Integration & Continuous Integration

SWE 261P

Material adapted from Francisco Servant

Integration

- **integration**: Combining 2 or more software units
  - often a subset of the overall project (!= system testing)

- Why do software engineers care about integration?
  - new problems will inevitably surface
    - many systems now together that have never been before
  - if done poorly, all problems present themselves at once
    - hard to diagnose, debug, fix
  - cascade of interdependencies
    - cannot find and solve problems one-at-a-time

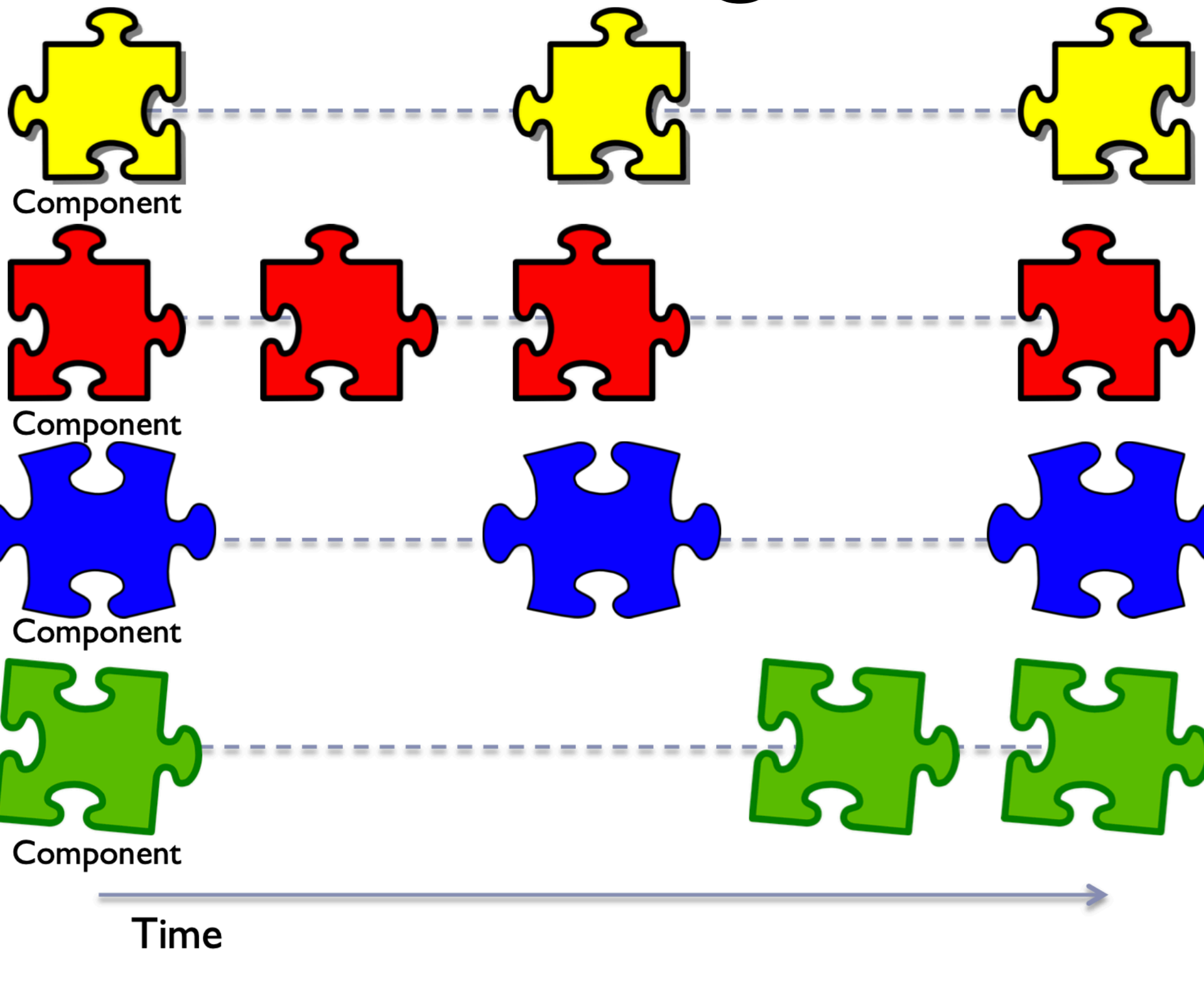
Daily builds

- **daily build**: Compile working executable on a daily basis
  - allows you to test the quality of your integration so far
  - helps morale; product "works every day"; visible progress
  - best done automated or through an easy script
  - quickly catches/exposes any bug that breaks the build

