# The Functional Edge

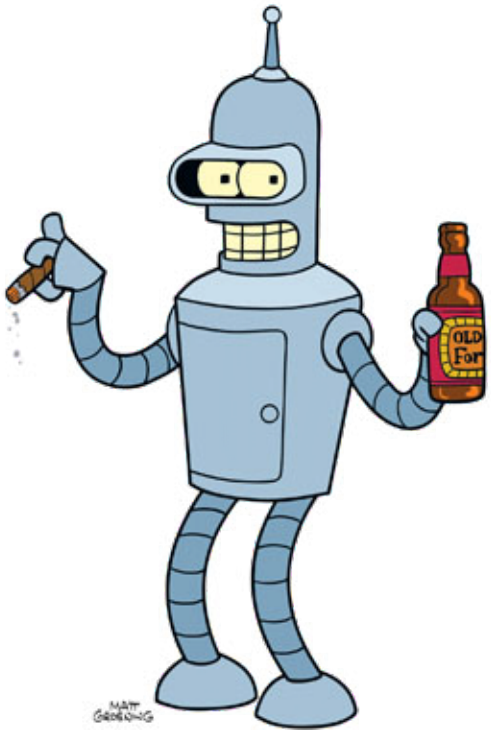
## Robotic Testing with Bowler Studio and Clojure

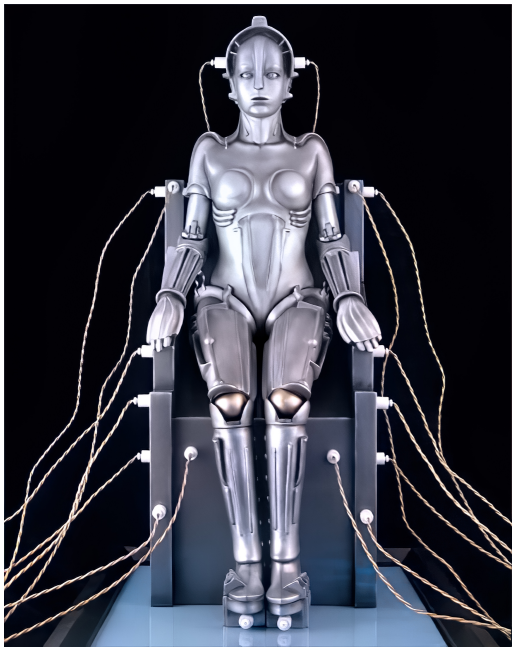
Thomas Gebert
Software Engineer

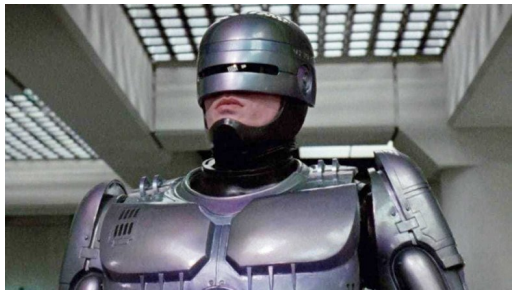
# Who Am I?

- Software Engineer in New York.
- PhD student.
- Math Enthusiast.
- Cartoon Fanatic.

# Robots!







- Robots are objectively cool.
- Part of their coolness is due to how hard they are to build.
- They are also expensive!

# Robots!

## Why are they expensive?

- Unlike most tech projects, they require physical matter and thus require physical prototypes.

## Prototypes requirements:

- Planning.
- Ordering/outsourcing of external parts.
- Assembly
- Dealing with breakages or mismeasurements.

# Converting Robots Into Software.

## Advantages of Software

- Software and computers are extremely cheap.
- Software is malleable.
- Software is easy to back up/version.
- Software can be easily written by anyone with access to hands, a computer, and an internet connection.

# Digital Twins.

- A digital representation of a real physical product.
- Allows for simulation of a cyberphysical systems *before* we build anything.
- Helps avoid wasted time and resources.
- Simulation will inform the real-life design, and the real-life design can feed back into the simulation.

# CAD Modeling.

- Lies somewhere between “technical” and “art”.
- Most tools are graphical.
  - AutoCAD
  - CorelCAD
  - FreeCAD
  - many, many others.
- Covers an entire spectrum across the entirety of things you might want to design/model.

# Annoyances.

- GUI programs are much harder to automate.
- Since it involves “a human touch” the results are not reproducible.
- You can’t use Vim keystrokes for everything.

# What do we do?!

- We like plain text.
- We are functional programmers.
- We know better than the unenlightened.

# Constructive Solid Geometry.

- An approach to 3D modeling and CAD.

## Concepts.

- We have primitive shapes (cubes, spheres, cylinders, etc).
- New shapes can be created by applying boolean operators to existings shapes. (Union, Intersections, Differences)
- Can be expressed in plain text due to its simple and declarative design.
- If only there were some kind of software paradigm that emphasized declarative programming...

# OpenSCAD.

- Open Source CAD software for all the major platforms
- A purely functional programming language.
- A declarative and compositional way to create 3D models.
- Models are defined purely in plain text.
- Objects are composed of primitive shapes using unions, differences, hulls, etc.
- Allows for use of variables, coding, and mathematical operations.

