CS3213 Project – Week 11

Integration Testing | 30-03-2022

- ❏ Introduction to Integration Testing
- ❏ Integration Strategies
- ❏ Summary of Testing Strategies

Integration Problems (1/2)

2 UNIT TESTS, 0 INTEGRATION TESTS
via reddit.com/r/programmerhumor

https://c.tenor.com/7c9bvnQbGCIAAAAd/unittest-unit.gif

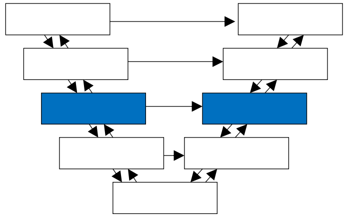
Integration Problems (2/2)

https://c.tenor.com/T7gxakoKzAAAAAAd/unit_vs_integration_tests.gif

Integration

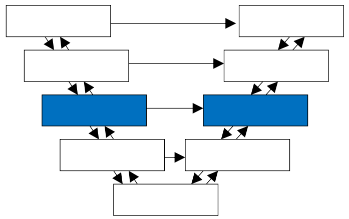
The process of **combining** software components, hardware components, or both into an **overall system**.

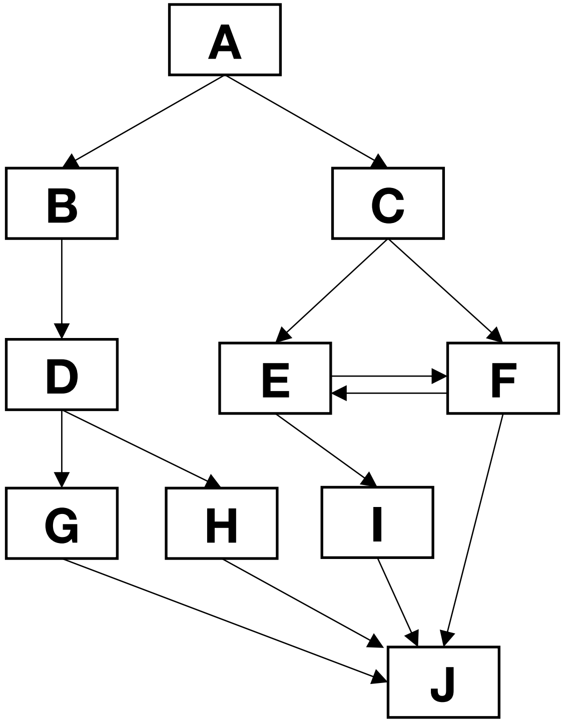
[IEEE Std 610.12 (1990)]



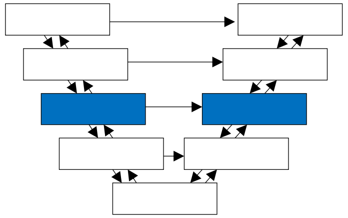
- ❑ The **software architecture** provides the construction and assembly plan (levels/granularity of integration).
- ❑ Typical problem: **Incompatible interfaces** (syntactic and semantic conflicts due to different understanding of the specification and sloppiness or - far worse - lack of specification)
- ❑ Challenge:
  Components are available at different points in time

Sample Architecture
(Component Dependencies)





