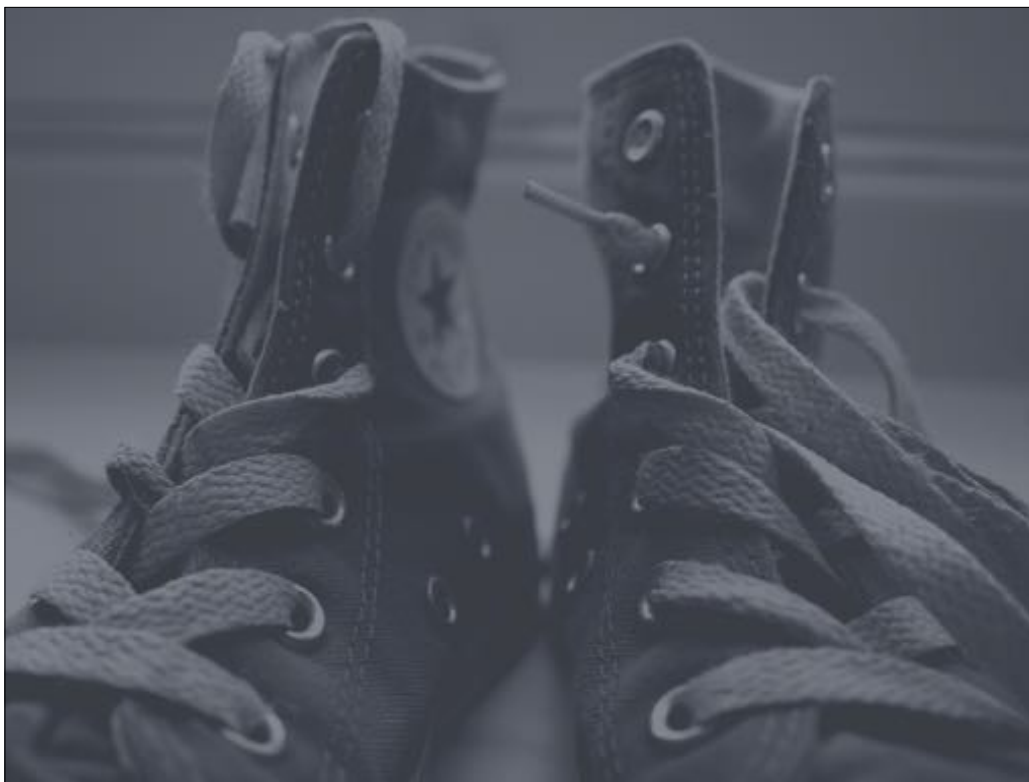




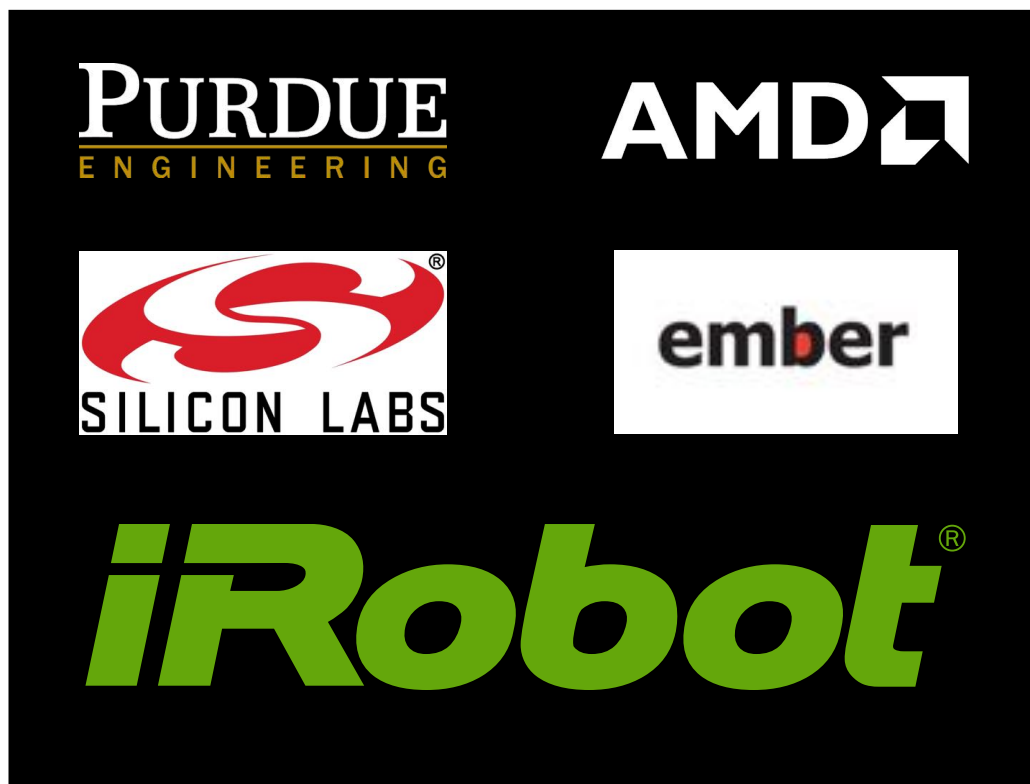
Empathy Driven Development

Chris Svec
Senior Principal Software Engineer, iRobot
[@christophersvec](https://twitter.com/christophersvec)

This is the “Empathy Driven Development” talk as delivered in May, 2015 at the Embedded Systems Conference in Boston, MA. The speaker notes are a rough transcription of what I said during the talk.



Who has ever started a new job? Okay, now when you started your most recent new job, who had everything they needed to learn their new software/hardware/product and contribute immediately? Who found it painful/tricky/difficult to learn what they needed to become productive? Think about that as we talk.



Let me introduce myself.

Bachelors and Masters in Electrical & Computer engineering from Purdue (Go Boilers!).

I designed chips at AMD, then moved up the hardware/software stack to Silicon Labs and Ember where I continued working on chips by writing internal firmware. And now I'm an embedded software engineer at iRobot where I do embedded software for our Roomba vacuum cleaner robots.



