CS1530, Lecture 14: Integration

Bill Laboon

What is Integration?

- Combining separate software components into a larger system
- Can be classes, packages, libraries, subsystems, applications, servers, processes, etc.

Integration

- Could take minutes, could take weeks
- Contains more potential than average for defects to crop up

Why?

- Developers work on smaller parts of the project and understand that small part
- Developers usually grouped by subsystem/class/package/etc
- Developers have less communication with people on other teams than those on their own team
- Developers have less understanding of other parts
- Often reliant on documentation (remember Agile rule: communication over documentation)

You know what happens when you assume…

- Different assumptions can cause your teams' work to not interface with other teams' work
- Ambiguities pop up
- Sometimes you're not even aware you're making assumptions

Case Studies of Integration Failures

- Mars Climate Orbiter - crashed because Lockheed Martin subsystem used American units, NASA subsystem (and rest of system) used metric
  - Note in this case, Software Interface Spec actually did define input/output to be metric
- Mars Polar Lander
  - Shaking in legs convinced the other systems that the craft had already landed!
- Gimli Glider - filled up fuel using old imperial units instead of metric

Case Studies of Integration Failures

- Unknown unknowns
- Sometimes you think you know how something works

Regular Expressions

```
// bash shell

(15485) $ cat sample_grep.txt
foo bar

(15486) $ grep "monkey" sample_grep.txt

(15487) $ grep "foo" sample_grep.txt
foo bar

(15488) $ grep "foo" sample_grep.txt
foo bar
```

Regular Expressions

```
// JavaScript

pattern = new RegExp(“foo","g");
pattern.test("foo bar")
=> true
pattern.test("foo bar")
=> false
```

Other Possible Issues

- Subsystem doesn't build
- Subsystem just doesn't function correctly
- Interface not to spec, or problems in spec
- Performance or other non-functional problems
  - A HUGE issue. Often difficult to measure quality attributes until late in the game.
- Subsystem not reliable
- One system ready, another not

Hardware / Software Integration

- These are especially big issues in mixed hardware/software systems, especially if hardware isn't available until after software has started to be written!

