MOMENTUM: Project Proposal

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: project estimation is extremely important, but is sometimes not taken very seriously by software teams as it can be difficult to manually estimate how long chunks of a project will take each person in the team and how this will affect the team as a whole. On the other hand, in-depth project estimation takes significant time and resources as well as consistent participation from the whole team. We believe that we can build an accessible product that would address this unrelieved problem of software estimation with quick and accurate estimations.

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 project estimation. Overall, we believe that the implementation of our product would improve the overall experience of software development teams as well as promote pragmatic programming practices.

II. Approach

A. Goals

Our project involves extending an established tool that uses information gathered about a user to predict their work schedule and better estimate the timing of their software development process. The planned approach is to focus on enhancing existing estimation tools and incorporating a better user interface, and then further build other features if we complete our milestones early. Momentum will be the only tool that is simple to use, does not require high maintenance, and is cheap and accessible for the public.

B. Existing Solutions

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

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 where developers must consider

Evidence-Based Scheduling (EBS) is a method and piece of software for time 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 time 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.

C. Our Approach

In our approach to project completion time estimation, we plan to first extend one of the existing software tools (listed above) by focusing on creating a better user interface. Our goal is to create a new user interface that would have a low barrier of entry, and would be easy to maintain. Current estimation tools require users to manually input all their data into their tool while following the tool's many specification. 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 could quickly learn how to use our extension to estimate their projects' completion time. In addition to improving the way data is entered into the tool, 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 so that the analyzed information is easier to understand for the users.

After we complete the new user interface, we plan to reduce the amount of manual input required from the user to form a good estimation. The current tools require users to manually log all of their data. Some of the required inputs include breaking down the project into individual tasks, making personal estimations on how long it will take to complete each task,

keeping track of the time spent on each task, and then inputting the estimation and the actual amount of time taken by the user. Having to manually input all this data is tedious, which turns the users away from using the estimation tool. There are many ways to reduce the amount of manual input from the user. We could attempt to automate the estimation time for specific tasks based on various parameters such as task type or an estimation of the number of lines of code needed to complete the task. We could also improve upon the way time is tracked for each task. Instead of having to manually track the time spent and having to manually log the time taken afterwards, we could implement a way to automatically enter the data on the task once the user specifies that the task is complete.

We could also attempt to optimize task assignment and scheduling so that each member of the project is assigned tasks that closely reflects their strengths. The team member that has more experience in a certain task is likely to be able to complete the task faster than another member who has less experience in that task. By appealing to each member's strengths, it is likely to reduce the completion time of the project.

Depending on the amount of time remaining after we complete the user interface, we will attempt to implement one or more of these ideas to improve the estimation process. Reducing the amount of manual input required from the user reduces the time spent on the estimation process and makes the tool easier to use. By making the tool less tedious and more intuitive, we hope it makes the tool more appealing and accessible to the general user.

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

Individual A results:

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

Figure 1. *Anticipated table with measurements for final report.*

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. 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 above. 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	**- MVP for new user interface and time estimation backend ("horizontal direction")** **- 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	**- Decide on the "vertical" direction to take beyond first MVP.** - Vertical directions: Potentially domain-specific - Take lessons from MVP and improve it. - Investigate user research and evaluation methods.
Week 5	**- MVP for additional workflow enhancements.** - Establish metrics for software success and schedule user research.
Week 6	
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

META

A. Hours Spent on Assignment:

Leon Pan : 2 - 3 hours
Haley Ruth : 3 hours
Anita Leung: 3 hours
Josh Pollock: 3 hours
Austin Ha: 3 hours