- **smoke test**: A quick set of tests run on the daily build.
  - NOT exhaustive; just sees whether code "smokes" (breaks)
  - used (along with compilation) to make sure daily build runs

- **continuous integration (CI)**:
  - adding new units immediately as they are written.
  - automated CI tools: Hudson, Jenkins, TravisCI

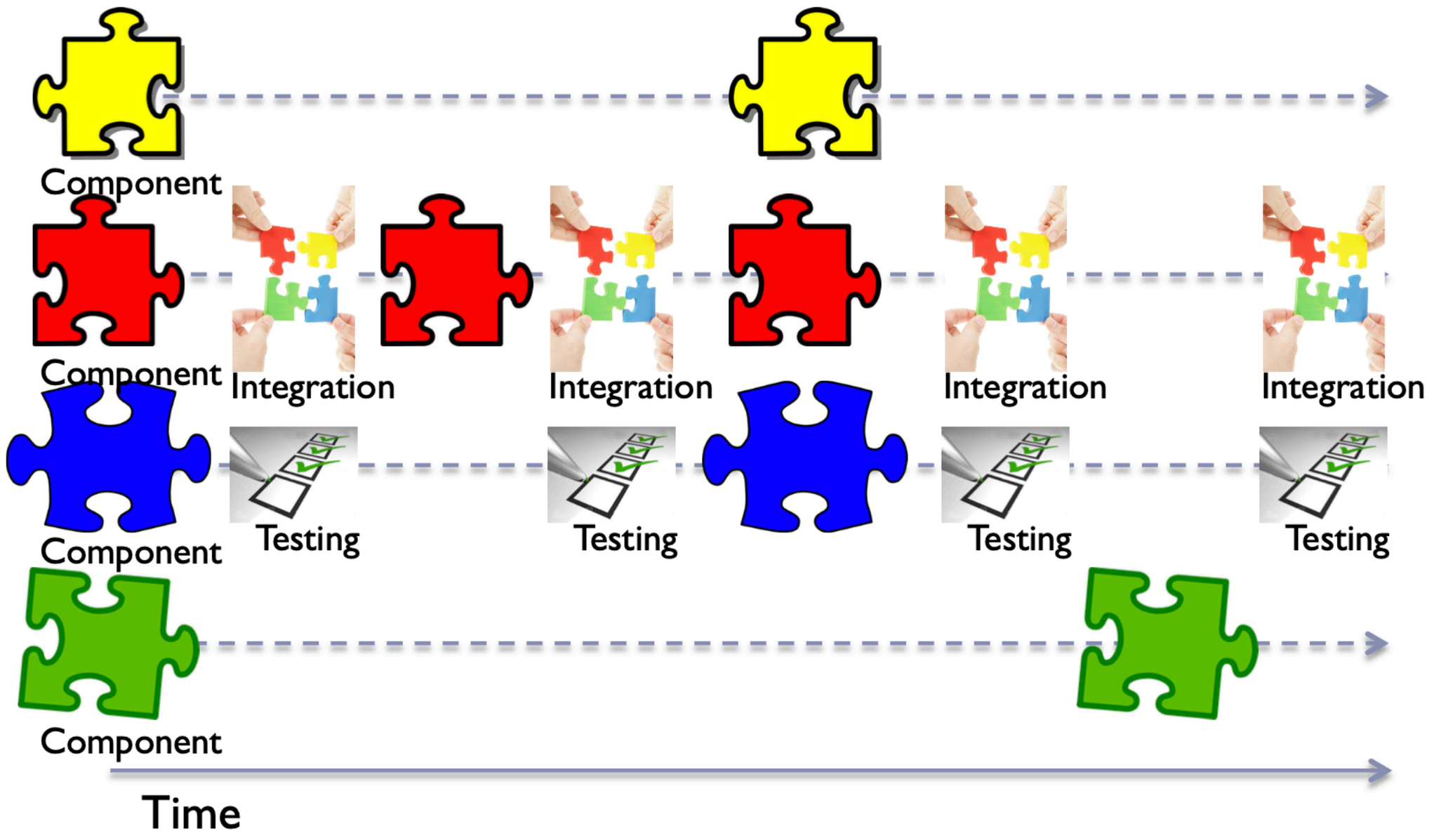
Integration



Integration

Testing

Continuous Integration



Continuous Integration

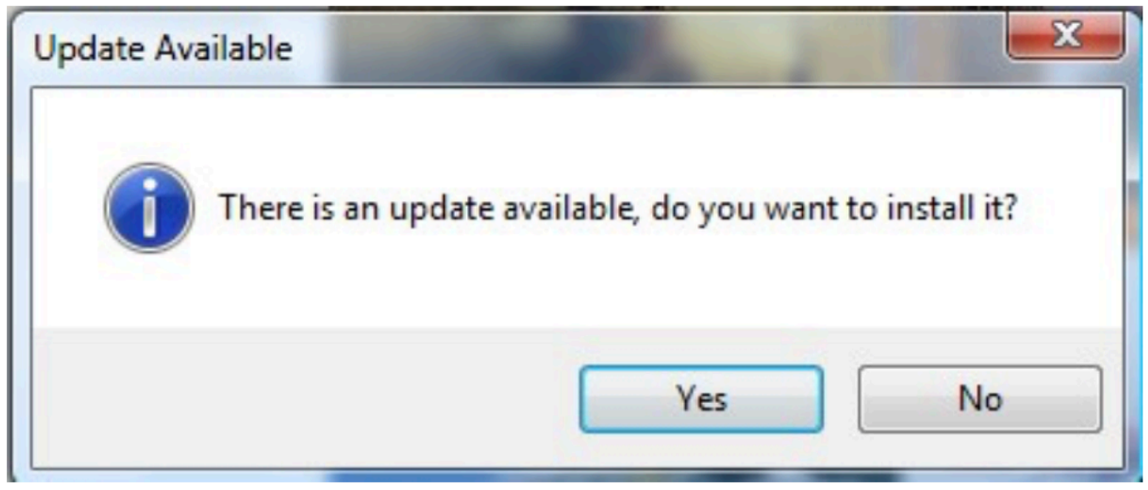
