

Latitude

A thesis for the development of a special breed of product developers.

June 2021

Created by:



Samuel chan
@onlyphantom

Created for:



Supertype
<https://supertype.ai>

Latitude

A thesis for the development of a special breed of product developers.

Content

Analogy

Audience

Conviction

Occupational

Anecdotal

Customers

Shipping

Multidisciplinary

Examples

Deliberation

Hammer

Supertype

Culture

Byproduct

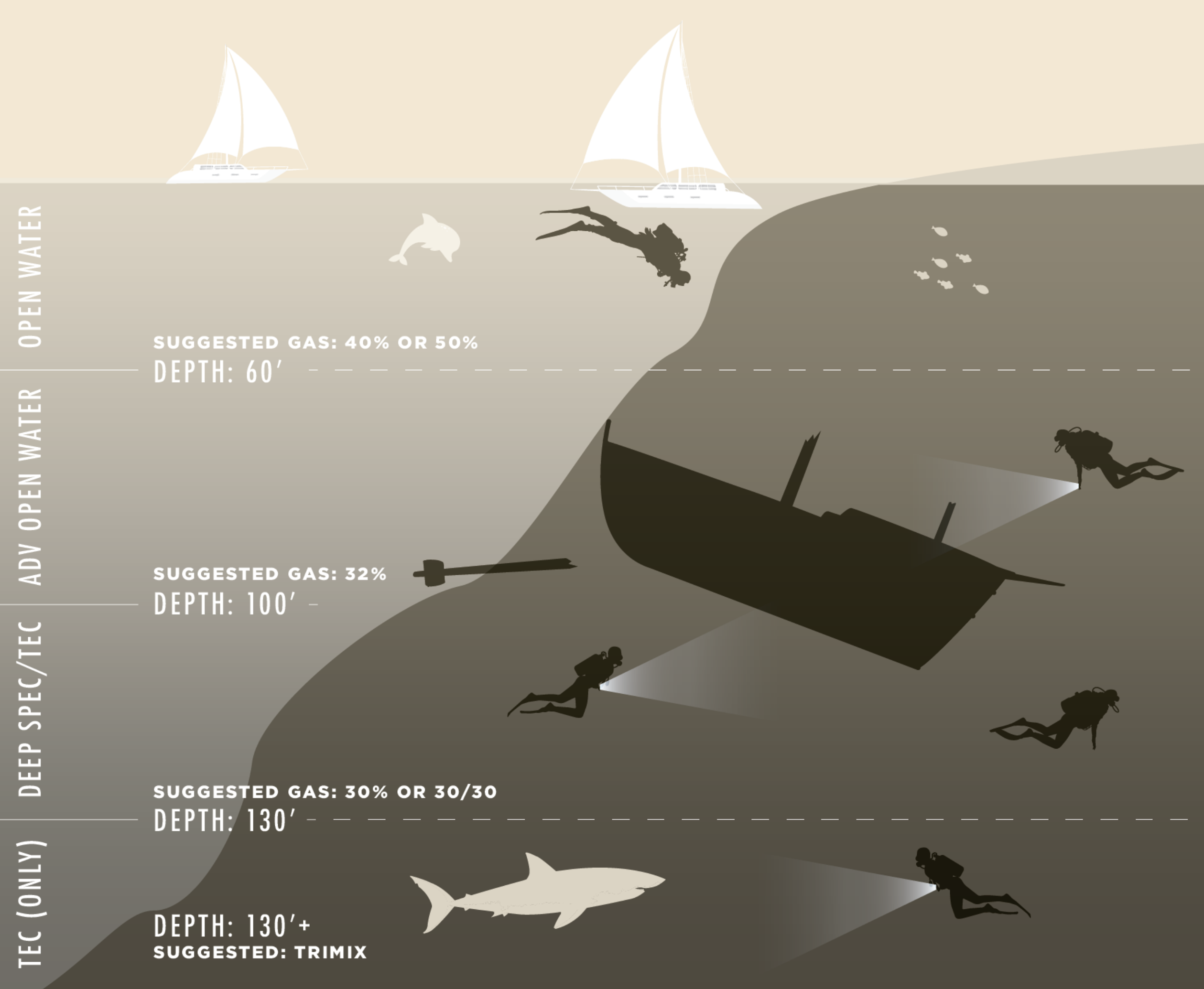
Objections

Actions

Rules

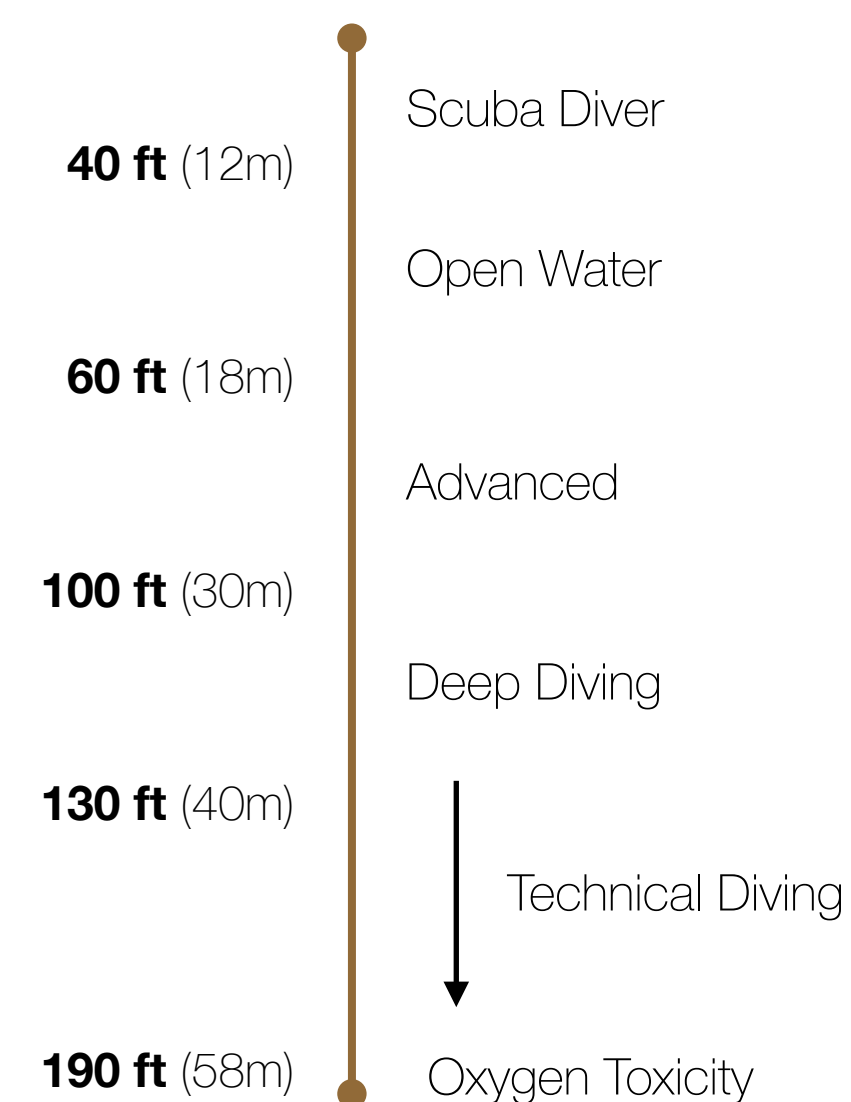
A human being should be able to change a diaper, plan an invasion, butcher a hog, conn a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a new problem, pitch manure, program a computer, cook a tasty meal, fight efficiently, die gallantly. **Specialization is for insects.**

Robert Heinlein



Depth is not just a number. That's not to say humankind has never far surpassed what was thought physically possible. World records holder Ahmed Gabr dived to 1,082 feet (332 meters) in 2014, succumbing himself to a pressure of 485 pounds per square inch. Most people's lungs would be crushed at that depth. A subsequent attempt to break this record (1,200 feet) was by Dr. Guy Garman, one of the most knowledgeable deep technical divers of our time, resulted tragically in death.

An overwhelming majority of divers do so **recreationally** with a maximum depth of 40 meters. They do it to have fun and seek out dive locations to **maximize this enjoyment**.



Analogy

Audience

Many people picked up programming and data science **recreationally**. They want to maximize this enjoyment — largely characterized by a semi-professional level of proficiency.

