FAIRCHILD LABS

I put my money on the longshots
All my ballers that's born to clock
Know I'ma be on top whether I perform or not
I went from lukewarm to hot, sleepin' on futons and cots
To King size, Dream machines, the green 5's
I've seen pies, let the thing between my eyes analyze life's ills
Then I put it down type braille
I'm tight grill with the phony, rappers y'all might feel we homies
I'm like still, y'all don't know me
I'm tight grill when my situation ain't improvin'
I'm tryna murder everything movin'
Feel me?

the monopoly of flow

LAB ONE - APPLIED RECURSIVE INTEGRATION

Recursive Integration

Recursive Integration is my engineering methodology. A methodology is an approach to solving a problem. An engineer's job is solving problems with data. My boss once asked me to teach engineers how to debug. I learned by working in lab, on the job. I never thought about my work as methodology. I had a mentor named Steve Fairchild, who was the best engineer I've ever worked with. My methodology is based on what I learned from him.

James Harden said, "*I am a system*," in the context of his job, playing basketball. A system is an implementation of a methodology. His system solves bucket problems. Put your buckets problem in Harden's lab; he'll use his methodology to optimize his system to get you more buckets.

James Harden is the greatest player of his generation because every move he makes is a complex calculation, based on data gathered from all the time he's been in the gym playing and watching basketball. His brain has accumulated petabytes of data about basketball. James has been in the gym hooping for more hours than most people spend sleeping – since he was a kid. His brain is a data accumulation machine of basketball knowledge.

I am good at debugging because I've debugged a lot of problems. James Harden is a basketball system because he's debugged a lot of basketball problems. I've spent most of my career in the lab. James has spent most of life in the gym.

The power of recursive integration is the value it creates for the future. I'm a good debugger because when I started working in a lab in the 1990s, it was the dawn of the internet age. I used search engines to find how others solved similar problems. When Steve Fairchild was that age, he had to either find a book, ask another engineer, or he just kept coding until he figured it out.

Michael Jordan and Hakeem Olajuwon did not have access to taped basketball games as children. If they saw basketball live on TV, it was rare, like the playoffs. They might have watched a local team play, but it wasn't every day. Even as top college players, they couldn't watch a breakdown while they were eating breakfast in their dorm. They had to wake up early, wait for the tech guy to rewind tapes, watch a few sets before they had to go to class. In the NBA in the 80s and 90s, they had access to tapes of other NBA games – but it was limited by how many VHS tapes the video guy could cram into his luggage. Compare that with today.

James Harden is highly leveraged by past basketball genius and technology.

Would James Harden have developed his system without having seen Kobe and Jordan play? James is tall, strong, athletic, extremely intelligent and left-handed; These traits give him an edge in most sports. He would have been a great athlete; but not the unbeatable system we know today. The foundation of James Harden's system is past basketball genius. Technology allows this genius, as well as Harden's own, to be appreciated by any human with a screen.

Today, you can see every basketball game, at any level, from any country, on your phone, wherever you are and whenever you want. Does this mean that there will be better players than James Harden in the future? Certainly. All the best guards in the NBA already copy his game; none of them have the southpaw hardware James has been training for thirty years. The James Harden system works so well because it was engineered for Harden's custom hardware.

Can a kid today become the next James Harden? If you're willing to put in the work and spend most of your life in the gym, it is possible. James must train his body relentlessly daily to make it perform at the level his system demands. In athletics, there is no getting around putting in the work.

In engineering, we can train while we sleep. We can let the machines do all the hard work, missing all the shots and running drills in the gym, while we do the fun stuff, like working out new plays and scrimmaging with teammates. The better we get at this, the faster we'll be able to solve the most challenging problems. This is the power of applied recursion. Even huge problems, like transportation in Houston, can be solved through recursive integration.

Leverage, Recursion, Integration and Black Boxes

Leverage is the primary force of engineering. Silicon chips are the highest leverage machines man has ever created. Other tools help us exert physical leverage. Digital tools allow us to leverage our minds. Recursion is applying leverage upon itself, accelerating change. Integration is simply breaking things down and putting the pieces together. Black Boxes are machines where you know the expected inputs and outputs, but not necessarily how things work inside the box.

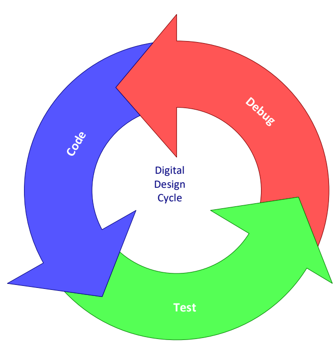
Systems

In James Harden's system, his brain is the software, and his body is the hardware. His system includes many other components; coaches, teammates, trainers, doctors, nutritionists, friends and family, are part of his system.

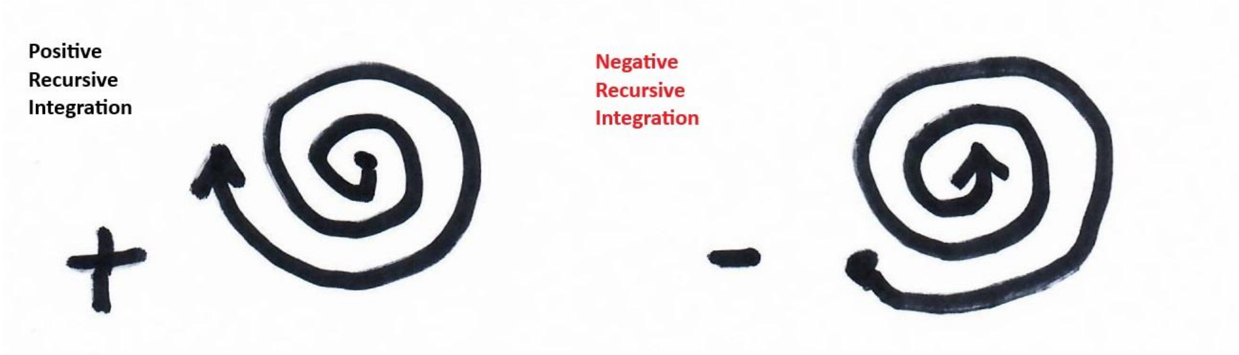
A transportation system includes anything that helps move people around. Almost everybody has the most basic system, legs and feet. In the modern world, all engineering is system engineering.

Design Cycle – Design, Debug, Test

Design, Test, Debug. That is the design cycle. In basketball, you set up a shot (design), shoot it (test), and if you miss you adjust and try again (debug). We learn more about solving a problem by the shots we miss (failed test) than the shots we make. Failure is key to success in both sports and engineering. Each time we go through the design cycle loop, we call it an iteration. The faster we can iterate, the quicker we can resolve the cycle into a solution, no matter the size of the problem.



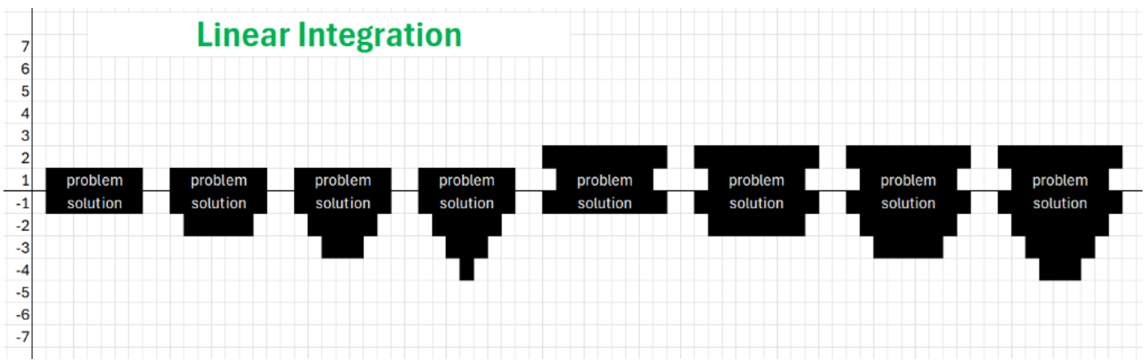
Debug is a word engineers use to mean, "figure out what is broke." In the 40s and 50s, computers were huge rooms with thousands of miles of wire. When a program failed, technicians would comb through all the components looking for insects that had shorted a circuit. From the beginning, smart engineers blamed the bugs even when it was their bad code that was at fault.



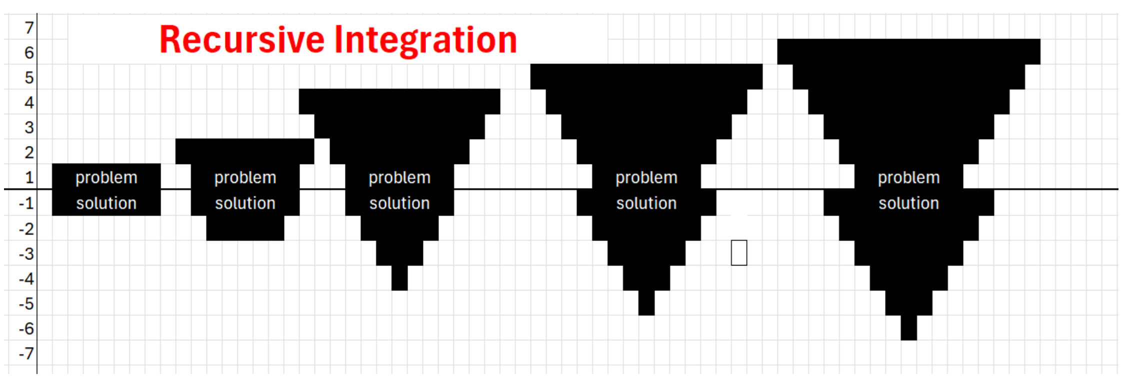
We use positive and negative in the mathematical sense, not in the good or bad sense. Negative recursive integration is making a solution better. In electronics, this means making something smaller, faster and more efficient. Positive recursive integration is expanding the scope of the solution, solving more or different problems with your system.

Applied Recursive Integration

System architecture usually focuses on positive recursive integration, and systems design usually focuses on negative recursive integration. In that case, we have a series of innovation steps that looks like this.



Linear Integration focuses on either negative or positive integration with each integration.



Recursive Integration is working on both the problem and solution at the same time, leveraging what we learn each iteration. We can solve bigger and more complex problems by constantly accelerating each iteration in both directions.

Lab One

Build a dash-cam system for ride share drivers.

Lab One: https://github.com/fairchildlabs/Lab1

Recursive Learning

We cover the lab material as fast as our skills allow, the team working together to move to the next level. We expect the first class to teach the next class, and to get their teams through the lesson in shorter amounts of time. The time that is reduced is the rate of recursion. This is recursive integration applied to an education system.

Key Skills Covered

- Computer Hardware
- SoC (System on a Chip)
- Linux
- Digital Cameras
- Data Storage
- SSH
- Github/Git
- Python
- Field Testing
- Problem Characterization
- Data Analysis
- Scripting/Shell
- C/Make

Lab One Validation Criteria

- Beta-test working camera unit
- Questions: Why are we building a dash-cam system? What is the value of video data? How can we use this system to solve a bigger problem?

Lab One Exam

Make an engineering Mount Rushmore. Create a list of the 4 greatest engineering achievements in the 100 years before you were born.

Example: I was born in 1970.

1. 1969 – NASA Apollo – Man on the Moon
2. 1945 – Manhattan Project
3. 1887 - Electrification (Nikola Tesla)
4. 1914 – Panama Canal

Contact Information:

Brandon Awbrey
scuzzydude@hotmail.com
(713) 305-5620

You

What is the average cars per household in the Houston Metropolitan area? How many households are in there in the Houston metropolitan area ? How many ride share drivers are there in the Houston Metropolitan area ? How many automobiles are in the Houston metropolitan area ?

Copilot

Here's the information you requested:

1. **Average Cars per Household**: The average car ownership in the Houston metropolitan area is about **2 cars per household**.

2. **Number of Households**: There are approximately **2.5 million households** in the Houston-The Woodlands-Sugar Land metropolitan area.

3. **Rideshare Drivers**: While specific numbers for rideshare drivers in the Houston metropolitan area aren't readily available, it's known that rideshare services like Uber and Lyft have a significant presence in the area. The number of drivers can fluctuate based on demand and other factors.

4. **Total Automobiles**: Estimating the total number of automobiles in the Houston metropolitan area involves considering the average number of cars per household and the total number of households. With 2 cars per household and 2.5 million households, there would be approximately **5 million automobiles** in the area.

Is there anything else you'd like to know?

You

Estimate the number of ride share driver in the houston metropolitan area and use that number to divide by 2.5 million households.

Copilot

Estimating the number of rideshare drivers in the Houston metropolitan area can be challenging due to the fluctuating nature of the gig economy. However, based on various reports and industry estimates, there are approximately **50,000 rideshare drivers** in the Houston area.

To find the ratio of rideshare drivers to households:

$$ \text{Ratio} = \frac{\text{Number of Rideshare Drivers}}{\text{Number of Households}} = \frac{50,000}{2,500,000} = 0.02 $$

This means there is approximately **1 rideshare driver for every 50 households** in the Houston metropolitan area.

Is there anything else you'd like to explore?

You

using shared memory to share a C integer with python

Copilot

You can share a C integer with Python using shared memory. Here's a concise example using the `multiprocessing` module in Python:

1. **C Code**: Create a shared memory segment and write an integer to it.
2. **Python Code**: Access the shared memory segment and read the integer.

