The Complexities of Replicability

I'm doing the same thing.
Why is it behaving differently?

Jason M. Gates

Department of Applied Mathematics @ CU Boulder — April 13, 2023

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2023-00327PE

Outline

- Getting to Know Each Other
- Defining Terms: The Three Rs
- Why Does It Matter?
- The Six Layers of Replicability
- Potential Problems
- Why Is This So Hard?
- Discussion

Getting to Know Each Other

Who am I? Who are you?

Getting to Know Each Other

Jason M. Gates

- Engineering physicist
  → computational mathematician
  → software engineer
  → DevOps evangelist
- University of Tulsa
  - MA: Applied Mathematics
  - BS: Engineering Physics
  - BS: Applied Mathematics
  - BA: German
- Colorado School of Mines
  - PhD: Mathematical and Computer Sciences (incomplete)
- Northrop Grumman Mission Systems
  - Automating installation / configuration of large-scale, distributed software
  - Real-time data processing algorithms
  - Evolutionary computation
- Sandia National Laboratories
  - Manufacturing solutions to nonlinear, coupled PDEs
  - Mathematical algorithms development
  - Software engineering best practices consultant
  - DevOps infrastructure consultant / team lead
  - Automated testing lead
  - Nicknames:
    - pipeline guru
    - git-fu master

Getting to Know Each Other

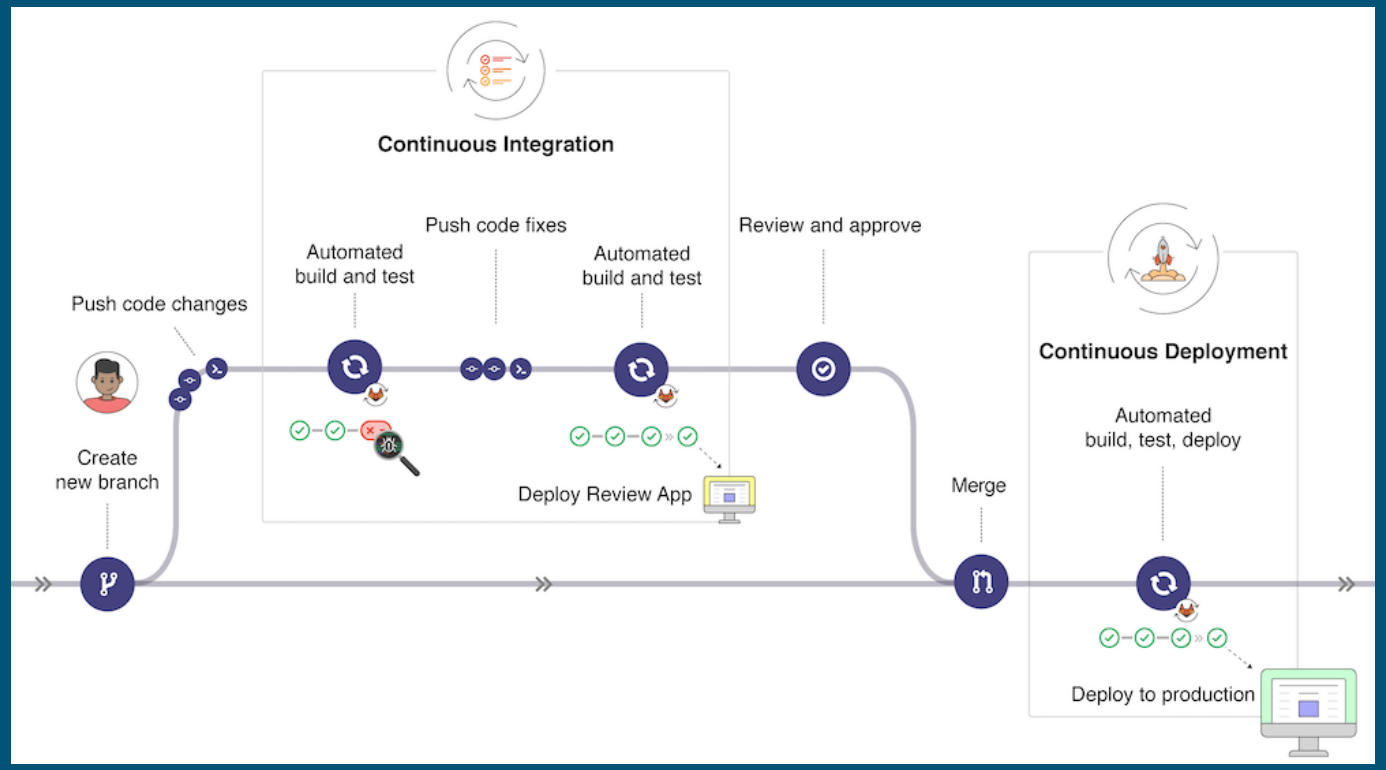
- What are you studying?
- What kind of programming do you do?
- What tools do you use?
- Do you like writing software, or do you see it as a chore that gets in the way of the research?
- Do you use version control?
- Are you familiar with the concept of continuous integration?