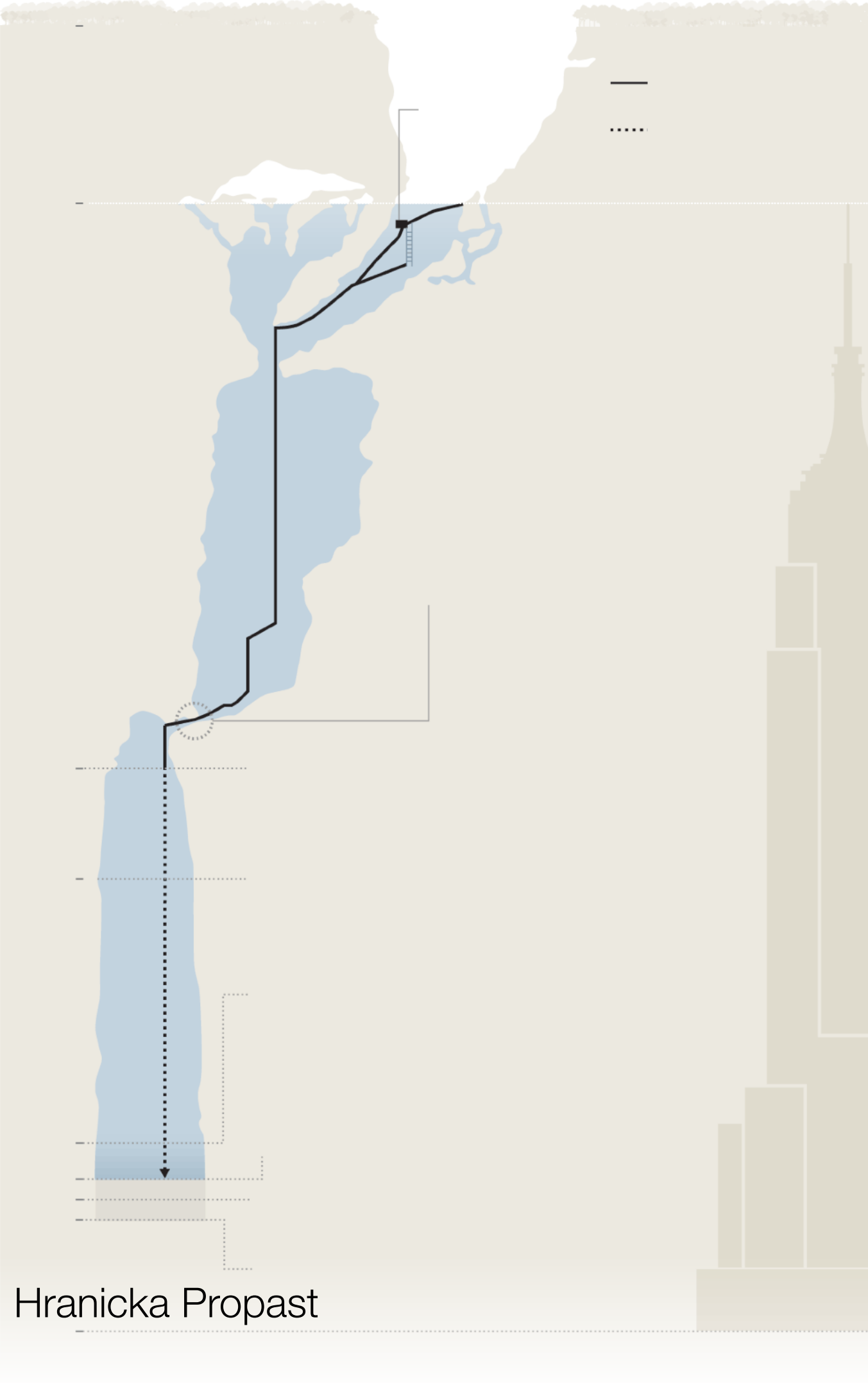
Audience

There is nothing wrong about doing data science recreationally or stopping at a certain level of “depth”. The choice is yours and only yours. This is not a learning path intended for those audience because your development needs can be better served reading one of the many articles available on this topic.

In fact, I actively cautioned my team members against taking advice from me on learning paths. In one instance, a team member (@evelinesurbakti) had to say — and I quote — “Please slap me with that hardcore” when I made the disclosure that **my learning path laid out in this thesis is generally a poor fit** to most people venturing into programming and data science.

One last time, find a learning path that suits your personal aspiration and ambitions. It is **most likely** not this guide.

Audience



Having made the disclosure that this guide isn't written for 90% of the programming population, this guide is free to directly address the remaining 10%.



Conviction

A large, bold, beige-colored number '10' is positioned on the left side of the slide. The '1' is a simple vertical bar, and the '0' is a thick, rounded ring. The number is centered vertically relative to the main text area.

There is a lot of debate as to the legitimacy of the 10x developer, with arguments on both sides arguing for or against the notion of software engineers that operate at a 10x productivity than his / her peers.

1. Supertype as an organization, believes and advocates for the 10x product developer. It exists, in our opinion, the same way that world-class musicians that play a wide range of instruments exist, or singer-songwriters that produce 10x the amount of titles and collaborations that their peers, or wildly prolific authors that produce literature work with a readership of 10x the average writer.
2. Through anecdotal evidence, we've observed individuals over the course of their career, producing software, releasing updates, building full-fledged games like Roller Coaster Tycoon (Chris Sawyer), Stardew Valley (Eric Barone) and fully-featured softwares (Donald Knuth, Steve Wozniak, Jeff Atwood etc). We observe highly multidisciplinary developers take on what would typically require a squad of developers and deliver admirably with very limited resources.

