MOMENTUM

I. Motivation

Programmers have a reputation for being terrible at estimating how long it will take to complete a project [1] [2] [3]. This is because developing software is inherently uncertain: there is no way to forecast every detail of a design from the outset, understanding the full requirements of the project as well as the hurdles that an individual will encounter along the way. (If this were the case, developing software would be a trivial task). This is only exacerbated by the fact that a huge number of programming projects are not completed by a single person, but by teams — if an individual can't predict how long something will take for themselves, what hope does a team have of estimating the completion time for an entire project? Ultimately, this is a problem that can have huge monetary, emotional, and interpersonal costs [4]. The reality is that programmers are usually much too optimistic [1] [2] [3] — they try to fit too many features into a release, and some have to be cut; they think that they can focus effort in one area of a project because it will be quick, but it ends up taking two, three, four times as long as they had thought initially. Having better time estimates lets us choose how we focus our efforts [4] [5].

We've all heard the horror stories about impossible deadlines set by project managers and the botched releases that follow from developers: software development effort estimation is extremely important, but is sometimes not taken very seriously by software teams since it can be difficult to manually estimate how long it will take each person in the team to complete pieces of a project and how this will affect the team as a whole. In-depth project estimation takes significant time and resources as well as consistent participation from the whole team — in other words, it requires a common framework for teams to adhere to. We believe that we can build an accessible product that would address this unrelieved problem of effort estimation for a whole project with quick estimations by members of a team.

Our project would allow these developers to better gauge their time and progress on work while also giving project managers the ability to manage and set feasible project deadlines for diverse teams of developers. This change would, in turn, allow companies to expend less on problems that arise due to improper effort estimation.

II. Approach

A. Goals

Our project involves creating a tool that uses information gathered about a user (and their team members) to predict their team's product ship date. The planned approach is a two-phase process, focusing on the two components of an effort estimation tool: the user interface and the effort estimation model (Fig. 1).

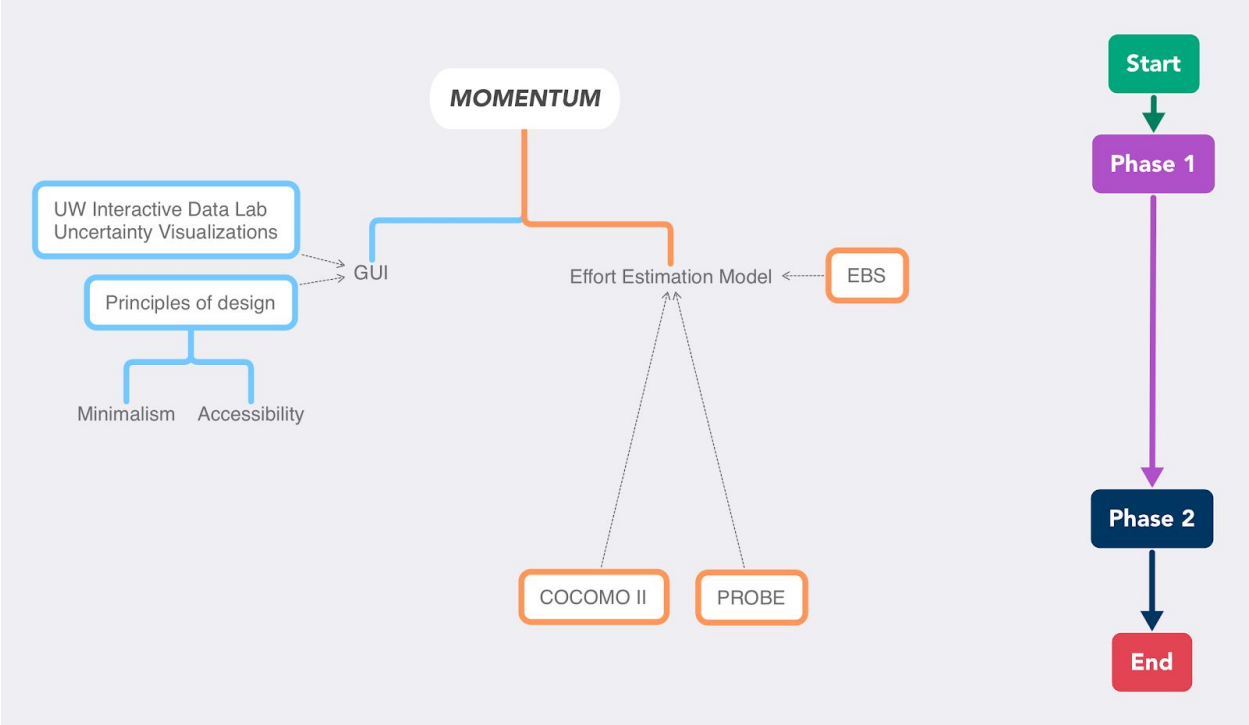

Fig. 1 A representation of of the two phases of our design process, as well as the two components we will be addressing.

As seen in Fig. 1 Phase 1, we plan to begin by focusing on implementing an accessible, simple, and intuitive GUI, using Evidence-Based Scheduling (*cf. section D*) as our initial effort estimation model. Momentum will be the only tool that is simple to use, does not require high maintenance or training, and is cheap and accessible for the public. Phase 1 is our main focus.

As seen in Fig. 1 Phase 2, we plan to refine our effort estimation model (*cf. section D*) in order to improve accuracy or ease-of-use. Phase 2 may not be completed within the time provided in this course, but offers meaningful extensibility for our project if left with enough time.

B. Existing Solutions

As mentioned in section A, there are two components in a software development effort estimation tool: a user interface and an effort estimation model. In section C, we will address existing GUI tools for software development effort estimation. In section D, we will discuss existing processes for effort estimation.

C. Existing GUI Tools

Existing tools related to our project include Manuscript, ProcessPAIR, and Process Dash. These tools are outlined below.

1) Manuscript: A project management tool for software teams [8].

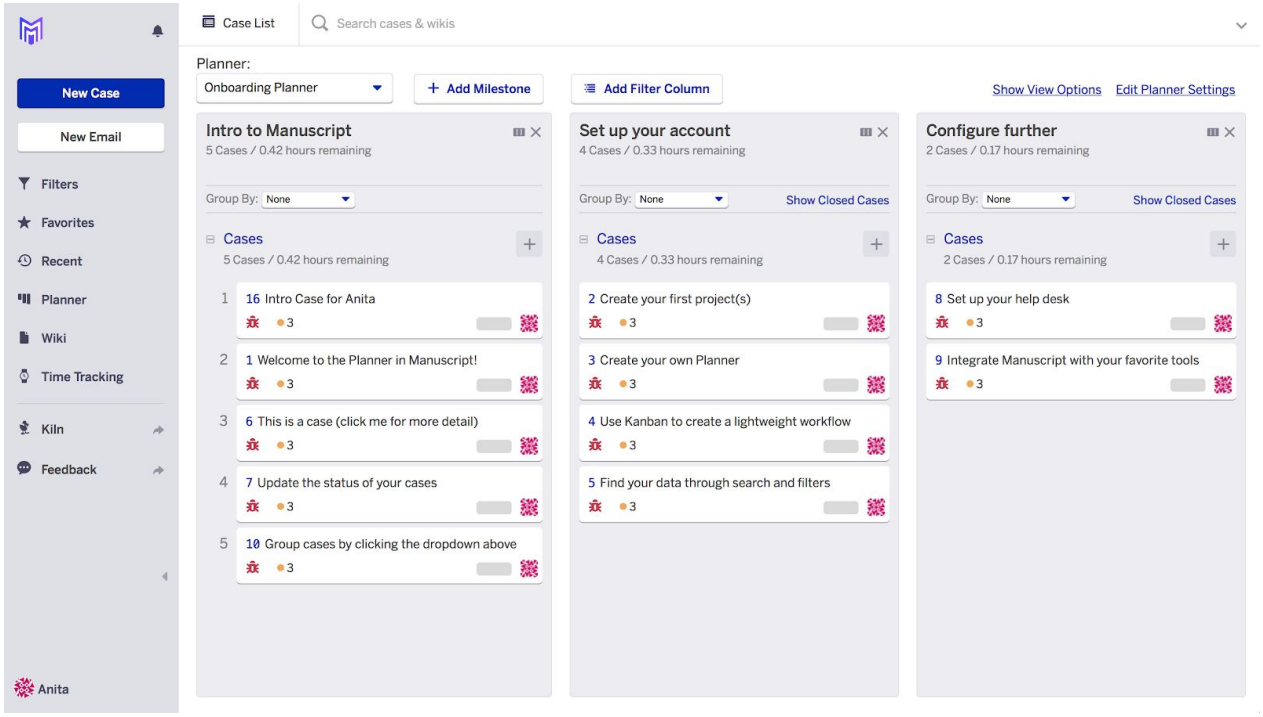

Fig. 2 Planner with tasks.