This talk came out of a set of best practices the software organization created, it was a 5+ page doc - even though I was an author of it, it bored me to tears. I wanted to see if I could find a single fundamental idea from which we could derive the many software best practices we had. Is there a single concept that will help us develop better software and hardware? Yes: EMPATHY. I think empathy motivates most, if not all, best software practices. Empathy is widely used in design and web dev fields for external customers & users. I want to bring empathy in-house.



Rather than just tell you my conclusion (“empathy”) I’d like to take you through my train of thought as I considered better software practices. I hope that by showing you how I thought about it you’ll be to think of something I haven’t thought of.



Before we dive into how we can better develop software, I want to first talk about something that gets in the way of many of us who do engineering for a living.

Impostor Syndrome

It's called Impostor Syndrome. Who's heard of it?



Idea: You don't deserve the success you've achieved.

Anything good that you've done was a result of luck, timing, or fooling others into thinking they are more intelligent and competent than they believe themselves to be.



Or this if you want something less threatening.

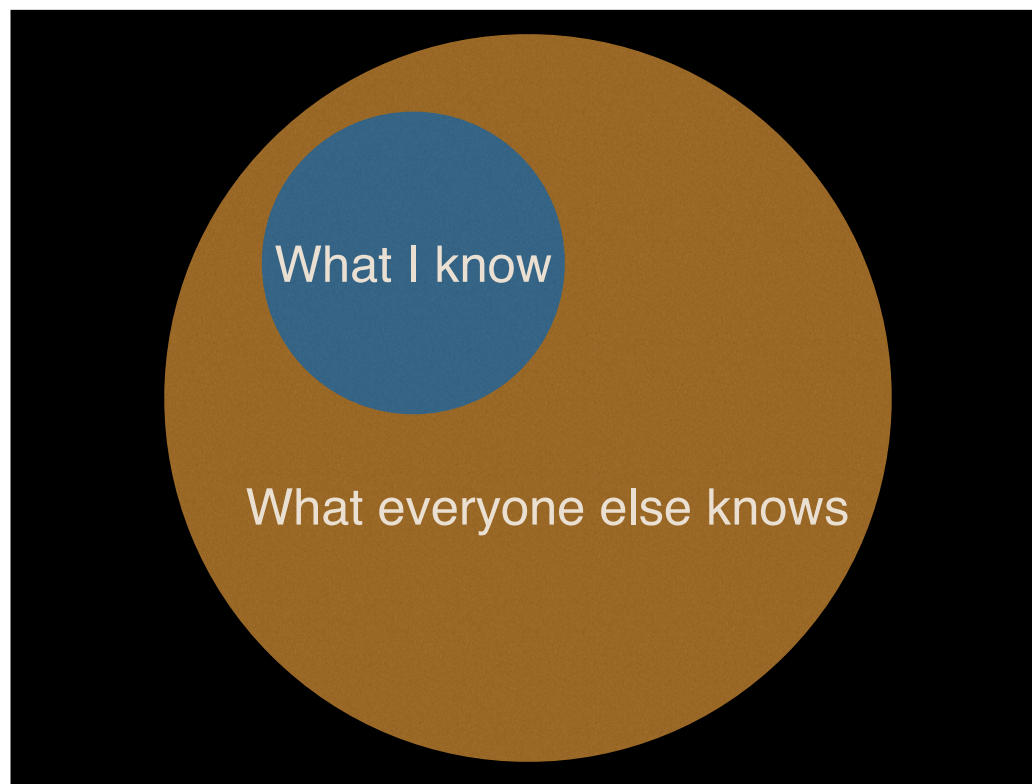
I sometimes have Impostor Syndrome.

Anyone else?

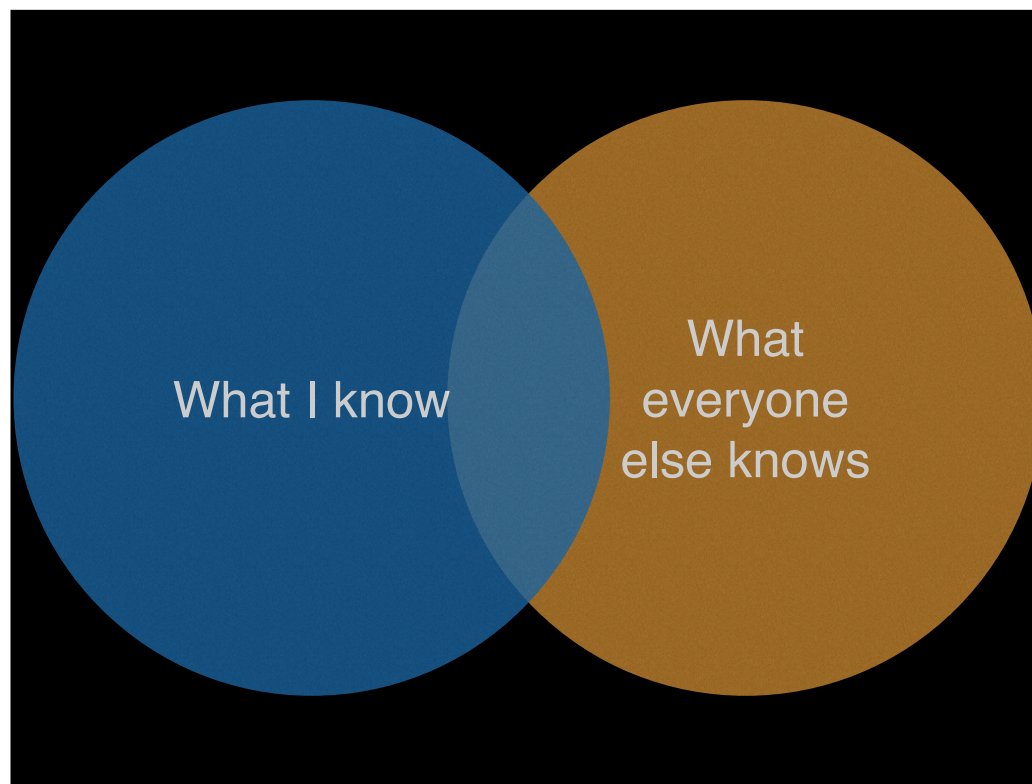
Usually 1/3 - 2/3 of a room raises their hands. Junior and (very!) senior people alike.

