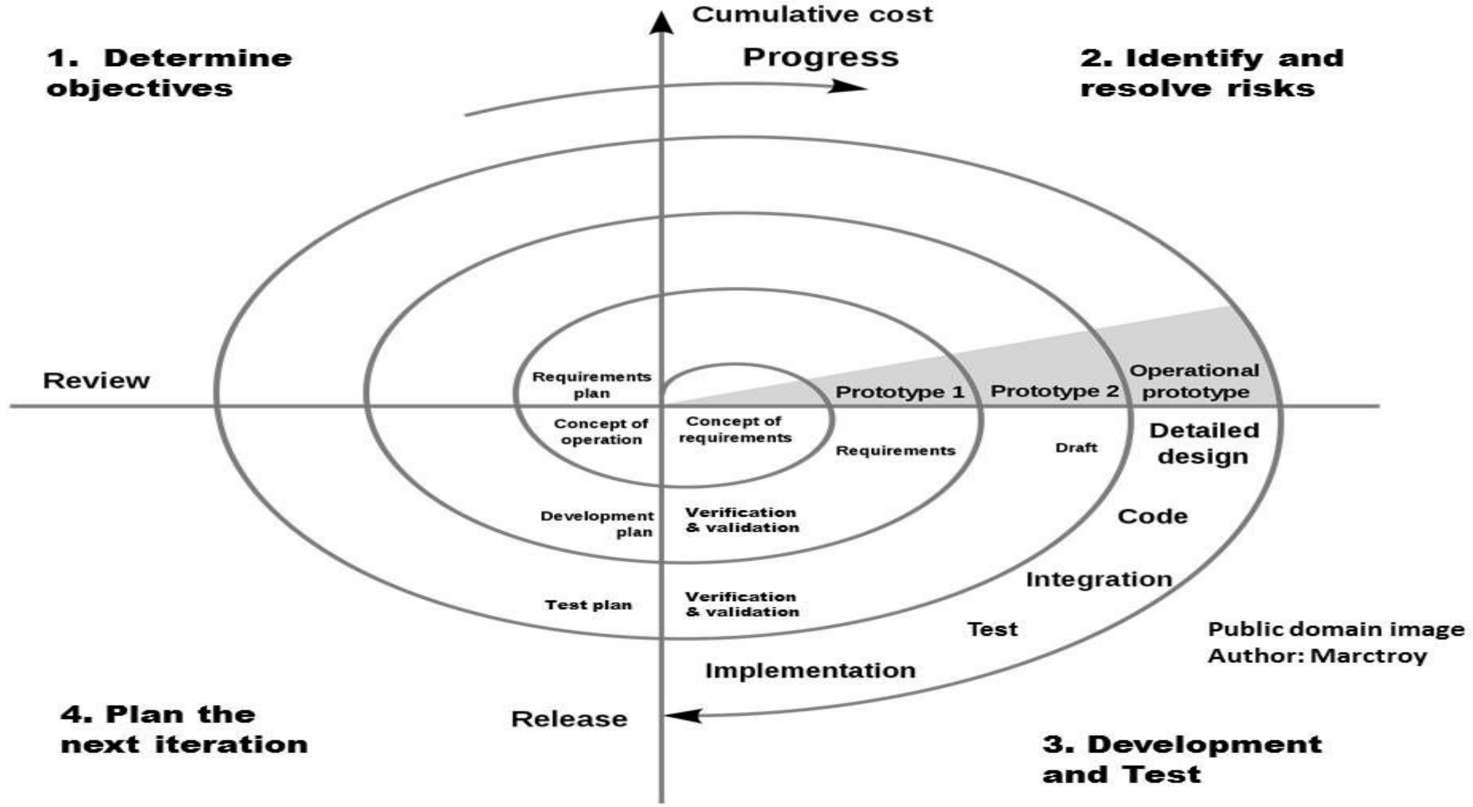
Software Engineering
Lecture 6: Implementation

Gregory S. DeLozier, Ph.D.

