CS2212

Introduction to Software Engineering

Software Testing Part 2: Integration Testing

Ask Questions Live
cs1.ca/ask

Integration Testing

- A seemingly legitimate question after unit testing:
  - *“If all units work individually, why do you doubt that they’ll work when we put them together?”*
- The problem, of course, is *“putting them together”.*
  - Data can be lost across an interface.
  - One component can have an inadvertent, adverse effect on another.
  - Subfunctions, when combined, may not produce the desired major function.
  - Individually acceptable imprecision may be magnified to unacceptable levels.
  - Global data structures could present problems.
  - And so on, and so on …

Integration Testing

- Integration testing is a systematic **technique for constructing the software architecture** while at the same time **conducting tests to uncover errors associated with interfacing**.

- The objective is to take **unit-tested components** and build a program structure that matches your **architecture design**.

Approaches to Integration Testing

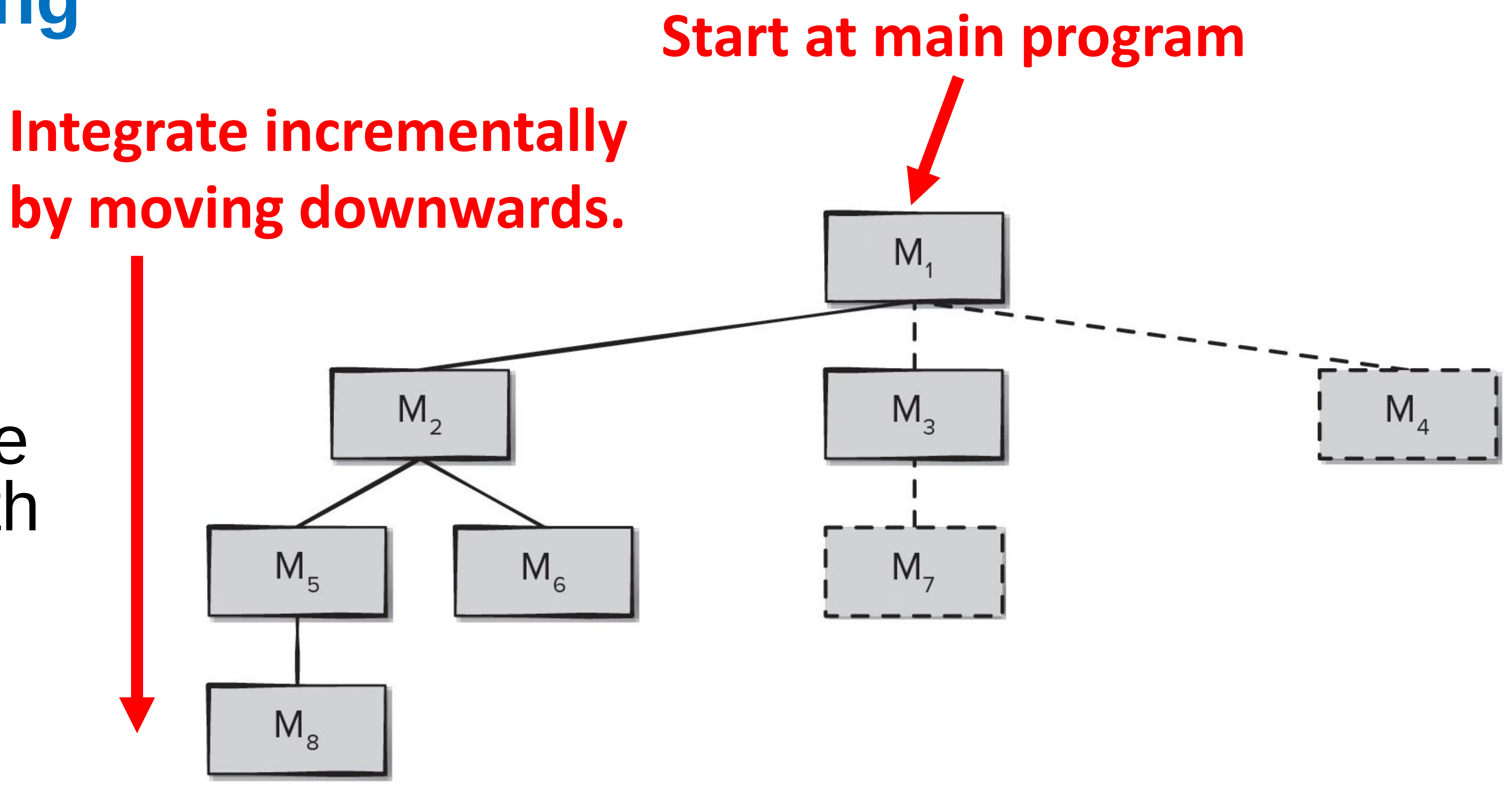
Big Bang Approach

- In the **big bang approach**, all components are combined at once and the entire program is tested as a whole.
- **Chaos usually results!**
- Errors are encountered, but correction is difficult because isolation of causes is complicated by the vast expanse of the entire program.
- Integration should be **incremental**. In **incremental integration** a program is constructed and tested in small increments, making errors easier to isolate and correct.

