[CS3704] Software Engineering

Dr. Chris Brown

Virginia Tech

11/27/2023

Announcements

- **Advanced Topics in SE Discussions**
  - In-class Wednesday (11/29)
- **Retrospective and Automated SE workshop**
  - In-class Friday (12/1)
- **PM4**
  - Due 12/4 and 12/11
  - More details later

Deployment

DevOps
CI/CD
Deployment Strategies

Learning Outcomes

By the end of the course, students should be able to:

- **Understand software engineering processes, methods, and tools used in the software development life cycle (SDLC)**
- **Use techniques and processes to create and analyze requirements for an application**
- **Use techniques and processes to design a software system**
- **Identify processes, methods, and tools related to phases of the SDLC**
- **Explain the differences between software engineering processes**
- **Discuss research questions and current topics related to software engineering**
- **Create and communicate about the requirements and design of a software application**

Warm-Up

Stand-up Meeting

- What have you done
- What do you need to do
- What is blocking you

- Have you ever shared a program you developed with other users? How did it go? What was your experience

Deployment/Maintenance

Dp

Goal: release, upgrade, and fix the software

- <u>deployment:</u> delivery of software to users
  - When software is completed, it must be *deployed* or delivered to the customer. Additionally, software must be *maintained* such that user problems are addressed after operationalization.
- *Software Artifacts:* All!

Deployment

- The process of software (completely implemented or partially completed increment) being delivered to users.
- Since modern software processes are iterative in nature, deployment happens not once but multiple times as software moves towards completion.
- Three main activities:
  - Delivery
  - Support
  - Feedback

Deployment Principles

1. Customer expectations for software must be managed.
2. Complete delivery package should be assembled and tested.
3. A support regime must be established before the software is delivered.
4. Appropriate instructional material must be provided to end users.
5. Buggy software should be fixed first, delivered later

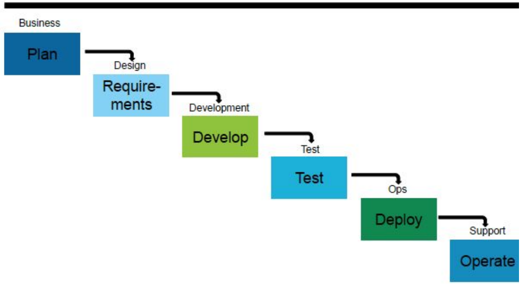
[Pressman]

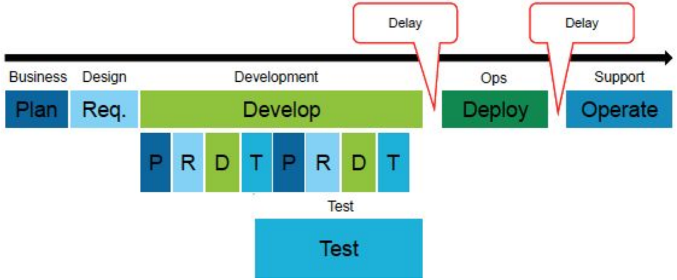
Deployment Problems

- Historically, the developers wrote the code, testers validated the program, and a separate IT operations team was in charge of monitoring and deploying the system.
  - Some teams still utilize separate groups.

DevOps

- Set of practices that combines software development (Dev) and IT operations (Ops) team responsibilities.
  - Concept informally originated in the 1990s; Motivated by agile methodologies
    - Teams couldn’t satisfy agile/XP goals



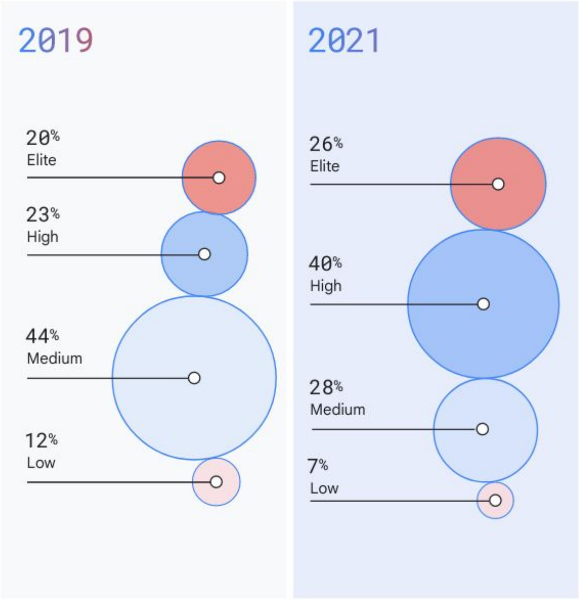


DevOps (cont.)

