

# Welcome to this Training Session with Theiagen Genomics



We will soon be getting started



# Software Development Practices for Public Health Bioinformatics

*Week 01: Design Documents & Development  
Environments*

Western Region WFD Offering Provided by the  
Washington State Department of Health in Collaboration with Theiagen Genomics

# Course Introduction

# Training Workshop Instructors



*Kevin G. Libuit, MS*



*Andrew Lang, PhD*

# Training Workshop Resources

## Training Information, Communication, and Support

- **GitHub Repo** created to host training resources and information:
  - <https://github.com/theiagen/Western-WFD-2024>
- **Support contact:**
  - support@theiagen.com

# Training Objectives

## Software Development Practices for Public Health Bioinformatics

- Develop a **comprehensive understanding** of software development processes and documentation
- Gain comfort in the use of **development tools and version control systems** for effective code management
- Learn to apply **testing and deployment strategies** to ensure code quality and stability
- Explore advanced tools and **APIs for data management** and workflow automation

# Public Health Bioinformatics

# Public Health Bioinformatics

## Young Discipline

- US public health systems adopted NGS and bioinformatics technologies **~10 years ago**
  - **Next-to-zero bioinformatics capabilities** across the US public health system in 2015
- First practitioners had **limited software experience**
  - Most were *wet-lab-turned-dry-lab* scientists

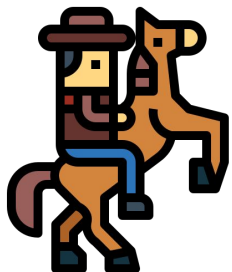




# Public Health Bioinformatics

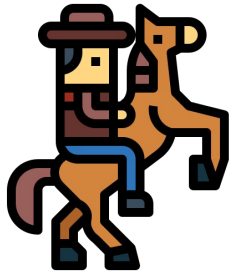
## The Wild West

- Public health bioinformatics was often **chaotic and unstructured**
  - Created **major challenges** in accessible, reproducible, and interoperability software solutions



# How to get a bioinformatics headache

1. See tweet about new published tool
2. Read abstract - sounds awesome!
3. Fail to find link to source code - eventually Google it
4. Attempt to compile and install it
5. Google for 30 min for fixes
6. Finally get it built
7. Run it on tiny data set
8. Get a vague error
9. Delete and never revisit it again



Slide from: T. Seemann, ASMNGS18

<https://www.slideshare.net/torstenseemann/how-to-write-bioinformatics-software-no-one-will-use/11>

# Public Health Bioinformatics

## Huge Leaps Forward

- Major progress was made in our field with the adoption of specific technologies:
  - **Software containerization**
  - **Workflow managers**
  - **Cloud infrastructure**
  - **Graphic User Interfaces (GUIs)**



*Adoption of these technologies led to an emergence of **more mature software development practices***

