Continuous Integration, Delivery, & Deployment

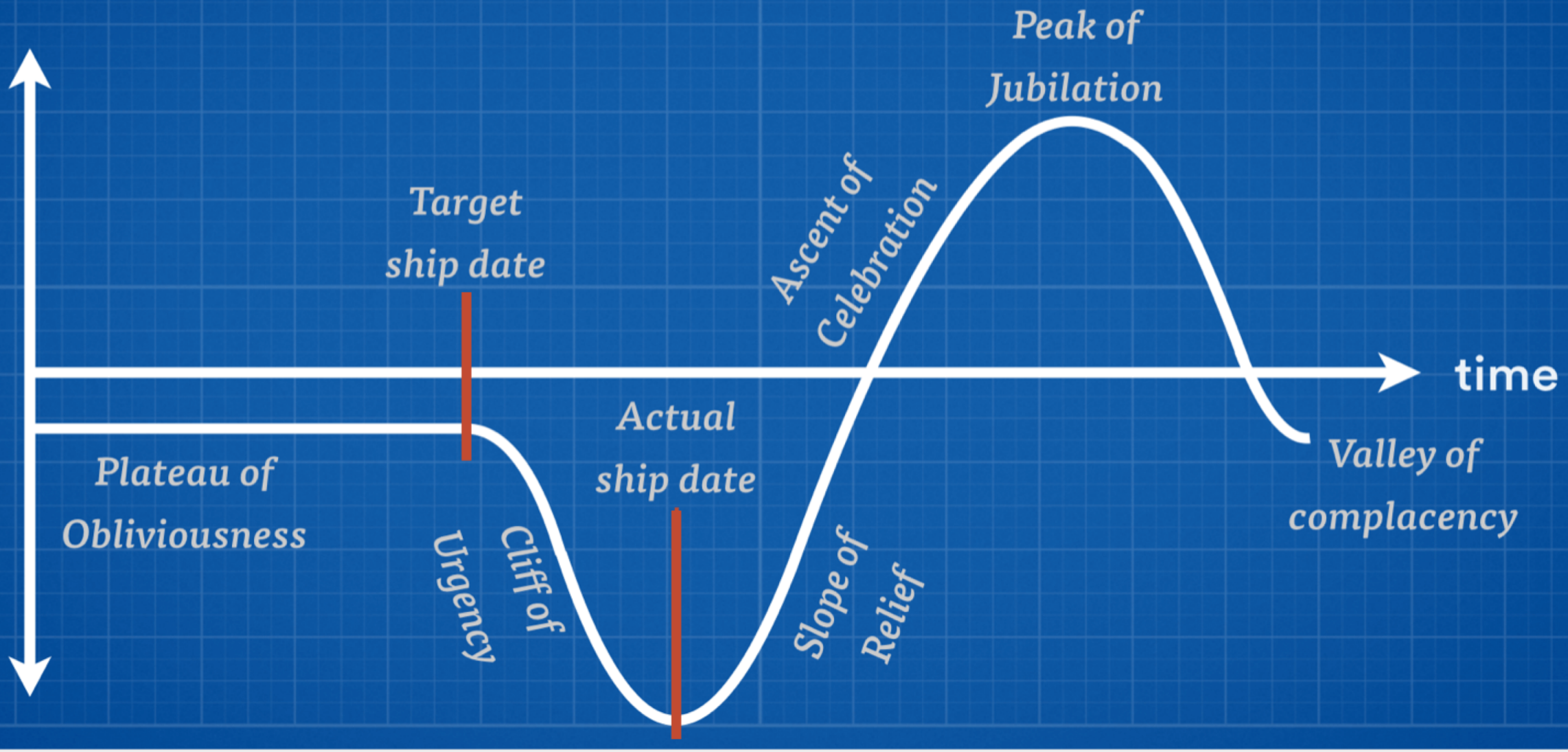
Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

Challenges with Traditional Release Method

- Servers must be set up by IT (sometimes manually)
- Third-party software such as application server, etc. must be installed.
- The software artifacts such as EAR or WAR must be copied to the production host.
- Application configuration must be copied or created.
- Finally any reference data needed must be copied over.
- As you can see there are lot of places where things can go wrong.
- With this process, the day of a software release tends to be a tense one.

