Design & Testing: Part Yin and Yang

Computer Networks (15-441) Fall 2007
Daniel Spangenberg
Design: Outside the Box

- Two types of applications
  - Data-centric
    - What type of data and what does it look like?
    - Where do we store it
  - Protocol-centric
    - How do I talk to the world?
- Mostly about interfaces
Design: Inside the Box

- How do I access my data?
  - Interfaces!

- How do I store my data?
  - Implementation!

- Interfaces alleviate implementation pain
  - Wrap a good interface around an implementation
Lessons to Be Learned
Lesson One

• Don’t Repeat Yourself (DRY principle)
• How much copy and paste do you use?
  • Put it in a separate function!
• Design a small set of orthogonal interfaces to your modules
  • Adhere to them!
Lessons to Be Learned
Lesson Two

- It's OK for code to be shy
- It's preferred! (unlike for you)
- Shy code...
  - Doesn’t expose itself in public
  - Doesn’t stare at others’ privates
  - Surely doesn’t touch others’ privates!
  - Doesn’t have a whole lot of friends
Lessons to Be Learned
Shyness (Example One)

Which is better?

```
int send_msg_to_user(int user_id,
    user_sock,
    char* msg);
```

```
int send_msg_to_user(struct user_t*,
    char* msg);
```
Lessons to Be Learned

Shyness (Example Two)

```c
int send_to_user(char *uname, char* msg) {
    struct user *u;
    for (u = userlist; u; u = u->next) {
        if (!strcmp(u->uname, uname))
            ...

    Consider factoring this into a separate function:
    void find_user(struct user *u, char* uname)
```
Lessons to Be Learned
Lesson Three

• Keep it simple
• No premature optimization
  • Even in the optimization contest, optimization generally not too important...
• Throw out unnecessary features / requests
  • Not so important in 441...
Lessons to Be Learned
Lesson Four

- Be Consistent
- Naming
- Style
- Doesn’t matter what you choose, but choose *something* (no memcpy vs bcopy)
- Decide and document memory ownership
- Make it explicit in interfaces!
A Note: Error Handling

• Detect at the low level
  • malloc() returns null!

• Report at high level
  • Not a good idea to abort()
  • Print an error message and attempt to continue...
The Testing Mindset

- Think like the adversary (like security!)
- Your goal is breaking the code
- If you can’t, you probably haven’t tried hard enough
- This ensures that in five days you won’t spend five hours tracking down that bug...
- Think about your code
  - Then write tests to exercise it
  - Hit the corners!
Testability

- Test at all levels!
- From the user’s perspective
- From the code’s perspective
- Bugs are easiest to find in a local scope
- Unit test things if possible
- Make granular integration tests!
Testing Methods

• Unit
• Integration
• Regression
• Performance
Unit Tests

- Tests specific features in a vacuum
- Generally reserved for internals...
  - Hash tables...
  - Linked lists...
  - Read/write buffers...
- Always in the language of the product
  - Use CUnit for 441 projects
Integration Tests

- “Do multiple pieces fit together?”
- Tests a major user-facing feature
  - Does JOIN work?
  - Does PRIVMSG work with nine targets?
- Generally utilizes a tool outside the product
- We will provide you with some samples
Blackbox vs Whitebox

- **Blackbox**
  - Implementation-agnostic test cases
  - Typical end-user use cases

- **Whitebox**
  - Implementation-aware test cases
  - Mainly for the corner cases/implementation details
Regression Tests

- Shows how a commit affects the product
- General idea:
  - Record what tests passed at rev N
  - See what tests pass at rev N+1
  - Look at the difference
- If it wasn’t broken before you regressed
More Regression

- New features may uncover *latent* bugs
- Write new test cases when found!
- Make sure the test does what you think it does
Performance Testing

- General principle: *Kick the shit out of it*
- Two approaches:
  - Isolate subsystems for analysis
  - Test the gamut for the big picture
- Regression testing is valid for performance too!
- Make sure you don’t make performance worse at commit
Want more?

• Joel Spolsky will give you some info (if you can take him!)  www.joelonsoftware.com

• There is the ACE framework  
http://www.cs.wustl.edu/~schmidt/patterns-ace.html

• Presentation on patterns for network apps  
http://www.ncst.ernet.in/education/apgdst/npfac/slides/NP-Patterns.ppt