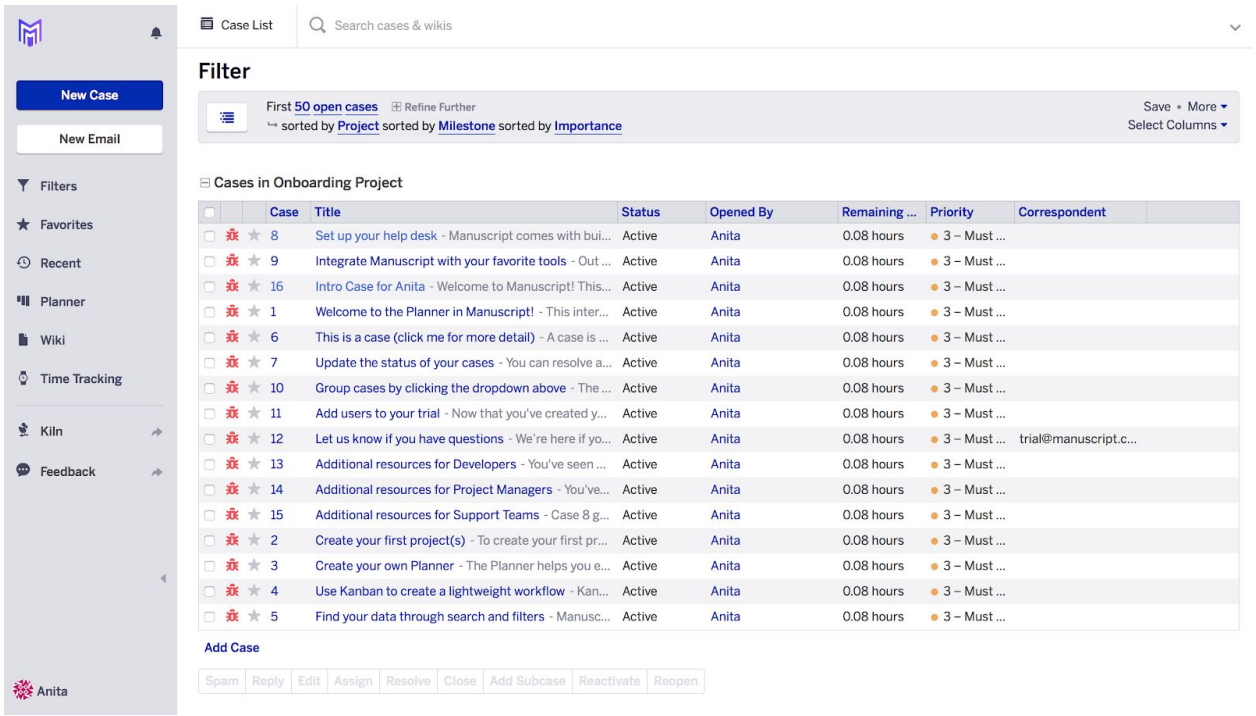

Fig. 3 A view with all your assigned tasks/cases.

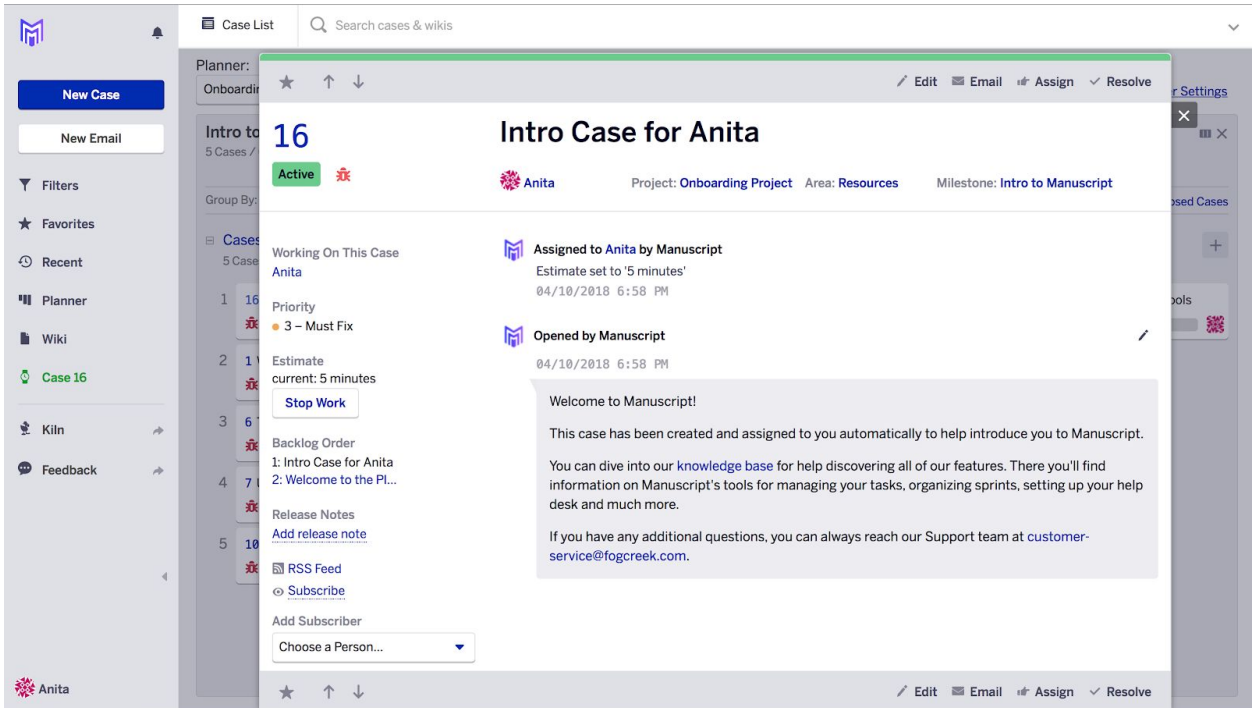

Fig. 4 View for task details. There is focus on ushering the user into reading the user guide to learn how to use the tool.

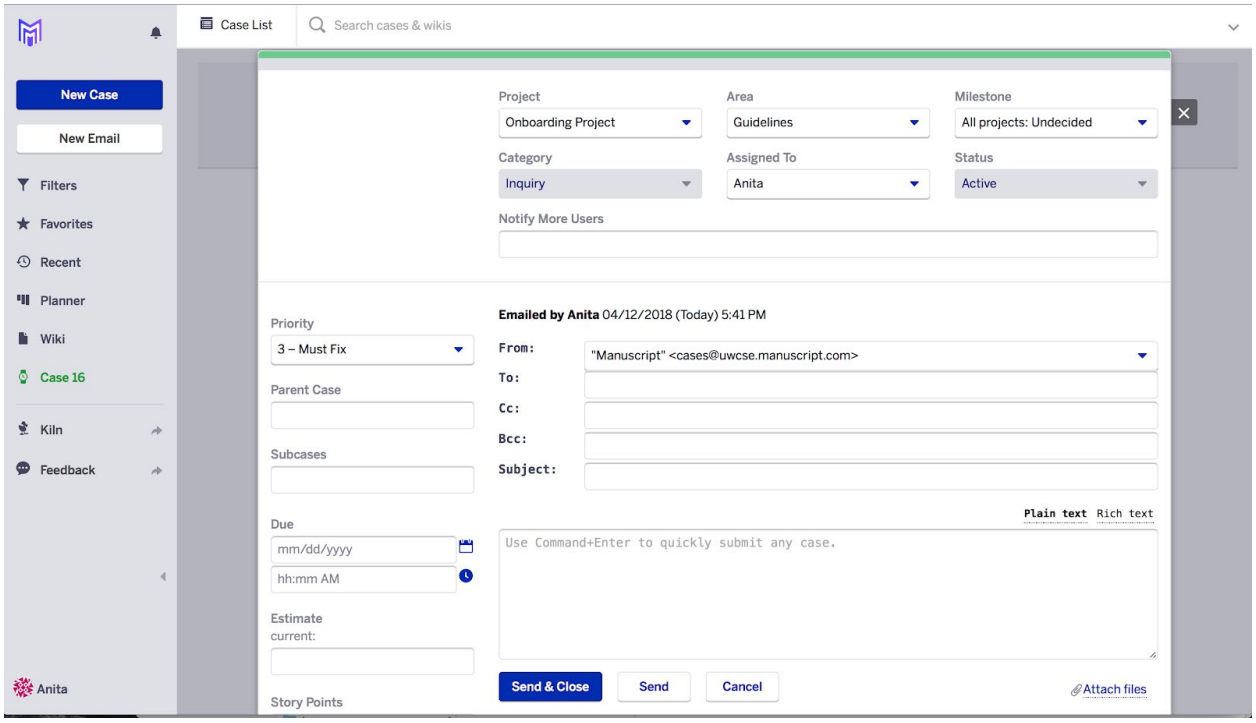

Fig. 5 There are features unrelated to effort estimation such as sending emails from this entity.

