SE302 – ПХ БҮТЭЭЛТ
Лекц 7

Developer Testing

- Чанарыг сайжруулах нь түгээмэл үйл ажиллагааны нэг
- Програм хангамжийг тестлэх олон төрлийн аргууд байдаг
  - Unit test
    - Класс, функц, жижиг програмын гүйцэтгэл
      - Нэг програмист эсвэл програмын баг бичсэн
  - Component test
    - Класс, Пакеж, жижиг програмын гүйцэтгэл
      - Олон програмист эсвэл програмын баг оролцсон

Developer Testing

- Integration test
  - Хоёр эсвэл олон класс, пакеж, жижиг програм хоорондоо хэрхэн зохицон ажиллаж байгаа
    - Олон програмист эсвэл програмын баг
  - Энэ төрлийн тест нь систем хоёр класстай болсон цагаас эхэлнэ
    - Систем бүрэн гүйцэт болтол үргэлжилнэ
- Regression test
  - Өмнөх тестийн төрлүүдийг давтан хийх
    - ПХ-ын доголдол олох зорилготой

Developer Testing

- System test
  - Сүүлийн байдлаар ПХ-н гүйцэтгэл
    - Бусад програм хангамж болон техник хамгамжтай
  - Энэ төрлийн тестэд:
    - Аюулгүй байдал
    - Гүйцэтгэл
    - Нөөц бага зарцуулалт
    - Цаг хугацааны асуудал
    - Доод түвшний тестэд ордоггүй бусад асуудал

Developer Testing

- Хөгжүүлэгчийн тестэд
  - Unit test, component test, integration test
  - Заримдаа regression test, System test
- Бусад тестийн төрлүүд
  - Тестийн тусгай хүн гүйцэтгэнэ, хөгжүүлэгчид хийх нь маш бага
    - Beta test (Хүлээлгэж өгөхөд бэлдэх тест)
    - Customer-acceptance test (Хэрэглэгч хүлээж авах тест)
    - Performance test
    - Configuration test
    - Platform test
    - Stress test
    - Usability test
- Тестийн ангилал:
  - Black box, white box

Developer Testing

- Тестийн ангилал:
  - Black box
    - Дотоод үйл ажиллагааг харахгүй
  - White box
    - Дотоод үйл ажиллагааг мэдэж байх
    - Хөгжүүлэгч өөрийн кодыг тестлэж байгаа шиг
    - Энэ хичээлийн хүрээнд white box тестийг авч үзнэ
- Testing
  - Алдаа илрүүлэх гэсэн утгатай
- Debugging
  - Аль хэдийн илэрсэн алдааны учир шалтгааныг олж зүгшрүүлэх

