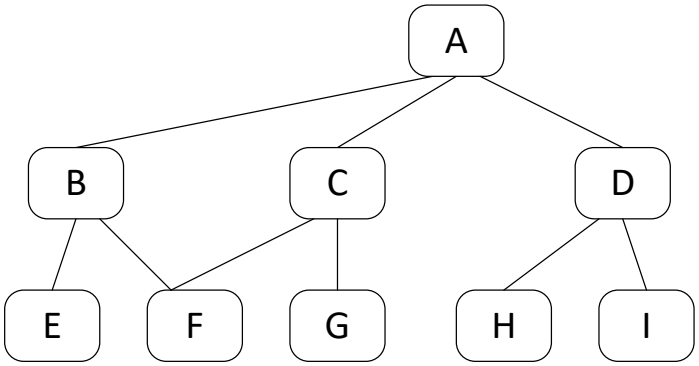
Integration Tests Planning

I4SWT

AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

Getting ready – mapping the dependency tree

- Integration test planning is helped along using a *dependency tree*
  - Depicts inter-module dependencies in a tree-like structure
  - *Does not* depict an inheritance hierarchy, layering or the like



A *depends-on* B, C and D
B *depends-on* E and F
C *depends-on* F and G
D *depends-on* H and I

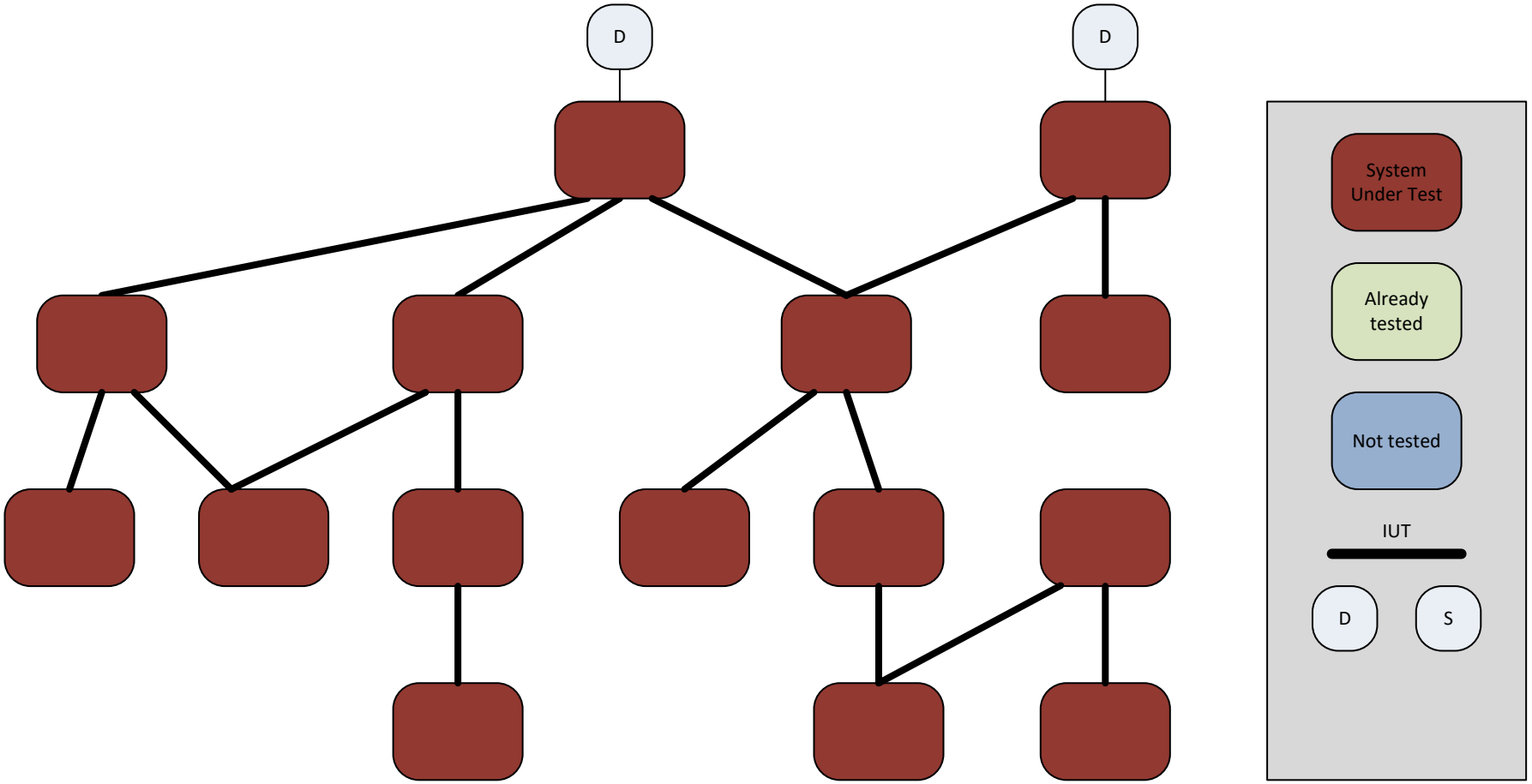
- Some dependencies are obvious from sequence diagrams, object diagrams, state charts, etc.
- Others require inspection (members, parameter types, ..)
- Loops must be broken using stubs