Values:

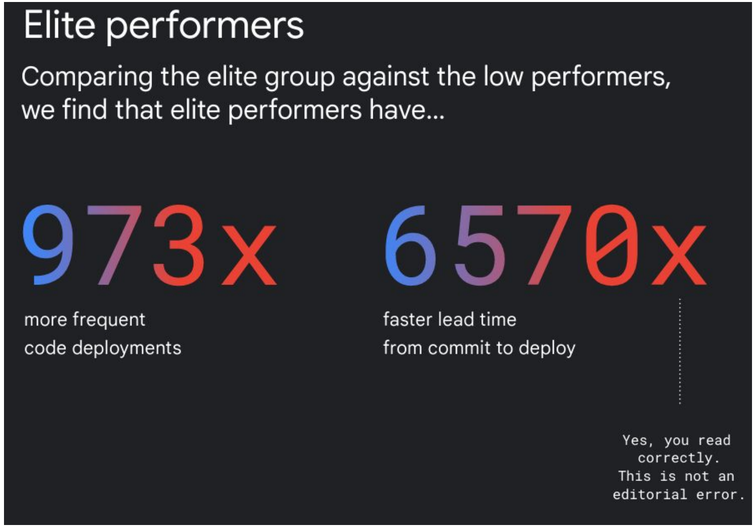
- No silos, no walls – no responsibility pipelines
- One team owning all changes for the lifetime of the project
- *You* are the support person
- Automate as much as possible

State of DevOps 2021



Elite: Multiple deployments per day; < 1 hour lead time
High: Weekly to Monthly deployments; day-week lead time
Medium: Monthly to 6-month deployments; month-six month
Low: Longer than six months; more than six month lead time

- *Lead time:* Time from code committed to running in production



Elite performers

Comparing the elite group against the low performers, we find that elite performers have...

973x more frequent code deployments

6570x faster lead time from commit to deploy

Yes, you read correctly. This is not an editorial error.

DevOps Toolchains

- **Toolchain:** a set of multiple tools to accomplish varying tasks as opposed to just one.
- **Pipeline:** a series of steps that must be performed in order to deliver a new version of software.
  - Will differ based on your team and/or company
- DevOps automation pipeline example:

Build

Test

Deploy

Run

Monitor

Manage

Notify

Key Deployment Concepts

- Staging
- Building
- CI/CD
- Test Automation

Staging

The process of building confidence in a proposed deployment candidate in the context of a *production-like* environment.

Waterfall Staging

A series of more and more expensive testing performed at each phase. *Can be hours or weeks...*

Build

A version of the software in a pre-release format only used by the development team.

- Failed builds indicate issue(s) with the program

Benefits:

- Minimizes integration risk
- Reduces the risk of low quality
- Supports easier defect diagnosis
- Improves morale

Build Activities

Activities for building software include but are not limited to:

- running unit tests, checking code style, executing integration/regression tests, static analysis, automated configuration checks,...

Integration

- **Integration:** merging changes with the source code repository
    - Broken integration occurs when code does not build, shared components work in one branch but not another, unit tests fail, etc.
- Manual Integration is expensive
    - Building, testing, and deploy can take hours or days…
    - Integration problems and bugs are detected too late.
    - **Operation locks:** manually locking down part of the production system while performing deployments

Continuous Integration

- A practice where developers automatically build, test, and analyze a software change in response to *every software change* committed to the source repository. A continuous integration server monitors the status of every commit and reports any problems.
- Every change in version control triggers the pipeline.

TODO: Discuss your experience with continuous integration or DevOps concepts.

Continuous Integration (cont.)

Motivation: Modules that worked individually were put together, and the whole system usually failed in ways that were infuriatingly difficult to find.

Yet in the last few years, integration has largely vanished as a source of pain for projects, diminishing to a nonevent.

- *2003:* 27% of companies ran daily builds
- *2013:* 69% have automated builds, 57% use CI tools [Hilton]

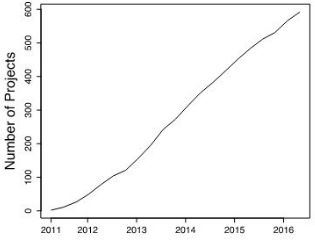


Figure 2: Number of projects using CI over time.

Continuous Integration (cont.)

Timeline

(1999) Extreme Programming (XP) rule: “Integrate Often”, “10-Minute Build”

(2000) Martin Fowler “Continuous Integration” blog post.

(2001) CruiseControl

(2003) IEEE Software, “22–35% companies surveyed run daily builds.”

(2005) Hudson/Jenkins

(2011) Travis CI

(2013) State of Agile, “57% companies surveyed use continuous integration tools.”

(2015) Spinnaker

(2016) Jenkins 2 “pipeline-as-code”

(2017) GitOps

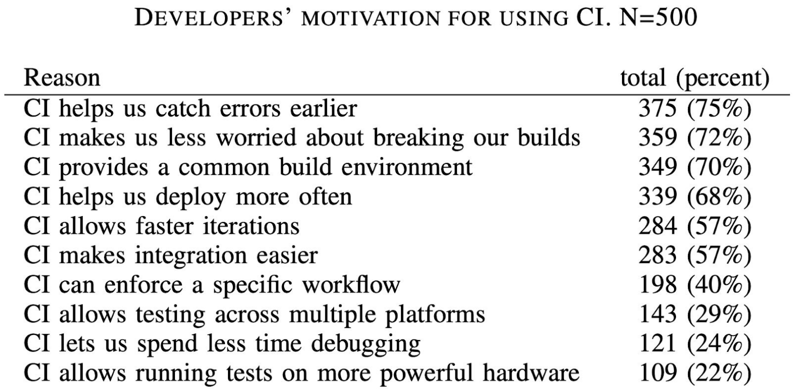
Continuous Integration Principles

1. Commit code frequently
2. Don’t commit broken code
3. Fix broken builds immediately
4. Write automated developer tools
5. All tests and inspections must pass
6. Run private builds
7. Avoid getting broken code

