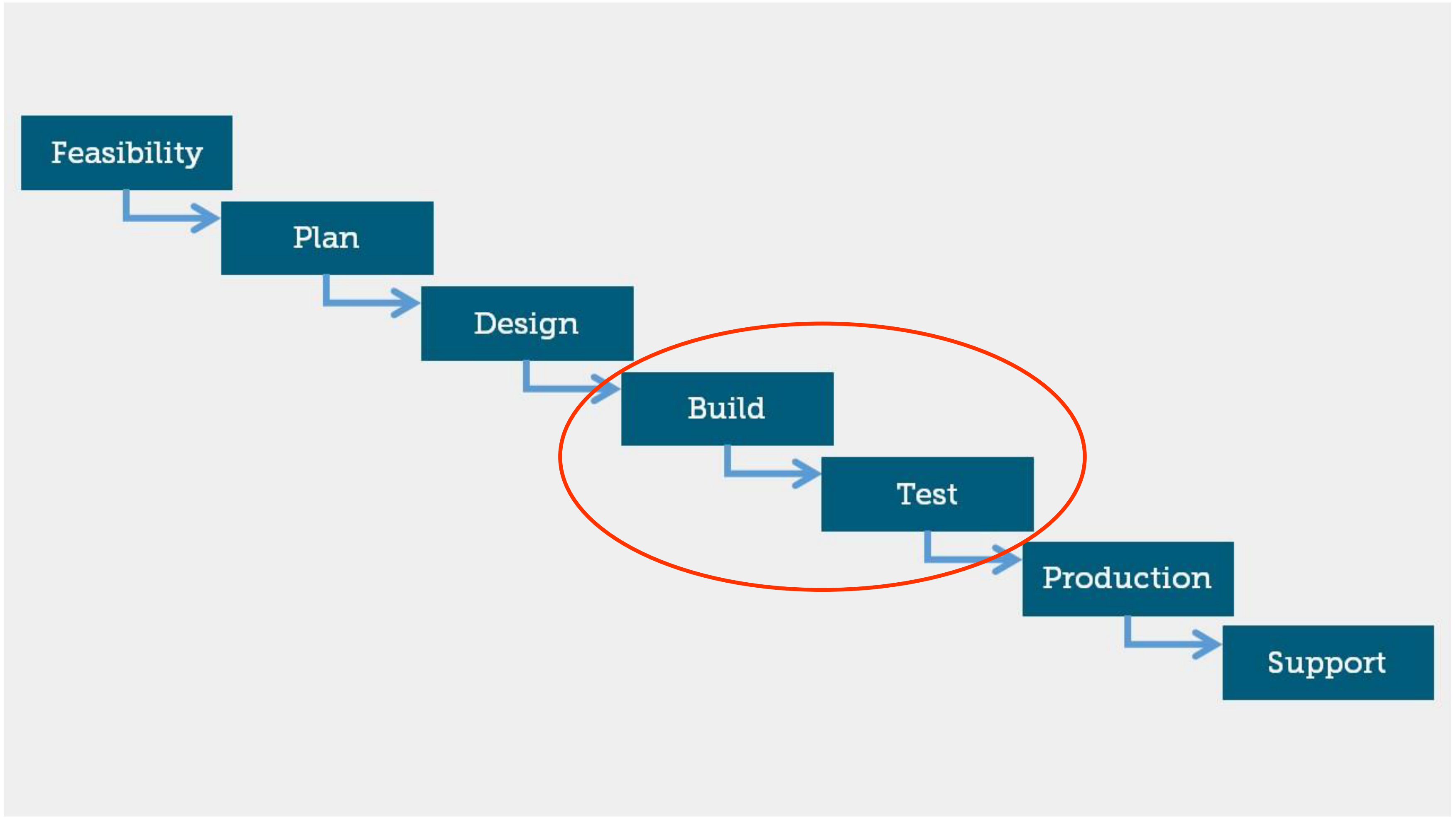
CI/CD, DOCUMENTATION

WEEK 10 – 10/28/2019

ANNOUNCEMENTS

- Homework 2 is due this Friday, Nov 1, 11:59 p.m.
- Project Milestone 3 is now due on Nov 3 11:59 p.m.
- Homework 3 will be released this Friday. Due on Nov 15, 11:59 p.m.
- Midterm grading is now complete.
- SQL will be a part of midterm 2 as well