Integration Testing

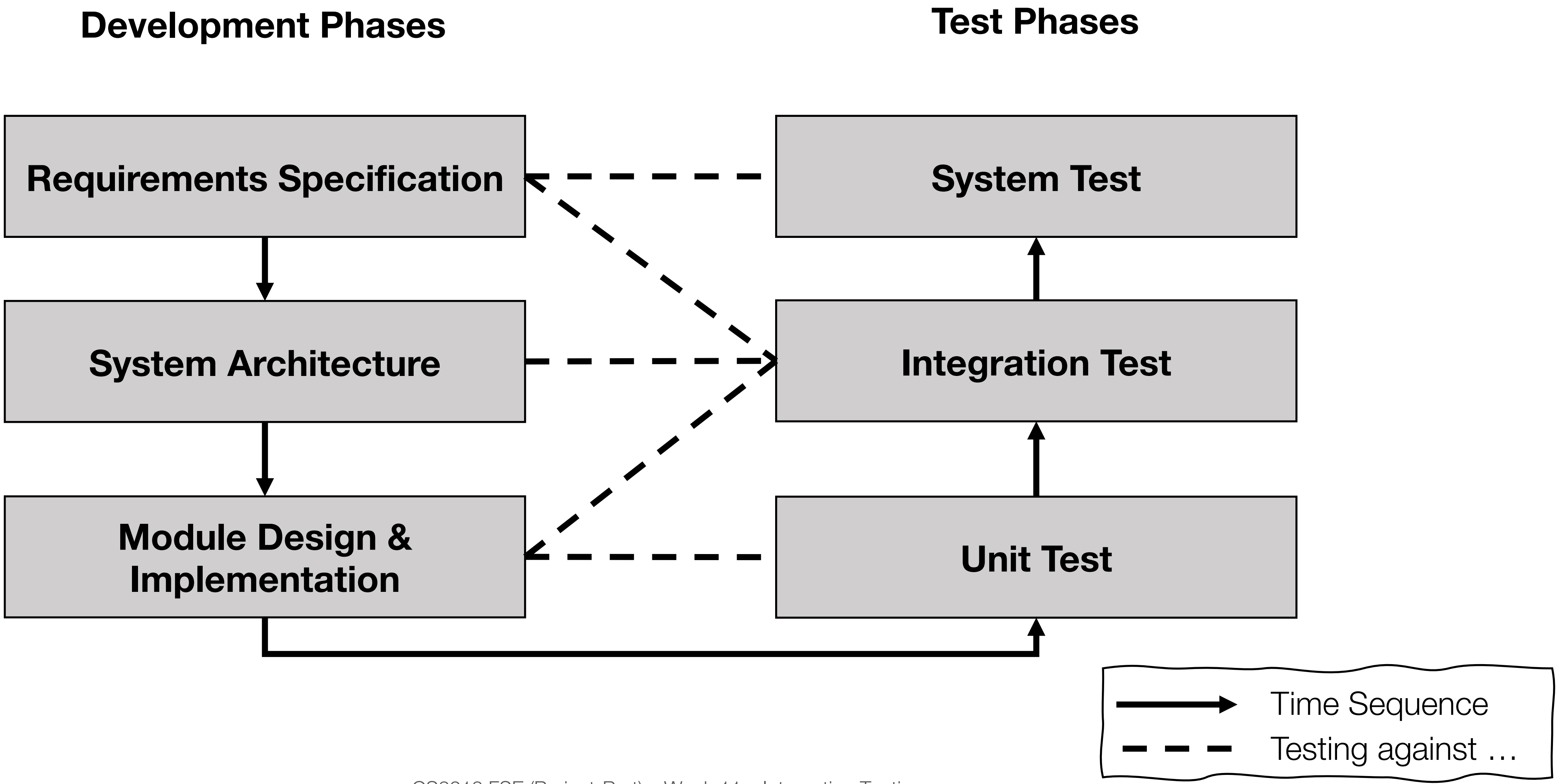


Testing in which software components, hardware components, or both are combined and tested to **evaluate the interaction** between them.

[IEEE Std 610.12 (1990)]

- ❑ Integration tests serve for the (syntactical and semantic) **evaluation of the interfaces**.
- ❑ It is less concerned with the errors of the individual components (unit testing) but with **consistency problems** between the components.
- ❑ When everything is integrated, the system test can follow.

Relationship between testing and developing software



Development Phases

Test Phases

Requirements Specification

System Architecture

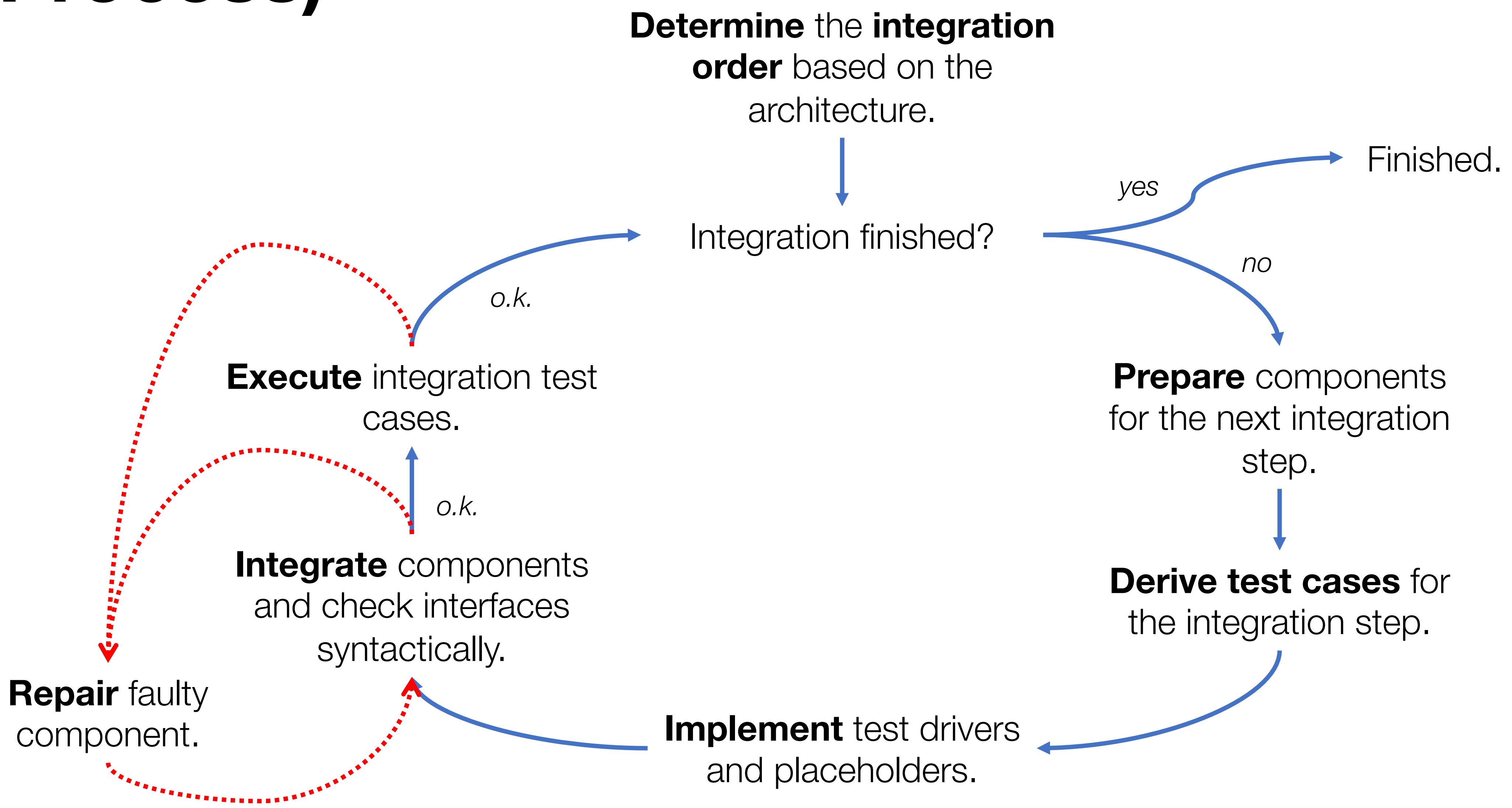
Module Design & Implementation

System Test

Integration Test

Unit Test

Integration and Integration Testing (Process)



Determine the **integration order** based on the architecture.

