LDTS 2021/2022

Rui Maranhão

Last Lecture

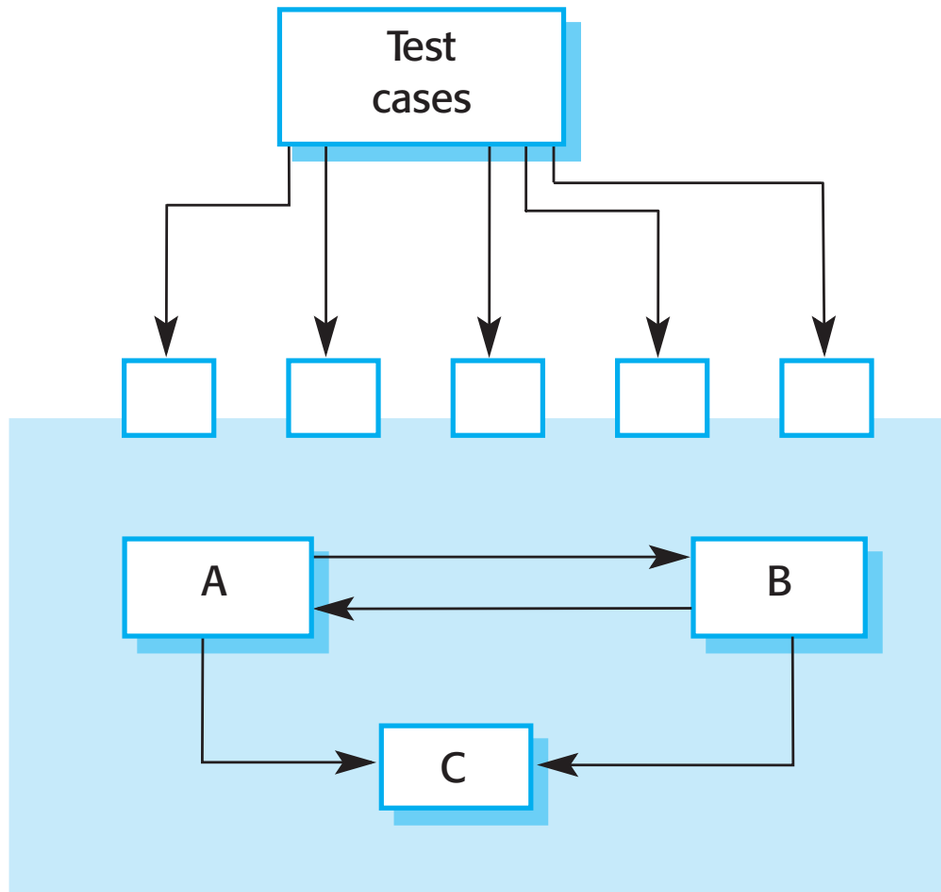
- Test doubles
    - JMockit
    - Spock
- Object testing

Today

- Component testing
- Continuous Integration
- Software Configuration Management Patterns
  - Build patterns

Component Testing

Failures resulting from interaction between units


Test
cases
A
B
C

Interface misuse

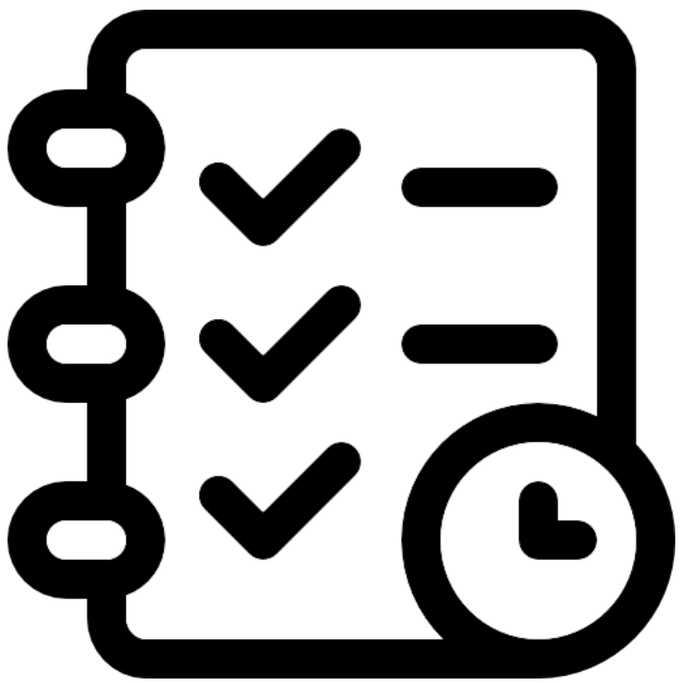
Interface misunderstanding

Test through the interface

Timing errors

Interface erros are one of the most common forms of error in complex systems

It is difficult: some interface errors only manifest themselves under unusual conditions



Incremental Integration and Testing

How can we test several units?

All together

or

step-by-step?

There are dependencies between units

=>

all together

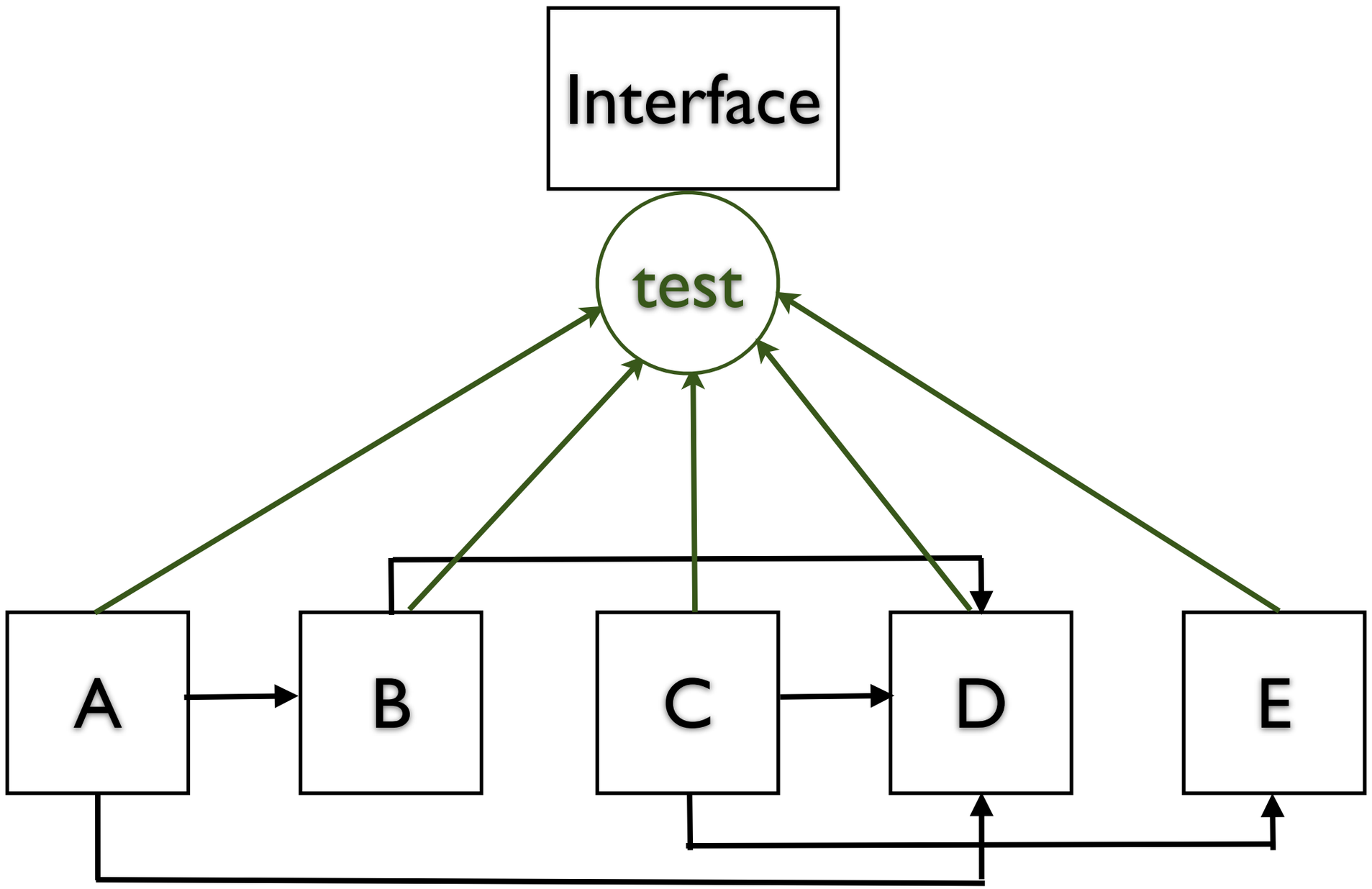
It is difficult to identify the fault behind a failure

