Laying Foundations For SE Choices

Exploring Software Engineering Responsibility & Pragmatism

Some First Words

- I worry that we focus on speed (velocity) and approach (processes) without having our direction clearly set.
- This deck presents ideas to help you set broad directions that respect real world constraints and embody aspirational values.
- Warning: Like many most things in SE there is subjectivity here, prepare yourself for acceptable choices rather than perfect ones!

We Code in the World

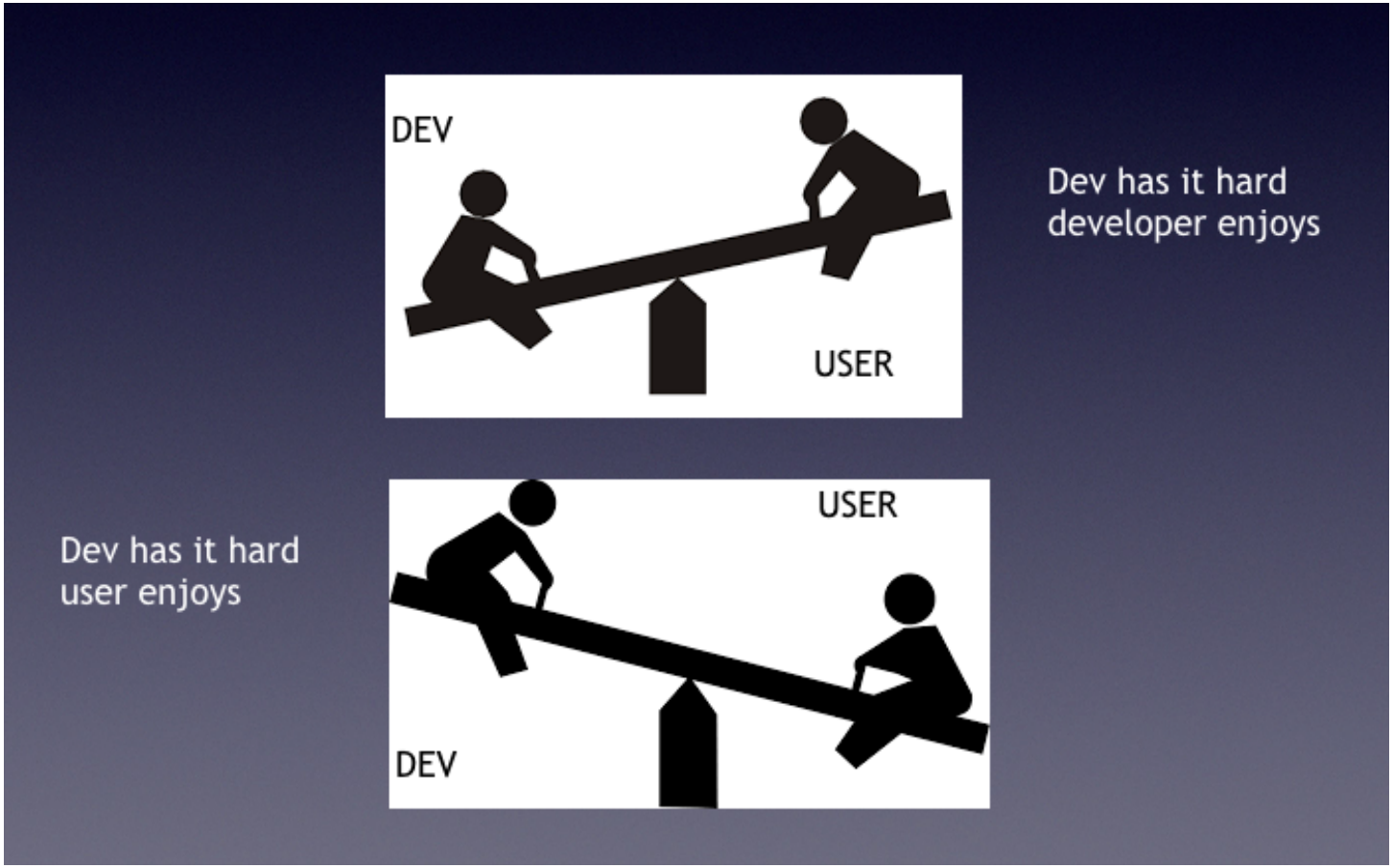
- As much as we might inhabit an abstract world of code algorithm and enjoy the more predictable world of code, the trying is we perform our work in the “real world” under real conditions for real people.

- We do not live in “Math Land” we do not construct “Theoretical Structures in Outer Space” we build in a messy universe for emotional people steeped in subjectivity…**we should proceed with caution!**