Continuous Integration Benefits

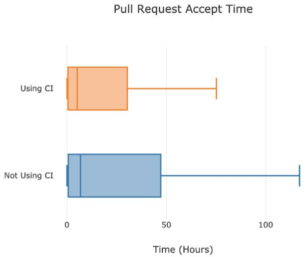
Benefits

- Detect defects and fix them earlier and faster
- Reduce repetitive processes (saves time, cost, and effort)
- Can release deployable software at *any time*
- Global mechanism for feedback
- Increases developers confidence about changes.
- Low-risk releases
- Happier teams



DEVELOPERS’ MOTIVATION FOR USING CI. N=500

Reason	total (percent)
CI helps us catch errors earlier	375 (75%)
CI makes us less worried about breaking our builds	359 (72%)
CI provides a common build environment	349 (70%)
CI helps us deploy more often	339 (68%)
CI allows faster iterations	284 (57%)
CI makes integration easier	283 (57%)
CI can enforce a specific workflow	198 (40%)
CI allows testing across multiple platforms	143 (29%)
CI lets us spend less time debugging	121 (24%)
CI allows running tests on more powerful hardware	109 (22%)



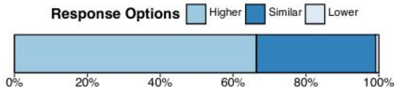


Fig. 2. Do projects with CI have (higher/similar/lower) test quality? N=444

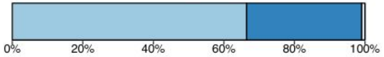


Fig. 3. Do projects with CI have (higher/similar/lower) code quality? N=438

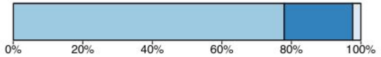


Fig. 4. Are developers on projects with CI (more/similar/less) productive? N=462

[Hilton]

Continuous Integration Risks/Pains

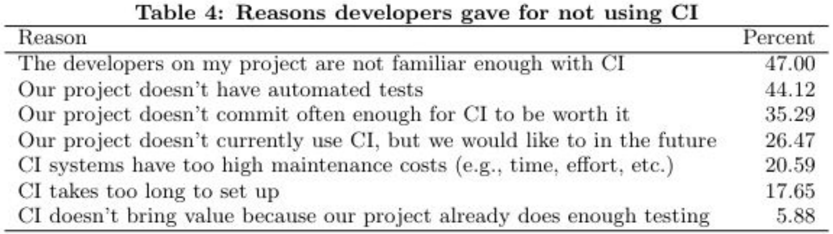
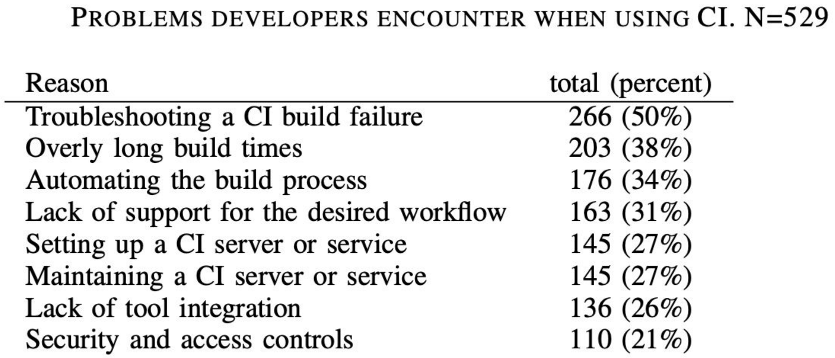


Table 4: Reasons developers gave for not using CI

Reason	Percent
The developers on my project are not familiar enough with CI	47.00
Our project doesn't have automated tests	44.12
Our project doesn't commit often enough for CI to be worth it	35.29
Our project doesn't currently use CI, but we would like to in the future	26.47
CI systems have too high maintenance costs (e.g., time, effort, etc.)	20.59
CI takes too long to set up	17.65
CI doesn't bring value because our project already does enough testing	5.88



PROBLEMS DEVELOPERS ENCOUNTER WHEN USING CI. N=529

Reason	total (percent)
Troubleshooting a CI build failure	266 (50%)
Overly long build times	203 (38%)
Automating the build process	176 (34%)
Lack of support for the desired workflow	163 (31%)
Setting up a CI server or service	145 (27%)
Maintaining a CI server or service	145 (27%)
Lack of tool integration	136 (26%)
Security and access controls	110 (21%)

"We find that...**larger projects** may outgrow Travis, and projects built with **C# and other special purpose languages** tend to be more likely to abandon Travis, perhaps due to a **contextual mismatch**" [Widder]

Risks

- Increased overhead in maintaining the CI system
- Too much change to transition from legacy system or no CI to new CI system
- Too many failed builds
- Additional hardware/software costs
- Developers should be performing these activities anyway

Continuous Delivery

- The concept of ensuring software changes (new features, configuration changes, bug fixes, etc.) can be *reliably* and *quickly* released to production environments.
    - Often paired with continuous integration (CI/CD)
- In the past, more frequent deployments meant lower quality software. Employing CI/CD practices allows for more deployments with *<u>higher</u>* reliability and stability of applications.