gdelozie@kent.edu

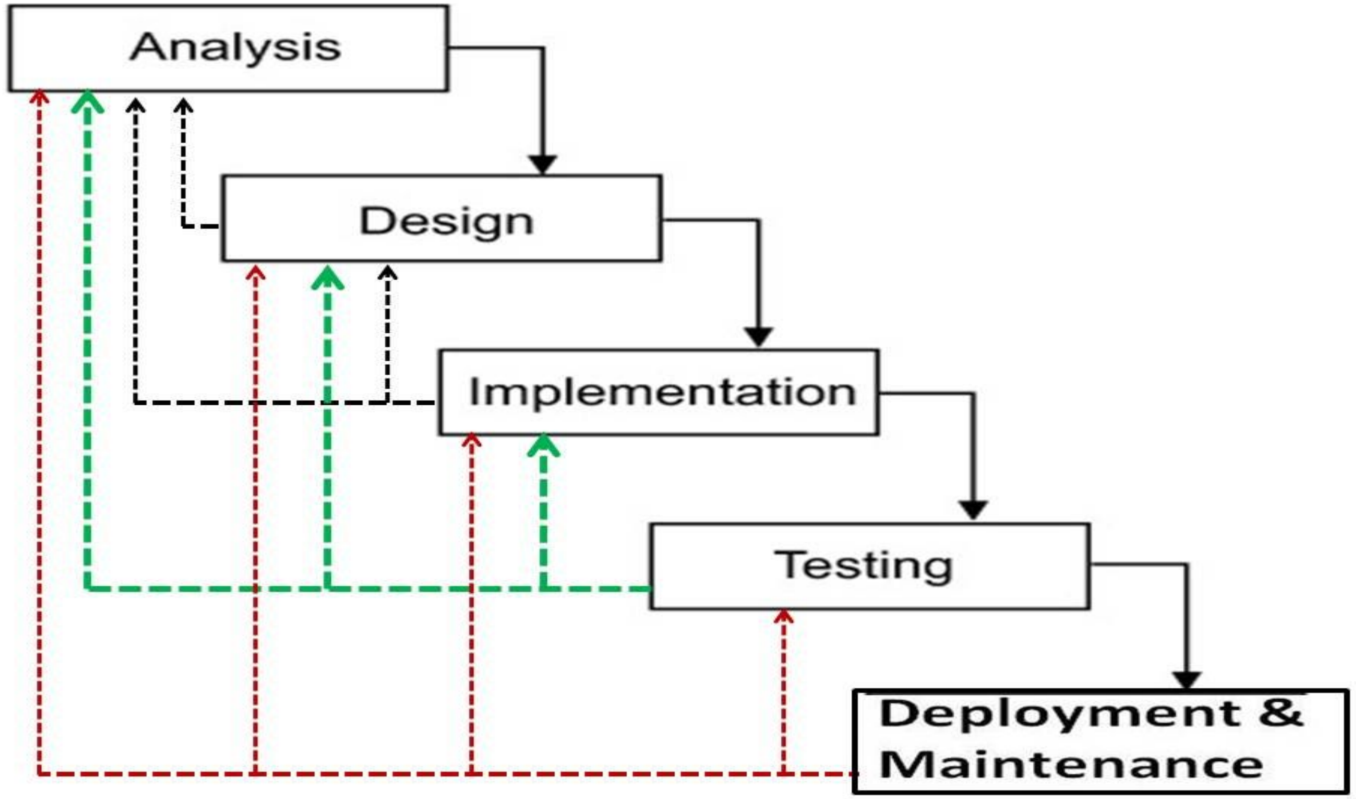
First, a little more about design

Boehm Spiral Model

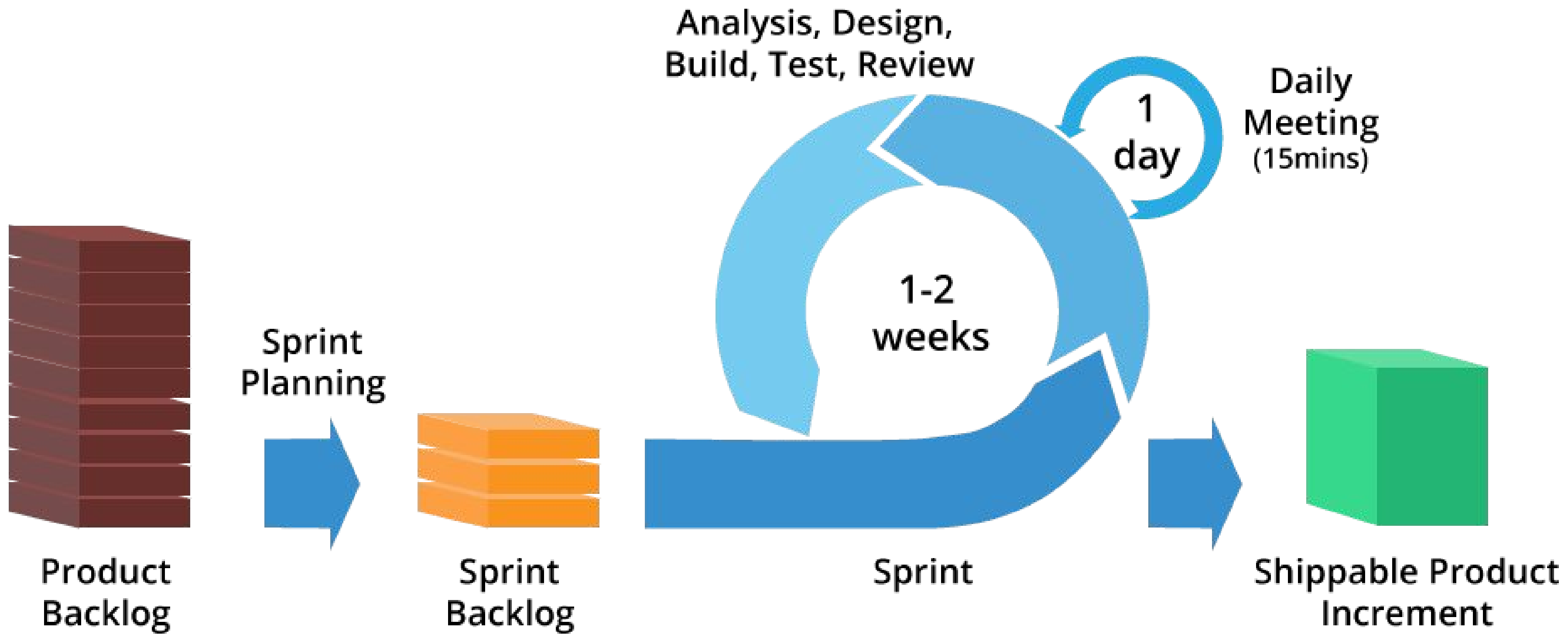


Public domain image
Author: Marctroy

Iterative Waterfall



Agile Software Development



In agile, design is continuous.

Remember this?

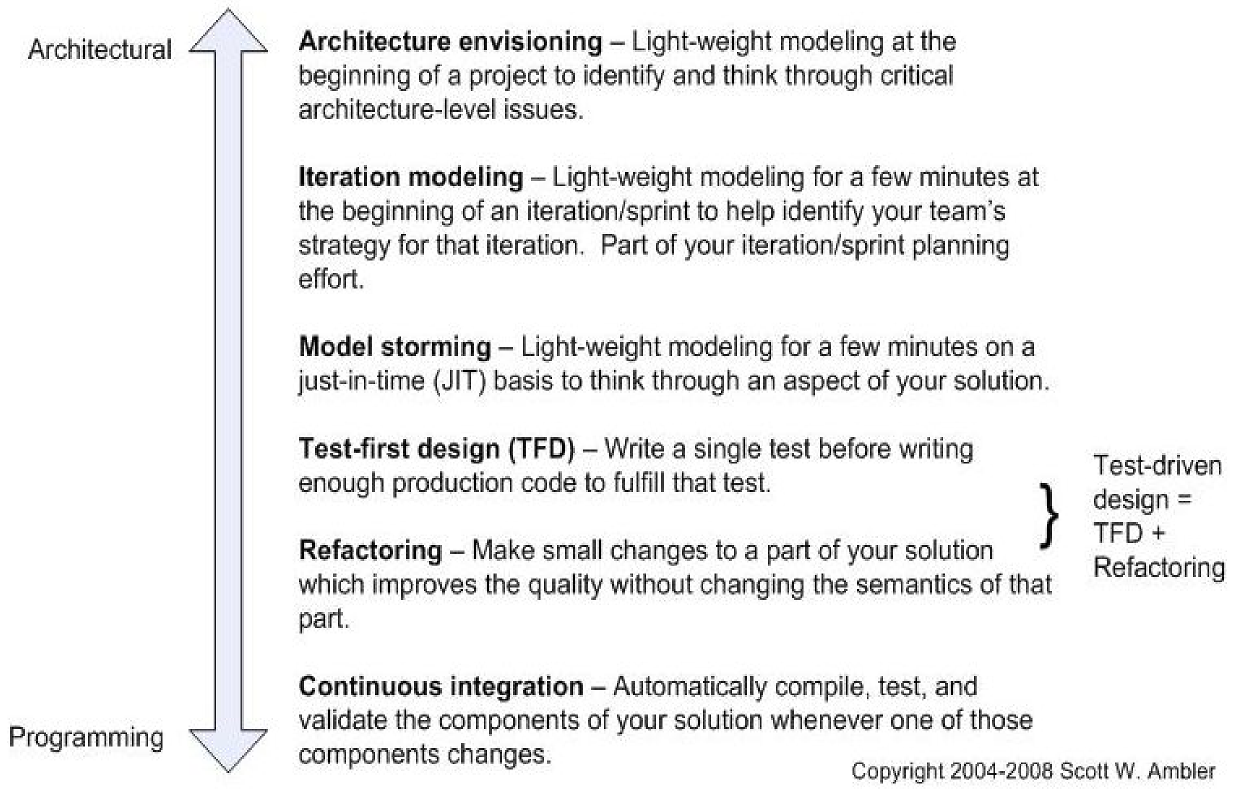
- New theory (Agile)
  - We will *never* ***(without a time machine)*** have a complete, perfect set of requirements
  - Nevertheless we have to write software, since the alternative is not to deliver value at all
  - We must create a way to write software in the presence of errors
  - ***If we can tolerate errors, endlessly eliminating them at great expense is a waste of time, and puts the delivery of any value at all at risk.***

So what are we designing?

Purpose of Design in Agile

- We are not designing an entire system
  - We don't know what that means
  -
- We want to represent what we have done so far
  - What have we built?
  -
- We want to design strategies for solving a problem
  - Communicate clearly with colleagues and stakeholders
  - Agree on a strategy
  - Refer to the strategy while coding
  -

Agile Design



Architecture envisioning – Light-weight modeling at the beginning of a project to identify and think through critical architecture-level issues.

Iteration modeling – Light-weight modeling for a few minutes at the beginning of an iteration/sprint to help identify your team's strategy for that iteration. Part of your iteration/sprint planning effort.

Model storming – Light-weight modeling for a few minutes on a just-in-time (JIT) basis to think through an aspect of your solution.

Test-first design (TFD) – Write a single test before writing enough production code to fulfill that test.

Refactoring – Make small changes to a part of your solution which improves the quality without changing the semantics of that part.

