This presentation was given at the MCN 2015 conference in Minneapolis.
For the 100th birthday of Balboa Park, the San Diego Model Railroad Museum opened an outdoor Centennial Railway Garden. Tiny, period-appropriate, model trolley cars loop slowly around meticulously-constructed scale buildings from the park in 1915... and it’s really worth visiting!

SDMRM contracted the Balboa Park Online Collaborative to add a technology component—to design and build iPad quizzes which would allow visitors to turn on lights, play sounds, and activate a fountain on the model.
To do this, we set up three Raspberry Pi computers, each running a node.js server, which presented the quiz as a fullscreen web page on the iPads, listened via websockets for a response, and then played .wav files through the headphone jack and triggered GPIO pins to switch on and off relays controlling lights and a water fountain. The three Raspberries Pi and the iPads were set up on a private subnetwork of Balboa Park’s high-speed WiFi.

We finished the programming and testing a month ahead of schedule, and handed it off to the museum’s contracted electronics guru for wiring, and were amazed that everything had gone so smoothly. (So naturally, you know what happens next.)
Three days before the garden opening, EVERYTHING was breaking.
The Raspberries Pi were mysteriously dropping off the network when left unattended overnight.

The Node server was sometimes not playing sounds, ...and sometimes crashing completely.

The relays—physical switches to turn on lights and the fountain—were not always triggering. Was this a software or hardware problem?

On top of all of this, the iPads seemed to be running extremely slowly sometimes, and the web pages would only partially load.
Obviously: not the kinds of things that you want happening right before an installation opening.

For this case study, we’ll link together ten lessons in fixing things and getting back on track. They may seem general, possibly even common sense, but they’re not always what you think of when you’re diving into a project, especially if you’re new to Raspberry Pi.

Since we won’t have time in ten minutes to get into the lovely esoterica of Raspbian administration over ssh, I’ll have a link to more resources at the end, as well.
First: While we had tested for hours in the office without a hitch, we had not tested for *hours*, and we’d also tested *...in the office.*

We needed to test for **longer** to see—let alone get enough data to troubleshoot—some of the the server crashes.

We needed to test **on-site**, away from the steady perfect WiFi of home and office, to find out that the wireless connection would drop in the first place.

Learn how to log your errors (by piping a process into a .log file on invocation), so that you can get more information about what’s going on when you leave it running overnight.
Oof. The problem with the wireless connection was, well, *multiple problems.*

**Exhibit A:** Raspberry Pi USB WiFi dongles are *small* antennas. We could have purchased larger ones, but there didn’t seem to be a need. The Park’s WiFi signal was a full four bars in the installation location.

**Exhibit B:** There’s a flag in the Pi network configuration that tells the WiFi dongle that it’s ok to go into power-saving sleep. While this should be turned off by default, it’s a potential gotcha.

**Exhibit C:** The Garden itself was behind two large buildings on the edge of the Park’s WiFi network, on the edge of an access point. Best we can tell, the Pi would sometimes attempt switch access points, when it received interference from other wires, objects, or people in the way. It wouldn’t reconnect until you manually restarted the wireless or rebooted the Pi.

**Exhibit D:** The iPads were silently connecting to the wrong subnetwork and not seeing the Raspberries Pi. This was making an iPad either stall out or load a sluggish, non-functioning cached version of the site.
Our solution was to use ethernet cable to hardwire all three Raspberries Pi to a spare wireless router on a private, non-internet network. Simpler system, less competition. We also locked down the iPads to that network, and set all devices, Pi and iPad, to have static IP addresses.
It's hard to test these conditions, though, when you don’t have a Raspberry Pi on hand. Since they're physical devices, only so many people (uh, one) can be working on them at the same time.

When we finished up our software work, we needed to hand off the configured Pi computers to be put into a permanent enclosure and wired to the electronics in the model.

If we had purchased a spare Raspberry Pi at the time, we could have continued testing it, on-site, while the production units were being wired.
One thing we did do well was to make the configuration of each Pi identical. Regardless of their role, each Raspberry Pi had all of the sound files and scripts on it. This way, we could configure one, and then save a copy of the SD Card disk image to copy to the others. If one of them died, we could copy this disk image to a new SD card, and have a spare Pi up and running within an hour.

(*In future projects, we discovered that git version control was useful here. When the source code is the same on all the Raspberries Pi, it’s an easy git push / git pull.*)
With the disk image of each Pi identical, how were we able to run different code on each Pi?

We set up the Node server script so that it allowed us to pass parameters to it when starting up the server. These arguments on the command line could tell it which model it was to control (and which sounds to play) as well as how long to run the fountain or turn on lights.
Speaking of the node server, why was it sometimes not playing sounds? Why was it crashing?

Node lets you add functionality by importing node modules using the Node Package Manager, or NPM. There are at least three available modules to play sounds, which call native, system-level soundplaying code.

According to our logs, the module we were using was failing, and halting the node server. We ended up having to take a whole day to test all of the options, and in the end, rebuild one of them from source code on GitHub, to get it to work.

As common sense as it sounds, when you are planning the project, imagine all of the ways that it can fail and start thinking of (and researching) contingency plans.
There are several different ways to do everything, whether it be picking between multiple node modules for playing sounds... or simply launching a script at startup—crontab, rc.local, and so on.

When you're troubleshooting, sometimes the problem is even knowing the correct name for your problem, which means you'll need to search the internet just to find out the terms you need to search the internet to find an answer.

Learning your way around a command line and learning the basics of Linux (and Debian in particular) can help.
After solving all of these other problems, sometimes the fountain would still fail to start. Having looked at everything we could, the software appeared to be working, and we were even hearing the audible *click* of the relay when it switched on.

Thankfully, we were able to call in the project’s electronics guru, who patiently explored everything with his multimeter to suss out what might be wrong.

In opening and closing the enclosure for the Pi and electronics (many many times), we had loosened one of the wires! A few literal turns of the screw and everything was working again.
As you know from exhibit design, you need to think of maintenance, especially in those crucial days in between the install and the opening.

Make sure that you can easily access the Pi and any other electronics, that you can easily connect ethernet and USB, that the pins are not obstructed if you need them, and that it’s not impossible to eject the SD card in case you need to replace it. Also, make sure that you have access to the room or area where everything is being stored. (I have funny stories about this, ...ask me afterward.)
Finally, make a point of documenting everything you do and learn. We found it really useful to do this in a shared, living, photo-illustrated Google Doc that multiple people could read and contribute to.

This takeaway is a no-brainer. You'll forget everything on this project when you start your next one; documentation lets you clear your head guilt-free, and also give back to the community with what you learned.
So:
Test long term and on-site.
Go wired over wireless.
Always purchase spares.
Make it uniform.
Write code that takes parameters.
Test multiple options & always have a Plan B.
Learn the search terms.
Work closely with other experts.
Design for easy access.
Write good documentation.
Thank you!

We didn’t have nearly enough time to get into the nitty gritty of when to choose a Raspberry Pi over an Arduino, how to set up a Raspberry Pi, what services you should install, how to set up a startup script, and many other things, so I’ve put them into a cheatsheet at the link above. Download and let me know if you have questions.

Q & A time!