Step-by-Step Guide

C Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main() {
    int segment_id;
    int *shared_memory;
    const int size = sizeof(int);

    // Allocate a shared memory segment
    segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);

    // Attach the shared memory segment
    shared_memory = (int *)shmat(segment_id, NULL, 0);

    // Write an integer to the shared memory segment
    *shared_memory = 42;

    printf("Shared memory segment ID: %d\n", segment_id);
    printf("Shared memory value: %d\n", *shared_memory);

    // Detach the shared memory segment
    shmdt(shared_memory);

    return 0;
}
```

Python Code

```python
import ctypes
import os
import sys
from multiprocessing import shared_memory

# Get the segment ID from the C program output
segment_id = int(input("Enter the shared memory segment ID: "))

# Attach to the existing shared memory segment
shm = shared_memory.SharedMemory(name=str(segment_id))

# Create a ctypes integer from the shared memory buffer
shared_int = ctypes.c_int.from_buffer(shm.buf)

print("Shared memory value:", shared_int.value)
```

```python
# Clean up
shm.close()
shm.unlink()
```

Explanation

1. **C Program**:

   - Allocates a shared memory segment.
   - Writes an integer value to the shared memory.
   - Prints the segment ID and the value.

2. **Python Program**:

   - Attaches to the shared memory segment using the segment ID.
   - Reads the integer value from the shared memory.
   - Cleans up by closing and unlinking the shared memory.

This approach allows you to share an integer between a C program and a Python script using shared memory. If you need more detailed information, you can refer to the Python documentation on shared memory.

REDS

Recursive Engineering Design System

Purpose

REDS is an industry sponsored educational framework, with a common goal to evolve a recursive engineering design system with a common abstraction domain. The initial feasibility project will focus on digital systems design.

Implementation

The REDS framework consists of three main programs for students. These programs will be run by students, with oversight from local educators and industry professionals.

MSC – Middle School Camp

A summer camp to teach students the primary tools of a STEM discipline. In the digital track, students will learn how to code.

HWS – High School Work Program

An after-school and weekend program, where students will develop lab skills by working on actual systems problems for sponsor companies. Students will develop the software needed for the REDS through direct recursion on the problems they encounter.

UML – University Methodology Lab

The University Lab will formally validate the tools and processes developed by the HWS, and produce academic research from the data generated by the other programs.

Funding

The long-term goal is for the programs to be self-sustaining. The HWS will generate income by performing valuable work for industry. It also serves as a talent development pipeline for STEM employers.

In the feasibility stage, REDS will seek sponsorship from local communities and industry.

Contact Information:

Brandon Awbrey
713-305-5620
Brandon.awbrey@microchip.com

Common Abstraction Domain

Aboard *USS Texas*

"Showtime!" exclaimed Captain Graybeard.

"Ready, Captain," said Ensign Nguyen. "We've got stable current on three buses."

"Attitude?"

"PJ verified Vesta gravity-assist with a Mars return in 405 hours, full burn. Reaction mass margin of 8%," reported the navigator, Ensign Reyes.

"Excellent! Comms?"

"Comms offline sir," XO Marjanović reported. "We need an EVA for repair. We still have entanglement with SOLSTORM.[1] We're backchannelling base telemetry, but no acknowledge from Earth yet."

"Suit up!"

"Captain," Ensign Nguyen said, "There's a problem. We can't do an EVA."

"I am the system for solving problems, Winnie. Hit me."

"We're at 325% reaction power," Ensign Nguyen reported.

"We're cookin', what's the problem?"

"That *is* the problem, Captain."

# # #

The Texas had left Phobos Station on a hard burn, trying to rescue the freighter *Stark*. Enroute to Mars, *Stark* had begun a course correction burn, and lost communication, due to a solar flare at the margins of SOLSTORM detection. During the storm, *Stark* tried to maneuver but continued to burn six minutes too long at an attitude that sent them on an orbit into the outer reaches of the solar system. *Stark's* SOLSTORM telemetry indicated that crew was alive in the hardened-life pods, which had not been jettisoned.

Texas burned hard, but midway received its own SOLSTORM warning, giving them just four minutes to adjust to a lie-flat attitude. The lie-flat maneuver had never been attempted during an active burn.

The lie-flat maneuver put the massive water tanks that held the reaction mass in between the crew compartment and the sun, shielding the crew from the radioactive burst. The maneuver was normally handled by the AI with the main engines shut down. *Texas* could not halt the burn and still intercept *Stark*. If *Texas* couldn't catch her, *Stark'*s crew and passengers would be lost. *Texas* was the only ship in range with the raw delta-V to catch Stark. *Stark's* present orbit would return to the inner-Solar system in 486 years.

PJ[2], the ship's AI, refused to execute the lie-flat maneuver under burn. While it could use the thrusters to adjust to the correct attitude, under burn there would be too much oscillation from sloshing in reaction mass tanks to damper before the surge hit. In addition, their intercept vector would require them to perform a long braking burn to return to Mars. PJ was certain it could protect the crew from the burst, it couldn't avoid risk to the fusion engines and several other exposed subsystems. Since the Texas

[1] SOLSTORM – Early warning network for Solar Storms that uses entangled quantum communication to give spaceships a few minutes of warning to protect crew and equipment from damage.

[2] *Texas's* AI had originally been named Calvin, after an original rocket scientist. When Captain Graybeard first stepped aboard, his command to Calvin was, *"Rename yourself PJ and go stand in the corner and watch my six, until I have some important work for you to do."*

was already travelling fast enough to escape the solar system, losing those systems would put the crew at risk, as they would have no way to slow down and return to the inner solar system.

PJ said, *"Sorry Captain, the maneuver during burn violates Asimov's first and second laws. I can't do it."*

"Little Baby Johnny, you ready to check in?"

"Yes, Captain," said XO Marjanović, who like all the Martians, towered over their captain. "Engineers, clutch time! Let's go!"

XO Marjanović and ensign Nguyen joined teenage ensigns Reyes, Patel, and Liu at the REDS console. REDS was a group of five workstations, all facing a central point. Each workstation looked like an arcade gaming chair, with a group of joystick controls on each armrest, and folded off to the side, a standard keyboard. The Martian teens had neural implants, so they used their minds to control the input to the system. The manual controls had been a compromise for chipless Earthlings or for those that had objections to cybernetic technology. Through their neural links, all the engineers shared a common display of a detailed blueprint of *Texas*. An engineer could focus on each subsystem, and delve down into the hierarchy of the components, revealing the source code or wiring schematic of every part of the ship. They could easily change code and have the AI simulate the results, providing a near instant test and risk analysis of any change they made. Five engineers could control separate sub-systems, or they could all shift focus and collaborate on a single problem together. The rest of the crew could not see the visualization directly, but panels around the bridge would render the models on flat screens.

Captain Graybeard observed on his command screen. It took the squad about twenty seconds to program the vectors for the attitude adjustment.

"Ready, thrusters!" Marjanović called.

"Ready, power!" Nguyen called.

"Ready, nav-comm!" Reyes called.

"Ready, life-support!" Patel called.

"Ready, top-level!" Liu called out.

Captain Graybeard cast a dramatic gaze into the central bridge camera.

"All right baby Martians, get ready, because it's about to get heavy up in here."

The captain looks off to his left. There was no one there, the AI was omnipresent in a spaceship. The crew had used a video panel to show an anthropomorphism of PJ so their ancient captain could have someplace to focus when he talked to the AI. At first, it had been endearing to the crew, who didn't have to vocalize their thoughts to speak to the AI. After a few weeks, it got silly.

"PJ, you're on the bench, Captain in command! Put all systems under crew control, override authorization Mamba-Magic-Skyhook-Diesel-Metta-Jackson, now!"

Graybeard turned back towards the forward bridge camera.

"I am the system! Let's get cookin!"

He makes a hand motion like he's stirring a pot of soup.

The course correction went off without a hitch, but the sloshing in the reaction mass tanks produced a large shimmy. When the wave of radiation hit the ship, multiple systems shut down and alarms went off. They lost one of four engines completely, and two of them produced oscillating surges of current on the power buses. They lost one power bus, but the team dynamically rerouted and were able to keep the reaction engines firing long enough to ensure an intercept trajectory with the *Stark*. They shut down reaction power to the two malfunctioning fusion engines and began repairs.

Meanwhile, Captain Graybeard ordered the shuttle crew to rescue the crew of *Stark*. The shuttle docked with *Stark* and recovered the four crew and six passengers. They did not have time to offload any of the valuable supplies bound for Mars. The shuttle team attached a solar sail buoy to *Stark*, which would bring it back to Mars orbit in nine years.

The Martian engineers continued to debug the failed fusion systems from REDS. Their single working engine could not produce enough power to brake and return to Mars. PJ informed them that their best orbit for Mars return on one engine was a Jupiter gravity-assist, but that would take six years. With the additional passengers, they only had enough food and consumables for 37 months. They'd saved *Stark* but put themselves at risk, exactly as PJ had warned.

Luckily, the propulsion system had been designed by the crew, they knew every component and had written the control code themselves. While the shuttle rescued *Stark*, XO Marjanović and his squad of teenage engineers were able to hotwire and reprogram the power system and get three of the reaction engines back online. If they could keep the three engines alive, they'd could slingshot around the asteroid Vesta, and still return to Mars in a little over two weeks.

# # #

"Captain," said Ensign Nguyen, "We're cooking, because we're too hot. The radiators can only dissipate 200% reaction power. We're dumping excess heat into the reaction mass, but the water is already at 93 degrees. Once the water starts to boil…"

Captain Graybeard said, "*Texas* was designed for full power burns."

"The shakedown was only for solar cruises sir," Ensign Nguyen said. "We've never run over 200%. The multiple engines are for redundancy, not a hard burn. We're also generating a lot of waste heat with the coupling feedback loop."

Texas had four fusion engines, but one was completely scrammed and had been used for spare parts in the repair. The hotwiring job on the reaction engines had the drawback that two of the three working fusion engines would only run at 100%, and the third, undamaged engine had to be running at higher pressure to provide a boost to the two repaired engines. They could only dial the single working engine down to 105% or risk losing the other two. They needed a 300% power burn to brake into a return slingshot trajectory, or risk being hurled into the outer solar system.

Graybeard, an Earthling, did not have a chip embedded in his skull. He used primitive hand gestures to bring up a screen of the ship's schematics, and quickly focused on the radiator assembly. Two knobby flanges of pipes and vanes protruded from *Texas* at its stern, just forward of the massive reaction mass tanks.

"There's a telescoping extension here," the captain said. "We just need to extend it. That should increase surface area and put the vanes out 94 meters, away from the heat radiating from the tanks."

"That's a problem, sir. Our only controls for the radiator are the flow rates valves."

"There's a push motor here on the schematic. It's wired to this GPGCU.[3] We just need to program it to extend."

"That's the problem. The Standard C-API wrapper around radiator control only gives us access to the flow values. There's no API for the extension motor."

"There's a manual crank. EVA!"

"It's 112 degrees on the surface near the tanks, sir," Ensign Liu said. "An EVA would be suicide."

"It's a software problem, hack a fix!"

The crew was silent for a moment.

"Sir," Ensign Nguyen said, with some hesitation. "The entire software stack for the radiator sub-system is written in AGGIE."

"AGGIE? PJ, what the hell is AGGIE? In context."

"*AGGIE is a 5-bit encoding scheme conceived on December 6, 1992 by a Texas A&M graduate students for a NASA code contest. Using only capital letters and excluding the letters "B" and "V". AGGIE*

[3] General Purpose General Compute Unit. Digital Logic lego bricks Martian engineers used to build larger systems.

has no symbol for "3" or zero, using "E" or "O" instead. AGGIE-SCRIPT is a programming language derived from AGGIE, a language unique in that it had no punctation or operators, nor any end of line characters. On November 10th, 2037, the 25th anniversary of A&M's victory over Alabama, provost DeAndre Jordan declared AGGIE the official file format and language of the school. Texas A&M spent over $550 million converting…

Graybeard interrupted the AI, "How much did bits cost way back in 1992 – I was just a baby."

"In 1992, a one gigabyte hard drive cost approximately $2000. On a per bit basis, this would equate to .00000025 dollars, or approximately 4 million bits per penny."

XO Marjanović said, "Captain, I think you're missing the point. The important factor isn't the cost of data bits, but the date that AGGIE was invented."

"Huh. Baby Johnny, you're telling me that—"

"Saint Johnny's birthday," said Ensign Nguyen, forcefully, irritated.

"You mean the entire university adopted the stupidest encoding scheme ever invented just because it was conceived on the day Johnny Manziel was born?"

"You know what they say about Aggies; From the outside looking in you can't understand it, and from the inside looking out you can't explain it," said Ensign Reyes.

"There must be somebody here who can code in AGGIE?"

The Martian teenagers exchanged bashful glances. They were first generation Martians, and of course, knew the real history of Aggies on Mars.

