Project Post Mortem Report

As a Boilermaker pursuing academic excellence, we pledge to be honest and true in all that we do. Accountable together – We are Purdue.

Signed: Parker Bushey, Kaitlyn Roberts, and Ashwin Senthilkumar

Date: December 15, 2021

Assignment Goal

In this assignment, you will provide a *postmortem* on your Project 2 implementation.

A postmortem is a reflective document that captures the successes, failures, and lessons learned from an engineering effort. Engineering is never a one-time effort; teams deliver a system and then continue to work together on the next system. Remember, software engineering is "programming integrated over time".

One key to an effective postmortem is making it *blameless*. Reflect on your team's experiences and report the data.

Relevant Course Outcomes

A student who successfully completes this assignment will have demonstrated the ability to

- *Outcome ii*: The ability to conduct key elements of the software engineering process.
- *Outcome iii*: Develop an understanding of the social aspects of software engineering… including…communication [and] teamwork.

Resources

Perspectives on conducting a postmortem

- Software Engineering at Google textbook
  - Chapter 2: How to work well on teams
  - p. 87 (box "Failure Is an Option")
- Postmortems at Google
- Postmortems at Amazon

Examples of real postmortems

- Microsoft
  - This website has some of Microsoft's postmortems, including e.g. this one.
- Cloudflare
  - An outage impacting thousands of their customers
- Google
  - A specific example following their philosophy above

Assignment

You will critique several aspects of your work on Project 2.

Rate your performance

Look at the grading breakdown in the project spec.

Approximately what grade do you expect to receive for Project 2 overall?
75%

Approximately what grade do you expect to receive on your implementation?
65%

The next sections will help you conduct a root cause analysis, thinking through how you got to this point (whether good or bad!).

Critiquing your plan

*At each point, you should indicate things you would **do again** or **change for next time**.*

In what ways was your plan (design, timeline, etc.) helpful? (At least 2 examples with explanation)

Example 1:
Planning for time to research allowed us to better grasp an understanding of new concepts like Google Cloud, Github Actions, and APIs before we got into the thick of coding.

Example 2:
Thinking about the way data would flow through our system was helpful as it let us look at the big picture and what Google Cloud parts we would need.

Example 3:
Marking the baseline requirements allowed us to allocate time to the more essential parts of the system if we were running behind schedule.

In what ways was your plan unhelpful or inadequate? (At least 2 examples with explanation)

Example 1:
Until the API spec came out, we were still a little lost about how data would be given to us and therefore how we could parse it. We could theorize and set up the Cloud parts based on that theory, but the theorized implementation ended up being modified a good amount by the time the final design finished.

Example 2:
Since we did not fully understand API at the time, the plan was not helpful in that aspect. Our plan described the functionalities the deployed app would have, but not how to deploy the app. We did not expect to find so many difficulties in this area or when testing/debugging the deployed app.

Were you over-ambitious in any aspect of your design? Did you reduce the scope of the design as a result?

We were overambitious in a lot of the design. It ended up that the API spec defined the requirements we kept rather than our original design plan. We had already planned not to do debloating, user logins / accounts, or authorization, but we were disappointed that the website had to be scrapped, and with it the download capability.

In Project 2, you defined the initial project scope yourselves. If you reduced scope partway through, discuss what factor(s) led you to do so.

There were a lot of other projects and things going on for Katie the last few weeks, and as she was the one who understood the cloud and had access to it, a lot of debugging fell to her. Like what was said above, we were not expecting the API deployment to be so difficult (see Piazza post @170).

Critiquing your process

At each point, you should indicate things you would **keep** or ***change for next time***.

- Did your team follow your team contract? If not, what aspect broke down?

Generally we kept to the guidelines laid out in our team contract. We might consider the period around Thanksgiving to be a period of communication loss. There was a period we didn't have a meeting for some time when we should have. We didn't strictly set up a weekly meeting time during the project plan, and thus it was very easy to skip a meeting if no one brought it up over GroupMe (our main method of communication outside meetings). Surprisingly in Project 1, we followed a better meeting schedule, but this time around we didn't address it. If we were to do this again, a set weekly meeting time would definitely have benefited productivity or at least, increased awareness of progress made.

- Did you meet your milestone deadlines?

The goals that we set for week 3 might have been too ambitious. The task that did happen to get finished took substantially less time than anticipated. The other two tasks, ingestion and version parsing, took another couple of weeks to be implemented.

Week 4 saw some deviation from the milestone schedule, but good progress was made overall. The GitHub Actions task wasn't completed, but it wasn't too far off. More progress was made on ingestion, but it wasn't complete. Additionally, the paginated list task was completed ahead of schedule, but at the cost of some other tasks intended to be done.

Continuous integration was finished up in week 5. Progress was made on the other tasks from the previous weeks, but not full completion.

We believe week 6 lines up with the lack of communication period described in the previous section. The security task got touched briefly but was foolishly pushed to a later date. While time was also put into ingestion/updating and the API, nothing was completed as planned.

