# Dagskrá í viku 5 – kafli 7 í P&M


- **Prófanir**
- Einingaprófanir í Python

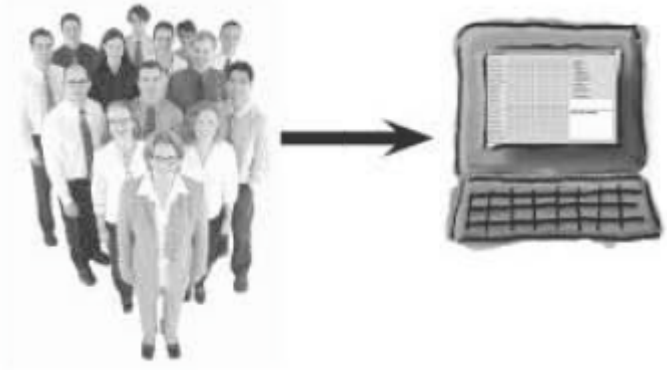
# Prófanir á hugbúnaði

- Útbreiddasta aðferðin til að auka gæði
- Finna **einkenni** villna

  ( síðan þarf að finna þær og leiðrétta (aflúsun) )
- Framkvæmdar af
  - Forriturum
  - Prófurum (sérhæfðir starfsmenn)
    - álagspróf, nothæfispróf, prófanir á mismunandi vélbúnaði, prófanir á uppsetningarferli, prófanir á afköstum,
  - Notendum
    - beta-prófanir meðal notenda

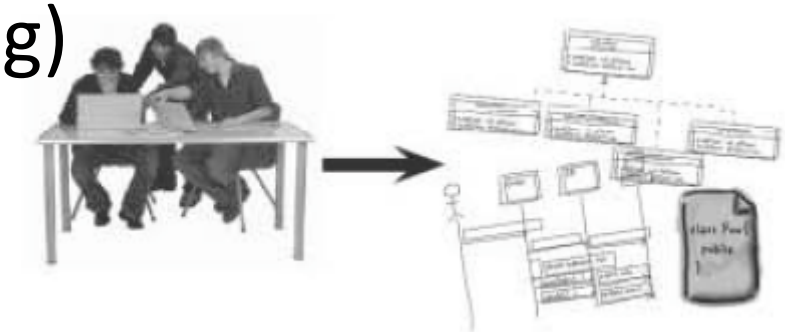
# Mismunandi sýn á kerfið

- Notendur (black-box testing)
  - Sjá kerfið að utan
  - Kerfið er “svartur kassi”

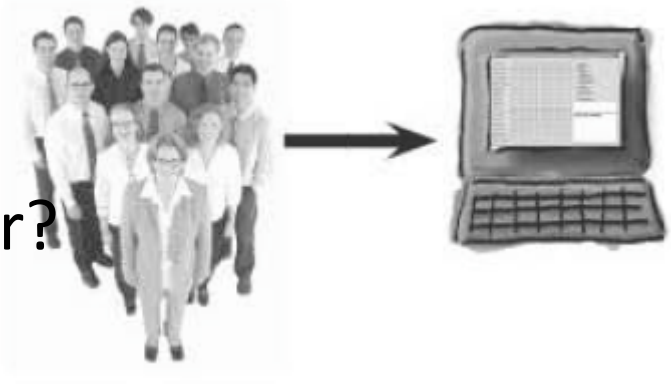


- Prófarar (grey-box testing)
  - Þekkja e-ð til innviða
  - Kerfið er “grár kassi”

- Forritarar (white-box testing)
  - Sjá alla hluta kerfisins
  - Kerfið er “hvítur kassi”



# Notendur



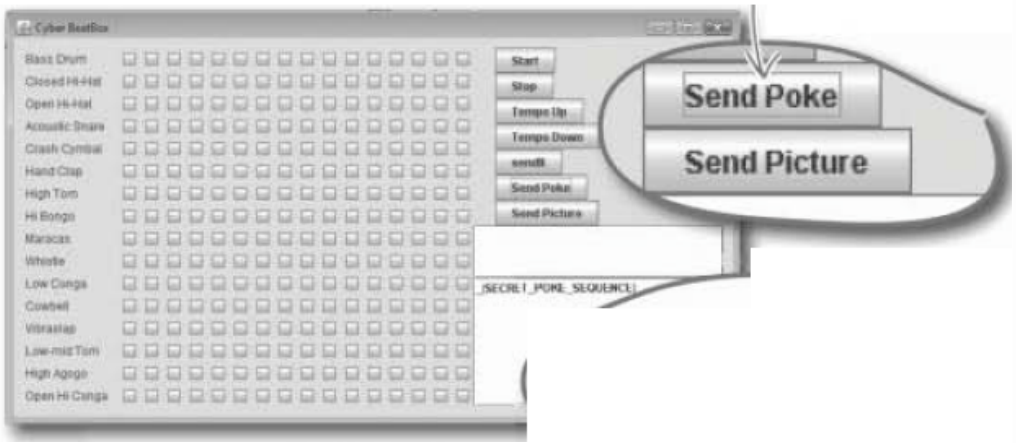
- Virkni
  - Útfærir kerfið notendasögur?
- Inntök meðhöndluð rétt?
- Úttök rétt reiknuð
- Eru ástandsfærslur í lagi?
- Jaðartilvik