Conviction

10

3. This is also an auditable claim. Open source projects have a public history of the project's code contribution from day 1. Most successful technology companies have a number of these developers among their first 20 hires.

4. Coding have a **remarkably low barrier of entry**, just like writing and playing basketball, lending to a very crowded field of participants. In these domains, there will be a 10x performer just on the back of the sheer volume of new market entrants who never moved past the amateur phase (Kobe Bryant vs the average person who play basketball recreationally). **Fields that have high barriers of entry are more averse to these phenomena** (eg. the best submariners, military aviators are unlikely to be 10x better than their peers) and fields that have a more predictable input-to-output mechanism (eg. The best lumberjack in the country is unlikely to chop 10x more woods than the worst in any given hour). Fields that are creative in nature are more susceptible to the 10x phenomenon. For the time it took three developers to build a high-school student attendance application, somewhere, someone may have written the first version of WordPress (Matt Mullenweg) or bring the genesis block of Bitcoin into fruition (2 months after the wallpaper). This is also true in songwriting, or writing in general (Stephen King, JK Rowling etc).

Occupational

Because of the lowering barrier of entry into coding, most coders are doing so recreationally. Think scuba divers and open water divers. But when a rescue mission is at hand, or an offshore construction calls for some underwater engineering inspections, the situation has room only for occupational specialists. Even if you plan to spend a lot of time near the surface for recreational dives, developing your ability to go deep makes you a specialist.



Anecdotal

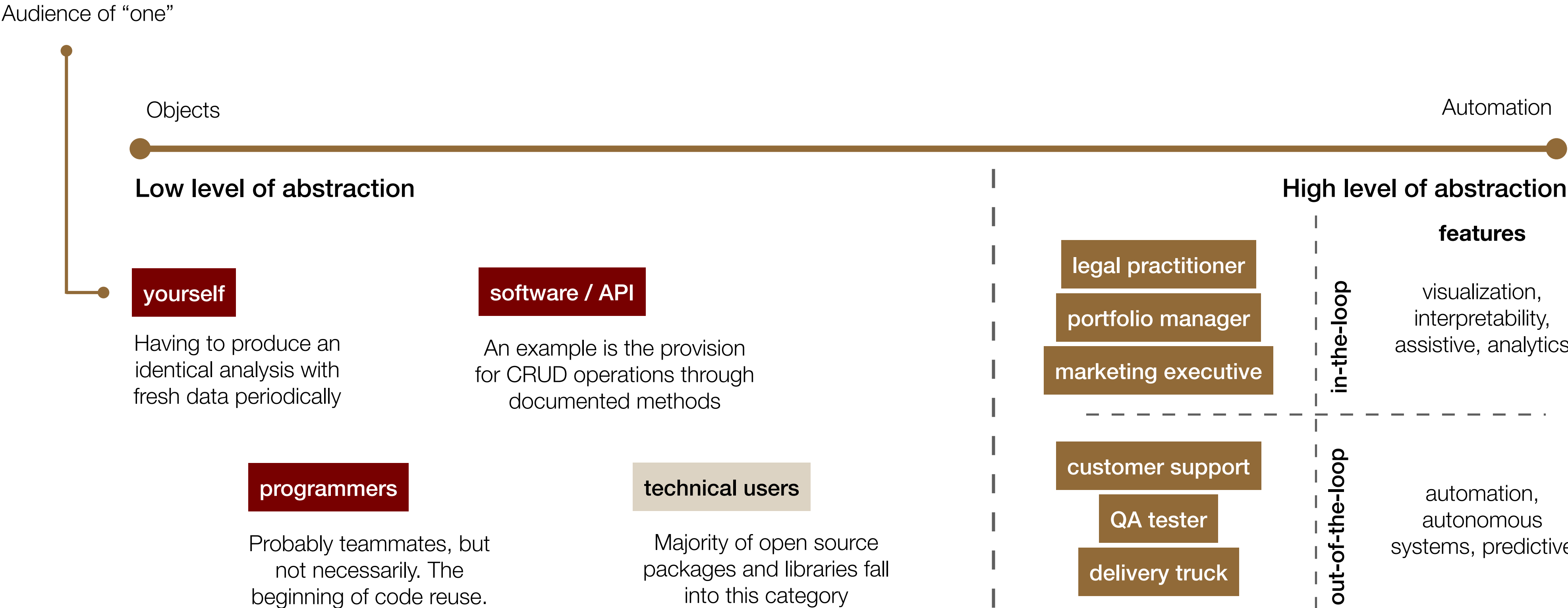
If your foray into programming was running code in “code notebooks” like Jupyter and Google Colab, you may think writing code iteratively, cell-by-cell is how most developers write code.