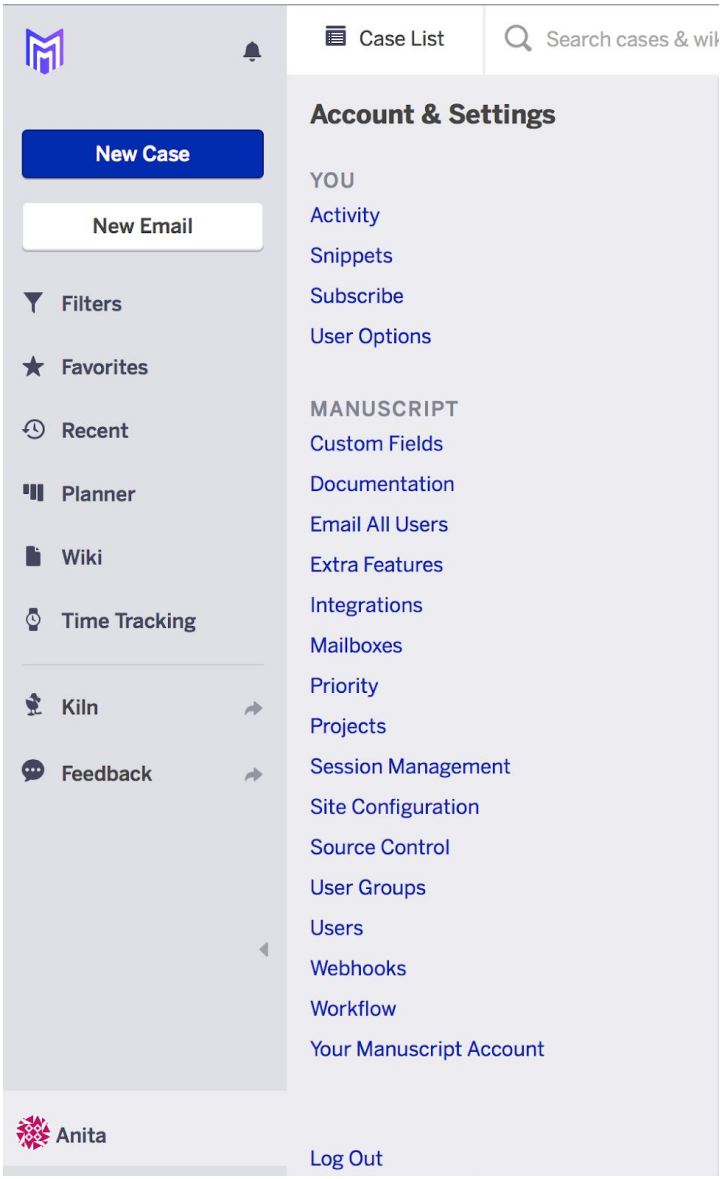

Fig. 6 There's an overwhelming number of features unrelated to effort estimation.

2) ProcessPAIR: A tool that analyzes performance data to identify performance problems and rank their root causes [9].

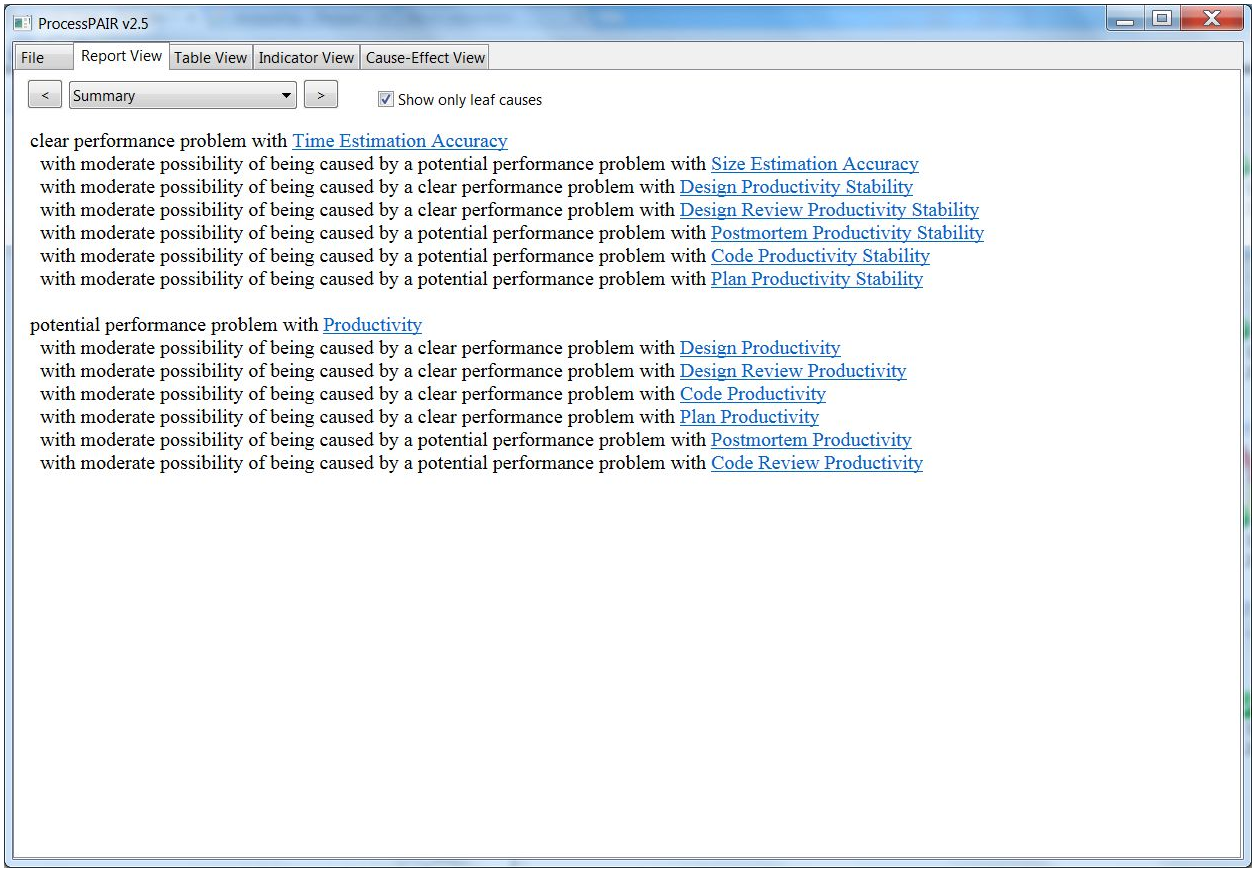

Fig. 7 Report view that shows the summary of performance problems in text. It's difficult to tell apart what has performance issues and why.

Indicator	Summary	Program 1	Program 2	Program 3	Program 4	Program 5	Program 6
Time Estimation Accuracy	**	1.19	1.96	1.41	1.17	2.54	1.77
Size Estimation Accuracy	***		1.81	1.37	0.67	1.40	0.79
Productivity Estimation Accuracy	**		0.92	0.97	0.57	0.55	0.44
Productivity Stability	**		0.99	0.97	0.57	0.54	0.43
Plan Productivity Stability	***		0.36	0.38	0.30	0.41	0.41
Design Productivity Stability	**		1.77	0.94	0.37	0.32	0.33
Design Review Productivity Stability	*				0.42	0.36	0.43
Code Productivity Stability	****		0.64	1.81	1.11	1.30	0.78
Code Review Productivity Stability	***				0.76	0.80	0.53
Compile Productivity Stability	**		3.10		0.69	0.34	0.67
Unit Test Productivity Stability	*****		9.44	1.18	0.86	1.17	1.13
Postmortem Productivity Stability	***		0.16	1.17	0.83	0.41	0.20
Estim. to Hist. Productivity Ratio	*****		1.08	1.00	1.00	0.97	0.97
Process Quality Index	*****	0.00	0.00	0.32	0.21	0.23	0.98
Defect Density in Unit Test	***	29	3	9	34	38	0
Defects Injected	***	86	45	77	59	94	67
Defects Injected in Plan	*****	0	0	0	0	0	0
Defects Injected in Design	**	57	42	64	17	0	22
Defects Injected in Design Review	*****	0	0	0	0	0	0
Defects Injected in Code	***	21	3	14	42	94	44
Defects Injected in Code Review	*****	0	0	0	0	0	0
Defects Injected in Compile	*****	0	0	0	0	0	0
Defects Injected in Unit Test	*****	7	0	0	0	0	0
Process Yield	****	73	92	88	43	60	100
Design Review Yield	***	0	0	79	100	100	0
Design Review Productivity	*			377	159	91	87
Code Review Yield	****	0	0	60	20	60	100
Code Review Productivity	***			210	159	151	96
Defect Density in Compile	*****	0	0	0	0	0	0
Defects Injected	***	86	45	77	59	94	67
Process Yield	****	73	92	88	43	60	100
Design to Code Ratio	****	0.46	0.17	0.46	0.84	1.46	0.98
Code Review to Code Ratio	*****	0.00	0.00	0.50	0.48	0.60	0.57
Design Review to Design Ratio	***	0.00	0.00	0.61	0.57	0.69	0.65
Productivity	***	37.2	36.8	35.9	20.9	17.6	13.3
Plan Productivity	**	2100	754	367	183	187	169
Design Productivity	*	175	310	232	90	62	56

Fig. 8 Table view of tasks with indication of performance problem. The table is not intuitive to understand. For example, there is no clear way of understanding what the numbers mean without reading a help guide.