After sending the first seven Challenger missions to Mars and discovering water and all other critical sources of minerals needed for a self-sustaining colony, Mars patron El Jefe had picked his young crew of colonizers wisely. He picked the brightest and hardest working people, choosing a pair of students from all the major Texas public universities, including Texas A&M. El Jefe was the king of the publicity stunt. When it came to A&M, there really was only one choice for the male candidate, an Aggie who had dropped out of his degree program in 2014, but decades later, had decided to return to school. The other A&M candidate was the XO's mother, a Nobel prize winning nuclear physics PhD from Serbia, who had chosen A&M over UH and UT because she didn't like big cities.

The cover story of what happened during those glorious first two years of Mars colonization was now a fairy tale that had been made into a streaming series for Earth children. How, at great risk to his own health, this brave Aggie had risked his life to save the colony during a series of storms, mishaps, and malfunctions, and finally, after establishing order and training his robot army of twelve into a super team, Scobee City was ready for the second wave of colonist, which arrived two years later. It was a story of triumph over adversity, and a tragedy as well; due to radiation exposure, the hero had to be evacuated back to Earth orbit for medical treatment.

The Challenger Mars missions had been a much more diverse coalition, in their inclusion of international astronauts, they had failed to include any Aggies on any of the first seven missions. What really happened was much more pathetic. In his zeal to accomplish the most "Mars firsts" by an Aggie, the hero had tricked Scobee City's AI into overriding his radiation exposure limits, so that he was authorized for unlimited surface missions. He became first Aggie to throw a four-hundred-yard pass on Mars. First Aggie to bungee jump into Valles Marineris. First Aggie to survive a Mars sandstorm in a monster Cybertruck. And finally, perhaps most importantly, the first Aggie to be treated for severe radiation sickness on Mars. After returning to Earth orbit, he had gene therapy to repair his DNA.

In orbit he remained. It turns out he'd also tricked the AI into deleting his daily health protocol, which mandated strict exercise regime for the colonists in case they needed to return to Earth. Ten years later, El Jefe asked him to be the first Aggie manager of the first Buc-ee's in space. To this day, voyagers to the outer solar system would stop at Lagrange L2 Buc-ee's to get a selfie with the most famous Aggie as a rite of passage.

The young Martians learned the secret history from their parents, but El Jefe made so much money from the streaming rights, keeping it secret seemed a fair tradeoff. There was one first he didn't accomplish. He wasn't the first Aggie to procreate on Mars. The XO's genome was a source of great mystery, but he was born three years after the most famous male Aggie left the planet. His mother would only say, on choosing his name, *"Since they say Mars kids will likely be very tall, I wanted him to have a tall role-model to look up to. And also, in case he's not so tall, for him to remember, height isn't everything."*

No one on the bridge knew AGGIE.

Ensign Patel piped up, "My mom got drunk with Saint Johnny in a bar in Houston, does that count?"

"That would make me the most qualified," said the captain. "I was in a bunch of different clubs with Johnny. Did I ever tell you about back in the day when I was in the ---"

Ensign Nguyen didn't want to hear the story again, "Captain, the reaction mass is at 95 degrees. We're going to have to vent soon."

Captain turned and faced the bridge screen, stood tall and straight, preened his beard, and then commanded.

"System Time! PJ, write some code to extend the radiator extension trusses to maximum length, and then translate that code into AGGIE, and then inject it into the GPGCU of the radiator. Get cookin' or else we'll all be cooking!"

Half a second later, a message was displayed upon the bridge screen:

Code composed in standard C.

Translating into AGGIE............HALT 0x0000002A!

Red warning lights flashed.

Asimov violation – zeroth law. Please reboot.

"Winnie, what's wrong with PJ?"

"Captain, PJ asserted on error decimal 42. Apparently translating code into AGGIE violates Isaac Asimov's zeroth law."

"I know one, two and three, but what is zero again?"

"*A robot may not injure humanity or, through inaction, allow humanity to come to harm*. Apparently, if a modern quantum computer tried to translate code into AGGIE it would destroy humanity."

"Don't they have AIs in College Station?"

"A&M modeled their AI after the cow brain."

The captain stroked his beard.

"Let me get this straight. We can either get deep fried and return to Mars, or remain on this trajectory, slingshot around Jupiter but die of starvation?"

"We've only got seven minutes before we've got to vent!" the XO reminded.

"Mamba time, baby Martians! Winnie, you're my code ninja. Hack a fix."

# # #

Not that the captain had ever asked, but Ensign Nguyen did not like being called Winnie. Her given name was Drexler, and her call sign had been *Glide*, obviously, before this showboating Sun Devil from Cali showed up to boss her squad of Martians.

The billionaires on Earth had loved the idea of an all-Martian crew. *Texas* was the most technologically advanced ship ever built.

All the power, propulsion and control systems had been designed on Mars by Martian engineers, led by XO Marjanović. The auxiliary systems design had been outsourced to Texas public universities. UT designed life support, UH the Comms, and Tech did the interior design. Wanting to be fair, El Jefe's trustees had awarded the simplest system, the radiator, to Texas A&M. It was basically plumbing.

Texas was supposed to be an inspirational message for Earth children, to inspire them to greater engineering innovation and a quicker path to a green and sustainable Earth.

The natural leader of this crew should have been the XO, who not only was the first human born on Mars, but at eight years old had invented the portable fusion reactor that now powered the green economy on Earth. At nine years old, he invented the gravity boot. This allowed little Martians to skip the hours on the treadmill or centrifuge and just play regular teams sports to develop their muscles. An added side effect, on Earth, the boots allowed people just to glide on an anti-gravity wave whenever they wanted to go, putting the automobile industry out of business. The designs were open-source, free from patents. El Jefe's motto of *Save You Money* put these wonderous inventions in the hands of people rich and poor, all over the world.

Even after Martian innovation had saved Earth from destroying itself, they still couldn't envision giving the keys to a trillion-dollar ship to a teenager. The old fools didn't even really debate it.

Transcript of the International Space Billionaires Committee

Mark Z. – *It's worth a trillion dollars, We can't let a bunch of teenagers run it.*
Queen B. - *What we need is a leader, a systems person, who isn't intimidated by a bunch of young seven footers.*
Steve B. - *I know just the guy! He is a system!*

And the sports metaphors; Hey, captain *Let's get cookin*, read a room. *Mamba Time* and *Hey Winnie, throw some Shaq-Fu at the condenser coil*, didn't really motivate this particular crew. Their parents had primarily come from Space City, Texas, where all the rocket scientists lived. This bearded prima donna shows up just to boss people around and tell the people back on earth, "I am a system." Everybody knew the job should have gone to Johnathan Paul Marjanović. Instead, the XO had to listen to the captain mock him, calling him, *Baby Johnny.* Sure, the XO had been the most famous baby ever born, but that was sixteen Earth years ago.

Ensign Nguyen's personal trove of inspiration wasn't stupid cliches, but simple words to live by. *Believe It, and then get it done*! All Martians, even ones raised outside the rich history and tradition of Scobee City, held one maxim in their heart; *Always Think Big!*

While turn of the millennium billionaires fought each other on social media for the best plan for humanity to colonize space, and tried exotic schemes like crypto or a breathing-tax to fund their ridiculous endeavors, a successful local H-Town businessman took another approach. He went to Vegas, and bet his entire fortune on the Longhorns, Texans, UH Cougars men's and women's basketball, Rockets and Astros to win their respective championships in the same calendar year. The odds were about six billion to one. When Altuve hit the walk-off home run on [redacted] [redacted], [redacted][4] to win the bet, El Jefe became the owner of the entire gambling industry, including most of the land in Nevada and Atlantic City. Because Vegas had laid off action overseas, he became the majority owner of real-estate in Macau, Singapore, and Monte Carlo. El Jefe became the richest human in the history of the world. He immediately auctioned off his land windfall and invested all of it into his own plan to colonize the solar system. Showing what you can do when you put a real Texan in charge, he brought in professionals from all over the world to teach STEM subjects and skills to Houston area school children. Within a few years, he launched the Challenger missions.

[4] Science fiction writers who make side deals with gamblers should not be trusted.

Winnie was jacked into REDS and saw instantly what her four fellow systems engineers were seeing. All three of them were looking at the display of the source code of the firmware for the radiator assembly GPGCU.

A very long line of "text" without punctuation – this was the raw AGGIE-SCRIPT.

ADDTIMESTAMPTO7ANDSETAGGZARATORESULTSETAGGZARCTOZEROMOEFOREACHFLOWALE…

"We're not going to decode this in seven minutes." Nguyen said. "We got to think outside of the box."

"If we had comms…"

"Comms wouldn't help," Nguyen said. "We're nine minutes round-trip to Earth."

"Maybe if there were another Aggie in radio distance."

"Wait a minute," Nguyen brought up a window that connected her to the manifest database.

"Captain! There's an Aggie on board. One of *Stark's* passengers. It looks like they're a vet…."

"A vet, all right," the captain said, looking introspective. "Now we're talking. A vet might be about my age. It would be nice, a grownup to talk to, I'm liking the sound of this. A shorty Aggie vet---"

"Captain," said the XO, "You must NEVER use that word."

"It's okay, Baby Johnny, I'm from Earth. I can say it. It is funny and endearing when I say it."

"Captain," said ensign Liu. "Do you really think it's endearing when you're taller than 99.9% of the people on Earth."

"She's not a veteran, sir," said ensign Nguyen. "She's a veterinarian."

"A what? An animal doctor? On a trip to Mars?"

"Coach Doctor Cheng Xin," Ensign Nguyen explained. "She's an assistant coach of the women's basketball team, and chair of Texas A&M's genetic engineering program."

"Medical Bay! Please send up Doctor Cheng Xin!"

"Funny," said the captain, "Why they send a basketball coach animal doctor up in space?"

PJ answered, *"There is a loophole in Texas law that allows Texas A&M to pay anybody with 'coach' in their title an unlimited amount of money."*

"She got the supermax?"

"Doctor Cheng Xin was the top free agent genetic engineer," said Patel. "They had to make her a coach, the only way the Aggies could outbid the Musk and Bezos people. She's in charge of an Aggie project called Mars mambas."

"Snakes on a spaceship?"

"These snakes were modified for terraforming," said Patel. "These snakes burrow into the regolith and turn the toxic Mars dust into oxygen and water and produce a valuable biological soil compound with their feces. The Mambas are designed to live for hundreds of years and grow to lengths of 400 meters."

"Wait a minute. I'm pretty sure that was a movie, back when I was in the —"

"In Frank Herbert's Dune series, it was worms," said Patel. "The Aggies changed it to snakes to avoid royalties. They were over budget from all the coach's salaries."

The bridge door opened. A small woman entered.

"***La Puma Rojo***," exclaimed the entire crew, including the captain.

They all recognized her. The 150cm point card who had led the University of Houston Women's basketball team the NCAA championship. Her seven clutch three pointers in the final three minutes of the game had led to victory, the second rung in El Jefe's bet that had funded their colony. Unfortunately, like most college athletes who don't turn professional, the world had lost track of Cheng Xin. After graduating from UH with a biology degree, she'd continued her studies and become a Veterinarian.

She'd gone on to invent a gene mutation process that led to specialized therapies for all sorts of diseases on both animals and humans.

"I'm Cheng Xin," she said. "Hey, I recognize you! You're the guy Steph used to splash all over—"

"No need to rehash ancient history in front of the children," said the captain.

"Who you calling shorty? We can hear every word you say on the bridge in the medical bay."

"Sorry about that. We got a situation. Can you code in AGGIE?"

Cheng Xin let loose a long, low, satisfying hackle.

"For 8.7 billion dollars a year, you bet I learned AGGIE. Jack me in."

As Cheng Xin approached the REDS station, the captain turned to PJ.

"8.7 billion? That's better than supermax. Who is her agent?"

"*El Jefe*," said PJ.

"She got a bag," said Graybeard.

"8.7 billion is a bag," said PJ.

Graybeard shakes his head, "I shoulda never left Texas."

Reading Notes

- Humans born on Mars will likely be very tall because of the reduced gravity.
- Martian children will become competent systems engineers out of necessity.
- History has proven that humans are most creative when they are young. Training them to use engineering tools at younger ages can lead to fantastic solutions to difficult problems.
- Using tools with a common abstraction domain facilitates rapid and efficient teamwork under pressure.
- Using tools with an uncommon abstraction domain can be dangerous or fatal.
- The author was taught how to code in a NASA summer camp in 1985. 37 years after the camp, he found out that it was funded by Texas A&M and taught by an Aggie PhD.
- When you have a challenging engineering problem, who should you call and what should you say?
    - "Houston, we have a problem."

REDS

The Recursive Engineering Design System (REDS) is a system for designing digital systems. A fully integrated toolset is used to abstract the entire hierarchy of a system, down to the source level of the logic. It is a single tool for design, test and debug of digital systems. REDS is recursive, as designs and architectures evolve, it maintains a common abstraction framework.

It's the holy grail of computer aided design. The fundamentals of digital logic do not change. They are constant in simplicity. The complexity is achieved by scaling AND, OR, NOT. What changes in the abstraction model. REDS will leverage the fact that all modern engineering systems are foundationally digital systems and build a common abstraction model around the core simplicity.

Integrated Development Environments (IDEs) allow rapid development of complex software applications. REDS will attempt to extend this same kind of leverage to system design.

