Racket in Production

Brian Mastenbrook

CTO Wearable Inc.

_ in Production

Production of _

- Portable wireless flash drives
- A little WebDAV NAS in your pocket
- Useful for lots of things

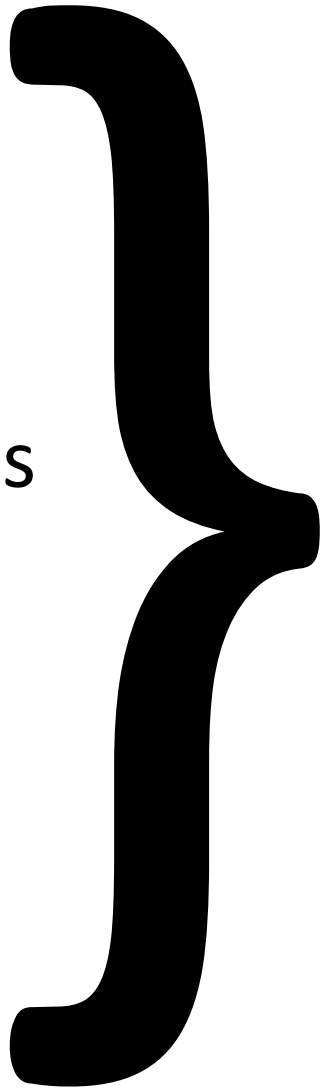
How to Make Electronics

- Send a contract manufacturer lots of parts
- ???
- Profit!

- They put the pieces together
- You tell them how to make it
- … and how to test it

- Plug it in
- Flip the switch
- If the red light comes on, it’s good

- Plug it in
- Make sure it mounts
- Copy all the preloaded files
- Eject the drive
- Turn it on
- Connect from a tablet
- Play a video file
- Be sure to turn it off



Lots of manual effort!

The Fraternity of σσσσσσ

σσσσσσ creed

- Design your products to reduce the complexity of manufacturing
- Test everything you can before it leaves the factory
- Up front investment in quality can *save you money*

Two Perspectives on Quality

- Tests passing
- Error message
- Segfault

- Defect rate
- Product return
- Recall

- Program
- Test
- Control

- USB connectivity
- CPU functionality
- SD card interface
- WiFi functionality
- Battery power

Why Racket?

- Flexible, expressible language
- Safety
- Built in cross-platform GUI
- Database library
- Low level networking (UDP)

Test Fixture 1.0

- Embedded fanless PC
- Connects to five devices
- WiFi tested in RF isolation box
- Sensor for RF box lid
- Controls USB power via relays

Lessons learned

- USB hubs suck!
- Modules and contracts are great!
- One computer won’t scale
- Need a distributed system?

Test Fixture 2.0

- A distributed system of BeagleBones!
- Each BeagleBone controls one device under test
- Controller BeagleBone handles shared state

Test Fixture 2.0

Well, that didn't go so well...

Lessons learned

- BeagleBones suck!
- Distributed systems are great!
- They recover automatically when things crash
- … after some amount of time

Test Fixture 3.0

- Back to the PC
- Server-class stuff
- Sixteen USB controllers
- WiFi + Bluetooth

Invariants over version

- Lots of things going on asynchronously
- Robust to unexpected conditions
- Needs to always work

Actors and Messages

- Actors are Racket threads
- … which exchange messages
- … and encapsulate state
- Controller
- Programmer
- GUI
- External interaction

Untyped to typed

1.0: Untyped with contracts

2.0: Limited use of typed, removed before deployment

3.0: Modules and messages are typed, states are dynamic

Typed Messages

```
(define-message ping ([x : Number]))
(define-message pong ())

(define-message set-target
   ([t : Thread]))
```

State Machines

```
(define-state-machine ping-ponger
  ([state : Number 0]
   [target : (Option Thread) #f])
  #:initial start
  #:problem start
...)
```

Transitions

```
(define-state start
   [#:timeout #f]
   [#:enter]
   (define-transition set-target
     (printf "got a target: ~a\n" t)
     (set-ping-ponger-target! context t)
     (goto-state 'send-ping)))
```

Demo

Lessons Learned

- Modules and contracts are great
- Types are even better
- Macros are great
- Find the right encoding for the problem domain