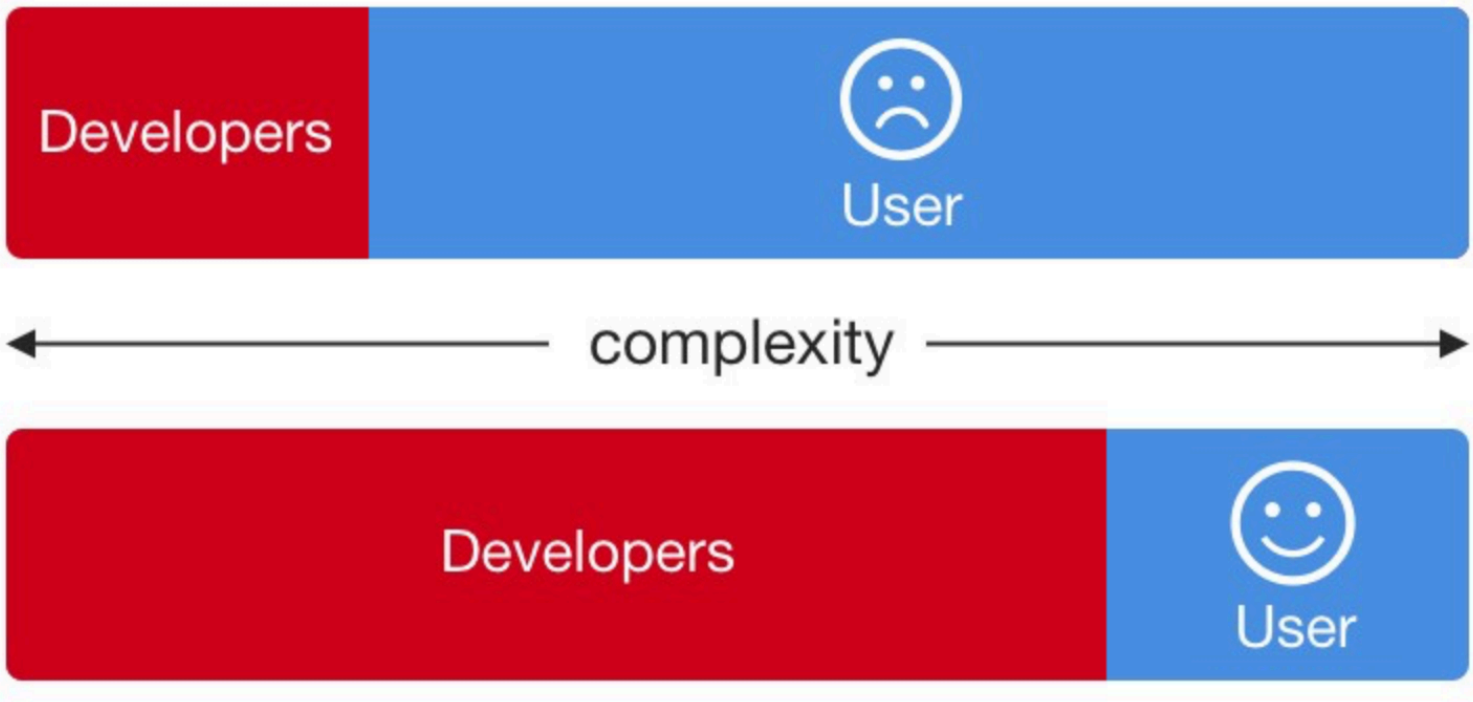
Things to ponder

- Who Do We Build For
  - Ourselves?
  - Others
- Usually software should be more in service of users than ourselves

Mental Model
UX / DX Seesaw



Smaller it Better!

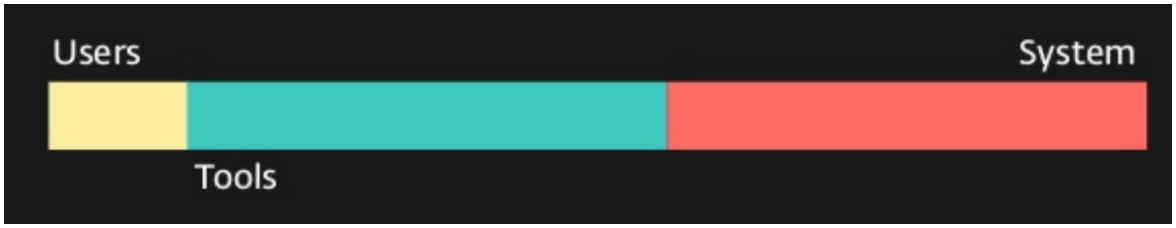


Relation to Tesler's Law?

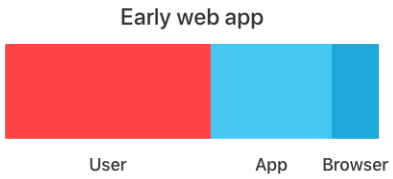
- In Software there tends to be a Conservation of Complexity
  - "Every application has an inherent amount of irreducible complexity"
- The question is who has to deal with that complexity? User or developer? Our choices often effect this, we might move it but do we eliminate it?

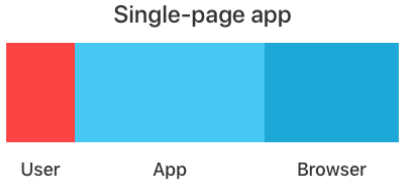
https://en.wikipedia.org/wiki/Law_of_conservation_of_complexity

Complexity Balance Visualized



More Specific Example





Shift the burden around, but the general complexity is preserved!

More to Ponder

- We are responsible for what we code, **not** just that the code is correct
- Our software is heavily involved in people's lives, so we should treat it with care.
- Don't dodge that last one, as even if we are not building a "life and limb" application our choices can effect people adversely !

Example UX Thinking for Engineers

- Problem: Human Fallibility
- Solution: Provide fail safes or back-ups
- UX/UI Examples
  - Many ways to access - keyboard / mouse, desktop / mobile, offline / online, etc.
  - Many ways to engage - linear / non-linear, textual/graphical, read / scan, passive / active

Things to ponder

- Sometimes we are in a spot where we feel we just build the thing we are told and our focus is on how to build that well? Hard to tell if that is right.
  - Does this absolve us from the potential negative outcomes?
  - Are externalities out of bounds since not code specific?
- The answers are clearly no if you consider it with any depth, but consider then if we want some say in what and why things are done outside of code we likely have to involve ourselves beyond the source code!

Things to ponder

- The last point about is metaphorically a “just working on the Death Star” vs “commanding the Death Star” idea at its extreme.
- If we absolve ourselves of such concerns via paycheck or role, that is a choice we can certainly make (and probably sleep better!).
- To me that absolution or avoidance of possible troubles is more the view of a coder, while the consideration of them the view of a Software Engineer—so pick your desired path.

Inescapable Fact: The software you make will effect people and the world. Act accordingly.

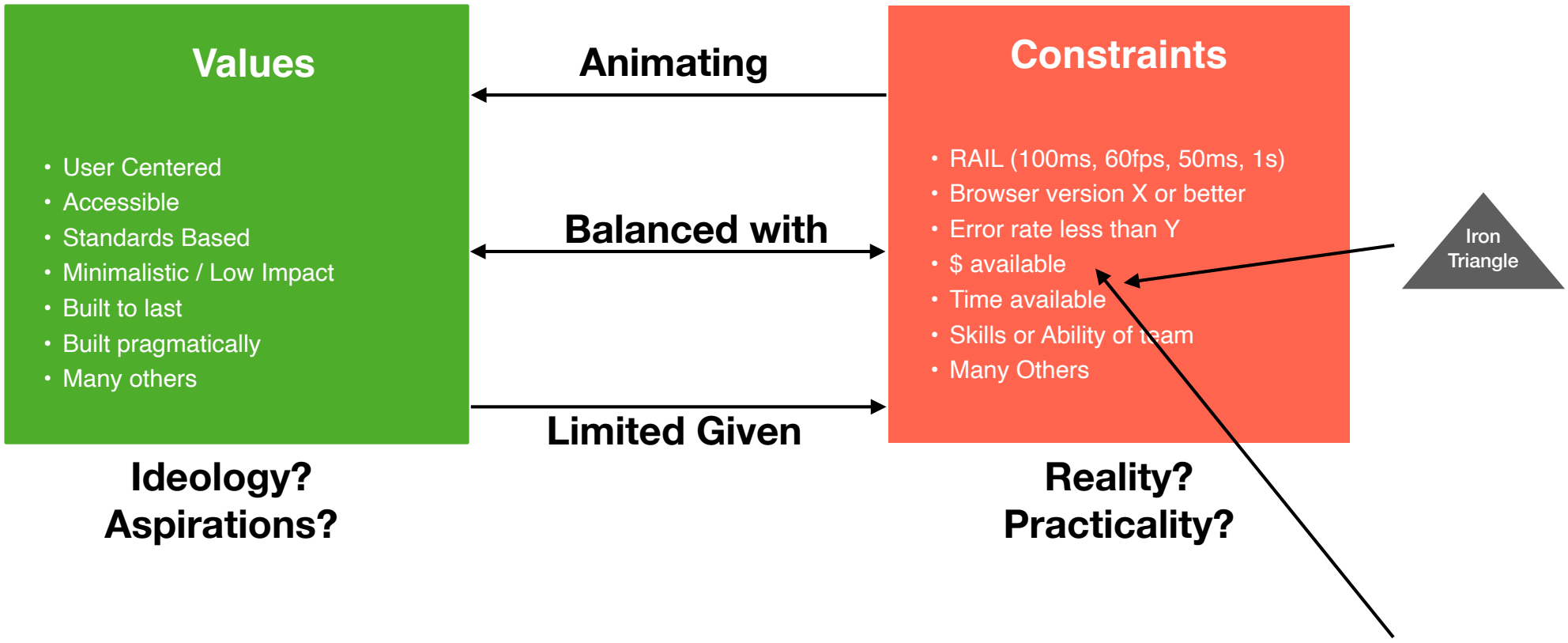
TL;DR - Just try to make informed and purposeful decisions and don't duck responsibility.