} Test-driven design = TFD + Refactoring

Continuous integration – Automatically compile, test, and validate the components of your solution whenever one of those components changes.

http://agilemodeling.com/essays/agileDesign.htm

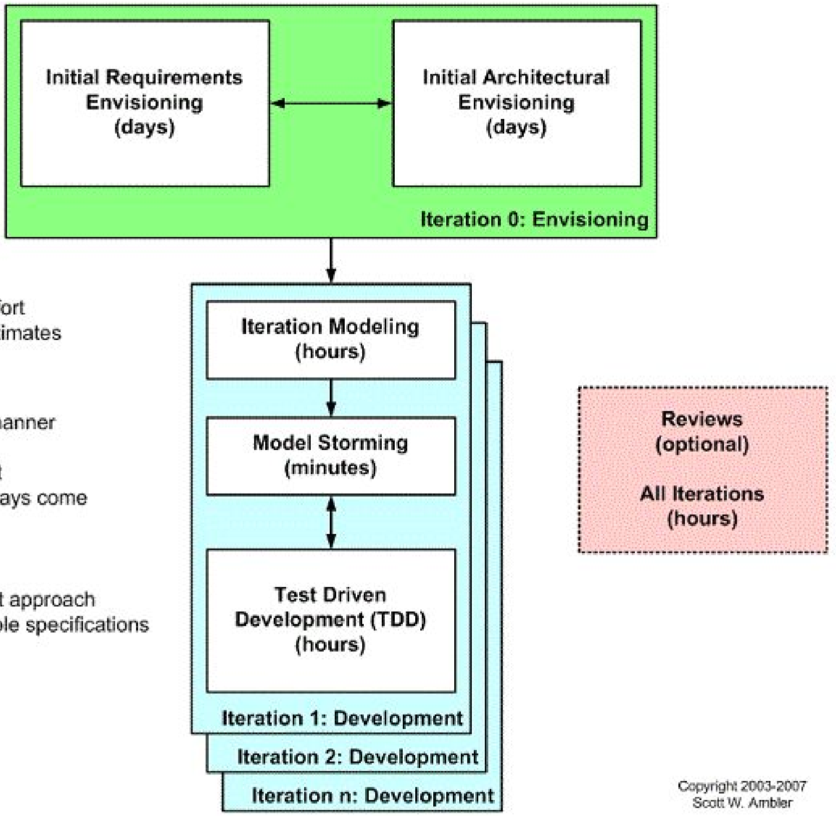
Agile Design

- Identify the high-level scope
- Identify initial "requirements stack"
- Identify an architectural vision

- Modeling is part of iteration planning effort
- Need to model enough to give good estimates
- Need to plan the work for the iteration

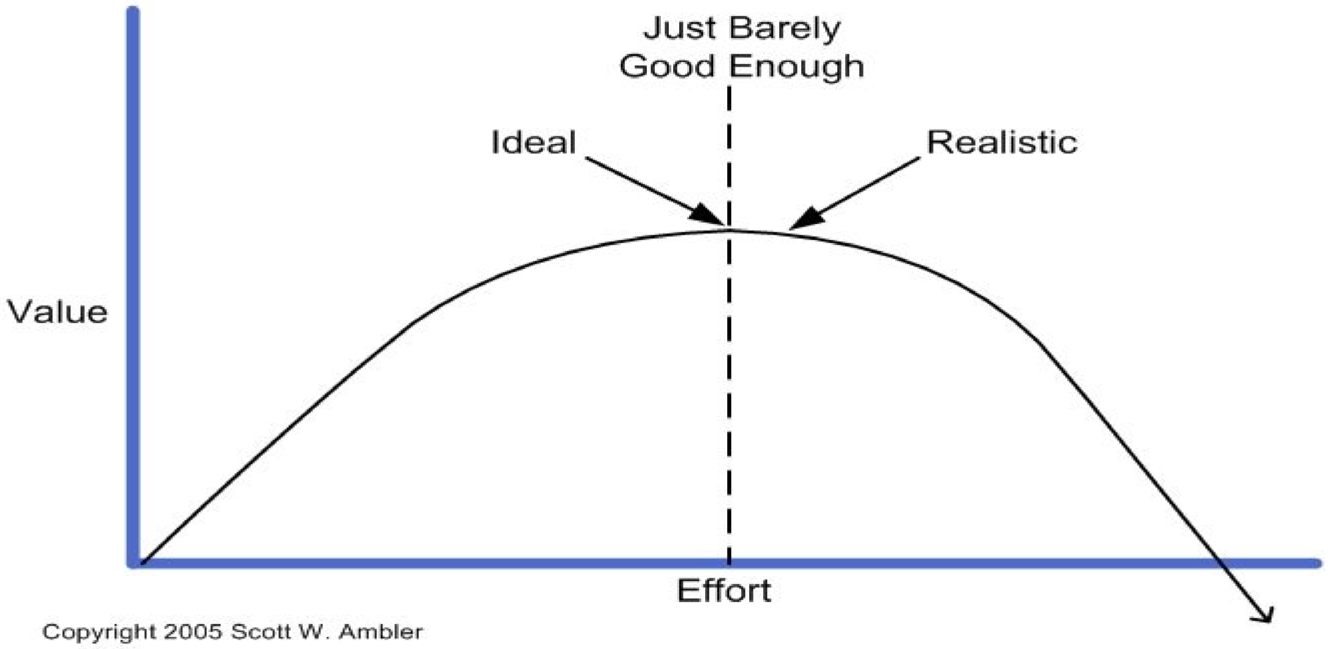
- Work through specific issues on a JIT manner
- Stakeholders actively participate
- Requirements evolve throughout project
- Model just enough for now, you can always come back later

- Develop working software via a test-first approach
- Details captured in the form of executable specifications


Initial Requirements Envisioning (days)
Initial Architectural Envisioning (days)
Iteration 0: Envisioning
Iteration Modeling (hours)
Model Storming (minutes)
Test Driven Development (TDD) (hours)
Iteration 1: Development
Iteration 2: Development
Iteration n: Development
Reviews (optional)
All Iterations (hours)
Copyright 2003-2007 Scott W. Ambler

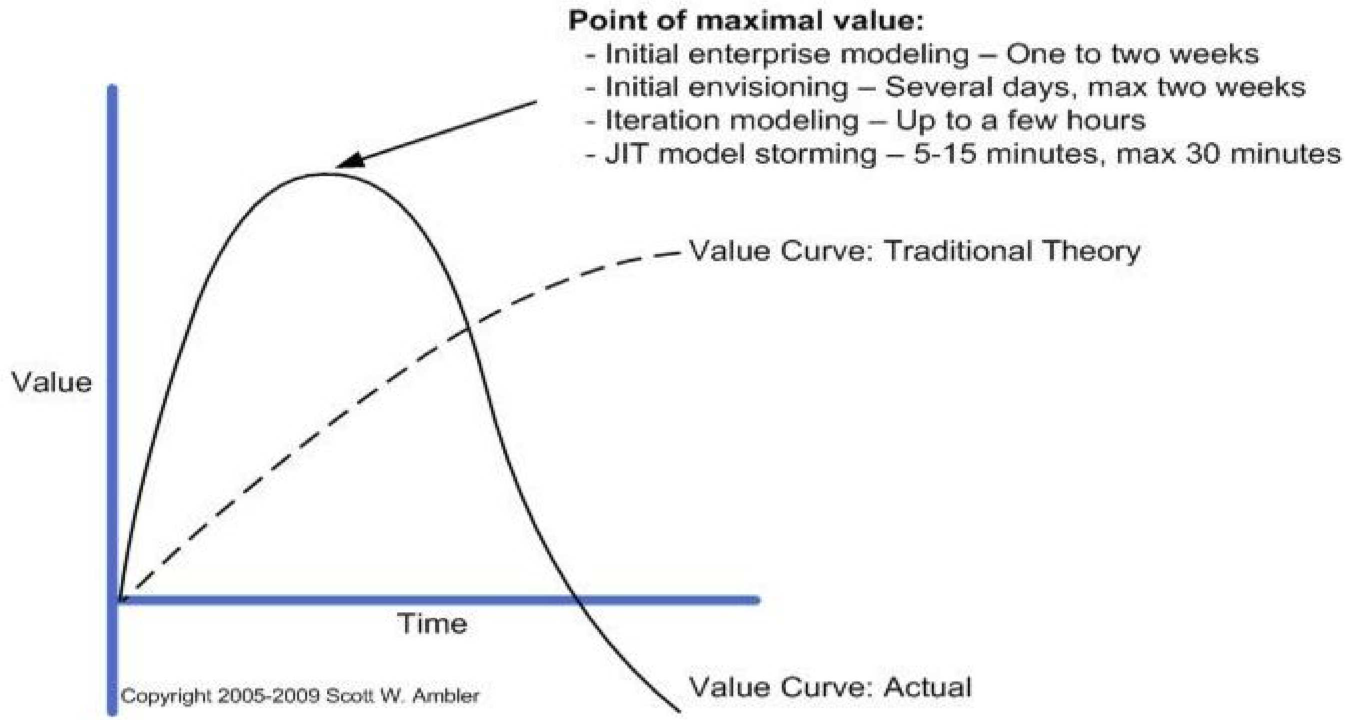
http://agilemodeling.com/essays/agileDesign.htm