Integration finished?

yes

Finished.

no

Prepare components for the next integration step.

Derive test cases for the integration step.

Implement test drivers and placeholders.

Integrate components and check interfaces syntactically.

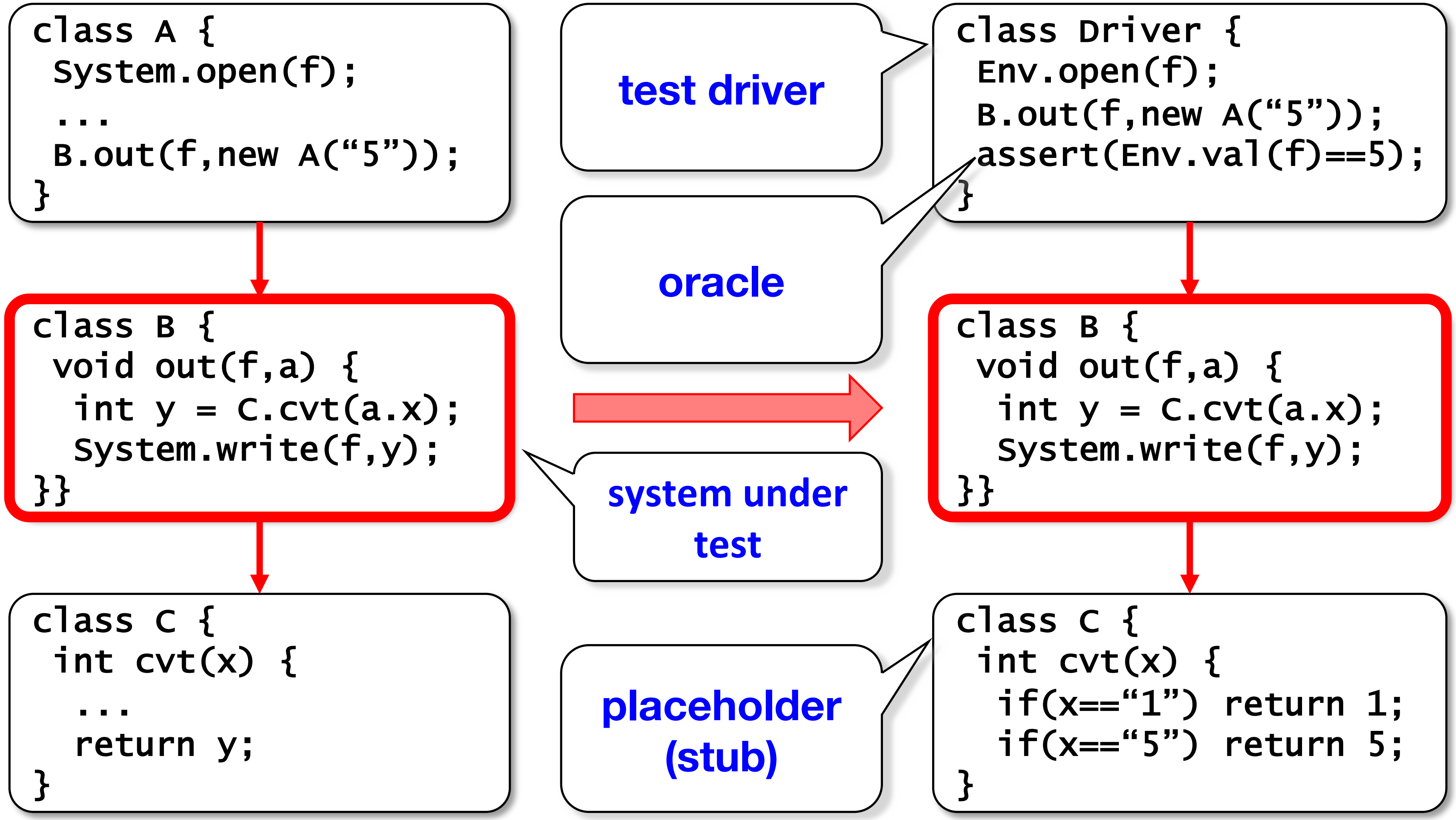
o.k.

Execute integration test cases.

o.k.

Repair faulty component.

Integration and Integration Testing (Notation)



```
class A {
 System.open(f);
 ...
 B.out(f,new A(“5”));
}
```

```
class B {
 void out(f,a) {
  int y = C.cvt(a.x);
  System.write(f,y);
}}
```

```
class C {
 int cvt(x) {
  ...
  return y;
}
```

test driver

oracle

system under test

placeholder (stub)

```
class Driver {
 Env.open(f);
 B.out(f,new A(“5”));
 assert(Env.val(f)==5);
}
```

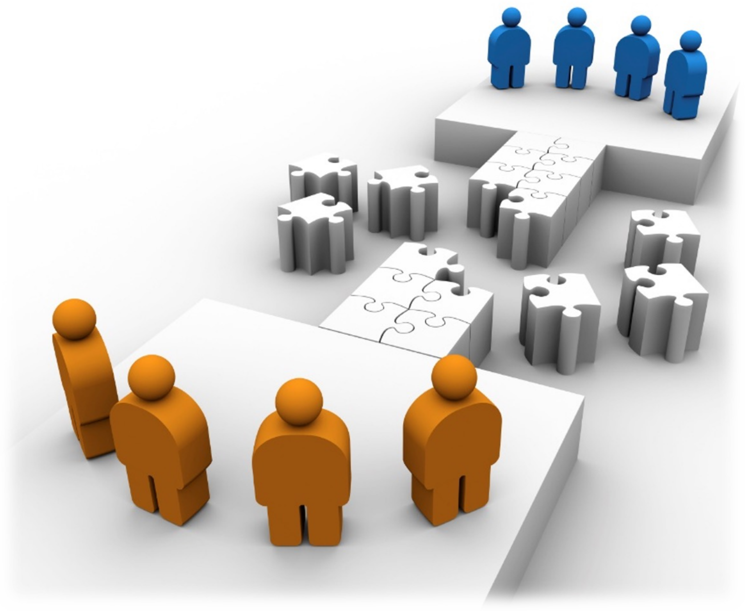
```
class B {
 void out(f,a) {
  int y = C.cvt(a.x);
  System.write(f,y);
}}
```

```
class C {
 int cvt(x) {
  if(x==“1”) return 1;
  if(x==“5”) return 5;
}
```