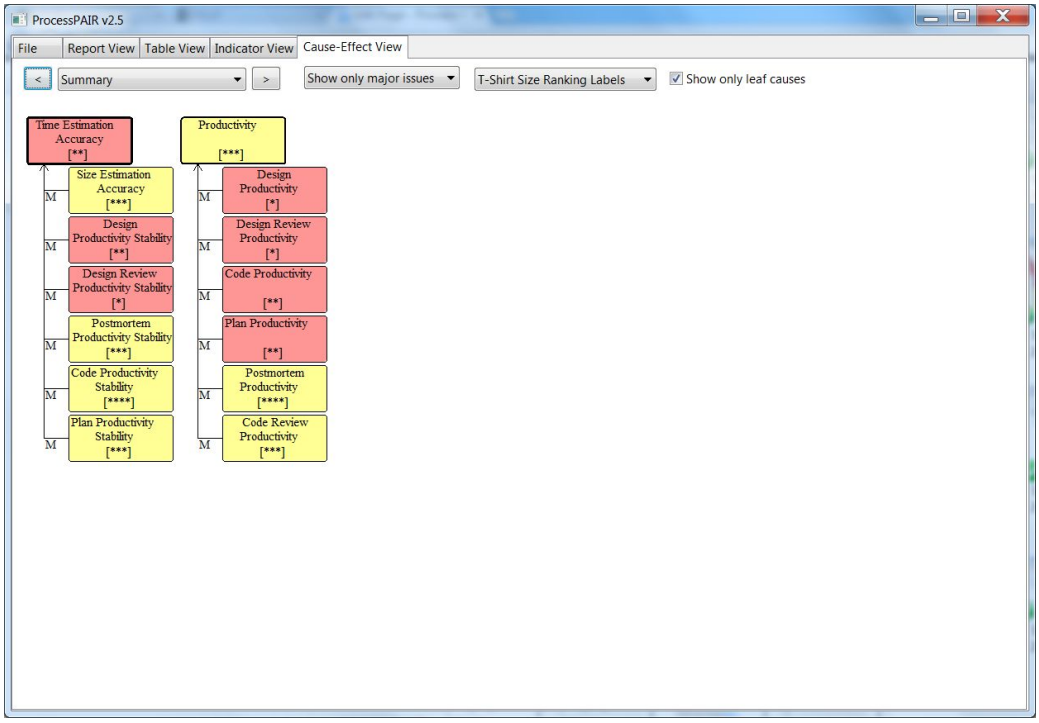

Fig. 9 This view prioritizes causes of performance problems. However there is not much information shown on why prioritization is in this way.

3) Process Dash: Process Dash is a project planning tool that tracks each team member's projects and lets individuals visualize their progress [10].

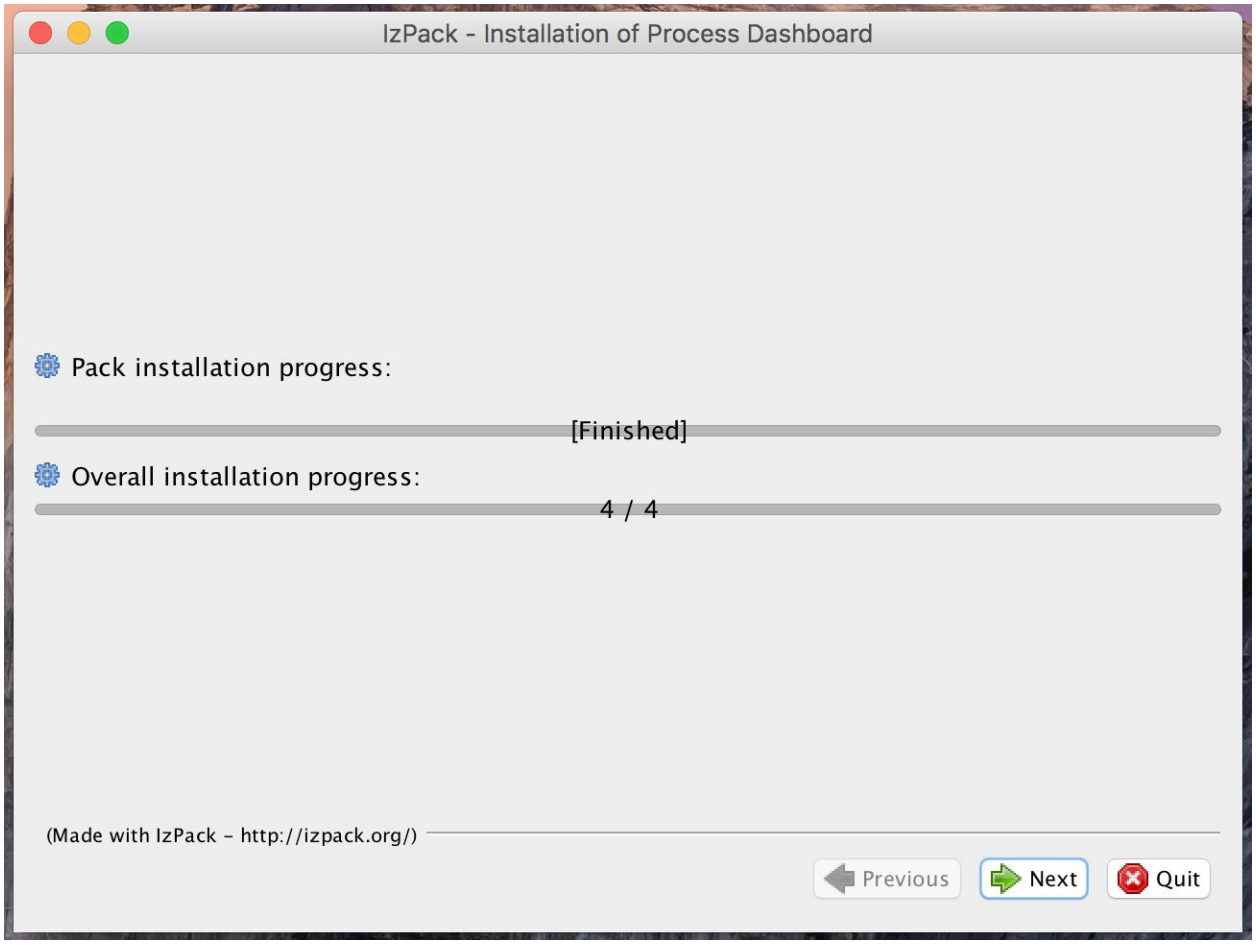

Fig 10. An installer is needed to download the tool, complicating the steps to access the tool.