Solution: All dev should be "VC" driven!?

VC != Venture Capitalist

VC = Values + Constraints

Values & Constraints



Constraints are good?

Constraints Working Slide

- In this course what are our constraints?
- I'll start with the easy one - time!
  - Let's get specific though!
  - Question: What can we fit into that time? Does this require estimation? How do estimate if we haven't done something before!?!
- What else beyond time?
  - Hints: Iron Triangle, Team Member Specific issues

Values Working Slide

- What are the values we subscribe to?
- Are these specific to SE in general or is the team, org, culturally, etc. specific?
- Now let's state some values we might want to subscribe to for your team's projects

Careful Now - My Huge Hedge

- You'll note I have said **nothing** about what YOU should do other than you should consider some values and respect the constraints
- The particular values YOU adopt and the degree you respect the constraints are the CHOICES YOU AND ONLY YOU MAKE
- My only belief that is unshakable is that for you to be considered a Software Engineer is that you acknowledge you are indeed responsible.

🤓 *I can share my current values if you like*

Practice Intentional Engineering

- Given the last point whatever you code should follow the intentions you define based upon your values and the project's constraints
- Without intentions that are clear and set you are performing at best a blind or random walk towards a solution.
- You might luck out and get there, but it will be surely be art or luck and **not engineering**.

Pick a Few Intentions at Most

- It is tough to intend to build fast code, cheaply, that is reliable, fun to use, and easy for you to dev
- You need to pick your intentions carefully and have just a few of them so you can reasonably refine your work to meet them.
- Be ware of the Prof's favorite trade-offs!

