Архитектурын зарчмууд

Програм хангамжийн архитектур (Software Architecture)

2012

С. Бадрал

Агуулга

- Нэр томьёо
- Класс ба архитектурын төвшний загварууд
  - SOLID
  - Average Component Dependency
- Архитектурын төвшний загварууд
  - Common Closure Principle
  - Common Reuse Principle
  - Acyclic Dependencies Principle
  - Average Component Dependency
  - Stable Dependencies Principle
- Дүгнэлт

Нэр томьёо

- Principle - Зарчим
- Substitution - Орлуулалт
- Segregation – Тусгаарлалт
- Dependency – Хараат байдал
- Inversion – урвуулах
- Stable – тогтвортой
- Instable - тогтворгүй

OX загварын зарчмууд

S - Single Responsibility Principle

O - Open Closed Principle

L - Liskov Substitution Principle

I - Interface Segregation Principle

D - Dependency Inversion Principle

Single Responsibility Principle - SRP

> “There should never be more than one reason for a class to change.”

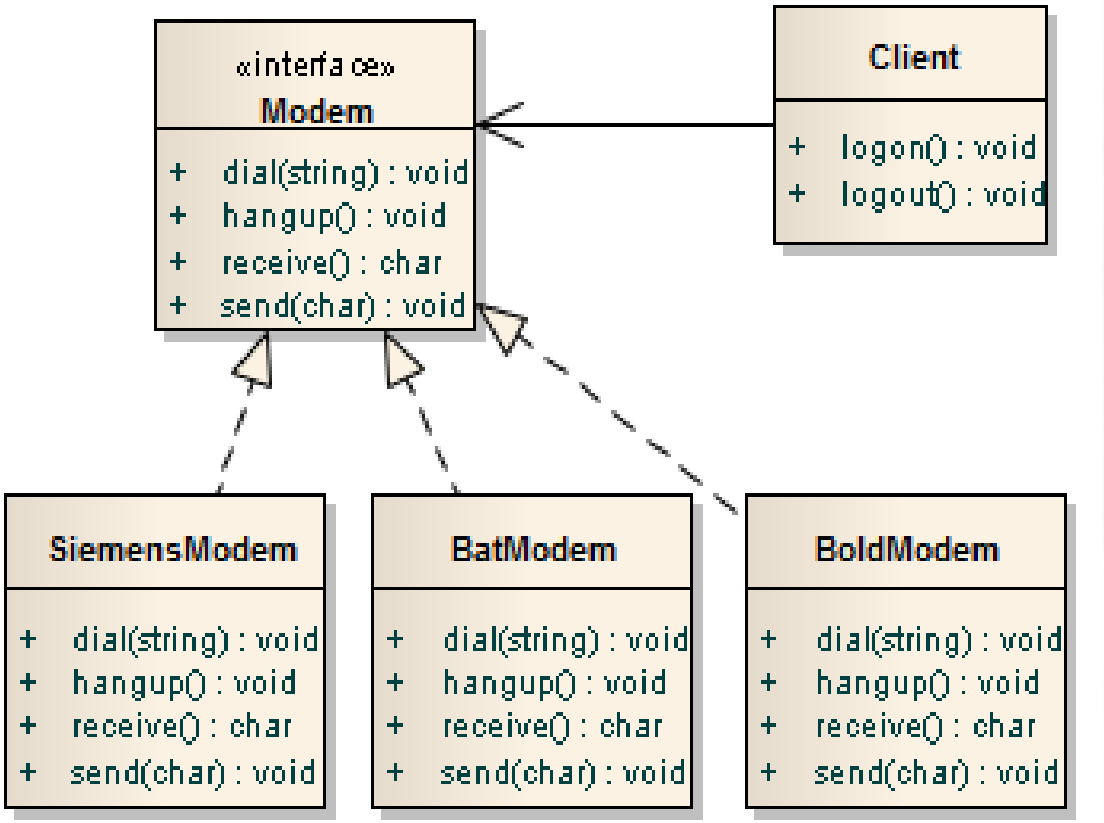
- Хариуцлага = Өөрчлөх шалтгаан

```
Interface Modem {
        Public void dial(String pno);
        Public void hangup();
        Public void send();
        Public void receive();
}
```

- Модем 2 хариуцлагыг хэрэгжүүлж байна.
- Хоёр хариуцлага өөр өөр үндэслэлээр өөрчлөгдөж болно.
- Хоёр хариуцлага өөр өөр нэгжээр дуудагдаж болно.

Open Closed Principle - OCP

A module should be open for extension but closed for modification.