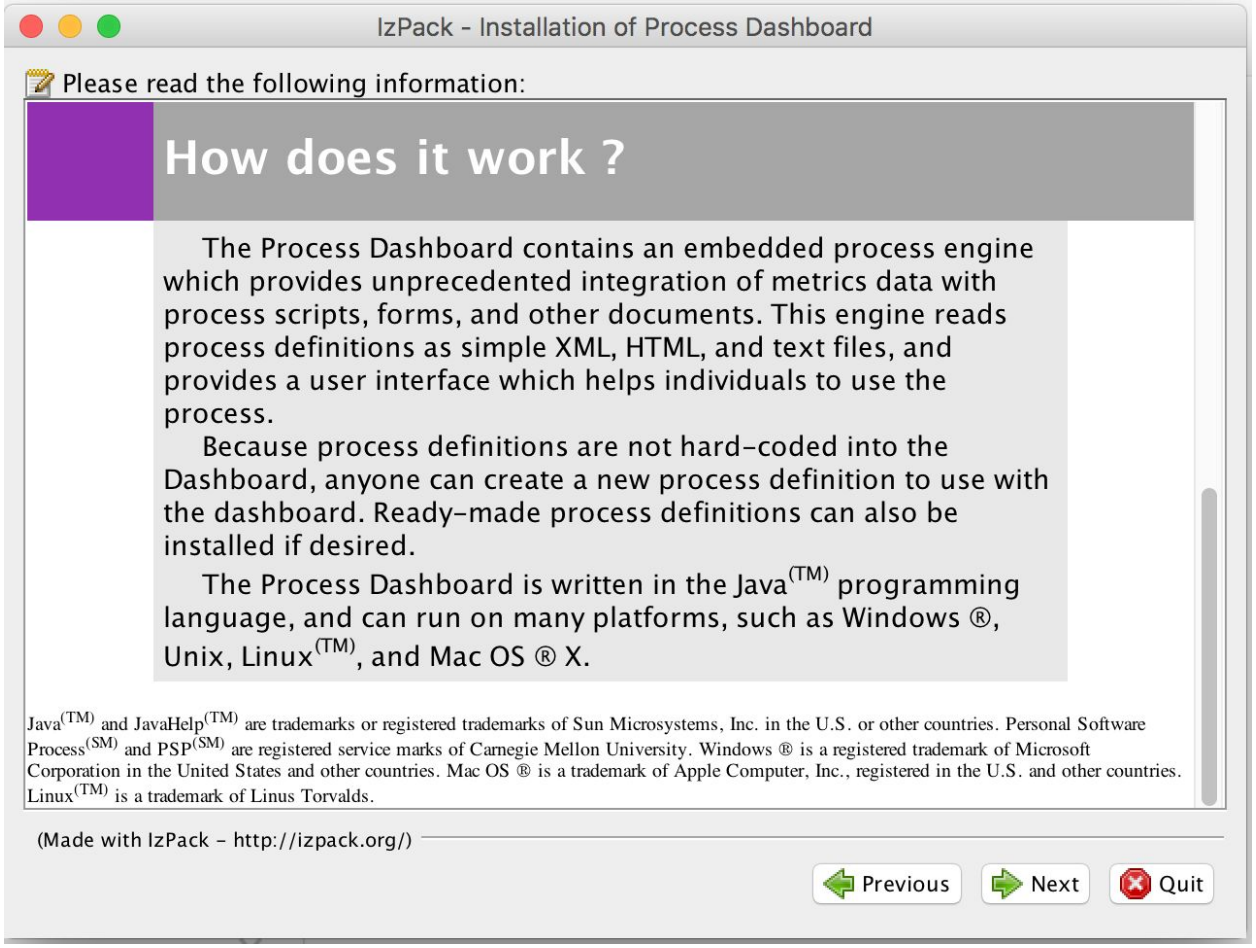

Fig. 11 Reading the user manual is needed to understand how to user the tool for its main purpose.

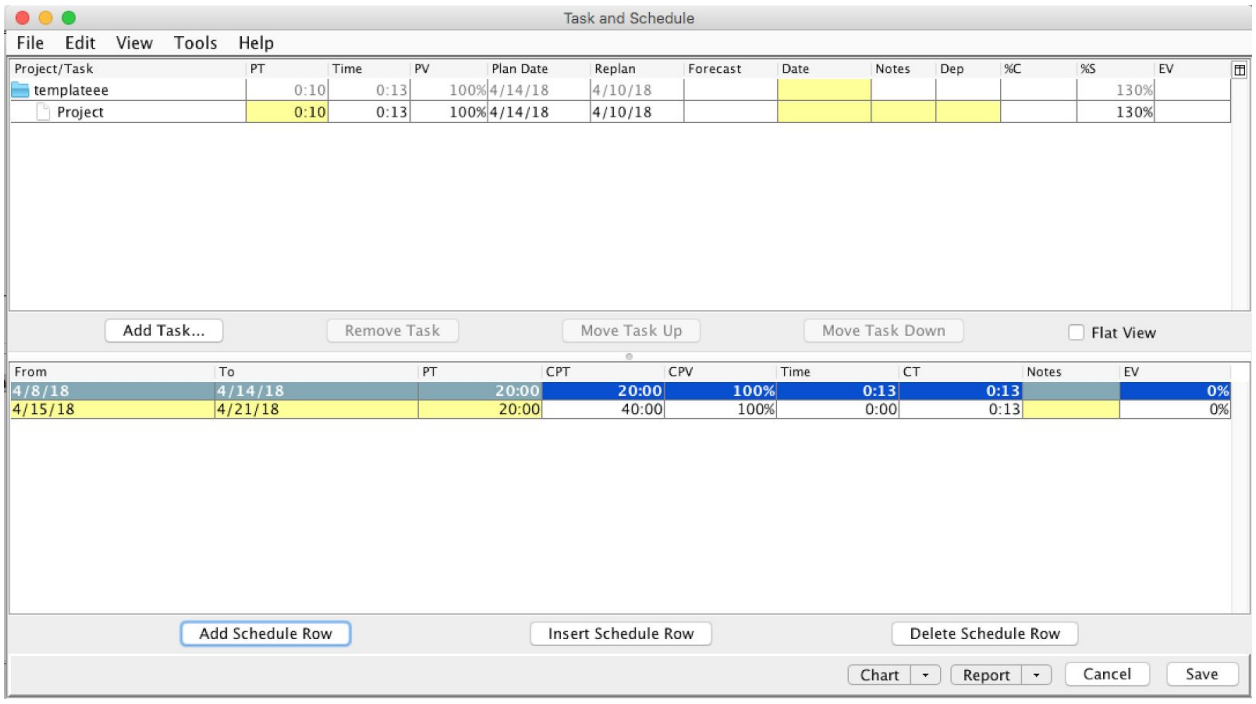

Fig. 12 The main interface of ProccessDash which is not intuitive to use. There is no indication of what to do after adding a schedule row.

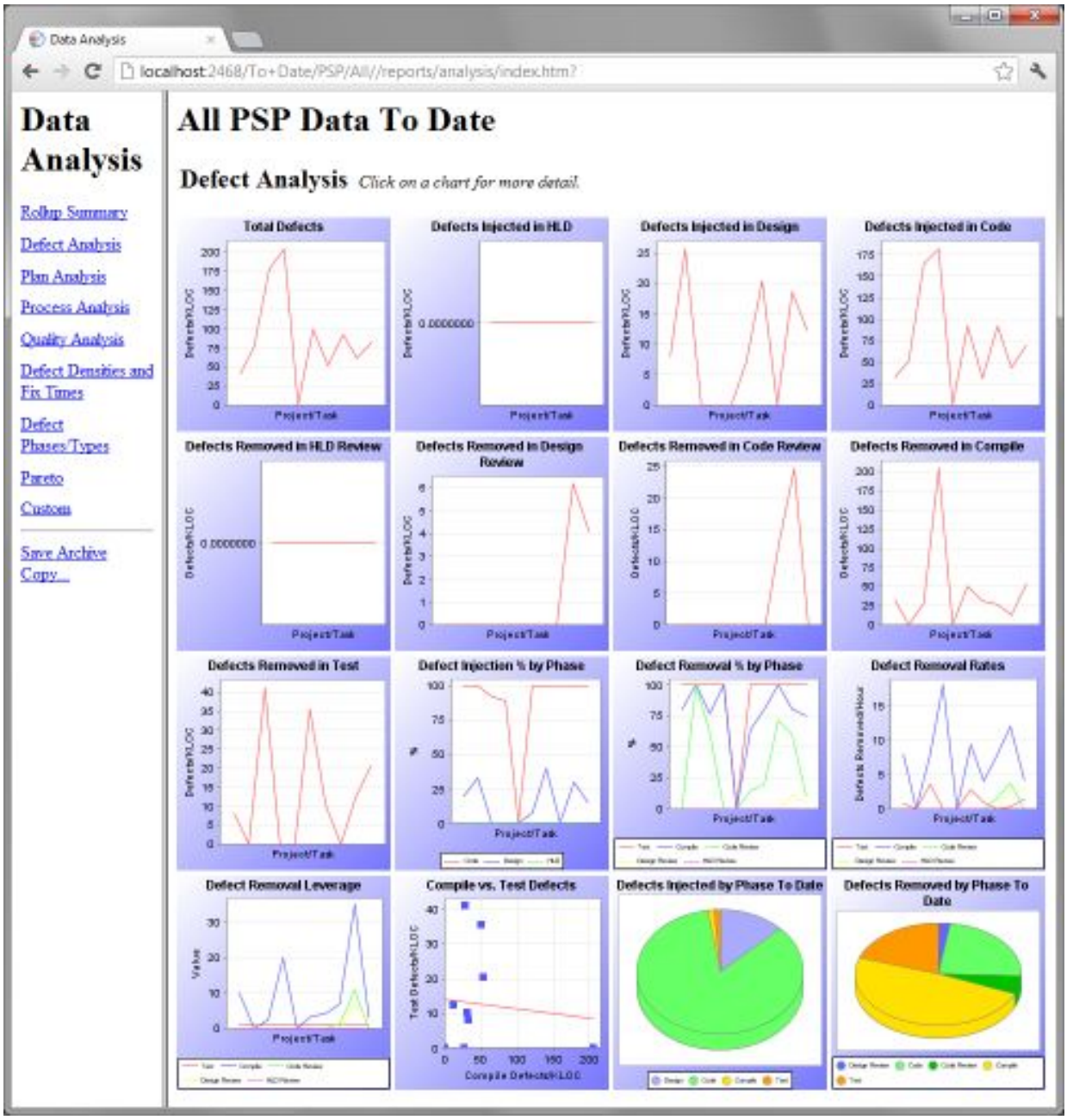

Fig. 13 An assortment of graph visualizations are shown all at once, bombarding the user with unnecessary and complicated data.

D. Existing Effort Estimation Processes/Models

Although there do not appear to be any popular standalone effort estimation tools, there are many existing effort estimation processes embedded into larger software project management products. There are two notable existing methods for effort estimation: PROBE and EBS.

