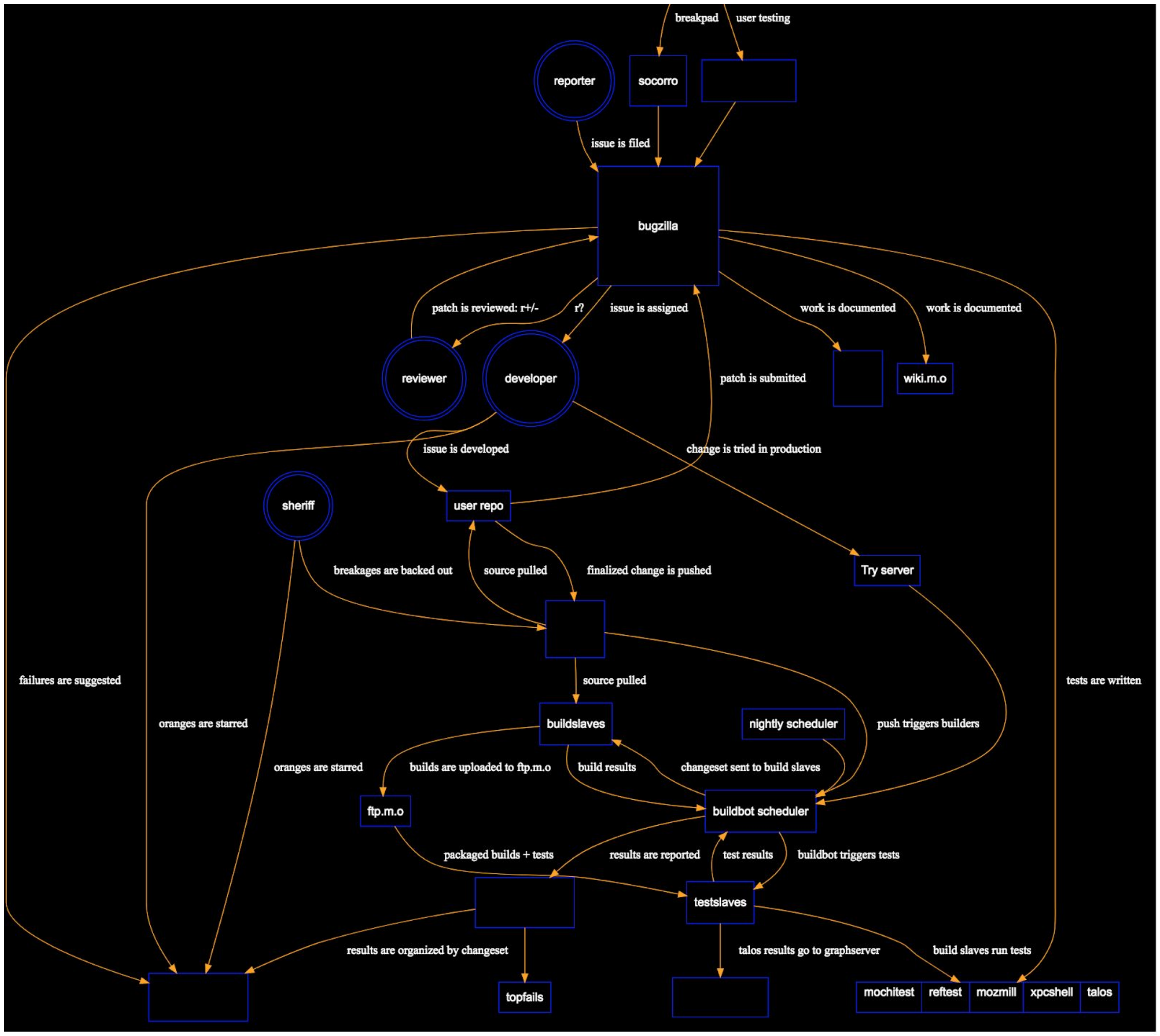


Figure 1 - Mozilla Development Workflow

1. Issue is filed on bugzilla. E.g. https://bugzilla.mozilla.org/show_bug.cgi?id=516464
2. Issue is assigned to a developer (e.g. me)
3. Work is done in the local user repo.
4. A patch is submitted and then sent to bugzilla.
5. A reviewer is added to the patch.
6. The reviewer then provides either r+ or r-. The former indicating it’s okay to land, the later indicating major changes are needed.
7. A “test build” can also be generated by pushing to the “Try server”. This will also run all the regressions tests and provide results.
8. If everything goes well, you can then push your commit to the trunk version of the repo.