Integration Strategies

Problem definition

- ❑ In what order are the components integrated?
- ❑ When is it as effective and efficient as possible?
- ❑ Components are ready at different times.
- ❑ Testers should not be idle just because a component is not ready.



This results in different integration strategies...
(next slide)

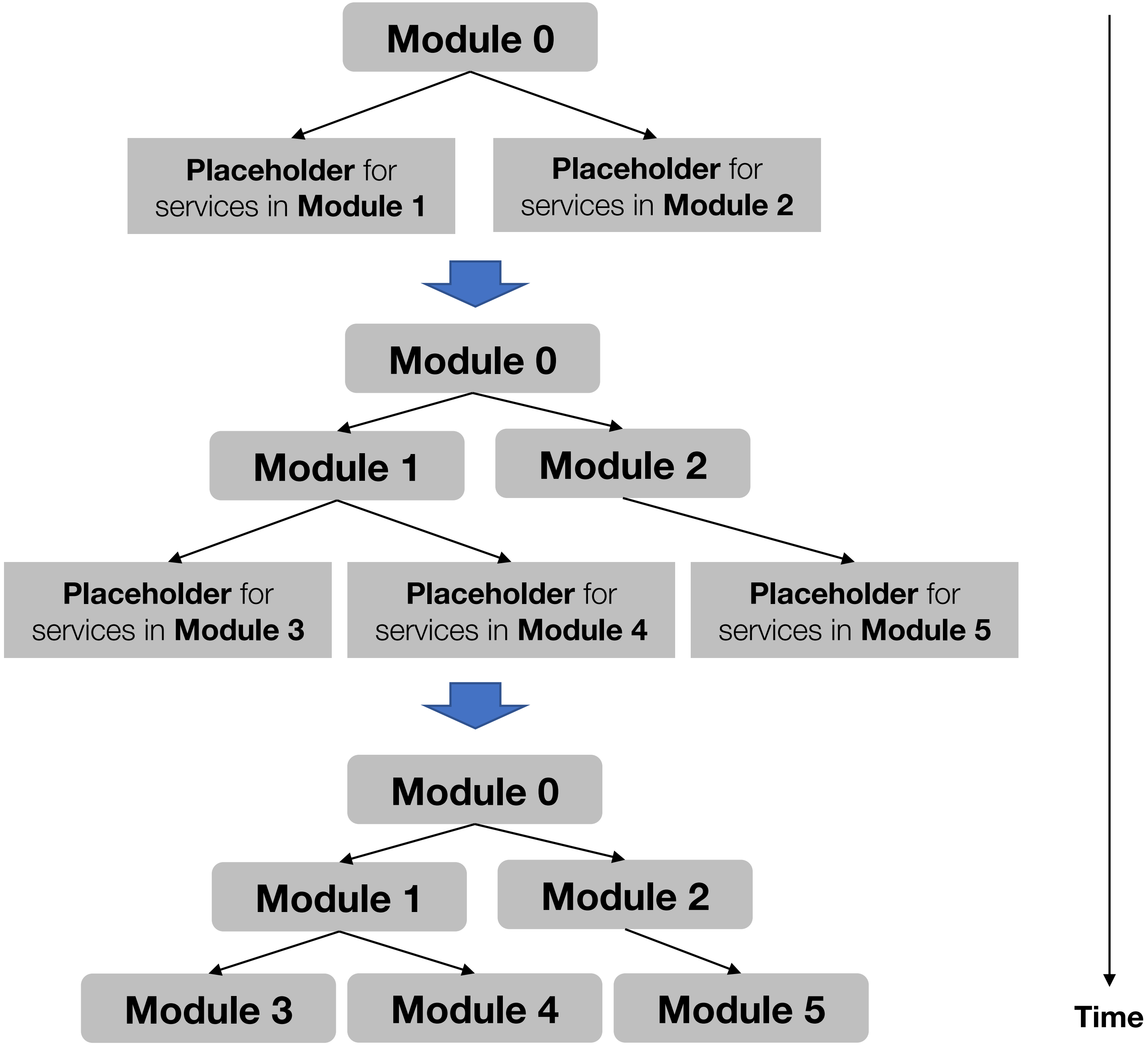
Integration (testing) strategies and procedures

- ❑ Big-bang integration (integration in one step)
- ❑ Incremental integration
  - ❑ Strategies:
    - ❑ Bottom-Up
    - ❑ Top-Down
    - ❑ Outside-In
    - ❑ Continuous Integration
  - ❑ Partially integrated system usually not executable
    → **test drivers** and **placeholders** (stubs/dummies) required
  - ❑ Number of test drivers and placeholders varies depending on strategy
  - ❑ Goal: Minimum effort for test drivers and placeholders!
- ❑ Integration test method: Static vs. dynamic

Big-Bang-Integration

- i.e. **integration in one step**
- in principle **very attractive**, because:
  - System is immediately complete
  - System can be tested without test drivers and placeholders
- **Practically hardly (successfully) possible**, because:
  - Components contain too many errors and inconsistencies
  - System hardly executable
  - Fault Localization
- **Unfortunately often encountered in practice**
- Therefore only possible if **high quality of components** and **good consistency of interfaces** are ensured before integration