Realizing that many people feel the same as you goes a long way to neutralize Impostor Syndrome.

Another way to neutralize it is to expose the lie it's based on.



You might buy into this lie too: you think you know a little bit, in the blue circle, and everyone else knows everything you know plus a lot more. The truth is more like this...



You know some things, and other people know some things. There is some overlap in what we each know, but we've each got tons of unique experience/assumptions/approaches, no matter how little or much experience we have.

Impostor Syndrome is a big topic. I just wanted to mention it today; maybe that will start other conversations about it later.



Let's start at the end.



The following quote is the conclusion of my talk.

Richard Hamming, inventor of Hamming codes and Hamming distance, founder of the ACM, Turing Award winner, who worked with computers longer than most of us have been alive, and he says:

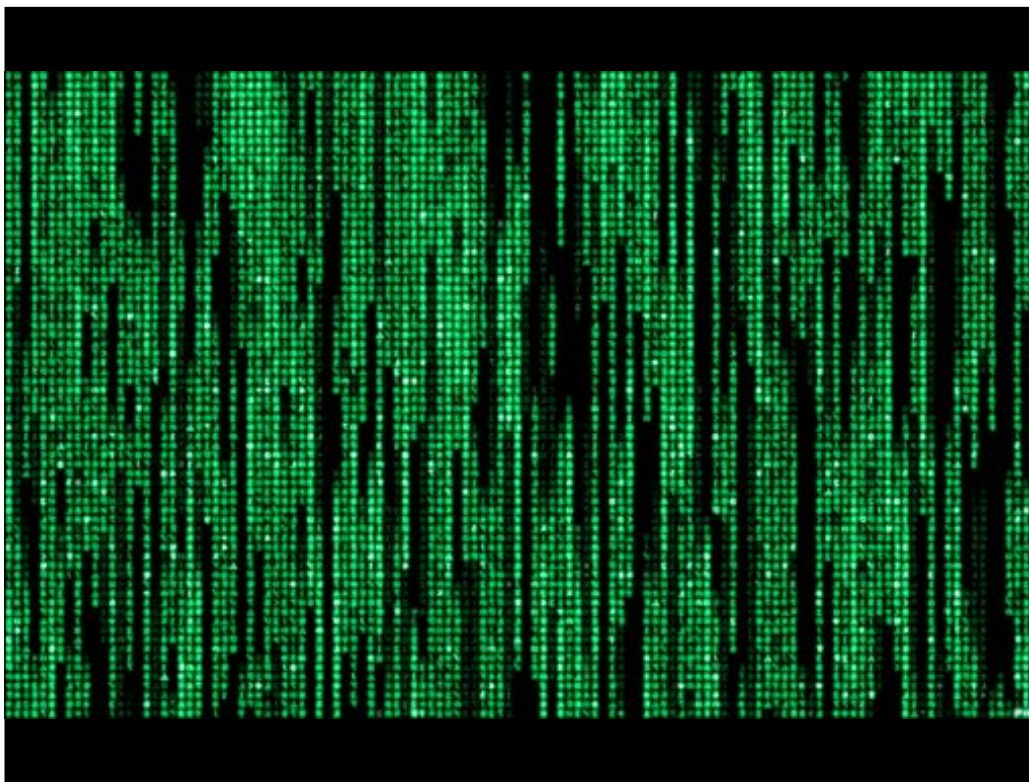
“The more I study programming, the more I understand that I’m looking at human beings.

Remember: write psychological rather than logical. Write so that it can be followed.

So that you, five years later will know what you were doing.”

- Richard Hamming

When he says “write psychological rather than logical” he means to write FOR PEOPLE rather than for the compiler. That’s the conclusion of my talk. Let me tell you how I got there.



Back to our day jobs: engineering.

We want to do better engineering, and so we usually talk about how we can get better at technology. Tech is relatively easy to discuss and teach. Hard core tech is interesting to us; it probably brought us all here in the first place.

Unfortunately though, tech products that merely do what they say they do on the box is not enough to be successful. It's table stakes. It's not a competitive advantage anymore because for the most part, technology works. Don't get me wrong, getting it right is hard, but at the end of the day we know to create functional products.

Necessary
but not
sufficient

Functional technology is absolutely necessary for success, but not sufficient. If func tech isn't enough, than what is? I assert that our competitive advantage is not technology, but instead each other and how we work together. We earn our paychecks and make great tech products by working together better - not by typing faster.



Our industry has historically celebrated the Lone Hacker - a lone trenchcoat-wearing hacker who can hold an entire design in her head sitting in a dark room typing furiously.

(name that movie, anyone?)

