Budding Budget

owsenk	Kyle Owsen
lizs3434	Elizabeth Schibig
ischaaf	Isaac Schaaf
jbktsang	Jessica Tsang
hstefan	Stefan Holdener
mjsc	Maxton Scott Coulson

Process Description

Software Toolset

Since we want our app to be available to the largest audience possible, Budding Budget will be developed using HTML, CSS, and javascript with AngularJS, and then ported to native apps for Android, iOS, and Windows Phone using PhoneGap. This will allow us to reach a wide audience without having to create separate versions for different operating systems, and allows us to create the app using primarily Javascript, an easy to learn and very flexible language.

The web backend that syncs data between the app and the cloud database will use MySQL with Node.js for the external API. This will give us some flexibility, and integrates well with PhoneGap. Since we are all familiar with GitHub, we plan to use that for version control, as well as some bug tracking.

Group Dynamics

At this time, it is difficult to determine exactly how we are going to split our team into smaller groups. We plan to, once we know more clearly what each part of the project entails, split into groups focusing on the app itself, the backend database, and potentially a third for the analytics webpage.

Since none of us have extensive design experience, or a particular specialization in any of these fields, we expect the membership of each of these groups to fluctuate. Additionally, we believe that more fluid team design - potentially swapping out roles week to week - will allow a more even distribution of the workflow, since teams will likely need more people some weeks than others.

As we only have a six person group, we've opted not to designate a team specifically dedicated to testing. Instead, unless otherwise specified, developers are expected to handle tests of both their implementation details, as well as adherence to requirements.

If there's some disagreement about the requirements or implementation of the app, and the two parties cannot come to a resolution, it will be put up to a team discussion and vote at our next meeting. If a consensus cannot be reached, both teams will come up with an appropriate game of skill or chance to decide the outcome.

Schedule/Timeline

Week of 1/18: Discuss and finish software requirements specification
Week of 1/25: Software design specification & Design presentation slides. Research more technologies and think about integration
Week of 2/01: Practice presentation & Finish coding for zero-feature release
Week of 2/08: Zero-feature due, continue to code for days, flesh out features
Week of 2/15: Finish beta release, prep for beta demo
Week of 2/22: Feature complete release due
Week of 2/29: Release candidate due
Week of 3/07: Final release due, prep for presentations

Risk Summary

Our biggest risk is that the user will not find the app compelling. With most apps, if the user does not use them for days or weeks at a time, they can come back to the app later with little or no issue. If they skip budget tracking for just one day, the information given by the app is no longer valid.

We plan on mitigating this risk by forcing the user to either update their assets, or confirm that the budgeting information is still correct if they miss a day of tracking. We plan to do extensive user testing so the tracking is easy and appealing to the user. If we are unable to overcome this risk, there isn't really a feature we can cut to fix it, which is why it's the most important risk.

Another potential risk lies with the algorithm used to track spending. Though we don't believe it will be too complicated, the number of variables and potentially vague ending date of the calculation could make it more complex than we think.

We can manage this risk by creating the algorithm early in development, so we can adjust the direction our development goes based on any shortcomings in the process we create. If we

cannot make a satisfactory algorithm, we'll have to cut back on some of the more ambitious elements of the app, like an indefinite end date, or considering recurring charges.

The addition of a cloud sync component to the application adds a new risk - there's a chance that if a user has two devices connected to the same account, their information could get out of sync and confuse the database.

Our plan for this is to settle conflicts based on whatever syncs to the database first, and to force the other device to pull data from this sync. Since we see the user tracking a budget on multiple devices at the same time as an edge case, this solution being less than elegant from a user perspective is fine so long as the database remains consistent. If we cannot make this work, we'll need to either only allow one device to be associated with an account at any one time, or cut the online feature entirely, but these options should only be seen as a last resort, doomsday scenario.