RaceChipper
Ensure participant safety during long races
Gimball3000, Class of 2016
Danielle Neuberger, Randy Goodman, Anshul Kapoor, Tyler Schoen

Faculty Coach
Rick Weil
Project Sponsor
AJ Blythe

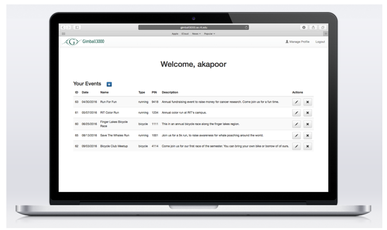
Trillium HEALTH
Your neighborhood health center
R·I·T
Software Engineering
Rochester Institute of Technology

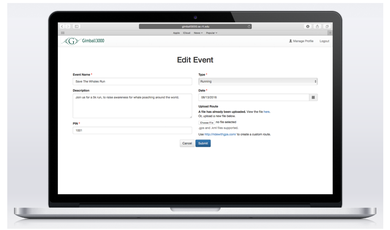
Background

RaceChipper is an application that assists in race event management. RaceChipper seamlessly integrates with Staff that conduct mass-racing events. The system enables racers to be tracked via Bluetooth beacons, and an iOS application that synchronizes data with a web-server. Using this application, race staff are capable of tracking individual racer location, status, and relative ETA, as well as communicate with other staff, all within the confines of a single app. Through this application, the iPad identifies specific bluetooth signals native to RaceChipper beacon devices. When a beacon is detected, the positional information of the iPad is relayed to a web-server, that serves that information to the rest of the iOS applications being used to conduct the race event. This way, the race staff will form an informational network that can track all competitors, for the purpose of preventing injury and attending to racers in need of aid. The application is designed around the possibility of intermittent network connectivity, storing and maintaining competitor information and synchronizing with the web server once connection is reestablished. RaceChipper also includes a web application that allows event organizers to create and configure events. RaceChipper is the future of competition management and event organization.

Technologies

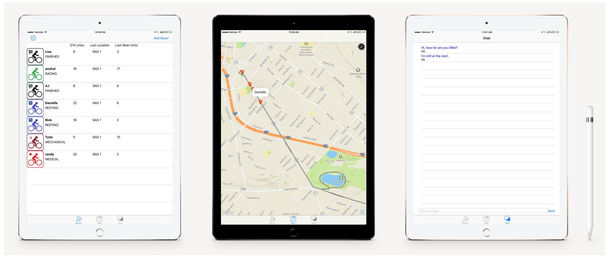
Key Features





Web App

Account Registration • Event Management • Register Bluetooth Devices



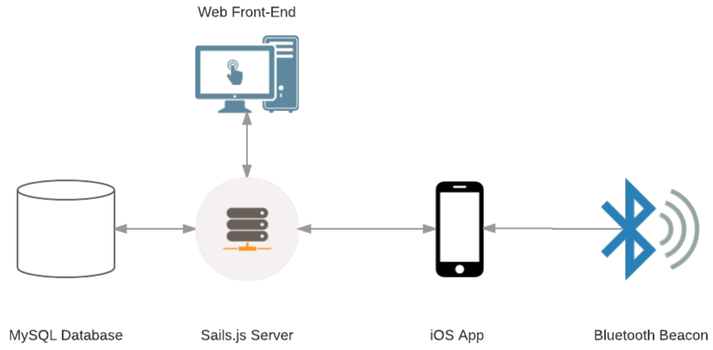
iOS App

Login/Logout • Racer Registration • Bluetooth Device Management

Map • Racer ETA Calculation • Chat • Check-in/ Check-out Racers

High-Level Architecture

The architecture of this project relies heavily on a Sails.js server that manages the interactions between the browser-based front-end, iOS application, and the persistent data stored in a MySQL database. The iOS application gathers positional data from independent Bluetooth beacons and relays the information to the database, as well as displays the aggregated information to users.



Process

In selecting a process model to follow, our team first looked at the qualities of our project and assessed what the model would need to provide. Upon initial assessment, we found our requirements were relatively stable enough, not to necessitate heavy requirements management. Most of the features and designs were well thought out and presented to us when we initially met with our sponsor.
On the other hand, the project involved interfacing with Bluetooth Devices, developing an iOS application, and displaying information on a map as the primary feature of the project, with which our team had very limited to no experience. In order to manage the high risks associated with these important project components, we needed a software development model that allowed for extensive risk management.

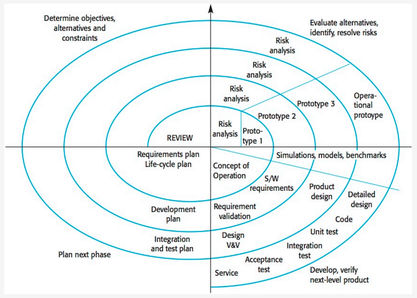


Figure 1: This figure shows the Spiral Model of software development. Of note are the iterative nature and prevalent prototyping.

Our team chose the Spiral software development model to follow during the course of our project. Although this model can be somewhat process and documentation heavy, it's iterative nature and focus on risks and prototyping strongly benefitted our team. Prototyping especially became essential for providing the customer visibility into progress, as well as allowing our team to become more familiar with the technologies and tools.

Metrics

Our team kept metrics on a variety of process and project elements during the project. Measurements taken on a weekly basis included information on bugs, lines of code, commits, tests, risks, and requirements.
From this, we derived our key metrics:

- Number of Bugs / SLOC (source lines of code) – *indication of quality*
- Test Statement Coverage (% program statements called during execution of test suite) – *indication of test suite maturity and coverage*
- Number of Commits / Week – *indication of development speed*
- Number of Risks Added / Week – *indication of risk stability; should decrease over time*
- Percent Relevant Risks (non-inactive status risks) – *should decrease over time as risks are mitigated and become inactive*
- Requirements Volatility – *how many requirements additions and changes are made weekly; should decrease over time*
- Developer Time Estimation Accuracy – *related to development speed; used to gauge and adjust amount of work developers take on*

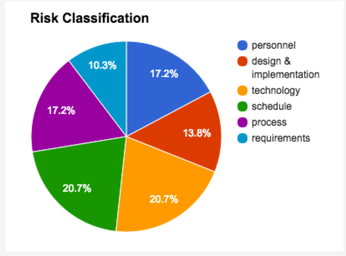


Figure 2: The chart shows how risks were distributed for the project. Our top two risk categories were technology (includes appropriate tech stack choices and device functioning) and schedule (includes appropriate scheduling and time management).

Figure 3: This graph shows how the percentage of relevant risks decrease over the course of the project as we progress through the phases of the spiral model and mitigate them.

Future Work

Our team has a variety of recommendations for future versions of RaceChipper. These can be broken down in the following two categories:

Web App

- Register Users: Registration of event participants in batches
- Export Race Data: Event owner can export all of the data for an event
- Add/Transfer Ownership of Event: Event owner can edit the ownership of the event to other registered users
- Help Page: An online guide of how to use the various features of the application

iOS App

- Offline Mode: A more efficient way to handle offline requests and persist data
- Improved ETA Calculation: Accurate ETA estimations of a racer to a given checkpoint. Incorporate route type, terrain type, individual speeds and other key factors
- Enhanced Usability: Provide a master landscape view (this includes having the map, chat and racer information all in the same view)