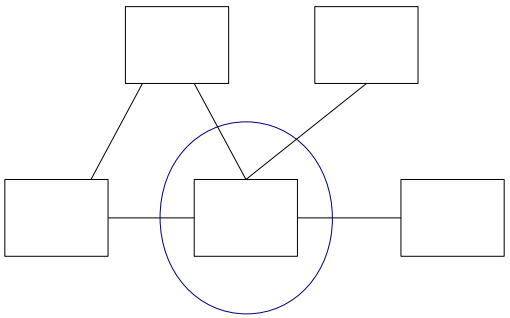
Topics for this Lecture

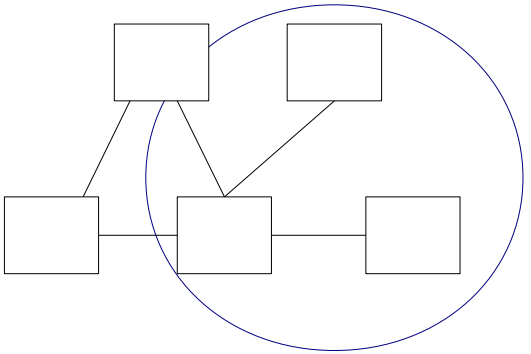
- Testing Granularity Levels
- The Concept of Integration Testing
- System Integration Techniques: Incremental, Top-down, Bottom-up, and Big-bang

Testing Granularity Levels

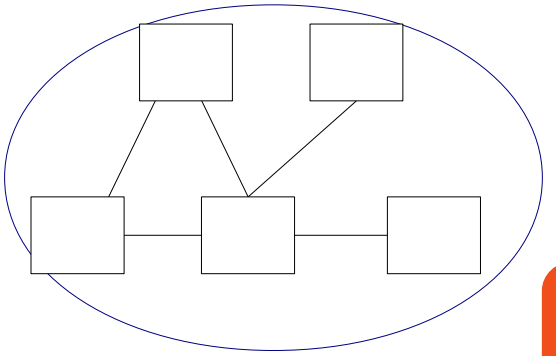
- Lets consider a software system.
  - A system made out of components that interact with one another
- The first level we consider in testing is called **unit testing**, which is the testing of an individual unit or module in isolation.
- The next step is to consider a multiple modules and their interactions, which is called **integration testing**. The **Integration testing** is the testing of interactions among different modules and it can be performed according to different strategies.
- The last step is to test the complete system as whole, which is called **system testing**.



Unit Testing



Integration Testing

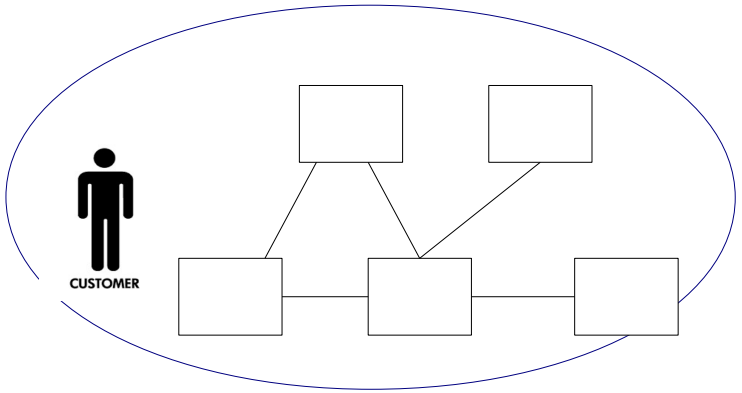


System Testing

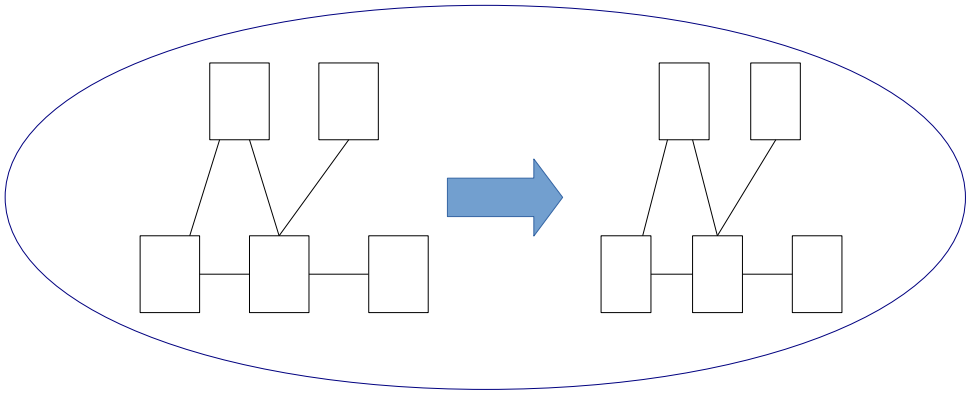
OSU
Oregon State
UNIVERSITY

Testing Granularity Levels

- **Acceptance Testing** which is the validation of the software against the customer requirements.
- **Regression Testing** is a type of testing
  - We perform regression testing every time that we change our system
  - We need to make sure that the changes behave as it is intended and the unchanged code is not negatively affected by these changes/modifications