Background Information

- How you keep track of the work you do on the computer
- Work is saved in chunks (commits) into a repository
- Commit messages detail what changes were made and why
- Commits are organized into branches to keep streams of work separate
- Branches are integrated into the main development branch when work is complete
- Resources:
  - Getting Started—About Version Control | Pro Git, 2nd Edition
  - Foundations of Git—Certification Course | GitKraken

Background Information

- While working on a feature branch, you want to ensure your code still functions as expected with the new changes
- Need some way to continuously configure, build, test, install, run, etc., with each commit you make



- Resources:
  - CI/CD Concepts | GitLab
  - An Introduction to Continuous Integration, Delivery, and Deployment | DigitalOcean
  - What is Continuous Integration? | Atlassian

Defining Terms: The Three Rs

What is *replicability* anyway?

Defining Terms: The Three Rs

- Repeatability (Same team, same experimental setup): The measurement can be obtained with stated precision by the same team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same location on multiple trials. For computational experiments, this means that a researcher can reliably repeat her own computation.
- Replicability (Different team, same experimental setup): The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts.
- Reproducibility (Different team, different experimental setup): The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently.

Association for Computing Machinery (2020). *Artifact Review and Badging*. Available online at: https://www.acm.org/publications/policies/artifact-review-and-badging-current (Accessed 2023-01-09).

Defining Terms: The Three Rs

- Repeatability: I can do exactly what I did at some point in the past and achieve the same results.
- Replicability: Someone else can do exactly what I did at some point in the past, on the same or an equivalent machine, using the same tooling I used, and achieve the same results.
- Reproducibility: Someone else can achieve the same results I did at some point in the past, on an equivalent machine, using their own tooling.

Why Does It Matter?

What problem are we trying to solve, exactly?

Why Does It Matter?

- New reality: Most industries are software industries
  - Whether practitioners realize it or not
  - Whether they like it or not
- Pro: You can do a lot of good with software
- Con: You get all the baggage that comes with software
- Con: Being a domain expert is no longer sufficient—you need some software engineering expertise
- Pro: The software engineering side of things is more fun anyway 😄
- Good news: This talk will make you substantially more qualified than most of your peers for dealing with this reality

Why Does It Matter?

Don't Be This Guy

Why Does It Matter?

- Problems propagate through the three Rs.
- Generally you only have control over the first two.
- Solutions also tend to propagate through, so if you want to be good at reproducibility, you'll need to be good at repeatability and replicability as well.

The Six Layers of Replicability

How can we best think through the problem space?

The Six Layers of Replicability

6. Orchestration: The scripting that orchestrates running Layer 5 for all the supported configurations / environments / machines
5. Interface: A single approved way for all users / developers / CI services to interact with the code (e.g., clone, configure, build, test, package, install, deploy, run, etc.); the automation of all the lower layers
4. Configuration: The ability to consistently configure your code for all the supported environments on all the supported machines
3. Environment: The ability to consistently load blessed collections of software from Layer 2 on all your machines
2. Software: Any software installed on top of the base
1. Base: The base machine & OS setup for any machines you'll be running on

Layer 1: Base

Questions

- What collection of machines do you officially support?
- How are they provisioned?
- How do you guarantee consistency across them?

Practices

- Machine provisioning and setup is automated and version-controlled
- Monitoring alerts when machines get out of sync
- Everyone uses the supported machines

Tools

- Providers: OpenStack, AWS, GCP, Azure
- Provisioning: TerraForm, Vagrant, Pliant, Pulumi
- Setup: Ansible, Chef, Puppet, SaltStack, Packer
- Monitoring: Prometheus, Grafana, Sensu, SolarWinds

Layer 2: Software

Questions

- What software (compilers, libraries, tools) do you install on your supported machines?
- What versions do you support?
- How do you guarantee consistency across machines?
- Are there discrepancies between software supported on some machines vs others?