FOLLOWING THE WATERFALL METHODOLOGY



TRADITIONAL “WATERFALL” TESTING PRACTICES

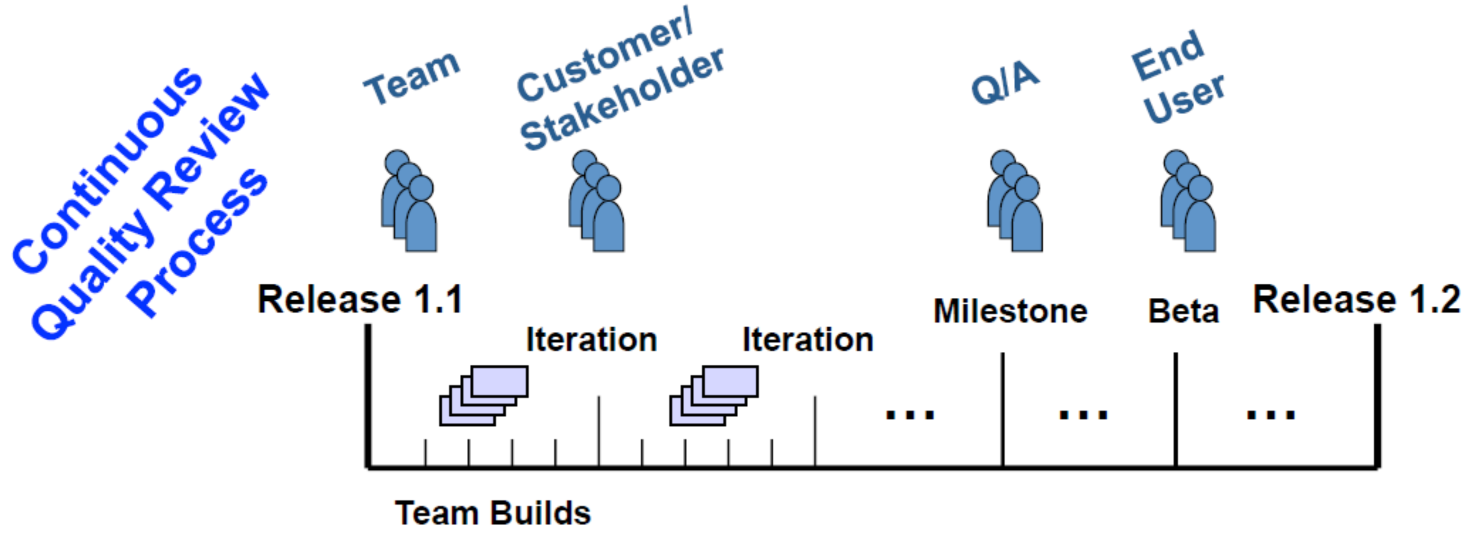
- Testing occurs as a project phase towards the end of project
- Lots of lead time for test planning, test case generation, test environment and infrastructure setup
- Test cases don’t change often (tied to requirements)
- Cost of creating test cases is borne once
- Few changes to system once it is specified and designed
- Tests are executed periodically
- Initial round of testing to ensure system meet requirements
- Regression testing after any significant change to ensure nothing was broken

FOLLOWING AGILE PRACTICES

- Development is continuous
- No separate phase for testing
- Develop and test continuously, feature by feature
- Features change, new features come up
- Testing must adapt to changes

CONTINUOUS INTEGRATION

- Development is continuous
- No separate phase for testing
- Develop and test continuously, feature by feature
- Features change, new features come up
- Testing must adapt to changes