Acceptance Testing



Regression Testing

Integration

- **Integration** refers to the software-development activity in which you combine separate software components (units) into a single system.
- **Importance of the Integration Approach**
  - 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
- *In engineering fields other than software*, the importance of proper integration is well known
- The football stadium add-on at

  the University of Washington collapsed because

  it wasn't strong enough to support itself

  during construction.

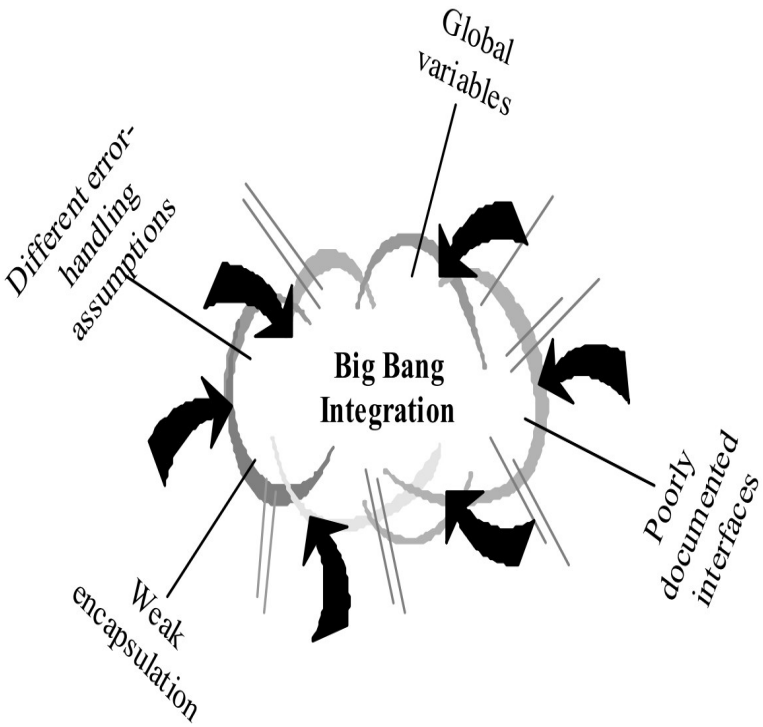
Oregon State
UNIVERSITY

Integration testing

- **Integration testing** is the combined execution of two or more classes, packages, components, or subsystems that have been created by multiple programmers or programming teams.
- This kind of testing typically starts as soon as there are two classes to test and continues until the entire system is complete.
- **Challenges**:
  - Combined units can fail in more places and in more complicated ways.
  - How to test a partial system where not all parts exist?

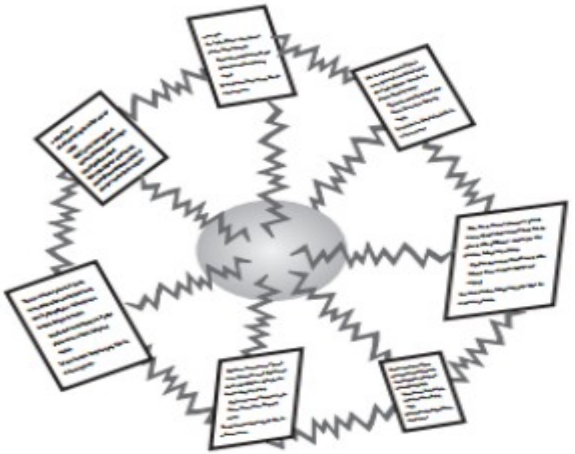
Phased (big-bang) Integration

- First, design, code, test, and debug each model. This step is called "unit development."
- Next, combine all those modules to construct the entire system.
- Then, test and debug the whole system.
- **Phased integration** can't begin until late in the project, after all the classes have been developer-tested.
- For small programs phased integration might be the best approach.