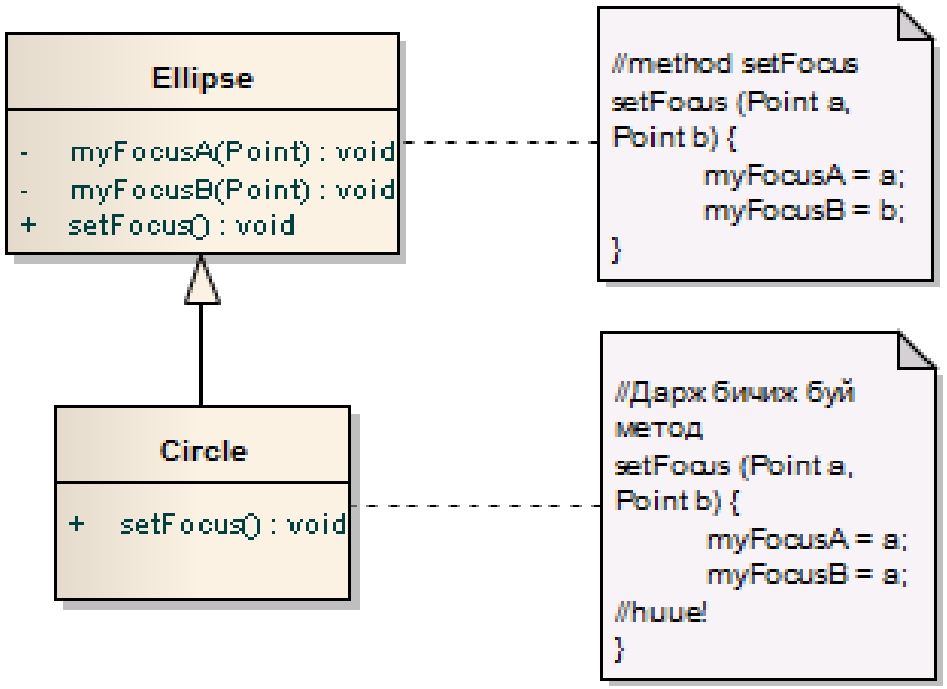
Open Closed Principle - OCP

- Элементүүд өргөтгөхөд нээлттэй харин өөрчлөхөд хаалттай байх хэрэгтэй
- Элементүүд “нийлүүлэгдсэнээрээ” хэрэглэгдэх хэдий ч өргөтгөх боломжтой байх хэрэгтэй
- Полиморфоор хэрэгждэг
- Ирээдүйн өргөтгөл нь оршин буй эх кодыг өөрчлөгдөхгүйгээр хийгдэх боломжтой байх хэрэгтэй
- Зөвхөн хийсвэрлэлийн хараат байдал зөвшөөрөгдөж, конкрет хэрэгжүүлэлтийн хараат байдал зөвшөөрөгдөхгүй

Liskov Substitution Principle - LSP

"Subtypes must be substitutable for Their base types."

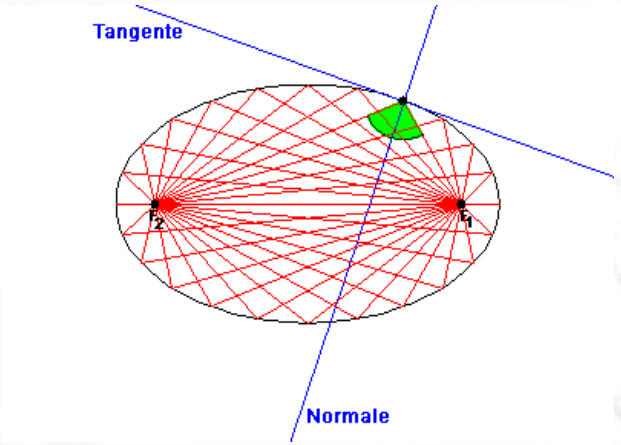
- Дэд төрөл өөрийн дээд төрлийн бүх гэрээг биелүүлнэ.
- Дарагдсан методууд нь урьдач нөхцлөөс илүү хүчтэй эсвэл дараах нөхцлөөс сул нөхцөл авахгүй.



Тойрог ба Эллипсийн эрсдэлтэй модел

Liskov Substitution Principle - LSP

```
Public void myMethod (Ellipse e) {
        Point p1 = new Point (10, 10);
        Point p2 = new Point (20, 20);
        e.setFocus (p1, p2);
        assert (e.getFocusA() == p1);
        assert (e.getFocusB() == p2);
}
CirIce c = new Circle();
this.myMethod(c);
```



- Жава-д InstanceOf-г их дуудах нь дээд класс эсвэл зурвасыг хэрэглэхэд алдаа гарсан гэдгийг заана.

Interface Segregation Principle - ISP

> Many client specific interfaces are better than one general purpose interface.

- Хэрэв А Класс олон клиенттэй бол клиентийн методуудыг зурваст задлах хэрэгтэй. Клиентэд хамаатай зурвасууд нь А класст хэрэгжүүлэгдэнэ.