Just Barely Good Enough



http://agilemodeling.com/essays/essays/barelyGoodEnough.html

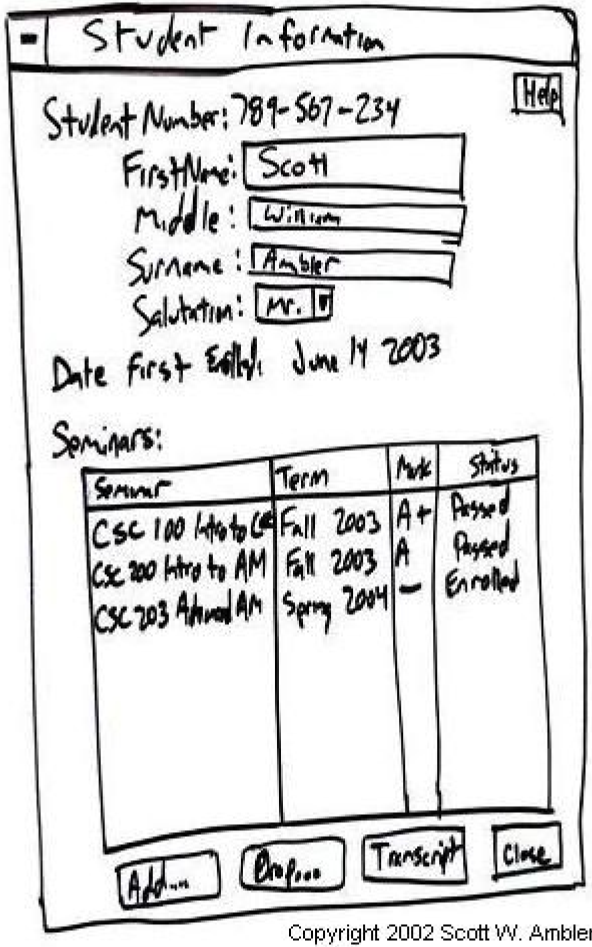
Just Barely Good Enough



http://agilemodeling.com/essays/essays/barelyGoodEnough.html

Model Storming

Fast creation of models…



Student Information

Help

Student Number: 789-567-234

FirstName: Scott

Middle: William

Surname: Ambler

Salutation: Mr.

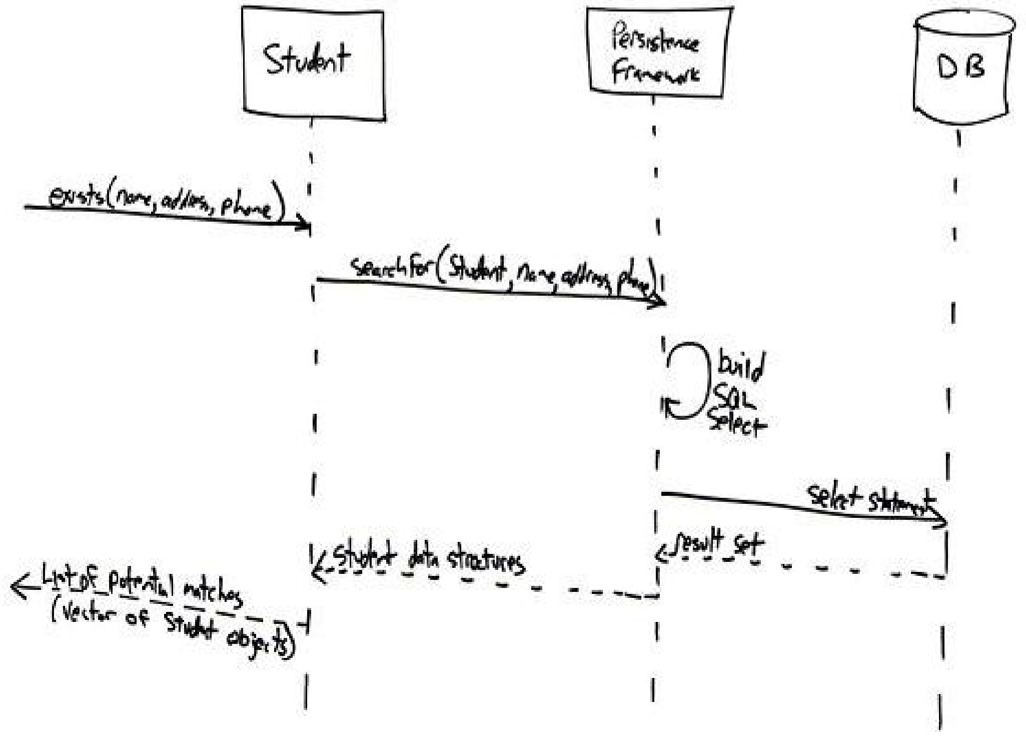
Date First Enrll. June 14 2003

Seminars:

Seminar	Term	Mark	Status
CSC 100 Intro to C	Fall 2003	A+	Passed
CSC 200 Intro to AM	Fall 2003	A	Passed
CSC 203 Advncd AM	Spring 2004	–	Enrolled

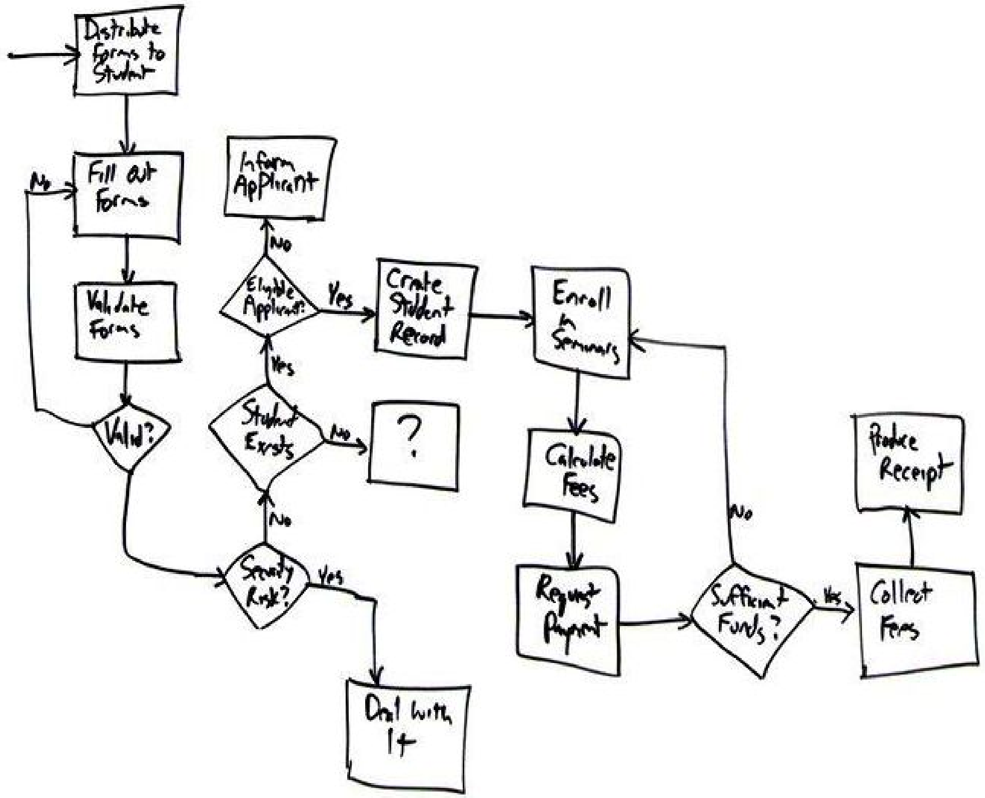
Model Storming

Enough to communicate


Student
Persistence Framework
DB
exists(name, address, phone)
searchFor(Student, name, address, phone)
build SQL Select
select statement
result set
Student data structures
List of potential matches (vector of Student objects)