You’re mistaken. If you want to create value, **you need to write code for others**. You need customers for your code. You get paid for having customers.

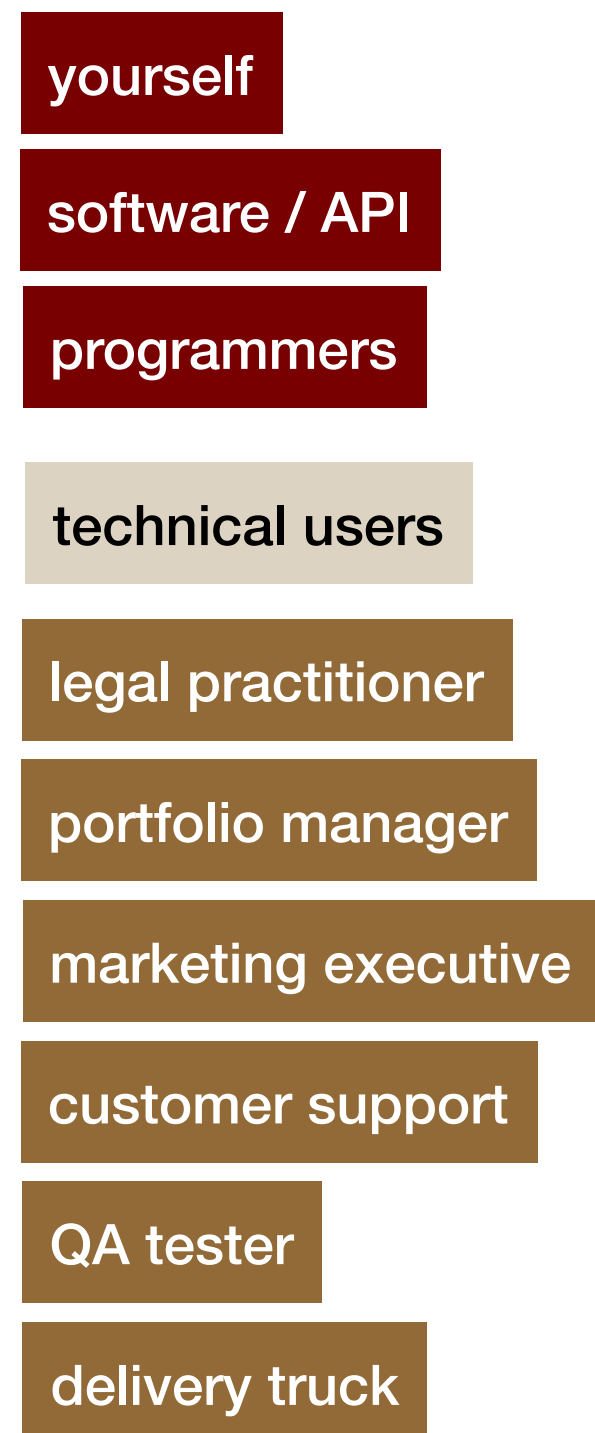
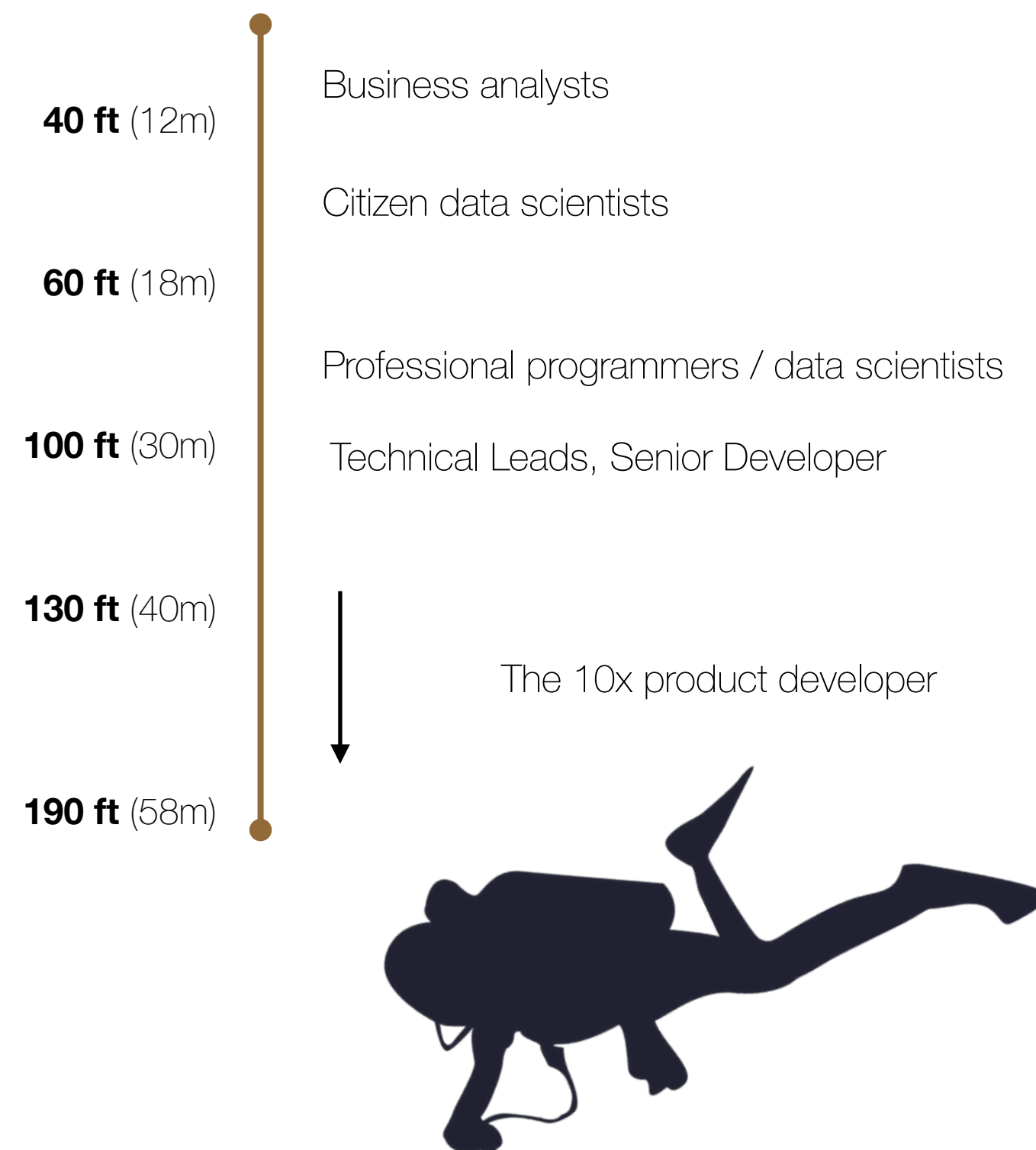
Recreational divers dive for their own leisure. Many coders in the “data science” domain have never had a customer because the only person reading and using that code is themselves.