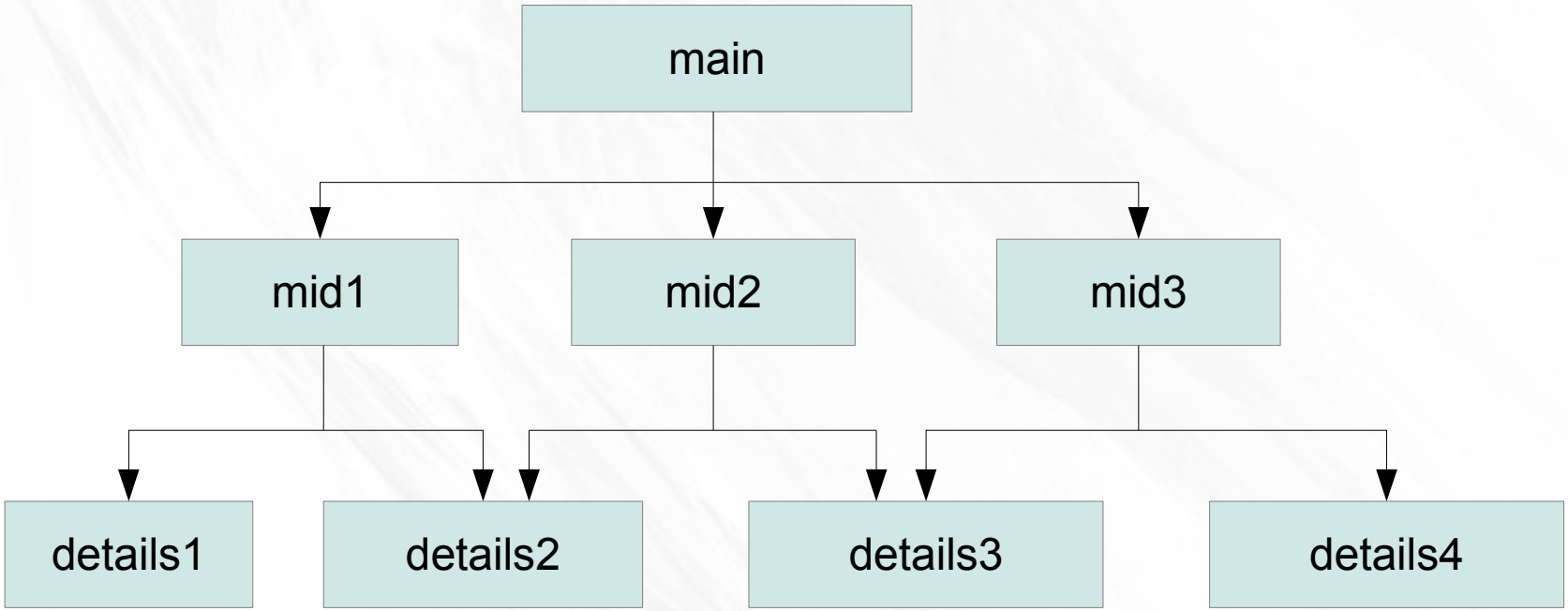
- Клиентэд хамаатай гэдэг нь нэг тодорхой зурвас хэрэглэж буй төстэй клиентүүдийг ангилна гэсэн үг.

- Хэрэв нэг эсвэл хоёр Клиент төрөл нэг өөрсдийн методоо хэрэглэж байвал тухайн методуудыг хоёр зурваст хоёуланд нь нэмнэ.

Dependency Inversion Principle - DIP

Depend upon abstraction.
Do not depend upon concretions.

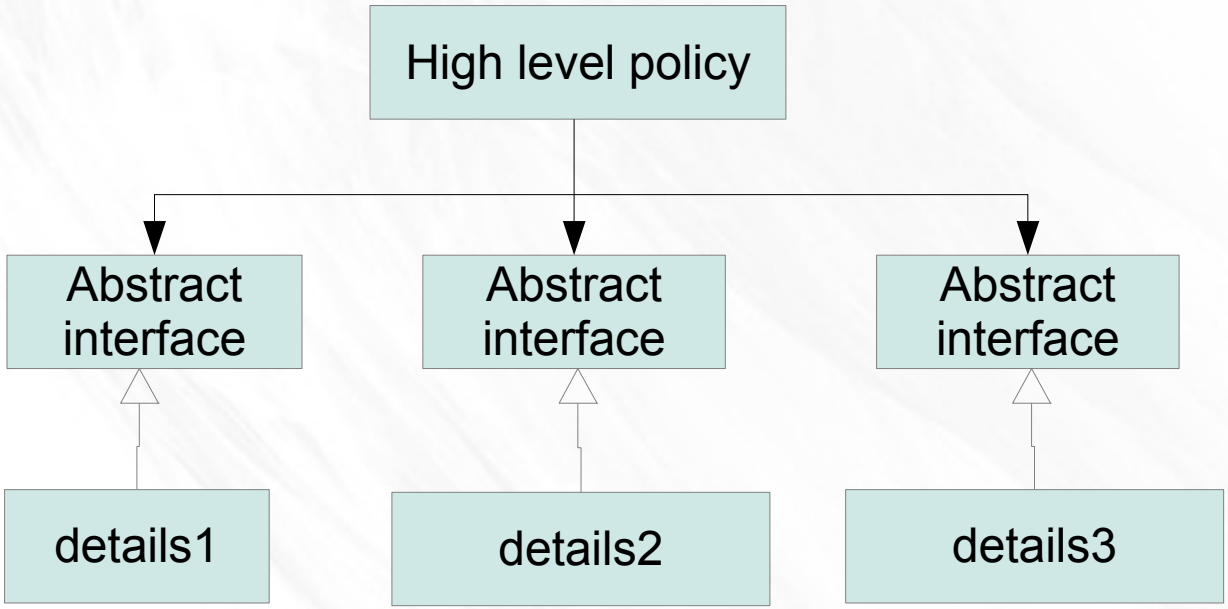
- Процедурал зохиомжийн хараат байдлын бүтэц



- Энд дээд төвшний компонентүүд нь нарийвчилсан хэрэгжүүлэлтээсээ хамаарна.