Model Storming

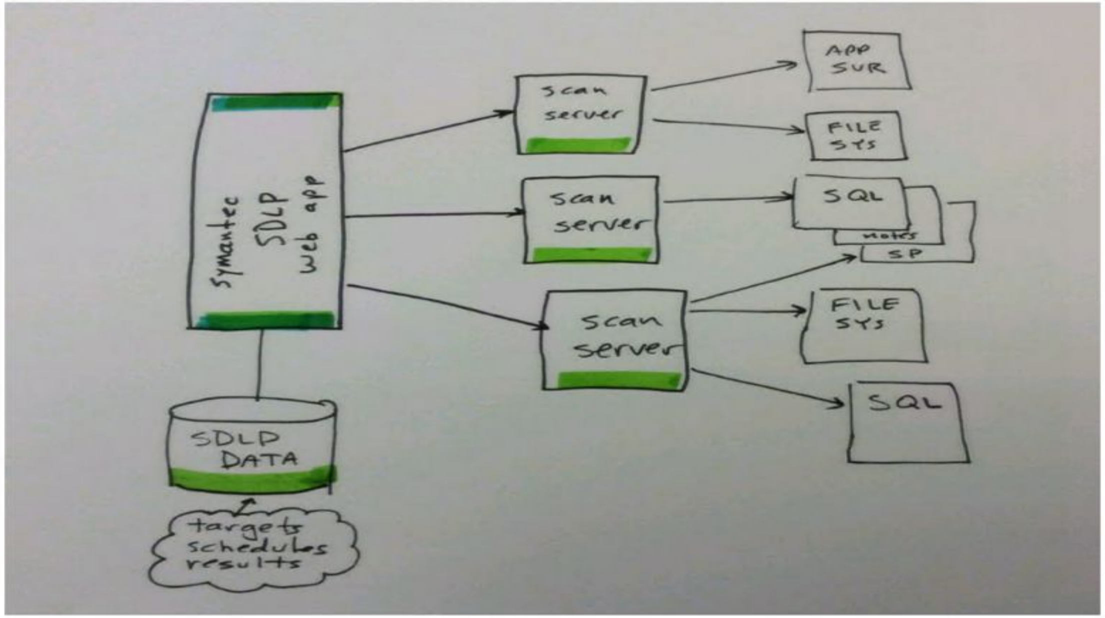
Used for reference...



Example model...

Current application

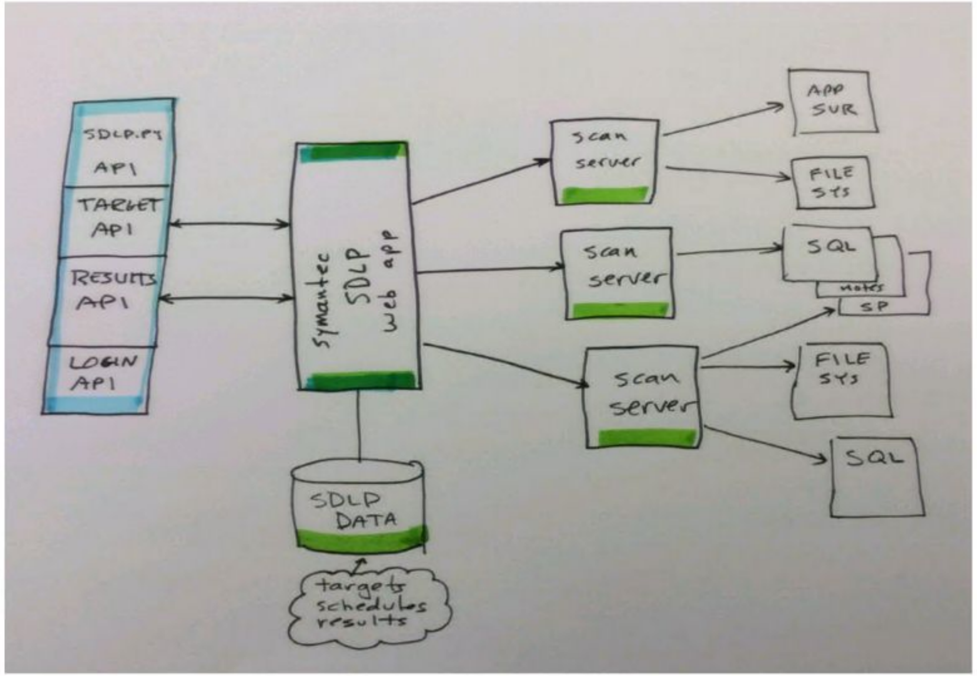
SDLP Application



Example model...

Proposed API module

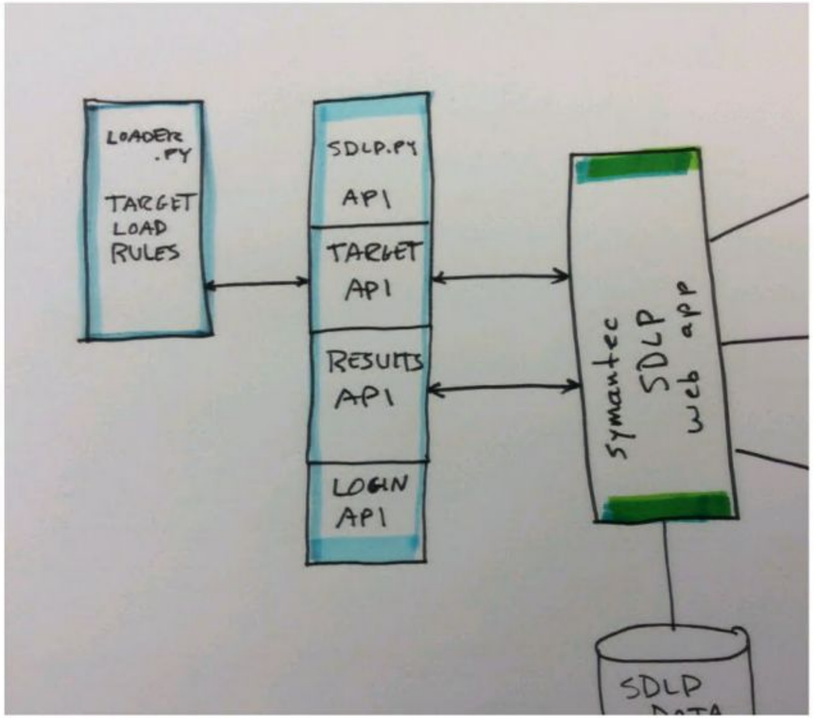
SDLP API Module



Example model...

Another proposed module

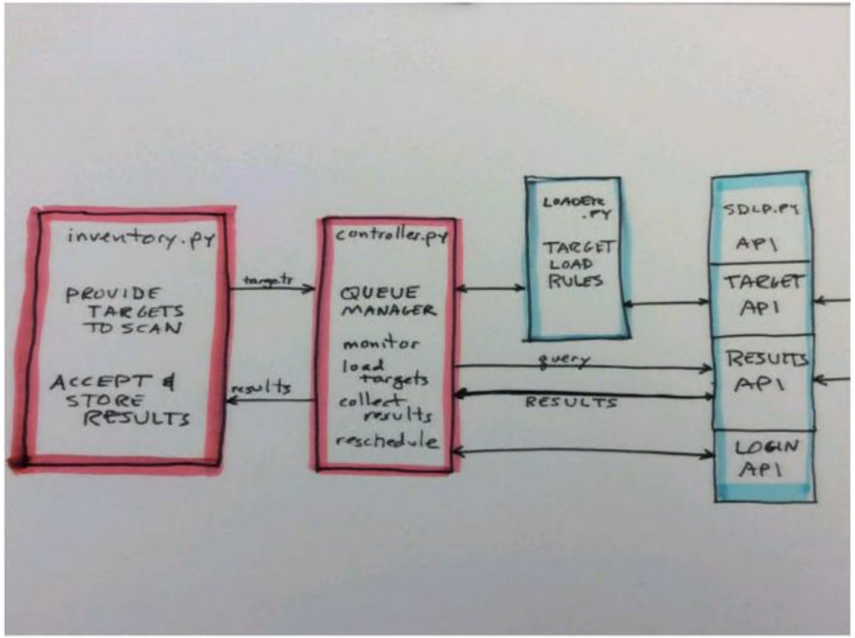
Target Loader



Example model...