=>

step-by-step

Big bang integration

all together


Interface
test
A
B
C
D
E

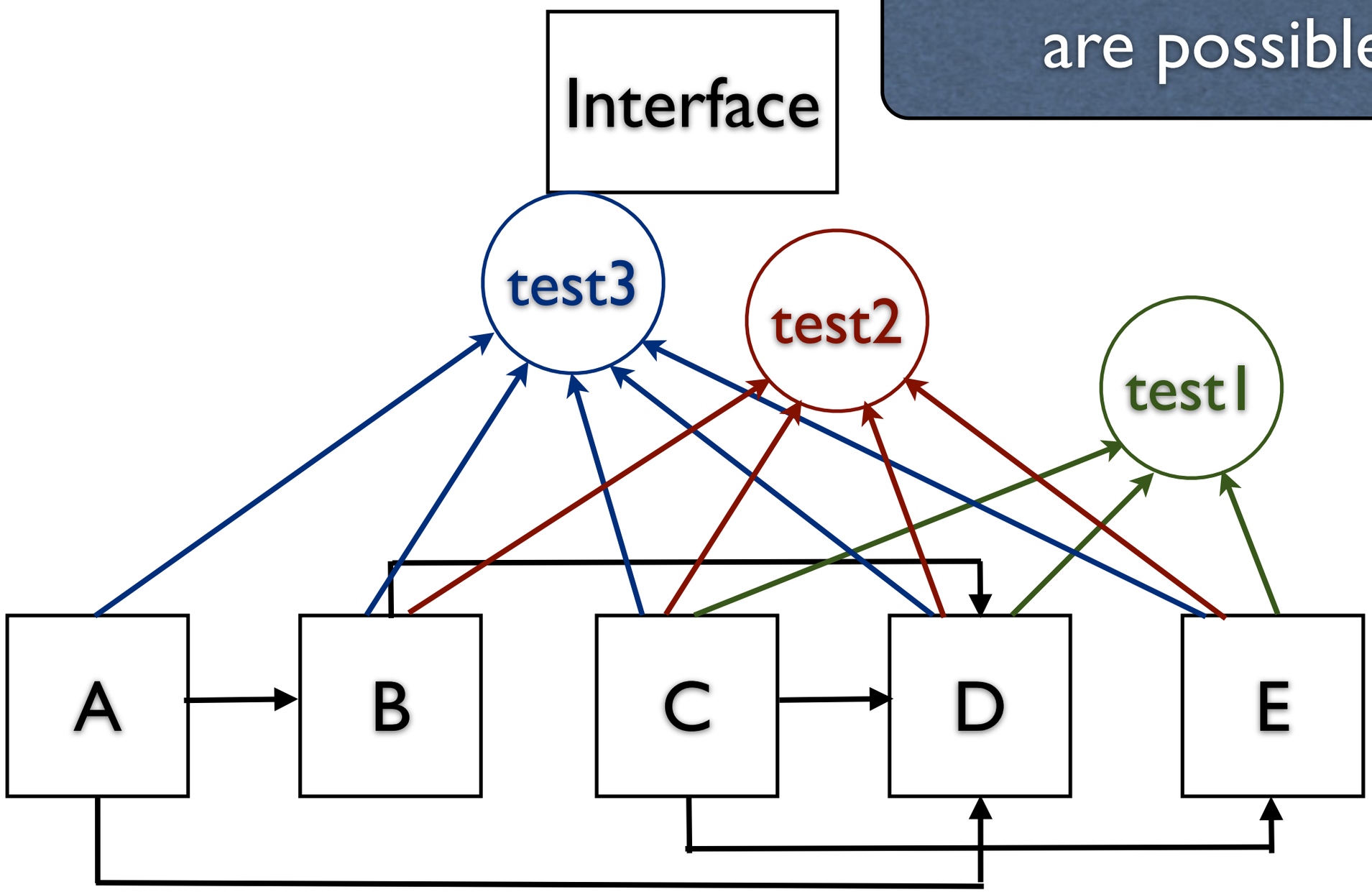
Advantages and Disadvantages

What is more important to test first?

