CS1632, LECTURE 2: TESTING THEORY AND TERMINOLOGY

Wonsun Ahn

Key (🔑) concept to the course

Expected behavior vs observed behavior

Expected behavior vs observed behavior

You need to know what “should” happen under some circumstances, then check to see if that behavior actually occurred.

For example, assume I have a function foo, which accepts an integer, a, and returns a float. What should happen if I send in the value a = 42?

This is a simple idea, but it’s the “Fundamental Theorem of Testing” (although note that we may violate it later…)

Example

Assume foo is supposed to return the square root of the passed in value a.

When I send in the value a = 42, then I expect to be returned the value 6.48074069841.

When I send in the value a = 9, then I expect to be returned the value 3.

When I send in the value a = -1, then I expect….

THE IMPOSSIBILITY OF EXHAUSTIVE TESTING

- Let’s say we want to ensure that our square root method will never fail, no matter what we send in. Assume we are using a standard Java int (signed 32-bit integer)
- How many values do we have to test?

4,294,967,296

What if there are a 1000 method calls?

- Assume each method call accepts one 32-bit int argument
- Remember that methods in a Java-like language can have side-effects (in other words, they are not pure functions)
  - Can access / modify global variables, global data structures
  - Means a method call is affected by all previous method calls

4,294,967,296 ^ 1000

THAT'S EQUAL TO...