Integration test patterns

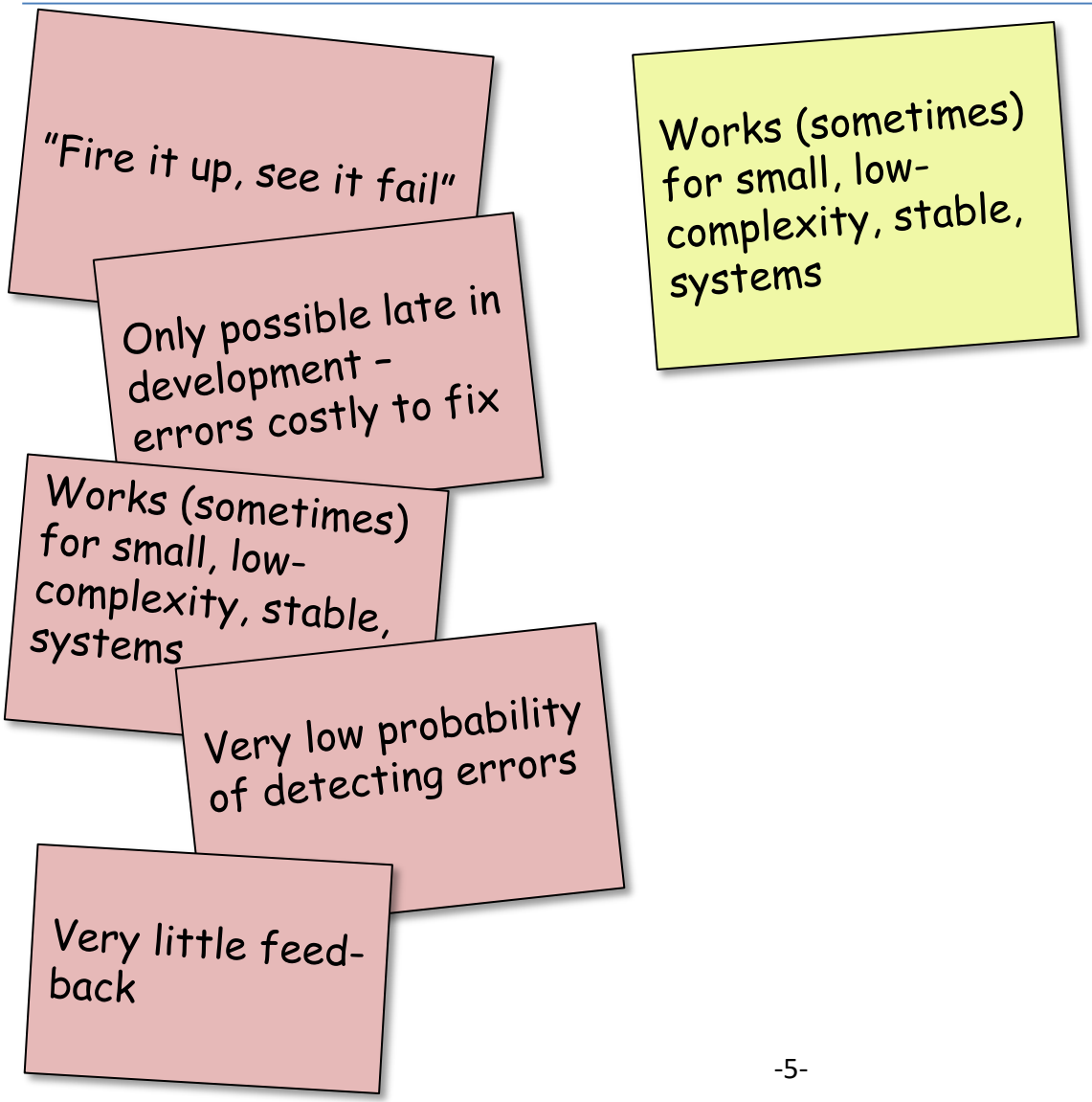
- *Integration test patterns* are used to plan and execute the integration tests.