Bottom-up integration

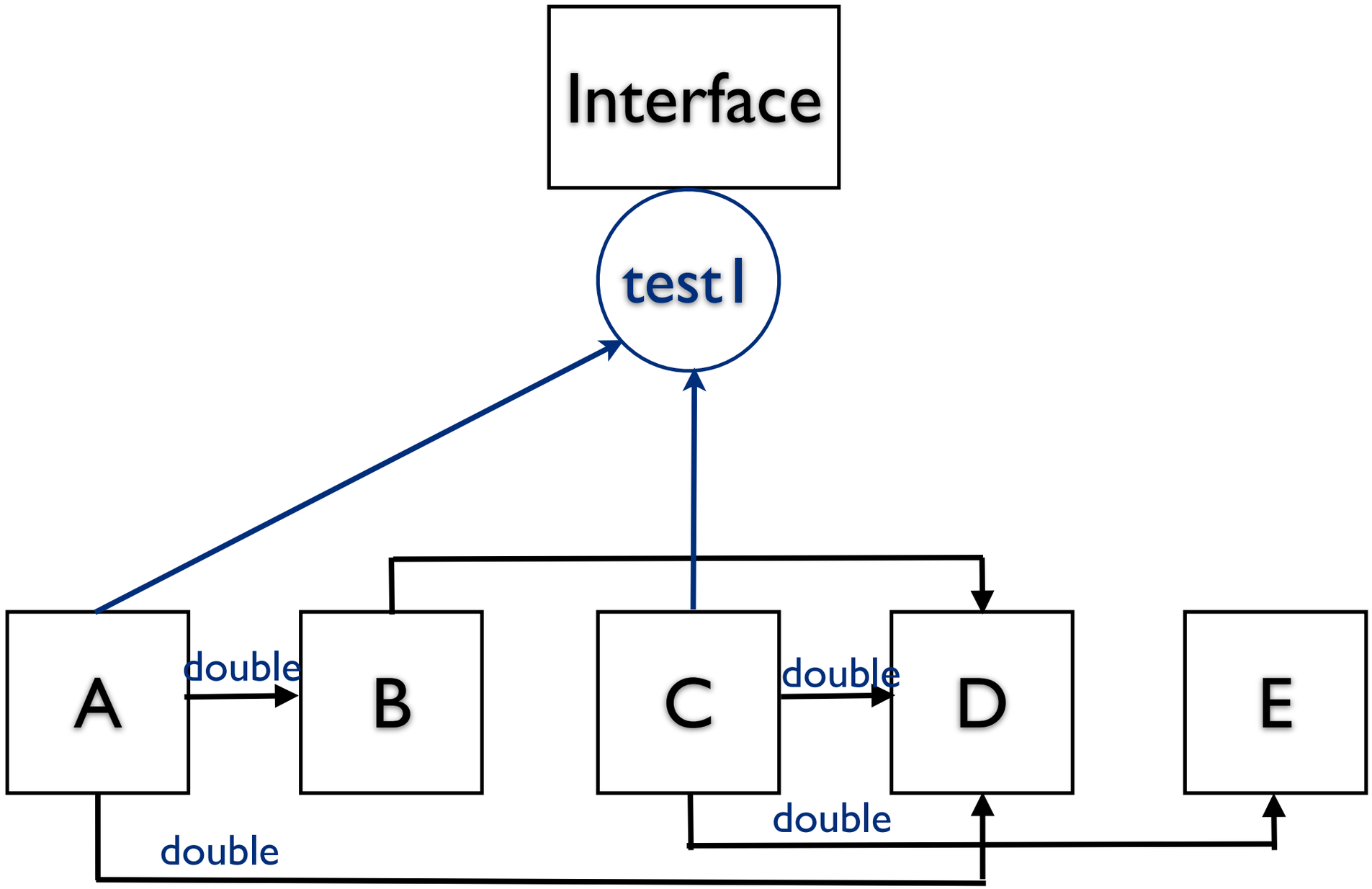
step-by-step

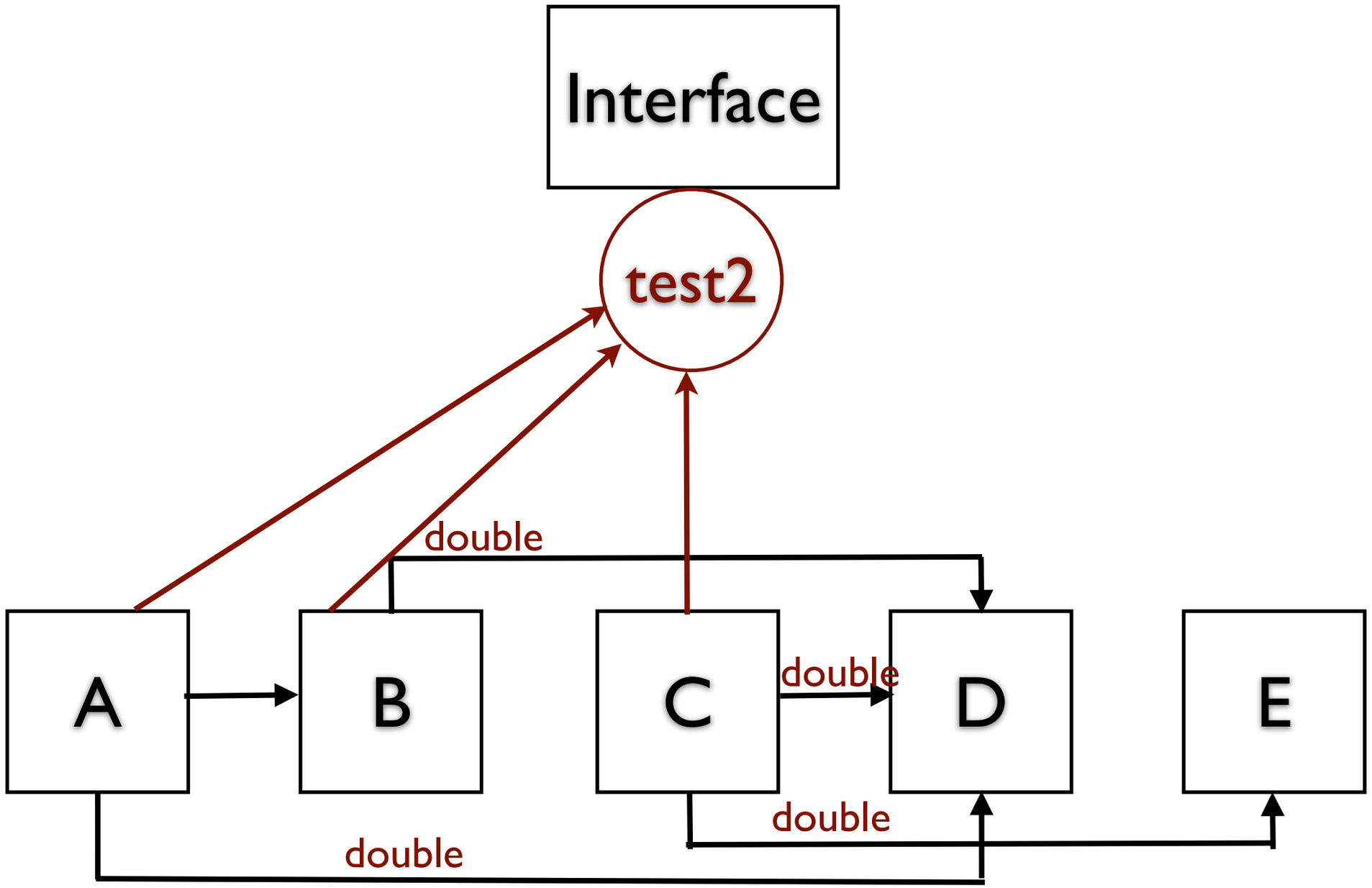
Other combinations are possible

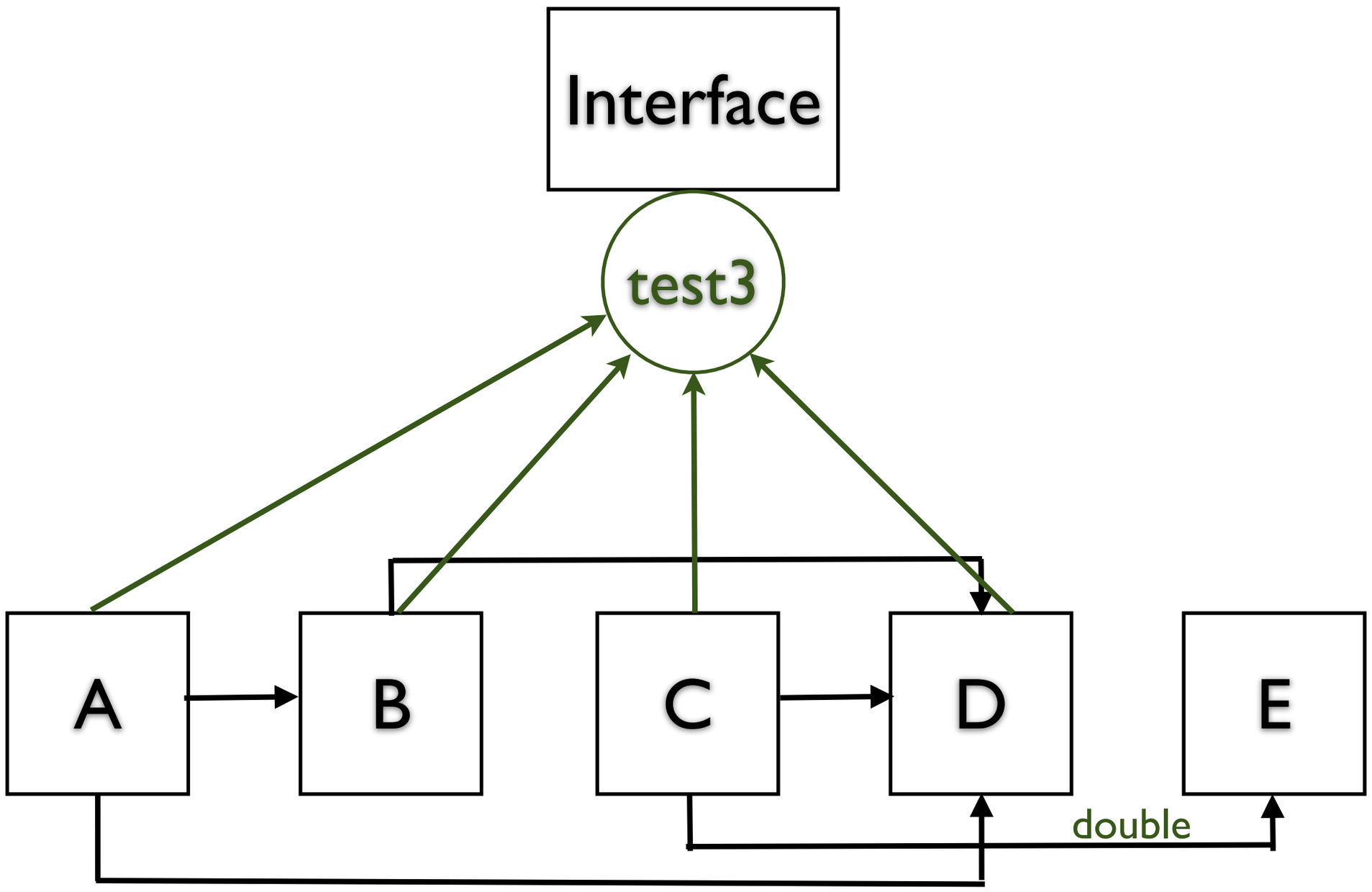


Advantages and Disadvantages

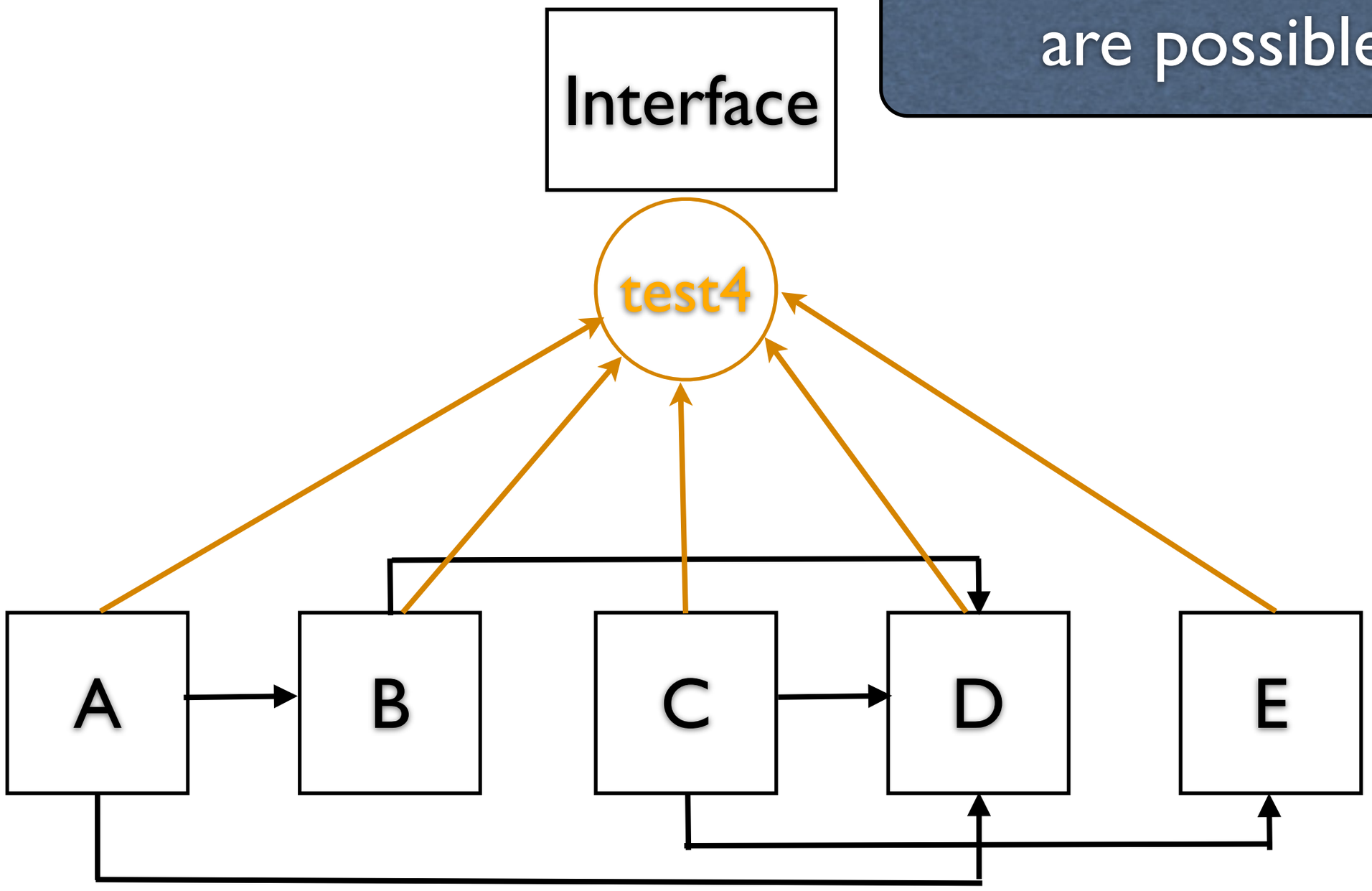
Top-down Integration


Interface
test1
A
double
B
C
double
D
E
double
double


Interface
test2
double
A
B
C
double
D
E
double
double


Interface
test3
A
B
C
D
E
double

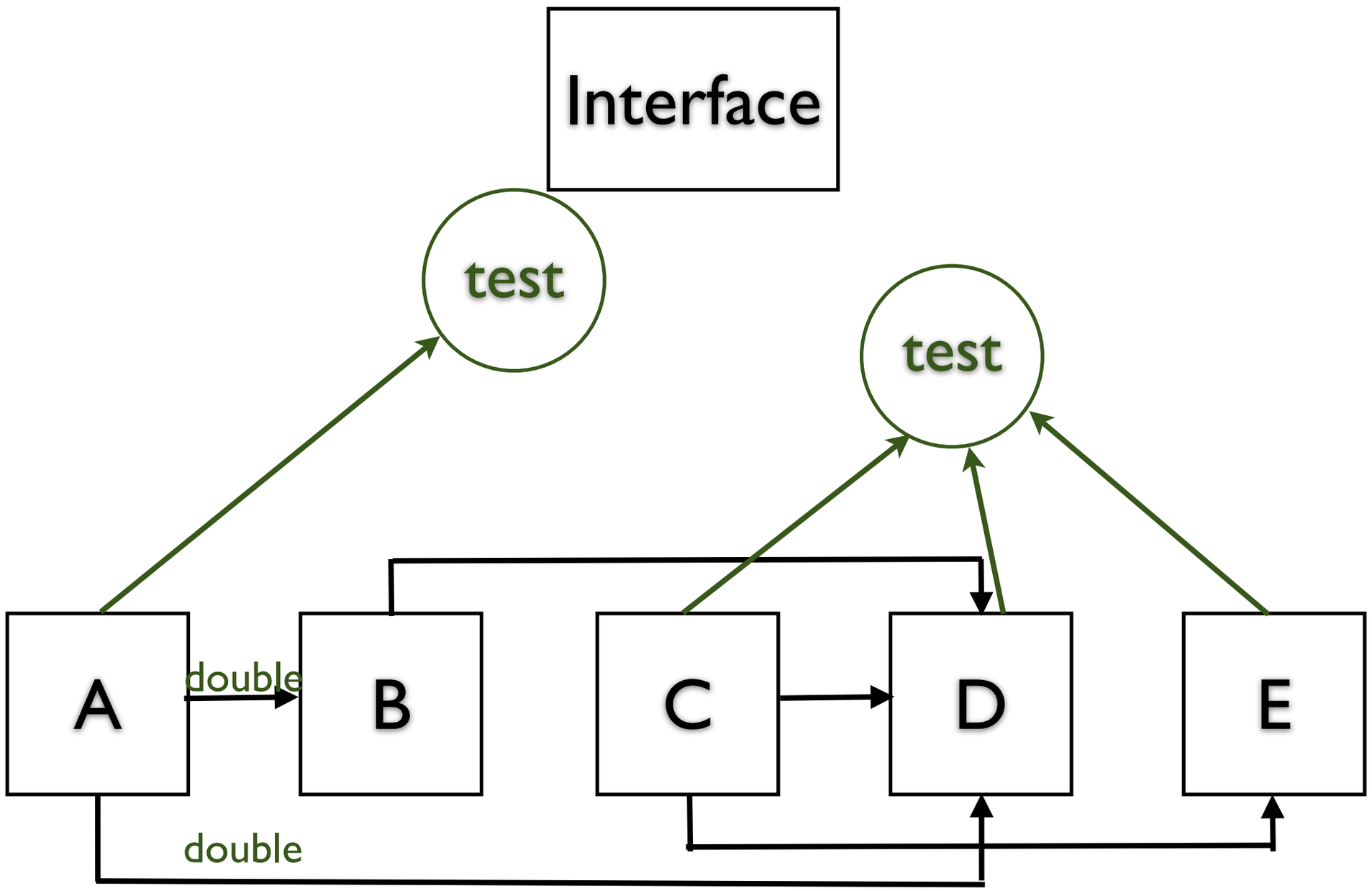