Customers



Customers



One way to maximize your impact is to start writing code as if they're going to be used by more than just yourself.

One way to get customers is to learn how to ship your code and take it out of your code notebook. **Nobody pays for the code you wrote in that notebook.**

You get customers by shipping code. One way to start shipping code is to stop putting yourself in arbitrary brackets (eg. I'm a UX developer therefore...; I'm a backend engineer and doing front-end isn't my...; I need a database administrator to help me because I'm not ... etc). **Stop it. Ship your code.**


Shipping

It begins with **empathy** and it ends with **pride**. Taking pride in the work you do. Taking pride in the ownership. Taking pride in having an audience — your customers. **You need to have pride.**



Empathy will drive you to create user experience that is customer-centric. It instills humility by putting your customers' needs before your own. It means trading your convenience for your customer's. If that convenience means a python package, you will ship that package. If it means creating a seamless integration into their back end, you will ship that API service. If it means creating a responsive web dashboard using modern front end technologies, you will ship a dashboard. When you don't how, you will put yourself through inconvenience to learn because customer-centricity instills humility. **You need to have humility.**

Multidisciplinary

- 
1. It's harder to be the top 1% in anything than to be the top 10% in three or more things.
 2. The diminishing returns from being top 10% in any particular domain to top 1% makes this highly unattractive in completely rational terms (folks who pursue mastery into the top 1% are not doing it out of reasons). **It almost always is more rewarding and rational to acknowledge the law of diminishing returns.**
 3. Still hyper-specialize if you want to; But understand that you are deliberately handicapping your path to becoming a 10x product developer.
 4. You will have the most success if you adopt a t-shaped development path with as much emphasis on breadth (latitude) as on depth.

