Agenda

- Definition of a module
- Benefits of modularity
- Pitfalls
- Analyzing packages with JDepend
- Adventure
- Summary

Definition of a module

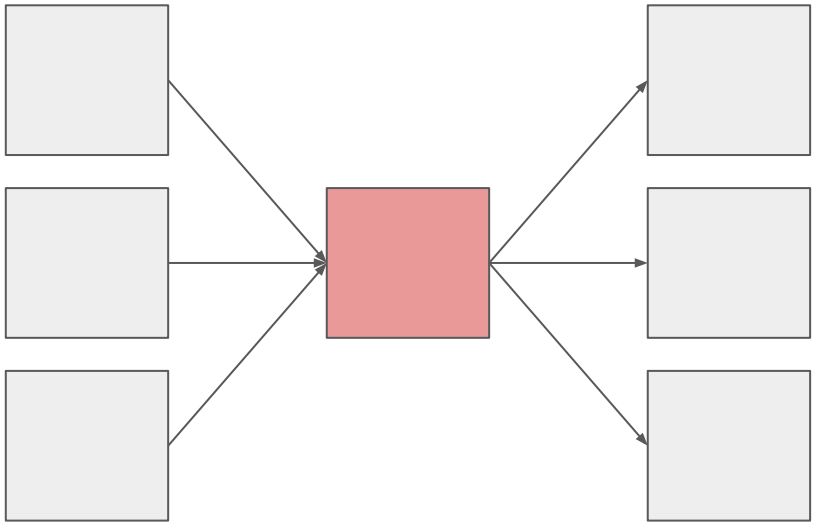
- Set of related functionality
- Can depend on other modules
- In java we use classes, packages and jars

Benefits of modularity

- Reduce complexity - make it easier to understand
  - Conveys intent
  - Enforces boundaries
- Changes are restricted
- Testing is easier
- Reuse

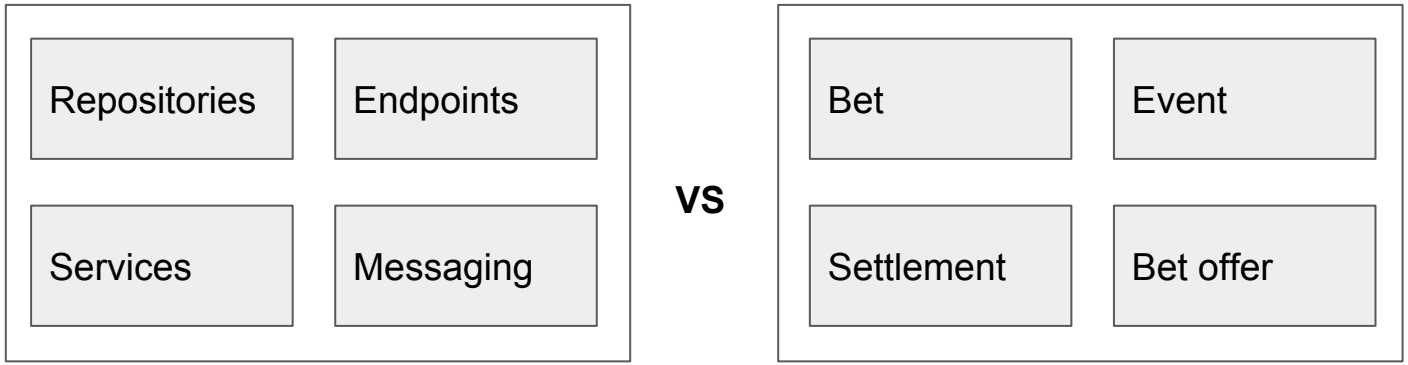
Pitfalls

- Coupling to other modules
  - Using other modules
  - Being used by other modules



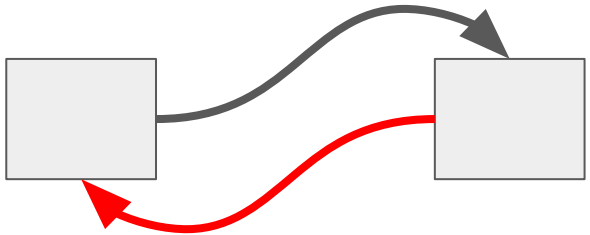
Pitfalls cont.

- Not being cohesive - functionality is not related
- Different types of cohesion
  - Coincidental
  - Logical
  - Functional
  - … and a few others



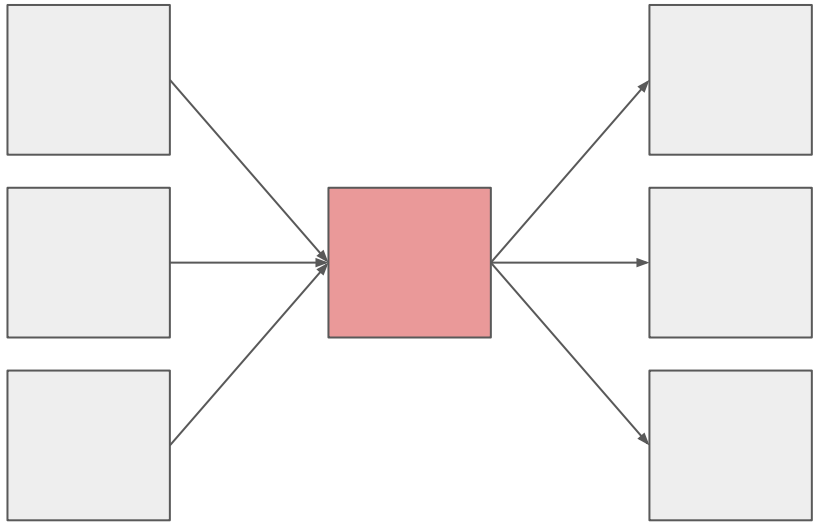
Pitfalls cont.

- Being neither abstract nor concrete
- Circular dependencies



Analyzing packages with JDepend

- How many depend on this package?
- How many packages does it depends on?
- Abstract or concrete?
- How unstable is it?
- Is it a trouble maker?



Example

- Analysis of two packages
  - letter.func
  - letter.console

letter.func

Stats:

Concrete Classes: **2**
Abstract Classes: **4**

# of packages using it: **3**
# of packages used: **0**

Abstractness: **67%**
Instability: **0%**
Trouble maker: **33%**

Depends Upon:

Not dependent on any packages.

Used By:

letter.console
letter.service
letter.writer

letter.console

Stats:

Concrete Classes: **7**
Abstract Classes: **2**

# of packages using it: **1**
# of packages used: **4**

Abstractness: **22%**
Instability: **80%**
Trouble maker: **2%**

Depends Upon:

letter.core
letter.func
letter.service
letter.writer

Used By:

letter.writer

Adventure

Summary

- Advantages of modularity and pitfalls
- Modularity metrics
- Using stan4j