A takeaway from this process is that they work very hard to prevent issues downstream. Both static and dynamic testing must be performed before any changes are accepted.

Several tools exist to handle this process and support communication in the community:

- IRC: IRC is used to allow live, informal discussion. This is very useful for discussing issues with the project community or getting help.
- Bugzilla: Much of the actual development work and discussion is done in Bugzilla. Bugzilla tracks many properties of the issues, including priority, status, component, assignee, etc. It also houses most of the discussion relating to that specific bug. Indeed, frequently, discussion from IRC is moved into a comment on that bug to provide traceability.
- MozReview: Although previously code review was done on Bugzilla itself, Mozilla has since developed their own infrastructure using ReviewBoard, called MozReview. Now patches are submitted to this site and the reviewers can provide feedback, etc.
- TaskCluster: Mozilla uses continuous integration, but rather than using something like Jenkins, they maintain their own build infrastructure. Previously this was called BuildBot, but is now called TaskCluster. More information can be found on the release engineering page: https://wiki.mozilla.org/ReleaseEngineering

Tasks

Description

Task A - Get Mail after Remove Password (in pw manager) doesn't prompt for new password (remembers/keeps deleted/old/bad password in cache)

Bugzilla Link: https://bugzilla.mozilla.org/show_bug.cgi?id=516464

This issue relates to a bug in the password manager used by Thunderbird. Thunderbird currently will save your email account password locally so you do not need to type it in every time. However, it appears that if you change the password remotely (e.g. on Gmail), and then delete the (old) local password Thunderbird was storing (but don't close the app), and try to "get mail", the old password will still be attempted because it is still stored in a cache. This should not happen and so my task is to make sure that whenever a password is deleted from the password manager, it is removed from the cache as well.

My fix for this task was submitted as a patch in MozReview and I commented on the bug with the following details:

> "https://reviewboard.mozilla.org/r/236670 provides a patch which "fixes" this issue on IMAP. That is, performing the STR in comment 12 now prompts for new password, rather than attempting the old one.

To do this, nsImapProtocol::GetPassword() now always attempts to get the password from the password manager (or user), and ignores cached values. For this to work nsImapIncomingServer::AsyncGetPassword() now also starts by clearing the existing password from cache so that a re-attempt is made.

So although this works, it is quite subpar. In an ideal world we should get notified that an account password has been removed from the password manager, so that we can call nsImapProtocol::ForgetPassword(). However, the TB-specific password manager code seems to only open the toolkit password manager, so I'm not sure how we would get such a notification. Maybe the reviewer will some idea?“

Artifacts

Task A - Get Mail after Remove Password (in pw manager) doesn't prompt for new password (remembers/keeps deleted/old/bad password in cache)

1. Bugzilla Link: https://bugzilla.mozilla.org/show_bug.cgi?id=516464 - Link to issue in issue tracker with comments by preceding QA, developers, and me.
2. Patch on MozReview: https://reviewboard.mozilla.org/r/236670/diff/1 - Link to my bug fix. Mozilla doesn’t use pull-requests, but instead requires review of each patch, so this is very similar.

QA

Strategy

Several quality assurance mechanisms were applied:

1. **Code Review**: In order to make sure my approach is adequate and to improve quality, Mozilla requires code review.
2. **Local testing**: To verify that the issue was properly addressed, I locally reproduced the issue before my changes. After my change, I repeated the same steps to assure the issue was resolved.
3. **Local regression testing**: To verify my change didn’t break other functionality, I always performed a local regression test (xpc-shell: https://developer.mozilla.org/en-US/docs/Mozilla/Thunderbird/MailNews_xpcishell-tests)