91171950785279002509723338967578745479915846704161093266316858586590561893971899669618585741969928523387207745761469132800197981751886314439367551781375622355182494132784122242065904571923125296267977156632231162513245
43070355970530225215612468985466808580243659545868971431191906116382126383427634800506871882582153907201719884414418866487912157597627253893662165574468243228700314685915827483905821076923315530782979561709655922726932
30942153795374906434644294709320337977283581908983329406571317899264697722812061019380899236287721753962035495230692778928668862404748824570782188307505182107784007030038445100444660699812696177811947887576560255655908
39684527043545238966575573403559687951326963892838818040660850711941690965464303296715997735315114393298314639765906325716916902265058485269112748232055283264916137426095401499226015801097484231501236124348080603774582
48309478650101041149164099071980888428421368546177179447495527132137857711089612888118312376663794867678135461880066435240486394452956529788529633837378507602411635143827166518753745556991400676914560717482710875903621
73711860044704084625336091543500691714323919052366981476276437065678838563998878381885773000341053862602758014572499348041757394660676973509282166930632955880631414046371713765667379841266106690291293383395081456948512
01748679819045973059635627618804971155841944169287806872340397299377605190620469299082702993326951264149037879410723677767085085619444240438253667000617814806792353740813593652081830044375906970007383971690175067381164
96856048986391207546309917618762444947550045733654312891655363179758753293636106046873154110605833286838622753457575754632405563247121235420503068705932612812423730604617707474908267202439896279150918198244130312153582
12115365367031228798361900436098033593939317108507725449460576386728537579974232974920375624778180923231343430936921083911139534659465800121513010419901123405611013130169487104000337304071364153665375401392391571941559
51689480400999761684077927466992190189673014569128322473733058959493641343438083324990599872647828146559491250565907817965564228047463516269798592276478265498502288667448764514477511517488432996198979728103818498694329
29215610249668503146259070821149147147929671318636093993055261646254668852152256223974186547033820664324635426978298582188285589712230975600324625417593418038572317911977583717648988763575138455279752979839590902979465
13522186617719293774695529057040663935309335892299198791524237701995080418148253222726701943143791040308457583407799395569470968122500963439811675572475959611793311581144998431728888764229631959149698707010011920915003
48562249270525324932249759835996854932467548669022136744548330632615545175351012218282500250610087479650864554497383187477097935862023421473171711484641968407993497083283831690153260491677028567298028388862349683564297
59705090730476854531567246951502200043433665070248298446185971155224458673695436284935491043078408600113757452408977665439302231558859343031612693920949653983249653540586738417754022522371175536071967190921501772406926
20980023992930495414830641411644455664330843363657287811979996492743740691150412105301771458223402797056844529954715900646233754267861944256911059891234426533672099268369543158016134038936671758535029408020838733922701
53115211059806226605339810254098811434005506705660238583972530934203626456667447050280426434407146084463844557676231376145368465076041733818936624509202579048128118406191307095564038850600579284504917698986384288769207
48664932697620208305902491339825931013795092510341172118372458616534736074123602439122702942241955080805738531528651080242146153895816621263769903867197484178118641718558329259982627315712447366427920414561433567394573
08114036780697679783341964181398779863871025415770527055336426195241935221037058819757293995048374550819894222334319683340221385890807072484235545990481780316101856266200860564034686501297109898065953529438125588949324
22584722634890931081243356161736189915233968129907439049015691222260227443224611111500807098151836802194371233157448935859841962696238686678704566197978152142493878887304368391542652290664310132765507145505463076295716
50128910962661940478119939159806757686024198773028092601855291898050622497661356201163684378850172037399421215068245561343325285356592012591311784550393137415359042915578453153620223491605527664643875118015432832806053
198111151232515229653713454407762650524669837520914045854861909367458874566508760789160702913223927728007529735726074388041982638528921416864247711572881283538001106835446598432243170893021429378896556776459613920903704
73274287938090683917540357950743848749217350818082417777694163054823528542866124233219791730361162020893741006417456721097578900625585271917414813711243285365360713240935112537687502353662549143565740533243101519462438
644953280690663288102105837540931331032882805229979716016564675101124387212122500726527821208598503621726442301618421428577221233804861744865377785293548216832624613867888349030211472436785584248475479178882183717433693
96014783628533549260842862242006394831929589376558734119082694146131929973825734930816757160607652858999523782850359528013781037135707033245059781779843643418441051467371181772423410381605071312319605959717198055966289
59195234736744295166104361159360683036163270202634552213513318566484835719403880220554033411796349372495875586118656207435543996163187158526968403732471105139039185642674088138396652624325867988110931314982914084051978
45725819381920308684240484484895447237705042852968891527063132811833365451819936318508754168233009554017739501850777358510541912533110741021804219896419802358275706876762266917266438464675594333784867480343557358947354
48732740215206929436333628804794981719953369755105959470423321715011000131877547406260799269166421891075050917361068365315450114841224548170357095043100966938379009438490468208206471876118259240141006749939528679101009
64206882680473908139252299286389412685826872118366179890355300112968574503353132511242959447268830745086095616201428164516006842586675171587915412964811341669966410073680636661573440251376914912866556128990683119032449
034176463111442799492870062177647662756693303366255863104891484478589577569081242157664752940024790629382545585169913923273890731266736364918635663674009747192507598364366197907583956085292625500666087449730382255060
48304984084392557094146486862343926263196876267332053548430552024066991674373678350083624926671265472746238488720834245889184613602895314135581868837559735296094556498461553569091675076360406777996909483738743390919096
78020238234149134995869090366492740058885709001364519514244721833733934300152259696432668144071059650399294221351755638277275674759801798201332587400568052987096170141043198807805986243798903781937254970856378737268604
64463074541443366212534369684813551693564936687254941994522934018195017525549444700868823993115636034505529461045301060444933263556878289734998459492921561463614328398619833494453541594386780361789258359429578657442241
278750129821242509853455254082343976676322784815305733349935732081398147936568808336894474701914343812413184588265294713070183737202492690154745208876587343392923311589488849535636213550805440113278384271916562263763520
39722684020150152597638897706071847162525995050001361251800596808445848537576364111221755369986030360621366345532458637615648908140533087938693489508859067427442899832936297018998228449831175764807232863648922187671016
11351195619259805644169191318079341020584815138006601145763705637068546617800187772177796458571189488073490895635270926467285961127822477927853992571778847731132691047698190664829001789320595128841188792506877062579097
174796930788273116126688447542429560045820489926800784531078272007111163972073005235829026536101025799287759959206873392655993435124866201190839798779071380018942899033123296996732164485279686156579886935157603735477676
88171833634237729506529854937067332274928489685775706748317741309921732522408624164689770061472968956984330802866600149652352663267117018018005048565315871324035399562544124091808660250364716982216776762713148483190201
33060710757097556172829131816771602541386447269496164043404483399094935584070401279683924483408673738377092080244683066027833095239862678817090737935343959060011556231957975179403067981594889330371163905022902237396276
4091075519566057499424903725306195328374308091791810165052653503077660908223164471043406263905344031520182132013000761830308833178382109642733923978710854460357091072928739692277111756018614686106634321952938137996608
19426911683752869554731449884793943898892172669177138946201684065003046090134922482829326394901367779289690617251263450410412926093413570538510501057041126993327840505014890476294508401823374653367432653177988952307803
92314576200503423597968367372809336735996907140973667671428226007380695766185164250650016216771668162576408426646905432743090364395271974904731268728621503370319375930779994574095357126317236912267575371995190901805593
07849699501658776406304767707673573931741290259186841720383735311673320179029952862572753657067818949525864619245819944009529159128396942840691266543209312090444342470604790725815609989670564251784018052956072578525871
10595668561420513067073525584016125889995194437311780560399847532160785620030285068004948206238286171290286662645556323849029300918292472784497730305116748421516078510258210907429415144317360547314932626065327360243862
91288197779696647358206451825116988631320595390442997608759132222017057289162026843989591487287945061431515735679460451787531220462162654097837015440951944883754281142294559331055660377889666445294777394725798978241129
674438343863415696398444496667595778 29376

TEST CASES!