Multidisciplinary

It is not the only way to ship software. But multidisciplinary helps you architect solutions faster by having a full grasp of the complexity and all the moving components of a project. Here’s an example of components within the development of a web application:

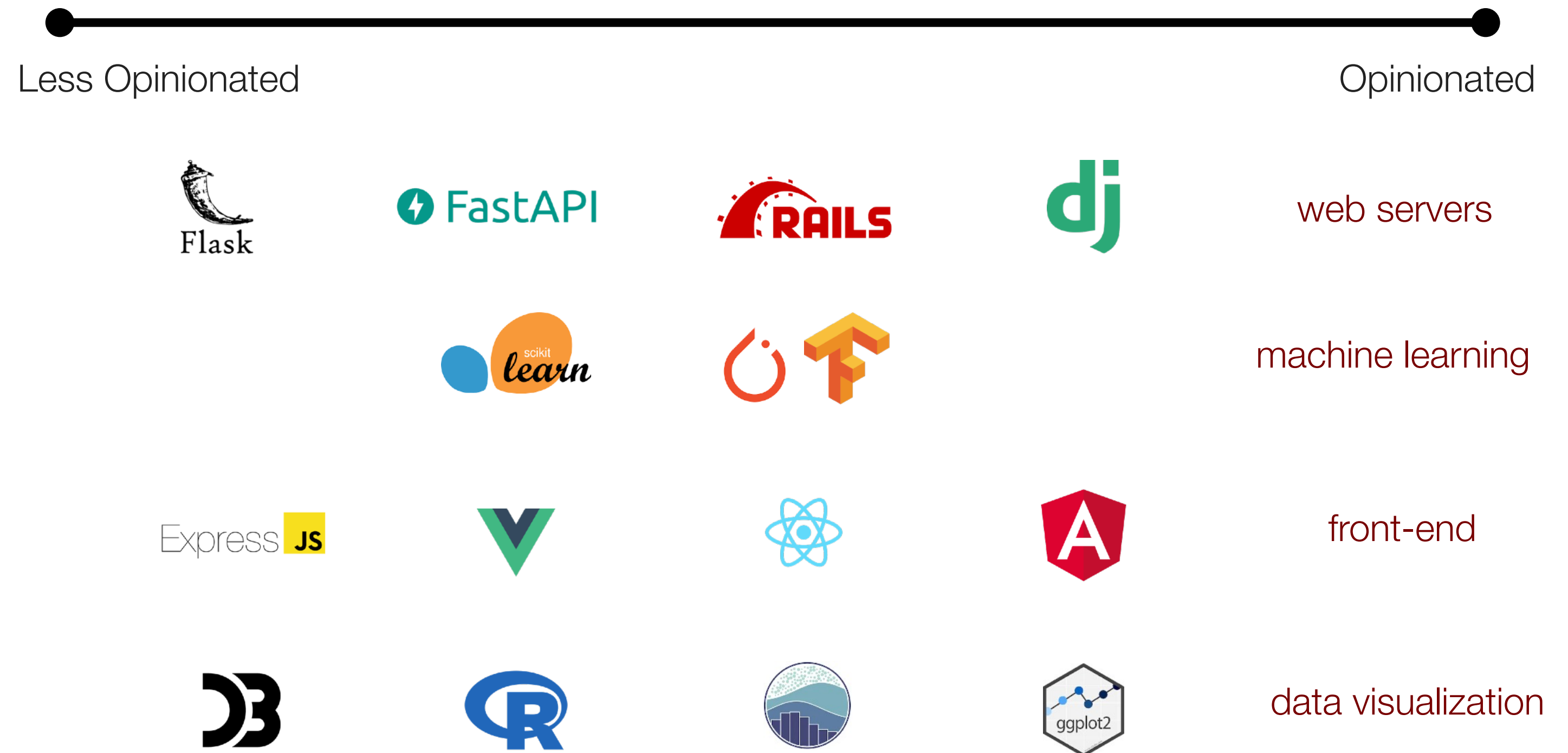
Model	View	Controller	Engineering
Database Choice	Front End (HTML + CSS)	Visualization logic	Version Control
Entity Relational Model	Interaction (jQuery, JavaScript)	Cleansing logic	Test suite
Serializers	CSS Framework (Bootstrap)	Filtering logic	Automation (Github Action)
Unstructured Data	Analytics Snippet (Google Analytics)	Authentication logic	Database Migration
Permission levels	Cookies	Interaction logic	Virtual Machine / Containers
RESTful Service	Administrative Area (Admin view)	Routing logic	Continuous Deployment
	Responsive / Mobile View	Utilities logic	Environment Variables
	Front-end Framework (React, Vue)	Machine Learning logic	Storage and Assets
	API Documentation View	Form logic	JWT Authentication
	Template Inheritance	Permission logic	Task Broker
			Caching System
			Payment Gateway

Examples



Deliberation

While you construct your latitude, there are established blueprints that will serve you well. For example, getting well-versed in at least one of each row in the illustration will allow you to quickly scaffold and ship a web application.



Hammer

I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail (Law of the instrument).

This is an anti-pattern for the 10x developer who cares for his customer. It is either apathetic (you know it's the wrong solution for the problem, but this is the only solution you know) or you're suffering from tunnel vision (you think it is the right solution because it is the only solution you know). Solutions you implement will be sub-standard, and you're trading your customer's convenience for yours.