Continuous Delivery vs. Deployment

- ***Continuous delivery*** focuses on automation steps and strategies to safely prepare changes for production.
  - i.e. releases, versioning, etc.
  - Keep the code in a deployable state at any given time
- ***Continuous deployment*** focuses on the actual deployment across environments and distribution of the application.
  - “Distribution of bits to users” (i.e. packaging)

Test Automation

- The rise of agile practices led to more frequent customer feedback.
- Too many tests were a liability
  - More difficult to update, organize, and maintain
  - Slow tests slow everyone and everything down
- **Test automation:** the practice of automatically reviewing and validating software products

Test Automation (cont.)

Especially integration tests!

- *Reminder:* Testing groups of subsystems, and eventually the entire system, to uncover errors associated with interfacing between units.
- Lengthy integration phases before delivering the software internally (to test teams) or externally (customer), during which literally nothing got done.

Advanced Testing for Deployment

In addition to adopting testing tools and frameworks to automatically run tests, these techniques help improve testing practices in CI/CD processes.

- Mutation Testing*
- Fuzz Testing*
- Chaos Engineering
- Test Case Prioritization
- Test Case Generation

* **Fault injection testing:** understanding how the system behaves when stressed in unusual ways

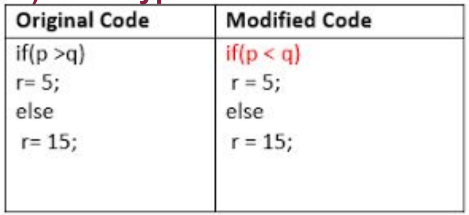
Mutation Testing

Mutation testing is a form of whitebox testing that involves modifying the program in small ways with *mutants*, causing the behavior of the original program to change. Unit test cases should be able to detect program behavior and fail, killing the mutants. Usually with *compile-time injections*.

Example mutant operators:

- Statement deletion (removing line(s) of code)
- Statement duplication or insertion (adding line(s) of code)
- Replacement of boolean expressions (switching True and False values)
- Replacement of arithmetic operators (i.e. +, -, *, /, %)
- Replacement of boolean operators (i.e. >, <, >=, <=, ==, !=)
- Replacement of variables within the same scope
- and more!

Ex) What type of mutation is this?

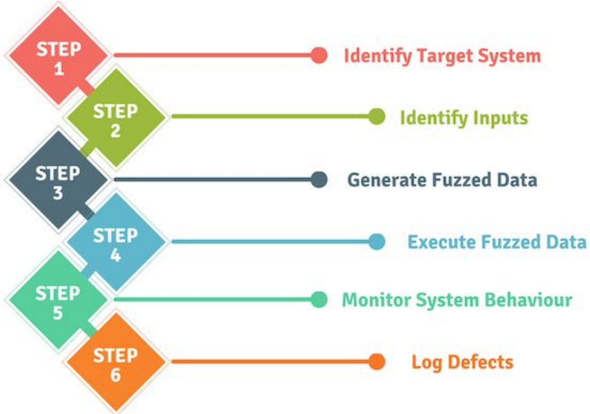


Original Code	Modified Code

if(p >q)
r= 5;
else
r= 15;
	if(p < q)
	r = 5;
	else
	r = 15;

Fuzz Testing

- **Fuzzing:** The process of injecting invalid, malformed, or unexpected inputs into a system to reveal defects and vulnerabilities.
- Fuzz testing automates this ^ to identify failures.
    - Heavily used for security
    - *Runtime injection*



Chaos Engineering

- The concept of building software to withstand unexpected or chaotic conditions.
    - Intentionally introducing harm to the system to see how it reacts (hardware and software).
    - *“Imagine a monkey (or group of monkeys) gained access to your system, data center, etc.”*

Test Case Prioritization

- As the size of code bases increases, test suites also grow bigger.
    - Harder to maintain
    - More time to run
- Selecting test cases to run in a test suite on the basis of different factors
    - During a build, only run important test cases
    - Can prioritize based on code coverage, features, risk, functionality, etc.

Test Case Generation

What if we could automatically generate test cases?

- Ongoing work in research and industry, beyond the scope of this class…

Deployment Strategies

With the rise of iterative processes and DevOps concepts, how is modern software actually deployed now?

- Basic deployment
- Rolling deployment
- Blue-green deployment
- Canary deployment
- A/B Testing
- Shadow Deployment/Dark Launch

[Harness]

Basic Deployment

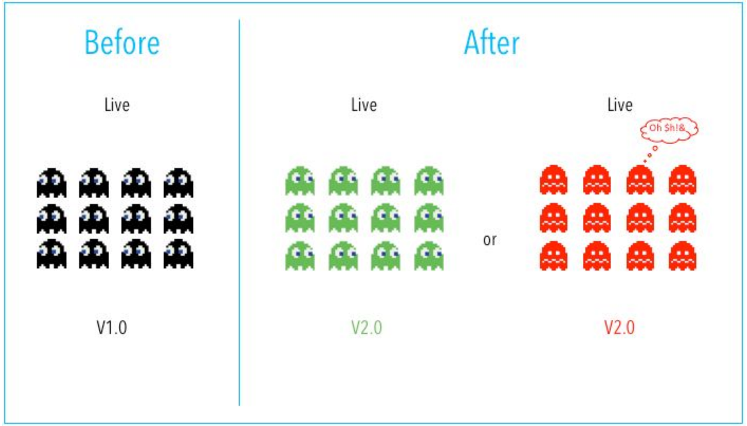
- Target environment is updated at the same time.