Approaches to Integration Testing

Top-Down Integration Testing

- **Incremental approach** to construction and testing of the software architecture.
- Modules are **integrated by moving downward** through the control hierarchy, beginning with the main control module (main program).
- Modules subordinate to the main control module are incorporated into the structure followed by their subordinates, and so on.

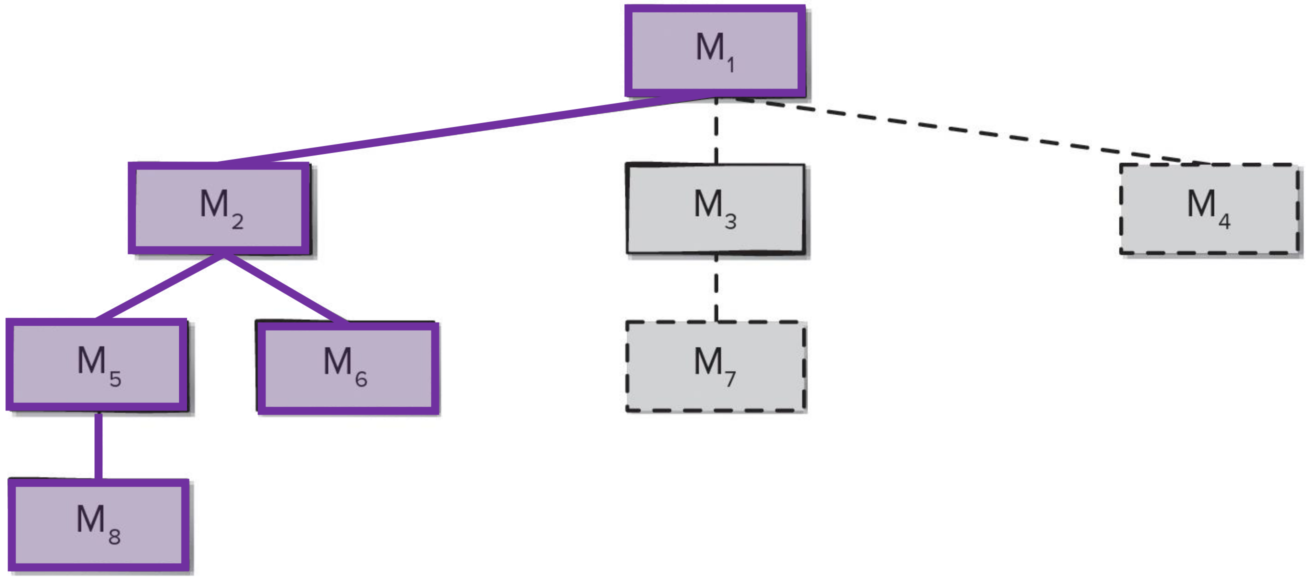


Approaches to Integration Testing

Top-Down Integration Testing

Two approaches to moving downward:

- **Depth-first integration:** integrates all components on a major control path of the program structure before starting another major control path.