Rather than try to devise commonality from the millions of design flows that exist in digital engineering today, we'll start with a fresh slate. The current generation of students have used software and digital logic their entire lives. It's ludicrous to think that engineers that were born in the time of slide rule could optimize a design system for the present or the future. REDS is tomorrow's system.

Education System (Vertical Axis)

REDS is a work-study program. The goal is to evolve and implement an integrated engineering design system. Sponsored by industry, REDS will develop both the tools and the engineers needed to design advanced sustainable systems for use both here on Earth and beyond. REDS will implement a recursive curriculum, so that the competency of new engineers in industry is advanced at a measurable rate.

REDS begins with a middle-school camp (MSC), where students are taught the fundamentals of digital design and the primary tool to do it - code. After completion, students can complete additional course work and become qualified for the high-school work-study program. The camps will be fully sponsored and look to apply the most local leverage.

The high school work-study program (HWS) is an employment program run by sponsor companies. Student workers perform lab experiments and are driven to advance their skill level directly in measurable and productive ways, with wage increases to match their achievement. The program will be part-time, after school and on weekends. This is a merit-based system measured by demonstrated ability. Students who show leadership participate as mentors for the summer MSC.

The University Methodology Lab (UML) is a partnership between sponsor companies and a local University. The UML develops and validates the tools and methodology to measure a set of engineering problems.

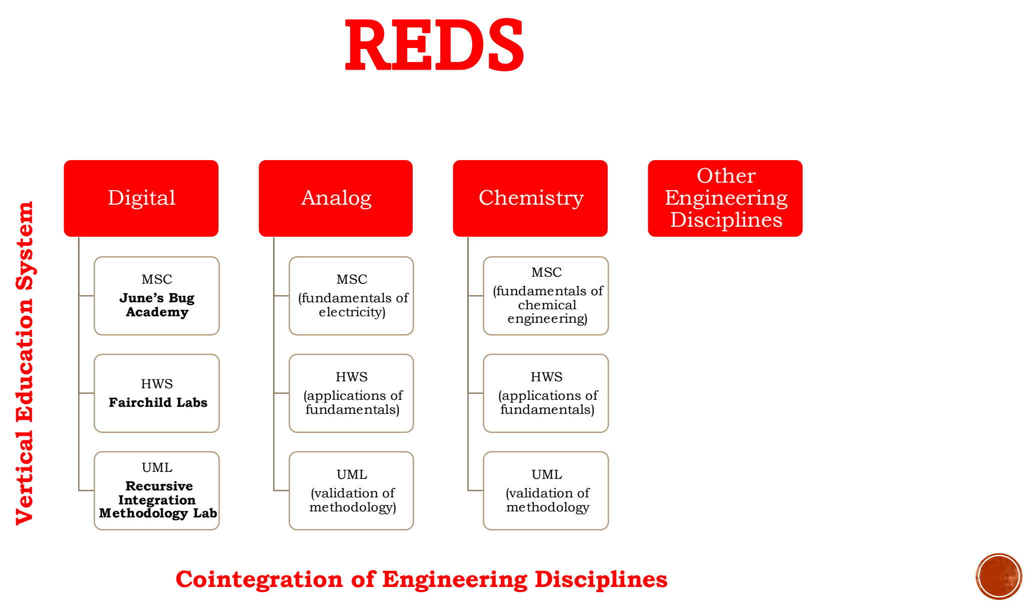
Forced Recursion

At every level, participants are expected to improve and evolve the REDS toolset while using the toolset to optimize systems. One year, students might automate test execution and analysis. The next year, the next batch of students might be expected to automate debug. Each new batch of students should try to make redundant the process work of the last class.

The goal is that engineers entering the profession through the program bring in skills that re-leverage their employers and push their senior engineering peers to higher levels of leverage – or out the door. This prevents stagnation and forces innovation – or failure.

I have not failed. I just found 10,000 ways that won't work.
Thomas Edison

Fail faster is the is mantra of modern engineering. Today, with modern tools and the right approach engineers can fail 10,000 times in the blink of an eye.



Cointegration (Horizontal Axis)

All modern engineering is digital engineering. In all disciplines, engineers depend on software and digital logic to solve problems. If I see the flight engineer break out a slide-rule, I'm getting off the plane. What our predecessor engineers accomplished without digital integration is amazing. We should challenge ourselves to push the boundaries of engineering even further by fully leveraging the tools we've inherited. Moore's Law is virtually free leverage for every profession. For engineers, it is a responsibility.

REDS will establish itself as an educational franchise, bringing together disparate engineering disciplines under the common core of digital integration.

By focusing the common student initiatives around a fully integrated design system, we can develop the engineering talent, tools and systems we need to build a self-sustaining colony beyond Earth. Sustainability is the key engineering challenge in this age and is an absolute requirement for outposts beyond our atmosphere. This is primarily a marketing angle, to get students excited and inspired by STEM careers. If there is an opportunity to integrate with actual space exploration and colonization enterprises, then the students should drive that engagement. The sponsor companies' interest should be sustainable engineering efficiency today, here on Earth.

REDS will partner with engineering leaders in all disciplines, establishing multiple vertical tracks. The MSC core curriculum will be digital logic with sponsor specific labs and projects. HWS programs focus on tactical and relevant work from the sponsors. The UML could be company or industry sponsored.

Constant Integration (Depth Axis)

The third axis of the program is constant integration. A single sponsor may choose to have multiple MSC, HWS or UML for different phases of system design. For instance, a chipmaker may have a complete vertical in Texas that focuses on architecture and validation, while another vertical in China that focuses on design integration and verification. Within a large metro area, a sponsor may have multiple disciplines with multiple local schools and universities.

The horizontal axis focuses on commonality among engineering disciples and systems, the depth axis focuses on integration with a single design process. It's a way for sponsor companies to develop talent through a common framework. Companies cannot realistically expect to source all their new engineering talent from one geography or program, but it does set an internal standard of expectation for hiring as well as a guidepost for other educational institutions that want to refine their target curriculum for industry. The key measurement is the rate of recursion.

Recursive Value Measurement

Following engineers from early education through their professional careers and measuring their skill level with common tools gives both educators and sponsors access to a powerful data set. You cannot judge the result of any educational program for young people deterministically while they are young. You must see the results of application of that knowledge and skills once they are productive adults. What we measure while students are in school is how well they retain things in their short-term memory – primarily, the ability to cram for a test. An engineer's real leverage is how they can apply things broadly. Education provides a foundation, the real measure of value is how that skill set and knowledge in retained, applied, and recursed upon – either through improvement, or by passing their knowledge and experience on to the next generation.

A recursive constantly improving toolset forces an innovation challenge through the engineering ranks of a sponsor. A recursive educational system forces application of that leverage upon the sponsor's management and their educational partners. Long term, if you are improving your workforce's engineering skill level at a measurable rate, you should be able to measure improvement with both business and academic metrics.

Recursive Education Development System

This system also forces recursion upon the education system.

My engineering mentor, Steve Fairchild, taught me some very basic rules about methodology applied to digital systems. The most important was, when any part of digital system changes, you must remeasure all parts of the system again. This is because digital systems design was all about ratios. When ratios change, system behavior changes.

The approach, I call *recursive integration*, makes it easy to debug and understand complex digital systems. This methodology itself almost always leads to simple optimized solutions. "*Recursive integration,*" is really jargon so the boss doesn't have to tell his boss, *"We fail as fast as we can and measure it so we learn to fail faster."*

It's easy to apply recursive integration to human systems problems. Applied in the profession digital engineering work environment, it would work like this; The boss only gets his raise, bonus and equity if the engineer working for him can do the bosses job by the end of the year. Engineering majors don't need to worry, this is purely hypothetical, in the real world, nobody's boss is stupid enough to approve this methodology.

That's why the education system is the way to go. You set a goal for a recursion level. Is it one year? Depends on the subject. In engineering, and other digitally leveraged scientific endeavors, it should be very rapid.

An education system to validate a colonist population for another planet is the perfect example. Why? Their lives and all the lives on the colony depend on it. There is no "Mother Earth System" to provide a backup for human children left alone with just machines to care for them. That's the challenge.

Because digital systems are the primary leverage tool of all engineers, an engineer's leverage at entry level should progress in step with Moore's Law. The three-tiered system provides a checkpointed process with multiple measurement points to ensure that it does.

Recursive Responsibility

Is this a brutal way to look at education? Not all disciplines of education leverage Moore's Law as heavily as system design, but almost all do to some extent. It's a constant factor in every aspect of education.

The challenge of Mars is a good test ground. Every potential Mars explorer and colonist needs to be a highly competent, fully qualified systems engineer. Their lives depend on it.

Brutal or not, this is real. We need simple systems that trained engineers can troubleshoot, repair, or redesign, on the fly, with zero help from the mother planet. Systems will fail, without engineers to fix them, people will die. The time or cost of Earth rescue may exceed what Earth is willing to spend.

Those systems – and the tools they use to design the systems -- should be the responsibility of the people who put their lives at risk. This is the exact same logic that the Mercury astronauts used to convince NASA to put manual controls in the Mercury spacecraft. Without ownership of that responsibility, Gordon Cooper and *Faith 7* would never have been able to return to Earth.

Engineers are responsible for the tools they use.

Recursive Risk Assessment

Can we build systems to explore and colonize Mars? That's a silly question, of course *we can*.

Should we build systems to explore and colonize Mars? At what cost?

The cost of resources, both public and private, is of great concern, both for capitalization of the endeavors, and as a measurable way of factoring the greatest expense, human lives.

Ultimately, good engineering is the optimization of designs for the most efficient systems. This is real green engineering. The metric of value for REDS is not cost efficiency, but resource efficiency, which is the required optimization when resources are limited. On Mars or on Earth, human lives are our most valuable resource. Avoiding waste and conserving resources so that humans can thrive is the optimal solution, no matter what planet you're on.

What are we willing to spend and risk to explore our solar system? The engineers that risk their lives in these endeavors need to make that evaluation, using engineering methodology and risk metrics they develop. Exploration of any kind is an inherently risky endeavor, and it has costs.

Common Abstraction Domains

To understand REDS, we can look at science fiction. When the crew of Star Trek's *Enterprise* encounter any kind of systems problem, be it communications, life support, transporter, weapons, or the warp drive, the chief engineer can go to any workstation, and within a few seconds, instantly locate the problem and usually fix it or work around the failing component. It's a common plot device, saving the average viewer from having to understand the science and engineering behind the systems. Even fictional series set in the present use these kinds of systems. It's common for a super spy to call up a nerdy hacker who can instantly get access to any bank account, every surveillance camera and even into the control room of nuclear power plants, from the same magic software application.

While these are fantastical fictional devices, they aren't far-fetched. When you reduce digital systems down to their simplest components – logical AND, OR, NOT operations -- they can be viewed through a common abstraction. Digital systems are inherently hierarchical, with components connected by data flows. The software abstractions in between are also easily represented with plumbing, and in fact, many complex application use "pipes" for inter-process communication.

Yet in practice, modern digital design is a crowded field of specialization in specific tools. In modern chip design, it's not uncommon to have dozens of engineers within the same organization looking at data with dozens of different tools. These abstractions are useful to each in their own way, but prevent leveraging the commonality. A common abstraction domain is the solution.

First Recursion (Houston)

REDS Franchise

The REDS parent organization is a non-profit that provides guidance, resources, and support to students seeking to develop REDS programs in their local communities. REDS will be a participant run organization, funded by individual donors in the communities where it operates.

Creative Expression Contest

Initially, REDS will promote the concept through writing and creative expression contests. Students will be challenged to consider problems and solutions, both technological and ethical, that need to be solved for a viable space colony. REDS will assemble of panel of writers, educators, and engineers to pick winning entries. Awards will be in the form of cash and scholarships. Winners and their schools will be provided resources, support and starter funding to enable development of REDS programs in their local communities.

Digital Design Education System- Houston

The first implementation will focus on architecture, which begins with proper characterization and modeling of the engineering problem a system is designed to solve.

June's Bug Academy (MSC)

Moto: ***If you wanna ad astra, gotta fail fasta!***

A summer camp for incoming 9th graders.

Mars Theme

The camp will partner with NASA Johnson Space Center, so that students would get a firsthand look at what the complex systems that manage spacecraft today look like, as well as the engineering manpower and specialization that goes into modern space mission management.

At the end of the camp, they'll understand to have a self-sustaining colony on Mars, these types of systems need to be engineered to a level so that every single person on Mars – even a 9th grader who might have been born there – can operate, design, and debug those systems, on the fly, with little or zero help from Earth.

Prerequisites

Basic Boolean Algebra screening after a single lesson taught by participating 8th grade teachers. REDS will provide the lesson plan and materials. Teachers recommend students who are self-motivated and have the focus to learn a challenging skill.

Curriculum

The students will learn the fundamentals of digital logic through direct application using the C programming language. They will learn the fundamentals of memory operations, including pointers, function calls and the call stack. They will learn and apply mathematical, logical and bitwise operators, and control loops. Students will design, debug, and test their acquired knowledge by programming digital hardware and compete for the best solutions to challenging problems. The projects should be fun, such as robotics or drones, but might be real R&D projects from sponsors.

Fairchild Laboratories (HSW)

An industry supported lab where high-school students break down digital systems problems, characterize them through advanced testing and instrumentation techniques, build mathematical models, and then implement virtual and material models of optimizations and solutions.