- This session covers the following patterns
  - Big Bang Integration
  - Bottom-up Integration
  - Top-down Integration
  - Collaboration Integration
  - Sandwich Integration

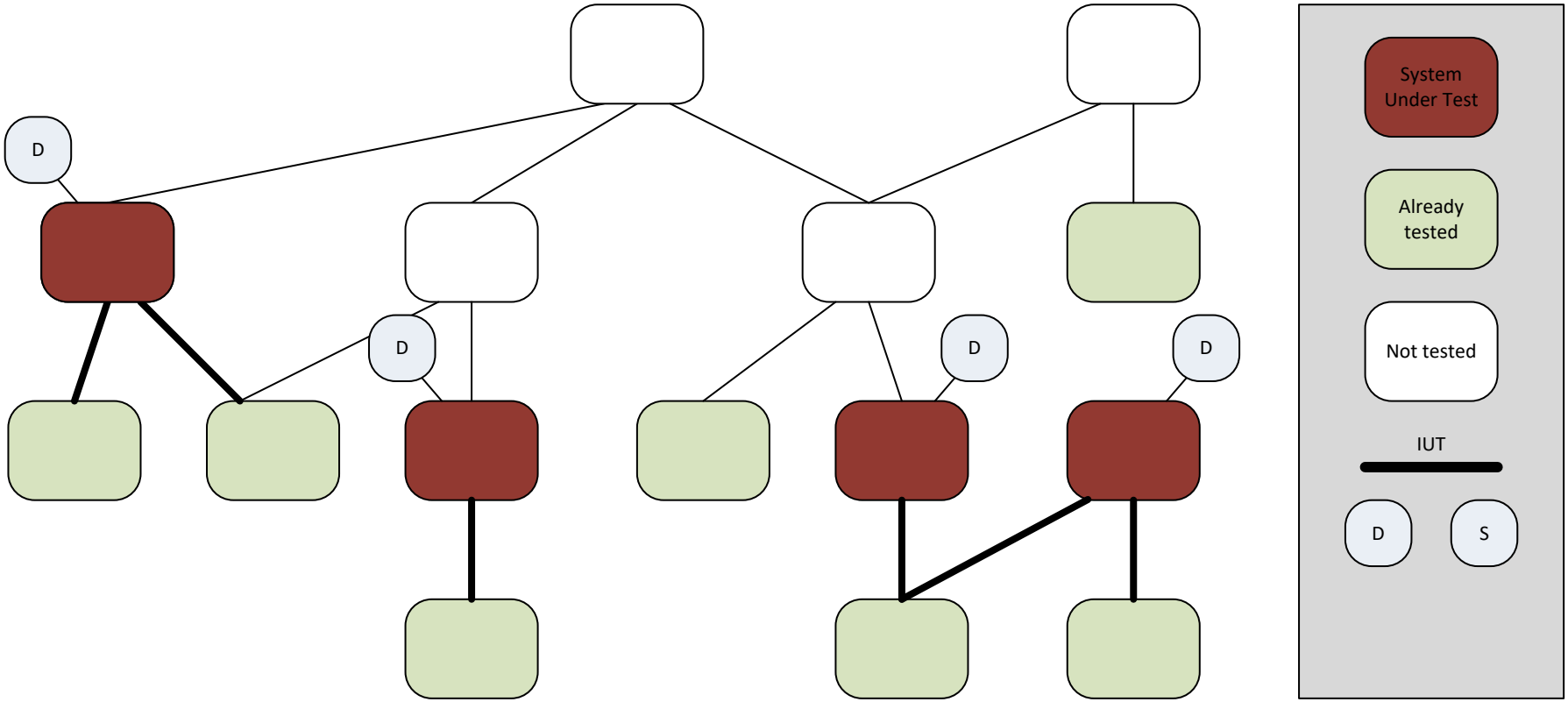
Big Bang Integration



Big Bang Integration



Bottom-up Integration



Bottom-up Integration

Requires many drivers at different levels

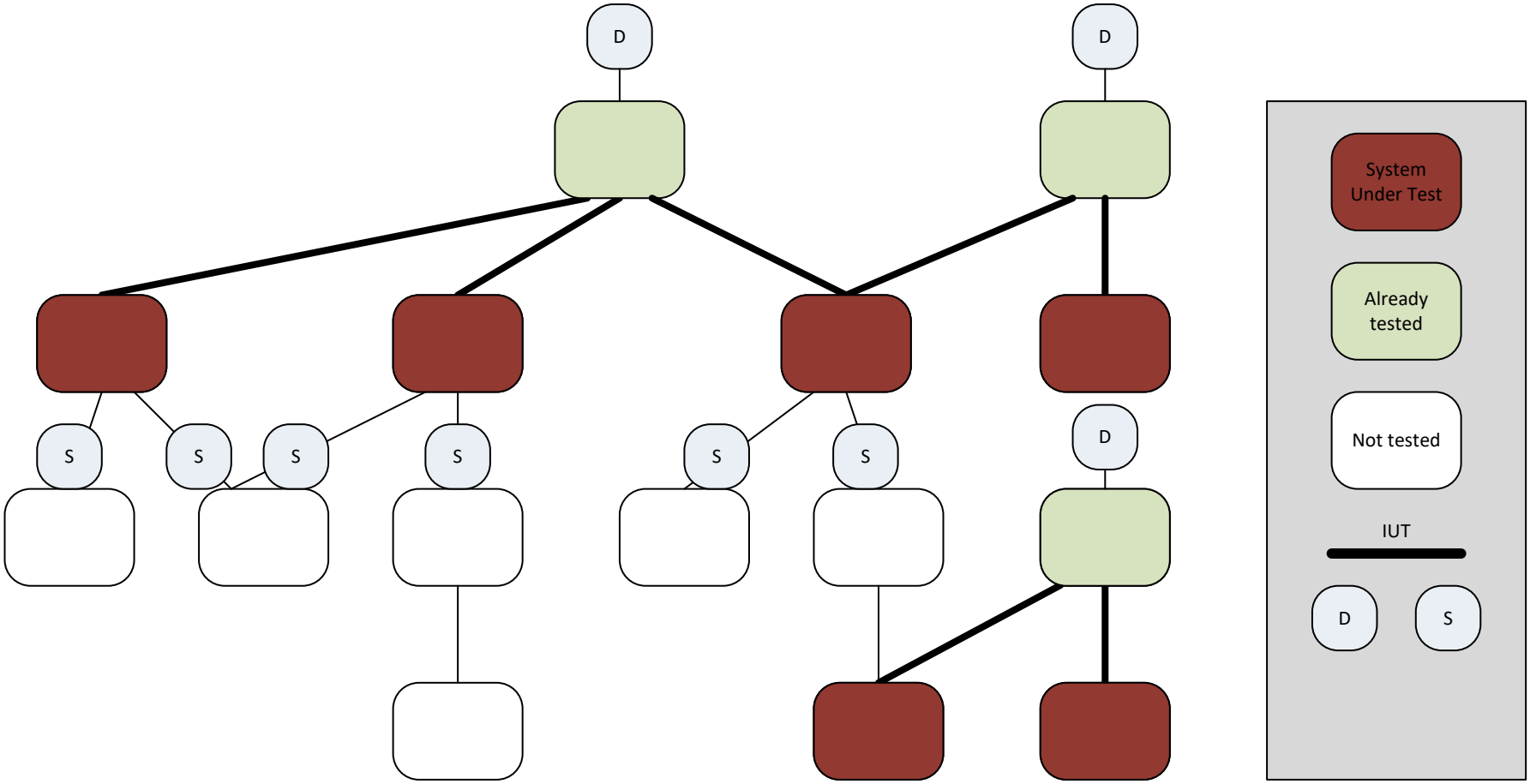
Postpones test of critical control com-ponent interfaces

