Chapter 29: Integration

- Integration is the software development activity in which you combine separate components into one single system

Importance of the Integration Approach

- Benefits
    - Easier defect diagnosis
    - Fewer defects
    - Less scaffolding
    - Shorter time to first working product
    - Shorted overall development schedules
    - Better customer relations
    - Improved morale
    - Improved chance of project completion
    - More reliable schedule estimates
    - More accurate status reporting
    - Improved code quality
    - Less documentation

Integration Frequency – Phased or Incremental

Phased Integration

- Was the norm for many years
- Phases
    - 1) design, code, test, and debug each class
        - Called “unit development”
    - 2) Combine the classes into one whopping big system
        - “system integration”
    - 3) Test and debug the whole system
- Problem with this is new problems arise and could come from anywhere
- Difficult to localize problems

Incremental Integration

- Write and test a program in small pieces then combine pieces one at a time
- Phases
    - 1) Develop a small functional part of the system
    - 2) Design, code, test, and debug a class
    - 3) Integrate new class with the skeleton
        - Test and debug the skeleton
        - Don’t add any more new classes until current setup works

Benefits of Incremental Integration

- Errors are easy to locate
- System starts working, partially, earlier
- Improved progress monitoring
- Improved customer relations
- Units of the system are tested more fully
- Can build the system in a shorted development schedule

Incremental Integration Strategies

- Need to actually plan the order of integrating certain parts

Top-Down Integration

- Classes at the top of the hierarchy are written and integrated first
    - The top is the main window
    - Stubs for all subsequent classes need to be made
- Class interfaces need to be very specifically defined
- Control logic of the system is forced to be tested early on
- Also allows you to start coding before the low level design details are complete
- Downsides
    - System level interfaces aren't developed until last
        - Errors in this also wont arise until everything else is built
    - You also need a "dump truck" of stubs to integrate from the top down
    - Also, if there is no "top" you cant really work top down

Bottom-Up Integration

- Write and integrate classes at the bottom of the hierarchy first
- Restricts the possible sources of error to the single class being integrated
- BUT doesn't uncover high level and design flaws until last minute
- Also, unless design is all done before coding, this violates compartmentalization principles

Sandwich Integration

- Start with the top level stuff and the bottom level stuff
- Then figure out the middle stuff

Risk-Oriented Integration

- Identify the level of risk associated with each class
- Determine which will be the most challenging to implement, and do those first

Feature Oriented Integration

- Feature doesn't need to be fancy, just any identifiable functionality of the system
- Features can be part of other larger features

- Builds up from a skeleton
- Advantages
    - Eliminates scaffolding for nearly everything except low level library classes
    - Each time a feature is added, more functionality becomes available
    - It also works well with object oriented designs

Daily Build and Smoke Test

- Every file is compiled, linked and put into an executable program every day
- This is then run to see what “smokes” lmao
- Benefits
    - Reduces risk of low quality
    - Makes it easier to detect defects
    - Improves morale
        - Nice to see things actually working
- At minimum, a good build should
    - Compile all files, libraries and other components successfully
    - Link all files, libraries and components successfully
    - Not contain any showstopper bugs
- Need to make sure to do the smoke test too, without it the build is kinda useless
- AUTOMATE THIS
- No reason to try and add code components every day
    - But don’t wait too long to integrate revisions
- Make sure developers smoke test their code before adding it to the system
- Create a holding area for code that’s to be added to the build
- Create a penalty for breaking the build
    - Just want to make sure people know the build is supposed to be healthy
    - Make the penalty light hearted like lollipops or something
- Release builds in the morning
    - Less stress and can test in the AM
- Make sure to build and smoke test under pressure
    - Good habit
    - Stresses the importance of high quality code at all times