Pros:

- Simple, fast, and cheap
- Should only be used if your product is not critical or deploying to a lower environment

Cons:

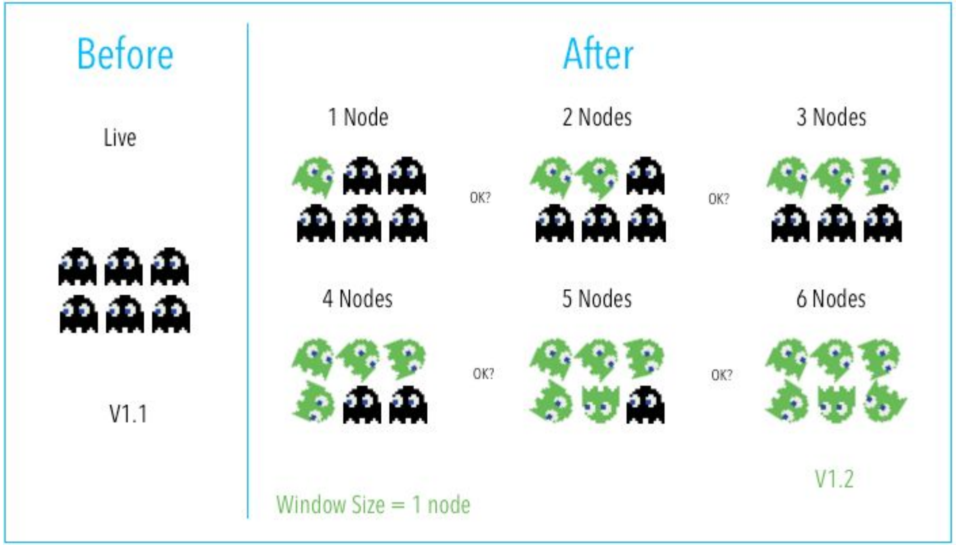
- Riskiest deployment strategy
- Difficult for rollbacks
- Not outage-proof



Rolling Deployment

- Target environments are incrementally updated in *N* batches.

+ Simpler roll back, less risk
- Requires support for old and new versions
- Slow



Blue-Green Deployment

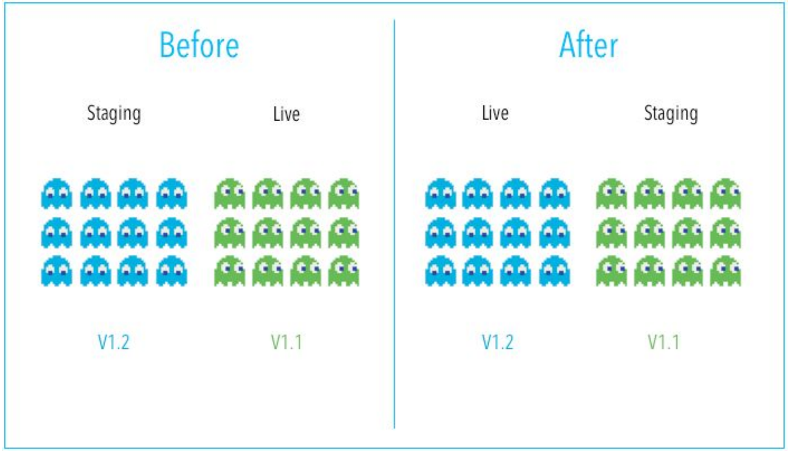
- Utilize two identical environments, i.e. *blue* for staging and *green* for production. Testing and validation testing can done in blue environment while user traffic is shifted to green. Then switch after successful testing and deployment.

  + Simple, easy to implement, less risk

  + Straightforward roll backs (just flip to other environment)

