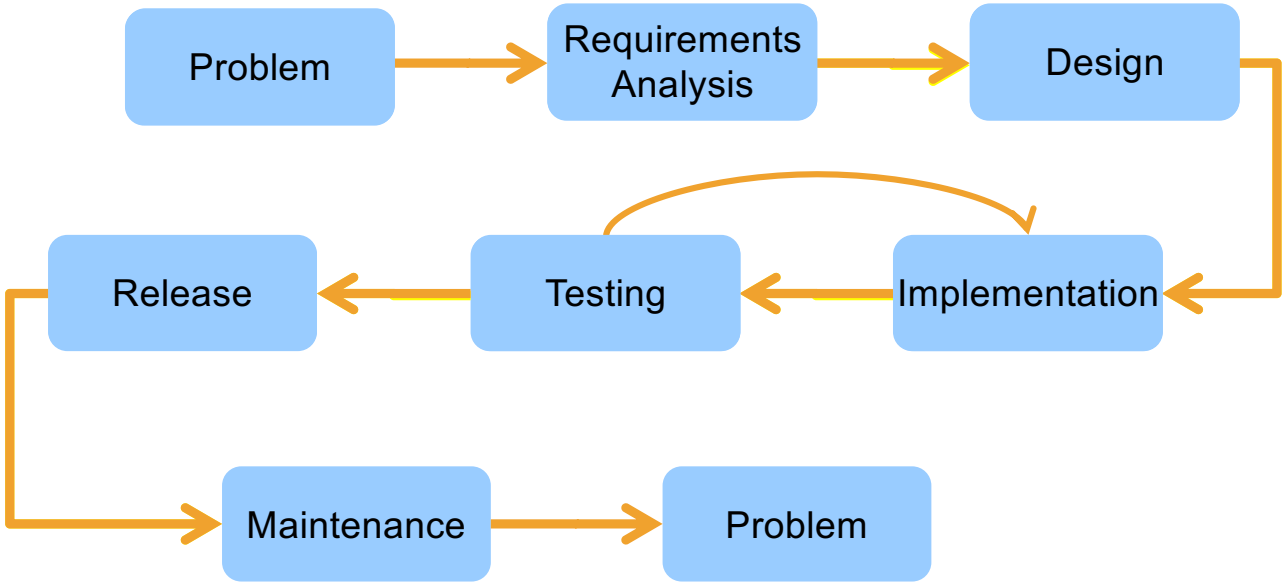
CS 4320 / 7320 Software Engineering

Construction *and Testing*

What is the SDLC?
Where does Construction fit?



Construction Fundamentals

Psst…here's a secret.

A lot of this also occurs in design.

Construction Fundamentals

1. Minimize Complexity
2. Anticipate Change
3. Construct for Verification
4. Reuse
5. Standards

Construction Fundamentals: Minimize Complexity

- Make it readable
- Keep it simple
  Don't be clever
- Follow standards
- Use modular design

For Python, here is a good reference:
http://docs.python-guide.org/en/latest/writing/style/#code-style

Construction Fundamentals: Anticipate Change

- Modularity and Cohesion
- Interfaces
- Organization: put things that change together, together
- "Loose coupling" vs. Appropriate coupling

  Things that change often: fewer dependencies

  Things that rarely change: more dependencies ok

Construction Fundamentals: Construct for Verification

Build it so that faults are easily found and diagnosed during development, testing and operations.

- Follow standards to support code reviews & unit testing
- Organize code to support automated unit testing
- Restrict use of complex, hard-to-understand code

Construction Fundamentals: Reuse

Libraries, modules, components, source code, COTS….

- Construction FOR reuse
- Construction WITH reuse

- When does it make sense? What are trade offs?
  Vendor support, code dependencies, …

Construction Fundamentals: Standards

Well thought out standards help achieve non-functional objectives such as efficiency, quality, and cost.

- Communication methods (document or message formats)
- Programming language
- Coding standards (naming conventions, layout, indentation…)
- Platforms (interface standards)
- Tools (notations like Crow's foot, UML)

Secure Coding

- Input validation
- Output Encoding
- Authentication and Password Management
- Access Control
- Error Handling and Logging

See.. OWASP Secure Coding Practices Quick Reference Guide

https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

Error Handling

Anticipate errors

A few error handling techniques:

- Return neutral value
- Logging a warning message
- Return an error code
- Shutting down

*What makes sense in **this** situation?*

Exception Handling

Detect and gracefully process errors & exceptional events

- Implemented with a Try-Catch block of code
- May process the exception or return to the calling block
- Include all relevant info in the message (*but not to the user)
- No empty catch blocks (!!!!)
- Know your library exceptions
- Standardize exception handling

Python Exception Handling

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except OSError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

Test-First Programming

Unit test cases are written before writing the code

Advantages:

- Detect and correct defects earlier
- Forces careful consideration of requirements and design

Reference for Python Test Driven Development:
http://www.obeythetestinggoat.com/pages/book.html