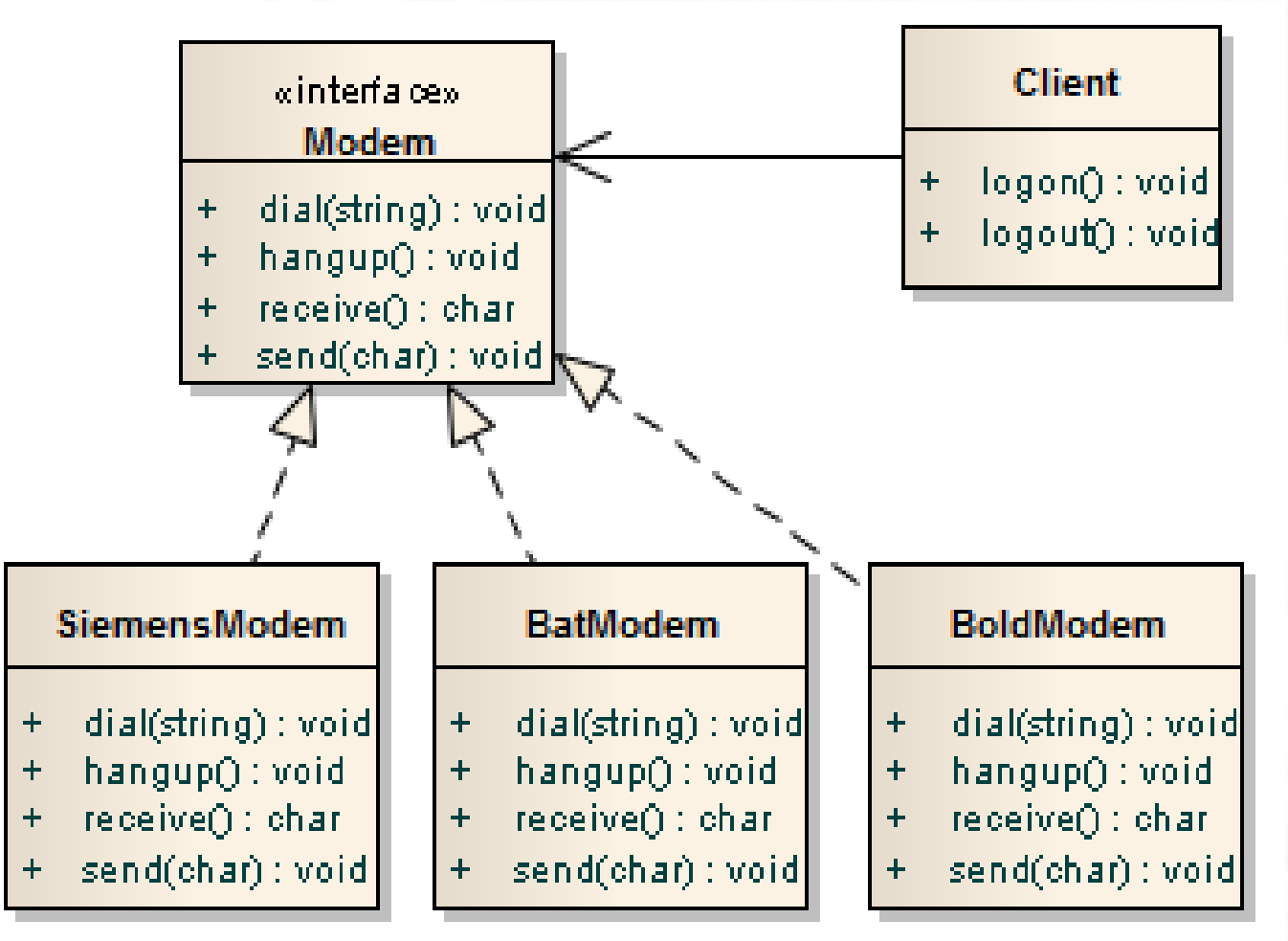
Dependency Inversion Principle - DIP

- ОХ зохиомжид хамаарлууд эргэсэн байдаг. Учир нь нарийвчилсан хэрэгжүүлэлтүүд хийсвэр зурвасаас хамаардаг. (Удамшил)



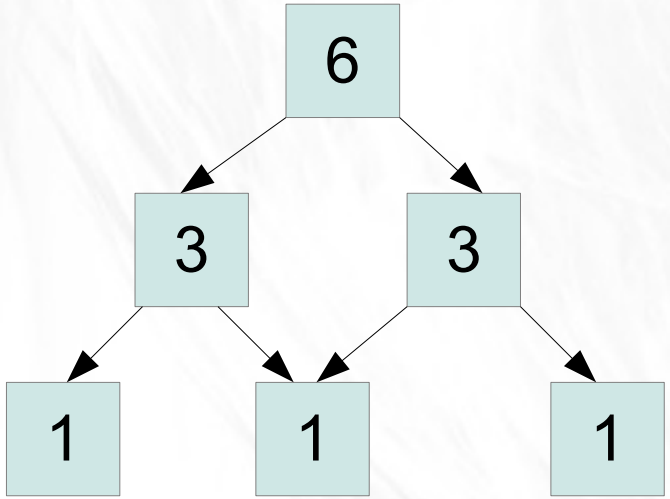
- Энд дээд төвшний элементүүд нь зөвхөн зурвасуудаас хамаарна.

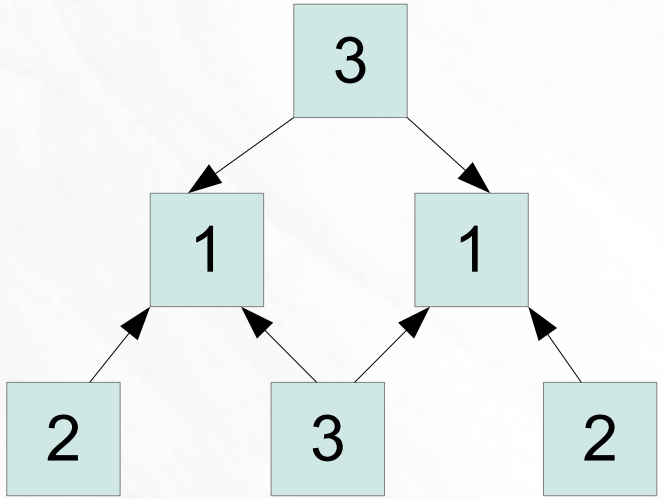
DIP Жишээ: Модем



Average Component Dependency - ACD

- ACD = ∑ Dep (Component) / #Components





ACD = 15/6 = 2,5

ACD = 12/6 = 2,0

- Энд дээд төвшний элементүүд нь зөвхөн зурвасуудаас хамаарна.