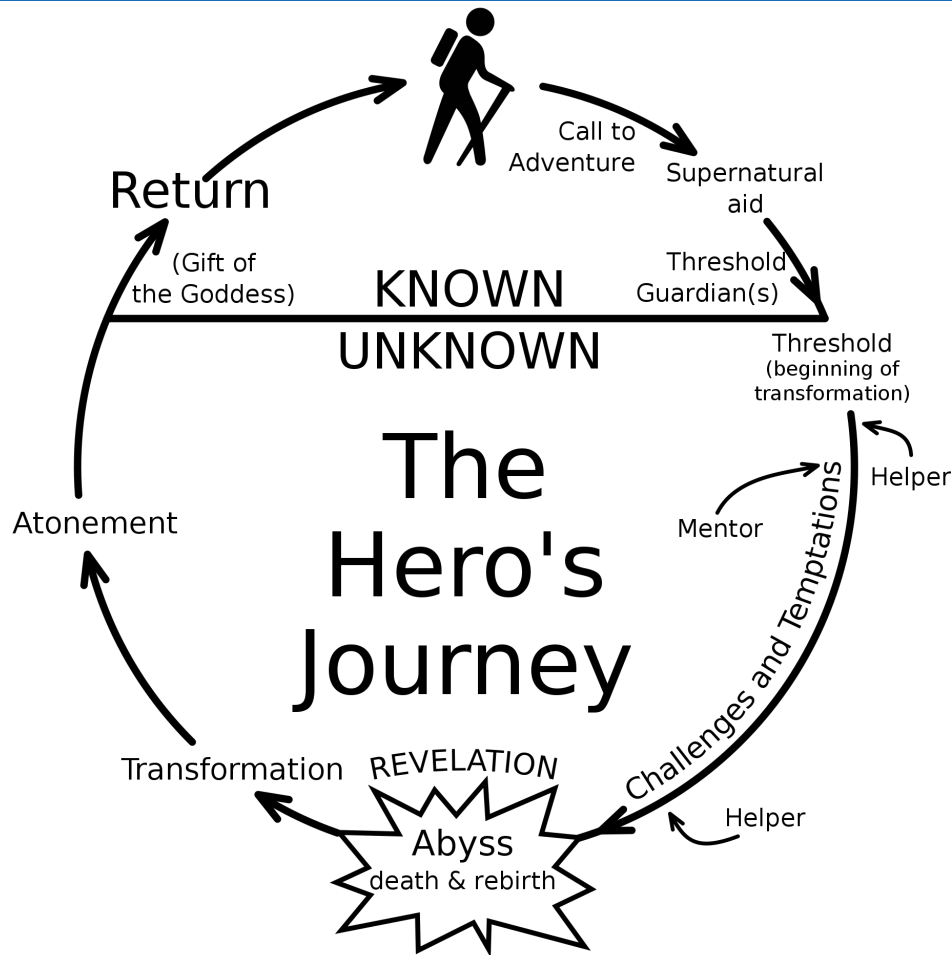
# Public Health Bioinformatics

## Maturing Software Development Practices

- Accessibility, reproducibility, and interoperability became a **major goal across the field**; forcing function for:
  - Adopting of **standardized development practices**
  - **Enhanced collaboration** across interdisciplinary teams
  - Development of software meant for **wide distribution** across the public health community

*These practices have become the **new status quo** in public health bioinformatics*