CONTINUOUS INTEGRATION - FOLLOWS THE AGILE APPROACH

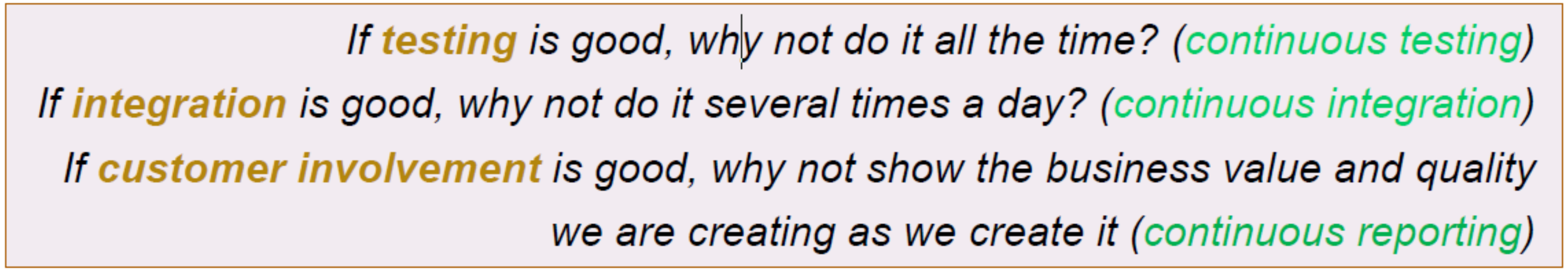
- Definition:
  - Integrate & build the system several times a day
  - Integrate every time a task is completed
- Lets you know every day the status of the full system
- Continuous integration and relentless testing go hand-in-hand.
- By keeping the system integrated at all times, you increase the chance of catching defects early and improving the quality and timeliness of your product.
- Continuous integration helps everyone see what is going on in the system at all times.

CONTINUOUS INTEGRATION

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily, leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible."

- Martin Fowler

https://en.wikipedia.org/wiki/Martin_Fowler



*If **testing** is good, why not do it all the time? (continuous testing)*

*If **integration** is good, why not do it several times a day? (continuous integration)*

*If **customer involvement** is good, why not show the business value and quality we are creating as we create it (continuous reporting)*

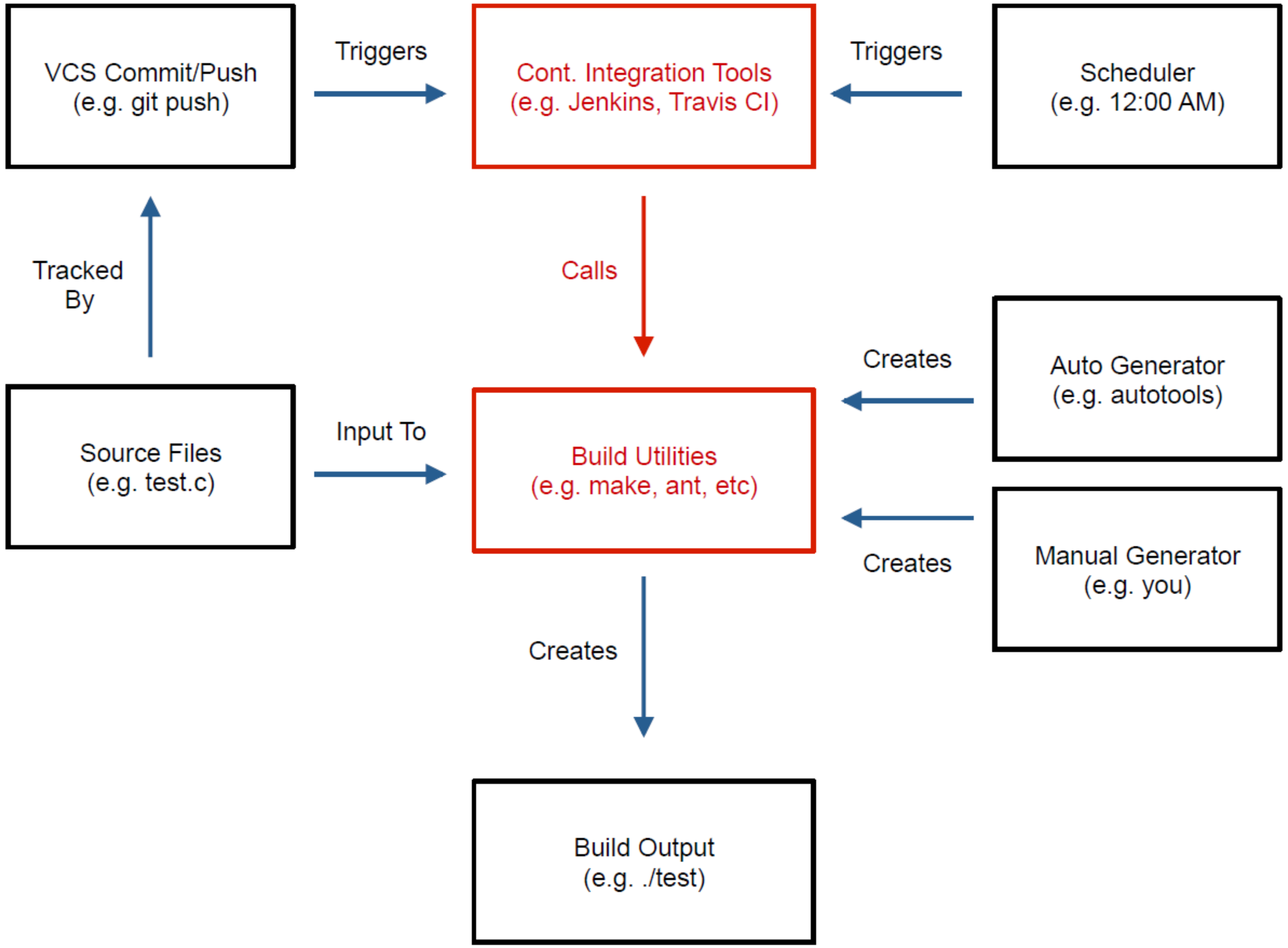
WHY DO CONTINUOUS INTEGRATION?