Reflects very "engineering-like" mindset

No (few) stubs to develop

Easy to cover interfaces at all levels

Top-down Integration



Top-down Integration

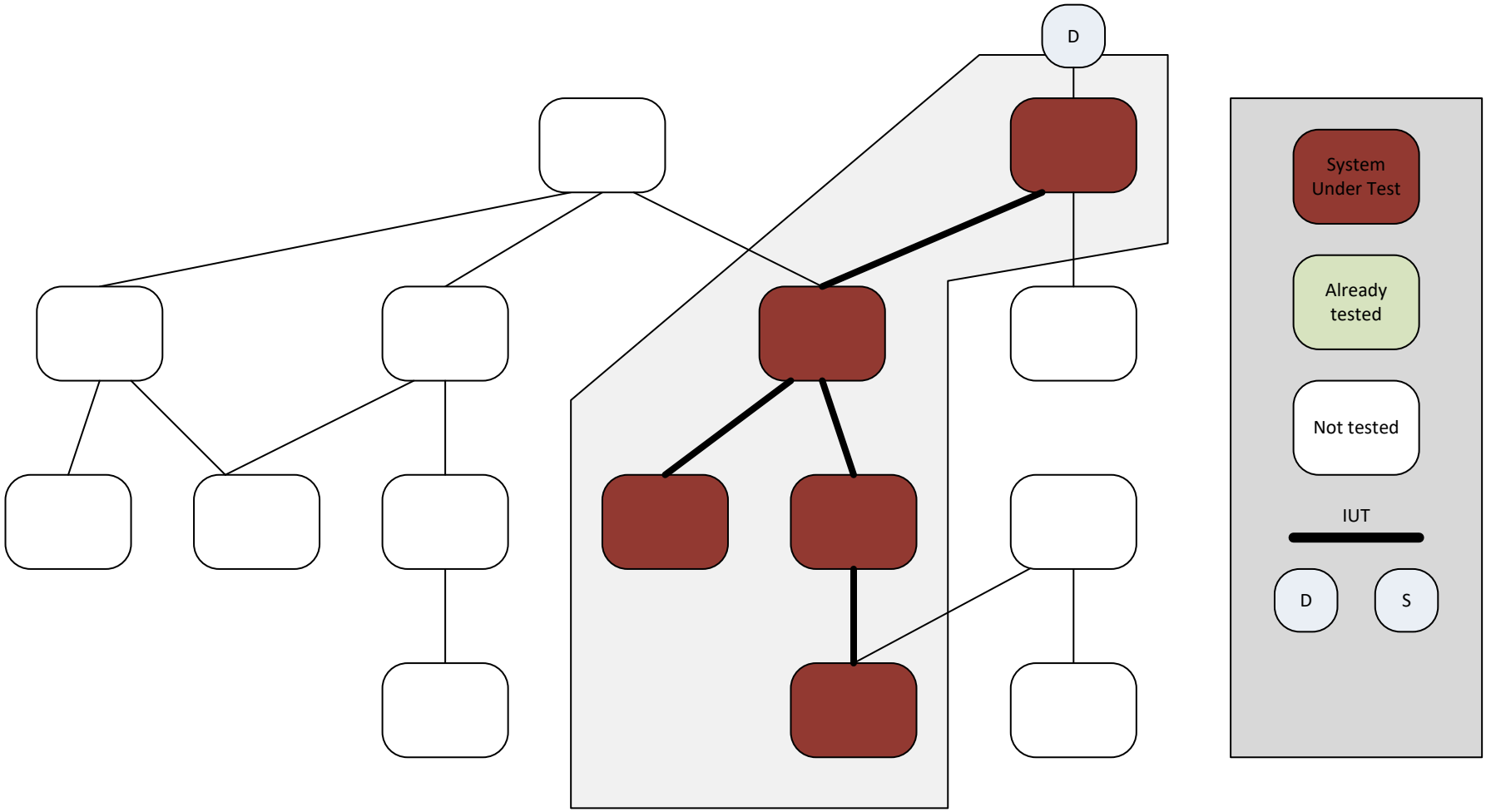