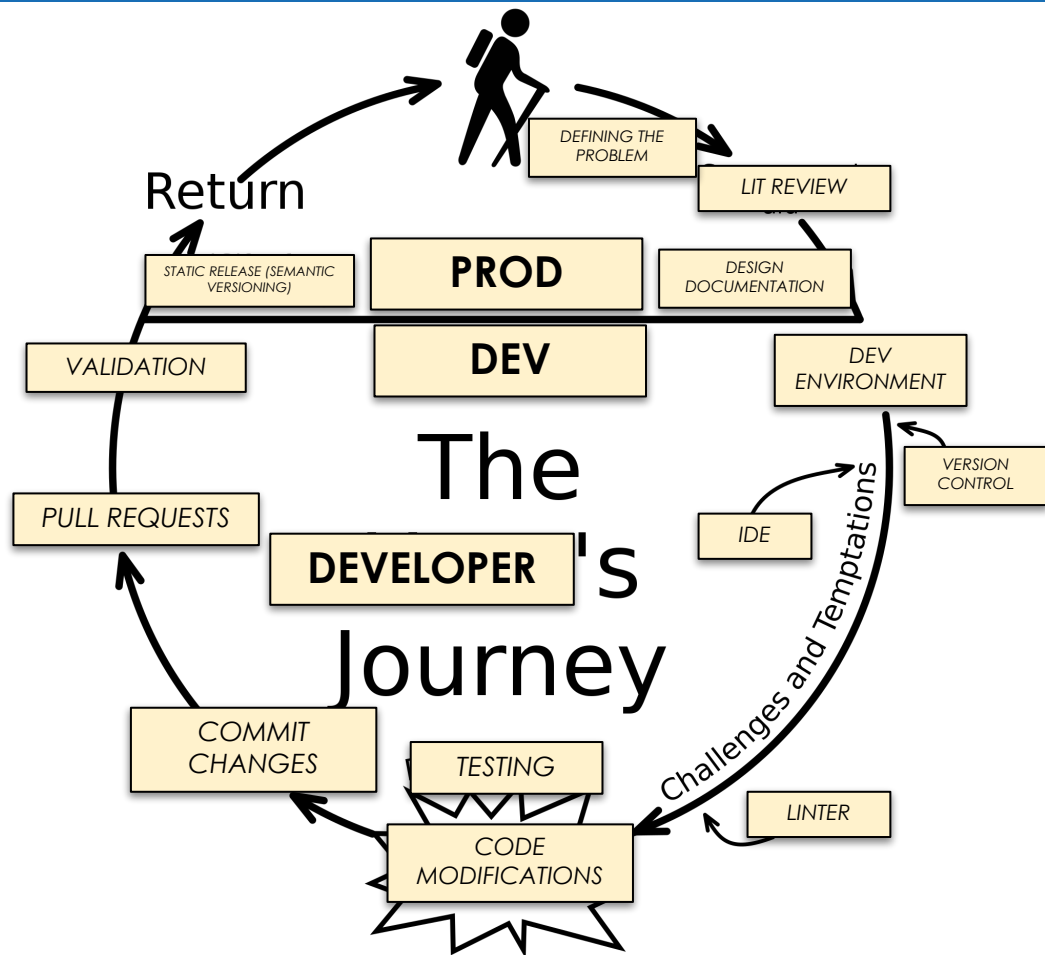
# Software Development Practices



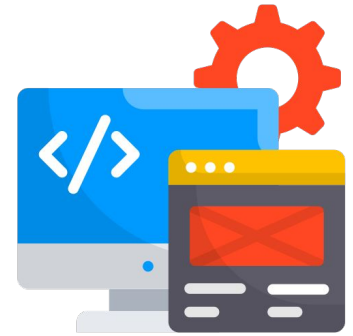
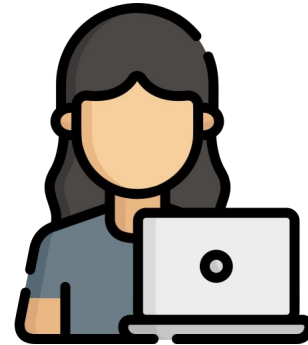
## The Hero's Journey

Framework where a protagonist **enters into the unknown**, faces challenges, gains new wisdom, and **returns transformed**.





**The Developer's Journey**  
Framework where a protagonist **enters into their dev environment**, faces challenges, gains new wisdom, and **brings changes into production**.



# ***A Note on Version Control Systems***

## **Version Control Systems (VCS)**

- Essential development tools that help manage changes to source code over time
  - Track (and save) modifications to the code

## **Git and GitHub**

- Git is a VCS software for managing code in repositories
- GitHub is a platform to host Git repositories

*Git repositories can be hosted on other platforms such as **BitBucket and GitLab***

More on **Git** next week!

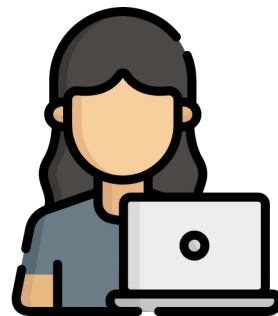


# Software Development Practices

## Developer's Journey

**Week 1 Focus**

- 1. Design Document**
  - a. Clearly defining the problem and the proposed solution
- 2. Development Environment**
  - a. Separate from production
  - b. Text editors and IDE's
- 3. Making Source Code Modifications**
  - a. Small interactive changes (version control)
- 4. Peer Review**
  - a. Collaborative development teams
- 5. Bringing Changes into Production**
  - a. Final testing
  - b. Static version releases



# 1. Design Document

# Design Document

## Defining the Problem and Proposed Solution

- Design docs are written plans that **outline the problem, objectives, proposed solutions, and implementation strategy** for a software project
- Ensure everyone has a clear understanding of the project scope and requirements (**as well as what is out of scope**)



***Living document** that serves as a **reference** throughout the development process*

# Design Document

## Major Components

- Problem Statement: Clearly define the problem that needs to be solved
- Objectives<sup>\*</sup>: Outline the goals and what success looks like.
- Proposed Solution: Describe the approach to solving the problem.
- Implementation Plan: Detail the steps, timeline, resources required, and integration with existing systems



*\*Should be informed by **literature review** and **community feedback***

# Design Document

## Literature Review and Community Feedback

- Conduct a review of existing publications and solutions related to the problem
  - Look specifically for **public health applications**
- Reach out to the **wider technical community**
  - MicroBinfie, StaPH-B, etc.