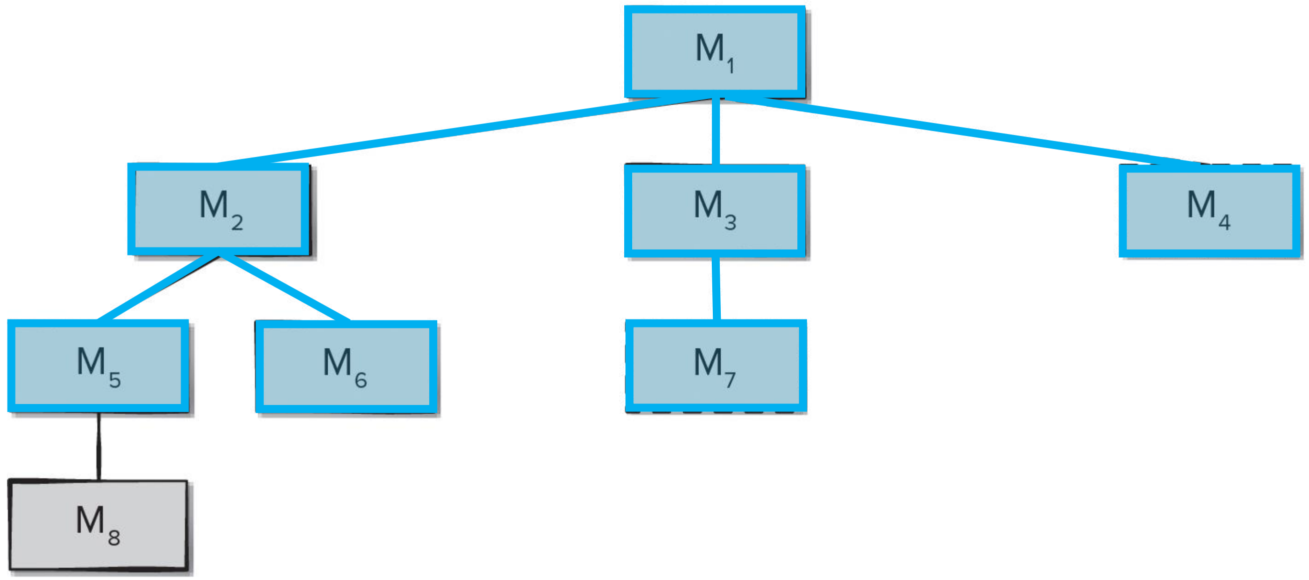


Approaches to Integration Testing

Top-Down Integration Testing

Two approaches to moving downward:

- **Depth-first integration:** integrates all components on a major control path of the program structure before starting another major control path.
- **Breadth-first integration:** incorporates all components directly subordinate at each level, moving across the structure horizontally before moving down to the next level of subordinates.



Approaches to Integration Testing

Top-Down Integration Testing

1. The **main control module is used as a test driver**, and **stubs** are substituted for all components directly subordinate to the main control module.

2. Depending on the integration approach selected (depth or breadth first), **subordinate stubs are replaced one at a time with actual components**.

3. Tests are conducted as each component is integrated.

4. On completion of each set of tests, another stub is replaced with the real component.

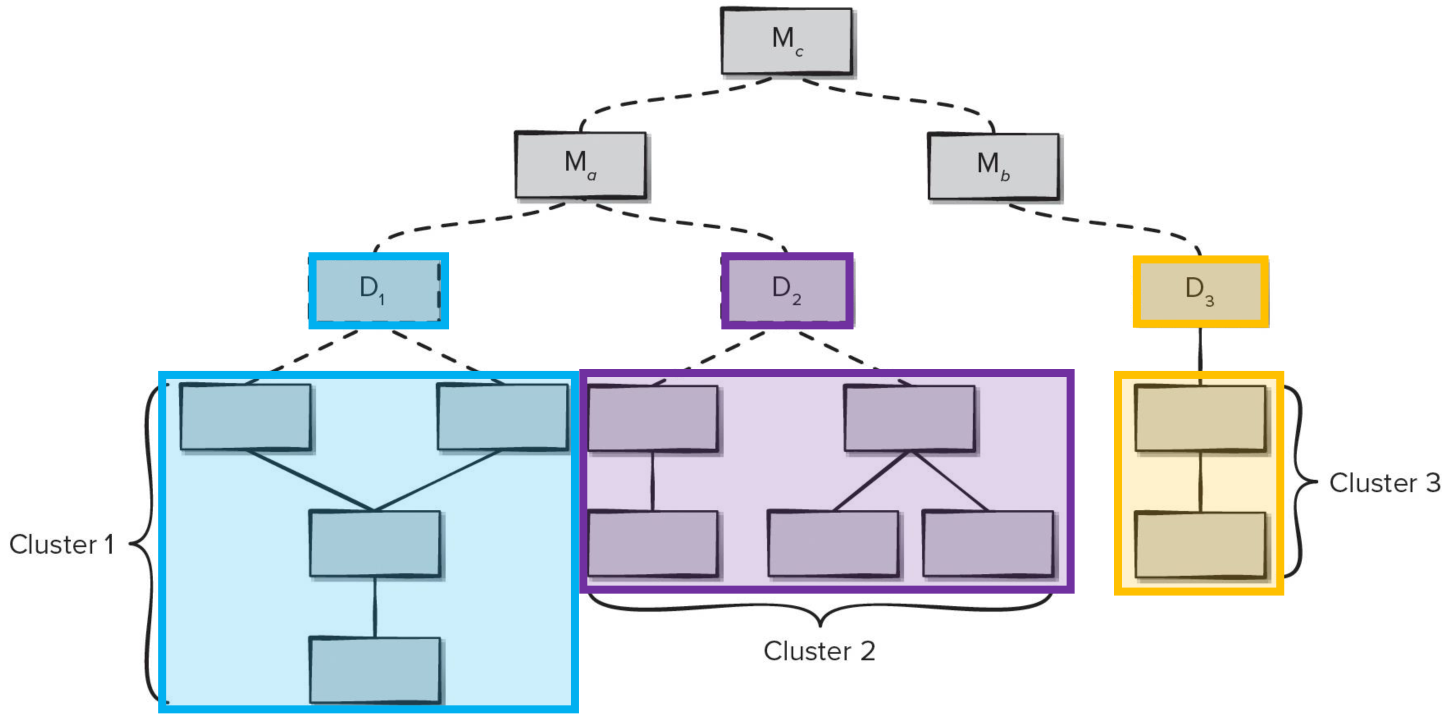
5. **Regression testing** may be conducted to ensure that new errors have not been introduced.