Role of Developer Testing in Software Quality

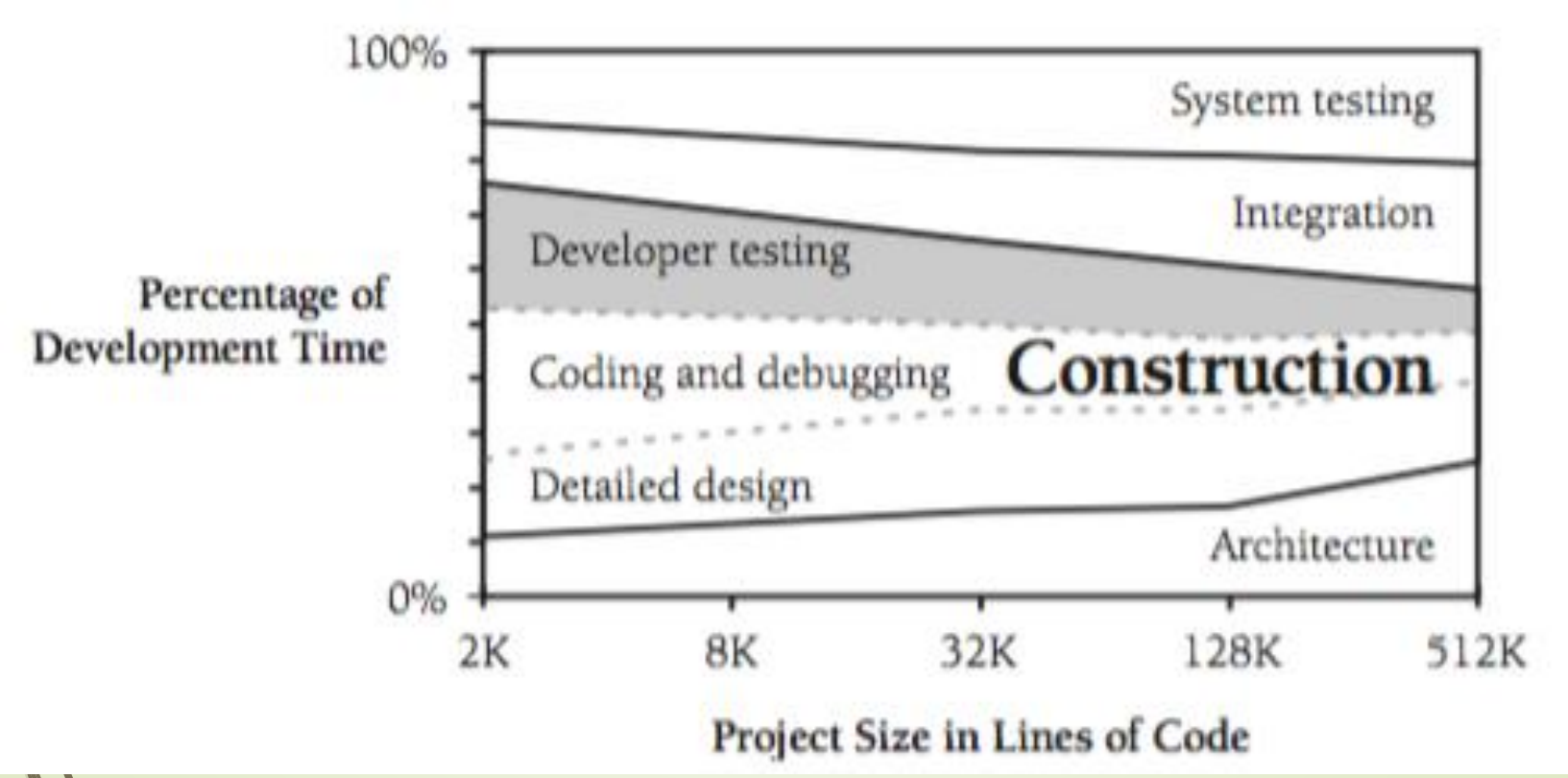
- ПХ-ийн чанарын чухал хэсгийн нэг
- Хамтын хөгжүүлэлтэнд алдаа илрүүлэх нь
  - Тестийн шатанд алдаа илрүүлэх зардлын хагас
- ПХ Хөгжүүлэлтийн үйл ажиллагаанаас бусадтайгаа адилгүй нь тестчилэл
- Тестчилэл нь ихэнх хөгжүүлэлгчдийн хувьд хүнд ажил байдаг
  - Зорилго нь алдаа олох, Нэг амжилттай тест нь ПХ-ыг эвдэж болно
  - Тестчилэл нь ямар ч алдаагүй гэдгийг хэзээ ч нотлож чадахгүй

Role of Developer Testing in Software Quality

- Тестчилэл нь өөрөө ПХ-ын чанарыг сайжруулахгүй
  - Илүү сайн хөгжүүл
- Тестчилэл нь таниас өөрийн кодноосоо алдаа хайхыг шаарддаг
  - Туршлагатай програмистуудаас мэдэгдэж буй 15 алдаанаас тест хийлгэхэд
    - Дундаж програмист 5 алдаа
    - Хамгийн сайн нь 9 алдаа илрүүлсэн
    - Ихэнхи илрүүлээгүй алдаа нь алдаатай гаралт
    - Алдаанууд нь илрүүлж болохуйц байсан…
- Та өөрийн бичсэн кодны алдаагаа илрүүлж чадна гэж бодож байна уу?