- Originated with the Extreme Programming development process
- Integration is performed continuously for small changes
- Developers commit their code every day
- The latest version of the code always builds and passes all tests

Why Use Continuous Integration?

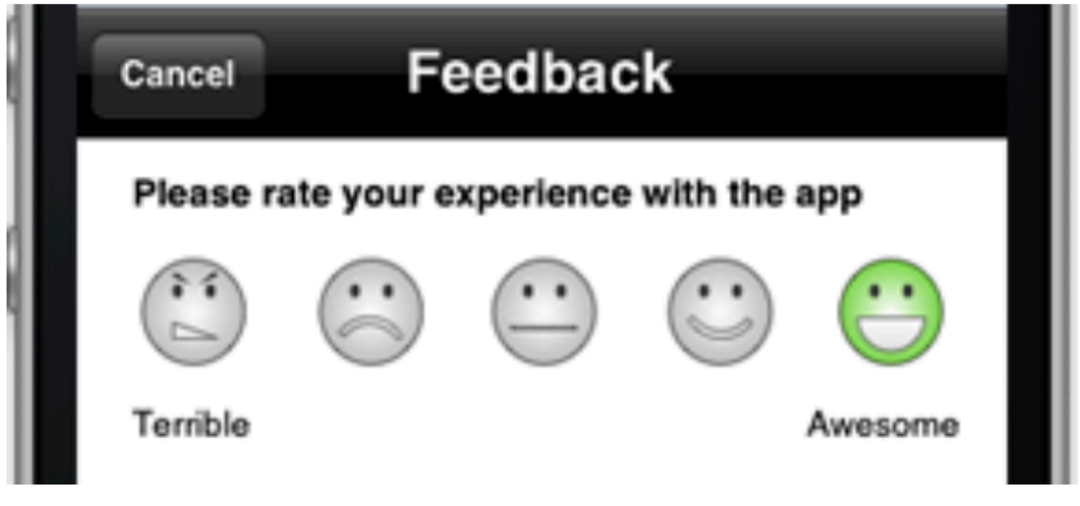
- Easier to predict development time
- Bugs are detected earlier, so they are:
  - Smaller
  - Easier to diagnose
  - Easier to fix (diff debugging)
  - Bugs are detected separately: Prevents bug interaction