Evidence

Code Review

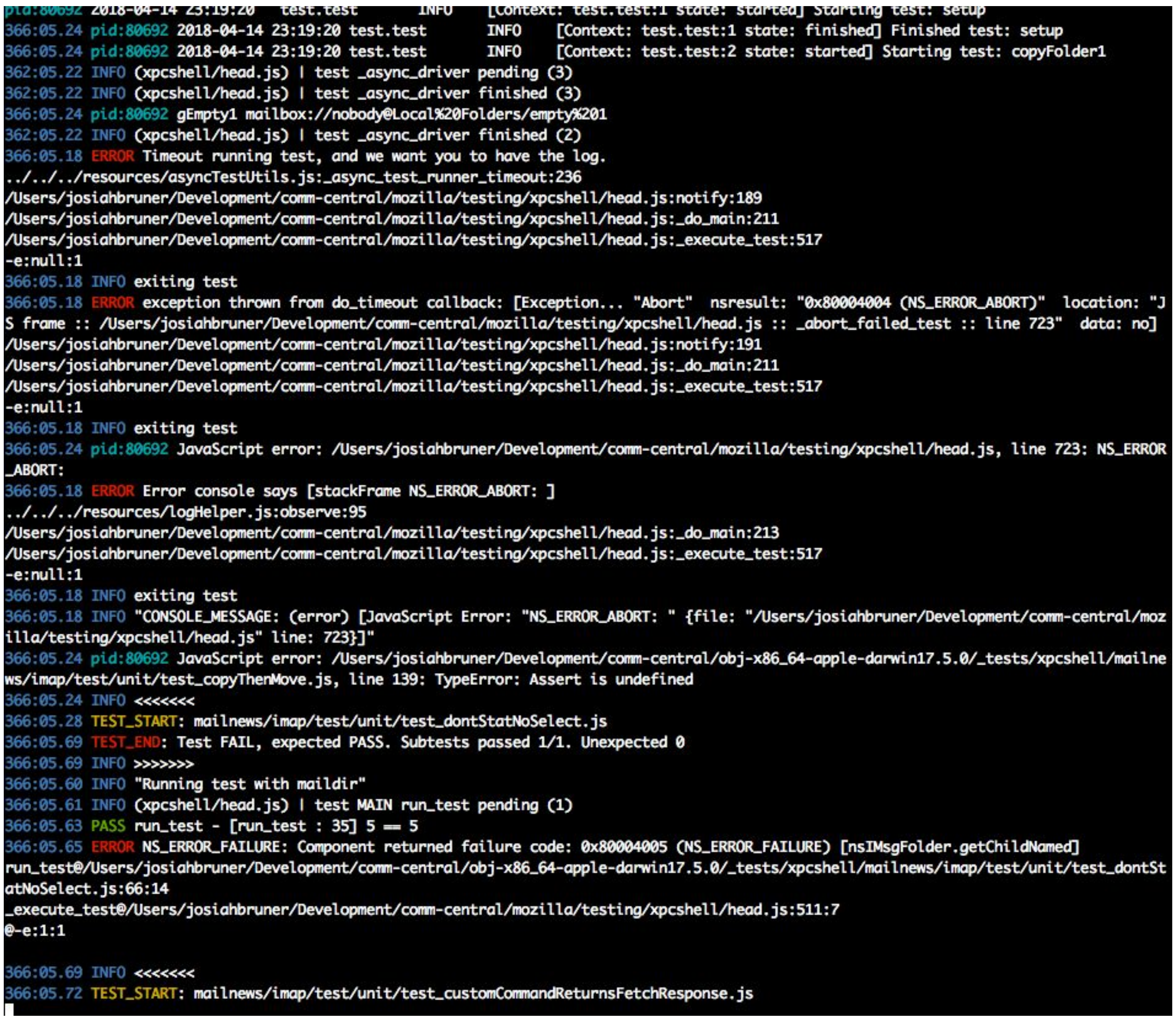
The code review was upload to MozReview and two reviewers were assigned. This review request can be located here:
https://reviewboard.mozilla.org/r/236670/diff/1#index_header
Note, however, that apparently it takes a long time for anyone to leave comments. Indeed, one of the reviewers (JorgK), wrote on IRC: "I'll take a look next week.", which means that feedback probably won't appear until after this project is due. He was able to provide some brief comments, but not on the implementation itself.