1) PROBE: PROxy-Based Estimation (PROBE) is a component of a larger system known as the Personal Software Process (PSP) [6]. Developers using PROBE first break down a large project into smaller subprojects, then relate the size and type of each of these subprojects to subprojects (known as proxies) the developer has completed before. This knowledge is used to estimate the size, complexity, and time required for a large-scale project.

PROBE focuses primarily on how to break down and estimate large software projects, but does not emphasize integration into a larger team process. Additionally, PROBE is often embedded in the larger PSP, which has a high barrier to entry. Developers using the PSP often must read long books or take classes before they are able to effectively use the system. PROBE is often introduced as a whitebox process, and developers are consciously aware of the underlying algorithm driving estimation.

2) EBS: Evidence-Based Scheduling (EBS) is a method and piece of software for effort estimation developed by Joel Spolsky at FogCreek and has been integrated into their product called Manuscript [7]. Like PROBE, EBS first begins with the team of developers breaking down a large software project into smaller subprojects. Individual developers then cut these up into tasks they think they can complete in no more than a day of work. Developers estimate the amount of time each component will take. The EBS software keeps track of how much time the component actually takes and creates a model over time of how good an estimator the developer is. The EBS software then uses this model to predict how long the rest of the project will take and produces a probability distribution for the estimated ship date of the project.

EBS focuses less on how to break a project into smaller pieces and more on how to use effort estimates to produce projections for project completion that vary over time. While this is useful, we argue that the process of breaking a project down is just as important as estimating the time it will take a project to ship since it forces developers to think deeply about the larger structure of their project. Additionally, existing tools for EBS have poor visualizations of the data they produce.

3) COCOMO II: COnstructive COst MOdel II (COCOMO II) is a method for estimating software cost and time, generally for large organizations [12]. It consists of two models for estimating the cost and schedule of software projects: Early Design and Post-Architecture. The Early Design model is used to investigate alternative designs before a project's shape has been fully realized. The Post-Architecture model is for projects that are entering the coding stage of development. COCOMO II takes a collection of factors such as team cohesion, project novelty, and specification flexibility and produces an estimate of how much a project will cost and how long it will take to complete. Although detailed, the estimation process places a significant burden on those who use it. COCOMO II bases its numbers on 23 different factors [13], all of which must be considered by developers. While developers likely benefit from considering these factors, COCOMO II's complexity creates a significant barrier to entry for new users.

E. Our Approach

In our approach to software development effort estimation, we plan to first focus on creating a better user interface. Our goal is to create a new UI that would have a low barrier of entry, and would be easy to maintain. For example, each page allows the user to perform a singular main function. There is a page that focuses on organizing the users' projects, a page focusing on details of a task, and a page focusing on giving an overview of running tasks. Current estimation tools require users to follow the tool's many specifications. For new users, it would take a non-negligible amount of time to first learn how to use the tool and enter their data correctly before being able to actually use the tool to estimate a project's completion time. We plan to improve the interface so that users can quickly learn how to use our product to estimate their projects' completion times. We also plan to improve the way the data is visualized. The current tools provides a few graphs of the data that are not necessarily intuitive. Our plan is to improve upon the data visualization in our user interface using research on visualizing uncertainty from UW's Interactive Data Lab so that the analyzed information (e.g. ship date distribution) is easier to understand for the users [11].

Depending on the amount of time remaining after we complete the user interface, we will attempt to improve the effort estimation accuracy of our product (Fig. 1, Phase 2). This may involve taking cues from the COCOMO family of estimation tools, PSP/TSP, etc.

F. GUI Mockup

A limited interactive experience with this GUI is available at the link below:

https://xd.adobe.com/view/b2653ac4-d8e2-4d64-90e0-d5cb276f5fb3/

All current GUI screens are provided and described below:

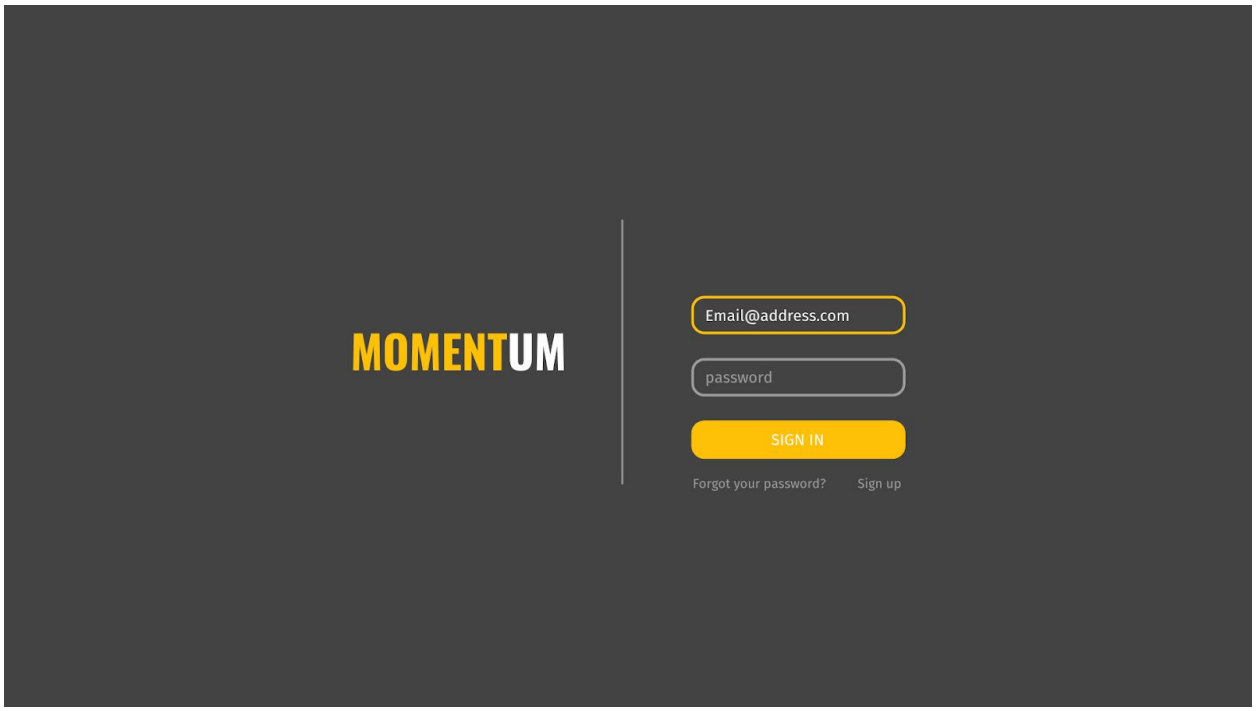

Fig. 14 Momentum Login Screen

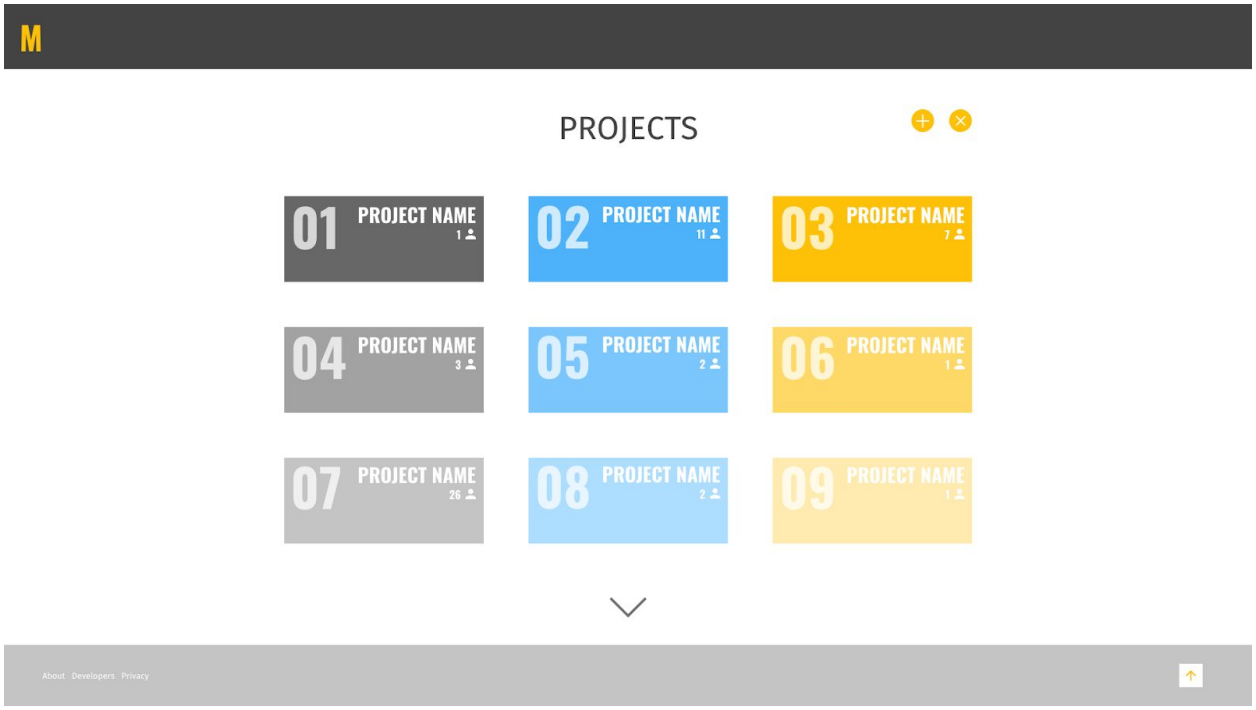

Fig. 15 An example landing page.