Why Use Continuous Integration?

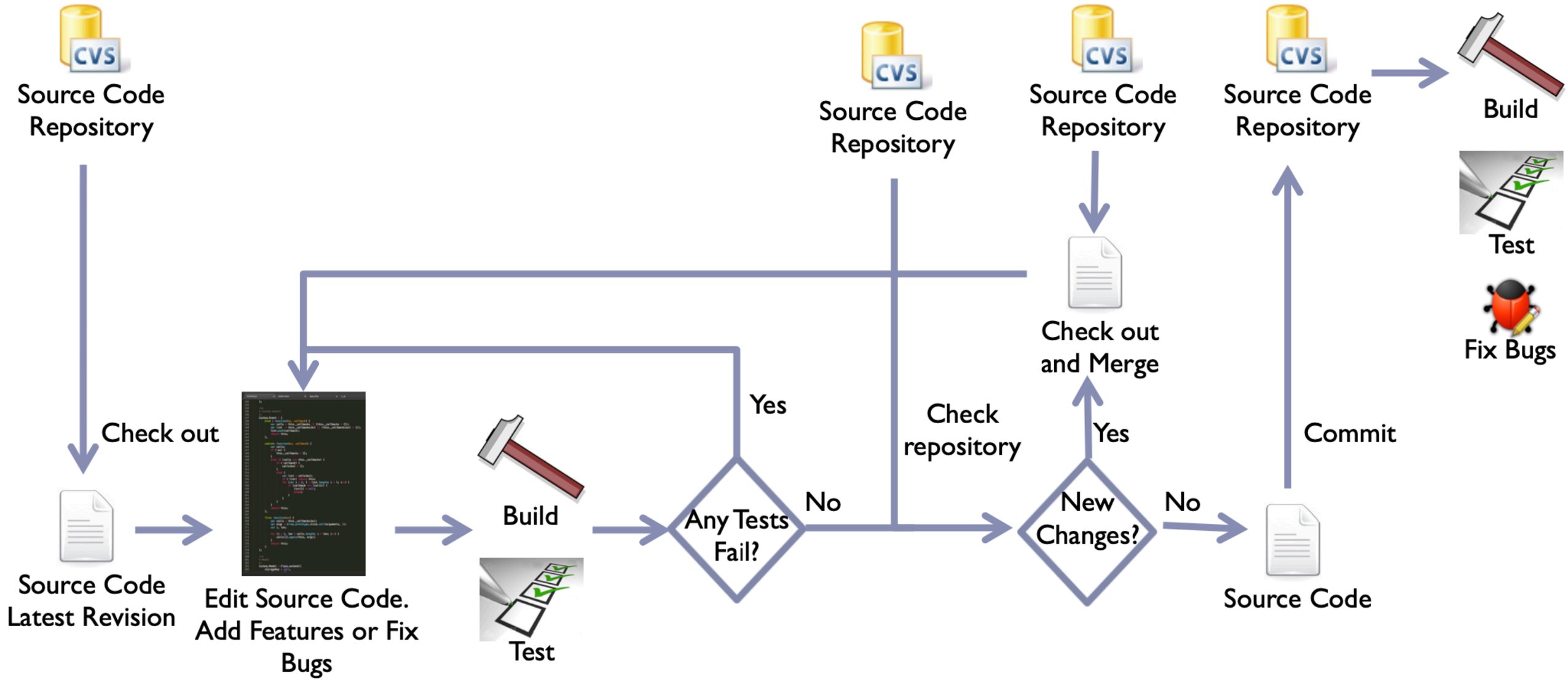
- Facilitates frequent deployment
- Users frequently get new features



- Developers get feedback more quickly



How Does Continuous Integration Work?