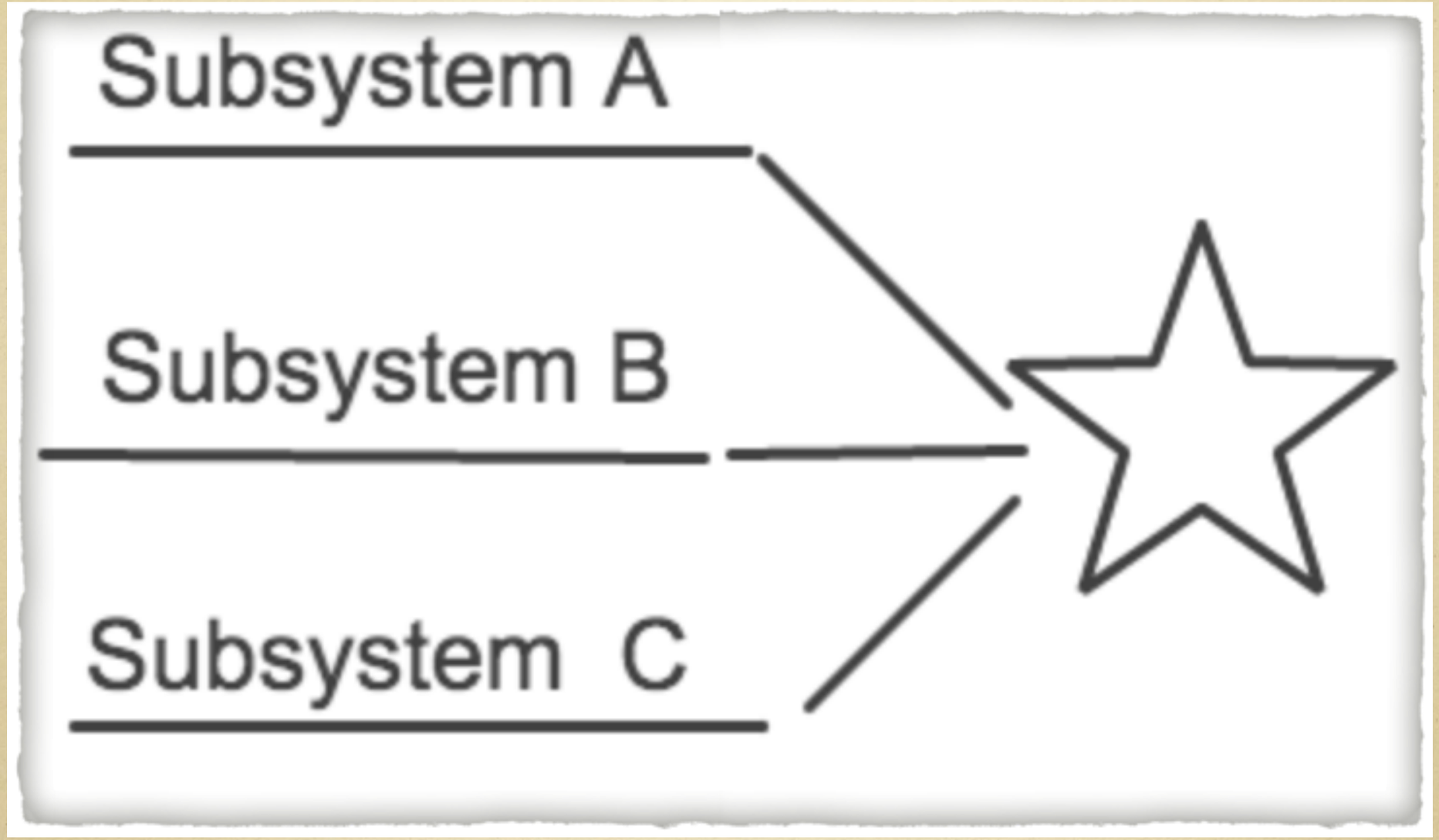
Integration Methodologies

- Phased
- Incremental
- Continuous

Phased Integration

- Design/code/test/debug each class/unit separately
- Combine everything at once
- System blows up. Fix, fix, fix. Repeat.
- Release version n!