Role of Developer Testing in Software Quality

- Хэр их хугацааг хөгжүүлэлтийн тестэд зарцуулах вэ?
  - Төслийн нийт хугацааны 8-25%



Role of Developer Testing in Software Quality

- Хөгжүүлэлтийн тестийн дараа юу хийх вэ?
  - Хөгжүүлэлтэнд хийгдсэн ПХ-ын найдвартай байдлыг үнэлнэ
  - ПХ-ыг зөв болгоход туслах чигт хөтлөнө
  - Ямар төрлийн алдаа ихэвчлэн байна
    - Зохих туршилтын класс сонгох
    - Технизийн хяналтын үйл ажиллагааг сонгох
    - Ирээдүйн тестийн тохиолдол загварчлах

Testing During Construction

- Тестийн ертөнцөд энэ төрлийн сэдвийг заримдаа орхичихдог
  - White-box
  - Glass-box
- Оролт гаралт болон түүний дотор талыг харж тестлэх нь үр ашигтай
  - Хайрцагны дотор талыг харсанаар
    - Яг таг нарийн тестлэх боломжтой
- Мэдээж сул тал бас бий
  - Энэ кодыг та өөрөө бичсэн

Testing During Construction

- Бүтээх явцад
  - Ерөнхийдөө та функц болон классыг бичдэг
    - Оюун ухаандаа үүнийг шалгадаг
    - Тэгээд дахин шалгах, тестлэдэг
  - Интеграц хийх явцад
    - Нэгж хэсэг бүрийг та тестлэсэн
      - Өөр хоорондоо зохицож байгаа эсэх
    - Хэрэв хэдэн нэмж функц тодорхойлвол
      - Тэр үед нь тэднийг тестлэнэ
      - Функцуудыг тус бүрд нь тестлэх нь тийм ч хялбар биш
        - Дебаг хийх нь илүү хялбар
      - Хэрэв нэг функц шинээр нэмсэн бол
        - Дебаг хийх процесс илүү хялбар байна

Recommended Approach to Developer Testing

- Хөгжүүлэлтийн тестийн системтэй арга нь
  - Бүх төрлийн алдааг илрүүлэлтийг хамгийн их
  - Хүчин чармайлтыг хамгийн бага байлгадаг
- Үүнд:
  - Хэрэгжиж байгаа шаардлагатай холбогдох шаардлагыг тус бүрээр тестлэх
    - Шаардлага дахь ерөнхий орхигддог зүйлс
      - Аюулгүй байдал
      - Хадгалалт
      - Суурилуулах
      - Системийн найдвартай байдал

Recommended Approach to Developer Testing

- Хэрэгжиж байгаа дизайнтай холбогдох дизайн тус бүрээр тестлэх
  - Дизайны үе шат эсвэл аль болох эрт (боломжтой) тестийн төлөвлөгөө гаргах
  - Функц болон классын дэлгэрэнгүй кодыг бичихээс өмнө тестлэх
- Үндсэн тест хэрэглэх
  - Шаардлага болон дизайнд нарийвчилсэн тестийн тохиолдолууд үүсгэх
    - Өгөгдлийн урсгалын тест, кодонд шаардлагатай тестийн тохиолдолууд нэмнэ
    - Наад зах нь, кодын мөр бүрийг тест хийх

Recommended Approach to Developer Testing

- Алдааны шалгах хуудас ашиглах
  - Өнөөдрийг хүртэл хийсэн эсвэл
  - Өмнөх төсөл дээр хийсэн
- Бүтээгдхүүнд нийцүүлэн тестийн тохиолдол загварчлах
  - Шаардлага болон дизайны түвшинд алдаанаас зайлсхийхэд тусладаг
  - Кодлох үеийн алдаанаас илүү үнэ цэнтэй
  - Тестийг төлөвлөж, алдааг аль болох эрт илрүүлэх
    - Эрт илрүүлэх тусмаа зардал бага

Test First or Test Last?

- Хөгжүүлэгчид заримдаа гайхдаг. Аль нь сайн?
  - Кодны дараа тестийн тохиолдол боловсруулах
  - Эсвэл эсрэгээрээ
- Тестийн тохиолдол эхлээд боловсруулах шалтгаан
  - Кодноос өмнө тестийн тохиолдлыг боловсруулах нь
    - Кодны дараа тестийн тохиолдол боловсруулах шиг хүчин чармайлт шаардахгүй
  - Энэ нь энгийн тестийн тохиолдол боловсруулах үйл ажиллагааны дараалал
- Эхлээд тестийн тохиолдол боловсруулвал, согогыг эрт илрүүлж, түүнийг илүү хялбар засаж чадна

Test First or Test Last?

