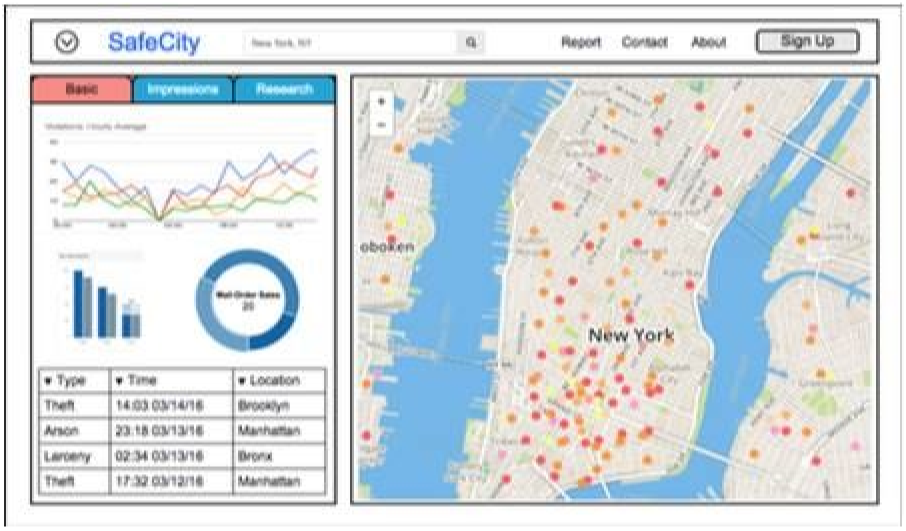
SafeCity Final Report



▾ Type	▾ Time	▾ Location
Theft	14:03 03/14/16	Brooklyn
Arson	23:18 03/13/16	Manhattan
Larceny	02:34 03/13/16	Bronx
Theft	17:32 03/12/16	Manhattan

Stefan Keselj, Eric He, Kavinayan Sivakumar

05.10.2016
Hansen Zhang
Dr. Christopher Moretti
COS 333: Advanced Programming Techniques
Princeton University

The purpose of this document is for us to give an overview and reflection of the steps we took to make SafeCity a reality. Notice that while the Product Guide has a screenshot of our actual product as its cover, this document has a screenshot of the initial mockup in our design document as its cover. There were two fundamental types of skills we gained and lessons we learned while working on SafeCity: related to project management and those related to software engineering. Both types were essential, and we were all in both types of roles throughout the project. We structure our discussion by addressing these two themes, and then offering some feedback on how to make the course better at the end.

Project Management

Changing Milestones

As the largest programming project any of us ever took on, SafeCity was an intimidating feat to set out to accomplish. Since the start, there was much uncertainty about whether or not our product was useful at all or feasible in a mere 7 weeks. The only way we could placate our nervousness about this was to diligently plan every stage and frequently update our plan as we went along. Our motivation for this was that if something was deeply wrong with our project, we would certainly want to know sooner than later, and the only way to find out something is wrong earlier is to constantly consider how the project could unfold from each point forward. As a result, our project experienced many tweaks to its trajectory along the way, which ultimately ended up enabling it to be the best it could be. In this section we briefly overview the most important tweaks of our project's path.

Since the start of our project, we have always had the basic conception of the application layout; the similarity between the initial proposal mockup on the cover of this document and our actual application page is substantial. However, as seems to often be the case, we were slightly over idealistic and ambitious about our project's scope at the start. In terms of breadth of data, we had initially planned to collect the data and build a visualization for all the crimes in the US, so that a user could pan and zoom out over the entire country and see the crimes that happened over an entire year. Furthermore, two of us wanted to show off the experience we had gained from courses in Machine Learning and Artificial Intelligence, and so we originally planned to include a research tab which featured more advanced crime prediction tools of potential use to the authorities or researchers interested in crime.

As we began our Prototype, we realized that merely cleaning large amounts of data is tedious, and that loading huge datasets to and from a database was often impossible given our computational resources. Furthermore, we realized that the US did not have one centralized source of crime statistics which we could scrape from and use to cleanly populate our graph, different cities did things completely differently and only the large cities had data. It was at that point that we decided to narrow the scope of our data from the entire US to only the large cities which provided good data. This proved to have been a beneficial decision because it stopped us from spending disproportionate amounts of time on data collection and management, and allowed us to focus on the more valuable facets of the product, from both a learning and business perspective.