  - More cost to replicate production environment

  - Shifts all user traffic at once

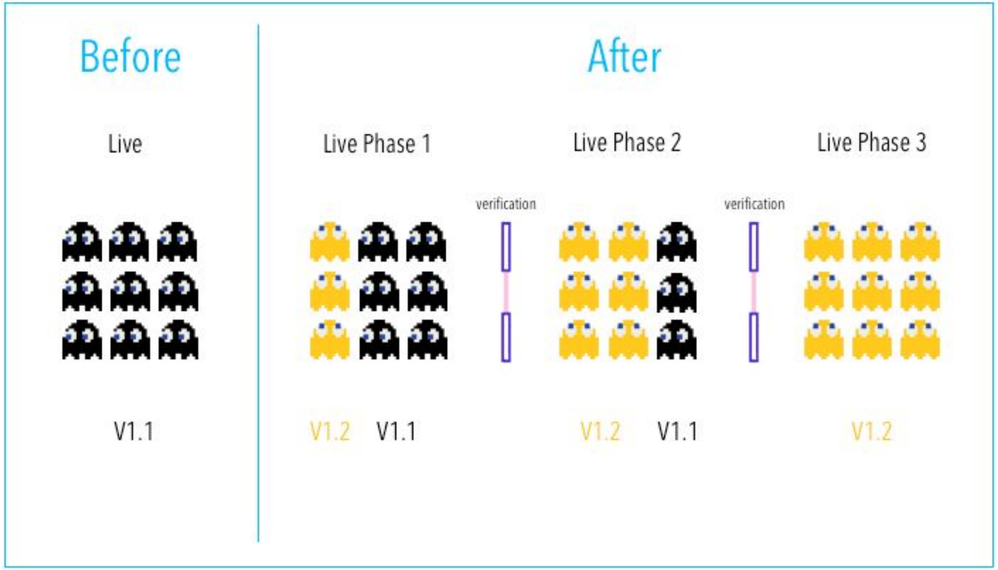


Canary Deployment

- Release an application incrementally to a subset of users in small phases.

+ Lowest risk!
+ Fast and safe roll backs
+ Cheaper than blue-green
- Testing in production needed

Note: Canary deployments got their name from an old British mining practice. Back in the day, miners used canaries to test coal mines' safety before they went in. If the canaries returned unharmed, the miners felt safe to enter. However, if something did happen to the birds, they knew that the mines were filled with toxic gases.



A/B Testing

- Run different versions of the same application simultaneously in the same environment for a period of time.
  - While other deployment strategies have an immediate goal of updating all nodes with a specific version, A/B testing is more focused on experimentation, exploration, and testing ideas.

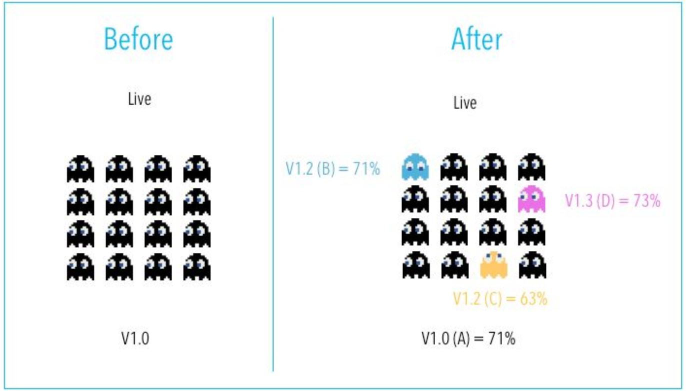
+ Easy and cheap to test new features

- Experimental nature

- Can be complex to automate

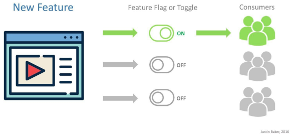
Different from alpha and beta testing!!!

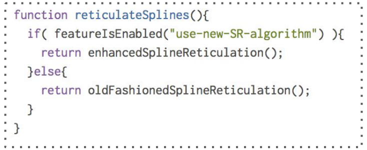
A/B Testing (cont.)



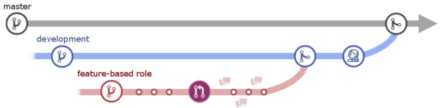
How do you run different versions of a program simultaneously? Route user traffic to specific versions based on a given criteria to perform measurements or observe using:

- Feature flags/toggles
- Branches
- Other A/B testing tools





```
function reticulateSplines(){
  if( featureIsEnabled("use-new-SR-algorithm") ){
    return enhancedSplineReticulation();
  }else{
    return oldFashionedSplineReticulation();
  }
}
```



Dark Launching

- The feature is available on the application, but not actually shown in the user interface.
    - Purpose is to monitor the load of the system with the new feature during day-to-day interactions.

+ Simple to implement
+ Tests load on system without production impact
- No user interaction

Example at Facebook:

“The secret for going from zero to seventy million users overnight is to avoid doing it all in one fell swoop. We chose to simulate the impact of many real users hitting many machines by means of a "dark launch" period in which Facebook pages would make connections to the chat servers, query for presence information and simulate message sends without a single UI element drawn on the page. With the "dark launch" bugs fixed, we hope that you enjoy Facebook Chat now that the UI lights have been turned on.” [Obasanjo]

Which Strategy Should You Use?

It depends…

- Company/team
- Programming language
- Type of system and usage
- …

Considerations:

- Minimize downtime
- Manage and resolve incidence with minimal user impact
- Effectively address failed deployments, roll back
- Reduce people and process errors
- Achieve predictable and repeatable deployments

Project Deployment

- You *are not* required to deploy your project for the demo.
- You *are* required to describe how you would maintain and deploy your project based on the content from the last two lectures in your final report!
- *Want to deploy your project on the cloud?*
    - Use https://cloud.cs.vt.edu/

      Get started: https://wiki.cs.vt.edu/index.php/Cloud_Quickstart

Project Update

- **PM4**
  - Divided into two parts
    1. **PM4.1 (due 12/4 before class)**
       - Final presentation slides
    2. **PM4.2 (due 12/11 at 11:59pm)**
       - Final project report
       - Black box test plan
       - Project retrospective survey (individual)