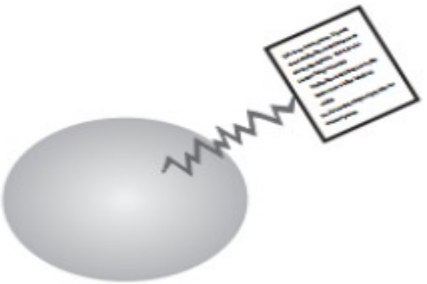


Incremental integration

- **Incremental integration**
- Develop a small, functional part of the system, which it will serve as a skeleton system
- Design, code, test, debug a small new piece
- Integrate this piece with the skeleton
- Test/debug it before adding any other pieces



Phased Integration



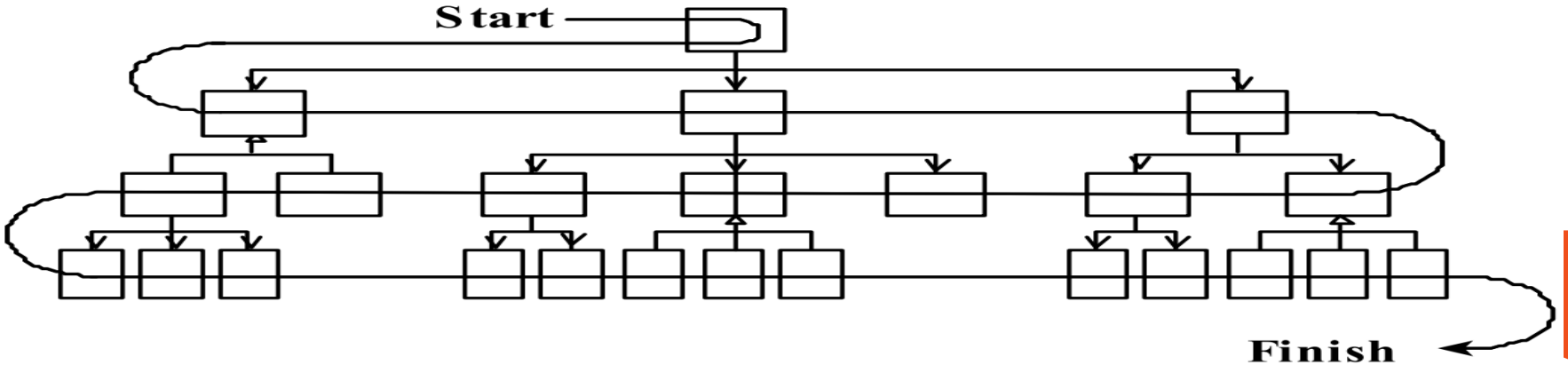
Incremental Integration

Benefits of incremental

- **Benefits**:
  - *Errors are easy to locate and fix*
    - reduces developer bug-fixing load
  - System is always in a (relatively) working state
    - good for customer relations, developer see early results from their work
  - You get improved progress monitoring
    - When you integrate frequently, the features that are present and not present are obvious.
- **Drawbacks**:
  - With incremental integration, you have to plan more carefully
  - May need to create "stub" versions of some features that have not yet been integrated

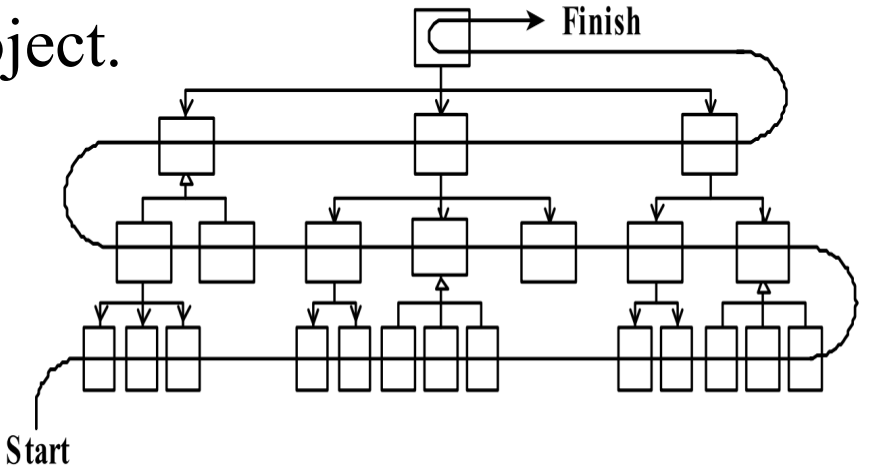
Top-down integration

- **Top-down integration:**
  - The class at the top of the hierarchy is written and integrated first
  - Must write (lots of) **stub** lower classes to interact with
    - Then, as classes are integrated from the top down, stub classes are replaced with real ones
- You can complete a partially working system early in the project.
- Allows postponing tough design/debugging decisions (bad?)