Hard to exercise low-level interfaces from the top

Needs lots of stubs (OK with isolation framework)

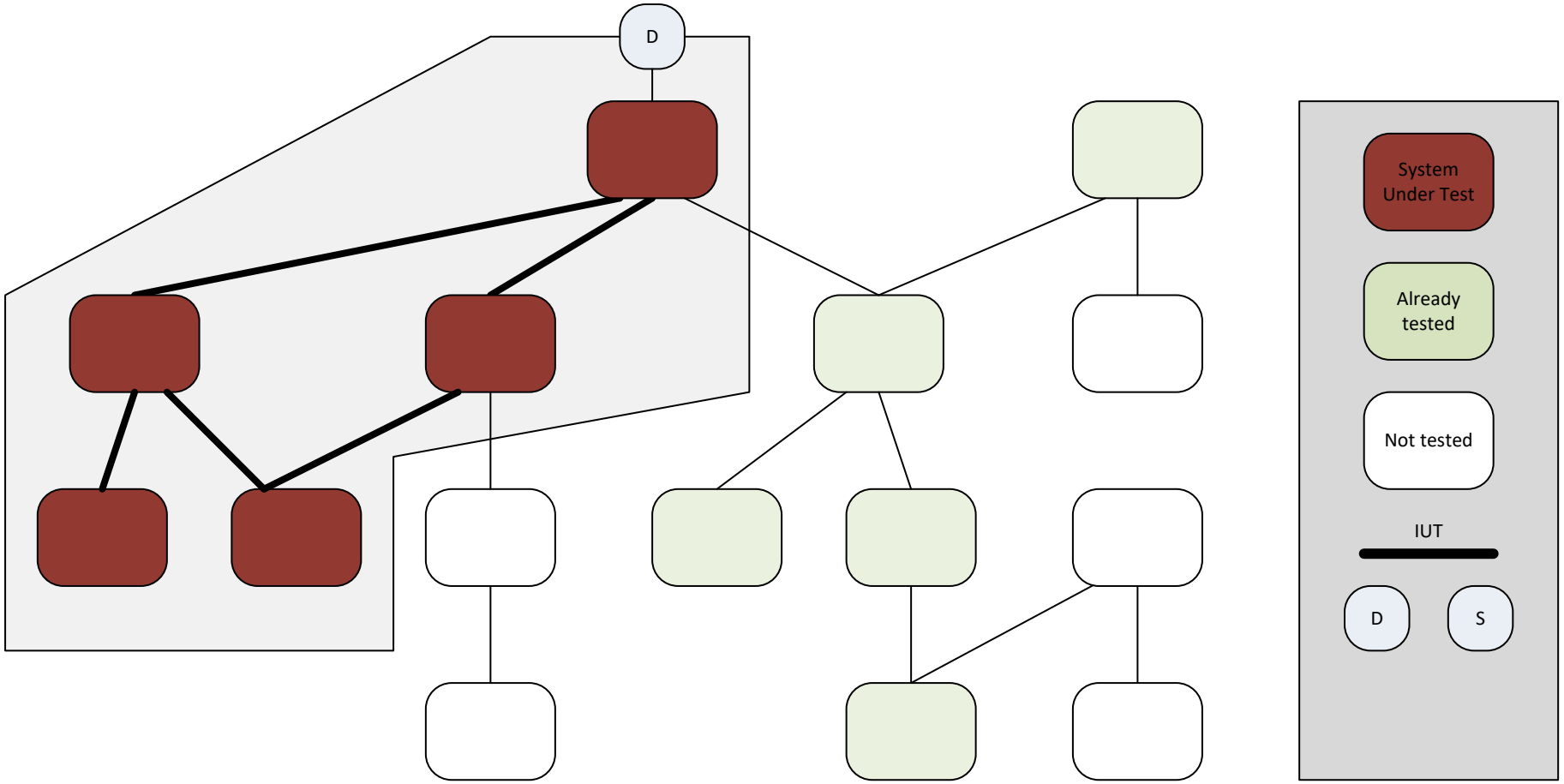
Early feedback on controller compo-nents