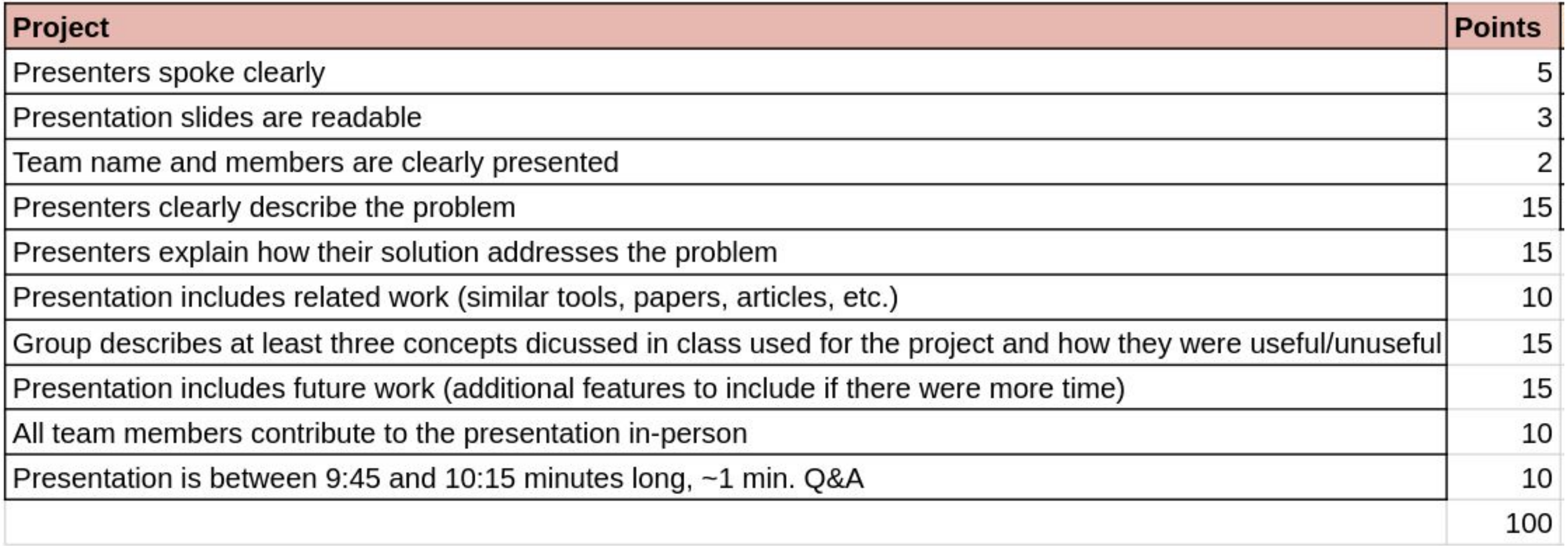
Final Presentations

Final project presentations will be in-class on the following dates:

- Monday (12/4) 4 groups
- Wednesday (12/6) 4 groups
- Monday (12/11, 3:25-5:25) 11 groups

Presentation times will be randomly selected. The presentation slides are due for all groups on 12/4 at 1:25pm.

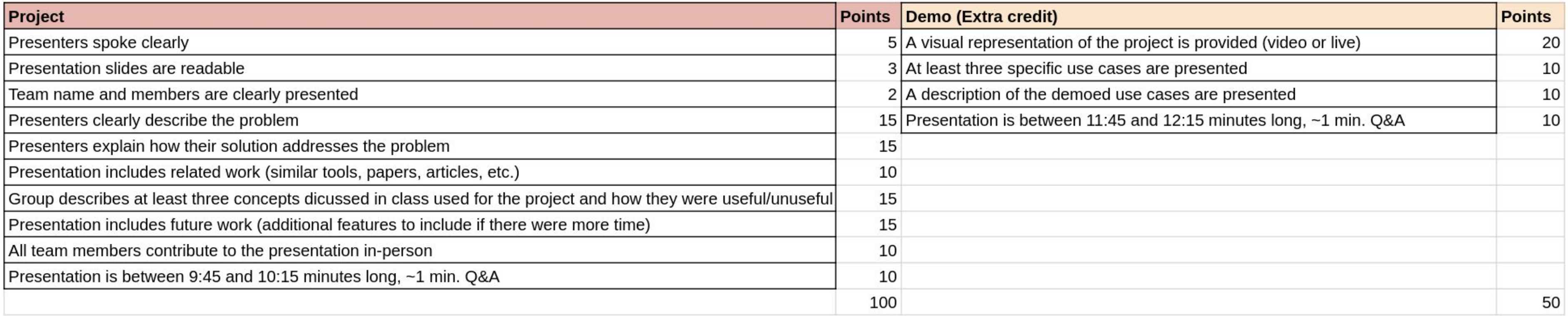
Final Presentation Rubric



Project	Points
Presenters spoke clearly	5
Presentation slides are readable	3
Team name and members are clearly presented	2
Presenters clearly describe the problem	15
Presenters explain how their solution addresses the problem	15
Presentation includes related work (similar tools, papers, articles, etc.)	10
Group describes at least three concepts dicussed in class used for the project and how they were useful/unuseful	15
Presentation includes future work (additional features to include if there were more time)	15
All team members contribute to the presentation in-person	10
Presentation is between 9:45 and 10:15 minutes long, ~1 min. Q&A	10
	100

Final Presentation Rubric (cont.)