Refine your -ility

Finely sanding my code for -------

Maintainability ✅

Reliability ✅

Secureability ✅

Usability ✅

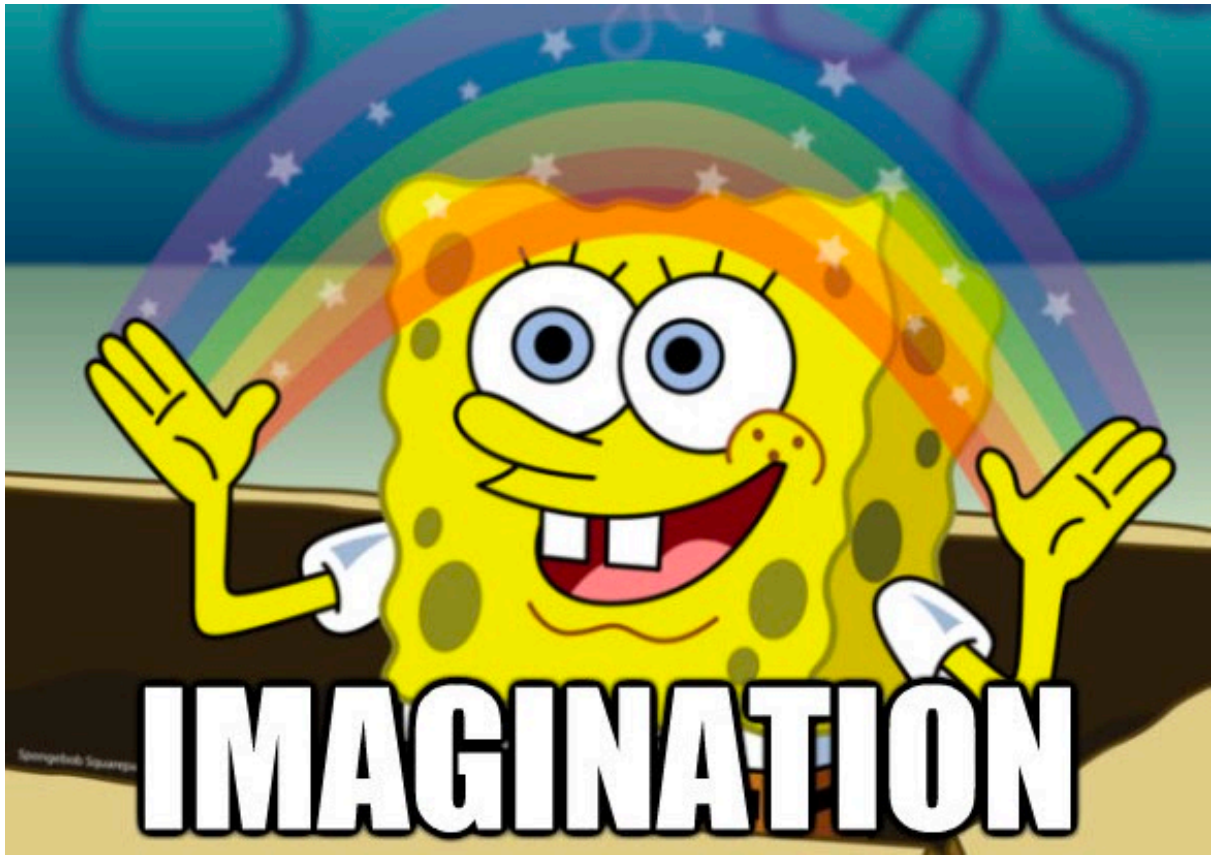
❌ **Performance - ility!**

🤣 *No -ility for that one*

"Percentage Effort" versus Binary Choice

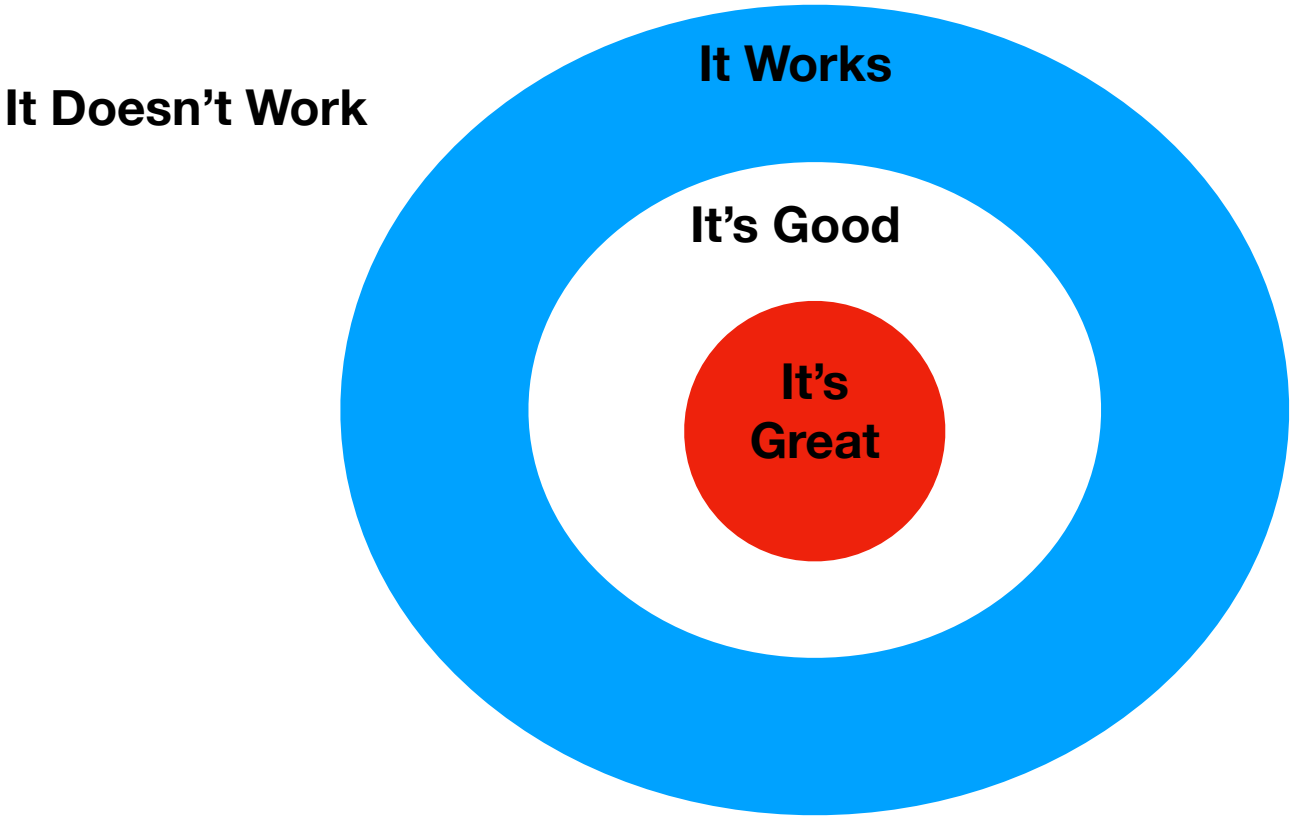
- When you think of the various -ilities they aren't necessary all or nothing
- Just because we might see the trade-off of security and usability, doesn't mean we have to forgo all security because we want usability - think balance or percentages.
- Your -ililities might be balanced with a percentage thinking of some fixed amount. Relate this to how you might distribute "points" on a game character's strength, dexterity, wisdom, and so on

-Ilities Spread

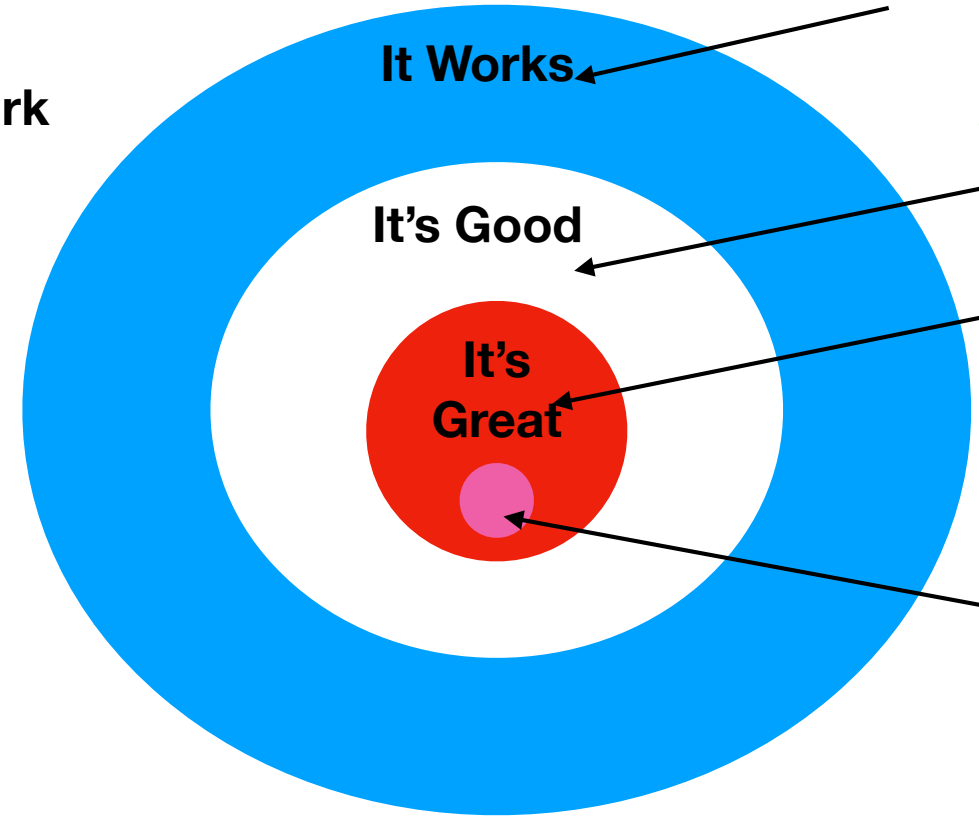


Still To Do: “Graphic or Gif on left of character stats or radar chart vs project character stats with a limit of points available to allocate, do the same for a software project”

Getting Our Code to Hit the Bullseye?