Phased Integration



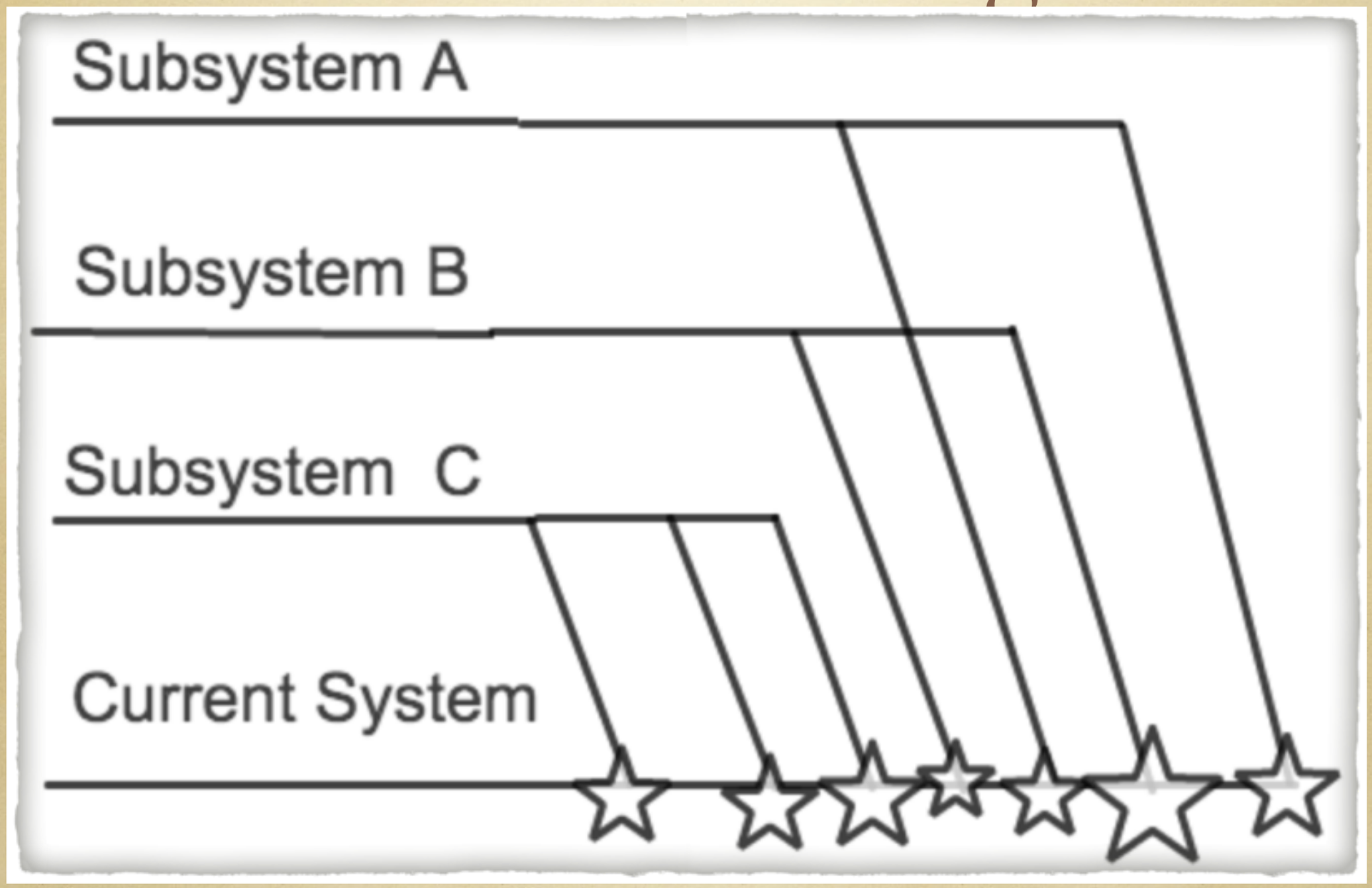
Drawbacks of Phased Integration

- Many. So, so many.
- If there is a defect, could be anywhere
- Usually done at end of schedule - if any problems, may need to push back release
- Must wait for slowest subsystem
- Time to panic - may take shortcuts to get out the door on time

Incremental Integration

- Design/code/test/debug new class/unit
- Add to rest of the system
- Perhaps minor blow-ups. Fix.
- Now have version n.0, next time n.1, etc.

Incremental Integration



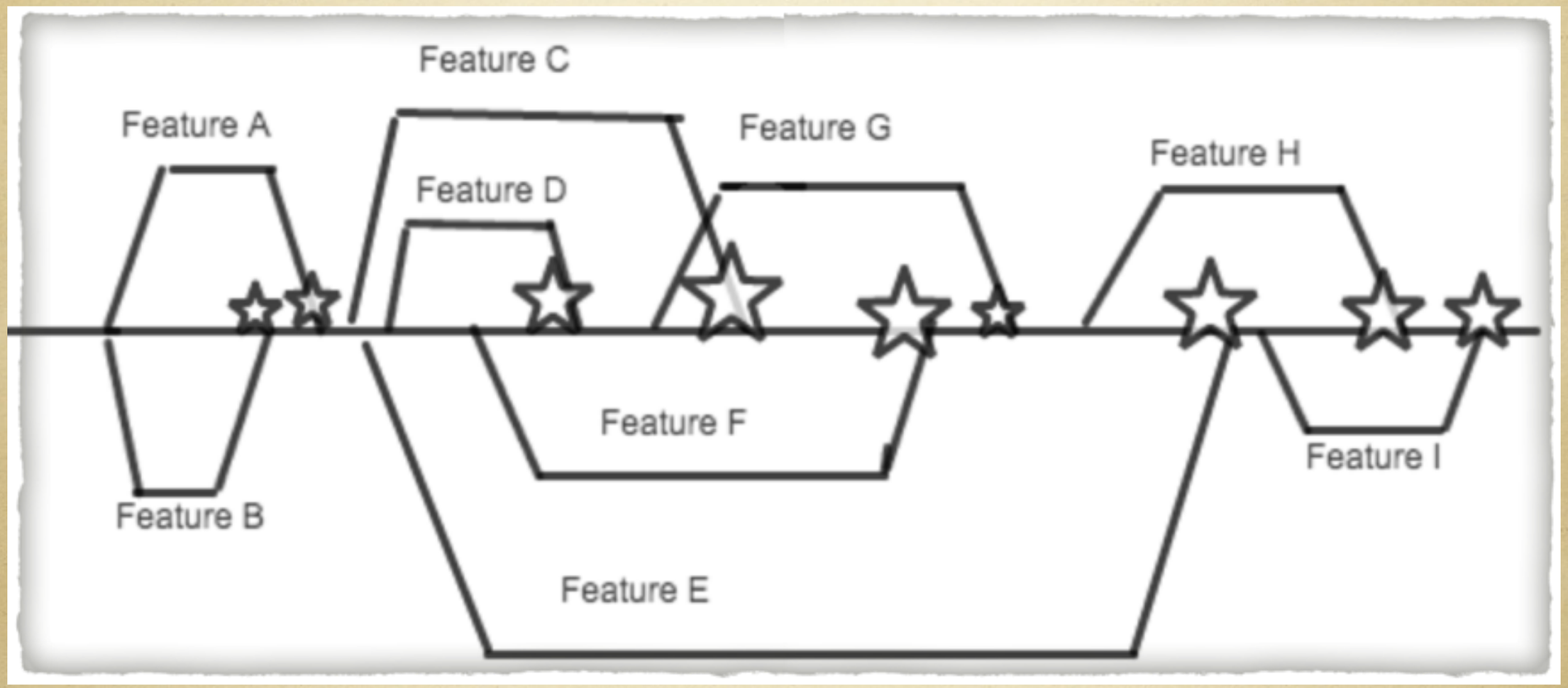
Incremental Integration Usually Considered Superior