Another proposed module

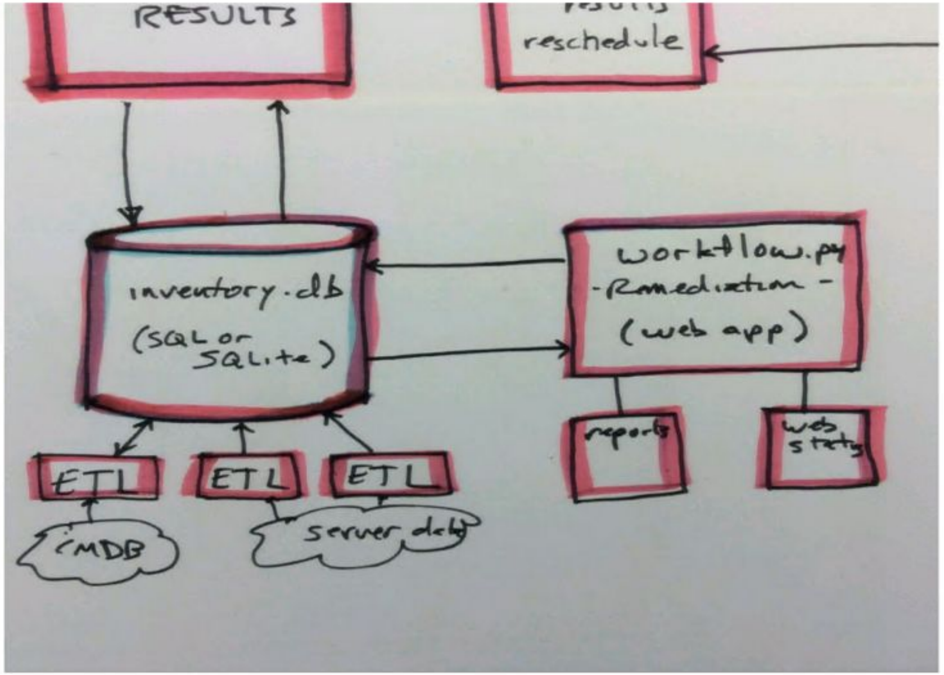
Inventory Manager



Example model...

Another proposed module

Alternate Storage



Goals of Iteration Design

- We know what problems we want to work on.
- We agree, in general, how we're going to solve them.
- We agree, in general, what a solution would look like.

Then what?

- *We agree, in general, what a solution would look like.*
- What does this mean we can do?

- We can describe how we would evaluate a solution.

Then what?

- *We agree, in general, what a solution would look like.*
- What does this mean we can do?

- We can describe how we would evaluate a solution.

- *We can define a test.*

What's a test?

- Here, a test is method to assess the correctness of a solution
    - (Yes, there are other definitions)
    -
- Preferably automated.
- Easily executed.
- Success means that a requirement has been met.

What's a test?

- Here, a test is method to assess the correctness of a solution
    - (Yes, there are other definitions)
    -
- Preferably automated.
- Easily executed.
- Success means that a requirement has been met.

- Written, based on a design, *before any production code has been written.*

Test Driven Development

Test Driven Development

- General process for creating a system

Do this:

1. Consider a requirement
2. Design a test
3. Have the test fail
4. Modify the system so it passes, along with all previous tests
5. Go to step #1

Test Driven Development

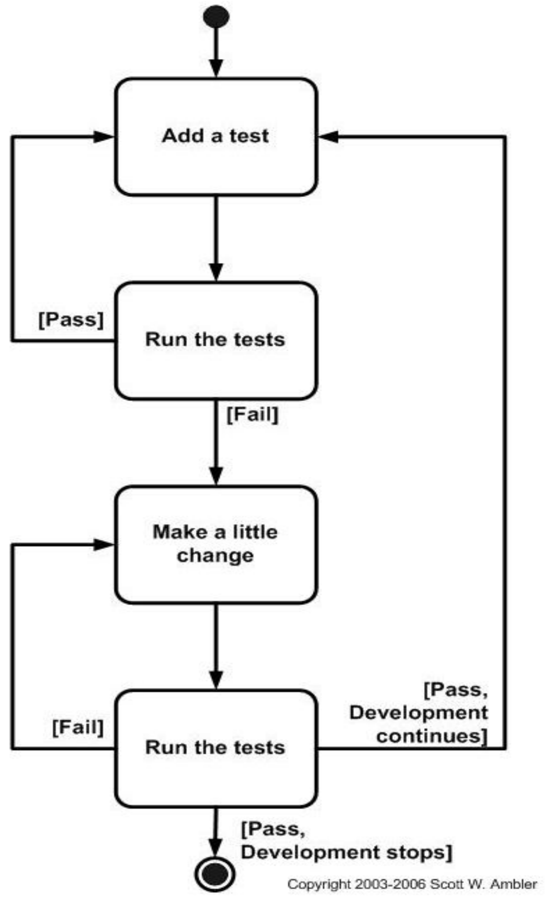
- General process for creating a system

Do this:

1. Consider a requirement
2. Design a test
3. Have the test fail
4. Modify the system so it passes, along with all previous tests
5. Go to step #1

YOU MAY NOT SKIP A STEP!

TDD Workflow



Add a test
[Pass]
Run the tests
[Fail]
Make a little change
[Fail]
Run the tests
[Pass, Development continues]
[Pass, Development stops]
Copyright 2003-2006 Scott W. Ambler

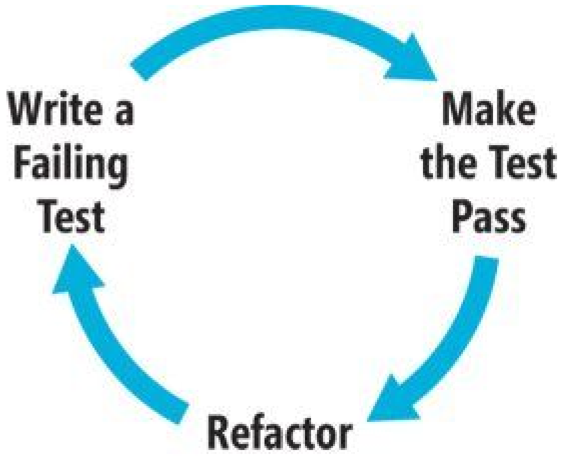
http://agiledata.org/essays/tdd.html

Beck's Rules

Kent Beck, who popularized TDD in eXtreme Programming (XP), defines two simple rules for TDD:

- ***You should write new business code only when an automated test has failed.***
- ***You should eliminate any duplication that you find.***