Tools

- Setup: Ansible, Chef, Puppet, SaltStack
- Package managers: yum/dnf, apt, Homebrew, conda/pip, yarn, Maven/Gradle, Spack
- Monitoring: Nagios, ninjaOne, N-sight, SolarWinds

Practices

- Software installation is automated and version-controlled
- Monitoring alerts when installed software gets out of sync
- Everyone uses the supported versions

Layer 3: Environment

Questions

- Which collections of software (environments / toolchains) will you support on which of your machines?
- How will users / developers / CI services load those supported environments?
- Are there discrepancies between supported environments on some machines vs others?

Practices

- Supported environment specifications are version-controlled and easy to read, use, maintain, and extend
- Use lock files for dependency management, where appropriate
- Everyone uses the supported environments, and loads them in the supported way

Tools

- Containerization: Docker, podman, Buildah, Dev Containers
- Environment Modules
- NixOS
- Custom scripting

Layer 4: Configuration

Questions

- Which collections of build- or run-time configuration flags will you support, for which of your environments, on which of your machines?
- How will users / developers / CI services specify those supported configurations at build- or run-time?
- Are there discrepancies between supported configurations across environments or machines?

Practices

- Supported configuration specifications are version-controlled and easy to read, use, maintain, and extend
- Everyone uses the supported configurations in the supported way

Tools

- Feature flags: LaunchDarkly, Flagship, Harness, CloudBees
- Custom scripting

Layer 5: Interface

Questions

- How will all users / developers / CI services interface with both your code and all the layers below?
- Can you capture exactly what was done, where, when, etc., for the sake of debugging or replicating?
- What flexibility needs to be built into this layer?

Practices

- Create a “one script to rule them all” as a single entry point for everyone interacting with the code
- Provide only the flexibility that is necessary, but no more
- Design this layer with Layer 6 in mind

Tools

- Build systems: Make, CMake, Maven/Gradle, esbuild, Pants
- Custom scripting

Layer 6: Orchestration

Questions

- How do you run everything from Layer 5 on all your supported machines, for all your supported environments, for all your supported configurations?
- How can you guarantee this doesn't diverge from Layer 5?
- What flexibility needs to be built into this layer?

Practices

- Design your Layer 6 scripting such that it mirrors that from Layer 5 at the high level
- Provide only the flexibility that is necessary, but no more

Tools

- Continuous integration: Jenkins, GitLab CI/CD, GitHub Actions, CircleCI, Travis CI, TeamCity
- Custom scripting

Potential Problems

What landmines do we need to watch out for?

Potential Problems

- Need clearly defined interfaces between the layers; otherwise:
  - Tendency toward highly-complected infrastructure (i.e., spaghetti code)
  - Debugging is much more difficult
  - As the infrastructure grows, maintenance and extension become exponentially more problematic
- Problems with repeatability and replicability can exist in all layers
  - Problems in lower layers are felt in upper layers
  - Attempting to handle problems in the wrong layer is a recipe for frustration
- Insufficient off-the-shelf tooling focused on replicability in Layers 3 – 6
  - Requires home-grown scripting
  - Assumes you really know what you're doing and will do a good job

The Six Layers

6. Orchestration
5. Interface
4. Configuration
3. Environment
2. Software
1. Base

Why Is This So Hard?

Shouldn’t this problem be solved already?

Why Is This So Hard?

- “Look kid, there’s a lot going on here that you don’t understand.” ~ Captain America
- Potential problems lurk all over the place
- There’s no silver bullet solution
- Doing this well requires:
  - A comprehensive high-level understanding of how all the puzzle pieces fit together
  - A good deal of sophisticated knowledge and experience across a handful of domains
  - Relentless adherence to software engineering best practices
- This kind of work tends to not get prioritized alongside the “real work”
  - Nobody cares until something breaks
  - There are seldom rewards for a job well done

Discussion

- What say ye?
- Were there any lightbulb moments for you?
- Is there anything you want to do as a result of this presentation?
- How can you engage with your university's IT personnel to work toward "replicability as a service"?