Students begin with basic lab work, running tests. As they progress through the breakdown process, they will earn skill-based promotions that directly translate to hire wages. Each incoming rank of students will be expected to recurse upon the REDS toolset and improve it so that the productivity – and wages – of the next incoming rank are leveraged to a higher level.

The long-term goal of Fairchild Lab is to be a self-sustaining enterprise. The lab will investigate specific problems from sponsors. Workers may propose derivative experiments based on insight derived from their assigned work. Students will be taught to seek out problems to solve.

Recursive Integration Methodology Laboratory (UML)

The university lab will perform graduate level validation of the tools, techniques, and teams in the HWS, and look for broader and specifical application of innovations driven through the lab. They will duplicate and correlate the results of the HSW, validating the results with mathematical proofs.

The goal is to get academics competing for grants from sponsors of the HWS and MSC programs based on data generated by those programs.

REDS Advisory Panel

Advisors have read the proposal and provided direct feedback and wish to be involved in the project, either directly or as advisors, as the project evolves. There is no assumption of commitment beyond that. A formal panel will be formed with a schedule and agenda once minimum viability funding is secured.

Brandon Awbrey
Carol Fairchild
Jim Wells
Pam Wells
Justin Allen
Paris Tompkins
Tom Woodard
Mike Zandy
Mark Heerema
Ishaq Unwala

Implementation

REDS

Governance

REDS shall be formed as non-profit 501(c) corporation. REDS will seek funding from individuals who become sponsor members. A board shall be elected by the sponsor members, which each member having one vote, regardless of their contribution level. The board shall serve as a proxy for an elected Shadow board of students. The Shadow Board will be elected by participating students of all REDS programs. The Shadow Board shall set the agenda, form a budget, seek additional funding, and expand the program.

Fairchild Labs

Sponsor Expectation and Responsibilities

Sponsor companies are expected to provide challenging and tactically applicable real-world tasks and experiments so that students can be taught real world engineering skills in working lab environment. The measure of success of the lab to the sponsor is the quality and value of the work the students provide through Fairchild Labs.

There is no expectation for the sponsors to provide or facilitate employment of the students after they complete the program. The leverage of a community's skilled young adult workforce is the community's responsibility. Businesses are cyclical and dynamic, demands and requirements change with the economy and technological trends, but educating and providing employment and entrepreneurial opportunity is a permanent responsibility of the local community.

The local community's responsibility is to build a talent base and create a competitive marketplace for the skills of the local workforce, driving up wages and lifting the entire community to a higher standard of prosperity.

Mission

To teach students valuable digital design skills in a working lab environment, while providing employment and a local engineering talent development pipeline.

Goal

To reach majority funding self-sufficiency within two years.

Governance

Fairchild Labs shall be formed as a non-profit 501(c) corporation, with board seats assigned to each of the founding sponsor organizations. There shall be a board seat representing the interests of the students filled by a non-sponsored member of the local community.

Schedule

Provisional schedule would be four-hour shifts, Monday-Thursday, likely from 4pm to 8pm, as well as a Saturday shift. Students are expected to commit to a minimum schedule, and likely one day (Monday) would be required for all participants for weekly team planning.

Saturday shifts will consist of a formal programming class of two hours followed by a work session.

Transportation

During the sponsored funding period (first two years) the plan is to provide professional transportation after school to Fairchild Labs facility, and home (within the school's normal boundaries) after work.

Parent and Teacher Engagement

<<Looking for advice here: My thoughts, working at Fairchild Labs will be just like any part time after school job – McDonalds or Chick-fil-A or whatever; quality and focus is required. However, the mission is to help these students develop useful skills and opportunities. How do we balance the expectations of work versus the mission; What level of involvement can we expect from Parents? Should we make continued employment conditional on grades? >>

Timeline

Planning periods align with school Fall and Spring semester, using "season" in same context as team sports. It's a part-time job, with the same fixed schedule for all participants, with reasonable accommodation for high school students. The Lab will have the same expectations for work ethic, respect and behavior of any local employer.

Date	Item
March 2024	Fill Advisory Panel
April 2024	Seek initial funding commitment
May 2024	Planning, feasibility recruiting
Summer 2024	Possible 1-2 student training, shake down methodology. Lab setup Equipment sourcing Training
Fall 2024	6 Student feasibility work-program "season"
Spring 2025	12 Student season
Summer 2025	Expanded hours work program
Fall 2025	12 Student season
Winter 2026	Self-sustained funding

Operation

The students will start with the basics, Assembling computer hardware, installing operation systems, configurating applications, and running benchmarks. The first test is correlation; students will compare test results and breakdown the differences between systems. They'll learn to use software tools to visualize the data they collect, and statistical techniques to isolate unexpected and unusual results. At each step, they'll be encouraged to optimize the process and tools, while constantly correlating their results to expectations.

In mainstream R&D, we continually encounter data that is unexpected but not pertinent to the problem we are actively investigating. Often we note these cases as curious, and hope to come back and investigate later, but because of operational demands, they remain mysterys. Fairchild Labs will investigate all unexpected conditions, not just for knowledge, but as a way for students to improve their problem solving skills.

Advancement

Students will be taught skills in a lab environment, working on real digital systems problems. The goal is to advance both their skill and pay as rapidly as possible. Promotion will be based on merit and demonstration of skill. Advancement will not be based on age, school grade level, or time in program. As the toolset improves, the expectation is that we can raise both the expected engineering leverage and wages of the participants entering the program.

Level	Digital Systems Architecture Skills
1	Basic workplace skills; Build and Configure computer systems; Run correlation test; Manage data sets; Analyze data and generate visualization of results; Generate reports; Basic troubleshooting of HW and SW; Presentation of results to sponsors;
2	Basic scripting; Advanced analysis; Advanced visualization; Breakdown of systems; Implementation of mathematical models; Propose design improvements;
3	Basic proficiency in C and C#; Integration of lab process into REDS; Advanced statistical analysis; Advanced troubleshooting. Simulation and emulation of systems;
4	Intermediate Proficiency in C or C#; Advanced debug; Advanced emulation and simulation; Propose new designs;
5	Lab Ready Functional Engineer

Budget

This budget is a rough planning budget for funding commitments. A formal operational budget will be presented to the board before operations begin.

Two Year Summary Funding Targets

Plan	Students	Total Cash Outlay	Targeted Founding sponsors	2 year per student	Matching Goal per Sponsor
Aggressive	17	823,808.00	5	60,361.50	164,761.60
Conservative	11	466,112.00	3	55,378.29	155,370.67
Minimum	7	323,840.00	1	51,734.86	323,840.00

Aggressive Budget

Fairchild Labs - Aggressive 2 year planning budget

Cash Funded Item	Fall Y1	Spring Y1	Summer Y1	Fall Y2	Spring Y2	Summer Y2	2yr Total
Weeks (per semester)	18	18	8	18	18	8	
Hours	20	20	30	20	20	30	
Student Workers	6	11	11	17	17	17	
Wage Per Hour (mid-point)	$15.00	$20.00	$24.00	$26.00	$28.00	$30.00	
Base Wages	$32,400.00	$79,200.00	$63,360.00	$159,120.00	$171,360.00	$122,400.00	
Administrative Wage Overhead	20.00%	20.00%	20.00%	20.00%	20.00%	20.00%	
Total Wage Budget Line	$38,880.00	$95,040.00	$76,032.00	$190,944.00	$205,632.00	$146,880.00	
Transportation Subsidy	$9,000.00	$9,000.00	$4,000.00	$9,000.00	$9,000.00	$4,000.00	
Fixed Overhead (Insurance, etc)	$5,400.00	$5,400.00	$2,400.00	$5,400.00	$5,400.00	$2,400.00	
Total	$53,280.00	$109,440.00	$82,432.00	$205,344.00	$220,032.00	$153,280.00	$823,808.00

In Kind gifted by sponsors	Fall Y1	Spring Y1	Summer Y1	Fall Y2	Spring Y2	Summer Y2	2yr Total
Dev PCs	$6,000.00	$5,000.00	$0.00	$6,000.00	$0.00	$0.00	
DUTS	$12,000.00	$10,000.00	$0.00	$12,000.00	$0.00	$0.00	
Rent (write off)	$9,000.00	$9,000.00	$4,000.00	$9,000.00	$9,000.00	$4,000.00	
Cloud	$9,000.00	$9,000.00	$4,000.00	$9,000.00	$9,000.00	$4,000.00	
Network	$1,800.00	$1,800.00	$800.00	$1,800.00	$1,800.00	$800.00	
Total	$37,800.00	$34,800.00	$8,800.00	$37,800.00	$19,800.00	$8,800.00	$147,800.00
Total Value (In-Kind + Cash)	$91,080.00	$144,240.00	$91,232.00	$243,144.00	$239,832.00	$162,080.00	$971,608.00

Conservative Budget

Fairchild Labs - Conservative 2 year planning budget

Cash Funded Item	Fall Y1	Spring Y1	Summer Y1	Fall Y2	Spring Y2	Summer Y2	2yr Total
Weeks (per semester)	18	18	8	18	18	8	
Hours	20	20	30	20	20	30	
Student Workers	5	5	7	11	11	11	
Wage Per Hour (mid-point)	$15.00	$17.00	$19.00	$21.00	$23.00	$25.00	
Base Wages	$27,000.00	$30,600.00	$31,920.00	$83,160.00	$91,080.00	$66,000.00	
Administrative Wage Overhead	20.00%	20.00%	20.00%	20.00%	20.00%	20.00%	
Total Wage Budget Line	$32,400.00	$36,720.00	$38,304.00	$99,792.00	$109,296.00	$79,200.00	
Transportation Subsidy	$9,000.00	$9,000.00	$4,000.00	$9,000.00	$9,000.00	$4,000.00	
Fixed Overhead (Insurance, etc)	$5,400.00	$5,400.00	$2,400.00	$5,400.00	$5,400.00	$2,400.00	
Total	$46,800.00	$51,120.00	$44,704.00	$114,192.00	$123,696.00	$85,600.00	$466,112.00

In Kind gifted by sponsors	Fall Y1	Spring Y1	Summer Y1	Fall Y2	Spring Y2	Summer Y2	2yr Total
Dev PCs	$5,000.00	$0.00	$2,000.00	$4,000.00	$0.00	$0.00	
DUTS	$10,000.00	$0.00	$4,000.00	$8,000.00	$0.00	$0.00	
Rent (write off)	$9,000.00	$9,000.00	$4,000.00	$9,000.00	$9,000.00	$4,000.00	
Cloud	$9,000.00	$9,000.00	$4,000.00	$9,000.00	$9,000.00	$4,000.00	
Network	$1,800.00	$1,800.00	$800.00	$1,800.00	$1,800.00	$800.00	
Total	$34,800.00	$19,800.00	$14,800.00	$31,800.00	$19,800.00	$8,800.00	$129,800.00
Total Value (In-Kind + Cash)	$81,600.00	$70,920.00	$59,504.00	$145,992.00	$143,496.00	$94,400.00	$595,912.00

Minimum Budget for Viable Program

Fairchild Labs -Minimum 2 year planning budget

Cash Funded Item	Fall Y1	Spring Y1	Summer Y1	Fall Y2	Spring Y2	Summer Y2	2yr Total
Weeks (per semester)	18	18	8	18	18	8	
Hours	20	20	30	20	20	30	
Student Workers	5	5	7	7	7	7	
Wage Per Hour (mid-point)	$15.00	$16.00	$17.00	$18.00	$19.00	$20.00	
Base Wages	$27,000.00	$28,800.00	$28,560.00	$45,360.00	$47,880.00	$33,600.00	
Administrative Wage Overhead	20.00%	20.00%	20.00%	20.00%	20.00%	20.00%	
Total Wage Budget Line	$32,400.00	$34,560.00	$34,272.00	$54,432.00	$57,456.00	$40,320.00	
Transportation Subsidy	$9,000.00	$9,000.00	$4,000.00	$9,000.00	$9,000.00	$4,000.00	
Fixed Overhead (Insurance, etc)	$5,400.00	$5,400.00	$2,400.00	$5,400.00	$5,400.00	$2,400.00	
Total	$46,800.00	$48,960.00	$40,672.00	$68,832.00	$71,856.00	$46,720.00	$323,840.00

In Kind gifted by sponsors	Fall Y1	Spring Y1	Summer Y1	Fall Y2	Spring Y2	Summer Y2	2yr Total
Dev PCs	$5,000.00	$0.00	$2,000.00	$0.00	$0.00	$0.00	
DUTS	$10,000.00	$0.00	$4,000.00	$0.00	$0.00	$0.00	
Rent (write off)	$9,000.00	$9,000.00	$4,000.00	$9,000.00	$9,000.00	$4,000.00	
Cloud	$9,000.00	$9,000.00	$4,000.00	$9,000.00	$9,000.00	$4,000.00	
Network	$1,800.00	$1,800.00	$800.00	$1,800.00	$1,800.00	$800.00	
Total	$34,800.00	$19,800.00	$14,800.00	$19,800.00	$19,800.00	$8,800.00	$117,800.00
Total Value (In-Kind + Cash)	$81,600.00	$68,760.00	$55,472.00	$88,632.00	$91,656.00	$55,520.00	$441,640.00

Microchip and UHCL

Methodology Lab Feasibility Study

Purpose

Determine Feasibility of a Methodology Lab in a University setting. Output of this study will be a validation report on measurement techniques, and an implementation plan for the permanent lab.

Background

In data storage changing from spinning media (HDDs) to flash (SSDs) requires changes in measurement methodology. Advanced design processes also require that new ASIC architectures are fully validated in virtualized environments. Virtualized environments suffer from stretches time domains, where legacy storage measurements often use single data points that take seconds or minutes. A new statistical based methodology is needed that derives necessary data from many small samples.

Microchip wishes to commission a study and validate our approach to both measurement and improving the system design process.

Staged Plan

This is a staged study designed to evolve towards an independent **Fairchild Systems Architecture Lab** as described in this proposal. Microchip proposes an initial staged feasibility and demonstration project as a starting point for this effort. We would like to consider a two-year plan.

Each stage in the plan is a semester.

Lab Operational Costs

Microchip chip would like to have flexibility from semester to semester with regards to the size of the team. We would like agreement to specify lab operating costs on semester basis, based on a rate per student worker per semester.[1] We expect the initial two-year study, at maximum, to involve no more than ten students in any given semester.[2]

Lab Overhead Costs

Lab fixed overhead (rent, power, IT costs) should be broken separately and specified either on yearly or semester basis.

Lab Built Out

One time cost for lab build out should be specified separately and cover the two-year feasibility phase.

Stages

Stage 1: Feasibility

Described in detail in the next section. Validate basic tool set and ready lab for future stages.

---

[1] Or may specify in units of two students. We want fully engaged students with expectation of working no more than 20 hours per week.

[2] Ten or the maximum under supervision under single professor, whichever is less.

Stage 2: Correlation
Students will correlate the most basic storage application, file copy, to synthetic benchmarks, using statistical analysis techniques.

Stage 3: Quantification and Optimization
Students will quantify measurements into single metric set, and then determine optimal values for application performance.

Stage 4: Modeling
Students will model their optimizations using a programable RAM drive and measure actual performance against their predictive optimizations.

Partner Sponsors