- Speed Up Each Feature/Sprint
- Automate Deployment
- Guarantee that Necessary Tests Are Being Run
- Track Build and Test Status
- Immediate Bug Detection
- Yields more Bug-Free Code
- A “Deployable” system at any given point
- Record of the history/evolution of the project

WHY DO CONTINUOUS INTEGRATION?

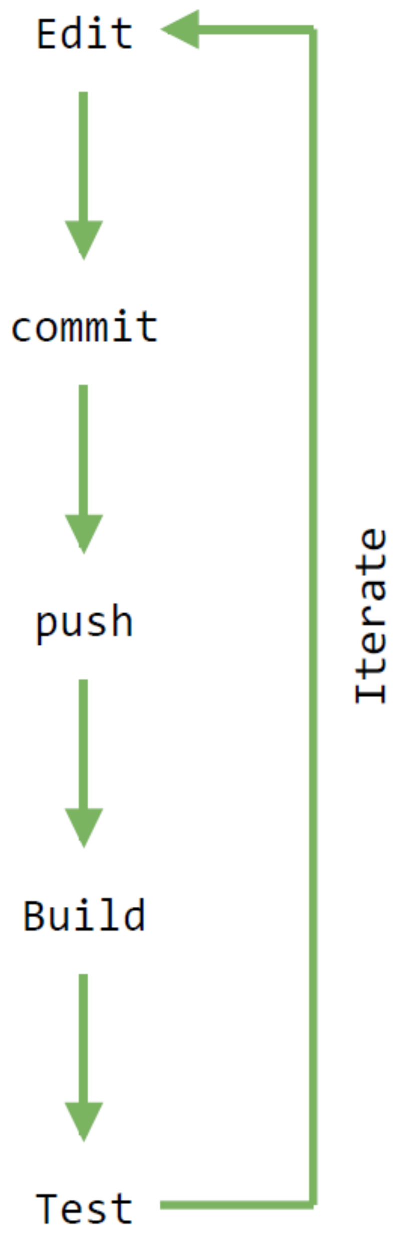
At regular intervals (ideally at every commit), the system is:

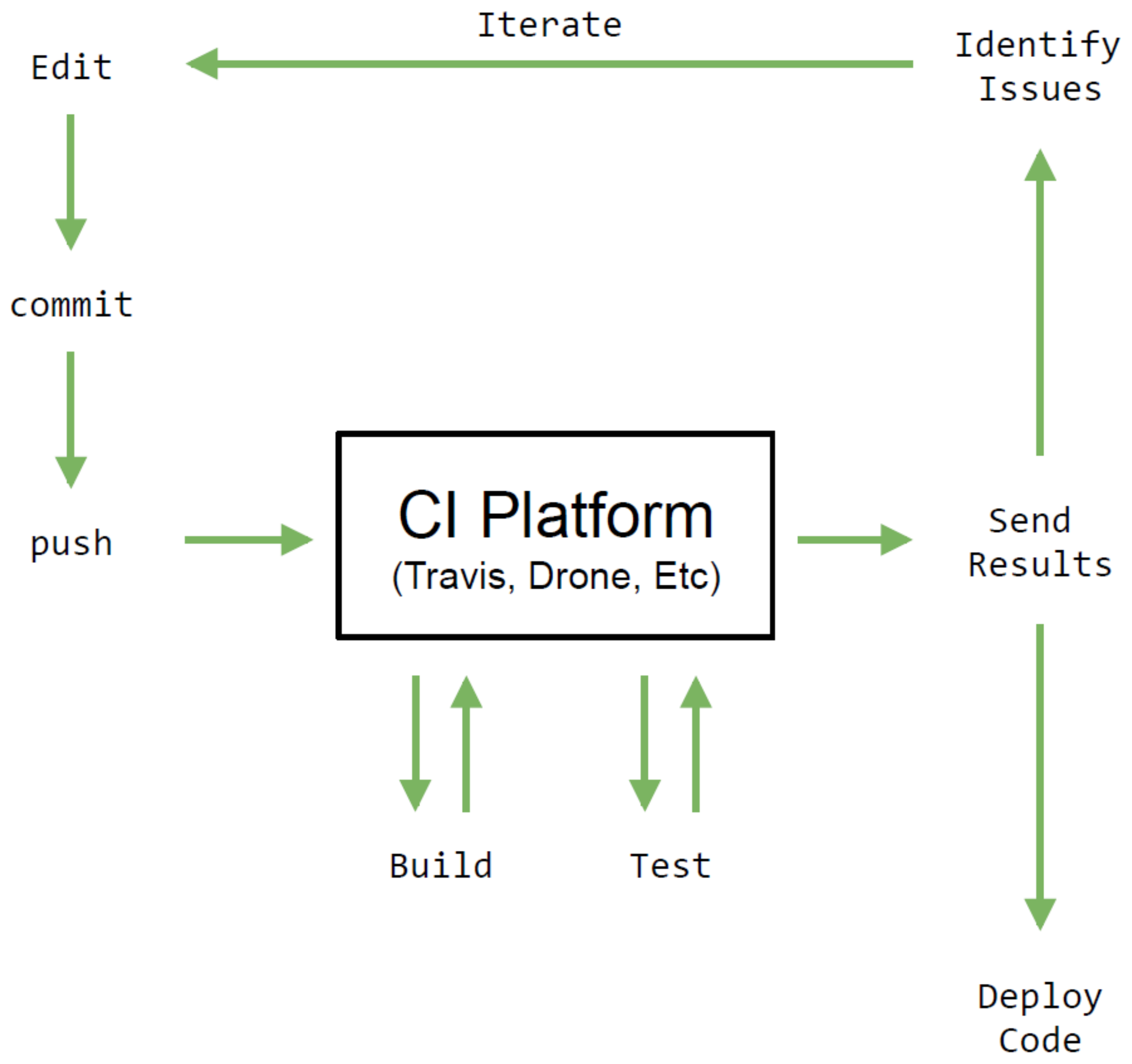
- Integrated
- All changes up until that point are combined into the project Build
- The code is compiled into an executable or package
- All code is Tested (regression)
- Automated test suites are run
- All code is Archived
- All code is versioned and stored so it can be distributed as is, if desired
- All code is Deployed
- All code is Loaded to an environment where the developers can interact with it