Facilitates concurrent HW and SW development

Collaboration Integration



Collaboration Integration



Collaboration Integration

Hard to exercise low-level interfaces

Participants not exercised separa-tely

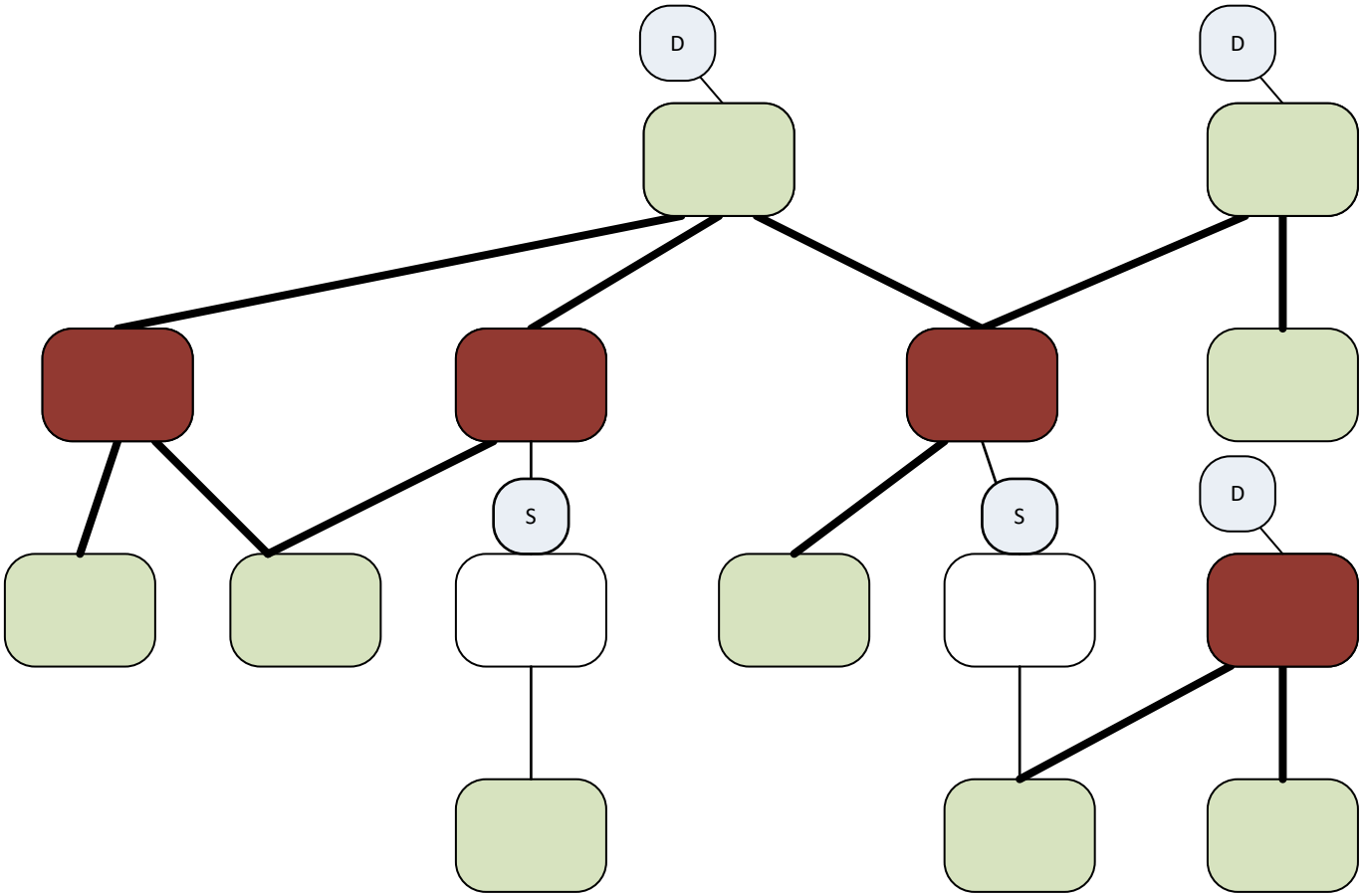
Needs lots of stubs (OK with isolation framework)