Hit the Bullseye?



"How is working defined? Are there degrees?"

"What does good even mean?"

"What does great mean?"

It's perfect!

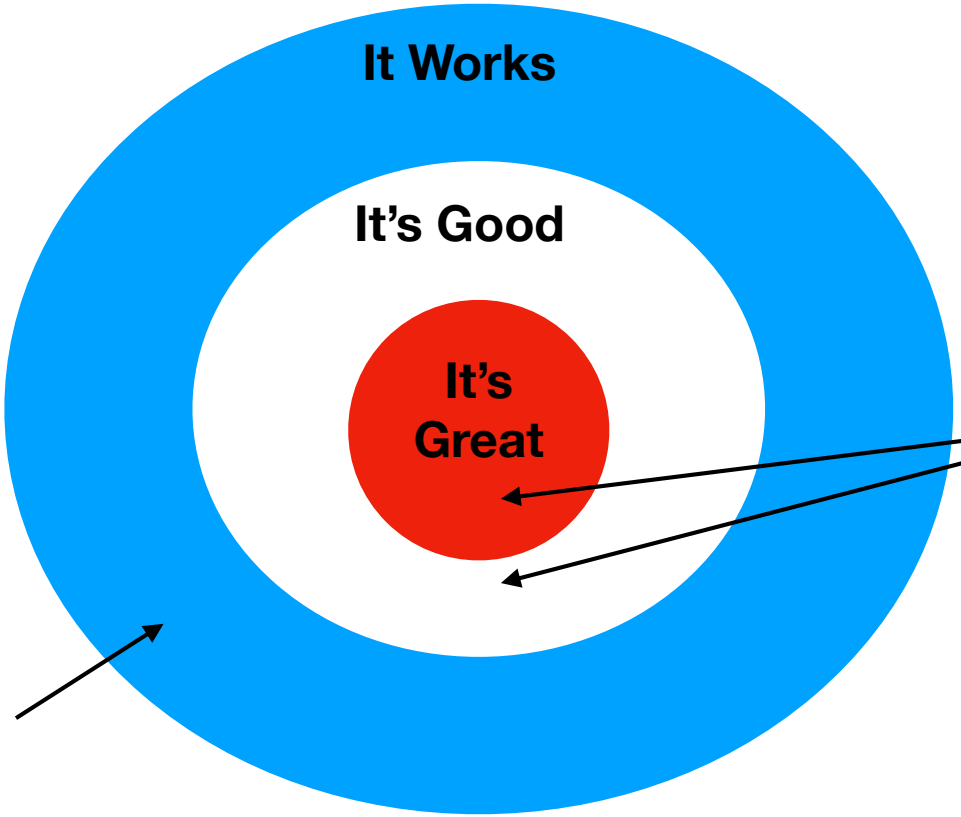
"Is perfect even possible?"

It Doesn't Work

"This one makes sense!"

🤔 *Could those subjective words relate to our intentions?*

Getting Our Code to Hit the Bullseye?



It Doesn't Work

Not a solution
Failed attempt

Functions and has significant positive subjective characteristics

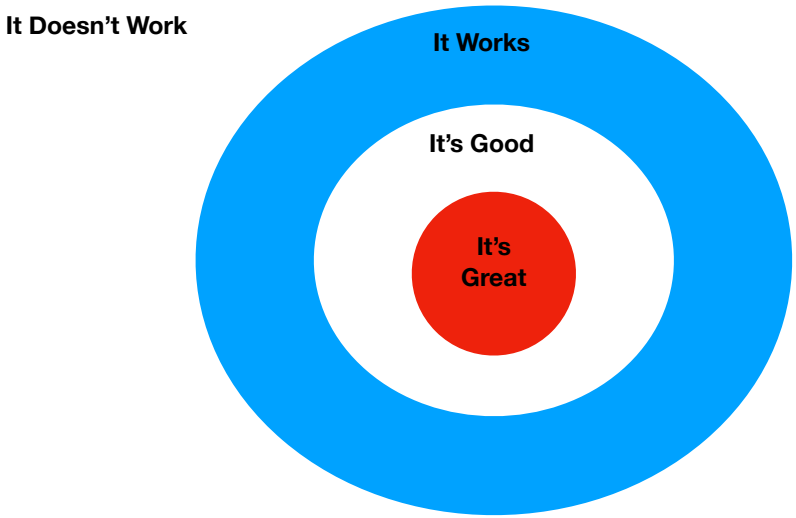
Examples

Maintainable
Performant
Elegant

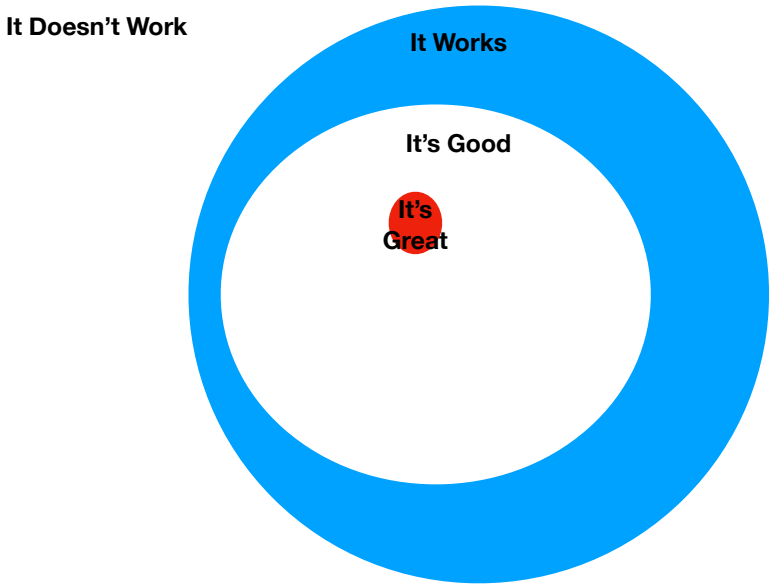
Functional with little positive subjective characteristics