Once our prototype was completed we had some basic data to work with and a basic visualization and were ready to tackle the meatier and more sophisticated analytics techniques that excited us so much. But a harsh realization that came to us after we spent the better part of a weekend struggling to merely get a basic and ill-formatted pie chart to work was that data representations are hard to get perfect on a web platform. There are so many nuances to display, and feeding in the data through the middleware was not as simple as a task as we imagined. There certainly was a big difference between hitting enter to load a local csv file onto an iPython notebook like we did in our aforementioned Machine Learning class, and writing an entire system to upload, import, and manage a database worth of data. The infrastructure for the machine learning algorithms we had initially envisioned was simply too costly, and our time would be better used fleshing out a system that showcases the skills we learned in this class.

Process to Make Progress

Perhaps one of the most valuable take aways from our work on SafeCity is our better understanding of what it truly means to have a process to make progress. At the start of the class, Dr. Moretti's discussion about how a process to make progress was essential to success in the course really resonated with all of us. We had all experienced the stress of having to pull an all weekend hackathon for a 226 assignment after not going to TAs the week before, or struggling to debug code for a 217 assignment and feeling stupid for not taking the time to factor code properly. At the start of the project, we all agreed that we would not fall into the trap of being lazy programmers, and would meet every weekday for 3 hours to code and then be stress free when the TA meetings came around at the end of the week.

Once more, we were slightly too optimistic about the way things play out. We did meet every day for 3 hours, but even with all that time we often found ourselves stressed to meet a TA deadline because our project was relatively ambitious. In this environment, we would often make compromises to the initial standard of excellence in programming we had hoped to uphold. It's just that when you're chasing a bug until 3 AM and then the application finally works, you aren't really concerned about whether or not the code is reusable or modular or scalable, it works! But then later these loose ends come back to haunt you, and cause another stressful situation, which causes more sloppy coding, and the downward spiral only continues.

The flaw in our process was two-fold: we were short sighted and we weren't targeted enough. Short sightedness is most clearly displayed through the situation described in the paragraph above. The reason we didn't view these situations as terribly alarming was that we were under the impression that was simply the way software engineering got done. Last summer, one of our members got acquainted with the software development process at IBM, and said that the most popular form of project management these days involved sets of "sprints" where an entire team works really hard for a short period of time to get a specific goal done. However, this exact methodology wasn't the most appropriate way for us to approach our project. This is because a typical IBM engineer's version of a sprint is likely much different from our version of one. Namely, an IBM engineer always gets to go home at the end of the day and put down their work, but we are more inclined to let work spill over into the evening and throughout the night, which resulted in overexertion and some bad programming deicisions. In our case, what we would have benefited from is a focusing of our efforts and limiting of our time so we could be more like the IBM employee. That is, instead of going into the night with the attitude that we're going to stay in the lab as long as it takes to get a certain feature done, we could meet in the middle of the day with the attitude that we are going to work as hard as it takes to get a certain feature done within this time frame.

Teamwork

Some of the most hard-hitting lessons we learned during the course of this project did not involve computer science or technology at all, they were purely about the dynamics between people in an extended project like this. Coming into this class, we all knew each other as acquaintances, but never quite spent as much time together as good friends might. As one would expect, this all changed when the project became underway and we spent the majority of our evenings together in the CS Tearoom coding. Overall, the project brought us together as friends, but this did not come without its fair share of rough patches. The source of these was simply the high stress situations that we found ourselves in and the sometimes clashing approaches to the project that we took, sometimes even clashing personalities that we brought to the project. These differences were further aggravated by the fact that we are all essentially peers and one cannot tell the other what to do, even when a member of the team is not exactly pulling his weight or is doing something in a way the other two do not agree with.