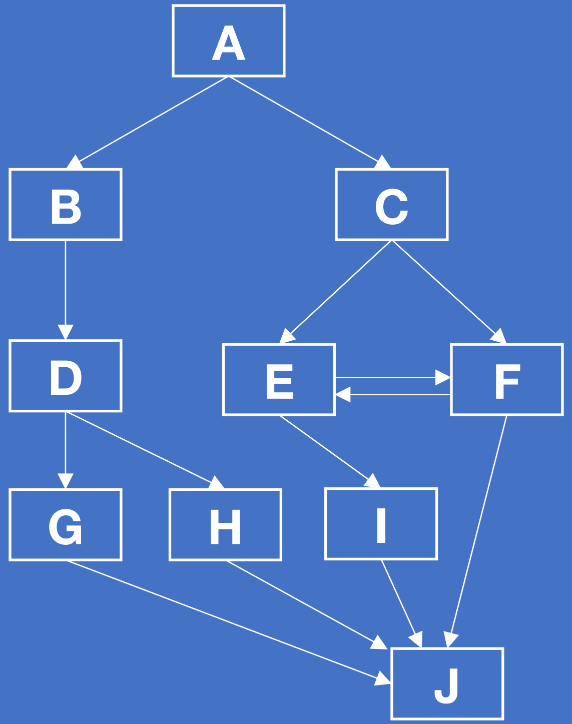
Top-Down-Integration


Module 0
Placeholder for services in **Module 1**
Placeholder for services in **Module 2**
Module 0
Module 1
Module 2
Placeholder for services in **Module 3**
Placeholder for services in **Module 4**
Placeholder for services in **Module 5**
Module 0
Module 1
Module 2
Module 3
Module 4
Module 5
Time

Top-Down-Integration

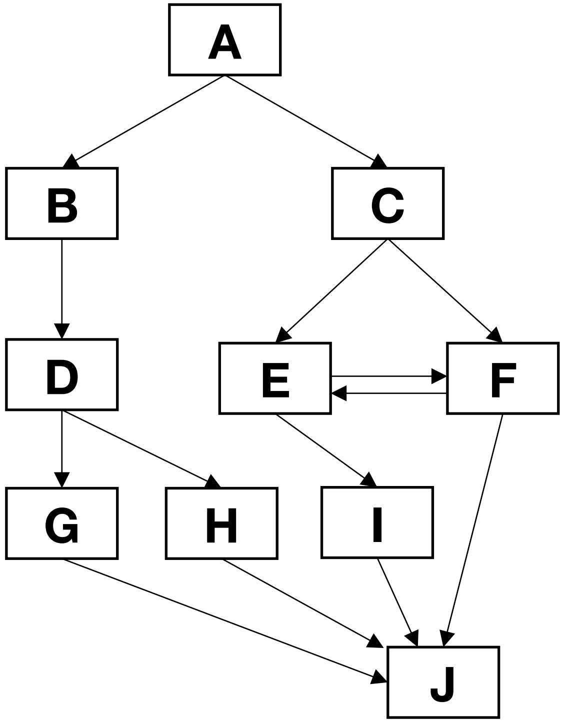
- ❑ **Advantages**:
  - ❑ Important control functionality is tested first.
  - ❑ Already at the beginning a product develops, which lets recognize the rough workflow.
  - ❑ Targeted testing of error handling in case of faulty return values of subordinate routines is possible, since return values are provided by placeholders.
- ❑ **Disadvantages**:
  - ❑ Many placeholders required.
  - ❑ With increasing integration depth the production of certain test situations in more deeply arranged modules becomes more difficult.
  - ❑ Interaction between software under test, system software and hardware is tested late.
  - ❑ Increasing personnel requirements during the test.

Top-Down-Integration (Example)



Top-Down-Integration (Example)