Target Shape Surprise

Ideal Target

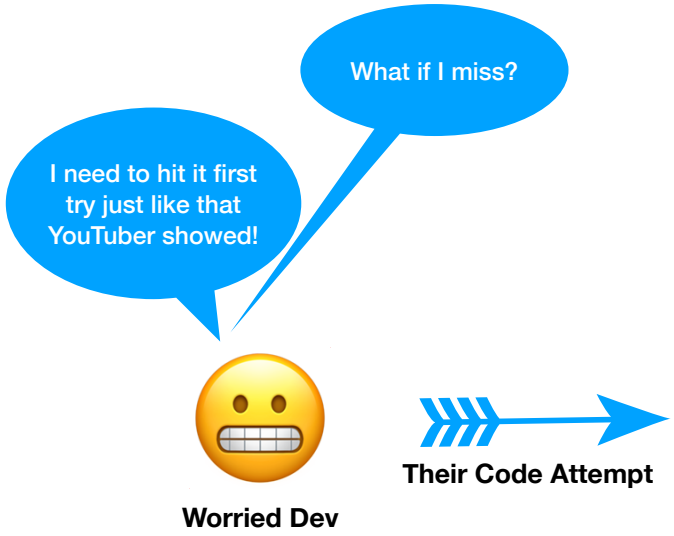


Actual Target

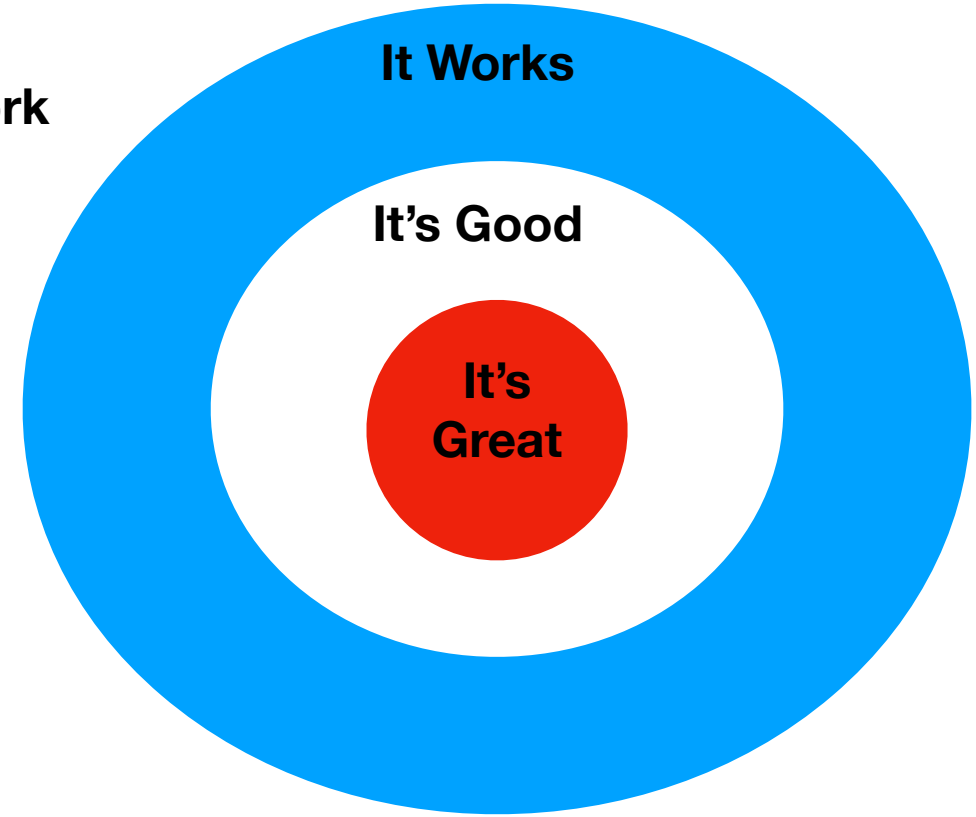


The "shape" of the works / good / great targets vary depending on your goals and values. Making them oblong was purposeful to represent the potential trade-off or lack of balance in some goals (ex. Performance vs Security)

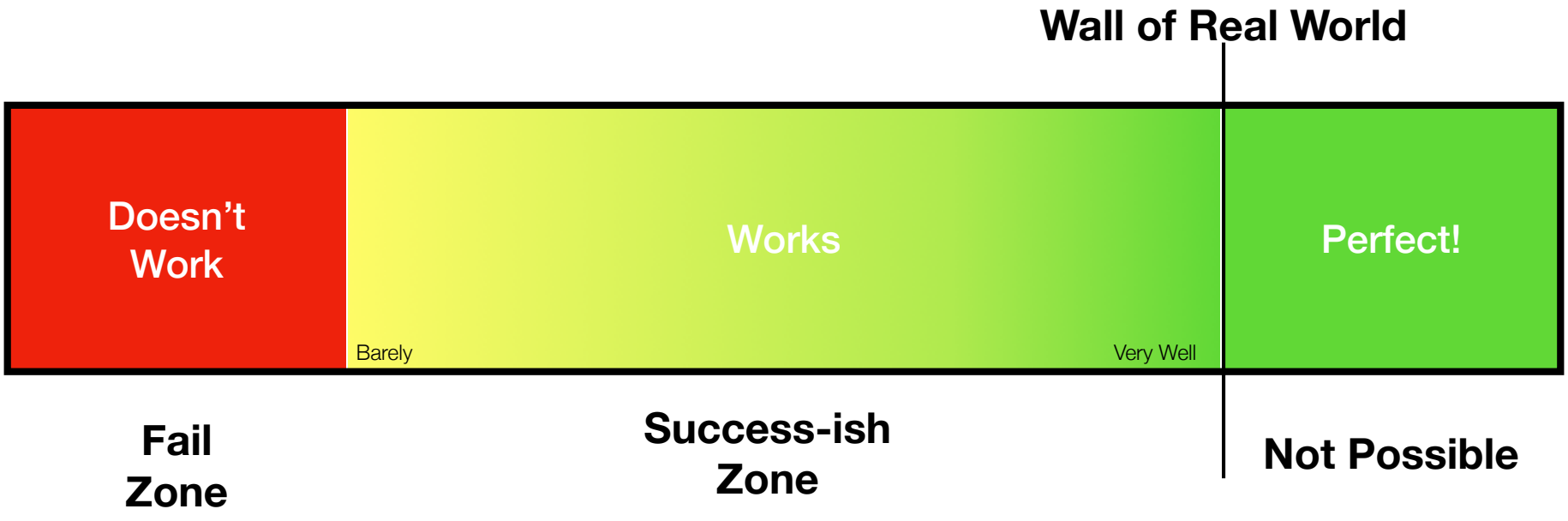
Hit the Bullseye?