What if the argument is an object reference?

- That object could be a list data structure, a tree, a graph, …
- How many shapes can a tree take?

Would testing all the combinations of arguments guarantee that there are no problems?

LOL NOPE

- Compiler issues
- Parallel programming issues
- Non-functional issues (performance, usability, etc.)
- Floating-point issues
- Integration issues
- Systems-level issues
- Ambiguous or misunderstood requirements

LOL NOPE

- Compiler issues
  - Your source code does not run on the computer, the compiled binary does
  - Depending on compiler and compile options, many different binaries can be produced! Besides the binary you used for testing.
  - What if the compiler has a bug? (Rare)
  - What if the compiler *exposes* a bug in your program? (More frequent)

```
int add_up_to (int count) {
  int sum, i;    /* some C compilers will init sum to 0, some will not */
  for(i = 0; i <= count; i++) sum = sum + i;
  return sum;
}
```

  - Big problem for C/C++, less of a problem for Java since all bytecode is run on JVM (In above example, Java Virtual Machine always initializes all variables to 0)

LOL NOPE

- Compiler issues
- Parallel programming issues
  - If there is a data race, the result of your program is undefined
    - Doesn’t matter whether you are using C/C++ or Java
    - Worst part: for many data races, result is correct 99% of time, masking the bug
  - Even with no data race, the result of your program is often nondeterministic (Depending upon the respective speed of each thread)
    - To thoroughly test, you have to vary the speed of each thread, even for same input

LOL NOPE

- Compiler issues
- Parallel programming issues
- Non-functional issues (performance, usability, etc.)
- Floating-point issues
- Integration issues
- Systems-level issues
- Ambiguous or misunderstood requirements

Testing = ART + SCIENCE

- There are techniques for testing which can reduce the number of tests necessary for sufficient test coverage.
- Defining what “sufficient test coverage” means is subjective.
- We must rely on domain knowledge to decide.

Equivalence class partitioning

- We can partition the testing parameters into “equivalence classes”
  - Equivalence class = a natural grouping of values with similar behavior
- For example, in our square root method:
  - Negative numbers (input) -> Imaginary numbers (output)
  - 0 -> 0
  - Positive numbers -> Positive numbers

Equivalence classes are strictly partitioned

- For any given input value, it must belong to one and ONLY one equivalence class (strictly partitioned)
- If there are values that belong to multiple equivalence classes, you probably need another equivalence class
- Example:
  - Right handed people -> writes with right hand
  - Left handed people -> writes with left hand

  Jane can write with both hands. Which equivalence class does she belong to?
  - Solution: add “Ambidextrous people -> writes with both hands”

Multiple partitionings

- Assume in the previous square root method, if the result contains a decimal point (e.g. *1.3* or *2.23i*), it prints in red, otherwise in black.
- Now we have two partitionings:
  - The positive / 0 / negative partitioning on the previous slide
  - The decimal / non-decimal partitioning on this slide:
    - Number contains decimal -> output printed in red
    - Number does not contain decimal -> output printed in black
- Therefore, a value now belongs to two equivalence classes, but in different partitionings (e.g. value *1.3* belongs to “positive” and “decimal” classes)
- A set of values can be partitioned in limitless ways

Values do not have to be numeric

- On Twitter, if you follow somebody, you see all of their tweets, unless they are writing directly to somebody you do not follow.
- Equivalence classes:
  - You do not follow person A -> DO NOT see the tweet
  - You do follow person A, they are not writing directly to somebody -> see the tweet
  - You do follow person A, they are writing directly to person B, whom you also follow -> see the tweet
  - You do follow person A, they are writing directly to person B, whom do you not follow -> DO NOT see tweet

Values do not have to be numeric

- Suppose Twitter only allows alphanumeric [A-Za-z0-9] characters, and tweets must contain at least one character. Tweets that contain any invalid characters are not posted.
- Equivalence classes (NV = number of valid characters, NI = number of invalid characters):
  - (NV >= 1, NI == 0) -> Post the tweet
  - (NV == 0, NI == 0) -> DO NOT post the tweet
  - (NI >= 1) -> DO NOT post the tweet (note NV is irrelevant here)

Test Each Equivalence Class

- Pick at least one value from each equivalence class
- This will ensure you capture behavior from each “class” of possible behavior
- Will find a good percentage of defects without exhaustive testing!
- We reduced the problem something a human can do! Woo-hoo!
- How to pick the input? Well, that is part of the art.
  - However, there are some good guidelines!