Week 7 was graced by a large push forward from Katie, but still many tasks were not completed. Security was taking longer than anticipated and ingestion/updating still needed completion proper.

The week leading up to the final deliverable was a scramble, and some optional requirements were cut for time. We did not deliver on time due to a combination of circumstances, but we believe the product was working acceptably when submitted.

- Did you track your progress well?

We kept track of each milestone in the milestone documents, and recorded any deviation from the planned progress in the planning document. Any roadblocks or reasoning behind not completing an assigned part was mentioned in the justification portion below the hours worked table. The "Progress Relative to Plan" column of the weekly progress section in the milestone documents also indicate how many hours were spent and what tasks were worked on/completed in that milestone.

- In what ways did you deviate from your plan? (At least 2 examples with explanation)

Example 1:
For the security requirement, we originally planned on performing security probing with RESTler, but this was ultimately dropped due to lack of time. The other parts of the security

requirement took much longer than anticipated so, it was decided the optional requirement of using RESTler would be left out. With better planning, security probing would've likely made it into the final deliverable.

Example 2:
The system that performs zipping and unzipping of files was originally intended to have the ability to take in multiple packages at once. However, this was overlooked when implementing and by the time we realized it was part of the plan, too much of the project had been written to work with the new implementation. It wasn't worth the time to make the changes necessary to rework such a minor part of the system.

- To what extent did your design inform your implementation?

Our design greatly influenced which Google Cloud parts we used since we started building our code off of a tutorial we found that informed our design. The tutorial made use of Firestore and App Engine, so there was little inclination to use anything else.

- Meetings: What aspects of your meetings were effective or ineffective? Did you apply anything suggested by the guest speaker Greg Wilson?

Our meetings were generally effective because we stayed on task during the meeting and made any roadblocks or issues clear so we could address them as a team.

- Describe a conflict your team experienced and how you addressed it.

As indicated in our project milestones, there were a few weeks where, the team as a whole, didn't make that much progress because of Thanksgiving break and exams. Moving forward, we were able to pick up traction closer to the end of the project and make up for it. We also changed requirements as needed to suit this.

- Discuss your experience working with the other team's Project 1 implementation. Cover topics like: Did you effectively conduct a component assessment test? Did you discover problems at the last minute (and if so, why didn't you catch them earlier)?

Before we could test the Project 1 implementation, we had to solve a different problem. This problem was getting the project to run on our machines at all. The first couple of weeks saw

little use of Project 1 because of setbacks caused by dependency issues and inability to successfully use the given run executable.

After all was sorted with running the project, we were able to confirm the implementation worked as expected and reviewed the validity of the score output. Only minor changes were made and none affected the original workings of the software.

Other than the problems we ran into initially, the implementation worked as expected for the rest of the project.

- Project 2 involved the use of deployment technologies (GitHub Actions, GCP) that were new to most students. Did you share knowledge effectively across your team? What mechanisms did you use to support knowledge sharing and troubleshooting? How could you improve in knowledge sharing?

Knowledge was shared between team members during meetings, with Katie explaining how the deployment worked and how data would be stored in Firestore. This was mostly effective, as other members were taught well enough that they trusted Katie with the API and Cloud parts, and could implement their portions in a way that would integrate with Katie's implementation. The knowledge sharing still could have been improved by enabling the other members to try out the deployment themselves, and being taught how to do that with all the needed settings.

Critiquing your product

Weaknesses

Identify and discuss three two weaknesses of your product (this includes all aspects of the product, including testing, CI/CD, API design, system architecture, implementation decisions, and documentation).

Weakness 1
What is the weakness?
The CI/CD implementation involves some manual checking of linter output to verify the stability/security of the pushed software.

Explain the presence of the weakness in your product.
This process is not 100% reliable as humans are prone to misread or skim over potential serious bugs. It also does not prevent a lazy developer from deploying the project without properly considering the continuous integration step.

How would you mitigate or eliminate the weakness?
There was definitely more research needed into GitHub Actions to learn how to prevent a deployment of the code if the checks come back negative. It would involve the CD workflow to be dependent on the CI workflows passing, but this was not something that we implemented.

Weakness 2
What is the weakness?
There is no automatic testing implemented of the API.

Explain the presence of the weakness in your product.
We had been manually testing because we were finding lots of bugs, and it became too late to implement any automatic testing by the time that was finished.

How would you mitigate or eliminate the weakness?
We would implement automatic testing in our run file and additionally as a Github Action.

Weakness 3
What is the weakness?
The Project 1 code has to be supplied a GitHub token, which we hardcoded without any security.

Explain the presence of the weakness in your product.
Because of the lack of time when this weakness was found, we did not bother with encoding it to save in the repo and decoding it automatically when GCP would deploy like we did with the GCP service account key.