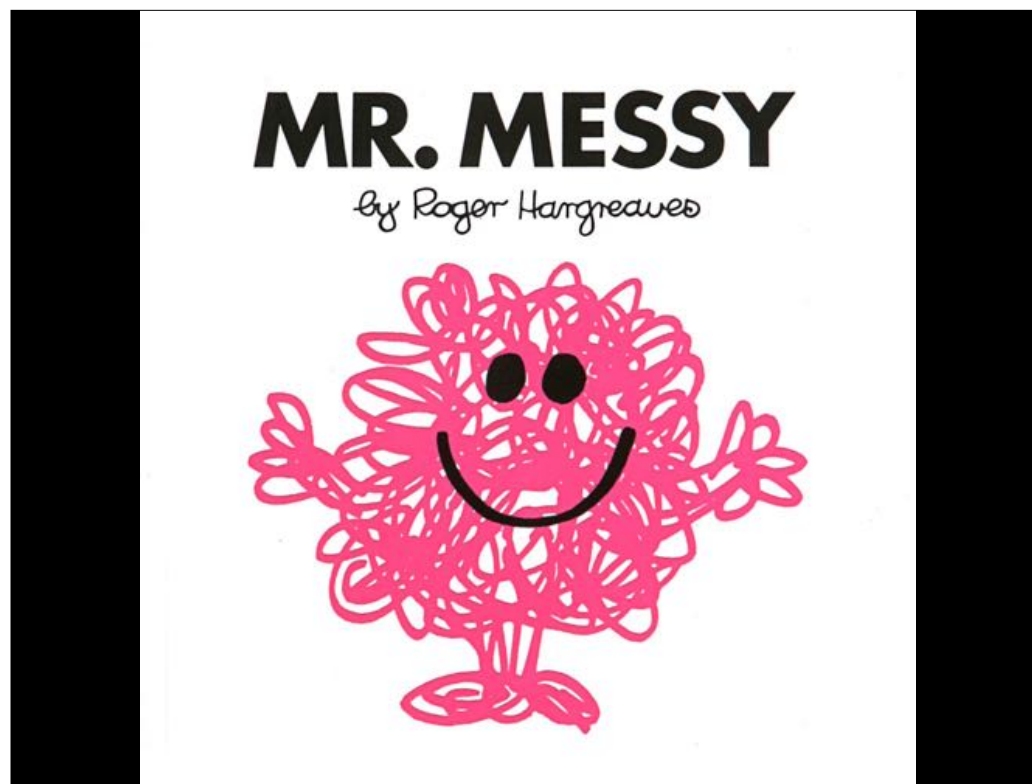
The Lone Hacker is NOT sufficient for creating complicated products any more.



Instead of the Lone Hacker, what we need are teams of several different people working together, each with their own different strengths, weaknesses, viewpoints, assumptions, haircuts and jewelry.



Or this if you want something less aggressive.
The problem with teams is that people are messy.



And as we add more people we get messier. Metcalfe's law says that we get messier with order N squared, which is a terrible scaling factor. We don't communicate effectively, we've all got different baggage and assumptions and motives and fears and ... yuck.

It's definitely easier to talk about technology than people and all of their baggage, so why would we bother talking about people?

Why bother?

Why bother with people?

I assert that if we focus on **engineers** before **engineering**,
the **team** before the **tech**,

Our products will be
more **understandable**,
more **maintainable**,
more **extensible**,
and more **enjoyable** to work with.



I intentionally chose the word “enjoyable” there. We optimize for code size, manufacturability, metal layers, task length, market risk, financial cost - we should optimize for engineer happiness as well! I want to enjoy working on my software.



This is my family. Every minute I spend away from them had better be worth it. I spend too many hours a day here to not enjoy what I'm working on. I don't want to dread having to poke around dark, scary corners of the code. I want to enjoy my time at work.



If you buy my argument so far, then we to figure out better ways to work together to create tech products. How do we even start thinking about the best ways to work together?



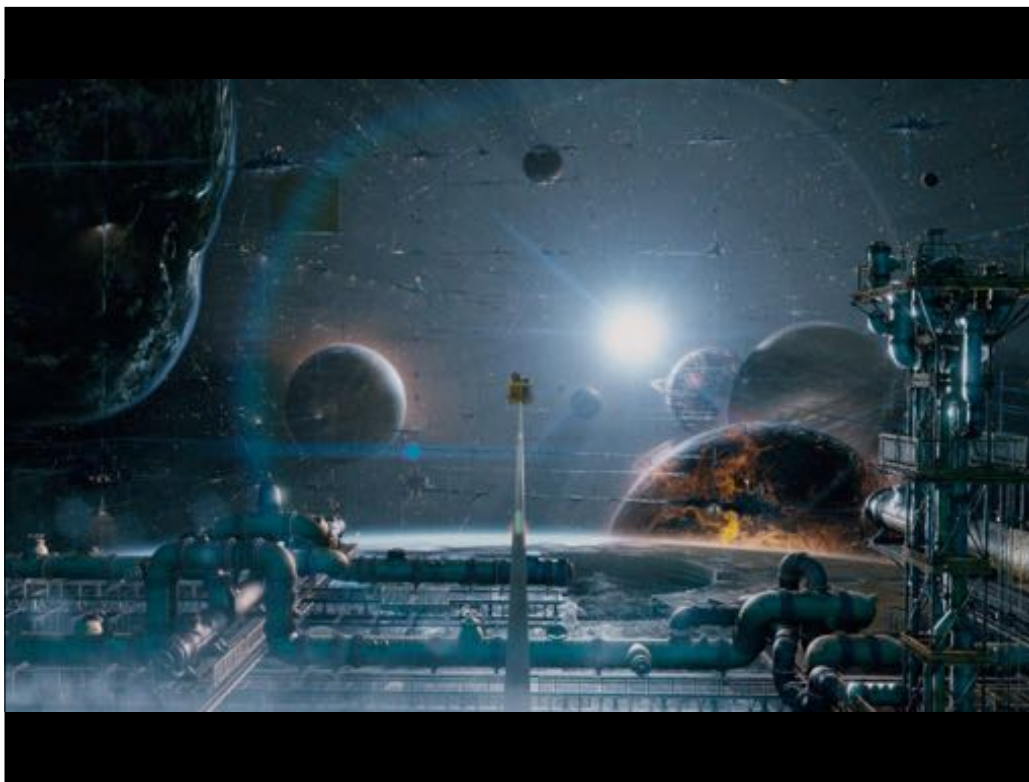
This is an important question - I knew I had to hire the best consultants money can buy and I hired these guys. They told me to take a **big step back** and consider what we do as engineers: what would you say you do here? Basically we get paid to think about fiction. Follow me as I walk you through my train of thought.

Fiction?

Stay with me here. We deal with fictions: abstractions, thoughts, concepts, metaphors, relationships, bits - even the sacred '0' and '1' binary values on which we base EVERYTHING we do is a fiction that our Electrical Engineering colleagues created for us.

Fiction!

We don't deal with physical stuff, we don't deal with atoms - we deal with IDEAS.



We create new systems and frameworks and worlds - we even create many of the constraints that we're later bound by.

Electrical and mechanical engineers do deal with physics and the real world, but there is still an incredible amount of creativity, flexibility, and imagination in how EE's + ME's deal with the Real World.

So I'm grouping all of us engineers in the room together when I say that our medium is Ideas. Thoughts.