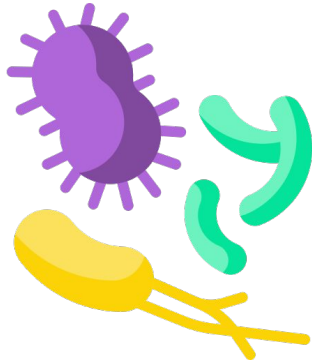
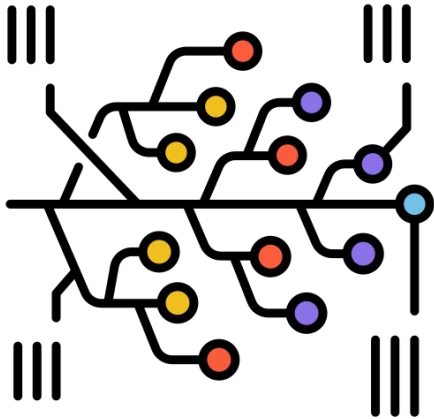
Helps to **identify open-source tools** that you will **incorporate in your pipeline**

May identify a dev solution that **already exists** to address your problem

# Selecting the Appropriate Tool for a Pipeline

**Define the problem** you are hoping to solve; highlight how your software will **help to inform public health decision-making**

- Be as **specific** as you can!
- Define how you will **measure the fit** of a software solution



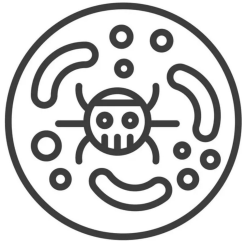
# Selecting the Appropriate Tool for a Pipeline

Read the literature to **define your options**

- Helpful to identify evidence of **public health application**

Seek **community feedback**

- StaPH-B, microBinfie, OAMD BRR/WFD
- **You are not the only one** facing this challenge



# Design Document

## Organization Specific Considerations

- Tailor your design document to your organization's processes
  - May be **additional details** that are helpful for your colleagues, e.g. state IT personnel may need to be involved to support some dev initiatives
- Quality Management Systems
  - **Defined testing and validation datasets** may be defined if you're updating an existing, validated pipeline





# Design Document

## Summary

- The design document is a vital tool that **defines the problem and the proposed solution**, informed by literature review and community feedback.
  - It ensures **clear communication** and alignment among stakeholders
- In writing a design document, you should be assessing other open-source solutions



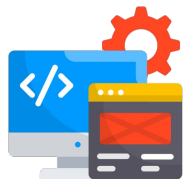
Seek tools that **align** with your project goals  
**Read the literature** and **engage the community**

## 2. Development Environment

# Development Environment

## Separating Prod from Dev Resources

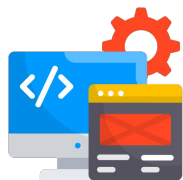
- Prevents **untested code** from affecting live production systems.
- Allows for **safer experimentation and testing**.
- Risk mitigation:
  - Isolates development activities, ensuring that any bugs or issues are **contained within the development environment**.



# Development Environment

## Maintaining Prod and Dev Environments

- Version Control Systems like Git help manage changes to source code over time
  - Can create **development copies** of the source code without modifying production source code
    - Developing in **forks or branches** of the production codebase



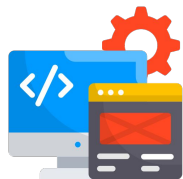
More on **Git** next week!