Common Closure Principle - CCP

Classes that change together,
belong together

- Томоохон төслүүдэд классууд нь дэд пакетуудад, пакетууд нь дэд системүүдэд хуваагддаг.
- Нэг Release-д хэдийчинээ олон пакет/дэд систем өөрчлөгдөнө, нийт системийн дахин build хийх, шалгалт хийх, суурилуулт хийх зардал төдийчинээ өндөр байна.
- Тиймээс өөрчлөгдсөн пакет ба дэд системийн тоог хязгаарлах нь тустай байдаг.
- Үүнд хүрэхийн тулд ихэнхдээ нэгэн зэрэг өөрчлөгддөг классуудыг хамтад нь нэг дэд системд ангилах болдог.

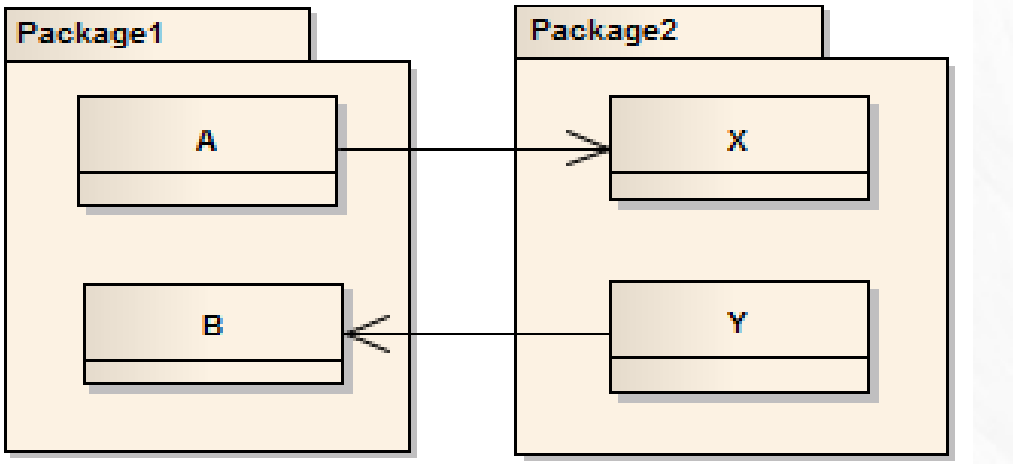
Common Reuse Principle - CRP

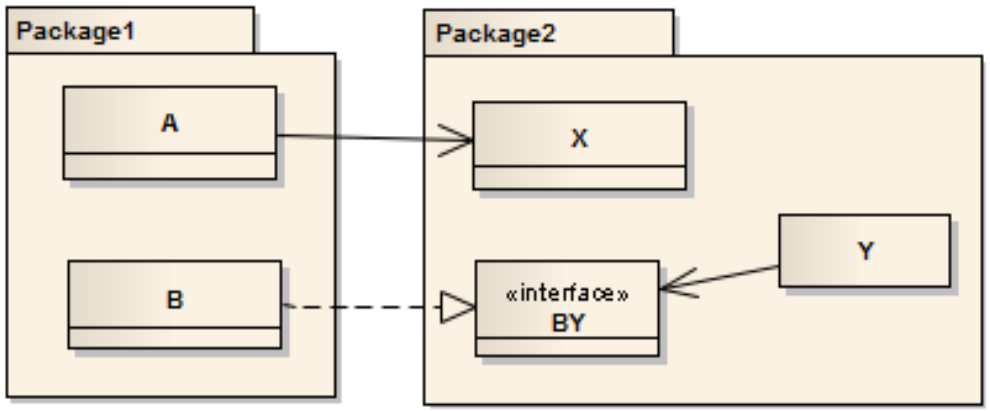
Classes that are not reused together, should not be grouped together

- Пакет эсвэл дэд системээс хараат байх нь энэ нэгжийн бүх элементээс хараат байна гэсэн үг.
- Хэрэв клиент нэгжийн зөвхөн цөөн хэдэн классаас хараат бөгөөд тэдгээрт нь өөрчлөлт ороогүй байсан ч дэд системүүдийг дахин шалгах болон суулгах (шинэ Release) зардал гардаг.