The lessons we draw from this seem obvious, but were not at the top of our list of priorities when we were starting the project because we viewed this as a technical project by nature. This was certainly not the only, or even the main, side of what happens behind closed doors to get a 333 project off the ground. The first lesson we learned is to always keep as clear and fluid of a communication channel as possible open between all team members. From our experience it is very often the case that different team members have different expectations and goals to a task, and although the team may discuss ideas aloud and vaguely agree with each other, they don’t actually reach a consensus unless each of them actively participates in a targeted discussion to resolve an issue. Unfortunately, this second form of soft agreement was incredibly common among our team members. It is because confrontation is unpleasant, so to avoid it we loosely agree while actually holding a contrasting opinion. But this only further contributes to the problem, because as these contrasting feelings sit and continue to brush up against one another, they simply grow bigger and bigger until they burst. In this sense, we found it was much better to be upfront with each other about what we really thought. We are not claiming that we did it perfectly or are even doing it perfectly now, but it is safe to say that we all have a newfound respect for reducing long term friction among group members by being upfront.

TA Mentorship

While working with teammates was a lesson in communication and more nontechnical aspects, working with a TA showed us the objective aspect of software development and how in industry, your progress is determined by specific milestones and targets that you have set with a superior. Although we had a project leader, having a TA as a project manager distant from the coding motivated us to achieve our goals. While if we had just had a project leader, we might have convinced ourselves that everything was fine when we did not hit milestones, as the project leader is technically still a member of the development team, and their personal bias towards the issues and bugs that come up could affect the outlook and motivation of the team. However, with the TA, we knew that there was little room for missed goals or excuses, and this resulted in us desperately accomplishing our goals for each week. This had a great benefit for us as by keeping the ship rolling, we avoided procrastinating on the project and could keep the morale high, as we could see our project take shape week by week.

Another great aspect of talking to our TA was having them be our first user and have them work with the application through all of its developmental stages. Usually with testing the application with users, they are presented the final MVP, after all the designing has been done previously by the developers. With the TA interacting with our application each week, certain design and feature decisions could be made without having to try one before another and well before work was put into it, saving us time and effort. Conversely, this also taught us the importance we should give to user feedback. Our TA Hansen did a good job weighing his feedback. If something was of low consequence, he gave it a lower priority than others that were more important for the overall application. In this way, we could prioritize changes that had to be done.

Software Engineering

Technology

The decision to use Meteor to build our application was rather spontaneous and not as grounded as it should have been. After hearing the framework being introduced in lecture, it seemed easy and simple enough to use. Also, it had a MongoDB backend, perfect for what we were trying to do, right? Well not quite. Meteor, although built in with a lot of convenient functionalities, also makes a few simple things, hard. Learning Meteor along the way meant that work that made the team members proud the previous week could turn out to be completely wrong the next week, requiring the entire structure to be redone.

When we first got our data from our Mongo collection to show up on the Google Map, we were ecstatic. At that point, we were only using a sliver of our total data, a test file that represented a mere 300 points. When we got every single point to show up on our map, we thought we were golden. However, once we upped our test file to represent 1500 points, and even more, we noticed the user would have all the data points on the client side, something that was unsustainable for the amount of data we wished to host on our database. After many searches on Google and StackOverflow, we found the proper way to manage the client side data was through publications and subscriptions. Even though we found the answer, we had to completely redo the way we stored data.

However, this would not be the last time that we had to make big changes to the application. Two huge problems that we encountered were how to make the application elements load only after the subscription was finished processing, and how to remember certain variables after reloading the page. Both of these problems were eventually solved by Iron Router, but it was not until we struggled with multiple packages such as persistent-session and Select2 that we finally found the solution.

In general, it was rather frustrating when it seemed that something which was expected to be built into Meteor, was not. This was very apparent when struggling how to implement the functionality Iron Router provided, but more generally, forced us to have heavy interaction with Meteor packages. Packages, in Meteor, were both blessings and curses. Our project primarily relies on a package that lets us use Google Maps in Meteor, which without we would probably be at a complete loss at what to do. Our application, at the end, relies heavily on around ten or so packages. However, packages can turn out to be nightmarish if not used carefully. For example, the datetimepicker package we choose worked will for its functionality, but caused an inexplicable formatting error that caused our homepage to be able to scroll downward seemingly forever. There was absolutely no good reason for the bug, until after some very meticulous CSS debugging that it was found that a element in our CSS just so happened to conflict with an element in the package. Using packages can be a headache, and sometimes random bugs that make no sense will just pop out of nowhere.