The intent is to evolve to an independent, self-sustaining Methodology Lab. With this goal in mind, it benefits Microchip and UHCL to bring in other industry partners as soon as possible. We would like the proposal to be amendable, and partners to join at any of the stages. Partners contribute not only funding, but expertise, as well as software and tools, and will work to correlate the lab's findings in their own labs.

The University should maintain a clear accounting of all material commitment by sponsors. This is so that responsibility for the governance of an independent lab can be divided proportional to the commitment of the sponsored lab.

Detailed Breakdown, Stage 1: Feasibility

Measurement Methodology
Statistical methodology requires small samples of short duration. These studies are initial steps in validation of the methodology. Two aspects of measurement will be considered, and these will be correlated to and compared against traditional measurement techniques.

This study should be performed by students with experience with a strong background in statistics and data visualization. Rudimentary C and Linux skills will be helpful.

Isolation
Data points are taken back-to-back, with a small delay for software cleanup and initialization. Compare this with deterministic "settle" times between data points.

Duration
Traditional benchmarks specify duration in terms of seconds. New methodology requires much shorter duration data points, specified in the number of operations. Compare short tests of various counts and determine if a deterministic value can be extrapolated from short tests. Evaluate the value of an incremental count and measure drift by count.

Test Methodology
This study will plan and implement basic lab infrastructure. The long-term goal is a REDS digital design lab, but we need to start with a framework for the students to recurse and optimize upon. The study will

plan lab infrastructure and come up with a starting point for students. Must Evaluate resources needed to open-source toolset and define limited support model.

Engineers are responsible for the tools they use. The lab is an engineer's most basic tool. Establishing a modern process for creating a lab for defining a methodology might be just as important and insightful as establishing the methodology.

Team (3)

Professor. Should have background in statistical analysis of data, visualization (charting) and previous hands-on lab experience. They will supervise students, set schedules and interface with sponsor.

Student, Measurement. Have basic C programming experience, make minor modifications to existing toolset. May need to transform data sets for analysis. Maintains process documentation.

Student, Test. Basic Linux system administration, familiar with lab environment. Will manage data sets and create skeleton process for test execution.

Equipment

The feasibility study will focus on the toolset and process, so testing can be done on legacy system, thus reducing cost. Microchip would like to provide (gift) raw servers and SSDs.

Stage One Estimated Budget – Lab Operation

Item	Unit
Weeks (per semester)	18
Hours	20
People	2
Wage Per Hour (mid-point)	$25.00
Base Wages	$18,000.00
Professor Supervision	$20,000.00
Raw Total	$38,000.00
University Markup	30.00%
Estimated Subtotal	$49,400.00

Fairchild Systems Architecture Lab

V0.15

"*Engineers are responsible for the tools they use."- Steve Fairchild*

Purpose

A laboratory to define and validate open measurement methodologies and tools for digital systems.

Background

Successful architecture of a digital system requires a proven measurement methodology. In complex open systems, this can be difficult, as no sub-system is completely independent.

Subsystem architects often use synthetic benchmarks to isolate and optimize their components. Without correlation to complete system applications, it can only be a design measurement, not a measurement of architectural performance.

Scale-out architectures dominate hyperscale and enterprise compute. This trend will continue and accelerate as silicon designs reach the geometry limit. In scale-out, the most important factors aren't raw performance of a subsystem, but the efficient operation of a node under a controlled load. There's an adage in the PC industry, "*You can never add too much memory*." That's not true in scale-out – Too much of anything is as bad as too little. Overprovisioning a node is a waste of capital, power, and time.

Digital architecture is very simple. Broken down into components, architects use data-flow analysis to size and connect the components. It's very much like designing a system to efficiently load railcars or a container ship. It can be modeled and visualized in this manner.

Fairchild Lab's primary mission is to develop a recursive methodology and toolset, one which self-adjusts and demands constant integration as parts of the system change.

Architectural Breakdown

Since the lab is funded and supported by the Storage Industry, we'll begin a system breakdown on high-performance SSD storage.

Recursive Engineering Design System (REDS)

The industry trend in chip design over the past fifty years has led to large design teams and longer design cycles. The trend for the future is smaller design teams and shorter design cycles. To get there, engineering teams must fully leverage Moore's law and constantly optimize architectures, designs and most importantly, the design process.

Key to rapid integration are tools and practices that allow teams to effectively debug and communicate complex designs. Unfortunately, larger design teams have led to more specialization. This has led to specialization of tools and processes, creating separate engineering abstraction domains. Teams using different abstraction domains must constantly reframe their understanding of problems into their own abstraction. This is inefficient and error prone.

Fairchild Labs will develop a fully recursive data and test management system, with cohesive and modern data visualization. It should also include integrated scheduling and management of lab resources and staff. It will use a common abstraction domain to relay all data and visual interpretation.

Lab sponsors encourage student workers to develop their own team structure and work processes, including terminology and leadership. We believe superior results will come from small teams of five or less working together on a common task set and objectives.

Characterization

The storage subsystem, or “stack” consists of many software and hard components. An architectural breakdown uses SW and HW tools to take measurements of the same load at various measurement points in the stack. By “breaking down” the stack, we can measure the efficiency of the components using traditional benchmarks, isolating the bottlenecks and resource constraints.

The lab will characterize production SSDs, controllers, and storage fabrics and produce a publicly available data set on baseline reference hardware.

The lab may also perform private characterization of prototypes under NDA for sponsor members.

Correlation

The lab and sponsor companies will work with software vendors, hyperscale data-centers, as well as end user customers in business and academia to define application reference baselines. By using the same hardware and software instrumentation we use for characterization, we can also characterize application loads.

Modeling

Using characterization and correlation data, we can build accurate models of subsystems, as we know the inputs and outputs rates and latency of each component. First, we’ll build models of existing components on a simulation and visualization platform. We’ll validate the models by running the same characterization and correlation benchmarks. At that point, we’ll have the tools to model new devices based on creative input from the students, as well as directed models from the sponsors.

Emulation

Models the prove worthy can be instantiated as software or hardware emulation and validated. The lab will maintain funds for student projects.

Publication

The lab will publish characterization reports openly after a review period, where sponsors may review and rebut methodology or correct deficiencies in their product. All characterization reports will fully document methodology and open source all tools required. The lab will also encourage students to integrate findings and innovations into their course work, practicing recursive integration on the curriculum.

Summer/Semester Co-Op

The lab will work with sponsors to develop semester and summer work programs for lab students. These programs should directly leverage student lab experience and should be goal oriented with measurable results.

Governance

The Lab shall be formed as an independent non-profit entity and managed by a board.

- Founding Sponsor Company – one permanent seat (propose 5 total)
- University Trustee – one permanent seat
- Alumina Trustee – one permanent seat
- Student Trustee – active participant selected/elected seat per semester
- Rotating Trustee(s) – one seat for every three non-founding sponsors

All Sponsors are committing to the educational and vocational improvement of the student-engineers as the primary mission of the board.

Budget

<Brandon: This is for open discussion; I just want to write something down to begin the conversation with both university and partners. I need to have more discussion with university and consult with existing labs >

The lab should be fully funded for a minimum sustainable period of five years. This is to ensure we can measure the effectiveness of working student-engineers over a typical college enrollment period. We propose that a five-year budget be committed by the Founding sponsor and split evenly, to fairly distribute commitment to the responsibility of the board seats. 50% of the budget for each sponsor is reserved for direct student tasks and managed as a credit for priority of future task assignments.

Semester Cost		Unit
Weeks (per semester)		17.2
Hours		20
People		5
Wage Per Hour		$25.00
Admin overhead %		5.00%
Professor supervision %		20.00%
Base Wages		$43,000.00
Wage Overhead		$53,750.00
Baseline Equipment		$50,000.00
DUTs (SSD/IOC/HDD - sponsor product)	IN KIND	
Total Per Semester		$103,750.00

5 Year Breakdown	Unit	
Founding Sponsors		5
Cost of 1 Team per semester		$103,750.00

Number of Teams	2
Years	5
Total Semesters	20
5 Year Minimum Budget	$2,075,000.00
Founding Sponsor 5 Year Commitment	$415,000.00

Non-Founding Members

The board should debate new sponsorships. We should add sponsors only with multi-year commitments and in line with the long-term capabilities and goals of the Lab itself.

Report Publication Disputes

The board should be hands off and the lab self-directed with guidance to accomplish their macro-tasks. We expect student-engineers to prove their findings with data. We will set a timeline for publication, where sponsors and the board have a right to suggest better techniques or methodology. The board should specify these rules in writing.

We expect that all sponsor companies should be able and willing to correlate the lab's finding using the lab's tools and methodology. If disputing finding, sponsors should provide counter-data and help with debug. In all cases, the direction should be a formal adjudication process, with publication held as long as necessary to resolve the issue of objection.

Confidentiality

Sponsors wishing for characterization of prototypes may wish to keep their activities and results confidential from public and other sponsors. The board shall devise a process of supervising and administrating these activities.

Compensation

The lab shall have competitive pay, based on skill, experience and leadership. Tasks are given a skill rating by sponsors. Student engineers may advance through the ranking and pay levels as fast as they are able to demonstrate competency. Specific tasks may have course prerequisites.

Skill Level	Competencies
1	Run tests, manage data sets, prepare visualization of data for review
2	Test infrastructure, debug of results, lead reviews, task development
3	C development of tools, models and infrastructure. Team leadership. NDA Tasks.
4	Advanced modeling, emulation. Sponsor Engineer I expectations [Engineer I]
5	Solicitation of work from sponsors, engaging with non-sponsor institutions. [Engineer II]

Plan of Action

- Spring 2023
    - Identify key professors.
    - Identify underclassmen with skillset and leadership potential.
    - Get 3 to 5 students working immediately (participants for fall 2023)

        - Work directly for Microchip until Lab is legally formed; or
        - Work for school, funded by sponsors
    - Prepare base toolset for open-source.
    - Identify student leadership for Fall/Summer
- Summer 2023
    - Develop budget for continuous operation for 5 years
        - Base budget for 10 students
        - Founding sponsor commit for five years
        - Goal is 5+ founding sponsors
    - Form lab as non-profit governed by trustees appointed by sponsor companies.
    - Staff/outsource administrative functions
    - Build out initial "lab"
        - 1 rack and 1 bench is all space needed initially
    - Microchip will host student leadership and train directly.
- Fall 2023
    - Begin recursive characterizations.
    - Aim for 10 students
        - Attract at least 5 freshmen to gauge 4 year effectiveness
    - Publish first Report
    - Open source base toolset

Implementation

<Brandon: The rest of this technical and is WIP>

Astros

Astros is a storage performance tool developed by Microchip. It was designed for statistical analysis and small burst sampling. Small burst sampling is important to chip designers, as it allows us to use the same techniques in simulation, where the time and compute resources required is expensive.

We would like to donate the source code to Fairchild Labs and for the staff of the lab commit to open sourcing all tools and providing limited support (release and bug tracking). The Lab is expected to source or design the tools necessary for the job. The lab's motto shall be *Engineers are responsible for the tools they use*. If the right tool doesn't exist, the lab will accept the challenge and built it.

Fairchild Metric

For decades, raw storage performance has been expressed as a rate while driving a load. The metrics are expressed as Operations per second (IOP/s) or as bandwidth, a multiplier of bytes per second. The loads would typically be described as queue depth, operation (read or write or combination), a block size, and a type of access (random or sequential).

Typically, vendors pick a small block size for random, and a large block size for sequential and express the max rates. This is simplistic and not useful for modeling. Why? Because real applications don't behave like that at all.

We propose the lab define a new metric, the Fairchild.

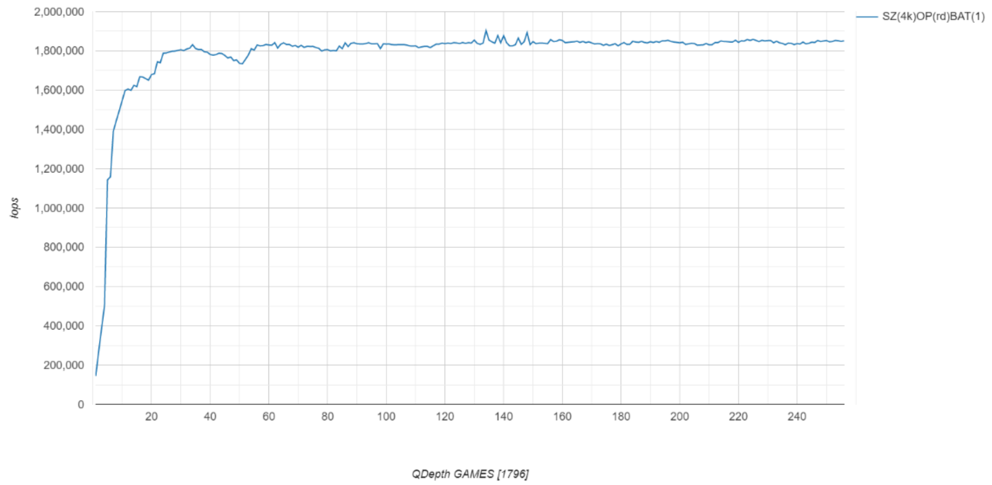


Figure 1- single IOP/s curve

This is a sweep of a queue depth load at a fixed block size. As queue depth grows, so does the raw IOP/s. A simple way to quantify this mathematically would be the area under the queue depth curve. This gives us a single factor metric that is more descriptive than a maximum.

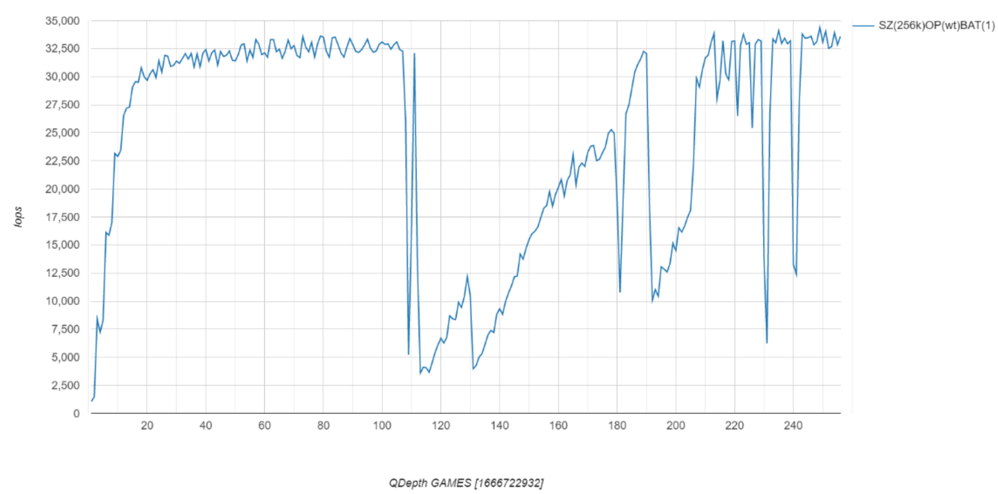


Figure 2 - incoherent curve

Consider curves like this. Scale-out solutions demand components that scale with a reasonable expectation of positive linearity. Non-linearity may be derived from a smaller than expected area, but not always.