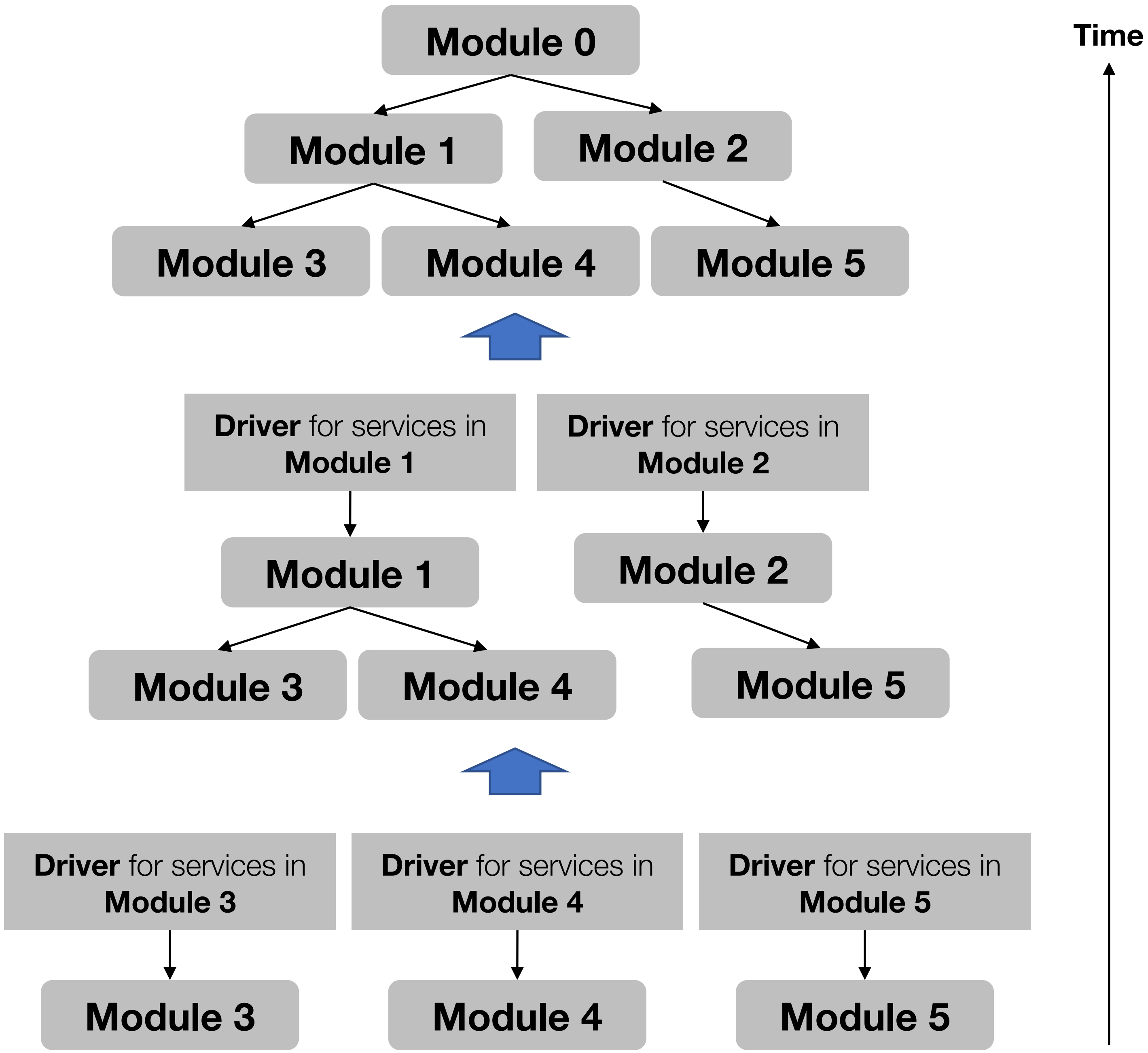
#	A	B	D	G	H	C	F	E	I	J	Driver
0	A	b				c					[d(A)]
1	A	B	d			c					
2	A	B	D	g	h	c					
3	A	B	D	G	h	c				j	
4	A	B	D	G	H	c				j	
5	A	B	D	G	H	C	f	e		j	
6	A	B	D	G	H	C	F	e		j	d(F)
7	A	B	D	G	H	C	F	E	i	j	
8	A	B	D	G	H	C	F	E	I	j	
9	A	B	D	G	H	C	F	E	I	J	



Integrated Component; **Placeholder (Stub)**

(Driver for F emulates queries for E)

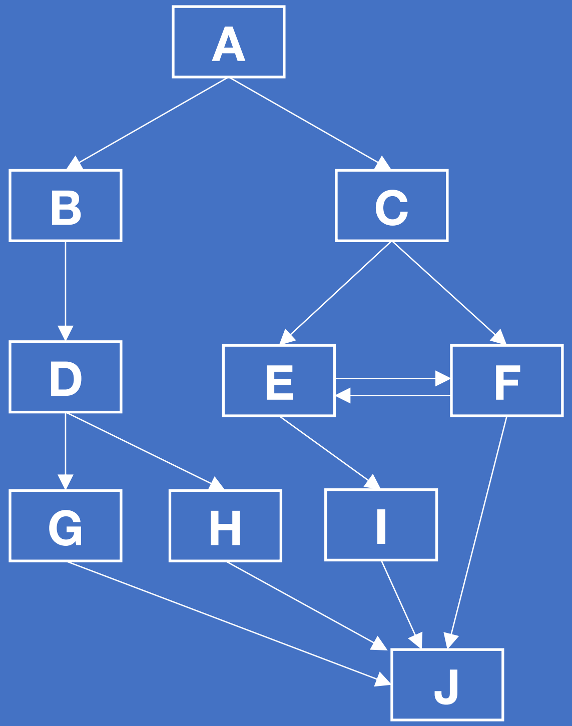
Bottom-Up-Integration


Time
Module 0
Module 1
Module 2
Module 3
Module 4
Module 5
Driver for services in Module 1
Driver for services in Module 2
Module 1
Module 2
Module 3
Module 4
Module 5
Driver for services in Module 3
Driver for services in Module 4
Driver for services in Module 5
Module 3
Module 4
Module 5

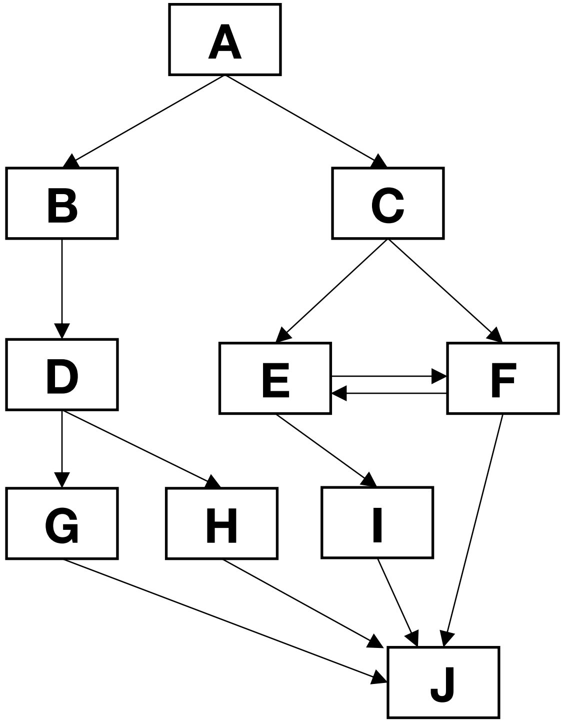
Bottom-Up-Integration

- ❑ **Advantages**:
  - ❑ Interaction between software under test, system software and hardware is tested early.
  - ❑ Since test data inputs are made via drivers, no complex back-calculation of inputs is required.
  - ❑ Intentional **erroneous inputs** to test the error handling are easily possible.
- ❑ **Disadvantages**:
  - ❑ Drivers required.
  - ❑ Focused testing of the error handling for **erroneous return values** of sub-components is hardly possible, since the real components are used.
  - ❑ A presentable product develops only at the very last, since the top/coordinating modules are added only then.
  - ❑ Decreasing manpower requirements as testing progresses.