Testing

Testing was something we had a good deal of experience with due to the previous assignments in the class, and this project challenged us to use our full kit in this regard. We unit tested any new feature along the way, ensuring that when we added something new, it did not break anything else. There were certain instances where bugs showed up, but we detected all these when we unit tested, so debugging was an isolated process on a specific piece of code where we knew where the problem was. Since our application is very data heavy, we first tested our features on a small piece of data, and then overloaded it with lots of data. In this case we were able to check performance, and even improve performance for our overall application.

In one situation, we had a bug where we thought it was IronRouter, as that was the newest feature we had added. As a result, we felt like since we were unit testing along the way, that must be where the error was. It took us about an hour to finally figure out that the problem was in a script we were launching as when different people had pushed to github, we were using the wrong script file, which was causing the bug. As a result, our testing mechanism was solid enough to first suspect a possible feature, but we were flexible enough as a team to do some further debugging and realize where the actual error was.

Testing also played a part in our teamwork. Since each one of us was usually working on a different aspect of the application or feature, we assumed that if someone pushed to master, then their code was spotless and worked. As a result, we had to rigorously test our code before we pushed, as if there were still errors, that would waste time for the other members of the team who might experience bugs due to code that's not their own. Testing created a healthy environment of accountability that we all felt mirrored something we might face in an actual workplace.

Design

Design was something we valued a lot throughout the entire project, but did not get a chance to fully implement until the final week before the demo. While we did have the overall layout of the website set from week 2, we didn't necessarily have a color scheme or fully know where things were going to be. All we knew was that we wanted the design to attract the user and complement the easy to use user interface.

It was this component of the project that really showed the importance of a diverse skill set within a team, as all of our three project members were primarily backend and sometimes middleware developers with little experience in frontend design. We all had to learn from scratch on how to make things look nice, whereas other groups might have had a specialist in front end. Although we believe that we succeeded in having a nice design, more prior knowledge of front end design development would have gone extra miles in regards to our project in the design.

However, the most important thing about design and what we did well was that we let the user interface dictate the design, and not the other way around. It is an important engineering doctrum that form follows function, and the same can be said for our application. We wanted the user to have a much easier time navigating through our application than through an existing rival's, and we made sure our design choices still allowed for a fast learning curve.

Product Iteration

One software development technique which we found was extremely useful in keeping our project on track and was product iteration. Since the start, Dr. Moretti emphasised that having a project which only works until everything works is a recipe for disaster. To heed his advice, we not only adhered to the specific product milestones, but created our own intermediate deadlines for minimum viable products in order to ensure that we always had a working model on our hands. Although this contributed to the stress over deadlines mentioned in previous sections, the particular decision to have our own customized product deadlines probably saved us more stress than it caused us. This is because quick product iterations allowed us to cut our losses as quick as possible when something didn't work in the way we expected it to and capitalize quickly on opportunities which we didn't know existed before.

For example, a quick iteration in the third week in which we set out to get all of our Analytics tabs complete resulted in us redefining our control panel to a more simple and elegant design which now is a key differentiator of our product. The reason we were able to make this choice better in the sprint than while we were planning over it spring break is that many ideas look good and cool until you have to implement them. When you're forced to sit down and code something up, you see what actual details it involves and can more accurately judge the business value it provides versus the labor cost. In the case of our Analytics tab above, we realized that the machine learning algorithms we had envisioned before were incredibly hard to construct, and actually detracted from the user experience of our average, casually curious user, so we updated our plan and refocused our app appropriately.

Version Control

Through the process of iteration and implementing new features, we used Github as our version control system. All of us had experience with Github before, so the technical aspects of using it were not a big problem, especially because there was also so much documentation online regarding how to use Github.

However, we learned the hard way on how to use Github properly in the project sense. This was because we faced a lot of merge conflicts early on in our project history. Because we all decided to work on the master branch, it was very difficult to pull and see the changes our teammates had made without losing our own work if we did not push. Moreover, these merge conflicts would sometimes take a lot of time to fix, which was a waste of time for us.