Supertype

We want to create a breeding ground for multidisciplinary, high-latitude product developers. Product developers that understood what it takes to ship a product, that don't believe in arbitrary brackets, that believes in the premise of the 10x* product developer.

It's a **development** program, first and foremost. Not a for-profit program. You will be put on intensive learning paths and do most of the learning on-the-job, while working on commercial projects. Depending on your contribution, you will participate in any revenue sharing split from projects you work on.

***Disclaimer:** 10x is probably a horrible name. It's probably also not intended to be understood literally (as in, it never intended to mean 10 literally), but the premise is sound.

Culture

How do we know if we've succeeded?

- 1. When shipping products happen frequently and earnestly**
- 2. When client empathy — not convenience — guides programming decisions**
- 3. When our natural inclination is on lateral expansion**

When these become the defining characteristics of our team members, the Latitude thesis will have served its purpose.

Byproduct

The byproduct of a multidisciplinary software developer is a widely branched-out network of knowledge nodes. The more interconnected and the wider they branch out, the more nutritions you're able to absorb, and you'll learn up to 10x faster. A developer with competency in 3 programming languages will be a faster learner of the 4th language compared to a one-language programmer. This is also true for human language.



Byproduct

As an example, a multidisciplinary, high-latitude developer can go from 0 to 80 when first learning about blockchain in perhaps one-tenth the time it takes the average reader. She would be able to draw parallels to Merkle trees (1979), a computer science concept that is also the fundamental underlying data structure used in popular version control systems, such as Git. She is able to draw parallels to what is a commit chain in Git, a tool she used on a daily basis. She understood **on first principle basis** what it means that the blockchain ledger is an immutable history — because so is Git. **The power of a widely branched-out knowledge nodes you acquire over your career as a high-latitude developer pays very handsome dividends, over a lifetime.**



Objections

- 1. Jack of all trades, master of none:** This is the kind of zero-sum, binary mode thinking that is circulated everywhere and has made an impression on many people. The implication is that one must either get very good at one thing, or very average in three. For reasons I explained in the [Multidisciplinary page](#) (law of diminishing returns), getting to a 8/10 in three areas will probably take about the same amount of effort and time to get to a 9/10 in one area. If you pick these three things really well ([highly complementary skills](#)), you will consistently be able to charge more, ship faster, and be a more valuable engineer on just about any team.
- 2. I want to do programming recreationally:** You should program as much as you like. Most technical divers do most of their dives in open water leisurely between occupational dives. A high latitude product developer acquires a very specific set of skills following a [T-shaped distribution](#) — this doesn't diminish the developer's appetite for recreational programming.

Actions

1. Talk to your unit leader, or mentor, to devise a “destination” for yourself.
2. Talk to your unit leader, or mentor, to devise a learning roadmap to arrive at that destination.
3. Work the roadmap into your daily routines and spend time on it everyday. You should have weekly milestones to know that you’re on the right path.

Your learning roadmap has to be deliberate and synergistic, and because it's not always obvious what the better choice is, I've laid down a few rules that can help guide your deliberation. These are not definitive, but they are very sound guiding principles. For example, if you are great with machine learning, and you have basic familiarity with Flask, then following Rule1: $1+1 > 2$, you will get better mileage from adding API development and documentation knowledge to the mix compared to learning, say, Tableau visualization.

Rules

- 1. 1+1 > 2:** The whole is greater than the sum of its parts. The addition of “Django” skillset may be highly synergistic to one developer, but have the opposite effect on another developer
- 2. Momentum:** If you are doing a lot of work with deploying models on the web, learning JavaScript (ES6) allows you to immediately improve your productivity and 2x your output by the end of the month. Being able to immediately deploy the newly acquired skills into your skill set is psychological rewarding and creates momentum.
- 3. Repetition:** Pick skills that you will naturally have to do a lot of due to the technology stack that you are developing on. For example, if you’re doing a lot of data analysis work in R, picking dplyr and tidyverse are very safe bets since they account for a large part of your billable hours. Doing more of something makes you even better at it, and mitigate the risk of forgetting the skills.
- 4. Foundational:** There are certain skills so foundational that it cuts through the entire skill stack, and the entire T latitudinal and longitudinal ranges. Python’s OOP concept is one, familiarity with web development may be another, familiarity with the command line (and Linux) is also a valid example; investing in communication skills (and English) make you a more valuable engineer regardless of project scope and size.

Disagree? Tell me I'm wrong on:

- **Facebook** (facebook.com/onlyphantom)
- **GitHub** (github.com/onlyphantom)
- **LinkedIn** (linkedin.com/in/chansamuel/)

Created by:



Samuel chan
@onlyphantom

Created for:



Supertype
<https://supertype.ai>