Local Testing

This is self-reported anyway, but I confirmed locally that my change fixes the issue. This was presented in the Bugzilla ticket here:
https://bugzilla.mozilla.org/show_bug.cgi?id=516464#c15 and
https://bugzilla.mozilla.org/show_bug.cgi?id=516464#c16

Local Regression Testing

I ran the xpc-shell local regression tests after my change has been applied. Plenty of the initial tests passed, but eventually I hit a significant number of failures:



```
pid:80692 2018-04-14 23:19:20 test.test      INFO    [Context: test.test:1 state: started] Starting test: setup
366:05.24 pid:80692 2018-04-14 23:19:20 test.test      INFO    [Context: test.test:1 state: finished] Finished test: setup
366:05.24 pid:80692 2018-04-14 23:19:20 test.test      INFO    [Context: test.test:2 state: started] Starting test: copyFolder1
362:05.22 INFO (xpcshell/head.js) | test _async_driver pending (3)
362:05.22 INFO (xpcshell/head.js) | test _async_driver finished (3)
366:05.24 pid:80692 gEmpty1 mailbox://nobody@Local%20Folders/empty%201
362:05.22 INFO (xpcshell/head.js) | test _async_driver finished (2)
366:05.18 ERROR Timeout running test, and we want you to have the log.
../../../resources/asyncTestUtils.js:_async_test_runner_timeout:236
/Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js:notify:189
/Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js:_do_main:211
/Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js:_execute_test:517
-e:null:1
366:05.18 INFO exiting test
366:05.18 ERROR exception thrown from do_timeout callback: [Exception... "Abort"  nsresult: "0x80004004 (NS_ERROR_ABORT)"  location: "J
S frame :: /Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js :: _abort_failed_test :: line 723"  data: no]
/Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js:notify:191
/Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js:_do_main:211
/Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js:_execute_test:517
-e:null:1
366:05.18 INFO exiting test
366:05.24 pid:80692 JavaScript error: /Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js, line 723: NS_ERROR
_ABORT:
366:05.18 ERROR Error console says [stackFrame NS_ERROR_ABORT: ]
../../../resources/logHelper.js:observe:95
/Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js:_do_main:213
/Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js:_execute_test:517
-e:null:1
366:05.18 INFO exiting test
366:05.18 INFO "CONSOLE_MESSAGE: (error) [JavaScript Error: "NS_ERROR_ABORT: " {file: "/Users/josiahbruner/Development/comm-central/moz
illa/testing/xpcshell/head.js" line: 723}]"
366:05.24 pid:80692 JavaScript error: /Users/josiahbruner/Development/comm-central/obj-x86_64-apple-darwin17.5.0/_tests/xpcshell/mailne
ws/imap/test/unit/test_copyThenMove.js, line 139: TypeError: Assert is undefined
366:05.24 INFO <<<<<<<
366:05.28 TEST_START: mailnews/imap/test/unit/test_dontStatNoSelect.js
366:05.69 TEST_END: Test FAIL, expected PASS. Subtests passed 1/1. Unexpected 0
366:05.69 INFO >>>>>>>
366:05.60 INFO "Running test with maildir"
366:05.61 INFO (xpcshell/head.js) | test MAIN run_test pending (1)
366:05.63 PASS run_test - [run_test : 35] 5 == 5
366:05.65 ERROR NS_ERROR_FAILURE: Component returned failure code: 0x80004005 (NS_ERROR_FAILURE) [nsIMsgFolder.getChildNamed]
run_test@/Users/josiahbruner/Development/comm-central/obj-x86_64-apple-darwin17.5.0/_tests/xpcshell/mailnews/imap/test/unit/test_dontSt
atNoSelect.js:66:14
_execute_test@/Users/josiahbruner/Development/comm-central/mozilla/testing/xpcshell/head.js:511:7
@-e:1:1

366:05.69 INFO <<<<<<<
366:05.72 TEST_START: mailnews/imap/test/unit/test_customCommandReturnsFetchResponse.js
```

These failures could be caused by a lot of things. For instance:

1) They could be caused by my changes. To verify this I would need to retry these tests without my changes applied.
2) They could be pre-existing failures. Again, to verify I would need to restart without my changes applied.
3) They could be because I configured some of the testing infrastructure wrong.