Other combinations are possible

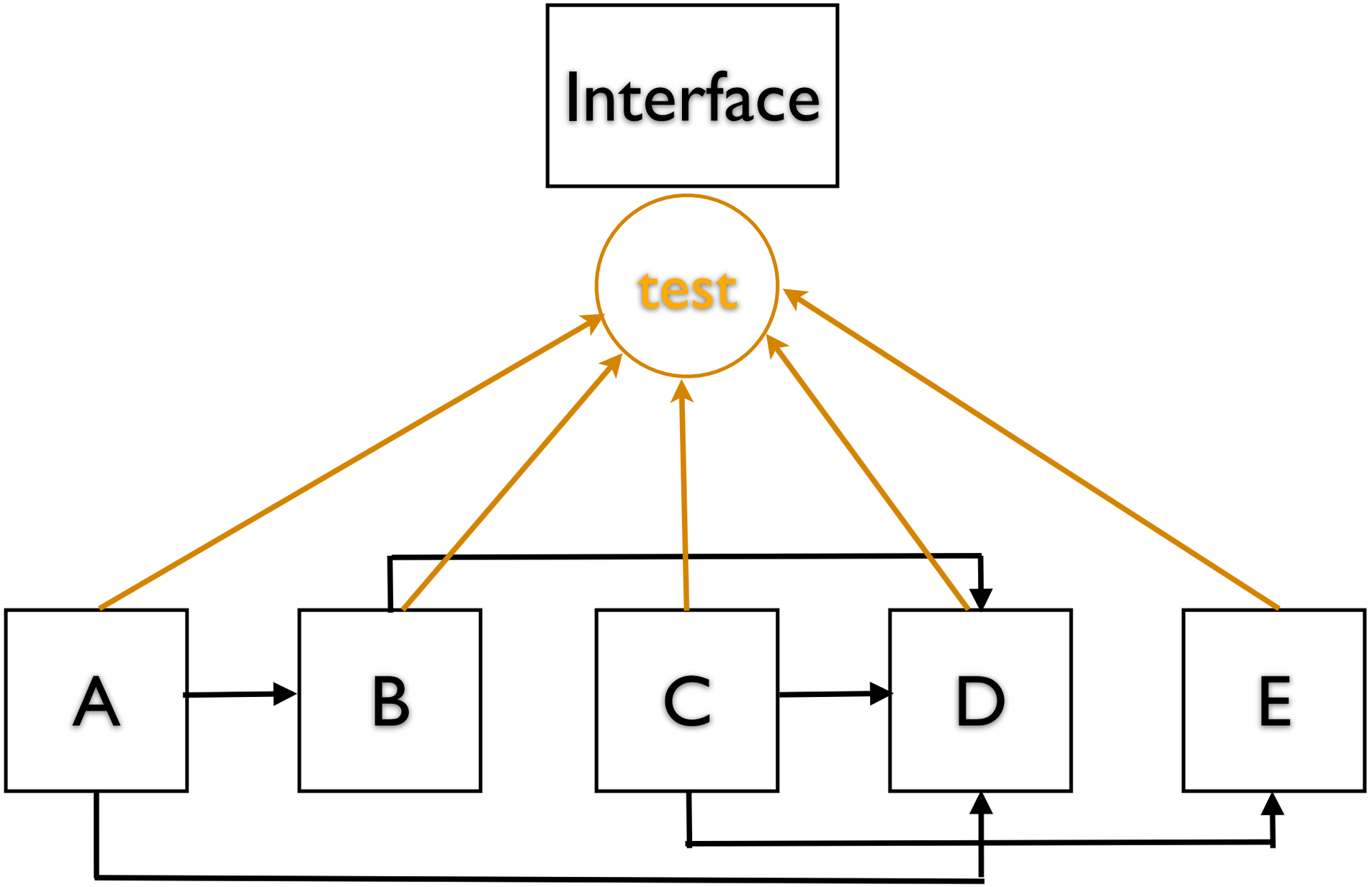


Test high level functionality first

user interface reusable prototypes

Sandwich Integration


Interface
test
test
A
double
B
C
D
E
double


Interface
test
A
B
C
D
E

Delay the integration of some units

reduce dependencies between teams

Integration Testing Coverage?

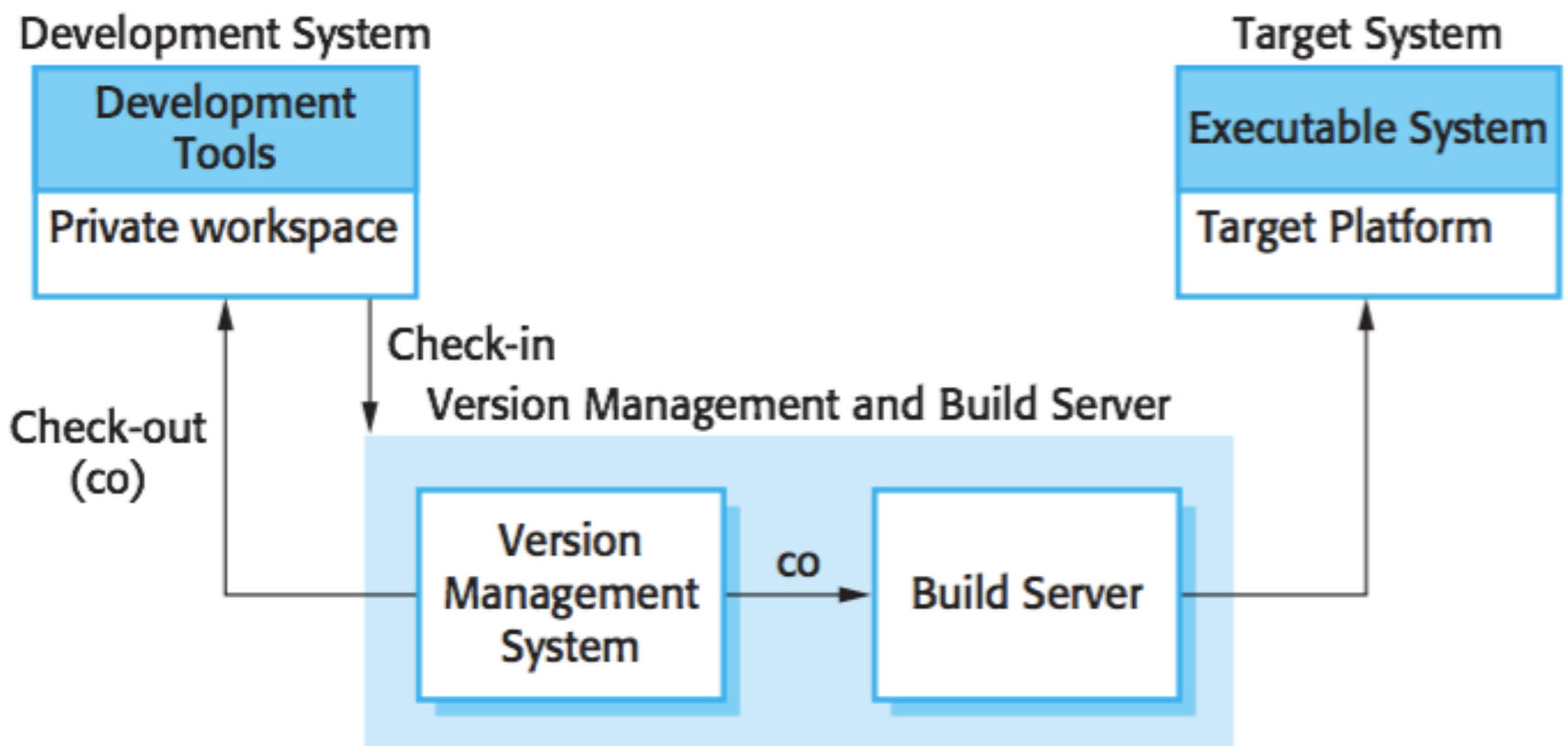
Interface erros are one of the most common forms of error in complex systems