It Doesn't Work

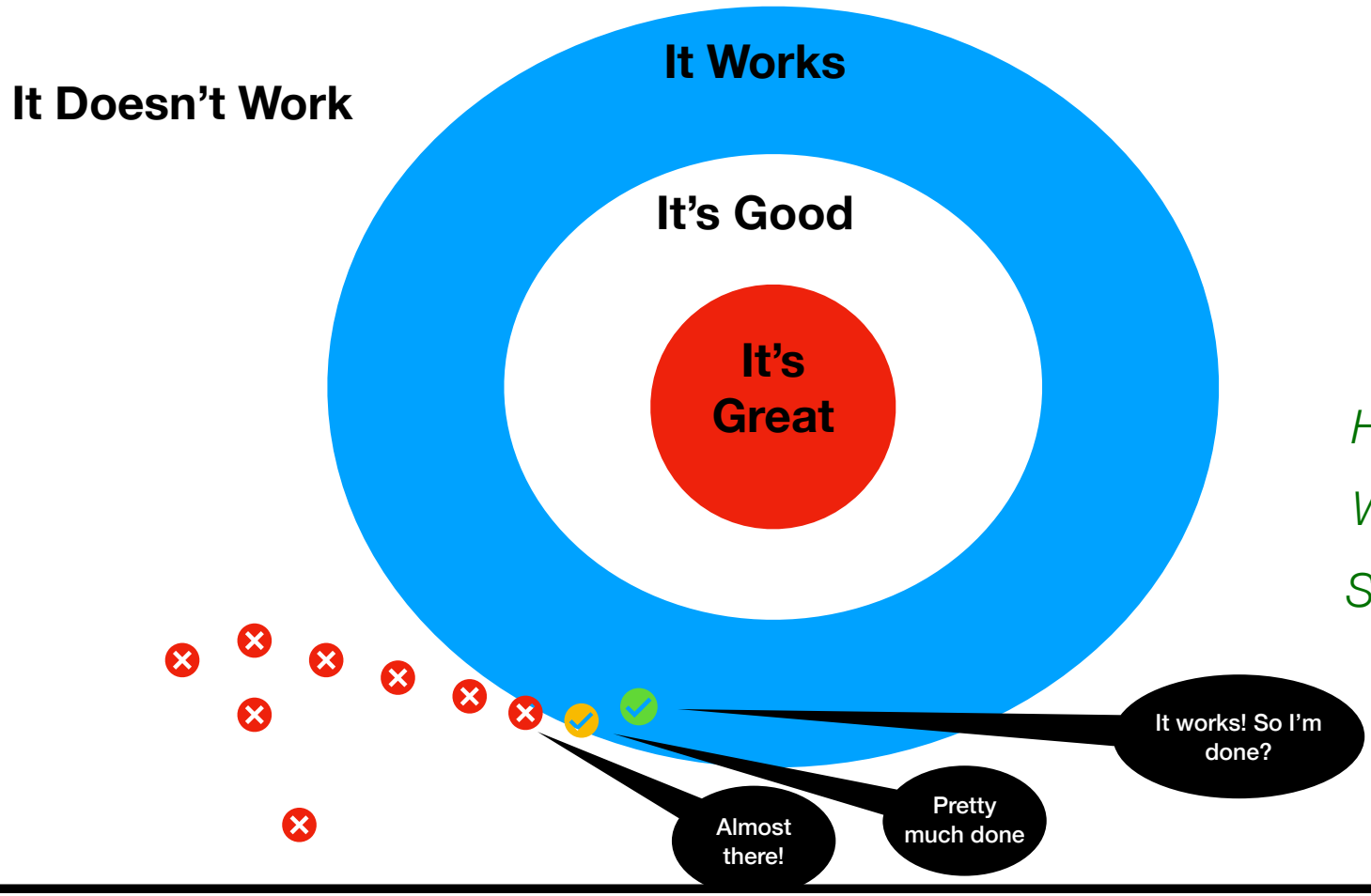


Danger - The Viz Form Isn't the Point



How far do we need to go into the zone?
How long and how much effort will it take?

Shots Fired!



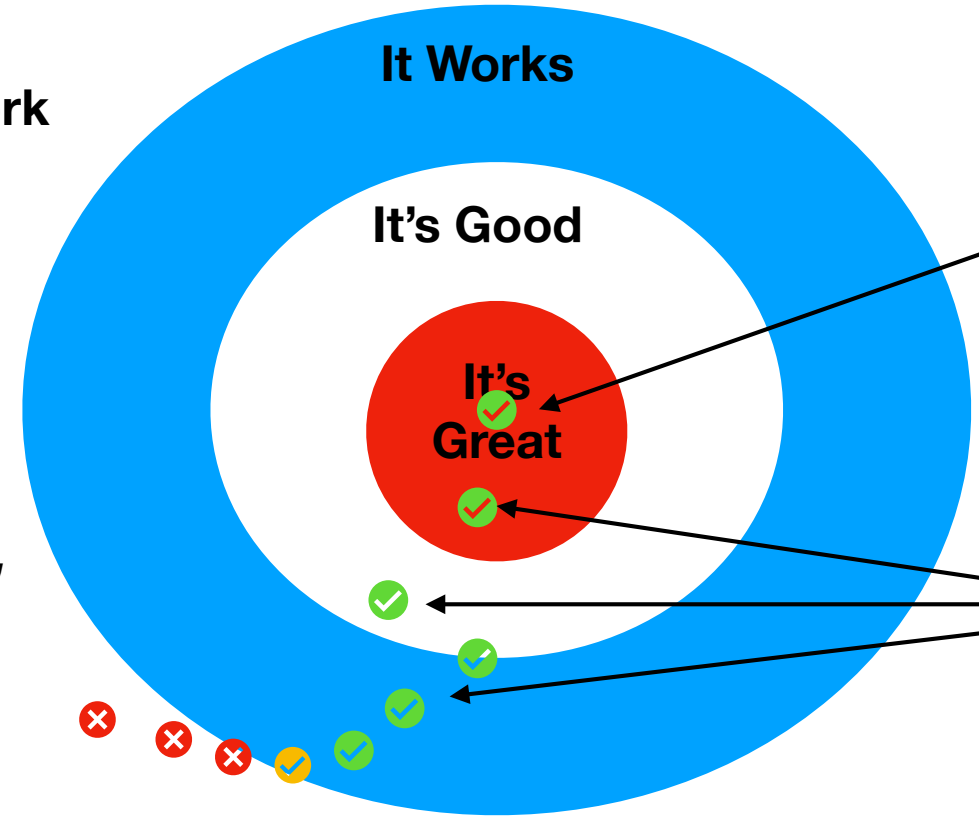
How many shots?
Over how long?

How close to the target?