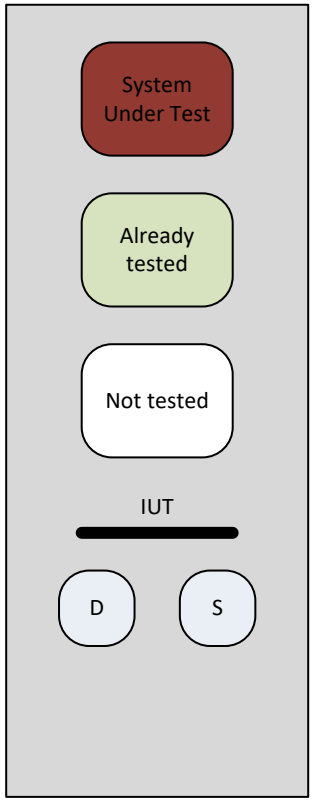
Intuitive for users (may follow use cases)

Especially useful for higher-level system tests (component, subsystem)

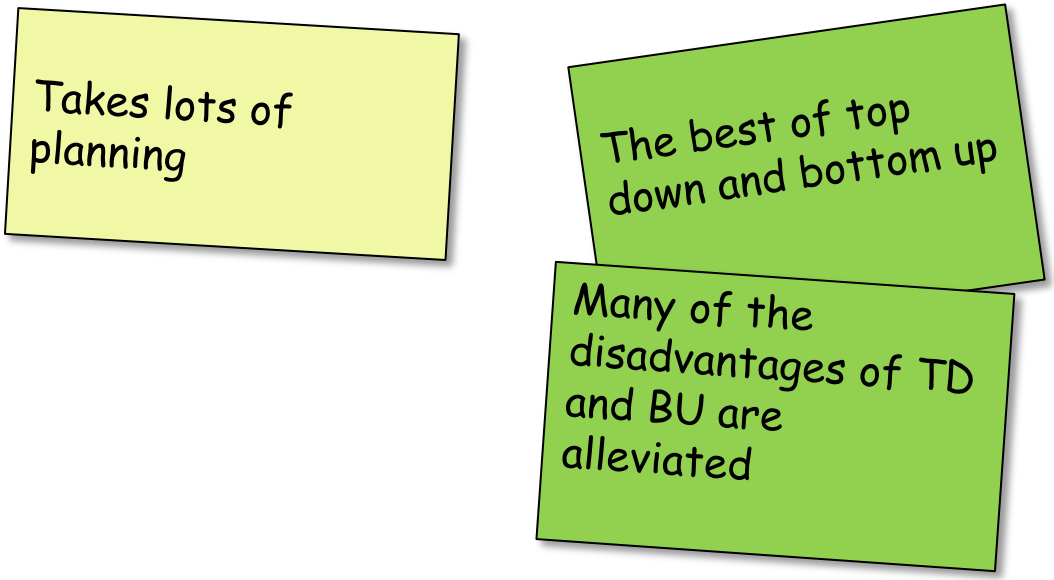
Models iterative development with UCs as unit