- For groups that reported they plan to implement (i.e. write code) and demo use cases for your project to obtain extra credit:
  - CheckMate, BHADS, Slackin’ Off, Raghav and Sons, Seven, X-Coders, Clash of Clans(CoC)



Project	Points	Demo (Extra credit)	Points
Presenters spoke clearly	5	A visual representation of the project is provided (video or live)	20
Presentation slides are readable	3	At least three specific use cases are presented	10
Team name and members are clearly presented	2	A description of the demoed use cases are presented	10
Presenters clearly describe the problem	15	Presentation is between 11:45 and 12:15 minutes long, ~1 min. Q&A	10
Presenters explain how their solution addresses the problem	15		
Presentation includes related work (similar tools, papers, articles, etc.)	10		
Group describes at least three concepts dicussed in class used for the project and how they were useful/unuseful	15		
Presentation includes future work (additional features to include if there were more time)	15		
All team members contribute to the presentation in-person	10		
Presentation is between 9:45 and 10:15 minutes long, ~1 min. Q&A	10		
	100		50

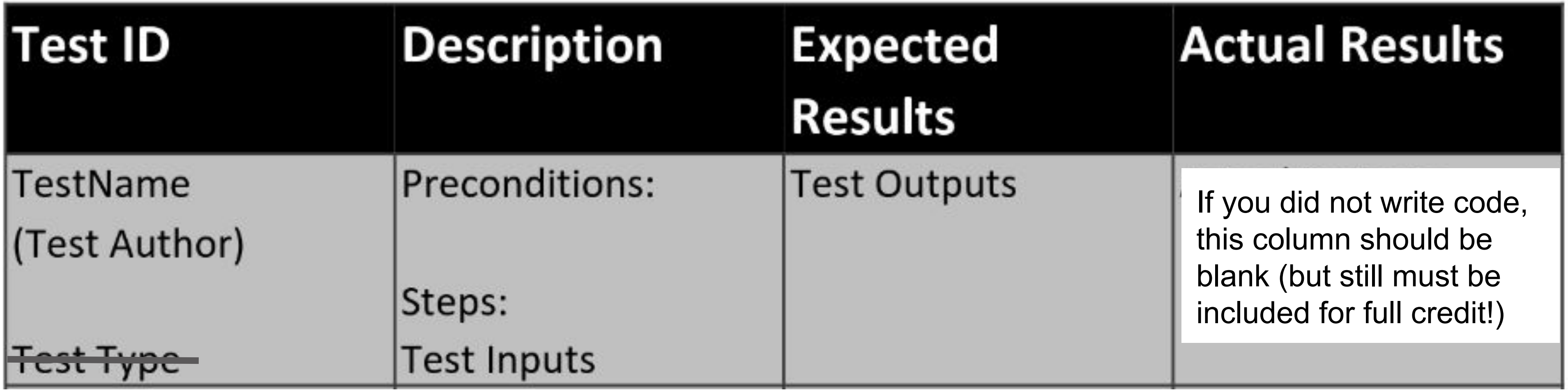
PM4.2 Submission

The remaining milestone materials (except the retrospective survey) will be submitted as a GitHub repository link on Canvas. The repo should include:

- The final report and black box test plan
- All of the previous project milestone artifacts
- Any other project management resources (i.e. issues, projects, etc.)
- Project code, if applicable.
- A relevant name and description for your project.
- A README file with an overview of the project and group members (and how to run your code, if applicable).

Black Box Test Plan

Plan should include 10 unique black box tests to test your project.



Test ID	Description	Expected Results	Actual Results
TestName (Test Author) ~~Test Type~~	Preconditions: Steps: Test Inputs	Test Outputs	

If you did not write code, this column should be blank (but still must be included for full credit!)

Final Report Rubric

Team	Points
Proposal is clearly written and organized	5
Spelling, grammar, etc.	5
Relevant title and team members are clearly presented	5
Proposal is in the correct paper format	5
Abstract provides overview of the project	5
Introduction explains the problem and motivates proposed solution	10
Related work section discusses existing related tools and/or research	10
Section describing high-level design decisions, implementation processes, and testing approach	20
Section describing how you would *deploy and maintain* your project using concepts discussed in class	20
Conclusion is included providing limitations and future work for the project	10
References are included	5
	100

Next Time…

- Discussion Presentations on *Advanced Topics in SE* [11/19]
- Class Retrospective, Automated SE Workshop on 12/1
  - This is <u>not</u> the Project Retrospective survey!
- Project Milestone 4
  - **Final Presentations in class on 12/4, 12/6, and 12/11**
    - **Slides due *before class* next week on 12/4!**
  - **Remaining items due 12/11 at 11:59pm**

References

- RS Pressman. *“Software engineering: a practitioner's approach”.*
- Smith et al. *“State of DevOps 2021”*. Google Cloud
- Harness. *“Intro to Deployment Strategies: Blue-Green, Canary, and More”*.
- Michael Hilton, et al. *“Continuous Integration (CI) Needs and Wishes for Developers of Proprietary Code”*. 2016
- Michael Hilton, et al. *“Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects”*. 2016
- Dare Obasanjo. *“Dark Launches, Gradual Ramps and Isolation: Testing the Scalability of New Features on your Web Site”*. 2008
- David Gray Widder, et al. *“I’m Leaving You, Travis: A Continuous Integration Breakup Story”*. 2018
- Chris Parnin