Acyclic Dependencies Principle - ADP

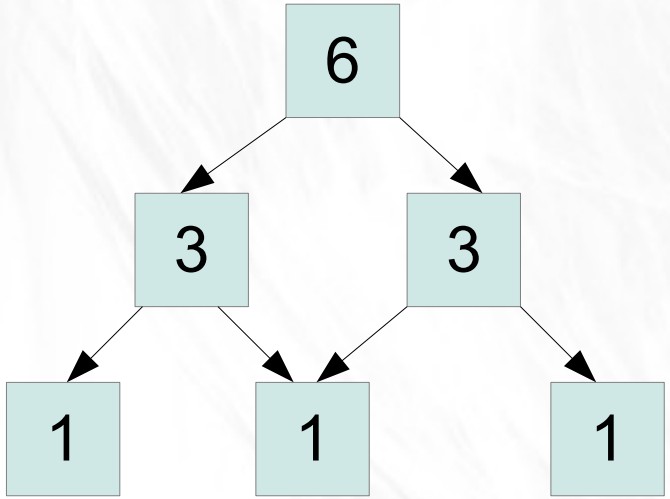
The dependency between packages must not form cycles



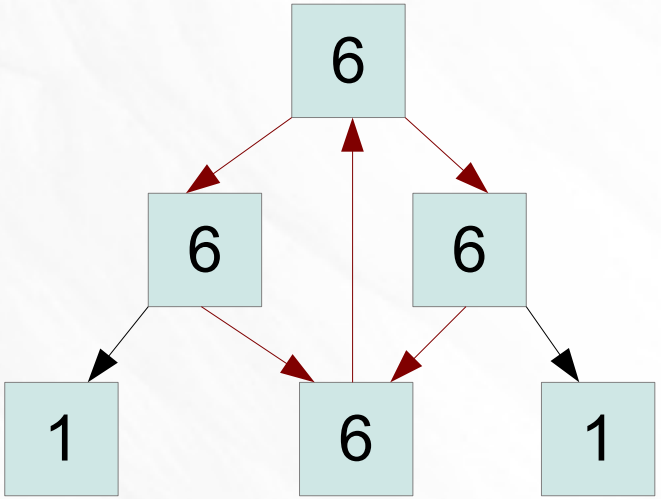


Average Component Dependency – ACD with Cycle

- ACD = ∑ Dep (Component) / #Components



ACD = 15/6 = 2,5

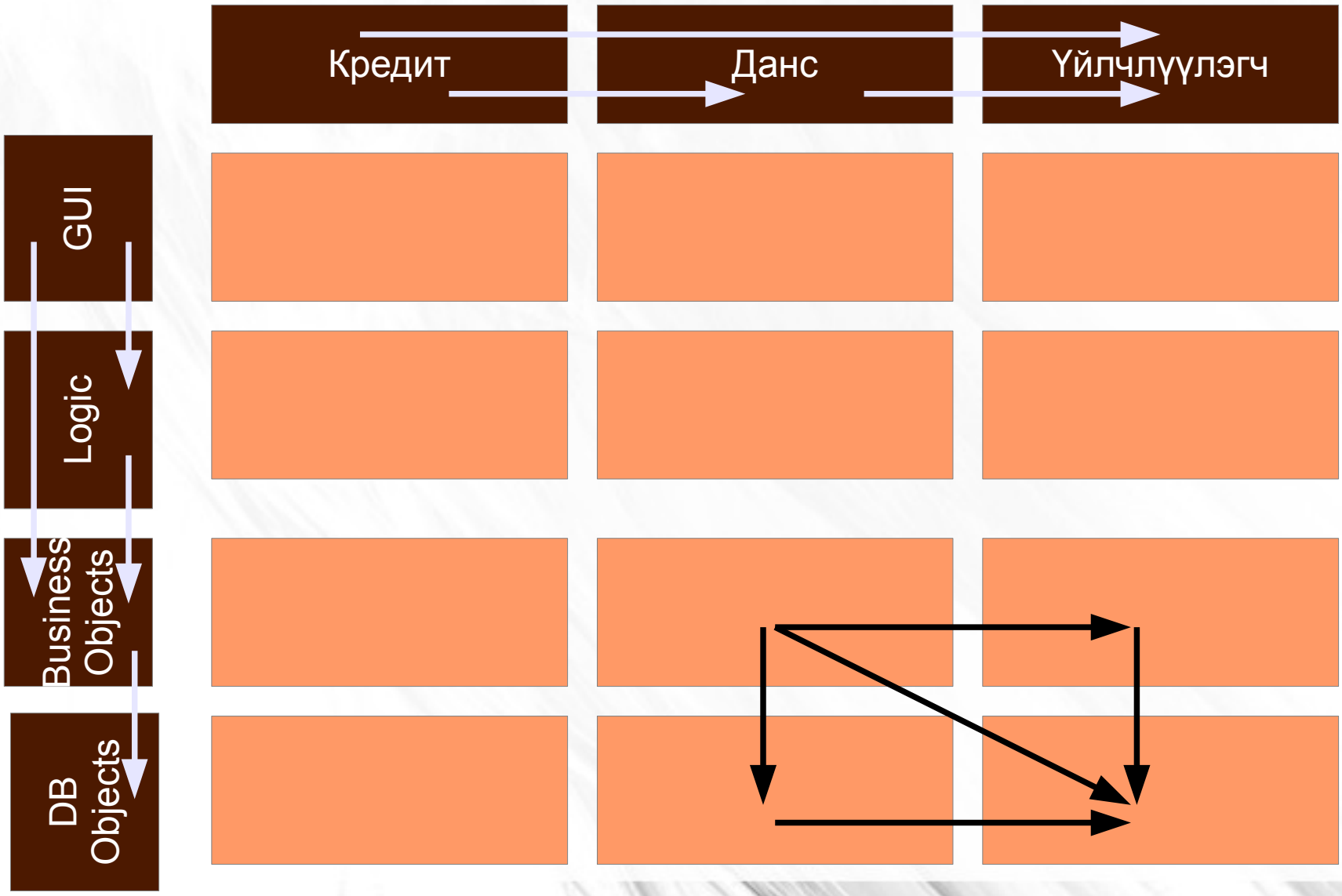


ACD = 26/6 = 4,33

- Циклүүд нь ACD үр дүнг муутгана.