Sandwich Integration





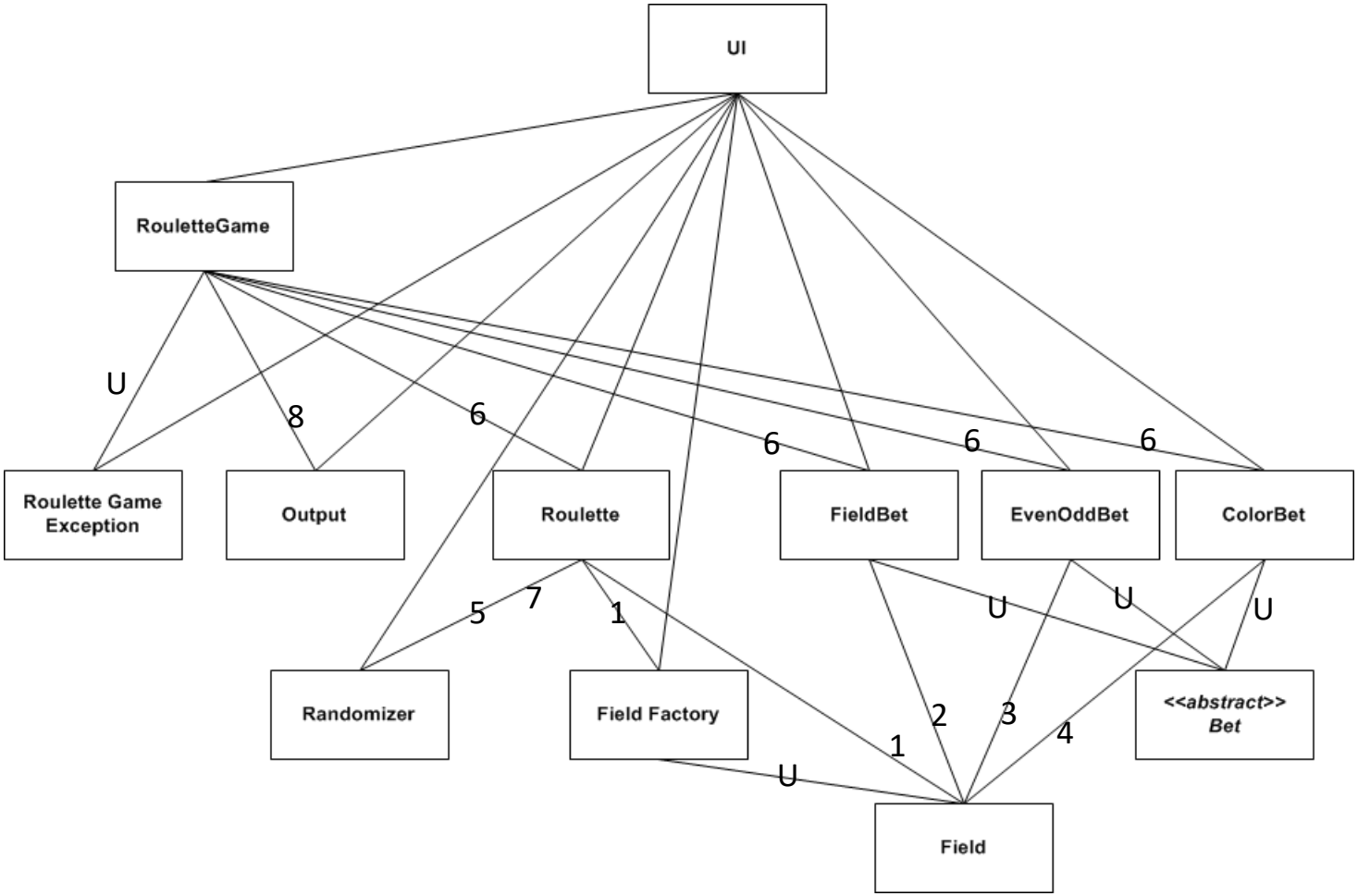
Sandwich Integration



Dependency tree

- Class diagrams (static information)
  - Remember polymorphic classes
- Sequence diagrams (behavioral information)
- Test cases for units
  - Which interfaces were defined
  - Which IFs were faked
- Call trees? Manual or tool?
  - Reveals "tool classes", objects passed around
- Dependency tree generator tool?

Dependency Tree



Bottom Up Plan

Step #	Roulette Game	Output	Roulette	FieldBet	EvenOdd Bet	ColorBet	Rando-mizer	Field Factory	Field
1			T				S	X	X
2				T					X
3					T				X
4						T			X
5			T				X	X	X
6	T	S	X	X	X	X	S	X	X
7	T	S	X	X	X	X	X	X	X
8*	T	X	X	X	X	X	X	X	X

T: This module is included, it's the/a top module, and the one driven
X: This module is included
S: This module is faked: stubbed or mocked
*Step # 8 is difficult to automate!

Test cases

- There is no interface coverage tool!
- Partial use cases
  - Look in sequence diagrams
- Original Unit Test cases for the top level unit(s)
  - (top level in the current integration step)
  - Does it make sense to reuse them all?

How to organize ITs

- Organize your IT steps in one or more separate projects under the same solution as the units – one test fixture for each step