What this tells me is that I will need to ask the reviewer of my patch to submit a “try build” on my behalf. This will trigger a fresh build on Mozilla’s build servers and run all the tests. This will allow me to compare any new defects caused by my patch. Since I have to rely on others, this probably won’t be done by the project deadline.

Plan Updates

In general, my plan and time schedule went as expected. As described in HW6a, I indicated that task A would likely take the entire time, but in the case it didn't, I proposed a second task (task b). In actuality, I was indeed not able to work on task B at all.

Work estimates on the subcomponents of the task however did differ quite a bit. The following table provides my initial work allocation plan, as well as the actual time spent:

ID	Description	Expected Duration	Actual Duration	Notes
0	Changes to build system happened so I had to do an entire recompilation.	Wasn't Planned	2 hours	
1	Locate password manager code (specifically the code that handles password deletion)	2 hours	1 hour	Found password manager code fairly quickly, but unfortunately determined it was actually not super relevant to fixing this bug.
2	Locate the code that stores passwords in the cache. Determine if an object store exists to touch this that can be used in password manager code.	2 hours	2 hours	
3	Write patch to delete from cache. Test. Repeat until the solution works locally.	4 hours	5 hours	In actuality the locations I thought were relevant to storing the password were only indirectly relevant, and plenty of time was spent trying something, figuring out there was another layer of indirection in the codebase, and having to find the actual relevant entry point.
4	Upload patch to ticket and trigger a "try build". Analyze test results.	1 hour	4 hours	Patch was uploaded, but I found out I can't trigger try builds myself. Instead ran

				local tests, which took a very long time.
5	Revise patch to fix test failures found by try build.	2 hours	N/A	Not relevant because a try build wasn't performed.
6	Submit for review. If comments, address them, else, land patch on trunk.	2 hours	1 hour	Some configuration changes were necessary to allow me to submit my patch for review, but I was able to in an hour. However, the reviewer said they wouldn't be able to review for another week, so in actuality this process was never completed.
		13 hours	**15 hours**	

Experience

Disclaimer: Since I had previously contributed to the Mozilla project substantially, this instance was in many ways "just like every other". However, I now have more respect for the lengths they take to "prevent issues early". Still, in an effort to keep this section of the paper interesting, I will instead reflect on my experiences contributing to Mozilla projects in general, not just focusing on this single instance.

Contribution to a Large Project

Mozilla Thunderbird is a very large project with hundreds of thousands of lines of code. Unsurprisingly, it turns out that contributing to a project of such size is not at all like contributing to a small project (whether it's a school project or small Github repo). Unlike smaller projects, a very significant component of the work I've done working on Thunderbird (and in this project) is trying to understand where things are and what does what. Large projects are a mess of integrations and indirection, but tend to lack documentation describing such a design. Therefore, you become forced to experiment. You imagine where relevant code would be, search for it in the source code, look at the code, determine where you need to go and manually follow source inclusion (because the project tends to be designed in a way where an IDE can't just link things for you).

There are also a ton of tools and configuration changes needed to get things functioning properly. Fortunately in the case of Mozilla, a lot of this has since been automated.

Unanticipated Challenges

From my experience with Thunderbird, there always tend to be unanticipated challenges in every bug. In the case of this project, the main challenge was being forced to work around a limitation in a shared component not owned by Thunderbird. I mentioned in the bug report that ideally when a password is deleted from the password manager, somehow Thunderbird will get notified and then delete the password from cache. Unfortunately, it didn't seem like the password manager provided such a callback. To make matters worse, the password manager component is actually shared with Mozilla Firefox, and so adding functionality would require modifying Firefox's source as well (in addition to requiring extra testing, dialog with developers, etc.)

Collaboration and Culture

Mozilla has a wide range of tools and methods for collaboration, as well as a distinct culture (everything should be done in the open if possible). I found that this culture helps with contribution, because there is a plethora of resources available online in Wikis, websites, and on IRC logs. If someone runs into an issue, they tend to document this somewhere so others can benefit.