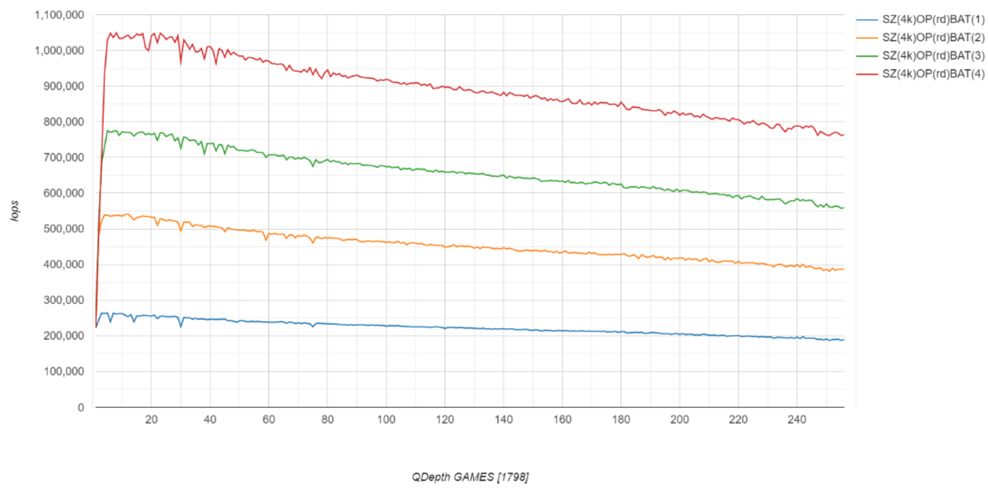


Figure 3 - CPU Scaling

Each series in this chart represents another dimension of the load, the number of “batters” or CPU threads that are used. It’s a good measure of the amount of Host CPU resources to “drive” a load. By splitting the load across more CPUs, we can scale performance to higher levels under otherwise identical loads.

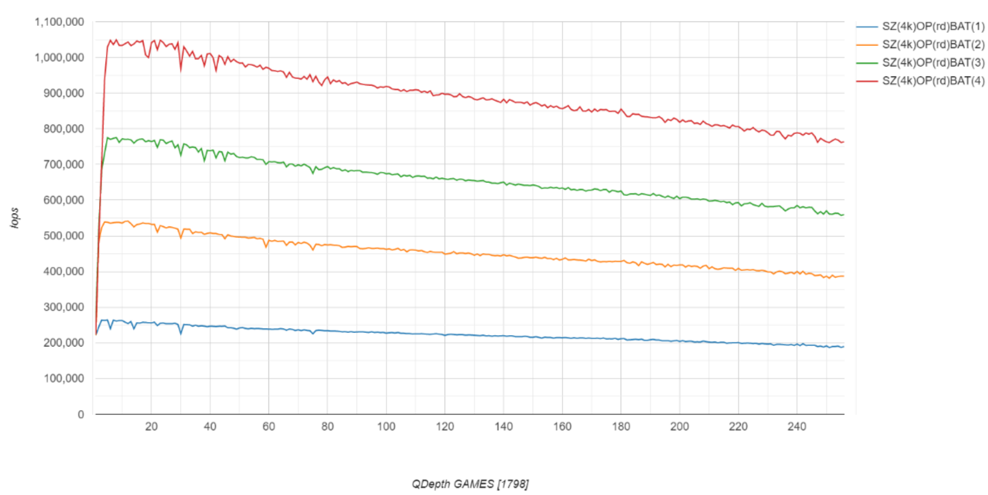


Figure 4 – TODO: Brandon figure out how to do a 3-D volume Graph

If you were to graph this in 3-D, instead of area, you could take a measurement of volume. Instead of graphing each batter as a separate series, you'd make batters the z-axis of your 3-D drawing. The volume of this would be a single factor metric of the gross performance.

Pilots use performance curves to calculate the fuel load on every single flight. They typically use software, but all pilots learn how to calculate this manually from printed performance curves. It's the same problem we face with scale-out. When planes carry too much fuel, the entire flight is inefficient. When planes carry too little fuel, the side-effects are much worse.

Pilots must consider many factors, such as wind direction, temperature, humidity, traffic when calculating performance. They use these factors as biases in their calculation, either manually, or with software. These calculations are very reliable. It's unusual to find a pilot hitchhiking to the next gas station.

We can apply this kind of analysis to digital systems. As we add dimensions of our load, we can extend the metric beyond the 3 dimensions, and still calculate a single factor. Area is the measure of a 2-D curve, volume is the measure under a 3-D curve. Hypervolume is the measure of a topography that has more than three dimensions. Visualizing hypervolume is difficult, as are most concepts in greater than three dimensions. Describing the inputs to the calculation in computing terms is simple – it's just a multi-dimensional array.

For calculating gross load, you first need to properly bias the dimensions of the load itself. Queue depth is an integer measurement, while block size can be expressed as bytes or kilobytes. Likewise, CPU count is an integer, while access type [random, sequential] is just a simple integer enumeration.

$$Fairchild_{grossload} = Fairchild_{gl} = Hypervolume\ of\ the\ tested\ load$$

$Fairchild_{gl}$ may have use cases. It is a full spectrum, allowing designers to measure gross scale of change when they are optimizing their designs. As a useful metric, it has problems. For instance, a high IOP/s rate at 4K reads may indicate strong transactional processing prowess, it's meaningless is the application has optimized on 32K and the bandwidth of the interconnect is the limiting factor. That excess of power is wasted potential in both the capital and power domains.

While not directly useful as a metric, the dataset is useful.

$$Faichild_{applicationbias} = Fairchild_{ab} = Fairchild_{gl}\ x\ Bias\ Matrix$$

Application bias is the measured relative bias of each data point in $Fairchild_{gl}$. This can be calucated by sampling profiles of applications loads, and correlating to data points measured in isolation. Where no correlation can be achieved, new values or dimensions of sampled data points can be added. The lab will recurse on a given application until correlation can be achieved with sampled benchmark data.

The next step is modeling a given $Fairchild_{ab}$ in software, emulation or hardware. With the model you perform the same profile. Once you have correlation with a model with both the application and sampling benchmarks, you can move to optimization for the targeted applications.

Once you have a spread of applications profiled, you can optimize an architecture or multiple architectures to a subset of the problems.

Recursive Lab Management

The Lab shall design a fully integrated system for testing, data management, visualization, data-mining and debug. Every published report shall be replicated on a second set of hardware, by a second student-engineer. The Lab's core testing systems will be "locked-down" and all Hardware or Software changes logged and revalidated against existing baseline.

This might be the biggest challenge, but one where student-engineer's might be able to provide industry with the greatest leverage. Many sponsors have decades of legacy process and practice and often use in-house designed tools that are less than optimal. An engineer's most powerful tool is Moore's law – we need to make sure we always use it to leverage our designs. Just as important is to leverage the design process.

Sustainable Path

The lab can become self-sufficient in terms of funding by contracting specific tasks. Some macro tasks the lab would be well suited for.

- Characterizing a sponsor's production products and calculating the Fairchild metrics for an array of applications.
- Profiling the $Fairchild_{ab}$ for a new application
- Optimizing a model for an application given a specific set of constraints
- NDA engineering studies of protypes and beta-level product

I recently had the experience of going full circle in my career, going back to a similar role to where I started. In engineering it is an enlightening experience – though I imagine that would be true in any career. One of the first things you learn is you weren't nearly as smart as you thought you were way back when.

I was lucky to start my engineering career at Compaq in Houston in the 1990s. In those days Compaq had a mix of electronics industry veterans and young and hard charging engineers, and though I had little experience, I was able to fit in well.

The engineer who most influenced my career was Steve Fairchild. As somebody who was on the front lines of a rather competitive battlefield, it is my opinion that Steve was the person most responsible for the success of both SAS and SFF HDDs.

Back in those days, I didn't think so fondly of Steve. He was big, loud and could be intimidating. My relationship with Steve was all business, I never got to know him outside of work, but I imagine in real life he was giant teddy bear. While he was unmerciful in his engineering criticism, it was all about the engineering, never personal or vindictive. I have watched Steve cut down both vendors and peers in exactly the fewest words possible and without any room for rebuttable. You did not act like you knew what you were talking about around Steve unless you knew what you were talking about.

Our team was responsible for qualification of SCSI HDDs, and my role was programming the Big-Real Mode DOS utility we used for testing, Psuite. Sometime around 1998, I ported Psuite from Compaq's internal SCSI driver library to Adaptec's CHIM library. In a rush to get their chip into production, I had been required to spend weeks in California helping Adaptec debug their chip with my software. It wasn't a lot of fun for me or for Adaptec, as it consisted mostly of me waiting on them to deliver some code to test and debug.

We had a plan to port Psuite to some modern 32-bit OS, and my idea was for Windows. My key selling point was vendors had to have Windows SCSI drivers, so we wouldn't need to do any babysitting, we could just give them the Windows Psuite application and they could debug their own drivers without my direct help. My own motivation was: there wasn't much career potential programming on DOS and the other alternatives were too specialized, but there was plenty of work programming on Windows. Linux was too immature and had even bigger technical challenges to overcome than Windows.

I never had the nerve to ask Steve or anybody else what Steve's actual job was, he was a member of the technical staff and knew everything. His role on our team was performance. His role for our software development team was to greenlight the performance of our software.

My approach is always code first, research later. In the early days of our discussion, I had already ported a basic version of Psuite and proven that the performance would be within 1% of our DOS tool. These were spinning HDDs, and even though we were testing the fastest available on the market, they were terribly slow compared to most other subsystems on a server, and I had the general attitude that performance methodology was not a big deal. My biggest challenge was on the functional side, where we did not have the same visibility to SCSI protocol compliance that we did with our DOS built-in driver. I had the support of my manager with the relative performance, with the caveat, “As long as Steve is okay with it.”

Steve wasn’t okay with it. After presenting my 1% data, Steve tore my plan apart, coming up with a list of around ten performance items that I would have to address. I seem to recall that at least half of the list were things that weren’t even measurable with our DOS tools, so not only did I have to code these new measurement techniques on Windows, with its rich libraries I could leverage, but then go figure out how to back-port the same techniques to DOS so that I could prove to Steve there was no difference.

This went on for months. We’d have these weekly meetings, where I’d want to talk about all the really clever coding I done to move forward on the functional side, and Steve would tell me I’m wasting my time until I’d finished the methodology tasks he’d created for me. It bugged me, because 90% of the problems we worked on were functional error recovery problems, and Psuite was clearly the best tool in the industry for that. Hard Drives are and were always slow, and it did not matter what software tool you used to measure them, they’d still be slow.

I remember after one particularly brutal meeting, I’d gone back to my office to reconsider what Steve had said about my data, and thought I’d caught him in a mistake. I took my printout to his office, and within about 30 seconds, he’d completely turned me around. Not only did I misinterpret my own data, I hadn’t even been asking the right questions. Before I could escape, Steve had me sit down, where he proceeded to draw a diagram on paper and lecture me for an hour, when all I wanted to do was go back to my office and cry. I know the subject was system level performance, and I know what Steve was trying to convey was that methodology was the most important part of measurement, the numbers themselves didn’t matter. I think he was trying to make me feel better about doing all this work that had absolutely no relevance to the current technology but might one day help him with whatever he was working on. I didn’t feel better about it. Boy, do I wish I’d paid attention, would have saved me a lot of trouble down the road.

When I completed the last item on Steve’s list, and I presented the data to Steve in our team meeting. He said, “What are you going to do about aborts?” No, “Good job, man.” or “You were right all along, I should have never doubted you,” but what are you going to do now about the real problem I was keeping you from working on all along? By this point, I’d had to complete so much code to validate Steve’s requests, the project was pretty much done, except for the hardest part. Steve was no help for me there. Apparently, he’d taught himself the Windows Storage Stack fast-path by having a young engineer do a bunch of seemingly useless experiments for him. He had no idea how the error path worked – that was for mortal engineers.

I left Compaq not long after finishing that project to work for Adaptec. During the early days of SAS, I worked very closely with Steve as a customer and he treated me with respect and continued to mentor me, especially when it came to how to deal with my own bosses. After I made a particularly embarrassing and potentially career ending mistake, Steve was gracious and gave me the single best piece of career advice anybody ever gave me. I’ll keep that gem to myself for now, but one thing Steve knew was that in computer business, the engineers and not management or marketing held the real power. For Steve, it was more than just an obligation to point out engineering weaknesses and come up with clever solutions. I believe he felt responsible for making well engineered products, and he knew with the right methodology and data, he could always hold marketing and management to account.

Sadly, Steve was a mortal engineer. I don't remember the dates, but it seems like he got sick right around the time HP shipped SAS. I didn't see much of Steve after that, we went out to lunch a few times a year. In the beginning Steve initiated these lunches. Steve always paid for his own lunch, long before it was HP policy not to accept paid meals from vendors. Somewhere along the way, I got the reputation as one of the few engineers Steve would actually go out to lunch with, so every few months marketing or sales of whatever company I was working for would get the idea that we could make an inside run at some kind of influence or get some information from Steve. I didn't bother with subterfuge with Steve, he would have seen right through it, I would just come right out and tell him what my people were after, and he'd usually say something like, "I'm not going to tell you that." Then we'd have a nice lunch and talk about engineering.

In 2010 I took a job in China. We corresponded a few times via email, Steve was very curious about both life and engineering in China. I came back to the states at the end of 2012, but only saw Steve once, walking very slowly down the hallways of HP. He looked exhausted but took the time to remind me that we needed to catch up and tell him all about China. This never happened. Steve died in 2013.

I've had a very successful career, and for the most part, I've been able to do the kind of engineering I think Steve would be proud of. I never got to tell him what he meant to me when he was alive, but as I've grown older I think about him more and more. Every time I smell some weak engineering from marketing, or management, or other engineers, I ask myself, "What would Steve say?"

Steve engrained in me the notion that engineers are responsible for the tools they use. And Steve said to me, "The methodology is the most important thing." The industry lost a legendary engineer when we lost Steve. He was the best engineer I've ever worked with.

This work is dedicated to Steve Fairchild.

Contact Information:

Brandon Awbrey
scuzzydude@hotmail.com
(713) 305-5620

JUNE SCOBEE RODGERS

In ***Collecting Responsibility***, I recount a moment of inspiration. I wrote about *Challenger* in terms of a postcard I'd lost. It was a critical breakthrough in my thinking. Later that night it was as if the basketball gods were blessing my idea because KPJ destroyed the Bucks, dropping 50 points. It is a good story, and it happened like I wrote it.

It wasn't until much later that it hit me; I'd already finished writing the book and realized something even more profound. It's an even better story. This work is a testament to genius, so I'll keep the embellishment down to a Texas minimum.

I was curious, I remembered *Challenger* well, one of the few moments in high-school I can see clearly through the cobwebs of time. I also remember who I was, what I was like, and there was some incongruity. How did my mom get me to a NASA nerd camp when I was fourteen?