NO!

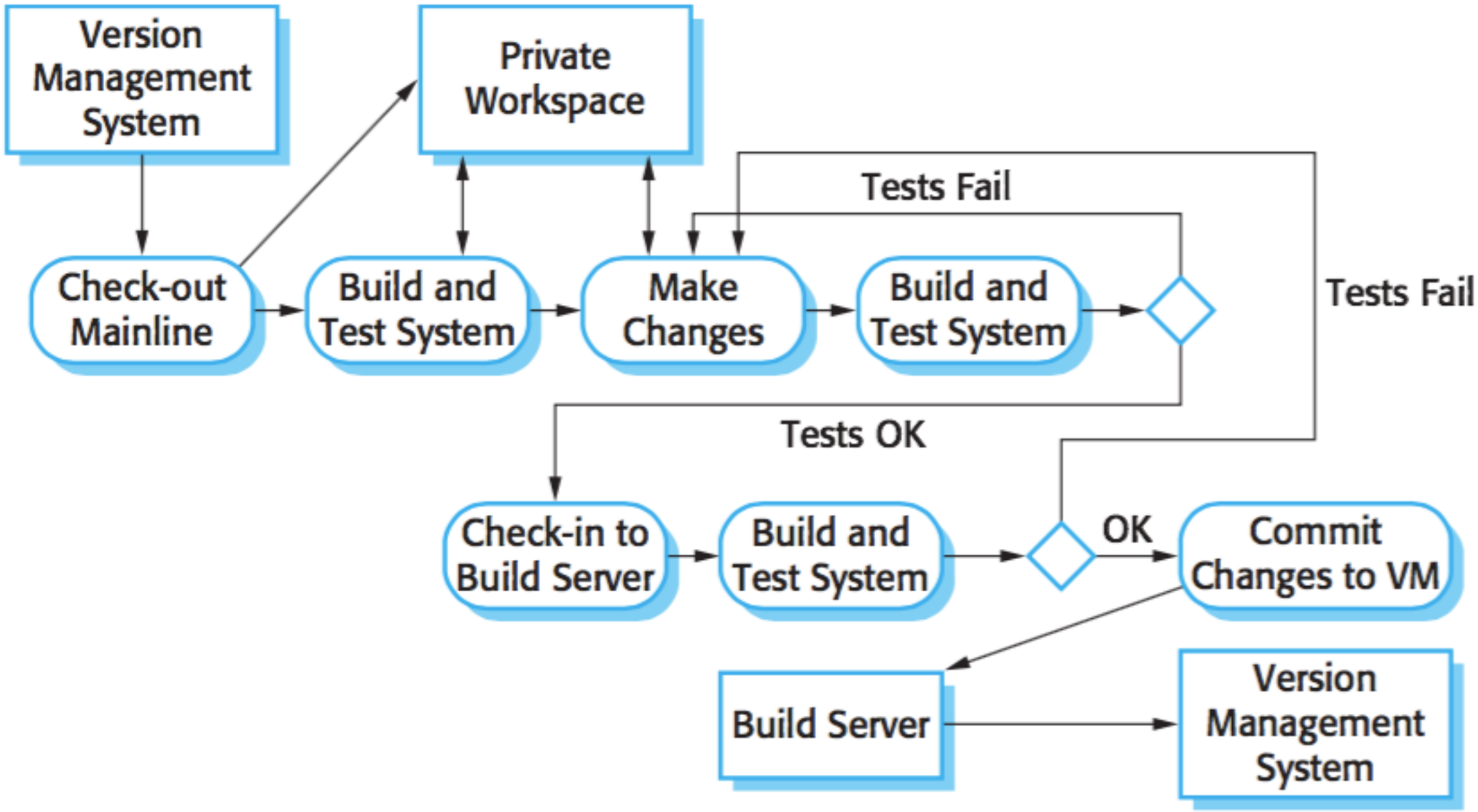
Unit Testing Coverage of the integration code!

https://martinfowler.com/articles/practical-test-pyramid.html

Continuous Integration



(Sommerville, Fig 25.11)



(Sommerville, Fig 25.12)

How is continuous integration organized?