# Development Environment

## Maintaining Prod and Dev Environments

- Mimicking your production environment
  - VMs can be based off of **production images**
  - Readily deployable in cloud environments, can also be setup on local systems with software like **VirtualBox**

*In this context, "**production image**" refers to a snapshot or template of a system's state at a **particular point in time***



# Development Environment

## Example Configuration: Dev VMs for Terra Workflows

- Mimicking your production environment
  - **Prod:** GUI front-end giving access to GCP resources, WDL workflows, and Cromwell engine
  - **Dev:** CLI environment with access to GCP resources, WDL workflows and Cromwell (or miniWDL engine)
    - Working with a **fork/branch of production workflow**
    - Can also create a dev Terra workspace for further testing



# Dev Environment for Terra Workflows

```
▼ TERMINAL bash - public_health_bioinformatics + ▾ 🗑️  
● kevin_libuit@libuit-2023-dev-vm:~/public_health_bioinformatics$ git checkout kgl-theiaek-patch  
Switched to branch 'kgl-theiaek-patch'  
Your branch is up to date with 'origin/kgl-theiaek-patch'.  
○ kevin_libuit@libuit-2023-dev-vm:~/public_health_bioinformatics$ miniwdl run ./workflows/theiaek/w  
f_theiaek_illumina_pe.wdl -i theiaek-input.json
```

← Back to list

ⓘ TheiaEuk\_Illumina\_PE\_PHB

Version:

Source: [github.com/thelagen/public\\_health\\_bioinformatics/TheiaEuk\\_Illumina\\_PE\\_PHB:kgl-theiaek-patch](https://github.com/thelagen/public_health_bioinformatics/TheiaEuk_Illumina_PE_PHB:kgl-theiaek-patch)

Synopsis:

No documentation provided

☐ Run workflow with inputs defined by file paths

☒ Run workflow(s) with inputs defined by data table

Step 1

Select data table:

Step 2

No data selected

☒ Use call caching ⓘ ☐ Delete intermediate outputs ⓘ ☐ Use reference disks ⓘ ☐ Retry with more memory ⓘ

SCRIPT

..

INPUTS

..

OUTPUTS

..

## Terra Dev Environment:

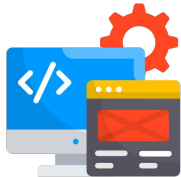
- **Copy of production code** through use of a Git branch
- **Mimicking prod environment**
  - CLI: miniWDL engine
  - Terra: Dev workspace

# Development Environment

## IDE v. Text Editors

- **Text Editor** - software application used for **editing plain text**
  - Example software: VIM, EMACS
- **IDEs** (Interactive Development Environments): software suite that offer a **GUI to various tools** that facilitate software development such as:
  - Code editing, debugging, and version control within a **single application**

Example software: Visual Studio and Atom

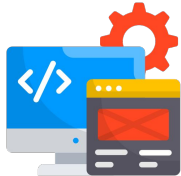




# Development Environment

## Advantages of IDE

- **Code navigation:** Enable developers to move through and understand structure of a large codebase efficiently
  - Particularly important with complex file systems
- **Error reduction:** Provide real-time syntax checking and debugging tools to catch errors early in development cycle
- **Resolving merge conflicts:** Provide visual comparisons and interactive editing to assist in resolving merge conflicts



**Note:** These features mean that IDEs tend to be **more resource-intensive** relative lightweight text editors like VIM

# IDE Example - Error reduction

```
output {  
  # Version Capture  
  String freyja_fastq_wf_version = version_capture.phb_version  
  String freyja_fastq_wf_analysis_date = version_capture.date  
  # Read QC - fastq_scan outputs  
  Int fastq_scan_num_reads_raw1 = read_QC_trim.fastq_scan_raw1  
  Int? fastq_scan_num_reads_raw2 = read_QC_trim.fastq_scan_raw2  
  String? fastq_scan_num_reads_raw_pairs = read_QC_trim.fastq_scan_raw_pairs  
  String? fastq_scan_version = read_QC_trim.fastq_scan_version  
  Int? fastq_scan_num_reads_clean1 = read_QC_trim.fastq_scan_clean1  
  Int? fastq_scan_num_reads_clean2 = read_QC_trim.fastq_scan_clean2  
  String? fastq_scan_num_reads_clean_pairs = read_QC_trim.fastq_scan_clean_pairs  
}
```