Логик Архитектур
Зөвшөөрөгдсөн холбоосууд

Логик Архитектур
Зөвшөөрөгдсөн холбоосууд



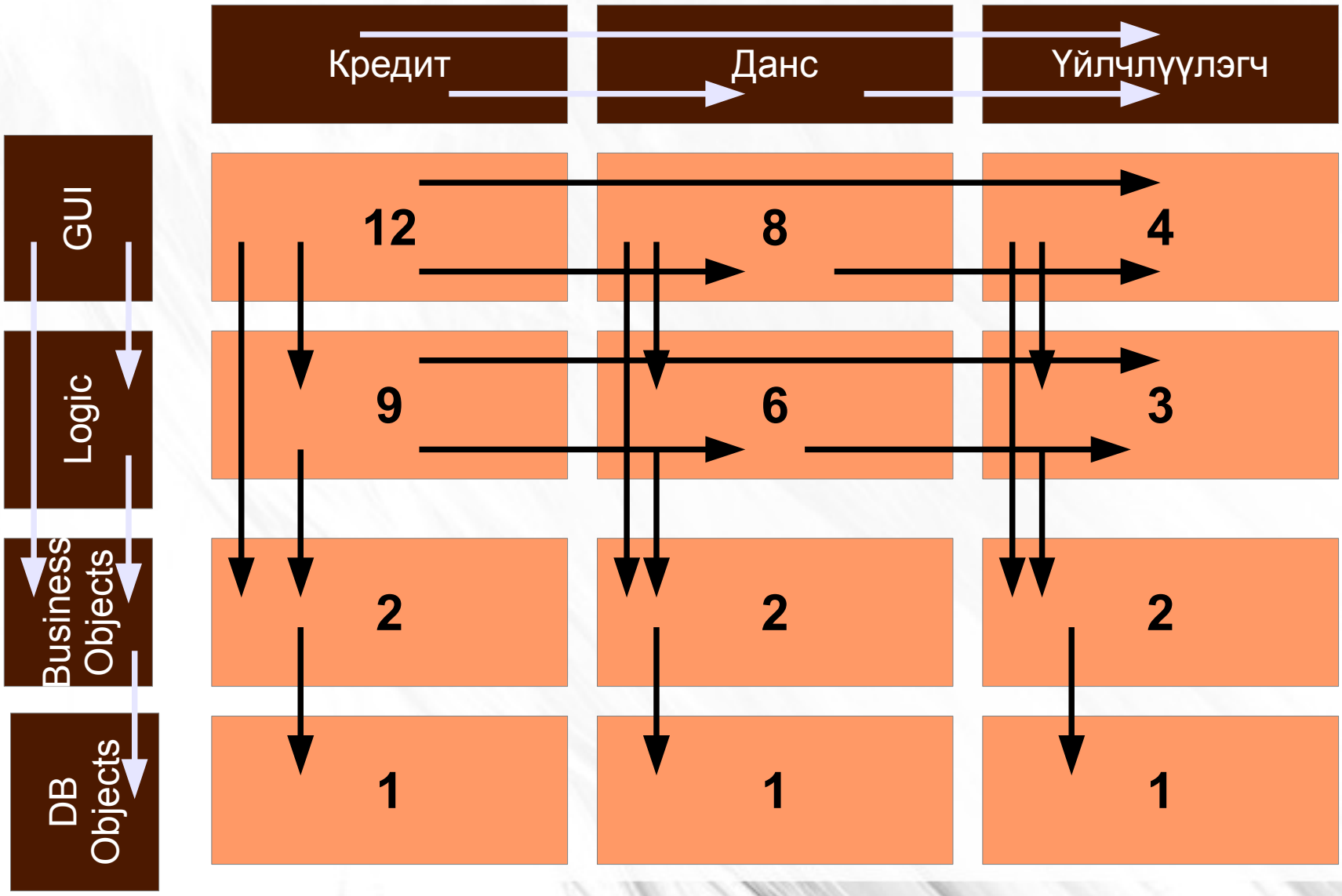
Average Component Dependency (1)

ACD = 60/12 = 5

Average Component Dependency (2)