Bottom-up integration

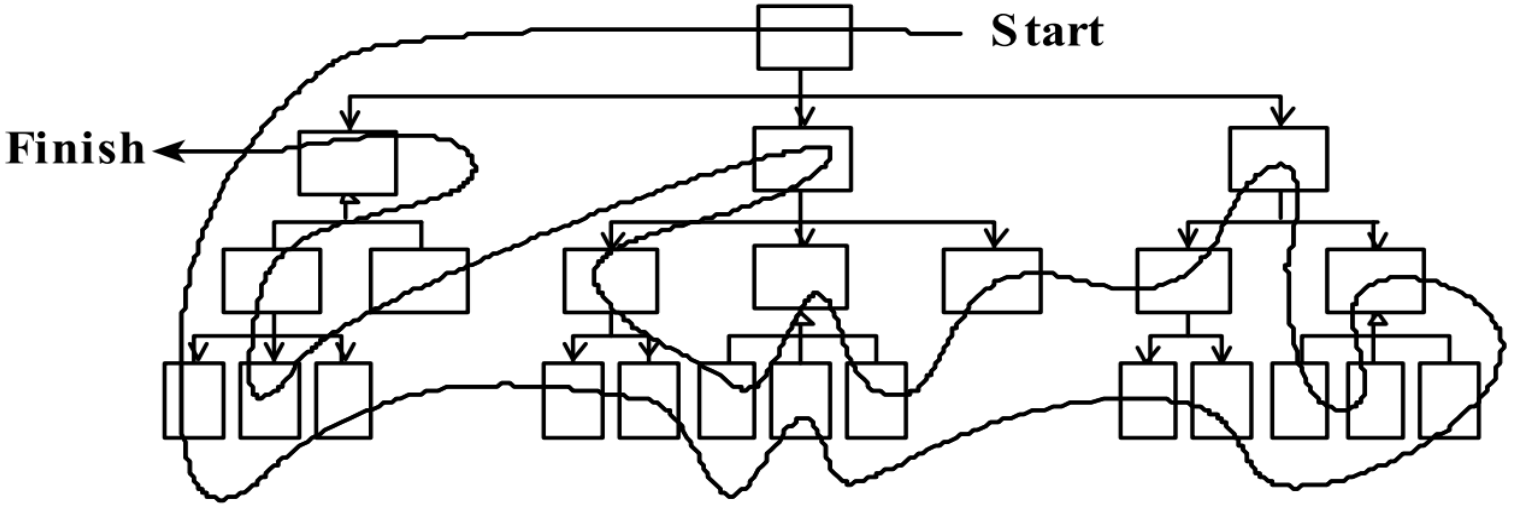
- **Bottom-up integration**:
- You write and integrate the classes at the bottom of the hierarchy first.
- Adding the low-level classes one at a time rather than all at once is what makes bottom-up integration an incremental integration strategy
- Integration can start early in the project.
- The main problem with *bottom-up integration* is that it leaves integration of the major, high-level system interfaces until last



Oregon State UNIVERSITY

"Sandwich" integration

- **"Sandwich" integration:**
  - You first integrate the high-level business-object classes at the top of the hierarchy.
  - Then you integrate the device-interface classes and widely used utility classes at the bottom.
  - Add middle layers later as needed
  - More practical than top-down or bottom-up?



Daily Build and Smoke Test

- **Daily build and smoke test**: Every file is compiled, linked, and combined into an executable program every day, and the program is then put through a "smoke test," a relatively simple check to see whether the product "smokes" when it runs.
  - Minimizes integration risk.
  - Improves morale; product "works every day"; visible progress "the *heartbeat of the project*". If there's no heartbeat, the project is dead.
  - Supports easier defect diagnosis. Quickly catches/exposes any bug that breaks the daily build.
- **Daily build:** For the daily-build process to work, the software that's built has to work. If the software isn't usable, the build is considered to be broken and fixing it becomes top priority.
- **Smoke test**: A quick set of tests run on the daily build.
  - NOT exhaustive, but it should be capable of exposing major problems.
  - The smoke test must evolve as the system evolves. At first, the smoke test will probably test something simple. As the system develops, the smoke test will become more thorough.
- **USING THE DAILY BUILD AND SMOKE TEST**. The idea behind this process is simply to build the product and test it every day.

References:

Code Complete: A Practical Handbook of Software Construction, Second Edition 2nd Edition by Steve McConnell
https://courses.cs.washington.edu/cse403/