TDD Workflow

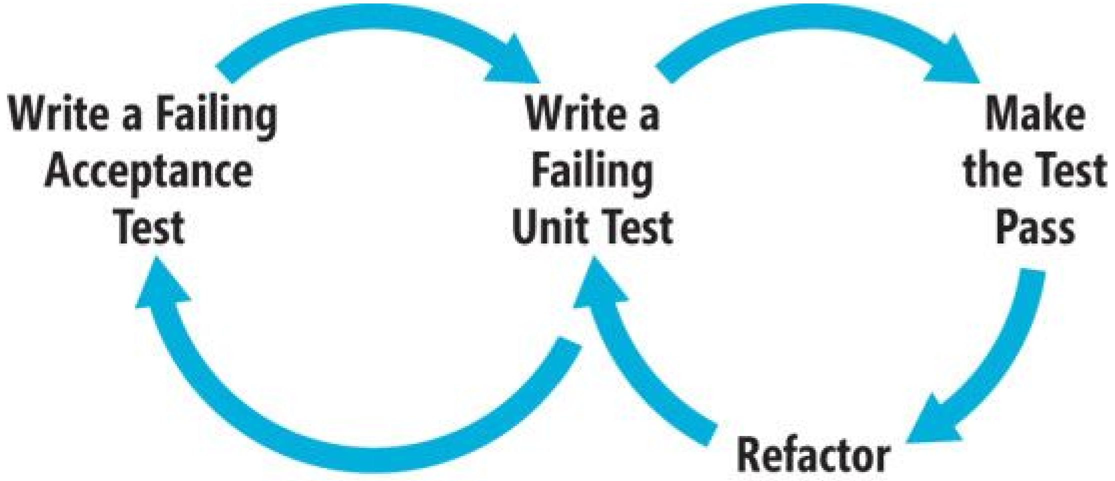


Beck's Rules - Consequences

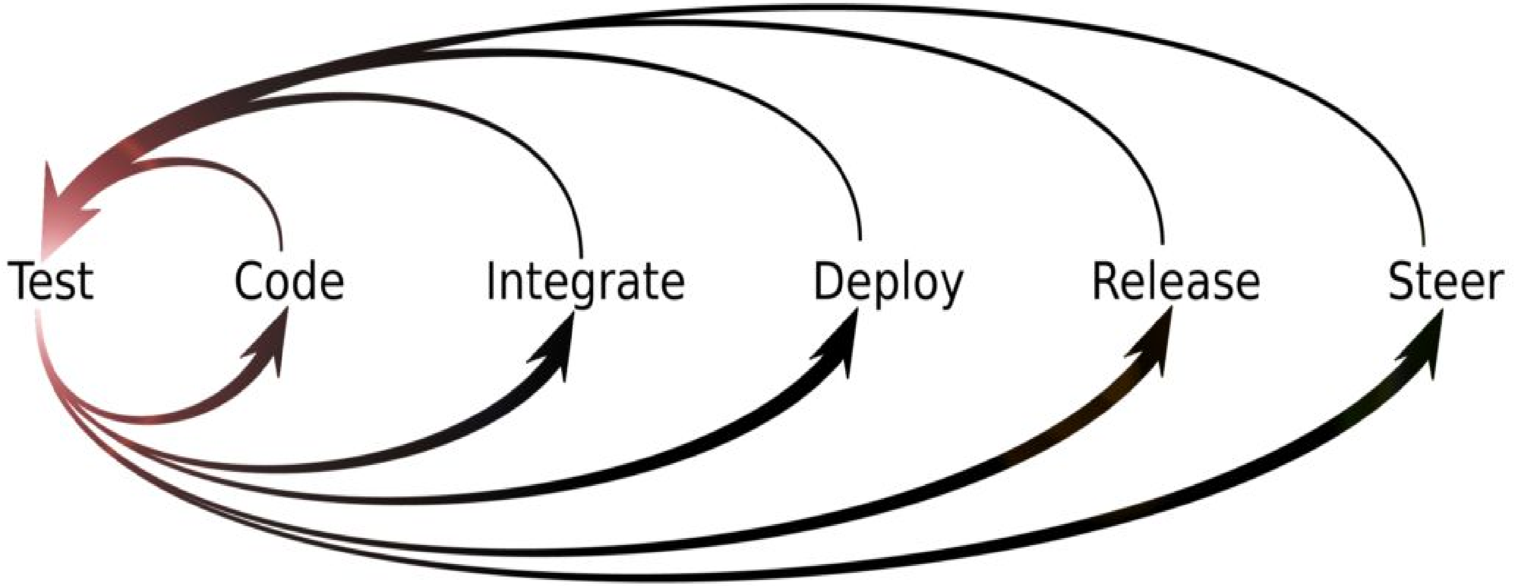
- You develop organically, with the running code providing feedback between decisions.
- You write your own tests because you can't wait 20 times per day for someone else to write them for you.
- Your development environment must provide rapid response to small changes (e.g you need a fast compiler and regression test suite).
- Your designs must consist of highly cohesive, loosely coupled components (e. g. your design is highly normalized) to make testing easier (this also makes evolution and maintenance of your system easier too).

- See more at: http://agiledata.org/essays/tdd.html#sthash.yH1KD2tL.dpuf

TDD Workflow - Acceptance

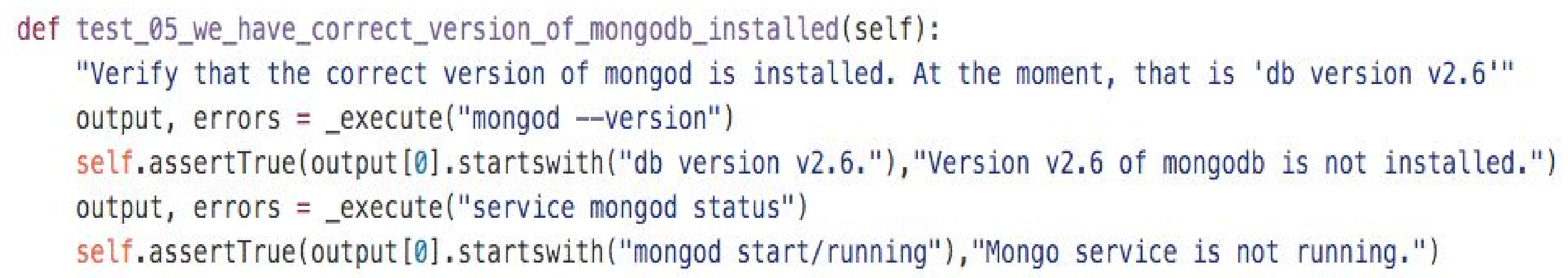


TDD Workflow - Global



Testing Deployment Example

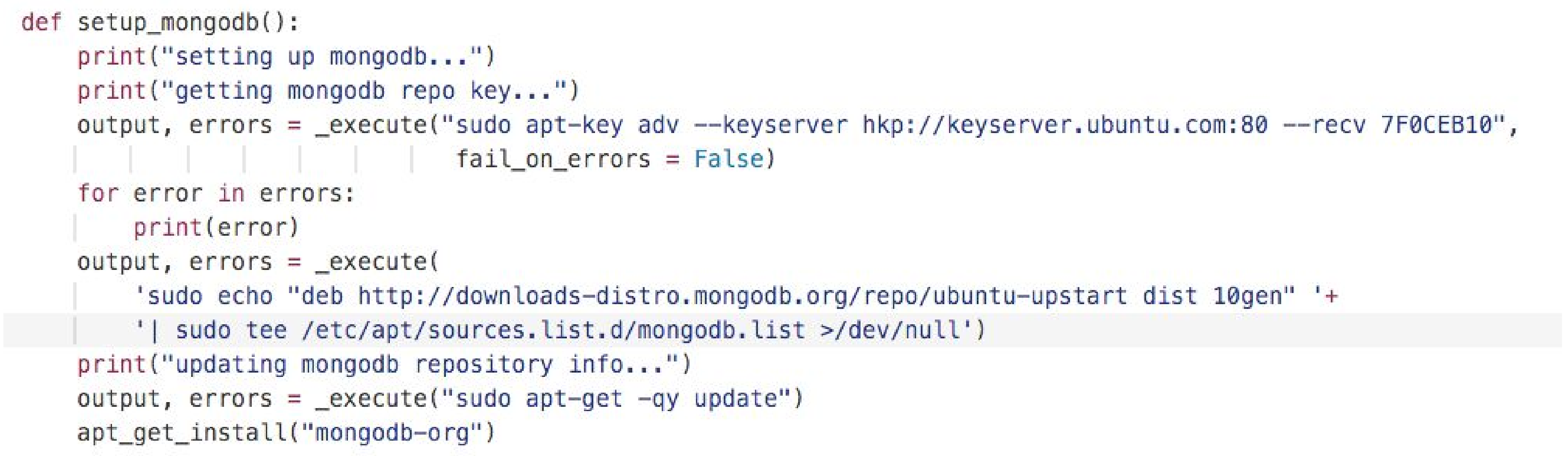
Verify that Mongo database is installed:



```
def test_05_we_have_correct_version_of_mongodb_installed(self):
    "Verify that the correct version of mongod is installed. At the moment, that is 'db version v2.6'"
    output, errors = _execute("mongod --version")
    self.assertTrue(output[0].startswith("db version v2.6."),"Version v2.6 of mongodb is not installed.")
    output, errors = _execute("service mongod status")
    self.assertTrue(output[0].startswith("mongod start/running"),"Mongo service is not running.")
```