*Catching errors with VIM – **manual review**; post-hoc testing*

# IDE Example - Error reduction

```
50 ···output {  
51 ···  # Version Capture  
52 ···  String freyja_fastq_wf_version = version_capture.phb_version  
53 ···  String freyja_fastq_wf_analysis_date = version_capture.date  
54 ···  # Read QC -- fastq_scan outputs  
55 ···  Int fastq_scan_num_reads_raw1 = read_QC_trim.fastq_scan_raw1  
56 ···  Int? fastq_scan_num_reads_raw2 = read_QC_trim.fastq_scan_raw2  
57 ···  String? fastq_scan_num_reads_raw_pairs = read_QC_trim.fastq_scan_raw_pairs  
58 ···  String? fastq_scan_version = read_QC_trim.fastq_scan_version  
59 ···  Int? fastq_scan_num_reads_clean1 = read_QC_trim.fastq_scan_clean1  
60 ···  Int? fastq_scan_num_reads_clean2 = read_QC_trim.fastq_scan_clean2  
61 ···  String? fastq_scan_num_reads_clean_pairs = read_QC_trim.fastq_scan_clean_pairs
```

Catching errors with VSCode – **Error highlighting**; suggested fixes

# IDE Example - Error reduction

```
50 output {  
51   # Version Capture  
52   String freyja_fastq_wf_version  
53   String freyja_fastq_wf_analysis  
54   # Read QC -- fastq_scan outputs  
55   Int fastq_scan_num_reads_raw1 = read_QC_trim.fastq_scan_raw1  
56   Int? fastq_scan_num_reads_raw2 = read_QC_trim.fastq_scan_raw2  
57   String? fastq_scan_num_reads_raw_pairs = read_QC_trim.fastq_scan_raw_pairs  
58   String? fastq_scan_version = read_QC_trim.fastq_scan_version  
59   Int? fastq_scan_num_reads_clean1 = read_QC_trim.fastq_scan_clean1  
60   Int? fastq_scan_num_reads_clean2 = read_QC_trim.fastq_scan_clean2  
61   String? fastq_scan_num_reads_clean_pairs = read_QC_trim.fastq_scan_clean_pairs
```

Expected Int instead of Int? -- to coerce T? X into T, try  
select\_first([X,defaultValue]) or select\_first([X]) (which might fail at runtime); to  
coerce Array[T?] X into Array[T], try select\_all(X)

View Problem (Alt+F8) Quick Fix... (Ctrl+.)

*Catching errors with VSCode – Error highlighting; **suggested fixes***

# Development Environment

## Summary

- Separating development and production environments is crucial to **mitigate risks**
  - Strategies such as using **separate compute environments, version control systems, and mimicking prod environment configurations** help achieve this separation effectively.
- IDEs **can enhance development productivity** with features like code navigation, active error catching, and version control integration

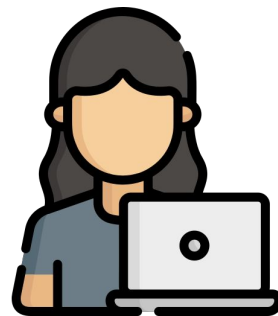
# Hands-On Exercise

# Software Development Practices

## Developer's Journey

**Week 1 Focus**

- 1. Design Document**
  - a. Clearly defining the problem and the proposed solution
- 2. Development Environment**
  - a. Separate from production
  - b. Text editors and IDE's
- 3. Making Source Code Modifications**
  - a. Small interactive changes (version control)
- 4. Peer Review**
  - a. Collaborative development teams
- 5. Bringing Changes into Production**
  - a. Final testing
  - b. Static version releases



# Exercise 01: Design Doc, Dev Environment, and Scripting with VSCode

## Exercise Goal

1. Review a design document for a development initiative
2. Access a development environment via GitPod
3. Use VSCode IDE to test code and script solution

