# Example Results

***CSM3 2012***

The goal of this project was to create a multi-platform desktop application that would analyse the user's music library, generate playlists for HIIT workout routines, and splice the songs in the playlist to produce an MP3 file that can be used on any compatible device. The playlist created would, as closely as possible, match the tempo (BPM) of the song to the target heart rate of each interval in the workout outline. Our application meets all of our client's functional requirements, as it successfully analyses music, creates custom workout routines, and splices songs into an MP3 file for a specific workout. Due to licensing issues however, the EchoNest APIs only allow 120 calls to its server per minute, resulting in a slow return in information. In addition, because of the dependency of EchoNest, the user must always be accessing the internet to run to program successfully. The implementation of audio cues, a song rating system, and a basic artificial intelligence that learns the user's preferences were all features that did not make it into the final product due to time constraints.

***Disney 2012***

This project has been tested and is working well in current version of both Google Chrome and Firefox. It will probably work in Internet Explorer, but may require increased coding to reach full functionality. It will not function in older browsers that do not support HTML5. The only remaining issue is that the scrubber may not always move smoothly after pausing and scrubbing back over a transition. This is a minor aesthetics detail, and does not affect the playback of the actual video. All functional and non-fuctional requirement have been met except for rendering the project into a single video file. This requirement was given a very low priority by the client, and could not be completed due to time constraints. However, the saved project contains all necessary data to reproduce a project, and should be easy to integrate with the client's existing rendering process.

Full Contact 2012

As an API, the system has no real UI, graphical or otherwise, and instead exposes its functionality programmatically, in the form of the structured Contact model and associated metrics with the extraction process. To facilitate quick developer feedback on how the system performs, the system is capable of generating detailed reports about the extraction process as compared to our corpus of human-extracted contacts from test emails. The test corpus is stored in JSON format, and the system can automatically be run against the corpus with the TestSuite tests. The generated reports are in HTML format, and contain the expected and actual contact information, as well as visual indication of how the signature extractor classified the parts of the email, and where data was extracted inside the actual signature.

The overall performance of the system in each of the extracted information categories is summarized in table 1 by the number of True Positive, True Negative, False Positive, and False Negative results in relation to the expected human-extracted contact information. These can

further be summarized by the Recall, Precision, and F1 score (the harmonic mean of recall and precision). Our approach sought to optimize precision over recall, in that the system would only emit contact information with a high certainty of existing at the expense of missing some contact information that could have been extracted to avoid incorrect data (false positives). Running the final system against the test corpus produced the following results:
[original document includes a table here]

While the performance of the system on the currently extracted fields is excellent with the test corpus, future work should extend the performance to both a much larger corpus of emails, as well as other relevant fields, including organizations (Company, Title), and social media links (Twitter, Facebook, LinkedIn). The modular nature of the system will facilitate the integration of these additional fields and test data.

***Newmont Tracker 2012***

Extensive testing was done for both the mobile and server side of the project. The mobile application was tested on a variety of devices (iPhones, iPod touches, and a few Android devices with different OS versions). Based on the results obtained, the application ran exceptionally well. An issue that should be addressed, however, is that in rare cases the mobile application can be very inaccurate and place the user several miles away from their true location. We believe this to be caused by the inaccuracy of the cell network. Unfortunately, this is a hardware and OS issue that we were unable to fix in software. Everything on the server side of the project works perfectly fine and there seems to be no issues at all. The website is fully functional and tested on Safari, Firefox, Chrome, and Internet Explorer. The site provides all the features requested by the client and more. It is  also intuitive and easy to use.

***WiiTools 2010***

**Lessons Learned**

- Threads, while easy to implement, are very difficult to implement right. Our original strategy of combining the parsers involved threads, and we ran into many issues with them not starting or stopping properly. Using processes instead is a much more reliable and simple way to allow for multiple functions to be executed at once.
- Sockets are very useful when trying to decouple a program. Especially once you remove the overhead of connecting and disconnecting, they provide an efficient, solid interface between parts of a program. They are also useful in their ability to abstract data, and allow for easy logging and use of those logs. This is because the process listening to the socket does not care how the data was created, it just reads it and performs the necessary functions.
- Python is a very powerful programming language, yet is very easy to use. Python also has a very extensive list of modules. If you need to do a specific task, there is a good chance of an existing Python module to do the work for you.
- It is very helpful to be the administrator of your development environment. We ran into many problems in the Alamode lab trying to get Bluetooth to work, among other things. In the end, the primary testing machine that interfaced with Bluetooth turned out to be a personal laptop that we could fully access and manipulate.

- Agile methods actually work. For developing the Wii Event Parser, it was very helpful to start by reading basic data, then take it up to reading data from a Wiimote, then managing several Wiimotes, then abstracting the parser and the Wiimote manager, until the module was finished. The “war room” method of having the whole team coding in the same room was very helpful for solving problems and working out design issues.