- Errors easier to find (less code changed, smaller steps, etc.)
- Integration is done throughout project (remember, things done often tend to be done right, or least better)
- Failures found earlier
- Monitoring progress is easier
- Improved customer / developer morale

Continuous Integration

- Integrate each change with build within hours (original definition: at least daily)
- Pooh-poohed by Code Complete, but tools have improved!
- Depending on QA, a change you make could be in production in less than an hour
- Heavy reliance on automation

Continuous Integration



Continuous Integration

- Developer works on feature (writes code, writes tests, smoke tests)
- Push to VCS (version control system)
- Several things automatically happen
  - Hound: does automatic code review
  - Automated tests kick off
  - Auto-assigns code reviewers
- If no hound comments and tests pass, code reviewers review code, set flag
- QA person automatically assigned, does final check and merges if good

Benefits of CI

- Faster development time
- Smaller commits / changes means integration easier
- Reliance on automated tools means less chance for human error
- Remember that which is done often tends to be done better

Downside of CI

- Problem: Long-running features or changes
- Solution: feature toggles
- Turn features off and on
- Adds complexity, but also allow for CI
- Also allow for accidental release of new features if you forget to add some UI in a toggle! (try not to do this)

Feature Toggle Example

```
if (FeatureToggles.get("Survey") == true) {
    showSurvey();
} else {
    showDefaultMessage();
}
```

Integration Strategies

- Top-down: work on highest-level classes (e.g. highest layer in a layered architecture)
- Bottom-up: work on lowest-level classes (e.g. bottom layer in a layered architecture)
- Risk-oriented: work on the riskiest parts first
- Feature-oriented: group by feature, not by class
- T-shaped: Start top-down, then go all the way on one chunk first, then another, etc. Can be similar to feature-oriented

Further Reading

- Cataldo, M. and Herbsleb, J. "Factors Leading to Integration Failures in Global Feature Oriented Development: An Empirical Analysis" (http://delivery.acm.org/10.1145/1990000/1985816/p161-cataldo.pdf?ip=150.212.123.236&id=1985816&acc=ACTIVE%20SERVICE&key=AA86BE8B6928DDC7%2E3F18A282B75518AA%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=520130042&CFTOKEN=54222915&__acm__=1434554321_f2bbd30d4f0234db0ecf0208becf00220)
- Code Complete chapter 29
- Muller, G. "Why is Systems Integration understood so poorly? Reflections on 3 decades of unforeseen failures" (http://www.gaudisite.nl/SystemsIntegrationReflectionKSEESlides.pdf)
- Netflix Tech Blog, "Preparing Netflix API for Deployment" (http://techblog.netflix.com/2013/11/preparing-netflix-api-for-deployment.html)