Abstractions. Worlds and their constraints.

Which is interesting, because that makes us sound more like fiction authors, writers, than engineers.



Fiction authors like Tolkien and J.K. Rowling create worlds, languages, maps, classes, abstractions, hierarchies, ecosystems, platforms, and characters.

That kinda sounds like what we do.

So what else do authors do?



An author's job is to put themselves into the shoes of their characters and their readers - to understand them completely, where they're coming from, their world views, expectations, reactions - everything - even if they personally disagree with them.

For instance, J.K. Rowling, the author of the Harry Potter books, needs to wear two pairs of shoes:



She needs to put herself in Harry Potter's shoes. What would the young orphan Harry do? Would he really fall in love with Hermione? Would he really hate Snape? Would he identify with Voldemort even a little?



Rowling also needs to put herself in the reader's shoes - your shoes - and understand your point of view as the reader: how much explanation does a reader need? How much foreshadowing is enough? Do you think Harry's actions and thoughts are believable? How can she draw you in and get you to buy in to her fictional creations? And she can't just understand one type of reader; she needs to understand the viewpoint of women and men and old and young.



Putting yourself in someone else's shoes is called empathy, and I think we should take a page from the author's playbook and use empathy in engineering.

empathy:
the ability to
understand and share
the thoughts and
feelings of others

Empathy is defined as: “the ability to understand and share the thoughts and feelings of others” or “the ability to put yourself in someone else’s shoes.”

It’s deeper and more fundamental than mere sympathy; you feel empathy with someone when you’ve “been there” and experienced something they’re experiencing, and sympathy when you haven’t.



This isn't some hippy-dippy touchy-feely new-age feel-goodery; empathy is part of basic communication. If you don't have empathy, you can't hope to communicate with anyone on more than a superficial level. If the people you're communicating with don't understand you, then *YOU* have failed, not them.

And since working together well requires good communication, we'd better figure out how to have empathy for each other.



who, me?

You might be surprised, but you already have empathy. Everyone in this room, even the skeptics, already has tremendous amounts of empathy, in fact! It's just not for people. We already have empathy for the compiler and computer: we ask ourselves, what is the computer thinking now? What does the compiler think the state of a var is, and what will it do next?

We know how to play compiler. We need to learn how to play colleagues.



Empathy is not a new tool in the engineering world.

Our colleagues in the user experience and design worlds have been using it for years to put themselves in the shoes of their end-users and customers.

What I'm trying to do is to bring it inside our engineering teams so we can have empathy for each other, as well as our futures selves.



now what?

So, if you buy my argument that engineers are more important than engineering, and teams are more important than technology, then now what do we do?

How do we actually use empathy in our day to day work?

At the beginning of this talk I mentioned that we came up with a number of software best practices at iRobot. Let's look at one of them through the lens of empathy. That best practice is that our software should be Readable.[CLICK]



readable

Software should be readable. The question is: readable by who?

Who is the most important consumer of our code? The compiler or people?

The compiler is a fool, and you can tell him I said that. Look at what a compiler is happy with:

```

#include <X11/Xlib.h>
#include <unistd.h>
typedef long O; typedef struct { O b,f,u,s,c,a,t,e,d; } C;
Display *d; Window w; GC g; XEvent e;
char Q[] = "Level %d Score %d", m[222];
#define H(r) (random()%r)
#define U l[ne]=a[l[n]-1]; ne=222
#define K c=-l.u; l=I[i]; l.t=0; c+=l.u
#define E l.e=-66!-l[l.e].d66(l[l.e].t=3)
#define M(a,e,i,o) a[i]=e,(a[i]=i)&&XFillPolygon(d,w,g,{void*}a,o,1,1)
#define F return
#define R while(
#define Y if(!t

    O p
    D,A=0,Z
    O,m=0,W=400
    = { 33,99, 165,
    XGCValues G= { 0,0
    T[] = { 0,300,-20,0,4
    d,-20,4,20,4,-5,4,5,4,
    0,-4,4,-4,-4,-4,4,4,4,
    M(i,a[i],m,12); } Me(i,l,o
    l.t=16; l.e=0; U; } nL(O t,O
    l.d=0; l.f=s; l.t=t; y=-l.c+b;
    %2xx; t=y||%2xy; l.u=(a+s>t?;
    U; } d(i C I){ O p,q,r,s,i=222;C l;
    -l.s>>9; q=l.c-l.c>>9; r=l.t=0;l.b;
    26) F s+=10; s=(20<<9)/(s||1); B=s>9;
    R i--66(xca[i]-q)(xsa[i]+d); F i; }
    Y{ r++;c=l.f; Y=s{c=l.u; l.t=0;
    (l.s>>9)++l.a,h=-l.a,l.a+2,l.a+2,0
    (b,l.s>>9,h,6); else XDrawPoint(d
    (l,20); K; } Y66l.c=266(d(i l)||b-
    A); Me(l,30); Y=1{ E;K; } else

    d(i){ O
    E; }R c--{--
    ,00<<8); if(!l.u){
    ,w,g,(l.s+=l.a)>>9,
    H{ if(h>M66(c+h
    c+l.t=0; } Y=166h
    H(W<<9),H<<9,1,i=
    1); I[i].d++;
    }R N(3)

    K;
    l.umc; c=0; } Y
    =2{ l.s=l.a+B;
    l.a= (l.e-l.s)/(l.h+
    20-h)||1; l.c+=l.b+0;
    M(b,l.s>>9,l.c>>9,6); }
    } L[i]=1; } } F r; } J(){
    R A{ XFlush(d); v66sleap(
    3); Z+=v+10; p=50-v; v%266h1
    ((a[A]=H(W-50)+25),50)<0 66A++;
    XClearWindow(d,w); for(B=0; B<A;
    oC(B++); R Z(d,c){ 266:H(p)66(2-
    ,rl(1+H(p),H(W<<9), 0,H(W<<9),H<<9,1
    ,0); usleap(p+200); XCheckMaskEvent(d,
    4,6e)6666-6666(4,a[H(A)]<<9,H-10<<9,e,
    xbutton.xec9,e.xbutton.yec9,5,0);S+=A*100;
    Bsprintf(m,O,v,S); XDrawString(d,w
    ,g,W/3,H/2,m,B); } }