# OpenSCAD.

## Shapes.

```
cube(5);
cube([5, 5, 5]);

cylinder(h=10, r=3, center=true);
cylinder(h=10, d=6, center=true);

sphere(r=10);
```

# OpenSCAD.

## Movement.

```
translate([1, 2, 3]){
    sphere(r=10);
}

rotate([90,0, 90]){
    cylinder(h=20, r=5);
}
```

# OpenSCAD.

## Boolean operations.

```
difference() {
    sphere(10);
    cylinder(h=100, r=5, center=true);
}

intersection() {
    sphere(10);
    cylinder(h=100, r=5, center=true);
}

union() {
    sphere(10);
    cylinder(h=100, r=5, center=true);
}
```

# OpenSCAD.

```
NUM_H = 10;
for ( i = [1:1:NUM_H]){
    rotate([0, 0, i * (360 / NUM_H)]) {
        translate([30, 0, 0]) {
            cylinder( r=5, h=30, center=true);
        }
    }
}
```

## Modularity.

```
module bigH(distance, r, h) {
    union() {
        translate([distance/2, 0, 0]) {
            cylinder(r=r, h=h, center=true);
        }
        rotate([0,90,0]) {
            cylinder(h=distance, r=r, center=true);
        }
        translate([distance/-2, 0, 0]) {
            cylinder(r=r, h=h, center=true);
        }
    }
}
```

# OpenSCAD.

## Everything composes!

```
NUM_GROUPS = 4;
NUM_H = 10;
for (j = [1:1:NUM_GROUPS]){
  rotate([0,0,j * (360/NUM_GROUPS)]) {
    translate([100, 0, 0]){
      rotate([0,90,0]) {
        for ( i = [1:1:NUM_H]){
          rotate([0, 0, i * (360 / NUM_H)]) {
            translate([30, 0, 0]) {
              bigH(distance=12, r=5, h=30);
            }
          }}}}}
```

# OpenSCAD demo.

- Let's take a look.

# 3D Printing.

- Also lies somewhere between “technical” and “art”.
- Allows average people to have some level of manufacturing at home or a lab.
- Decreases turnaround time compared to outsourcing to a shop or to a manufacturing plant.

# Issues with 3D printing.

- Printers break.
- They break again after you fix them.
- Your printed models can break mid-print if not designed well.
- Prints are slow, even on relatively fast printers.
- Resin printers in particular make a big mess and deal with toxic chemicals.
- You might spend a week making a print work, just to find out that your measurements are off by two millimeters.

# Issues with 3D printing.

Because the experience is awful, we want to minimize how many prints we have to make.

# CoppeliaSim.

- Freemium/OSS robotics simulator.
- Allows for importing 3D models and running simulations them.
- Programmable via Lua.

# CoppeliaSim Demo.

- Let's look at a quick example.

# CoppeliaSim Disadvantages

- Lua isn't a functional language.
- Not a straightforward path between running the model and then exporting straight to a 3D printer.
- Doesn't really let you edit the model directly within the program.

# BowlerStudio.

- Open Source CAD and Digital Twin simulation toolkit.
- Allows the use of Java, Kotlin, Groovy, Python, and Clojure for both CAD and simulations.
- Can simulate nearly every aspect of a robot
  - Motors
  - Physics
  - Collisions
- Can plug into and utilize microcontrollers within the simulations.

# Bowler Studio JavaCAD.

- Similar to OpenSCAD.
- Uses simple Java functions to build complex shapes.
- We do not like Java, so lets make a DSL.

# Clojure JavaCAD DSL.

## Shapes.

- Cubes/Rectangular Prisms

```
(defn cube [x y z]
    (.toCSG
        (Cube. (double x) (double y) (double z))))
(defn sphere [radius]
    (.toCSG
        (Sphere. (double radius)  200 200)))
(defn cylinder [radius height]
    (.toCSG
        (Cylinder.
            (double radius) (double height) 200)))
```

# Clojure JavaCAD DSL.

## Translations.

```
(defn move-x
  [obj n]
  (.transformed
    obj
    (.translateX (Transform/unity) n)))
(defn move-y
  [obj n]
  (.transformed
    obj
    (.translateY (Transform/unity) n)))
(defn move-z
  [obj n]
  (.transformed
    obj
    (.translateZ (Transform/unity) n)))
```

# Clojure JavaCAD DSL.

## Translations.

```
(defn move [obj x y z]
    (-> obj
        (move-y y)
        (move-z z)
        (move-x x)))
```

# Clojure JavaCAD DSL.

## Rotation

```
(defn rot-x [obj x-degrees]
    (.rotx obj x-degrees))

(defn rot-y [obj y-degrees]
    (.roty obj y-degrees))

(defn rot-z [obj y-degrees]
    (.rotz obj y-degrees))
```

# Clojure JavaCAD DSL.

## Rotation

```
(defn rotate [obj x-degrees y-degrees z-degrees]
    (-> obj
        (rot-x x-degrees)
        (rot-y y-degrees)
        (rot-z z-degrees)))
```

# Clojure JavaCAD DSL.

## Boolean operations.

```
(defn union-all [& objs]
    (reduce
      (fn [a b] (.union a b)) objs))

(defn difference-all [& objs]
    (reduce
      (fn [a b] (.difference a b)) objs))

(defn intersect-all [& objs]
    (reduce
      (fn [a b] (.intersect a b)) objs))
```

# Clojure JavaCAD DSL.

## Loops

```
(def NUM_CYLINDERS 10)
(->> (range NUM_CYLINDERS)
     (mapv (fn [i]
          (-> (cylinder 10 20)
              (move-x 40)
              (rotate-z (* i (/ 360 NUM_CYLINDERS)))))))
```

# Clojure JavaCAD DSL.

## Modularity

```
(defn big-H [distance r h]
    (union-all
        (move-x (cylinder r h) (* 0.5 distance))
        (move-x (cylinder r h) (* -0.5 distance))
        (rot-y (cylinder r distance) 90)))
```

# Demo

Let's see it in action.

# Conclusion

- Build some robots!
- Let the computer automate as much as possible.
- Remember the cool guy that taught you robotic simulations and code-based modeling when you're taking over the world.

# Contact me!

- thomas@gebert.app