VCS Commit/Push (e.g. git push)
Triggers
Cont. Integration Tools (e.g. Jenkins, Travis CI)
Triggers
Scheduler (e.g. 12:00 AM)
Tracked By
Calls
Creates
Auto Generator (e.g. autotools)
Source Files (e.g. test.c)
Input To
Build Utilities (e.g. make, ant, etc)
Creates
Manual Generator (e.g. you)
Creates
Build Output (e.g. ./test)

FOWLER'S 10 BEST PRACTICES FOR CI

1. Maintain a Single Source Repository
2. Automate the Build
3. Make your Build Self-testing
4. Everyone Commits Every day
5. Every Commit should Build the Mainline on an Integration Machine
6. Keep the Build Fast
7. Test in a Clone of the Production Environment
8. Make it easy for Anyone to get the Latest Executable
9. Everyone can see what's Happening
10. Automate Deployment


Edit
commit
push
Build
Test
Iterate


Iterate
Edit
Identify Issues
commit
push
CI Platform
(Travis, Drone, Etc)
Send Results
Build
Test
Deploy Code

drone.io: https://drone.io/

Public Projects: Free

Private Projects: Paid

Concurrent Builds: 1

VCS: GitHub, Bitbucket, Google Code

Travis CI: https://travis-ci.org/

Public Projects: Free

Private Projects: Paid

Concurrent Builds: A Few

VCS: GitHub

DOCUMENTATION

PREMISE

- The developer writing the code knows what the code is doing (or is supposed to do)
- Other people will [someday] have to read, understand, modify, test your code
- Documentation will help

TWO PRIMARY TYPES OF DOCUMENTATION

- External to your code
  - A feature/design specification document
  - A "wiki" type of website, document repo
- Internal to your code
  - Comments
  - Good coding habits

WHAT MUST WE DOCUMENT?

- What is this "data"
  - A database
  - Transaction data from user input/screen
  - Validation Rules
  - Source / Destination

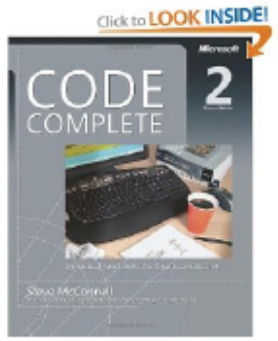
WHAT MUST WE DOCUMENT?

- What is the code doing to the data
- The meaning of variables
- Functions, methods, called procedures

“SELF-DOCUMENTING” CODE

- Program structure
- Variable Naming
- Class and Method Names
- Named Constants instead of Literals
- Minimized control flow
- Reduced data structure complexity

Self-Documenting Code



- Code should be written for humans
  - Compiler will keep the machine happy
- Quality Comments
  - Bad comments are worse than none at all!
- Naming Scheme
  - Make a name count!
- Coding Style
  - Decide on one and enforce its use
- Documentation Extraction Systems
  - JavaDoc, Doxygen, rdoc, etc…

Effective comments DO NOT repeat the code!

YOUR ORGANIZATION SHOULD DEFINE AND ENFORCE

- Standards for names
  - Variables – lower case, words separated by "_"
  - Methods/Functions – camelCase, no "_"
- Avoid numbers as differentiators
  - grade1, grade2, grade3
- Use a name that is self-explanatory – no comments needed
- Standard abbreviations
  - "dept" for "department"
  - "cust" for "customer"

CODE STRUCTURE

- Consistent Indentation
- Using braces {…}
- Where to declare variables
- Make it modular

FUNDAMENTAL PRINCIPLES

- The best documentation is the code itself.
- Make the code self-explainable and self-documenting, easy to read and understand.
- Do not document bad code, rewrite it! (Refactoring)

DOCUMENTATION

- Source code documentation generator tools
- Generate formatted, browsable, and printable documentation from specially-formatted comment blocks in source code.
- This allows for developer documentation to be embedded in the files where it is most likely to be kept complete and up-to-date.
- Javadoc for Java
- Doxygen for

  C++, C, Java, Objective-C, Python, VHDL, PHP, C#

HOW IT WORKS?

- Write comments in special format
  - Include html formatting
  - Use tags to specify specific kinds of documentation
- Leave the rest to the tool

DOCUMENTATION

A quick look at Doxygen:

http://www.doxygen.nl/manual/docblocks.html