```

Do you want to hang out with someone who thinks that's acceptable code? Here's a closeup...


```
#include <X11/Xlib.h>
#include <unistd.h>
typedef long O; typedef struct { O b,f,u,s,c,a,t,e,d; } C;
Display *d; Window w; GC g; XEvent e;
char Q[] = "Level %d Score %d", m[222];
#define H(r) (random()%r)
#define U l[ne]=L[n]<=; ne=222
#define K c+=L.u; l=L[i]; l.t=0; c+=L.u
#define E l.e=66-L[l.e].d66(L[l.e].t=3)
#define M(a,e,l,o) a[0]=e,(a[1]=166XfillPolygon(d,w,g,(void*)a,o,1,1)
#define F return
#define R while(
#define Y if(!t
```

```
} L[i]=l; } } F r; } J(){
R A) { XFlush(d); v&&sleep(
3); Z=++v*10; p=50-v; v%2&&hi
((a[A]=N(W-50)+25),50)<0 &&A++;
XClearWindow (d,w); for(B=0; B<A;
dC(B++)); R Z|dL(){ Z&&!N(p)&&(Z--
,nL(1+!N(p),N(W<<9), 0,N(W<<9),H<<9,1
,0)); usleep(p*200); XCheckMaskEvent(d,
4,&e)&&A&&--S&&nL(4,a[N(A)]<<9,H-10<<9,e.
xbutton.x<<9,e.xbutton.y<<9,5,0);}S+=A*100;
B=sprintf(m,Q,v,S); XDrawString(d,w
,g,W/3,H/2,m,B); } }
```

```
M(b,l,6669,l,6669,6); }
} L[i]=l; } } F r; } J(){
R A) { XFlush(d); v&&sleep(
3); Z=++v*10; p=50-v; v%2&&hi
((a[A]=N(W-50)+25),50)<0 &&A++;
XClearWindow (d,w); for(B=0; B<A;
dC(B++)); R Z|dL(){ Z&&!N(p)&&(Z--
,nL(1+!N(p),N(W<<9), 0,N(W<<9),H<<9,1
,0)); usleep(p*200); XCheckMaskEvent(d,
4,&e)&&A&&--S&&nL(4,a[N(A)]<<9,H-10<<9,e.
xbutton.x<<9,e.xbutton.y<<9,5,0);}S+=A*100;
B=sprintf(m,Q,v,S); XDrawString(d,w
,g,W/3,H/2,m,B); } }
```

The compiler is easily amused. Who cares what it thinks. Satisfying the compiler is necessary but not sufficient for great software. Now, if your company is okay with software that looks like that...

The iRobot logo is displayed in a bold, italicized, green sans-serif font. The 'i' is lowercase, while 'Robot' is uppercase. A registered trademark symbol (®) is located at the top right of the 't'.

is hiring...

www.irobot.com/About-iRobot/Careers

csvec@irobot.com

We are hiring all kinds of engineers - check out our website and email me or talk to me today or tomorrow!

Smart Motivated Eager

Approaching readability with empathy means that your code should be understandable by a new colleague. You can assume they're smart, motivated, and eager, but they don't have your memories, assumptions, or experience.

Meet them where they are, not where you are, and teach them what you know!

What we know matters, **but what our colleagues don't know matters more.**

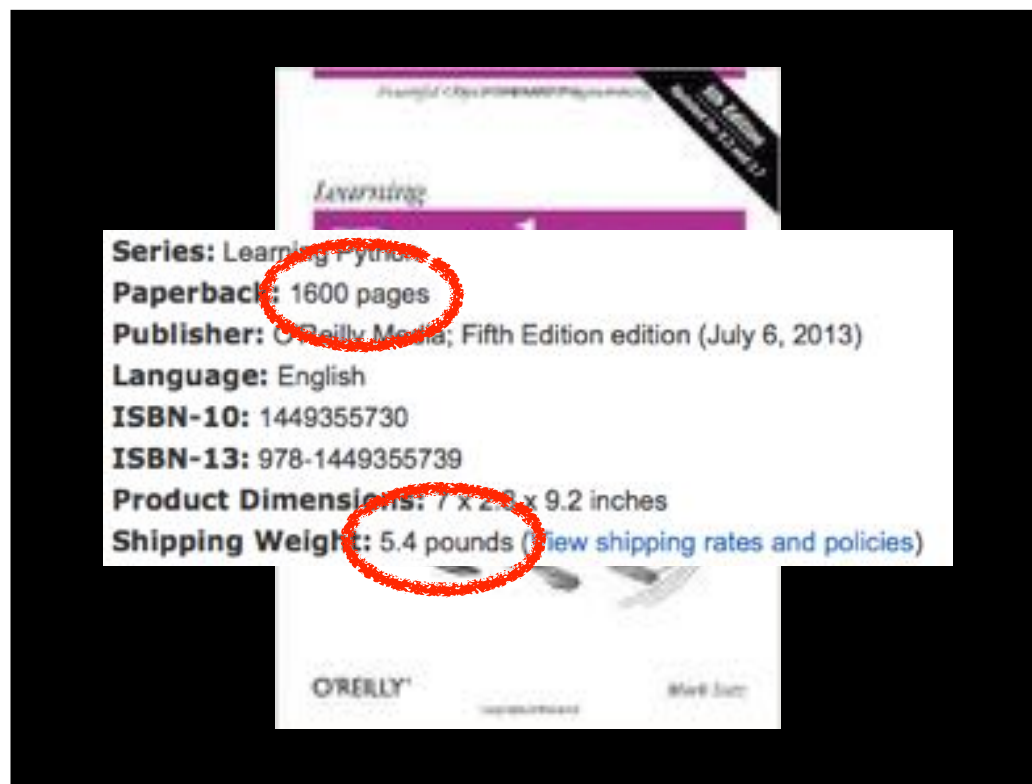


And don't overestimate your own memory either: one day you're going to be frustrated by some code, cursing whatever fool wrote this confusing mess, do a 'blame' on it, and realize it was you. Have empathy for your colleagues, and for your future self.