tests may take too long to execute

Collaboration and Isolation

in the build process

Private build

for private development level of quality

Project build

for project development level of quality

Continuous Integration

short cycles of isolation ↔ collaboration

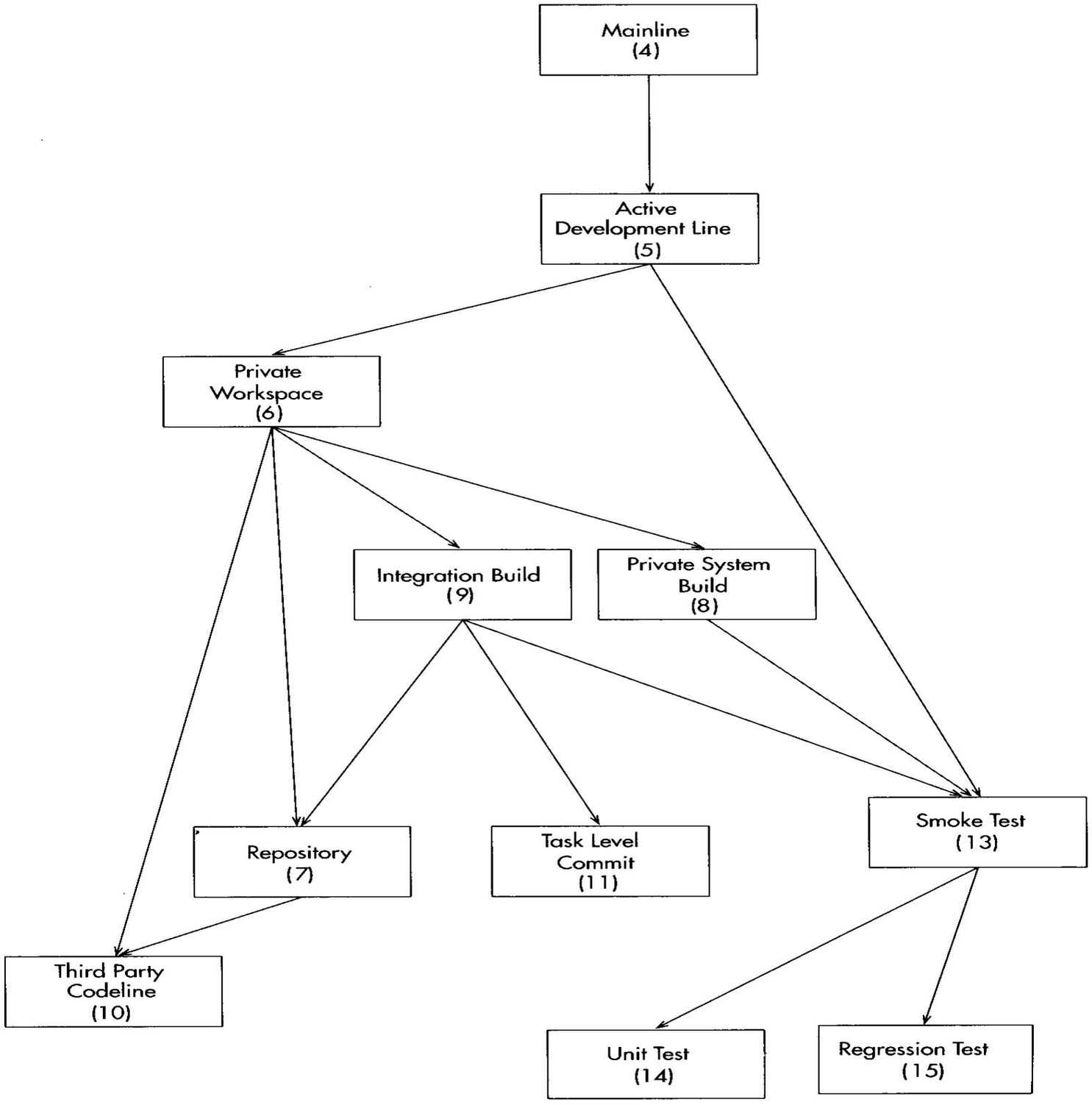
Software Configuration

stability and progress in a code line

Software Configuration Management Patterns

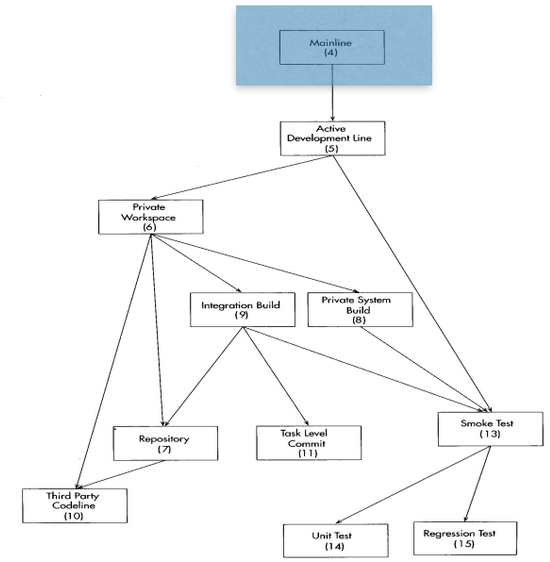
Berczuk and Appleton
http://www.scmpatterns.com/

Build Patterns



Workspace-related patterns

http://www.scmpatterns.com/book/SCMPatterns-RefCard.pdf



Workspace-related patterns

master, in the project

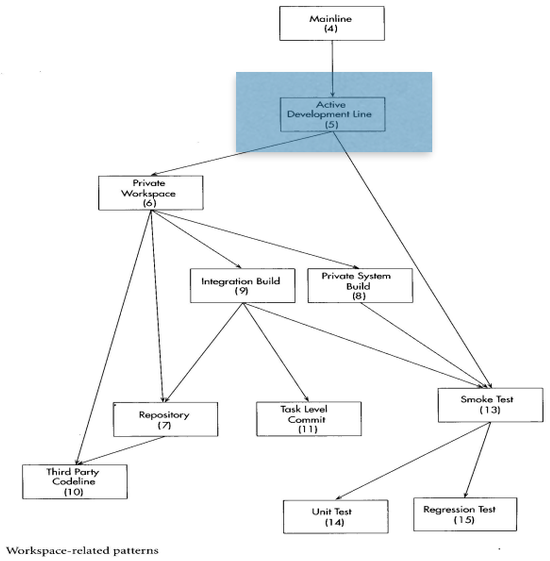
Mainline

the place of collaboration

How effective can the mainline be?

it can be disruptive

many people are using the mainline



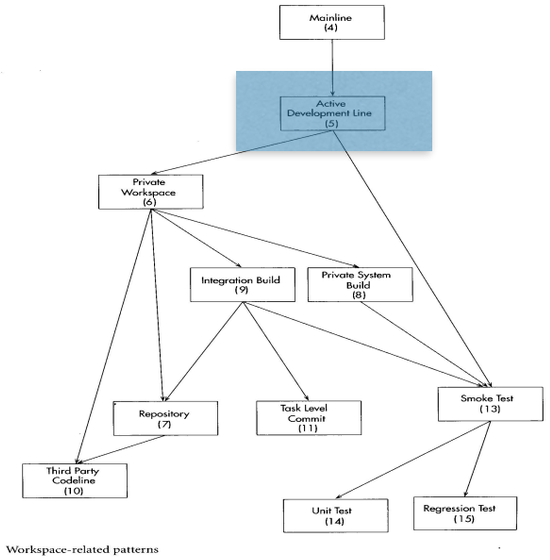
Workspace-related patterns

Active Developm ne

pushes in git

check the quality
the commits with tes

verify the quality of the mainline with tests

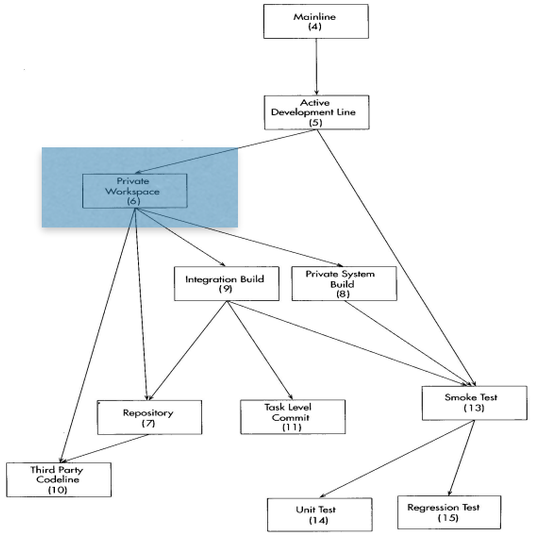


Workspace-related patterns

Developers want to work with the latest code

latest code is in the mainline

and in isolation



Workspace-related patterns

Private Workspace

isolation and collaboration

1. Update the source tree from the mai

fetch; merge

collaboration

2. Make changes

isolation

3. Do a private system build

isolation

4. Test with a unit test

isolation

fetch; merge

commit

collaboration

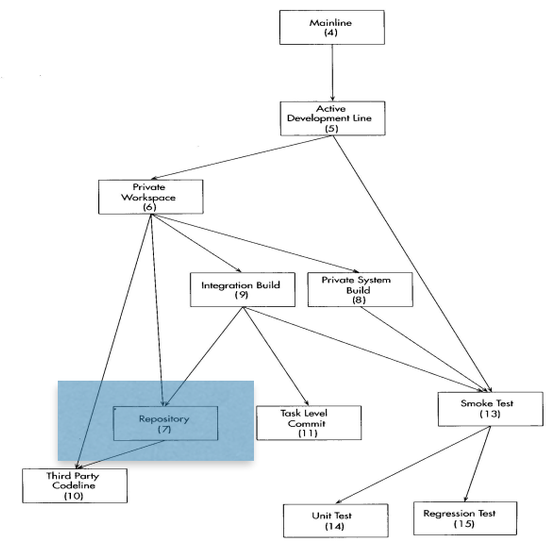
5. Update the workspace t late version of all components by getting the latest versions of all components that were not changed

6. Rebuild and run a smoke test

isolation

Where do we feed the workspace from

get the right versions of the components



Workspace-related patterns

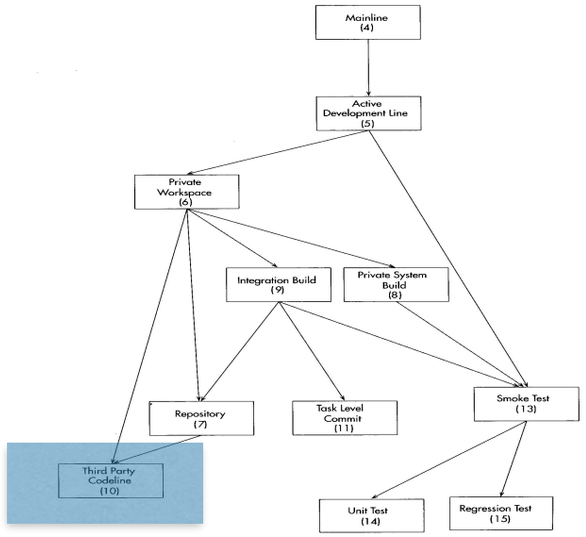
local repository in git

Repository

local or global?
(we'll learn about private versioning - later)

How can we integrate with third party versions

which library versions are being used
versions compatibility…



Workspace-related patterns

where is it in maven?

Third Party Codeline

use third party codelines and
the repository to build the workspaces

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${version.junit.junit}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.jmockit</groupId>
    <artifactId>jmockit</artifactId>
    <version>${version.org.jmockit.jmockit}</version>
</dependency>
```

Main

JMockit is a Java toolkit for automated developer testing. It contains APIs for the creation of the objects to be tested, for mocking dependencies, and for faking external APIs; JUnit (4 & 5) and TestNG test runners are supported. It also contains an advanced code coverage tool.

License	MIT
Categories	Mocking
Tags	mocking testing
Used By	393 artifacts

Central (39)

	Version	Repository	Usages	Date
1.38.x	1.38	Central	11	(Dec, 2017)
1.37.x	1.37	Central	4	(Nov, 2017)
1.36.x	1.36.3	Central	1	(Nov, 2017)
	1.36.2	Central	0	(Nov, 2017)
	1.36.1	Central	1	(Nov, 2017)
	1.36	Central	2	(Oct, 2017)
1.35.x	1.35	Central	12	(Sep, 2017)
1.34.x	1.34	Central	9	(Aug, 2017)
1.33.x	1.33	Central	36	(Jun, 2017)
1.32.x	1.32	Central	11	(May, 2017)
1.31.x	1.31	Central	9	(Mar, 2017)
1.30.x	1.30	Central	26	(Dec, 2016)
1.29.x	1.29	Central	9	(Oct, 2016)
1.28.x	1.28	Central	5	(Sep, 2016)

Main » 1.38

JMockit is a Java toolkit for automated developer testing. It contains APIs for the creation of the objects to be tested, for mocking dependencies, and for faking external APIs; JUnit (4 & 5) and TestNG test runners are supported. It also contains an advanced code coverage tool.

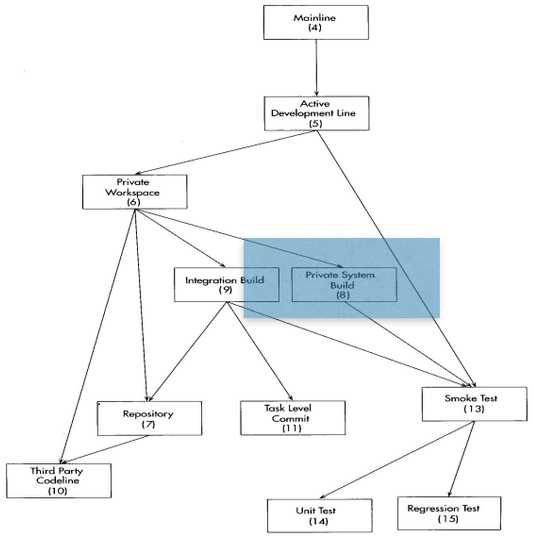
License	MIT
Categories	Mocking
HomePage	http://www.jmockit.org
Date	(Dec 31, 2017)
Files	pom (10 KB) jar (790 KB) View All
Repositories	Central Sonatype Releases
Used By	**393 artifacts**

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/org.jmockit/jmockit -->
<dependency>
    <groupId>org.jmockit</groupId>
    <artifactId>jmockit</artifactId>
    <version>1.38</version>
    <scope>test</scope>
</dependency>
```

