# Software Studios Titus Winters

# Principles of SE

### What's the Secret?

- It means a lot to Titus
- Why is Google good at software?
- Say you are presented with a good good deal
- There is no one tool that is going to be the macguffin to increase production

### It is not just one thing that brings modernization

- It is small and very incremental

### Software Engineering at Google

- The flamingo book
- SE is the multi-person construction of a multi-version object

### TIME, What is the lifetime of a code?

- Hours, Days - Assignments,
  - Definitely programming
- Weeks, Months, Years - Startups/Apps
- Years, Decades - Linux/Apache/Google
  - Definitely SE
- The experience range of most devs is in the assignments-Startups range
- Hyrum's Law
- With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody
- Thermodynamic truth
  - Randomness - a mitigation agains Hyrum's Law
  - We cannot control how things behave, we can only create systems that would increase efficiency
- Multiplicative Difficulty
  - Overcoming H's Law
  - Operating without Policy/Precident/Experience
  - Larger than the Usual Upgrade
- If you do not change anything about your dependencies in your old system, then you will find yourself in a rut

- Sustainability is the goal: for the expected lifespan of your code, you are **able** to change all of the things that you ought to change, safely.
- Many developers have never worked on a sustainable project with a recognized 5+ year lifespan
- SE is not merely programming - it is the art of making a program resilient to change over time
- Keep in mind the expected lifespan
- Sustainable code is capable of change

## SCALE, How are we going to grow?

**Resources**

  - Hardware
  - Software
  - Human
- No task should require extra heroics down the road
- Traditional Deprecation
  - Mark the old version deprecated, introduce a new one, and call it good
  - Mark the old version deprecated, introduce a new one, and mandate everyone update their code by some date. Delete old one on that date.
    - The Great Dish Debacle
    - Timmy if you don't do the dishes
  - Find a brave engineer to go through and build a single change that modifies the API in question
- Having two versions of a function in a codebase does not necessarily mean a deprecation problem

**Better Deprecation**

- The team responsible does the bulk of the work

**In a successful organization, everything that must be done repeatedly must consume sub-linear resources - especially sub-linear human effort and communication.**

- Don't fear scaling, it is large but often a sign of success

**Big merges are very expensive**

- No Weekly Merge Meeting
  - No long-lived dev branches
  - No choices where to commit
  - No choices which version to depend upon

**Review**

- Be mindful of super linear scaling

### Tradeoffs, Make evidence-based decisions

- Everything in SE is analyzing tradeoffs
- Aim for sustainability
- No super-linear scaling
    - Especially for humans
- Re-evaluate as needed
- Avoid "Because I said so" decisions

1. SE is more than just programming, it's Time
    1. Especially consider the impact of time
2. Be mindful of Scale
3. Make evidence-base decisions

### Google SWE Book

- This book gives a good idea
- There is a pdf of it on the internet
    - You won't get in trouble if you download it online

"It's programming if 'clever' is a compliment,
It's SE if "clever" is an accusation"

### This method of working is harder to implement with customers, than an in-house project

- Customers don't often know what they want
    - If you have contracts and clients that depend on your work, then there needs to be a lot of communication between each entities
    - Be a good communicator, communicate a lot

### What does it look like for someone who is just entering the field?

- It looks really hard
- It is hard to implement your practices, when you only know the stuff is in theory
- An awful lot of this is learned the hard way
    - You will make mistakes

### There are tons of legacy code in an Air Force base in Montgomery

- The oddities of software is that its progress of change is a lot faster than other fields
- Tech, in general, not improving stuff that works atm, is not a good idea
    - The field is ever-evolving
- Progress must be handled in increments

## To encourage this kind of development, a big team, small teams, or subgroups?

- A manager
  - In charge of keeping track of progress
- Tech Lead
  - Looks at code quality

  **Communication**
  - Needed very much
  - Small team size can be determined by who can be fed with two pizzas
- The size of a team and distribution of work depends on the project

## How do you make yourself stand out?

- Skills
  - Practice, practice, practice
  - Drills, Warmups
    - Must work on simple, small-scale pieces, kata
  - Get in the habit of writing tests
    - Write your tests before you write your code
    - Nobody likes to kill their creation, so write your tests first
    - Test are there to primarily keep your code of good quality over time
- Interview Hacks
  - Competitions are the only proxy to determine if you have practiced
  - Most interviewers want you to win
    - They are on your side
  - Ask questions
    - If they reprimand you for this, leave the job
    - Asking questions is a very good habit to have for engineering
  - Be honest
  - Prepare