Approaches to Integration Testing

Bottom-Up Integration Testing

- **Incremental approach** that begins construction and testing with **atomic components** at the **lowest levels** in the program structure.

1. Low-level components are **combined into clusters** (builds) that **perform a specific software subfunction**.
2. A **driver** (a control program for testing) is written to **coordinate test-case input and output**.
3. The cluster is tested.



Approaches to Integration Testing

Bottom-Up Integration Testing

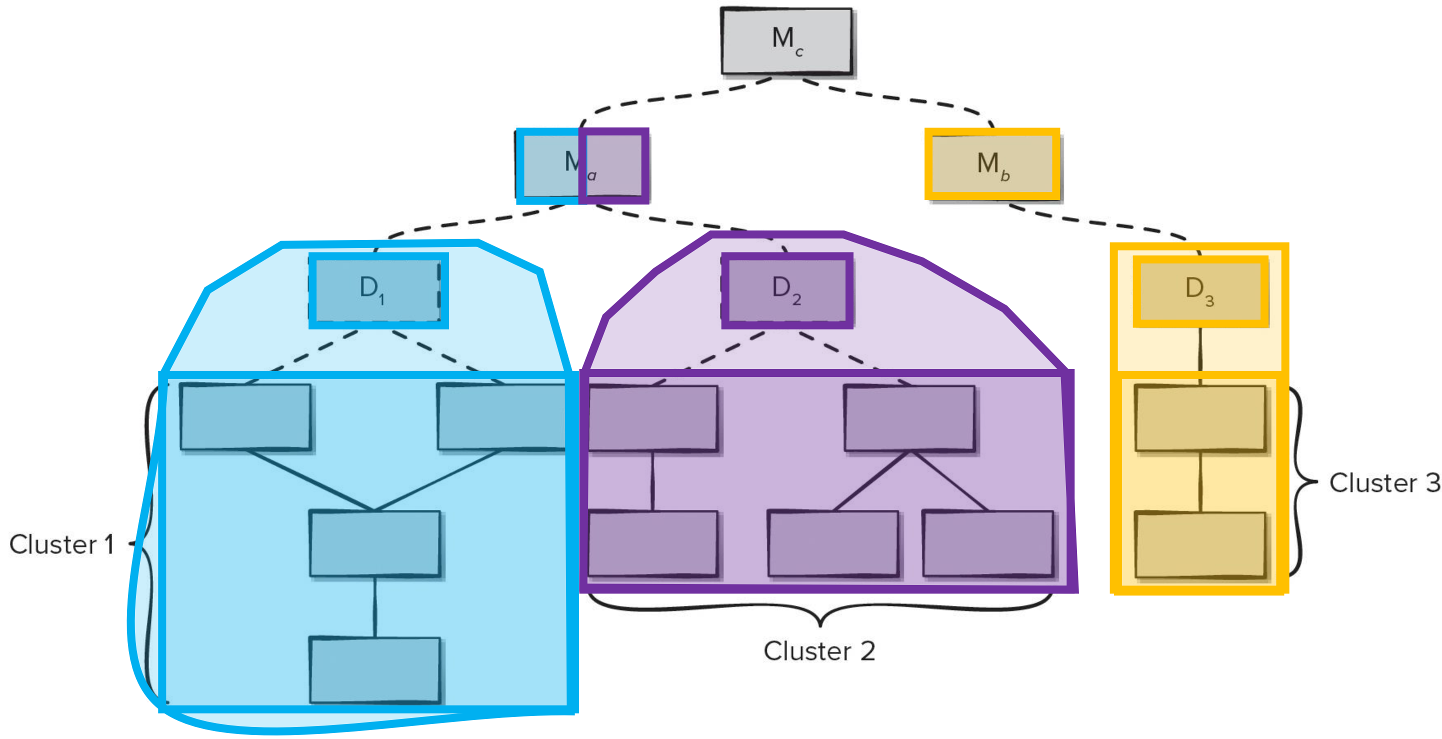
- **Incremental approach** that begins construction and testing with **atomic components** at the lowest levels in the program structure.