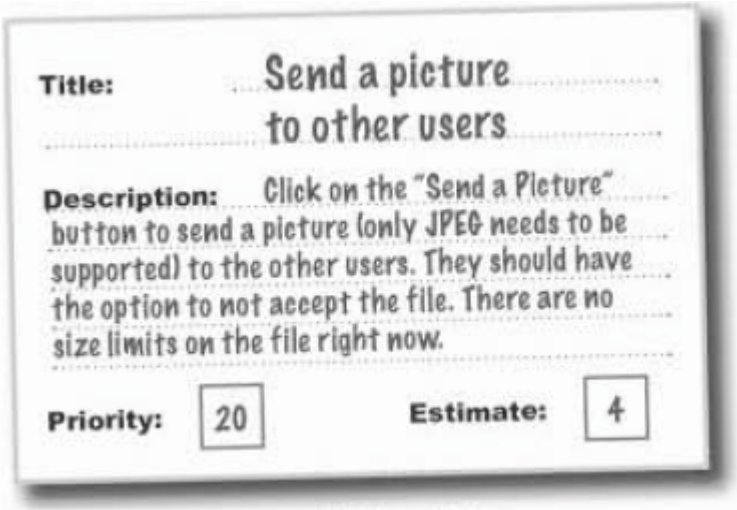
# Black-box testing

- Focuses on *input* and *output*
- **Functionality** – does the system do what the user story says it is supposed to do?
- **User input validation** – test invalid input and make sure the system rejects them with an error the end user understands.
- **Output results** – does the system show you the right output with this input?
- **State transition** – does the system move from one state to another according to very specific rules?
- **Boundary case and off-by-one errors** – test with a value that is just a little to small or just outside the maximum allowable value. (This is a very common mistake)

# Æfing: Svart/grákassa prófanir

- Notendur BeatBox geta sent myndir sín á milli



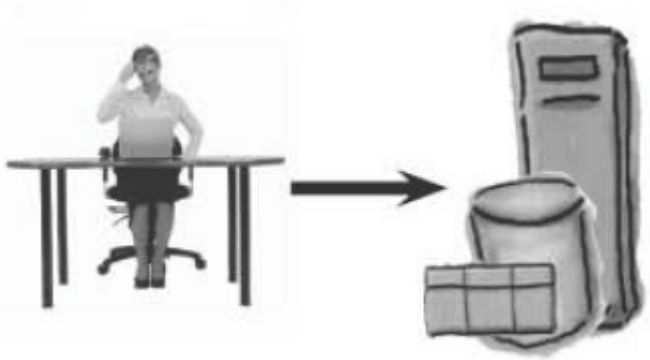

Útbúið 2 próf fyrir notendasöguna

1. **Lýsing**: Senda JPEG mynd til annars notanda

   **Framkvæmd**: ...

# Prófarar

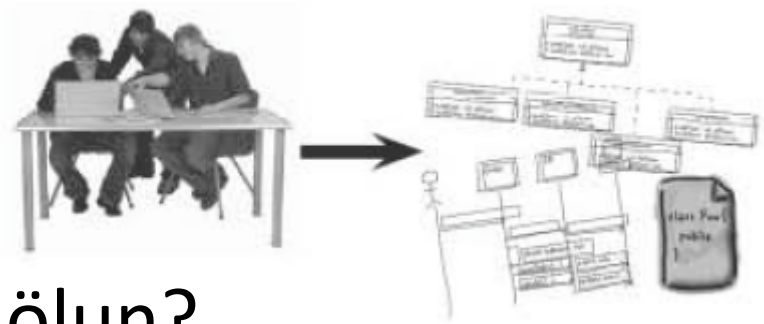
- Er innri skráning (t.d. gagnagrunnsfærslur) í lagi?
- Eru gögn sem eru send yfir í önnur kerfi (t.d. greiðslu-gáttir) á réttu formi?
- Hreinsar forritið til eftir sig?
  - kanna minnisleka, opin "port", "resource"leka, opnar skrár, ...