How would you mitigate or eliminate the weakness?
If we had more time we would have done what we did with the GCP key, which was implemented earlier in the project. For now, we just changed the token to use in the deployment so that the token publicly available in the repo being compromised would not affect our API.

Security

Describe your experience conducting the STRIDE analysis. Were there difficulties? Did you discover any security issues as a result of your analysis?

Conducting the STRIDE analysis really showed me how important the job of security really is. The main difficulty, and a very important one, was creating and analyzing the project data flow diagrams too late in the milestones. The answer to whether I discovered security issues from the analysis is yes. There were things that if discovered earlier in the engineering process, could have been addressed more thoroughly. The big takeaway from this is to begin the STRIDE analysis as soon as possible (as mentioned by Paschal after one of the milestones). Overall, the experience was a positive one, but unfortunately there was space for improvement.

Under what threat models is your system secure? (What assumptions do you have about, e.g., ACME Corporation's processes, GCP, etc. in order to make this guarantee?)

Because of the decision not to implement user authentication, we do not believe the system would hold up to the majority of threat models. Had we implemented this optional requirement, STRIDE would've been closer to being fully addressed, however there were a few key assumptions that needed to be made.

First, there was no backup made of the registry upon resetting. To achieve availability of the system, it would need to be assumed that workers with access to the reset ability have no malicious intentions.

Second, it was assumed that no code would ever run within the registry upon package upload. Hackers are clever, and if they are able to get a file into the registry, they might figure out how to run code in it as well. This was never accounted for during implementation of the system.

Finally, it was assumed that HTTPS and b64 encoding provides an unbreakable barrier to data being transferred to the registry. While a brute force attack would likely fail, there may be other ways a hacker could obtain the keys required to decode. No accommodations were made for this possibility either.

Maintainability

Discuss how well you documented your system and its design. For example:

- Do you think a new member could join your team easily?
- Could another team build effectively on your work?
- Are your system models in sync with your implementation? To what extent would a new member have to study the code and induce a model for themselves?

While the API spec is up to date with our implementation, no documentation was given on how to access the Cloud project or build a new one from scratch. This would make it hard for a new team member to understand the project without doing the same research Katie did. The new member would have to study the code as well to figure out what is relevant and what is part of the original tutorial code that we aren't using. If the next team taking on the project had knowledge on API and Google Cloud, they could add additional functionality to the app or add to what data is stored in the Firestore registry. They would have to do a good amount of work if they wanted to implement the website though.

Individual contributions

To help Prof. Davis assess whether the project was too hard or too easy, please review your milestone documents and populate this table.

Team member	Week #	Hours spent
Kaitlyn Roberts	1	0
	2 (Project Plan)	7
	3	5
	4	7
	5 (first deliverable)	12
	6	10
	7	30
	8 / 9 (final deliverable)	35
	Total:	106
Parker Bushey	1	0
	2 (Project Plan)	6
	3	10
	4	6
	5 (first deliverable)	6
	6	5
	7	6
	8 / 9 (final deliverable)	25
	Total:	64
Ashwin Senthilkumar	1	0
	2 (Project Plan)	6
	3	3
	4	2
	5 (first deliverable)	3
	6	5
	7	5
	8 / 9 (final deliverable)	30
	Total:	54

Summary

Review the postmortem you've conducted. Capture **two lessons learned** that you want to share with the class.

Lesson 1 (about some aspect of software design/implementation/validation):
API is harder than it looks, because there are many ways to implement them and not all implementations are compatible.

Lesson 2 (about teamwork):
Encourage more collaboration between teammates working on subsystems, because having a small part in each subsystem (even if you did not take a lead role in it) would help everyone understand each other, what they are doing, and how the parts need to fit together in the end.

Learning from Project 1

Most of you were on the same team for Project 1 and Project 2. In your Project 1 postmortem, you critiqued your plan, process, and product; you indicated several lessons that you learned; and you made recommendations for next time.
Review your Project 1 postmortem. In what ways did you learn from your experience in Project 1? In what ways did you repeat your mistakes? Discuss.

In some areas, we learned.
We learned to schedule a longer time to research and learn what we didn't know at the beginning of the project, which helped a lot. We felt less chaos while coding since we understood the tools we were using better. We also took to heart the lesson we wrote about using the recommended tools for the job, since we switched in the beginning to Python whereas in project 1 we used Java, and we utilized a GCP tutorial to inform us what tools were recommended to use. Another one of the lessons we learned from the first project was to label commits and explain the code to teammates so they understand how the code works and how they can understand how to implement it. In project 2, we made sure to explain the progress in code at each meeting and during the making of each milestone document. We also shared screens during these meetings to see what issues we might be facing.

In other ways, we made the same mistakes.
We still had the same problem of not having the full detailed picture during our design phase, and therefore being a little lost when starting to code. We were even more overambitious than we were for project 1, as we did not know how difficult GCP would be to utilize and that the API spec would need conversion.

Grading

This assignment is worth 15% of the overall Project grade. Treat it with the care it deserves.