- Эхлээд тестийн тохиолдлыг боловсруулах гэдэг нь
  - Бага ч гэсэн шаардлага болон дизайны талаар кодлохоос өмнө бодно
  - Үр дүнд нь сайн код үүснэ
- Эхлээд тестийн тохиолдлыг боловсруулах нь шаардлагын асуудлыг илрүүлдэг
  - Кодлохоос өмнө
  - Яагаад гэвэл муу шаардлагаас тестийн тохиолдол боловсруулахад хэцүү байдаг
- Test-first programming нь ПХ-ын хамгийн ашигтай практикуудын нэг

Bag of Testing Tricks

- Тестээр програм нь зөв гэдэг нь нотлогддоггүй
  - Тестий хэрэглээнд програм ажиллаж байна гэдэг нь л нотлогддог
  - Байж болох бүх оролтын өгөгдөл дээр тест хийх боломжгүй

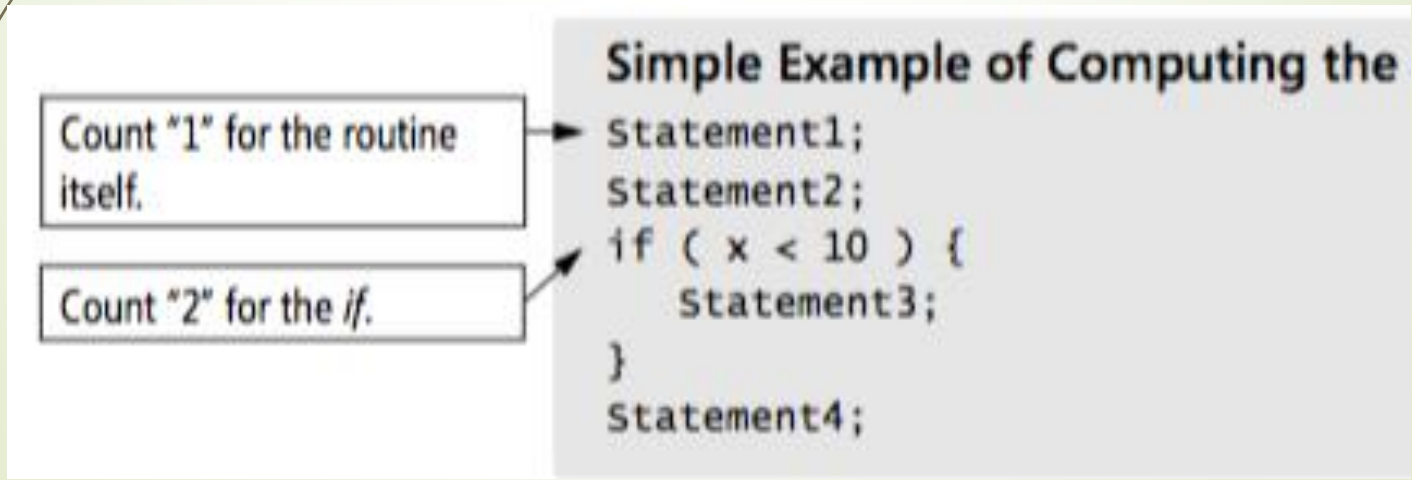
Name	26^{20} (20 characters, each with 26 possible choices)
Address	26^{20} (20 characters, each with 26 possible choices)
Phone Number	10^{10} (10 digits, each with 10 possible choices)
Total Possibilities	$= 26^{20} * 26^{20} * 10^{10} \approx 10^{66}$

Bag of Testing Tricks

- Дутуу тест (incomplete Testing)
  - Бүх боломжыг шалгах боломжгүй учир
    - Алдааг илрүүлэхэд дөхөмж тестийн тохиолдол сонгох
    - 10^{66} боломжтой ч цөөн хэд дээр алдаагүй бол
      - Бусад дээр нь алдаа илрэх магадлал бага
- Зохион байгуулалттай суурь тест
  - Тун энгийн ойлголт
  - Програмын statement бүрийг ядаж нэг удаа тестлэх
    - If, while …
  - Хамгийн бага тестийн тохиолдол нь програмын зам бүрийг шалгаж үзэх