Practices of Continuous Integration

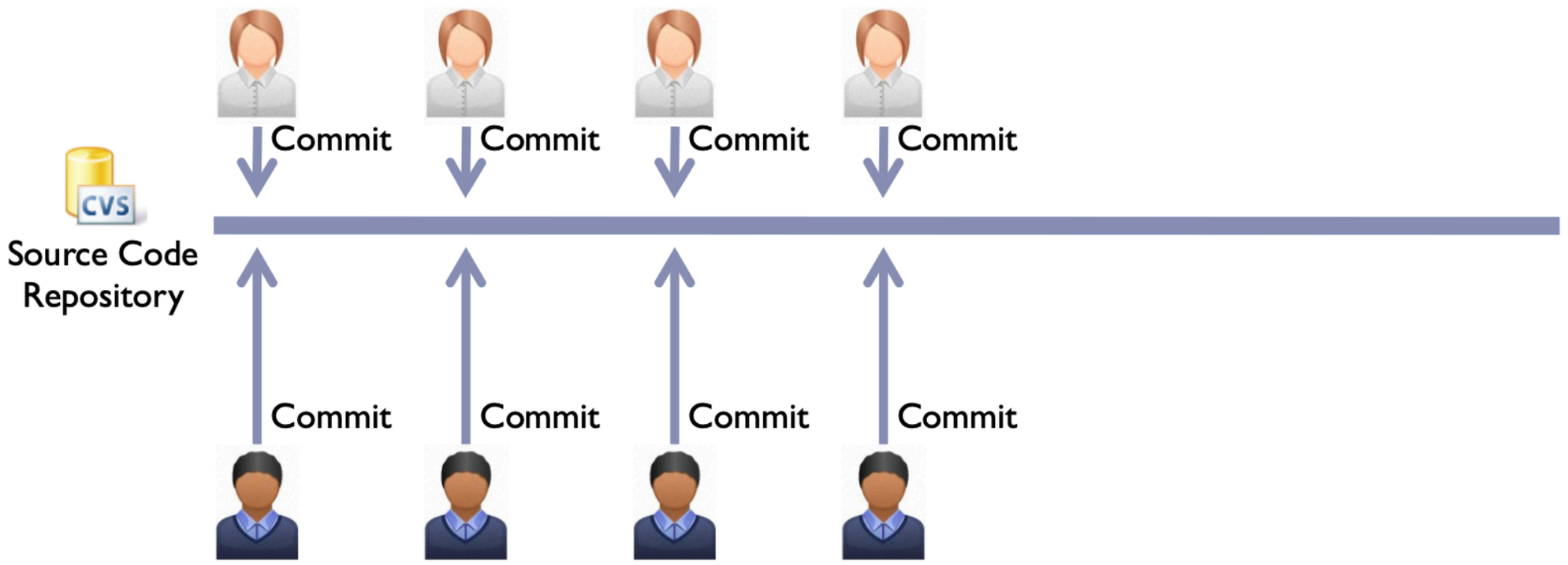
1. Maintain a single source code repository
2. Everyone commits to the main line every day
3. Every commit should build the main line
4. Automate the build
5. Automate testing
6. Keep the build and testing fast
7. Clone the production environment in the integration machine
8. Make it easy for anyone to get the latest executable
9. Everyone can see what’s happening
10. Automate deployment

1. Maintain a single source code repository

- ✓ Source-code Files
- ✓ Properties Files
- ✓ Test Scripts
- ✓ Database Schemas
- ✓ Install Scripts
- ✓ Third Party Libraries
- ✓ …