1. Low-level components are **combined into clusters** (builds) that **perform a specific software subfunction**.

2. A **driver** (a control program for testing) is written to **coordinate test-case input and output**.

3. The cluster is tested.

4. Drivers are removed and **clusters are combined**, **moving upward** in the program structure.



Approaches to Integration Testing

Continuous Integration

- **Incremental approach** that focuses on of merging components into the evolving software increment **at least once a day**.

- This is a **common practice for teams following agile development** practices such as XP or DevOps.

- Integration testing **must take place quickly and efficiently** if a team is attempting to **always have a working program** in place as part of continuous delivery.

- Makes heavy use of **Smoke Testing** and **automated testing/deployment**.

Smoke Testing

*“The **smoke test** should exercise the entire system from end to end. It does not have to be exhaustive, but it should be capable of exposing major problems. The **smoke test** should be thorough enough that if the build passes, you can assume that it is stable enough to be tested more thoroughly.”*

- Steve McConnell

Smoke Testing

1. **Software components** that have been translated into code are **integrated into a build** that includes all data files, libraries, reusable modules, and components required to **implement one or more product functions**.

2. A **series of tests is designed to expose "show-stopper" errors** that will keep the build from properly performing its function, causing the project to fall behind schedule.

3. The **build is integrated** (either top-down or bottom-up) with other builds, and the entire product (in its current form) is **smoke tested daily**.

Smoke Testing

Advantages

- **Integration risk is minimized**, since **smoke tests** are run daily.
- **Quality of the end product is improved**, functional and architectural problems are uncovered early.
- **Error diagnosis and correction are simplified**, errors are most likely in (or caused by) the new build.
- **Progress is easier to assess**, each day more of the final product is complete.
- **Smoke testing** resembles **regression testing** by ensuring newly added components do not interfere with behaviours of existing components.

Regression Testing

- Regression testing is the **re-execution of some subset of tests** that have already been conducted to **ensure that changes have not propagated unintended side effects**.

- Run whenever software is corrected or some aspect of the software configuration (the program, its documentation, or the data that support it) is changed.

- Regression testing helps to **ensure that changes do not introduce unintended behaviour or additional errors**.

- Regression testing may be **conducted manually**, by re-executing a subset of all test cases **or using automated testing tools**.

Regression Testing

The regression test suite contains **three different classes of test cases**:

1. A **representative sample** of tests that will exercise all software functions.
2. Additional **tests that focus on software functions** that are likely to be **affected by the change**.
3. Tests that focus on the software **components that have been changed**.

As integration proceeds, the **number of** regression tests **can grow quite large**; therefore, the regression test suite should be designed to **include only those tests that address one or more classes of errors** of each of the major program functions.

Validation Testing

- **Software validation** is achieved through a series of **validation tests** that demonstrate **conformity with requirements model** (e.g. user stories, use cases, etc.).
- A **test plan** outlines the classes of tests to be conducted and a **test procedure** defines specific test cases that are designed to ensure that:
    - All **functional requirements** are satisfied.
    - All **behavioural characteristics** are achieved.
    - All **content is accurate and properly presented.**
    - All **performance requirements** are attained
    - **Documentation** is correct.
    - **Usability** and other **nonfunctional requirements** are met.

Validation Testing

- Preformed after **integration testing** and uses **black-box testing** methods.
- A ***deficiency list*** is created when a deviation from a specification is uncovered and their resolution is negotiated with all stakeholders.

System Testing

- Software is only one element of a larger computer-based system.
- Ultimately, the **software is incorporated with other system elements** (hardware, people, information, and procedures) and a **series of system integration and validation tests** are conducted.
- These tests fall outside the scope of the software process and are not conducted solely by software engineers.
- That said, steps taken during software design and testing can greatly improve the probability of successful software integration in the larger system.

Types of System Testing

Recovery Testing

- Forces the software to fail in a variety of ways and verifies that recovery is properly performed

Security Testing

- Verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration

Stress Testing

- Executes a system in a manner that demands resources in abnormal quantity, frequency, or volume

Performance Testing

- Tests the run-time performance of software within the context of an integrated system

Deployment Testing

- Exercises the software in each environment in which it is to operate, examining all installation procedures and tools that will be used
- Sometimes also called configuration testing