Bottom-Up-Integration (Example)



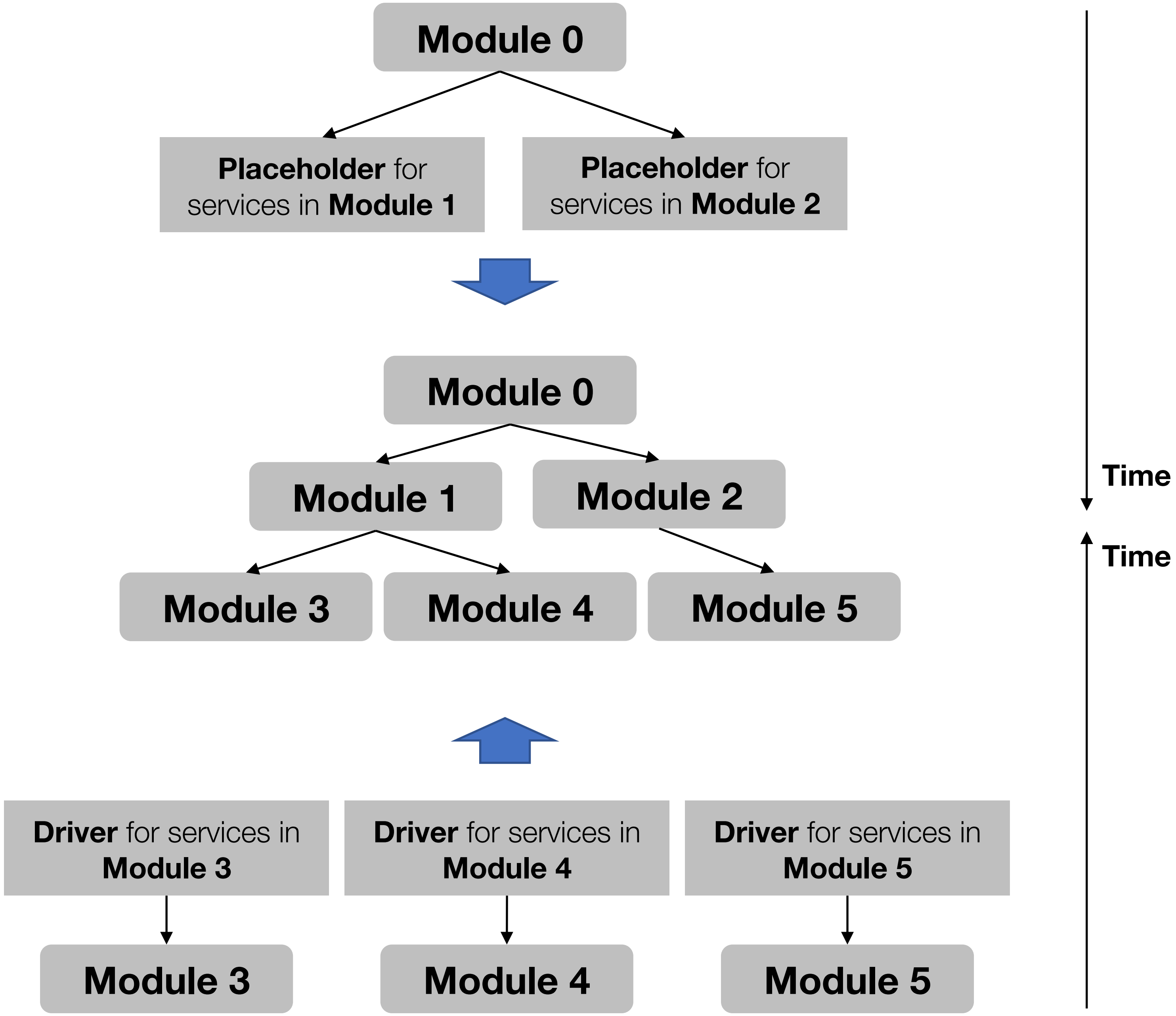
Bottom-Up-Integration (Example)



#	J	I	H	G	F	E	D	C	B	A	Driver
0	J										d(J)
1	J	I									d(J),d(I)
2	J	I	H								d(J),d(I),d(H)
3	J	I	H	G							d(J),d(I),d(H),d(G)
4	J	I	H	G	F	e					d(I),d(H),d(G),d(F)
5	J	I	H	G	F	E					d(H),d(G),d(F),d(E)
6	J	I	H	G	F	E	D				d(F),d(E),d(D)
7	J	I	H	G	F	E	D	C			d(D),d(C)
8	J	I	H	G	F	E	D	C	B		d(C),d(B)
9	J	I	H	G	F	E	D	C	B	A	[d(A)]

Only one **Placeholder (stub)**; but many **drivers** necessary.

Outside-In-Integration


Module 0
Placeholder for services in Module 1
Placeholder for services in Module 2
Module 0
Module 1
Module 2
Module 3
Module 4
Module 5
Driver for services in Module 3
Driver for services in Module 4
Driver for services in Module 5
Module 3
Module 4
Module 5
Time
Time

Outside-In-Integration

- ❑ **Advantages**:
    - ❑ Important control functionality is tested first.
    - ❑ Already at the beginning, a product is created that shows the rough processes.
    - ❑ Targeted testing of error handling
    - ❑ Interaction between software under test, system software and hardware is tested early.
    - ❑ Since test data inputs are made via drivers for those modules that are integrated from the bottom up, no complex back-calculation of inputs is required.
    - ❑ Intentional mis-entry to test error handling is easily accomplished at the bottom of the module system.
    - ❑ The manpower requirement is more constant during integration testing.
- ❑ **Disadvantages**:
    - ❑ Dummies and drivers required.