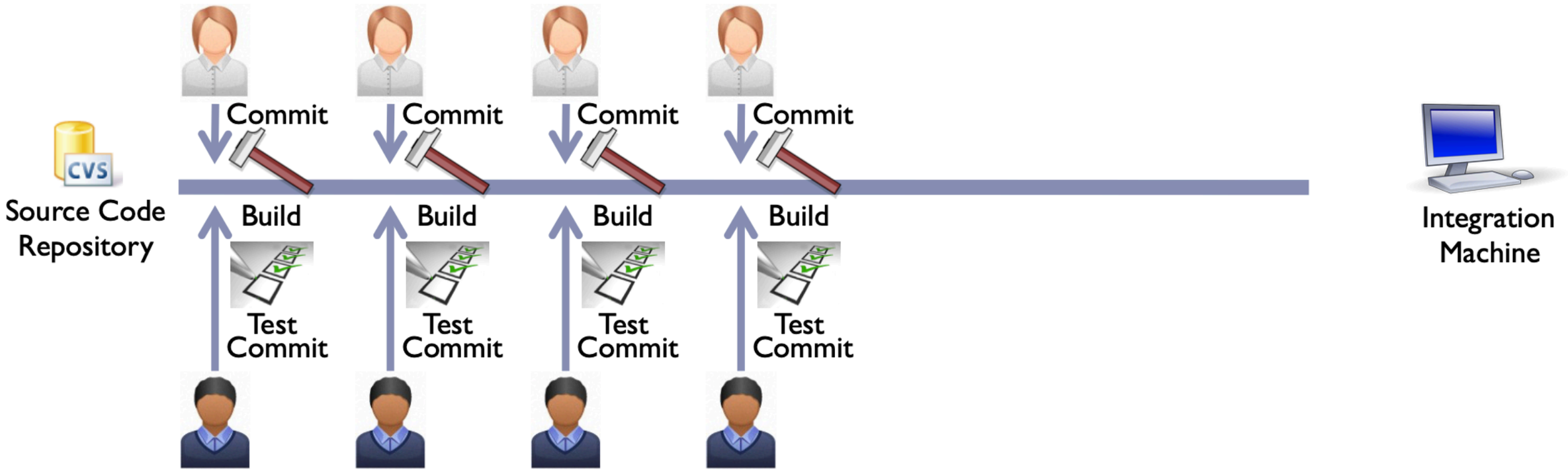
- ✗ Binaries

2. Everyone commits to the main line every day



- Developers should commit their code to the repository at least once per day
- Committing often allows for:
  - Early detection of conflicts (build and test conflicts)
  - Frequent developer communication
  - Breaking down implementation work into small changes

3. Every commit should build the main line



- The main development line should be kept stable
- Committed code may still fail building or tests
  - Developers may have not pulled others' changes before committing
  - Developers' development environment may have differed
- The developer that committed changes is responsible for them until they successfully build and pass tests in the integration machine

4. Automate the build

Manual build

- Complex
  - Compile
  - Move files around
  - Load database schema
  - …

Automatic build

- One single command
  - Make, Ant, Maven, Gradle, …
- Incremental
  - Only recompile for changed components
  - Divide system into many small components
- Command line more flexible than Visual IDE

5. Automate testing

Manual tests

- Not repeatable
- Big test cases

Automatic tests

- Create unit tests that test small functionality
- One single command
- Tests should assess their own pass/fail outcome

6. Keep the build and testing fast

- A slow build and test is the most challenging issue for continuous integration
- High impact: run many times
- Ideal time to build and test is said to be at most 10 minutes

7. Clone the production environment in the integration machine

- The integration machine should be a clone of the production environment
    - Same database software and operating system at the same versions, same libraries, same IP addresses, ports, hardware, …
- Understand the risks taken when an aspect is not cloned
- Use virtual machines
    - Virtual machines also allow parallel testing

8. Make it easy for anyone to get the latest executable

- The latest executable should be available for:
  - Demonstrations
  - Exploratory testing
  - …

9. Everyone can see what's happening

- Remember the visibility principle (process visibility)
- Monitors can show when the integration machine is:
    - Building
    - Running tests
    - Who did what
    - What tests/builds are passing/failing
- Websites allow global access

10. Automate deployment

- Automate with a single command
    - Deployment into production
    - Rollback from production
- Trial deployment
    - Deploy to only a few users or different versions to different users

Tools for Continuous Integration

- Jenkins
- Cruise Control
- Continuum
- Hudson
- TravisCI
- CircleCI
- GitHub Actions
- ...