Deployment Code

This makes the test pass:



```python
def setup_mongodb():
    print("setting up mongodb...")
    print("getting mongodb repo key...")
    output, errors = _execute("sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10",
                               fail_on_errors = False)
    for error in errors:
        print(error)
    output, errors = _execute(
        'sudo echo "deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen" '+
        '| sudo tee /etc/apt/sources.list.d/mongodb.list >/dev/null')
    print("updating mongodb repository info...")
    output, errors = _execute("sudo apt-get -qy update")
    apt_get_install("mongodb-org")
```

Test Frameworks

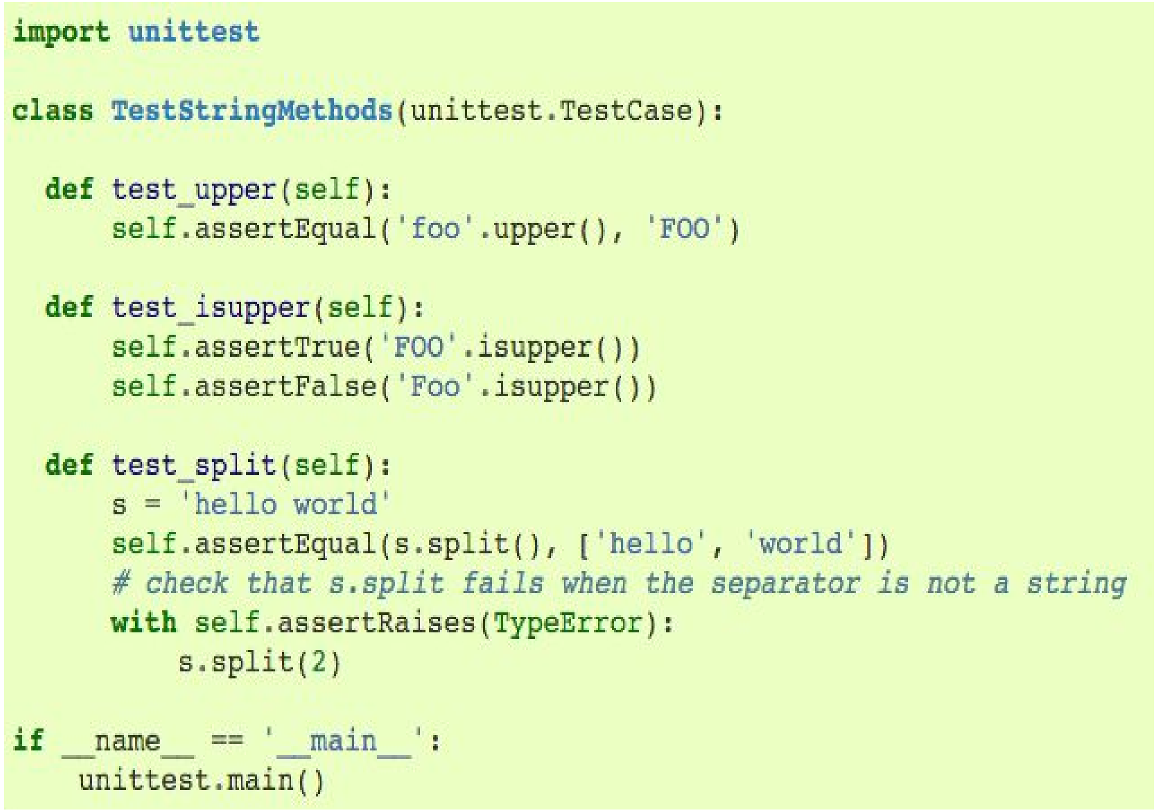
- Test frameworks organize tests and test code
- XUnit was a very early test framework
- PyUnit is a more recent framework based on the same model
  - http://pyunit.sourceforge.net/
  - https://docs.python.org/3/library/unittest.html
- Other languages have similar frameworks
  - Java - http://junit.org/
  - C++ - https://sourceforge.net/projects/cppunit/
  - Ruby - http://ruby-doc.org/stdlib-1.8.7/libdoc/test/unit/rdoc/Test/Unit.html
  - ...and so on...

PyUnit

From the docs…

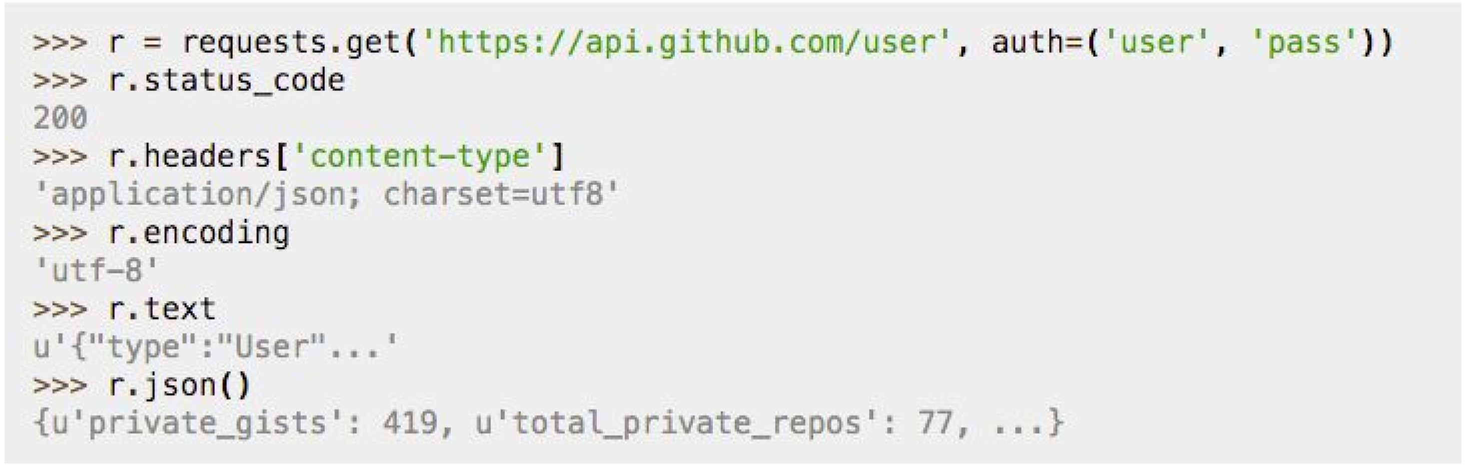
Standard module.

Easy to learn and use.



```
import unittest

class TestStringMethods(unittest.TestCase):

  def test_upper(self):
      self.assertEqual('foo'.upper(), 'FOO')

  def test_isupper(self):
      self.assertTrue('FOO'.isupper())
      self.assertFalse('Foo'.isupper())

  def test_split(self):
      s = 'hello world'
      self.assertEqual(s.split(), ['hello', 'world'])
      # check that s.split fails when the separator is not a string
      with self.assertRaises(TypeError):
          s.split(2)

if __name__ == '__main__':
    unittest.main()
```

Requests Module

- Allows Python to send internet requests.
  - http://docs.python-requests.org/en/master/
- Requests can be used in tests.
- Requests can get HTML or JSON results.
  - 

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
u'{"type":"User"...'
>>> r.json()
{u'private_gists': 419, u'total_private_repos': 77, ...}
```

Demo Time

Homework

- Add the feature of unary negation to the expressions module.
  - Write some tests
  - Add some code
  - Show the tests passing
- Identify some features at these web sites:
  - amazon.com
  - kent.edu
  - google.com
- Write some tests to verify those features.
- Write some tests to specify future features for your web site.