# Grey-box testing

- Gets you *closer* to the code
- **Verifying auditing and logging** – make sure your system is auditing and logging correctly, might require a log viewing tool, auditing report or maybe querying some database tables directly
- **Data destined for other systems** – check output data and data you're sending to other systems (when your system should be sending information to other systems)
- **System-added information** – Systems often create checksums or hashes of data to make sure things are stored correctly (or securely). Hand-check these. F.ex. System generated timestamps being created in the rigth time zome and stored with the right data.
- **Scraps left laying around** – Your cleanup has to be good. If not it can be a security risk or a resource leak. Make sure what should be deleted is really deleted (a mistake that cost Vodafone a lot recently). This includes memory leaks, checking registry, making sure that uninstalling applications leaves the system clean.

# Forritarar



- Prófa allar greinar í kóða
- Villumeðhöndlun í lagi
- Er virkni í samræmi við skjölun?
  - t.d. ef skjölun segir að tiltekið viðfang megi vera []
- Meðhöndlun “resource”a í lagi
  - Hvað gerist þegar minni, diskpláss, ... sem kóði biður um er ekki á lausu?

# White-box testing

- Uses inside knowledge at the deepest levels of testing.
- **Testing all the different branches of code** – You should be looking at *all* of your code. What data do you need to send in to get the class you]re looking at to run each of those branches?
- **Proper error handling** – Are you getting the right errors back when you send in invalid data? Is your code cleaning up after itself? By releasing resources like file handling, mutexes or allocated memory?
- **Working as documented** – Check if the parts are working as they say they will. If the method should be thread-safe test it from multiple threads. If a method needs a certain security role to call it then try it with and without that role…
- **Proper handling of resource constraints** – What does the code do if a method tries to grab resources it needs and can’t get them? Are these problems handled gracefully?

# Unittest in Python

- A few good links:
  - http://docs.python.org/2/library/unittest.html#
  - http://www.openp2p.com/pub/a/python/2004/12/02/tdd_pyunit.html
  - http://www.onlamp.com/pub/a/python/2005/02/03/tdd_pyunit2.html
  - http://www.diveintopython.net/unit_testing/index.html

# Einingapróf í Python **með unittest**

**Dæmi**: Prófa maxseq og maxdiff föllin (úr CodeCamp)

```
def maxdiff(values):
    md = abs(values[1] - values[0])
    for i in range(2, len(values)):
        md = max(md, abs(values[i] - values[i-1]))
    return md

def maxseq(instring):
    i = 0
    seq=[]
    while i < len(instring):
        j = 1
        while i + j < len(instring) and instring[i] == instring[i + j]:
            j += 1
        seq.append(j)
        i += 1
    return max(seq)
```

Vistað í skránni maxutil.py

unittest pakkinn kemur með Python

unittest er Python útgáfa af JUnit fyrir Java

# Einingapróf í Python **með unittest**

## Útfærum klasa **testMaxFunctions**

```
import unittest
import maxutil

class testMaxFunctions(unittest.TestCase):
    def test_maxseq(self):
        self.assertEqual(maxutil.maxseq('abc'), 1)
        self.assertEqual(maxutil.maxseq('aaxxxt'), 3)

    def test_maxdiff(self):
        self.assertEqual(maxutil.maxdiff([5,23,1,0]), 22)
        self.assertEqual(maxutil.maxdiff([23,5,1,0]), 18)

if __name__ == '__main__':
    unittest.main(verbosity=2, exit=False)
```

erfum frá unittest.TestCase

Próf fyrir maxseq

Próf fyrir maxseq

Nöfn byrja á “test”

Framkvæma próf

# Einingapróf í Python **með unittest**

## Útfærum klasa **testMaxFunctions**

```
import unittest
import maxutil

class testMaxFunctions(unittest.TestCase):
    def test_maxseq(self):
        self.assertEqual(maxutil.maxseq('abc'), 1)
        self.assertEqual(maxutil.maxseq('aaxxxt'), 3)

    def test_maxdiff(self):
        self.assertEqual(maxutil.maxdiff([5,23,1,0]), 22)
        self.assertEqual(maxutil.maxdiff([23,5,1,0]), 18)

if __name__ == '__main__':
    unittest.main(verbosity=2, exit=False)
```

erfum frá unittest.TestCase

Próf fyrir maxseq

Nöfn byrja á “test”