How do we verify the level of quality of private development

do not commit (push) changes that break



Workspace-related patterns

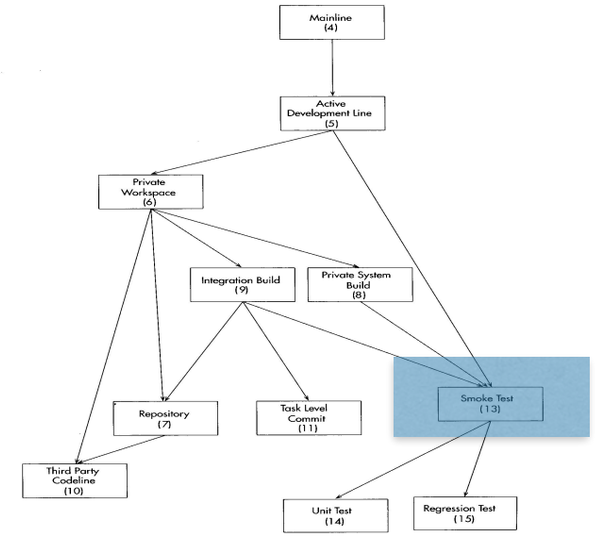
Private System Build

include the changes

How can we quickly verify the quality of the b[illegible]nd

long-running t[illegible]rage larger grained changes

decreases quality and collaboration



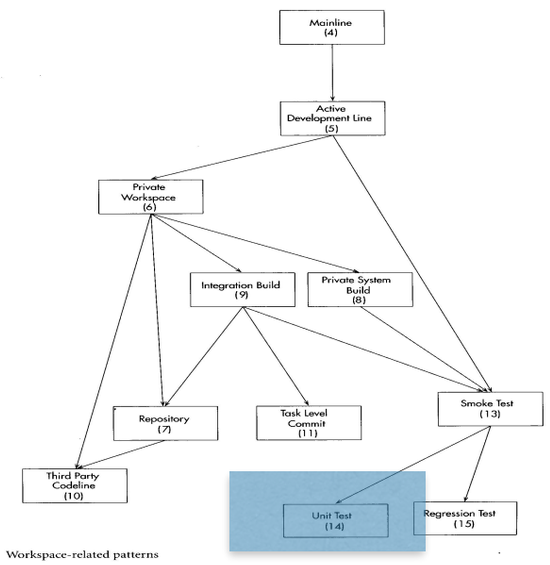
Workspace-related patterns

the smaller set of tests
that may catch more bugs

Smoke Test

the system has not broken in a obvious way

How can we ensure that our module does not break



Workspace-related patterns

Unit Test

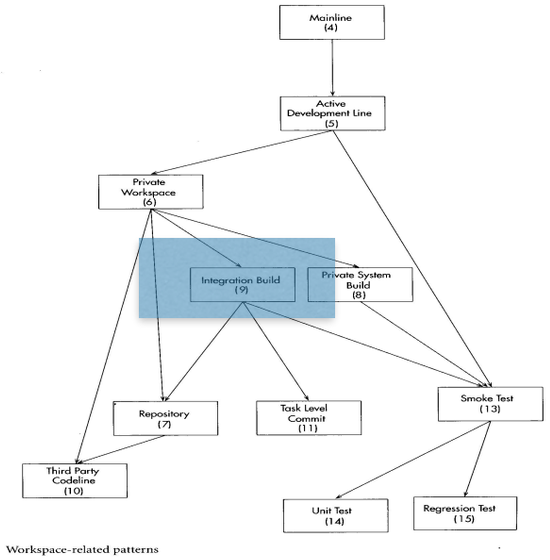
test-first

besides smoke tests more
detailed tests for the changed unit

collaboration

How can we integrate the work from different workspaces

it may be necessary to do a more thoroughly testing



Workspace-related patterns

every hour build

twice a day build

nightly build

Integration Build

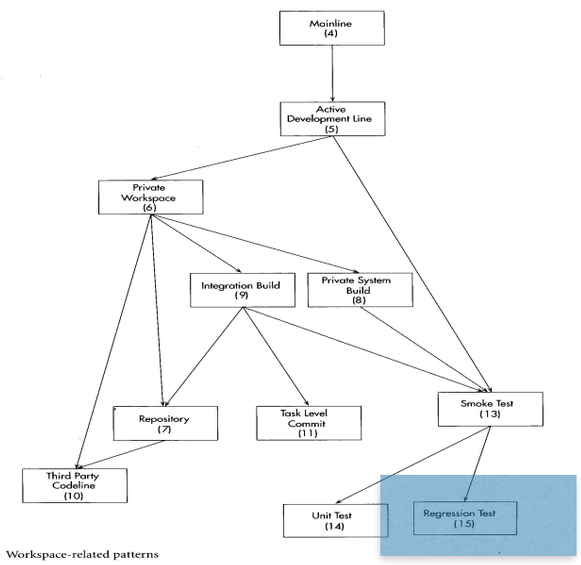
How long it takes to build the system

How quickly changes are happening

how often?

periodically

How do we ensure that existing code does not get worse



Workspace-related patterns

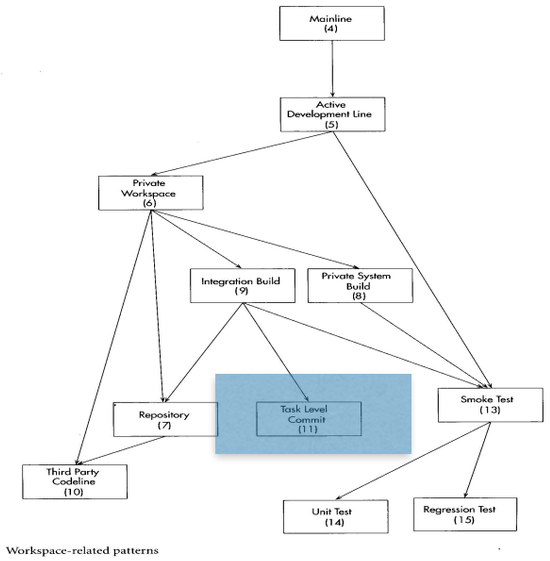
Regression Test

whenever we want to ensure stability of the mainline

Do not get lost during an integration commit

conflicts during a pull

debug other commits (pushes)