Interior and boundary values

- Theory: Problems are more prevalent on the boundaries of equivalence classes than in the middle.

Why?

- Suppose expected behavior is:
  - Method shall take citizenship and age as arguments
  - Method shall determine whether a person can be US president according to a set of rules
  - Rule 1: Person must be a US citizen to be US president
  - Rule 2: Person must be 35 years or older to be US president

- Suppose implementation is:

```
boolean canBePresident(int age, boolean citizen) {
   return age > 35 && citizen;
}
```

- Is observed behavior the same as expected behavior?

Equivalence class partitioning

CANNOT_BE_PRESIDENT =
[…19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34]

CAN_BE_PRESIDENT =
[35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50…]

Try to test both boundary and interior values

CANNOT_BE_PRESIDENT =
[…19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34]

CAN_BE_PRESIDENT =
[35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50…]

- At the boundary values (shown in red)
- In fact, there is a bug at: `age > 35`

Try to test both boundary and interior values

CANNOT_BE_PRESIDENT =
[…19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34]

CAN_BE_PRESIDENT =
[35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50…]

- Testing interior values is also important to see behavior in interior

Try to test both boundary and interior values

CANNOT_BE_PRESIDENT =
[…19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34]

CAN_BE_PRESIDENT =
[35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50…]

- **Are we done?**

“Hidden” (IMPLICIT) boundary values

- The boundary values we have gone over already are explicit – that is, they are defined by the requirements of the problem itself.
- Some boundaries are implicit – they are generated from the domain, architecture, hardware, or other elements:
  - MAXINT, MININT
  - Maximum precision of a floating point value
  - Allocation limitation (memory, hard drive space, network bandwidth, etc.)
  - Physical world boundaries (weight can’t be negative, Y2K won’t happen, etc.)
    - Side note: Y2K did happen and anti-gravity may yet happen

Add implicit boundary values

CANNOT_BE_PRESIDENT =
[MININT,…,-1,0,1,…19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34]

CAN_BE_PRESIDENT =
[35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,…,MAXINT]

- MININT, MAXINT: hardware boundaries
- 0: physical world boundaries (age cannot be negative)

Base, edge, and corner cases

- **Base case** – An element in an equivalence class that is not around a boundary (interior value), OR an expected use case.
- **Edge case** – An element in an equivalence class that is next to a boundary (boundary value), OR an unexpected use case.
- **Corner case** (or **pathological case**) – A case which can only occur outside of normal operating parameters, or a combination of multiple edge cases.

Black-, white, and grey-box testing

- **Black-box testing**:
  - Testing with no knowledge of the interior structure or code of the application
  - Tests are performed from the user's perspective, looking at the system as a whole
- **White-box testing**:
  - Testing with explicit knowledge of the interior structure and codebase
  - Tests are performed at the code-level (e.g. testing individual methods or classes)
- **Grey-box testing**:
  - Testing with some knowledge of the interior structure and codebase
  - Knowledge may come from partial inspection of code or a design document
  - Tests are performed from the user's perspective, but informed by tester's knowledge

Black-box testing examples

- Accessing a website, using a browser, to look for flaws
- Running a script against an API endpoint
- Checking to see that changing fonts in a word processor works

White-box testing examples

- Testing that a function returns the correct result
- Testing that instantiating an object creates a valid object
- Checking that there are no unused variables in a method
- Checking that exceptions are properly caught and handled

Grey-box testing examples

- *Reviewing code* and noticing that bubble sort is used. Then write a *user-facing test* involving a large input size.
- *Reviewing code* in a web app and noticing user input is not properly sanitized of code. Then write a *user-facing test* which attempts SQL code injection or cross site scripting.
- *Reading a design document* and noticing a critical network connection through which a lot of data passes through. Then write a *user-facing test* that stresses that network connection.

Static vs dynamic testing

- Dynamic testing = code is executed (at least the part that is exercised in that test run)
- Static testing = code is not executed

Dynamic testing

- If you’re thinking about testing, probably what you are thinking about.
  - Code is executed under certain circumstances
    (e.g. input values, environment variables, compiler, OS, runtime library, etc.)
  - Observed results are then compared with expected results
- Much more commonly used in industry
- The majority of the class will be about dynamic testing

Static testing

- Code is reviewed by a person or testing tool, without being executed
- Examples:
  - Code walkthroughs and reviews
  - Source Code Analysis
    - Linting
    - Model checking
    - Complexity analysis
    - Code coverage
    - Finite state analysis
    - … COMPILING!