IRC is also extremely useful. Although for this task I could mostly ignore IRC, that was only because I had lots of experience working on the project in the past. When I first starting contributing to Mozilla projects, I spent a significant amount of time asking questions on IRC. Since people contribute from all over the world, I was able to get support practically anytime of the day.

In the case of this project, I had very little direct interaction with the community or team leadership. The only discussion I had was with one of the main project leads, where I asked if he was the right person to review my change. However, in my past life I was actively involved with the project leadership (and in fact was part of that group). I was part of bi-weekly calls and we would discuss project status and what we've been working (think stand-up in Agile methodology). They would also help guide my activities and prioritize certain issues.

Note however that in many cases the community tended to be quite toxic. Indeed, this was the reason I left Mozilla several years ago. Although the core contributors and Mozilla employees were generally friendly, many users or part-time contributors were extremely aggressive. Back in the day I mostly worked on user interface-type activities, and reason a fair share of hate mail letting me know how much they hated it.

I did get one particularly useful comment form JorgK (a lead developer). The whole comment can be seen here: "https://bugzilla.mozilla.org/show_bug.cgi?id=516464#c17" There were two useful comments in particular:

> *“...this is a fix for IMAP, so how about POP, SMTP, NNTP? According to comment #12 it's also an issue for POP. Clearing the review for now since the patch is incomplete.”*

This is totally reasonable, as I forgot to clarify the reason for not implementing the other protocols (I wanted to see if the approach was valid and I don’t have a simple way to test those). This information though should’ve been provided initially, so I took it as a learning opportunity.

The second, more useful comment was:

> *“You can always find the peer of the module from https://wiki.mozilla.org/Modules/Core (well, I can't see anything with passwords listed there) or a frequent author in that directory and ask them.*
>
> *In fact, I'll do that now:*
> *Gijs, I know very little about how passwords are stored :-(Apparently TB caches passwords once retrieved from the password manager, so when a password is removed, we don't notice it. Is there any notification available for password removals/changes? Or could you give us any other hint or tip?“*

I was not aware of the page which provided a list of peers for a module. In addition, he kindly redirected my implicit question to the right person, so hopefully some more insight is gained.

What Worked and What Didn’t

If I were to start my own project from scratch, I would use much of Mozilla’s processes. I would definitely use an IRC-type communication method to make contributing easier. I would require code review and automated regression testing (although the tools would be different), in order to ensure quality. Although bugzilla is nice, in reality I would probably use Github (and therefore Github issues) for my own project since it wouldn’t have to be maintained and most modern projects seem to favor it.

I found that Thunderbird had several different automated test suites. Perhaps this is necessary, but it made it difficult to verify my code changes. Some test suites did not seem to work, while others did. I would try to use a single testing infrastructure in order to reduce complexity in this regard.

Mozilla does not seem to use automated static code analysis (locally or in their build servers). For my own project I would require such tools in order to mitigate the risk of trivial security issues or obviously wrong software implementations.

Relationship to School and Work

There are both similarities and differences between contributing to open source (Thunderbird) and school projects and work projects. I briefly describe each case below.

Working on large, open source projects is almost completely unlike working on school projects. The codebase is huge, and a lot of tools are necessary to get started. School projects tend to be small and easily manageable. If school projects have some kind of testing infrastructure, they are usually simple and fairly reliable. Open source projects, on the other hand, can have fairly complex and messy testing systems. On school projects the primary work is implementing some specific functionality, while on open source the primary job seems to be debugging or testing (which makes sense given what we've learned in class).

These properties, however, are fairly similar to working at large, private companies. In my experience, code bases at private companies are also large and require a lot of tooling and have a messy testing infrastructure. However, open source seems to do a better job unifying processes and providing communications methods (and promoting a culture of communication), rather than at companies I've worked. Where I've worked, a lot of material is shared only within your own team. In addition, in open source it is common to help (or even work directly) in other domains. At most companies, however, this is generally avoided as your own team has their own cost center, which incentivizes siloed work.

Advice for Future Students

Advise: Be very careful with time management; course starts "easy", and it's very easy to become too comfortable and eventually realize you put something off too long.

If you are willing to let future students see your materials..: Yes, I am willing to let future students see my materials.

References

[1] https://mail.mozilla.org/pipermail/tb-planning/2017-March/005298.html