ACD = 51/12 = 4,25

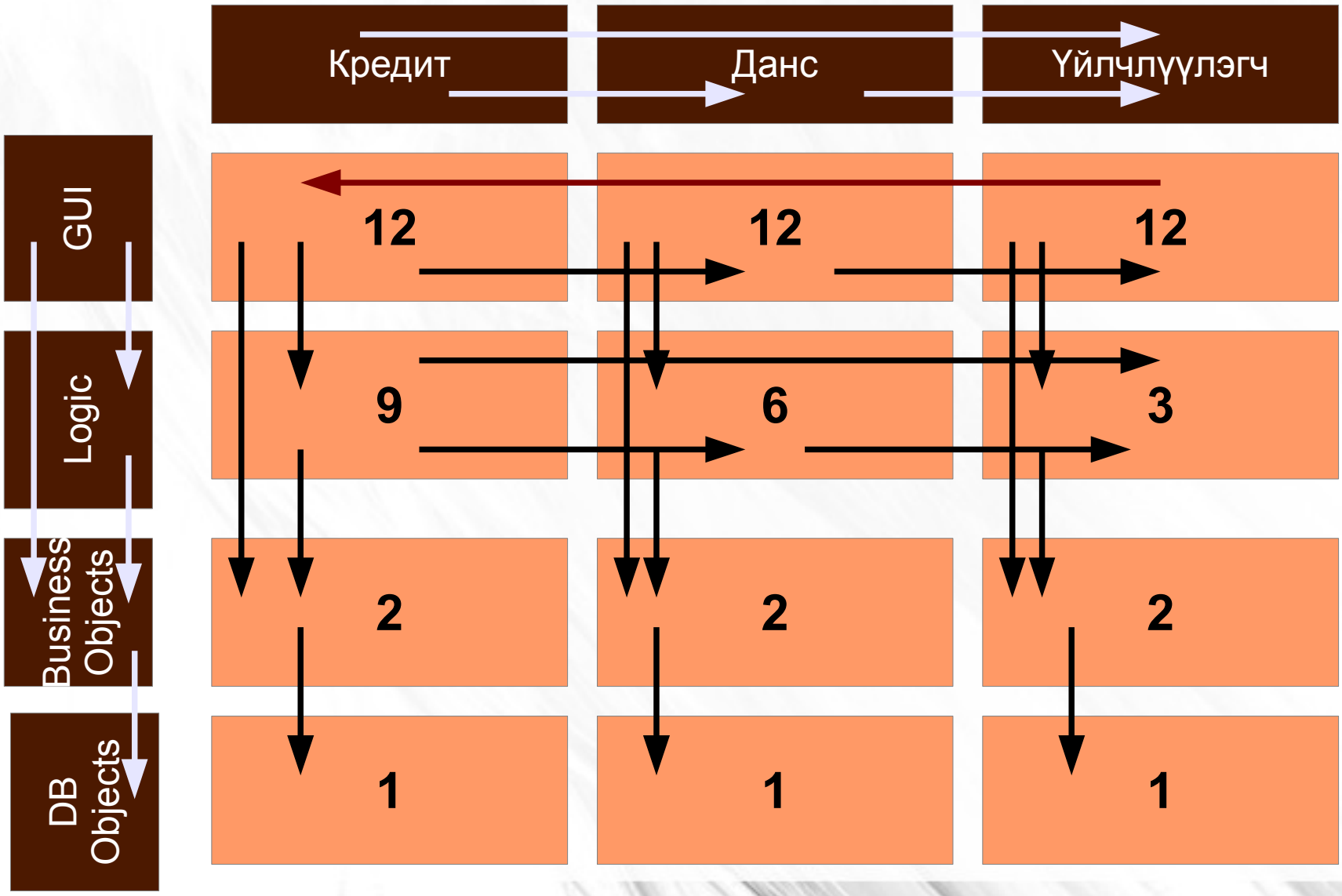
rACD = 15%



Average Component Dependency (2)

ACD = 63/12 = 5,25

rACD = ~23%



Stable Dependencies Principle - SDP

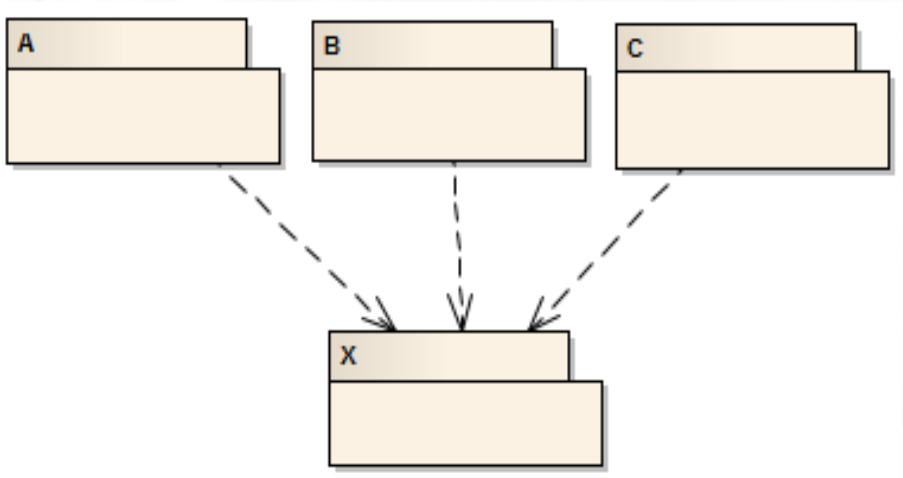
Depend upon packages whose stability is lower than yours

- Ca = afferent couplings

  Тухайн пакет доторх классаас хамааралтай гадаад пакетын классын тоо. (Incoming dependencies)
- Ce = efferent couplings

  Тухайн пакет доторх класс нь хамааралтай гадаад пакетын классын тоо. (Outgoing dependencies)
- Instability I = Ce / Ca + Ce. [0,1]

Stable Dependencies Principle - SDP



Instability I(x) = 0 / #Class + 0

- Хэрэв “Outgoing dependencies” байхгүй бол I = 0 бөгөөд пакет “stable”
- Хэрэв “Incoming dependencies” байхгүй бол I = 1 бөгөөд пакет “instable”

Дүгнэлт

- Удаан амьдрах систем бүтээхийн тулд системийн бүх төвшинд тодорхой, ойлгомжтой бүтцийг анхаарах хэрэгтэй.
- Энэ зорилгоор өч төчнөөн зарчмууд бий
- Бид тэндээс SOLID, CCP, CRP, ADP, SDP -тай танилцлаа
- Бүтцийн нөлөөллийг хэмжих цэгнүүр буюу метрик
- ACD-метрик, Тогтвортой байдал