After a full year of 6th grade, I'd left my nerd roots behind. I still read science fiction but wasn't hanging out with other nerds who did. Summer after 7th grade was the last run of my suburban rap/breakdance crew. Thankfully, this occurred before digital photography. Home videos of teenagers doing stupid stuff was for rich people. Summer after 8th grade, I was hanging with kids who listened to metal and hated school, friends I'd have through high school. I know I wasn't volunteering to go to a NASA summer camp.

My Uncle Clay, retired Air Force, had broken the news to me years before. One summer it was, *with your glasses you'll never be a fighter pilot*. Gave it to me raw, then told me how he found out he was colorblind. I think the story was he didn't even know until after he'd already signed up for the Air Force, hoping to become a pilot. The Air Force doctors told him. No jets for Uncle Clay, but he made a career out of military intelligence.

I'd left astronaut dreams behind in grade school, along with the nerds. I wasn't a computer geek. I didn't even like to play video games at home, though the arcade was cool place to hang out. I couldn't understand how I ended up at NASA camp at that age. I asked my mother.

"You got picked because you were good at math," my mom said.

"Oh really," I say, not sure why that would have made a difference.

"Did you have to pay?"

"Oh, no," said my mom. "You went because it was free."

And then I got it. I get along with my mom great now, but as a teenager if there was a way to get me far away from her and it was free, she was first in line. My mom could walk into a supermarket in the 1980s with a box of coupons and walk out with two carts of groceries and $50 cash in rebates.

She'd camp out like it was for concert tickets, to get rid of me for a few weeks in the summer. I can see my teen-self would have made peace with the fact it was all the way in Clear Lake, far away from where anybody might know me where I lived in Cypress.

I had no interest in computers at all. As a kid, I'd wanted to be an astronaut, but as a teenager, no way.

I don't remember much about the camp. We did a tour, saw mission control. We looked through a window at the computer room. I saw my first hard drive at NASA, it looked like and was the size of a washing machine. I remember the engineers were dudes in short-sleeve collared shirts who'd chain smoke and wore bad glasses.

I do remember learning to code on the Apple IIs. Only time I ever used Apple anything until I got an iPhone 25 years later. It wasn't even a real programming language, but this educational language called Logo that drew pictures on a screen. I don't really remember them teaching us much, but they must have. I don't remember if June was even part of the programming class, but she was part of our day.

While we were in some lecture scenario, I remember June talking about science fiction and ethics and the ***Laws of Robotics***. I never read Asimov's robot novels, I'd gone straight to *Foundation*, since my Uncle Clay had told me was the best Asimov. I might have mouthed off something to that effect.

I remember meeting Colonel Scobee, it might have been on the last day of camp. I hope I was respectful. I remember thinking, *he looks like somebody's dad, not a badass pilot*.

He was.

Both.

The tragedy is what stands out in my memory, what happened after. It was only half a year between the camp and the accident, but in a teenager's timeline it felt like a decade.

There was a lot of expectation in the news and among Houston schoolteachers about a teacher in space. We were freshmen boys, I'm sure we mocked it. My youngest kid is that same age now. His disdain for teachers matches mine at that age, for the same reason. People are complex, and kids see things through their direct experience until they've experienced more. Raising teenagers sure does change your perspective on being a teenager.

I have no memory of knowing that Dick Scobee was commander of the teacher-in-space mission ahead of the tragedy. I bet my mom told me and I tuned it out. That January day, realizing what it meant after the accident, was sickening. That memory became a stake in my identity. How shallow we can really be.

I can't blame teenagers for their thoughts on teachers and authority. Look how long it took me to realize the true value of this gift from June Scobee.[1]

Her gift was the greatest leverage in real terms anybody has ever given me. I raised a family and provided for them with this gift. I was 51 years old when I realized it. I was 14 when I went to that summer camp. 37 years. I have made an entire life out of what June taught me I could do in a free summer camp.

This is where the story diverges from what most teachers would want.

I had no interest in technology in high school. I read a lot and was smart and could figure things out. I didn't like being forced to learn what I thought was basic stuff. In retrospect, the skills I'd learned to that point, going to elementary and middle schools in Cy-Fair ISD prepared me very well for the life I've had. Long before the internet, I had the capacity to figure out what I needed to know and the skills to find the answers. I could read, I could write, and I could do math and basic algebra, and that has served me well though my life.

---

[1] A reliable Texan source has confirmed that the camp was created by June Scobee and sponsored by Texas A&M. Karma at work, a lifetime of Aggie jokes has finally caught up with me.

After 11th grade we found a loophole, and me and my friend dropped out of High School, took the GED and enrolled at University of Houston. Neither of us took to college any more than high school, but since we weren't 18 yet, it was an acceptable compromise for our parents. The next year, we both took off for Los Angeles. I pursued music and writing for a few years, doing a variety of odd jobs to support myself.

I knew what a PC was but never used one. In fact, rather than buy a computer, I spent a few hundred bucks on an integrated word-processor with a printer. I had zero interest in technology or engineering, I was writing about what I wanted to write about.

I had a bunch of food and retail jobs, delivery driver, bank teller, liquor store, gas station attendant. In 1992 I was working for a redneck in Waller delivering horse trailers, when my vagabond friend set me up with an air-conditioned job repairing laptop computers. I ordered parts and kept the operation moving along and learned how to put laptop computers together. The secret, I learned, was that if you put a laptop back together and have an even amount of screws leftover, you did something wrong.

That company was somehow connected to Tony in New Jersey, and it went under. My drummer's mom worked at Compaq Computer on the night shift assembly line and used some insider voodoo to get me a temp job under her general protection. We worked on a Printed Circuit Board (PCB) manufacturing line, we made huge motherboards for servers. It was an air-conditioned job, and I learned several roles rapidly, but none needed computer skills. Late at night, you had to stay awake or risk instant termination if you were spotted dozing by a supervisor. It was very difficult as the line was often stalled for hours at a time due to changeovers or supply chain issues. There was one lady who did stuff on the computer. She'd open reports and print them, then collate them manually and then print out another report, which she would give to the boss. You couldn't read books on the assembly line even if there was nothing to do. I quickly figured out the best thing to do was look busy when there was nothing to do or somebody would make me do something I didn't want to do.

Computers in those days came with a set of manuals, including a BASIC programming manual. Extra books that came with new systems were always around the line and were the one kind of reading material allowed on the floor besides specifications, work orders and schematics. I found that I could teach myself BASIC by figuring out how to collate all the files and print them out and save the lady some time each night. It probably took about a month.

Didn't go like I thought when I unveiled my innovation to the lady. Turns out she liked staying late to turn in the reports so she could get a half-hour overtime pay every day. Plus, complaining about staying late to the young people gave her a way to stay awake all night. My drummer's mom rescued me, but I was banished to another shift in a negotiated truce.

After a couple years of this, I found my way into an engineering team. I started doing tech stuff around the lab. I automated some stuff with batch files. An opportunity came up for a programming assignment in C, and my brother had a book about C, so I told them I was qualified. Turns out, I was, thanks to June Scobee.

From that path on, my career through digital logic has been smooth sailing. The basic skill I learned back in 1985 was all I needed. I call that integrative thinking, the ability to break down problems into smaller pieces, and solve the pieces. It's a common skill, used in all industries, with or without code.

The more you do it, the better you get at it. The real power is your ability to use code as a tool to apply even greater leverage. That's just looking at real world problems the same way as you solve code problems. It doesn't take four years of high school and multiple years in university to learn those skills. It takes identification of natural potential and a little directed on the job training. Generations raised on digital logic are naturally more adaptive to integrative thinking because that is how software works.

I remember I'd been programming in C++ for about two years when I finally got some real training in a class we took at Compaq. My mentor, who was a C programmer only, but light years beyond me in engineering terms, was sitting next to me struggling with some examples in C++ which was new to him. I remember him watching me do some trick on the command line in Windows that was illegal in Unix and him pointing that out and I felt so smug.

That memory is truth. How leveraged by the past we are. My bag of tricks in programing was like a cheat code in a game to him. He'd had to solve all these puzzles himself, through incredible hours of focused thought. Every time I ran into a difficult programming problem – even in 1998 – all I had to do was write the problem clearly in English and type it into a search engine and I'd find at least a hundred ways somebody had solved the same problem.

Leverage is the prime component of any engineering task. Digital leverage is the most powerful tool man has ever created. That is because all engineering is digital engineering. Good engineers in every field are the ones that use software as a tool. *Engineers are responsible for the tools they use*. That's the most valuable engineering lesson. The sooner engineers understand their basic toolset, the more effective engineers they will be. All industries are potential innovation frontiers for young people trained to use digital tools.

Coding isn't for everybody.

This ability is not the same as being good at math. The math we use in integrative thinking is very simple.

I don't know how June Scobee identified that I was a good candidate to learn how to code.

I suspect if you ran the same studies to identify kids who'd be good engineers in the 1980's on kids from the 2020's, it would result in a much broader candidate pool. How could it not? We give very young children digital devices to play games with, sometimes before they can walk or talk.

It could be some kids are good at games because they have an innate talent, like some kids can learn to juggle. I doubt it. Video games are designed by people who like to play games, so there's a level of insight you can gain from that.

There are some kids who will always beat a game, given enough time. You can study those kids, and by studying the choices they make, figure out the deductive logic they are using.

Some will follow logical deductive paths. Essentially, they reverse engineer the algorithms the game designers used.

Others might just use monkey-with darts and make a map of what works, either mentally or by recording results.

Both approaches are good strategies to find a solution, and both are demonstrations of integrative ability. The map strategy requires many more iterations. In digital engineering that is natural leverage – something computers can do very quickly. It's easy to design experiments to iterate through a set of possibilities, that's the leverage of Moore's Law.

Digital games usually make you go through a lot of work to keep experimenting with a map strategy, so kids with this ability may tend to give up more easily than those who use a more deductive path.

Other kids might just be lucky. Some will have extra-leverage – they used a cheat code. That's super integrative leverage – exactly what I do when I Google a solution.

This approach is a fair way to study and find kids that could be productive engineers.

You find these kids, wherever they are, teach them to code in a summer camp, and they will have a valuable and marketable skill that they can use directly in almost any endeavor they choose. More directly, you make sure these kids have opportunity to develop that skill in direct and equitable ways.

In Academic settings most kid's prime motivation is to get it over with. Applied learning is often driven by the desire to increase earnings, a real motivator. Having a chance at a high paid profession is a motivation that leverages all parties involved. Dedication is required, but knowing you have the basic skills is all it takes to get started.

Learning to code directly changed my life. Thanks, Mom, for banishing me to the free NASA nerd camp.

I remember Dick Scobee as just an ordinary guy with an extraordinarily cool job.

June Scobee didn't come across as ordinary. The correct Texas term for June is "sparkplug." She wasn't an ordinary mom and schoolteacher. She was like a live action version of a Jetson's supermom, whose kid would be expected to grow up become a general in the Space Cadets. She expected something of her students.

Thank you, June Scobee Rodgers.

I wonder what the success rate of those kids June taught that summer?

After the accident, June and the families of the crew founded Challenger Center. Challenger Center's mission is a natural extension of STS-51-L's mission – The Teacher in Space Project.[2] Challenger Center inspires and educates students in STEM through hands-on lessons that simulate space missions. In the 35 years since Challenger Center opened, they've taught millions of students and enabled hundreds of thousands of teachers.

What's the value of June's contribution to our civilization? It's very valuable, but immeasurable with property. You can't put a dollar value on dreams. They have intrinsic value.

If you read *Silver Linings*, June's book, you know she had to fight just to study what she was interested in. She gets her revenge by forcing the next generation – me – to learn something I was good at but had absolutely zero interest in. How ironic. And recursive. And Texan. And wonderful.

Genius.

My mission is to give every child on the planet the same opportunity that June gave me. To learn to code for free. I don't know how I'm going to do it. I know who I'm going to ask for help.

Thank you, June.

Ad astra!

[2] https://en.wikipedia.org/wiki/Teacher_in_Space_Project