A landing page (Fig. 15) will allow the user to manage multiple project at a time. The user can click on any one of projects to get to project page of the project he/she have selected. To create a new project, the user can click on the plus icon and it will direct him/her to our new project page. To delete a project, the user can click on the X icon and then click on the project he/she wants to delete. The down arrow allows the user to scroll down his/her list of project. In each of our pages, there is an up arrow on the bottom right that will take the user back to the top of the page.

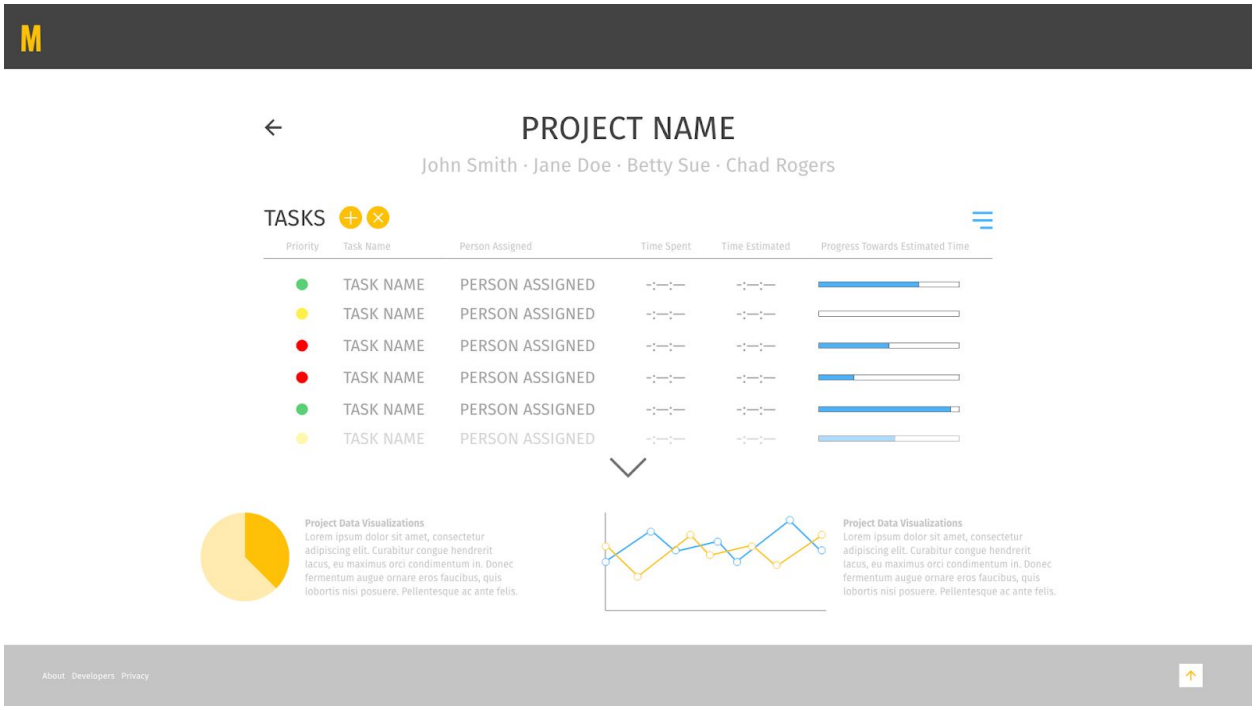

Fig. 16 An example project page.

A project page (Fig. 16) will allow the user to manage the project's tasks. The user can click on any one of tasks to get to task page of the task he/she have selected. To create a new task, the user can click on the plus icon and it will direct him/her to our new task page. To delete a project, the user can click on the X icon and then click on the task he/she wants to delete. The down arrow allows the user to scroll down the list of task. The icon above the progress bars allows the user to sort the project based on various parameters. The left arrow will take you back to the landing page. The name of the project is displayed at the top. Right below it is the names of the people who are part of this project. The graphs at the bottom of the page displays information about the project such as the distribution on when the project will be ready to be deployed.

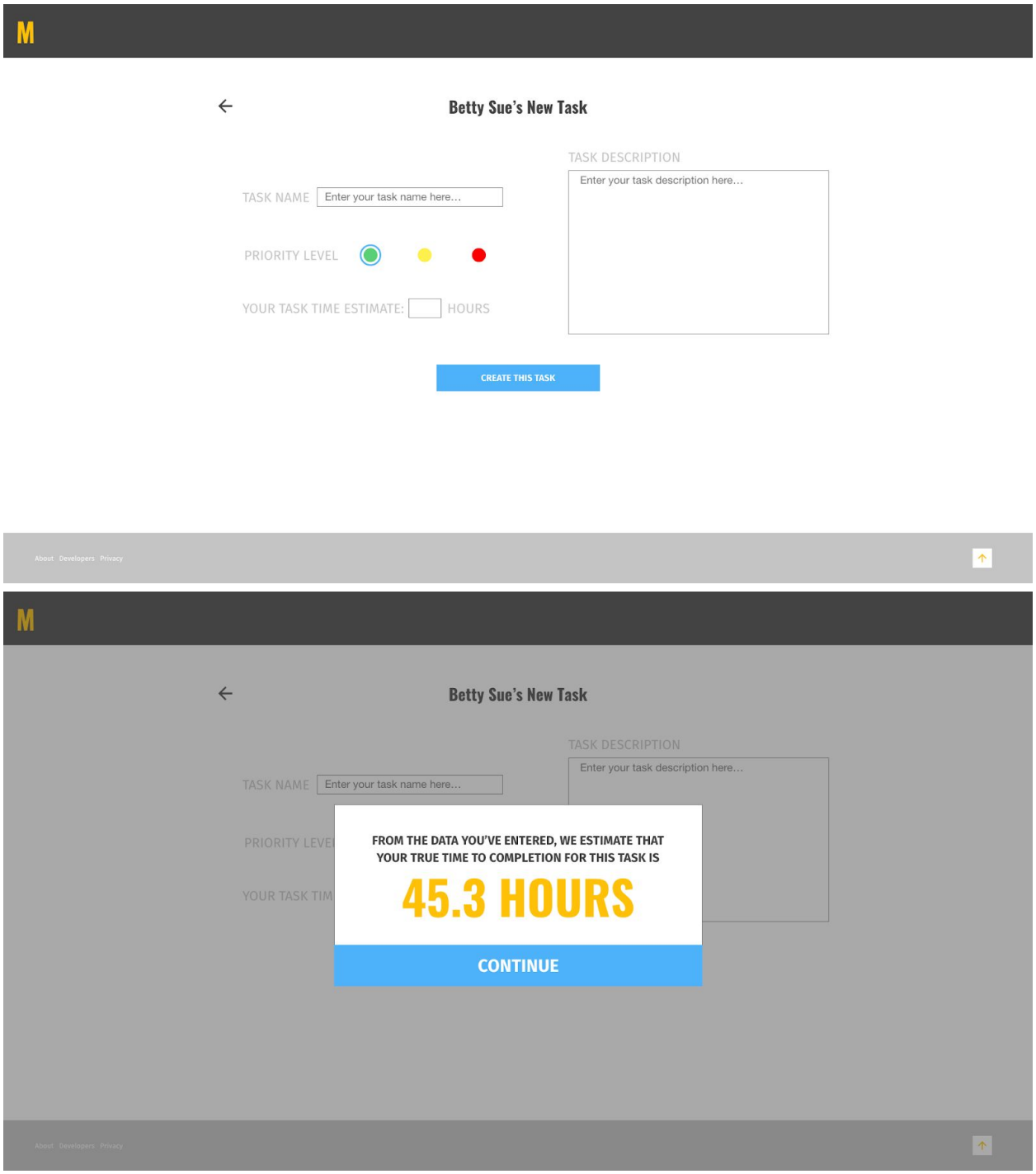

Fig. 17 An example net task page & estimated task time

To create a new task (Fig. 17, upper), the user fills in the task name, selects a priority level, gives an estimate for how long the task will take, provides a description of the task, and clicks the "create this task" button. After the task is created, there will be a pop-up (Fig. 17, lower) showing the user their true time estimate for the completion of the task. After the user presses "continue", they will be redirected back to the project's page. If the user decides to cancel the task creation, they can click the left arrow button to return to the project's page.