Static Integration Testing (1/2)

- **Syntax checking** of interfaces:
  - Many modern programming languages allow syntactic consistency checking between interface declarations and their usage.
- **Coupling** categorization:
  - The coupling between two modules is a **measure of their dependency**.
  - Software engineering recognizes several types of coupling. Which of these couplings exist can be determined by static analysis from the implementations.
  - Goal is the **weakest** possible coupling.

Static Integration Testing (2/2)

- ❑ Reveal **hidden dependencies**:
  - ❑ A hidden dependency between two modules exists, e.g., when an external variable is shared that a third module exports.
  - ❑ Such non-obvious dependencies can be detected by static analysis.
- ❑ **Intermodular data flow anomaly** analysis:
  - ❑ Rules analogous to those for variable usages within modules can be defined for interface parameter usages.
  - ❑ A violation of these rules is a data flow anomaly.

Dynamic Integration Testing

- **Prerequisites**:
  - **Executable** system or subsystem
  - Corresponding unit testing has been performed
  - Instrumentation of the test subject, if applicable
- Groups of **testing techniques** analogous to unit testing:
  - **Control flow**-oriented integration test
  - **Data flow**-oriented integration test
  - **Function**-oriented integration test

Integration Principles

❑ **Plan** the integration!

❑ Start integration **early**! (e.g., before coding)

❑ **Do not underestimate** the effort for integration and integration test!

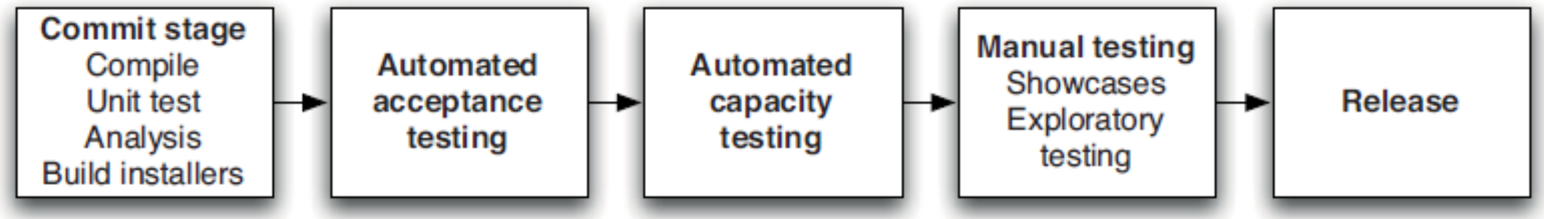
❑ Precisely record the total effort for the integration!

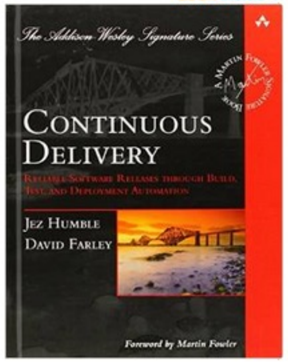
❑ Recognize and reduce **integration risks**!

❑ Repair detected errors cleanly and **completely**!

Continuous { Integration, Delivery, Deployment }

- Goal: Fully automate the integration, delivery, and installation processes.
- Delivery pipeline (Humble, Farley (2010))




The Addison-Wesley Signature Series
Continuous Delivery
Jez Humble
David Farley
Foreword by Martin Fowler

- Continuous Integration Server
  - Hudson/Jenkins
  - Bamboo

Humble, Jez, and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

Tools

mockito

mocking framework for unit tests in Java
https://site.mockito.org

CITRUS

framework for **automated integration tests** supporting a wide range of message protocols and data formats
https://citrusframework.org

Apache JMeter™

framework for **load test** functional behavior and measure **performance**
https://jmeter.apache.org

(Selected) Technologies

- Component frameworks, **Architectural Styles**
  - **3-tier architectures** (Web, Business, Persistence)
    - Java EE, EJB
    - REST (Microservices)
  - **Integration Architectures**
    - SOA
    - Enterprise Service Bus
- Transport Protocols / Exchange Formats / Interface Technology
    - TCP/IP
    - FTP
    - SSH
    - RMI (RPC)
    - JDBC
    - CSV
    - HTTP
    - XML
    - Web Services (SOAP)
    - Messaging
    - JSON
    - E-Mail (SMTP/POP/IMAP)

Overview: Integration Strategies

	Core Idea	Pro	Con
Top-Down ↓	Start point: Component that only depends on others, but has no incoming dependency. Other components are replaced by placeholders.	Little or no drivers needed as high level components are used as test environment.	▪ Can be expensive ▪ Low level components must be replaced with stubs.
Bottom-Up ↑	Start point: component that is not called. Larger sub-systems are created step by step.	No need for stubs.	Needs test drivers for high-level components.
Ad-Hoc ?	Start point: components are integrated as soon as they are ready.	No waiting times.	Needs both, stubs and drivers.
Big Bang ✸	Everything is put together at once.		▪ All errors at once ▪ Difficult fault loalization ▪ Time until integration is wasted

Any remaining question about Integration Testing?

More exercises in the lab tomorrow!

Conclusion

❑ Unit Testing ≠ Integration Testing

❑ Keep deadlines in mind: Final Code submission.
Do not forget the presentations!

Next Week (Project-Part) – Week 12:
Recap Project Topics

➢ Aspects of Version Control

➢ Recap Topics

Last Lecture