Próf fyrir maxseq

Framkvæma próf

Niðurstöður

```
test_maxdiff (__main__.testMaxFunctions) ... ok
test_maxseq (__main__.testMaxFunctions) ... ok
----------------------------------------------------------------------
Ran 2 tests in 0.015s
OK
```

# Einingapróf í Python **með unittest**

## Útfærum klasa **testMaxFunctions**

```
import unittest
import maxutil

class testMaxFunctions(unittest.TestCase):
    def test_maxseq(self):
        self.assertEqual(maxutil.maxseq('abc'), 1)
        self.assertEqual(maxutil.maxseq('aaxxxt'), 3)

    def test_maxdiff(self):
        self.assertEqual(maxutil.maxdiff([5,23,1,0]), 22)
        self.assertEqual(maxutil.maxdiff([23,5,1,0]), 18)

if __name__ == '__main__':
    unittest.main(verbosity=2, exit=False)
```

Hin eiginlegu próf, restin er “overhead” (stundum kallaður “boilerplate” kóði)

NB Það er ekkert til fyrirstöðu að vera með fleiri en eitt test_föll fyrir hvert fall/klasa

# Prófhögun – praktísk atriði

- Byggja upp prófasafn
  - Æskilegt hlutfall prófkóða/kerfiskóða sé 2:1 eða 3:1
  - (í SQLite er hlutfallið 679:1 !!!)
- Keyra öll próf með einni skipun
  - tímafrek próf keyrð sjaldnar
- “Regression” prófanir
  - Kanna hvort nýr fídús framkalli villur í eldri kóða

Hlutfallið í V1 og V2 hjá okkur verður talsvert lægra

# Prófhögun – praktísk atriði

- Er réttlætanlegt að eyða umtalsverðum hluta í próf?
  - Ef kerfið virkar ekki, fáum við ekki borgað
  - Smíði prófunarkóða er einfaldlega hluti af verkinu
- Erfitt getur verið að skrifa kóða fyrir suma hluta kerfisins eins og t.d. grafísk notendaskil
  - Látum í hendur notenda/prófara (virknipróf)

# Dekkun

```
public class ComplexCode {
  public class UserCredentials {
    private String mToken;

    UserCredentials(String token) {
      mToken = token;
    }
    public String getUserToken() { return mToken; }
  }

  public UserCredentials login(String userId, String password) {
    if (userId == null) {
      throw new IllegalArgumentException("userId cannot be null");
    }
    if (password == null) {
      throw new IllegalArgumentException("password cannot be null");
    }
    User user = findUserByIdAndPassword(userId, password);
    if (user != null) {
      return new UserCredentials(generateToken(userId, password,
                 Calendar.getInstance().getTimeInMillis()));
    }
    throw new RuntimeException("Can't find user: " + userId);
  }

  private User findUserByIdAndPassword(String userId, String password) {
    // code here only used by class internals
  }

  private String generateToken(String userId, String password,
                               long nonce) {
    // utility method used only by this class
  }
}
```

You'd probably only need one test case for all of the UserCredential code, since there's no behavior, just data to access and set.

You'll need lots of tests for this method. One with a valid username and password...

...one where the userId is null...

...another where the password is null...

...and one where the username is valid but the password is wrong.

...one where the userId isn't null but isn't a valid ID...

And then there are these private methods. We can't get to these directly.

Til að prófa **allan** kóðann þarf að keyra allar greinar hans (oftast óraunhæft)