Workspace-related patterns

one commit per small gra

consistent task

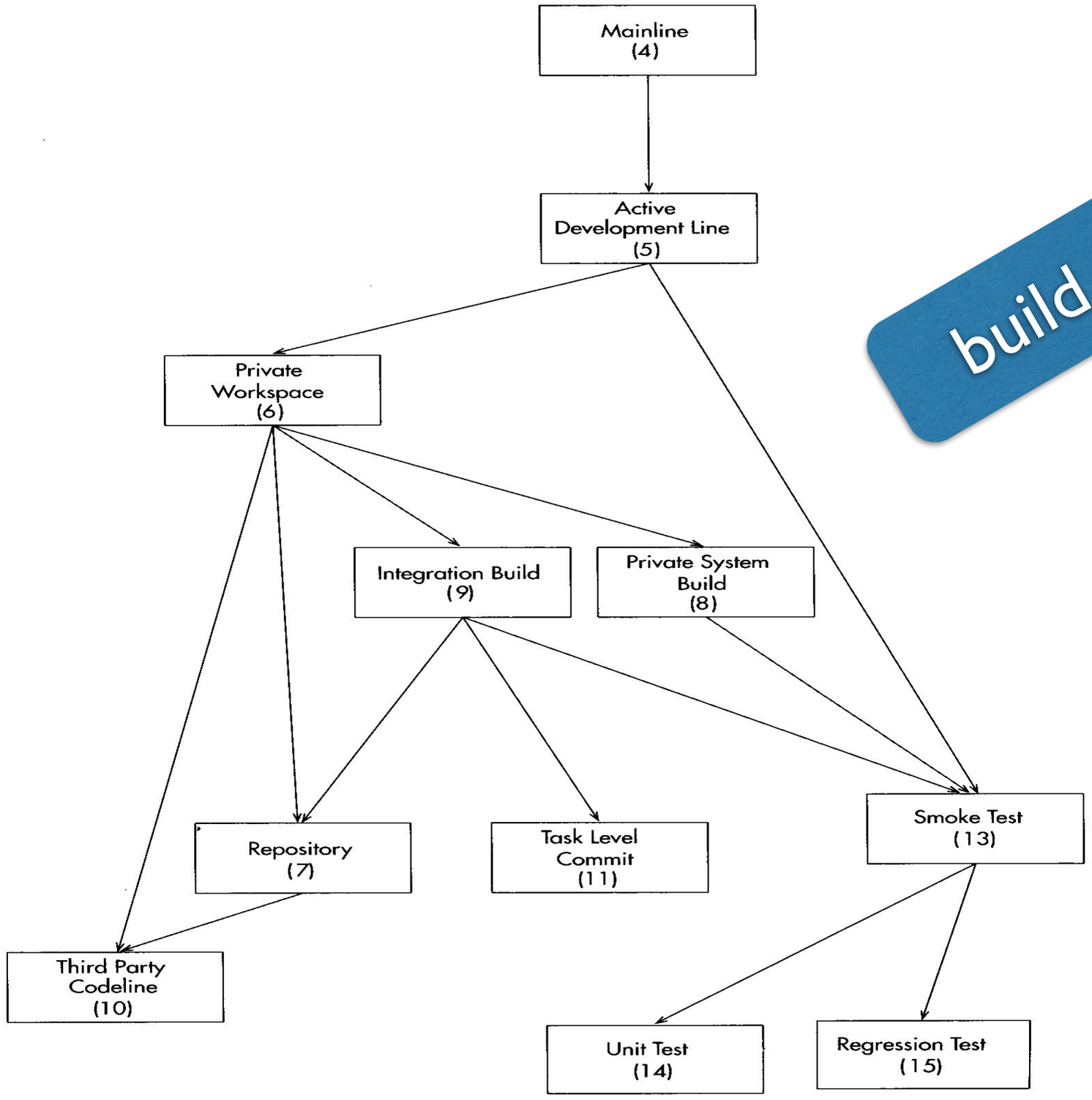
GitHub issues

that can

shared

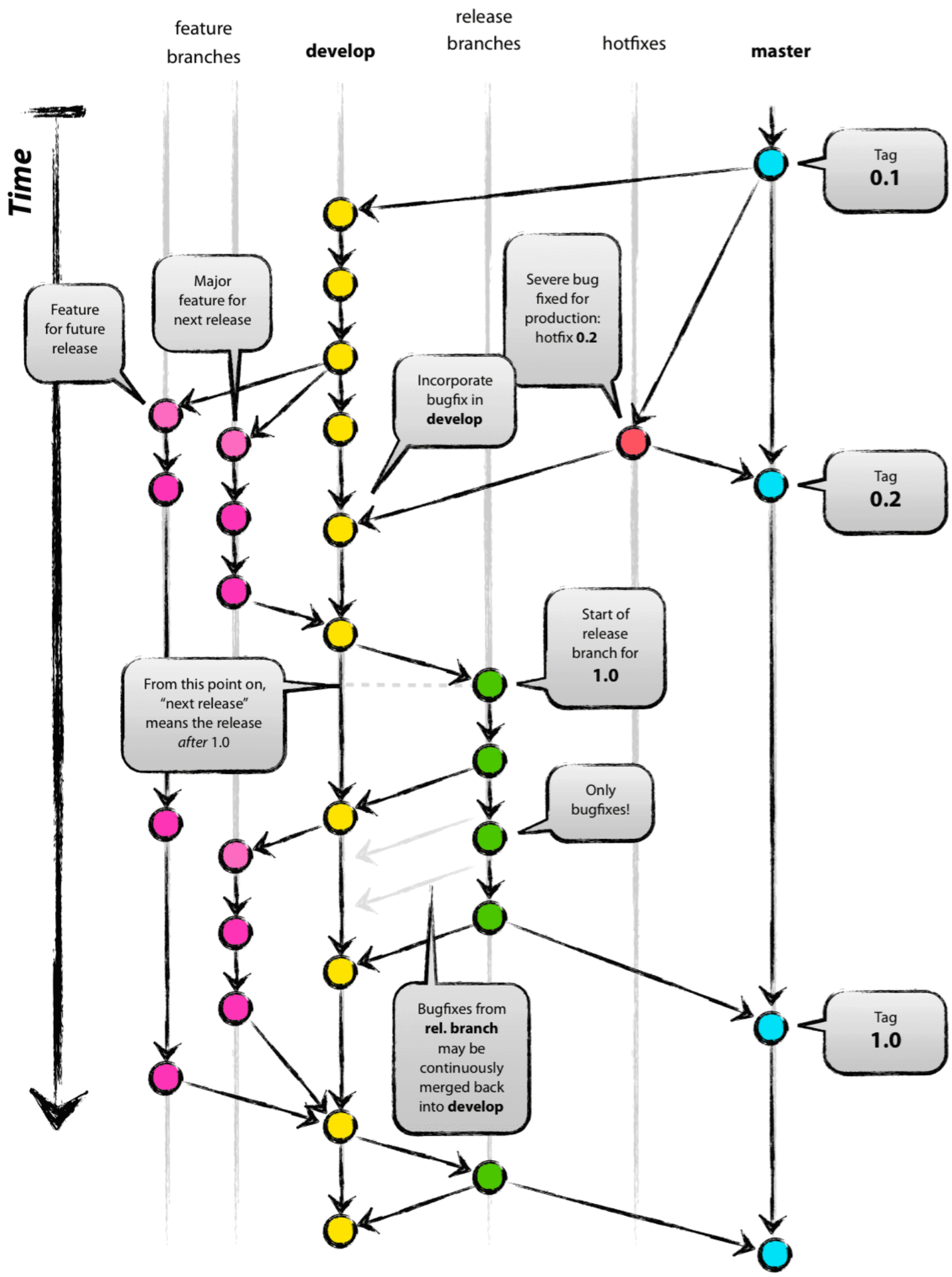
Task Level Commit

consistent rollbacks when something breaks

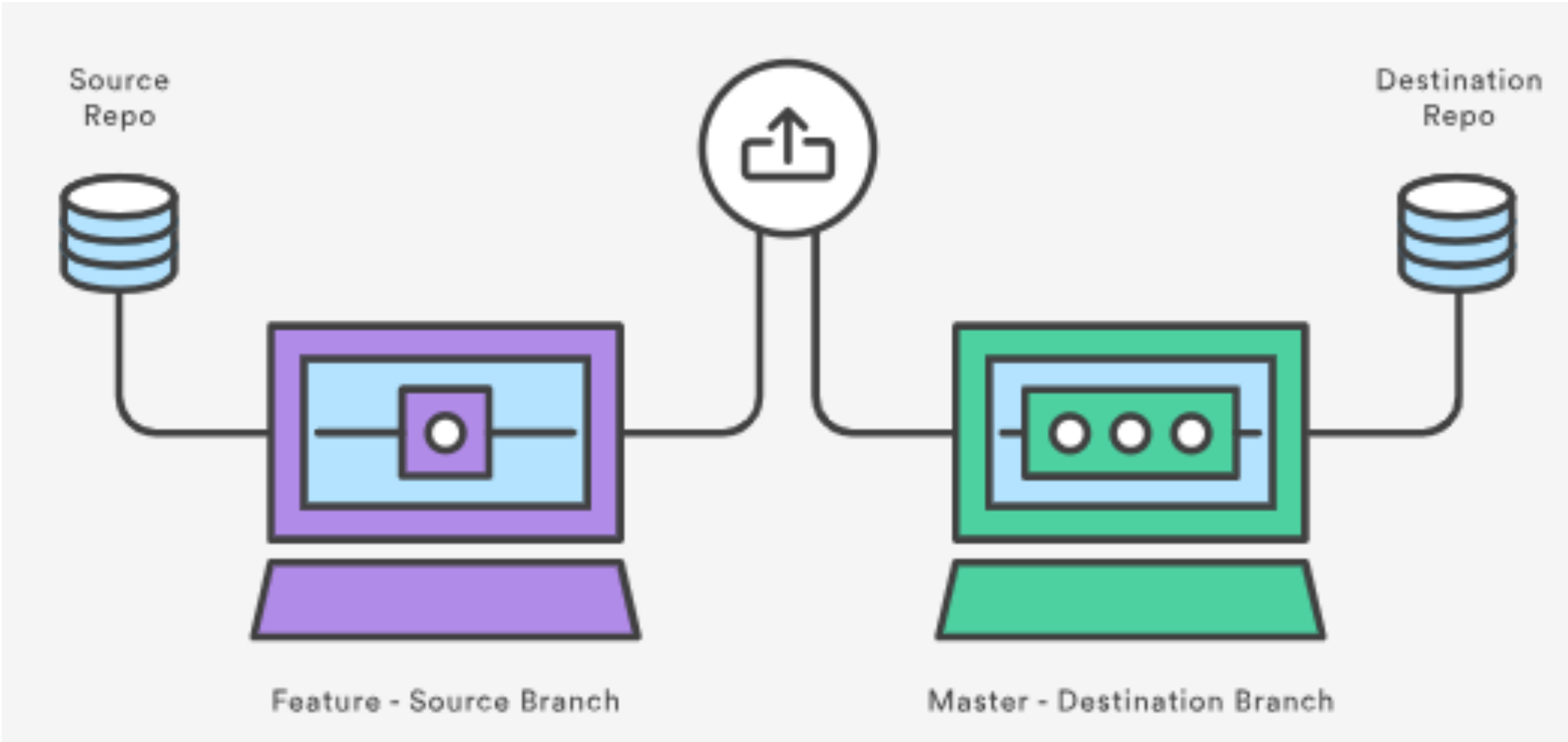


build patterns

Workspace-related patterns


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Pull-request



https://www.atlassian.com/git/tutorials/making-a-pull-request
https://help.github.com/en/articles/creating-a-pull-request