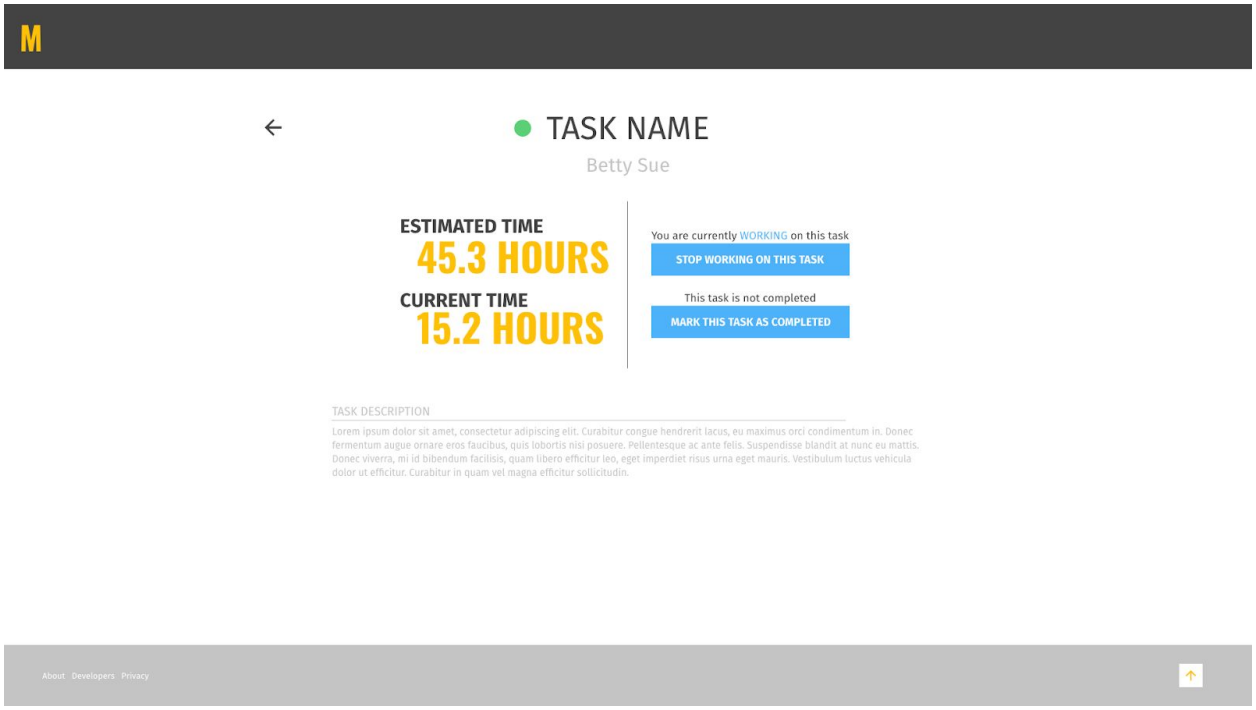

Fig. 18 An example task page.

A task page shows a task that is currently being worked on. This page shows the estimated time needed to complete the task as well as the time already spent on the task. The user can press the “start working on this task” button to begin or resume the timer when the timer is paused. Similarly, the user can press the “stop working on this task” to stop the timer when the timer is running. The button underneath is used to “mark this task as completed” after the task is complete. The name of the task is displayed at the top. The name of the person assigned to the task is displayed underneath the task name. The description of the task is displayed at the bottom of the page. The left arrow button takes the user back to the project’s page.

III. Challenges and Risks

A. Anticipated Risks

There are three large sources of risk in this project that we anticipate could cause us major problems:

1. Creating something too similar to another product.
2. Creating *another tool*. This is an issue that is core to our vision: we want to make something that people would actually use in the software development process. That being said, we don’t want our project to be another high-maintenance tool for developers and project managers to have to constantly maintain. This is, in many ways, antithetical to the process proposed in EBS and PROBE: for estimation to work correctly, you often have to manage historical data as well as break down specifications into anticipated tasks — things that are large tasks in and of themselves that may detract from a low-maintenance product.
3. Measuring the success of our product. Since our project focuses heavily on helping teams estimate the time needed for development of a particular software project, observing the actual usage of the deliverables we produce may be difficult (as the nature of this estimation is that it occurs over a longer period of time).

B. Anticipated Cost

We anticipate that we may be able to complete the basis of extending a better interface by week 7, which should leave us several weeks to add in additional features to the project.

C. Measurement of Success

TABLE I

ANTICIPATED TABLE WITH MEASUREMENTS FOR FINAL REPORT

Individual A results:

	Estimated Time	Actual Time	Accuracy
Task 1			
Task 2			
...			

Because an important feature of our tool is to tailor itself to the individual, it is best to conduct experiments that involve case studies with programmers.

To critique whether our project sufficiently solves time tracking and estimation problems for programmers, we intend to conduct user studies on peers in the computer science department and observe their behavior working with Manuscript, a competing tool. This way we can gauge what functionalities Manuscript have and lack.

In order to evaluate the success of our project, a possible experiment to check for success would be using small test cases to check that our project works as intended. For example, we can conduct experiments where the user passes in input, does their work as usual, and then we check whether the estimation given by the project is fairly close to how long it actually took for the user to complete the project. The table for measurements we could take is shown in Table 1. We would need to conduct this experiment multiple times with the same user to test whether the accuracy of the estimations improve over time, as well as experimenting with multiple individuals to test whether the program can fit engineers with various work styles. Our project would be successful if it is shown that Momentum's predictions of task has less than an average of 20% error after at least 100 inputs. The midterm "exam" would only test that Momentum would progressively estimate time accurately for one type of engineering work, while the final "exam" could include a variety of engineers who are completely new to the project and have different work styles. This would test both the usability of the project as well as its robustness. Ideally, our project would be tested by companies however this is not within the scope of the course.

IV. Proposed Schedule

Week	**Milestone**
Week 3	**- Locate case studies for project.** - Gain access to source code for software management tools that employ time prediction. - Continue research on existing time estimation strategies.
Week 4	**- MVP (minimum viable product) for new user interface and time estimation backend** - Investigate user research and evaluation methods. Conduct user research on competing tools
Week 5	**- MVP for additional workflow enhancements.** - Establish metrics for software success and schedule user research.
Week 6	**- Continue working on MVP's additional workflow enhancements.** **- Decide on the "vertical" direction to take beyond first MVP.** - Vertical directions: Potentially domain-specific - Take lessons from MVP and improve it.
Week 7	**- Complete "draft" of project.** **- Conduct first round of user testing to evaluate the effectiveness of our tool compared to existing solutions.**
Week 8	- Project draft refinements based on user testing
Week 9	- Begin project presentation work - Second round of project draft refinements based on user testing
Week 10	**- Finalization of refinements** **- Finalization of project presentation**
Week 11	**- Complete project presentation**

REFERENCES

[1] M. Jørgensen, “What We Do and Don’t Know about Software Development Effort Estimation,” *IEEE Software*, vol. 31, no. 2, pp. 37–40, Mar. 2014.

[2] M. Jørgensen, “The effect of the time unit on software development effort estimates,” in *2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, 2015, pp. 1–5.

[3] H. Barki, S. Rivard, and J. Talbot, “Toward an Assessment of Software Development Risk,” *Journal of Management Information Systems*, vol. 10, no. 2, pp. 203–225, Sep. 1993.

[4] I. Benbasat and I. Vessey, “Programmer and Analyst Time/Cost Estimation,” *MIS Quarterly*, vol. 4, no. 2, pp. 31–43, 1980.

[5] K. Srinivasan and D. Fisher, “Machine learning approaches to estimating software development effort,” *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 126–137, Feb. 1995.

[6] W. Humphrey, *The Personal Software Process (PSP)* (Technical Report CMU/SEI-2000-TR-022). Pittsburgh: Software Engineering Institute, Carnegie Mellon University. 2000.

[7] J. Spolsky, “Evidence Based Scheduling,” Oct. 2007

[8] “Manuscript - Project Management for Software Teams.” Manuscript - Project Management for Software Teams, www.manuscript.com/.

[9] “Introduction to ProcessPAIR.” ProcessPAIR, www.blogs.fe.up.pt/processpair/tutorials/intro/.

[10] “Functionality for Individuals.” The Software Process Dashboard Initiative, http://www.processdash.com/functionality-personal.

[11] Kay, Matthew, et al. “When(Ish) Is My Bus? User-Centered Visualizations of Uncertainty in Everyday, Mobile Predictive Systems.” UW Interactive Data Lab | Papers, ACM Human Factors in Computing Systems (CHI), 2016, idl.cs.washington.edu/papers/when-ish-is-my-bus/.

[12] “COCOMO II Model Definition Manual.” USC, http://csse.usc.edu/csse/research/cocomoii/cocomo2000.0/cii_modelman2000.0.pdf

[13] B. Boehm *et al.*, “COCOMO Suite Methodology and Evolution,” April 2005. http://csse.usc.edu/TECHRPTS/2005/usccse2005-509/usccse2005-509.pdf

META

A. Hours Spent on Assignment:

```
    Leon Pan: 4 hours
  Haley Ruth: 4 hours
 Anita Leung: 3 hours
Josh Pollock: 3 hours
   Austin Ha: 3 hours
```