We solved these problems by doing two simple things at the latter half of the project: by pushing often and working in separate branches. The first solution, pushing often, enabled us to store the latest working code on Github frequently. This lowered the chances of merge conflicts as we were not adding or deleting large amounts of code each time. The second solution, working in separate branches, allowed us to work on our own feature implementation while keeping in touch with what the overall project was looking like. This prevented screwing up the entire master branch if someone made a mistake and kept our code manageable, and we did this in particular for the login feature.

Extensions

Throughout the course of this project we had to face many tradeoffs between different features and implementations because of the severe time constraints we were under. We had the ambitions to make a product that could turn into a real company, but it is a harsh reality that there simply is a limit to what can be done given only 7 weeks, 3 people, and exclusively open source software. This being said, we plan to continue work on SafeCity into the summer and beyond, to hopefully make some of our more grandiose, long-term visions a reality.

Chief among these routes for further work is the incorporation of more data. SafeCity is by nature a data driven application; our data is a first class product which we deliver to the users, so clearly to have more of it means to deliver more value to the user. Unfortunately, there are many challenges associated with expanding our data to a much larger set. The main one is the added cost of getting more server space so that we can store and load this extra data in a timely fashion, which would likely take the form of an upgraded small business package on Heroku.

To justify spending the $25/mo on the Standard Business package from Heroku and to move towards turning SafeCity into a viable business, we would ideally like to incorporate some small advertisement banners on the right side of our app, which is often idle anyway. Although this might sound like a bad thing to do because the motivation behind SafeCity is social betterment and not monetary profit, an app can still be built for selfless purposes and yet take on revenue. The advertisements could be targeted to users so that they are not annoying, but helpful, and the proceeds could go to further expanding SafeCity or perhaps directly to emergency responders.

Feedback

Pacing

Probably the second most difficult week of this term was not at the end of the project, but at the beginning, during the week after spring break (although the demo week surely takes the cake as the most difficult). The second week really felt like a hiccup for us because we had just spent spring break thinking through our product and familiarizing ourselves with some technology and were really excited to sink our teeth into our app, but then we did not get the chance to work on it because of a class assignment. And not just any assignment, by far the most time consuming and difficult assignment. This caused us to panic in the week that followed because starting things up and overcoming the basic barrier to entry of building an app always takes some extra time, and we started out slightly backlogged from where we wanted to be. We understand that the course needs to have five assignments, and having an assignment due the first week or midterms week is slightly cruel, but perhaps the course could be restructured to have the third assignment being the hardest in terms of time spent.

Another potential improvement with regards to pacing is to require students to start learning the technology they think they will be using for their idea at least a week or two weeks before the start of the project. We were only thinking about project ideas, when in reality we should have been also learning Meteor in depth. This way, we could have jumped right into coding our project rather than having to code our project and learn Meteor along the way, and then potentially pay the price later on. This might save students a lot of time down the road in terms of debugging or making silly mistakes due to a lack of knowledge about the technology.

Interaction

One interesting activity we would have enjoyed is having the opportunity to periodically show our product to students with the same advisor as us, and getting to see theirs as well. This would provide groups added motivation to make each iteration of a product as best as possible, because their peers will see it. It would also be interesting because it mirrors the environment in incubators, where startups all working under the same roof gain motivation and ideas from watching other presentations. Moreover, this facilitates talk between different groups that could help with debugging, opinions on design, etc.

Many students, ourselves included, didn't really know what other students were doing until the demos. Although the sixty second presentations in class helped a bit, it was hard to gain any inspiration from what people were doing. Perhaps a list of every group and its accompanying project could be posted and then groups working on similar tasks could seek each other out and learn from each other, or at least make sure they are not doing too much of the same thing. Then they could compare and contrast methodology and learn from each others mistakes.

Acknowledgements

To wrap up, we would like to say thank you to everyone who helped us throughout this course. Thank you to Chris for always being there to answer our late night Piazza posts and never being too busy to offer thoughtful feedback about our project. Thank you to Hansen for encouraging us to do the best we can do and having the patience to meticulously analyze every detail of the app versions we build for him. Finally, thank you to Ben, AJ, Mike, Vishan, Louis, Ojima, Soham, Ryan, and our other various friends, roommates, and fellow classmates for taking the time to play around with our product and offer candid advice on how to make it more user friendly and useful. This product could not have been accomplished without all your help.