As I near the end of the talk, I want to leave you with a question, and a conclusion.

The question is the one I started with: "When you started your most recent new job, who had everything they needed to learn their new software/hardware/product and contribute immediately?" "Who found it painful/tricky/difficult to learn what they needed to become productive?" Think about how a new hire would answer that question at your current job. Ask your most recent new hire that question. Listen to the answer. Do something about it. Your first instinct is going to be, "Well, they need documentation!"

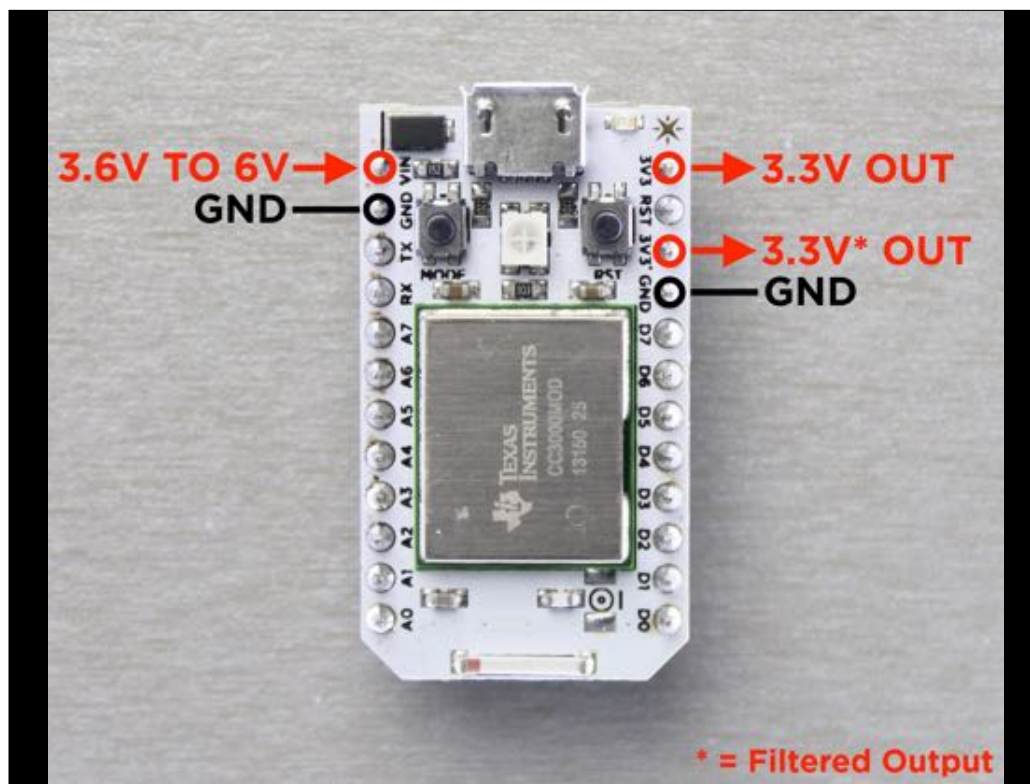


And so you go off and write some docs. And soon you have a book.

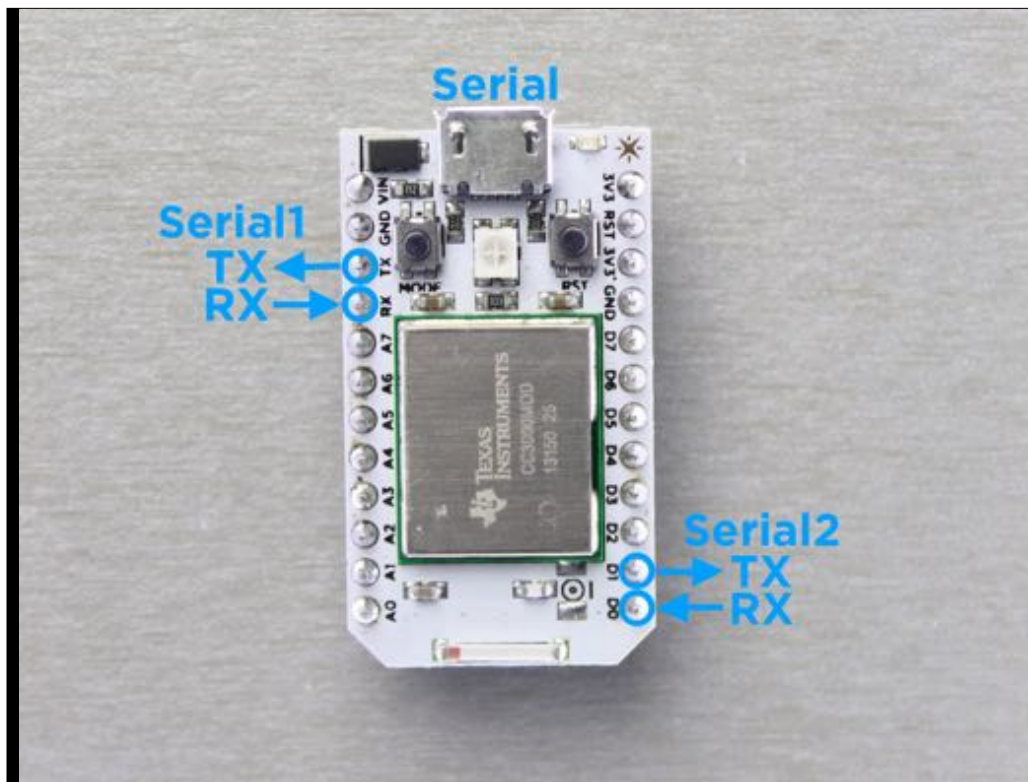
A 1600 page book. A 5 and a half pound book. That no one will read. And that doesn't answer any questions a new hire actually has. It's really easy to regurgitate everything you know without considering what a new hire needs to know, without actively thinking about where they're coming from.

I'm not saying docs are bad, but I am saying that **docs that answer questions that no one has are useless**. Stop and ask yourself, and your new hire, What do you need to know to do your job?