Emotional cycle of manual delivery



Continuous Integration

- **Continuous integration** (**CI**) is the practice of merging all developer working copies to a shared mainline several times a day.
- The main aim of CI is to prevent integration problems, referred to as "integration hell".
- CI is intended to be used in combination with automated unit tests written through the practices of test-driven development.
- In addition to automated unit tests, organizations using CI typically use a build server to implement continuous processes of applying quality control in general — small pieces of effort, applied frequently. In addition to running the unit and integration tests, such processes run additional static and dynamic tests, measure and profile performance, extract and format documentation from the source code and facilitate manual QA processes.
- This continuous application of quality control aims to improve the quality of software, and to reduce the time taken to deliver it, by replacing the traditional practice of applying quality control *after* completing all development.

CI – Best Practices

- Continuous integration – the practice of frequently integrating one's new or changed code with the existing code repository – should occur frequently enough that no intervening window remains between commit and build, and such that no errors can arise without developers noticing them and correcting them immediately.
- Normal practice is to trigger these builds by every commit to a repository, rather than a periodically scheduled build.
- The practicalities of doing this in a multi-developer environment of rapid commits are such that it is usual to trigger a short time after each commit, then to start a build when either this timer expires, or after a rather longer interval since the last build.
- Many automated tools offer this scheduling automatically.

Continuous Integration Principles

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Every commit (to baseline) should be built
- Keep the build fast
- Test in a clone of the production environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

CI Benefits

- Bugs are detected early
- Avoids last-minute chaos at release dates
- When unit tests fail or a bug emerges, if developers need to revert the codebase to a bug-free state without debugging, only a small number of changes are lost (because integration happens frequently).
- Constant availability of a "current" build for testing, demo, or release purposes
- Frequent code check-in pushes developers to create modular, less complex code

What is Continuous Deployment?

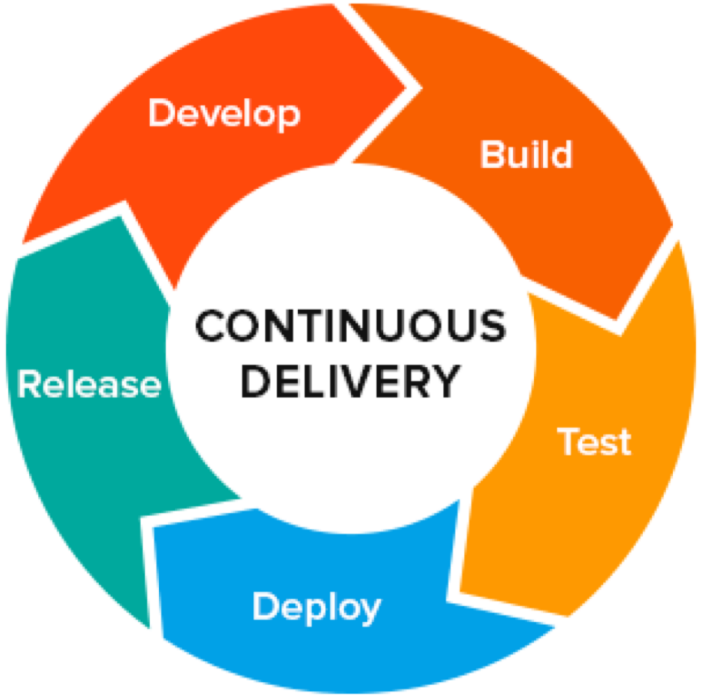
- Continuous Deployment is a software development practice in which every code change goes through the entire pipeline and is put into production, automatically, resulting in many production deployments every day.
- With Continuous Delivery your software is always release-ready, yet the timing of when to push it into production is a business decision, and so the final deployment is a manual step.
- With Continuous Deployment, any updated working version of the application is automatically pushed to production.
- Continuous Deployment mandates Continuous Delivery, but the opposite is not required.

CONTINUOUS DELIVERY

CONTINUOUS DEPLOYMENT

Continuous Delivery

- Continuous delivery is a DevOps software development practice where code changes are automatically built, tested, and prepared for a release to production.
- It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage.
- When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.



Continuous Delivery (contd.)

- With continuous delivery, every code change is built, tested, and then pushed to a non-production testing or staging environment.
- There can be multiple, parallel test stages before a production deployment.
- In the last step, the developer approves the update to production when they are ready.

Enabling Continuous Delivery

- Automate
  - The build, deploy, test, and release process must be automated so that it is repeatable.
- Frequent
  - Releases must be frequent.
  - The delta between releases will be small.
  - This significantly reduces the risk associated with releasing and makes it much easier to roll back.

Continuous Deployment

In continuous deployment push to production happens automatically without explicit approval.

Additional Resources

https://csye6225.cloud/