Stefna að 85% - 90% dekkun í praxís

Ýmis tól í boði sem mæla dekkun ("code coverage") - skoðum síðar

# Æfing

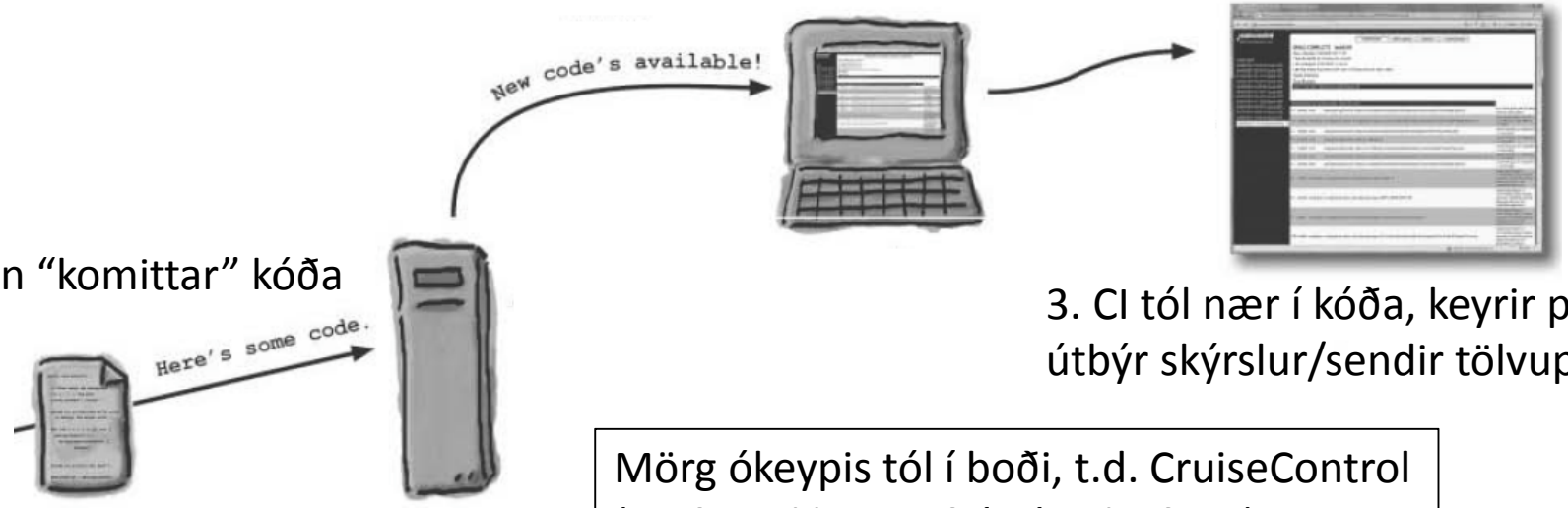
- [ ] Test the success cases ("happy paths").
- [ ] Test failure cases.
- [ ] Stage known input data if your system uses a database so you can test various backend problems.
- [ ] Read through the code you're testing.
- [ ] Review your requirements and user stories to see what the system is supposed to do.
- [ ] Test external failure conditions, like network outages or people shutting down their web browsers.
- [ ] Test for security problems like SQL injection or cross-site scripting (XSS).
- [ ] Simulate a disk-full condition.
- [ ] Simulate high-load scenarios.
- [ ] Use different operating systems, platforms, and browsers.

Merkið við þau atriði sem þið teljið að skipti máli til að prófin "dekki" kóðan sem best

# Samþætting prófana við útgáfustýringu

- Tengja prófanir við útgáfustýringu (cont. integr.)
  - Prófanir framkvæmdir sjálfvirkt þegar kóða er "kommittað"
  - Útbúa HTML skýrslu/senda tölvupóst ef e-ð bilar

2. Útgáfustýring lætur CI tól vita




1. Jón "komittar" kóða

3. CI tól nær í kóða, keyrir próf, útbýr skýrslur/sendir tölvupósta

Mörg ókeypis tól í boði, t.d. CruiseControl (verður ekki notað í námskeiðinu)

# Prófanir - eiginleikar

- Prófanir framkalla bilanir – leiðrétta þær ekki
- ... eru ekki trygging fyrir villulausu kerfi
  - Forritarar ofmeta dekkun prófa
  - Forritarar sýna frekar fram á að e-ð virki
- ... gefa mat á áreiðanleika kerfisins
- ... stýra endurbótum/leiðréttingum á kerfinu

# Villutölfræði

- “Bransinn”: 1-25 villur í hverjum 1000 línum af kóða (KLOC) sem fer úr húsi
  - Microsoft: 0.5 villur / KLOC
  - „Clean room“ aðferðir: 0.1 villur / KLOC
    - Formlegar aðferðir
    - Tölfræðileg gæðastjórnun
    - Tölfræðiúrvinnsla á prófniðurstöðum
      - Engin villa í 500 KLOC geimferjuhugbúnaði

Heimild: Code Complete e. Steve Connell

# What is your mission?

- Create unit tests to test your code.
- Make it so that you can run them all with one command or each of separately (doesn´t have to be a GUI, just something simple).
- Continue working on your program.
- In short: Follow the instructions in the assignment “Hópverkefni1c”

Good luck and if there are any questions feel free to e-mail me or use Piazza. I will be back on Monday the 10th of February.