Maybe it's something like this...



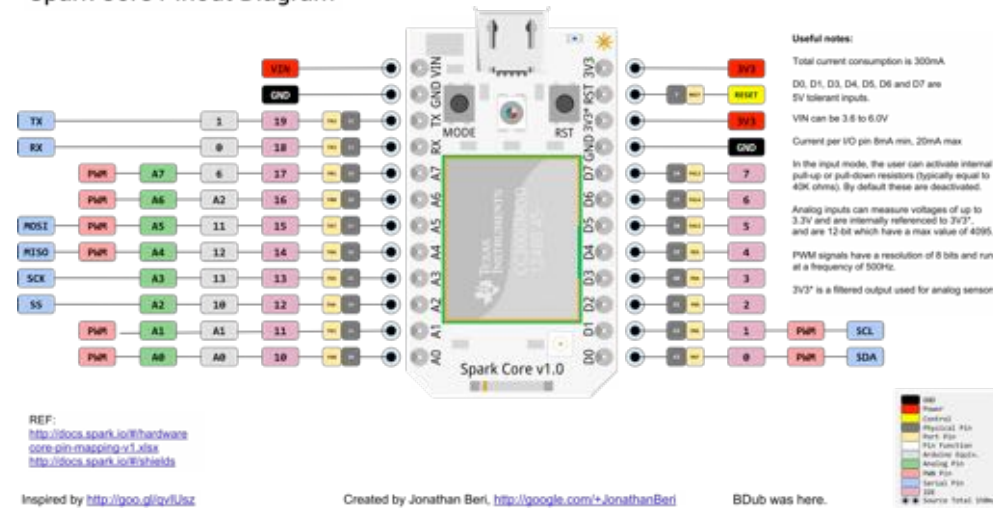
Spark's wifi module.
Or maybe like this...



Look! Direction arrows!!! Think of how many bugs you wouldn't have if directional arrows were everywhere, maybe even on the silkscreen.

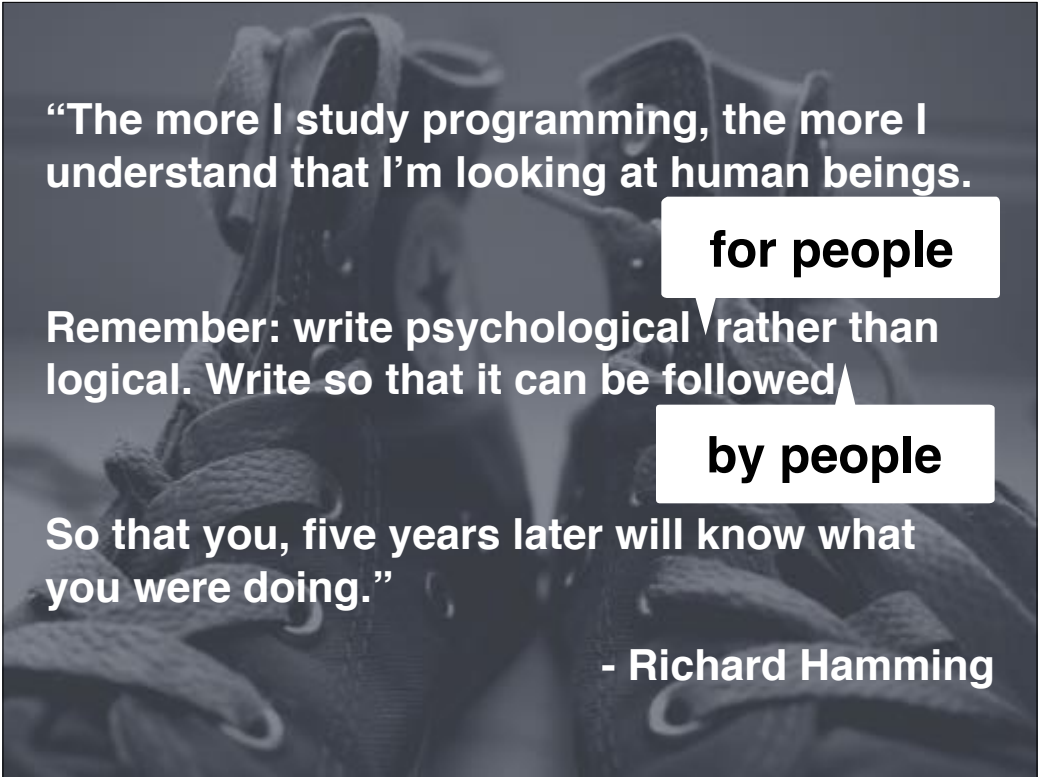
Or maybe like this...

Spark Core Pinout Diagram



You mean pinouts don't have to be black and white? That they can actually be color coded?

These are examples of what happens if you consider your audience, your colleagues, when developing hardware and software.



“The more I study programming, the more I understand that I’m looking at human beings.

for people

Remember: write psychological rather than logical. Write so that it can be followed

by people

So that you, five years later will know what you were doing.”

- Richard Hamming

To conclude, I’d like to revisit the opening quote by Richard Hamming.

This talk is a starting point. My hope with this talk was to get us all thinking about empathy towards our colleagues and our futures selves as an engineering tool. I hope something in this talk resonated with you and gets you thinking about how to apply empathy to your daily work.

Thank you.



Thank you!

Chris Svec

svec@saidssvec.com

[@christophersvec](https://www.instagram.com/christophersvec)

Image of shoes: D Sharon Pruitt, [https://
www.flickr.com/photos/pinksherbet/2037369037/](https://www.flickr.com/photos/pinksherbet/2037369037/)

Richard Hamming quote: [https://
www.youtube.com/watch?v=2e5_Z6oZ0rM](https://www.youtube.com/watch?v=2e5_Z6oZ0rM)

What I know Venn diagram: [https://medium.com/
@aliciatweet/overcoming-impostor-syndrome-
bdae04e46ec5](https://medium.com/@aliciatweet/overcoming-impostor-syndrome-bdae04e46ec5)

<http://docs.spark.io/hardware/>