Bag of Testing Tricks

- Тестийн тохиолдлыг тоолох энгийн арга
  - Функц эхлэхэд 1 -ээс эхэлнэ
  - Дараах түлхүүр үгнүүдэд 1-ийг нэмнэ
    - If, while, repeat, for, and, or
  - Case (switch) бүрт нэгийг нэмнэ, хэрэв default сонголт байхгүй бол дахиад нэгийг нэмнэ



Bag of Testing Tricks

- Өмнөх жишээнд хамгийн багадаа 2 тестийн тохиолдол үүсгэнэ
  - X < 10 үед биелэгдэх statement-д зориулаад
  - x>=10 үед биелэгдэхгүй statement-д зориулаад
- Энэ төрлийн тест нь кодын хэсэг бүр ажиллаж чадаж байгаа эсэхийг тестлэдэг

Bag of Testing Tricks

Example of Computing the Number of Cases Needed for Basis Testing of a Java Program

```
1  // Compute Net Pay
2  totalWithholdings = 0;
3
4  for ( id = 0; id < numEmployees; id++ ) {
5
6     // compute social security withholding, if below the maximum
7     if ( m_employee[ id ].governmentRetirementWithheld < MAX_GOVT_RETIREMENT ) {
8        governmentRetirement = ComputeGovernmentRetirement( m_employee[ id ] );
9     }
10
11    // set default to no retirement contribution
12    companyRetirement = 0;
13
14    // determine discretionary employee retirement contribution
15    if ( m_employee[ id ].WantsRetirement &&
16       EligibleForRetirement( m_employee[ id ] ) ) {
17       companyRetirement = GetRetirement( m_employee[ id ] );
18    }
```

Bag of Testing Tricks

- Жишээн дэх тестийн тохиолдолууд

Case	Test Description	Test Data
1	Nominal case	All boolean conditions are true
2	The initial *for* condition is false	*numEmployees < 1*
3	The first *if* is false	*m_employee[id].governmentRetirementWithheld >=MAX_GOVT_RETIREMENT*
4	The second *if* is false because the first part of the *and* is false	*not m_employee[id].WantsRetirement*
5	The second *if* is false because the second part of the *and* is false	*not EligibleForRetirement(m_employee[id])*

Bag of Testing Tricks

- Өгөгдлийн урсгалын тест
  - Хяналтын усргал болон өгөгдлийн урсгал нь компьютерийн програмд чухал
  - Энэ арга нь өгөгдлийн хэрэглээн дээр суурилсан
- Бүх кодын тал хувь нь өгөгдөл зарлан болон ачааллах хэсэг байна (Boris Beizer)

Bag of Testing Tricks

- Өгөгдөл дараах 3 төлөвт оршино
  - Defined
    - Өгөгдөл ачаалагдсан боловч, хараахан хэрэглэгдээгүй
  - Used
    - Өгөгдөл тооцоололд ашиглагдсан
      - Функцын параметр, эсвэл өөр байдлаар
  - Killed
    - Өгөгдөл нэг удаа тодорхойлогдсон, ямар нэгэн байдлаар тодорхойлогдоогүй болсон
      - Заагч, утгаа алдсан
      - For давталтын индекс
      - Файлын бичлэг заагч, файл нь хаагдсан

Bag of Testing Tricks

- Өгөгдлийн төлөвийн комбинаци
  - Defined-Defined
    - Нэг хувьсагчийг дахин тодорхойлох
    - Яг буруу гэж хэлж болохгүй
  - Defined-Exited
    - Локал хувьсагч, заавал утга олгох шаардлагагүй
  - Defined-Killed
  - Entered-Killed
  - Entered-Used
  - Killed-Killed
  - Killed-Used
  - Used-Defined

Bag of Testing Tricks

- Тест эхлэхээс өмнө эдгээр өгөгдлийн төлөвийн эдгээр гаж дарааллыг шалгасан байх
- Бүх боломжит тодорхойлсон хэрэглэгддэг замуудыг хэрэгжүүлсэн байх

Bag of Testing Tricks

- Structured basis test - 2
- Data flow test - 4

Java Example of a Program Whose Data Flow Is to Be Tested

```
if ( Condition 1 ) {
   x = a;
}
else {
   x = b;
}
if ( Condition 2 ) {
   y = x + 1;
}
else {
   y = x - 1;
}
```