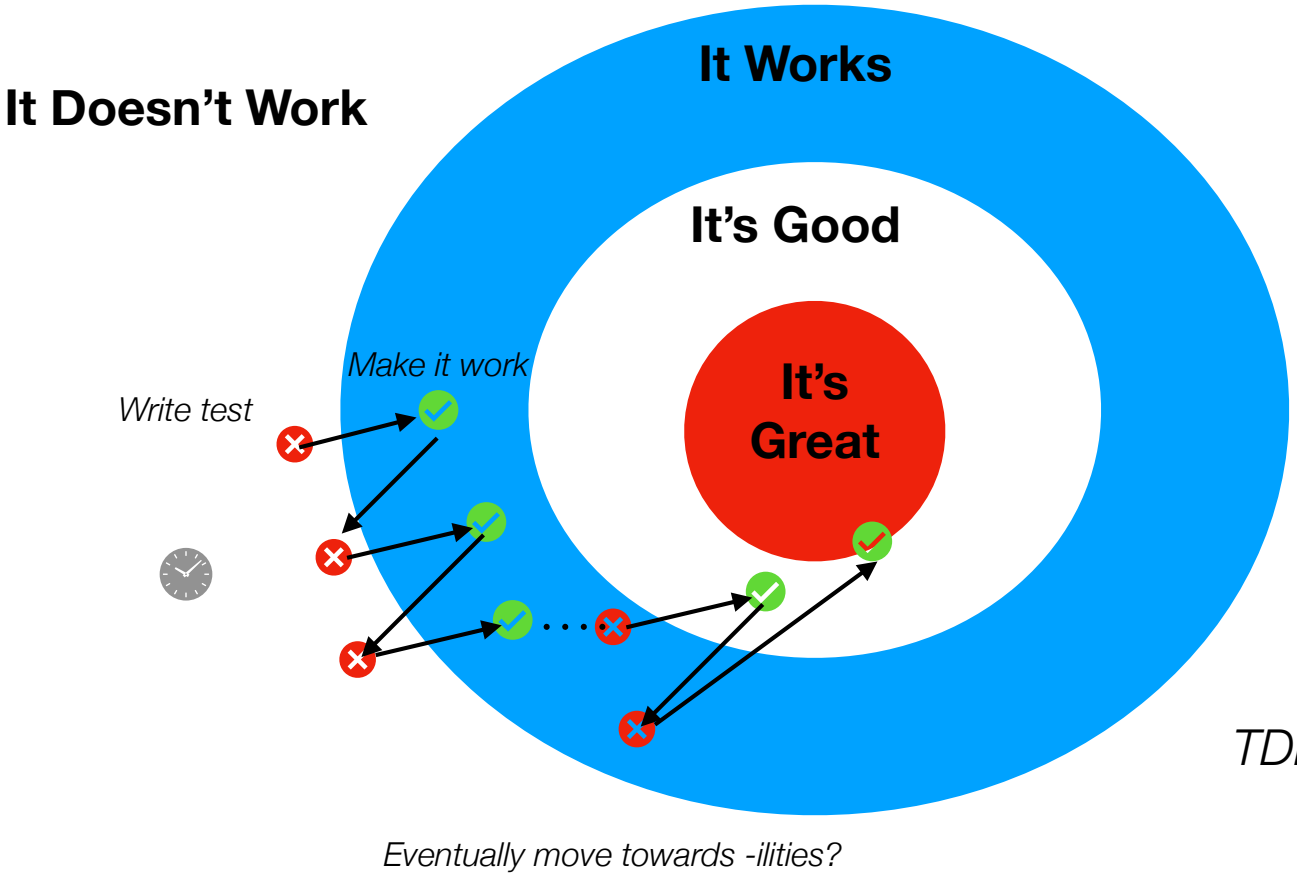
Who defines the target!?

Shape and size of target?

Hit the Bullseye?



TDD + Bullseye

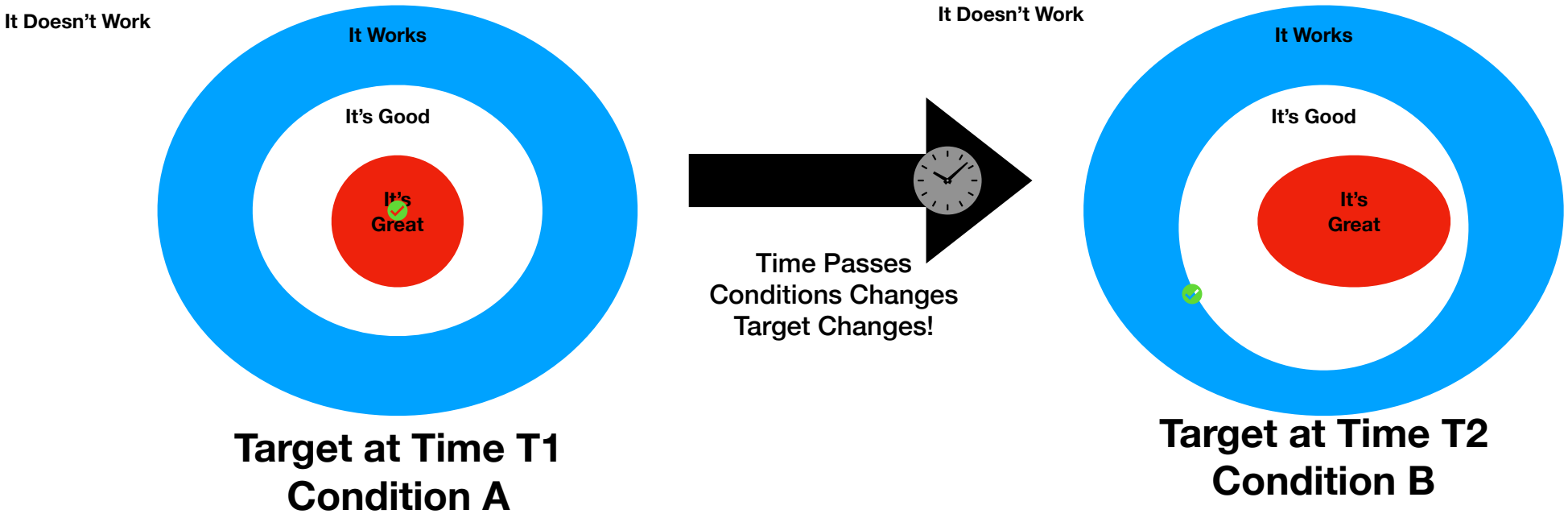


Eventually move towards -ilities?

TDD is very step wise and iterative

Time and patience required!

Target Changed



It's a moving target! Solutions aren't stable! Size and shapes can change!

Summary

- Inescapable Fact: We develop in a chaotic real world, for real people with subjective beliefs on all sides.
- Admitting to our role, adopting values and respecting existing constraints is a key characteristic of being a responsible engineer as opposed to a mere coder.
- Defining our project's values and using them in conjunction with constraints provides "guard rails" as we move iteratively towards our bullseye. This is the premise of intentional engineering.