CSC 406 Syllabus

Revised 28 March

Rethinking the Overreaction

After some dispassionate reflection, I have concluded that my proposal to rewrite the entire system in Python was an immature, capricious, ill-considered, unscholarly, and unsound overreaction to a poorly functioning product. In plain English -- it was a dumb idea.

I've spent a few days experimenting with Python/QT and Java/Swing as possible alternatives to the current Java/GWT implementation. Neither of the alternative platforms is a panacea for the problems with the current implementation.

It is clear now that the approach most likely to produce a usable product is to focus clearly on fixing the problems in the existing code, fully test the entire system, and release as soon as possible for client review. We will discuss this approach on the first day of class.

Accepting the Rethink

Per consensus on the first day of class, we have agreed to indeed proceed with the current implementation. Accordingly, the following items need to be accomplished before we release to the clients:

1. the immediate bugs must be fixed, and we must all redo the delayed user-level acceptance testing for each department
2. every model method must have a JUnit test, and it must pass it
3. every GWT screen must have a Selenium test, and it must pass it
4. the end-user acceptance tests must pass for all departments to which a release is made
5. all user-critical Jira issues must be resolved

During 405, we were building a prototype that was not fully tested, which we could have released as such to understanding clients. Having not produced a prototype suitable for release, we have now entered the deployment phase, where any released product must be fully tested.

Fixing Immediate Bugs

There are some acceptance test results from 405. Mine in particular itemize specific bugs that must be fixed immediately, so we can have another round of customer-level acceptance testing. See

```
http://scheduler.csc.calpoly.edu/releases/alpha/testing/acceptance/CSC/
    finals-week-report.html
```

In particular, all of the MAJOR ERRORs must be fixed.

The immediate point of user-level acceptance testing is to ensure that the functionality we now have in place is adequate to allow real department schedules to be generated. That is, we need to make sure that there are no significant user-level features that make it difficult or impossible to produce department schedules, in particular for Fall 2011 and Winter 2012. This level of human acceptance testing will go on in parallel with the code-level JUnit and Selenium testing.

Administrative Process

The following is the administrative process we will follow, as discussed during the 405 final exam period, and refined during the first class meeting of 406:

- Daily on-time attendance is mandatory, except for

- *o* James and Tyler H pre-approved class conflicts
- *o* Jake for the first four weeks, who has a pre-approved absence request
- *o* occasional critical absences, for which *advance notification* is required
- *o* medical family emergencies

• We will work in two-person teams, as described below under the heading "Team Pairings and Meta Pairings"

• We will conduct weekly code reviews, as described below under the heading "Code Review Process" and "Code Review Criteria".

On Mondays and Fridays, up to 10 minutes at the beginning of class will be devoted to a Scrum-style stand-up discussion of issues of potential relevance to the entire class. If no such issues exist, then no discussion will be necessary.

After the first two days of class, all class and lab time will be devoted to *working on the project*. Think of this as a job where you come to the office six hours per week to do work.

Team Pairings and Meta Pairings

The following is the tentative team organization we agreed to during the 405 final exam period:

Team	Project Focus	Code Review Pairing
Salome, Adam	Algorithm	Schedule Views
Matt, Tyler Y	Schedule Views	Algorithm
Kaylene, Jonathan	Resource Tables, Back End	Front End
Evan, Tyler H	Resource Tables, Front End	Back End
Jake, Carsten	Instructor Web App	Top-Level
James, Jordan	Top-Level UI & File Storage	Instructor Web App

The 'Code Review Paring' column indicates how the teams are paired for the code review process. Teams so paired will review each other's code.

Code Review Process

The following is the code review process we tentatively agreed to during the 405 final exam period:

• Code reviews are scheduled weekly.

• Commit by 1PM Monday.

• Review by 11AM Wednesday.

• Reviewers and reviewees communicate as necessary outside of class, including meetings to go over review results.

The code to review is any file for which one or more svn diffs exist since the commit for the previous week's review. There will be a script that will perform the diffs and list the files that need to be reviewed in a particular week.

Code Review Criteria

The following are the code review criteria we tentatively agreed to during the 405 final exam period, and refined during the first week of 406:

- follows standards posted on 406 web page
- it has good Javadoc
- it has a sensible design
- it has sensible execution, including passing tests

System and Acceptance Test Procedures

The following acceptance test procedure was determined during Wednesday of Week 2 of class:

- James releases to department sites once per week only at 12:55PM Mondays.
- Fisher and two other 406 members will perform department-specific acceptance testing on deployment sites by Friday at 11AM.
- Particularly noteworthy acceptance test results will be reported to the Friday stand-up
- It is critical for the first three weeks of acceptance testing to identify any missing essential program features, i.e., features that make it difficult or impossible to generate specific department schedules.

Process Details to Be Discussed and Resolved

The following matters will be discussed and resolved in the first two class meetings of Spring quarter:

- revised repo structure; *as of Wednesday 28 March a revised repo is in place, thanks to Evan*
- confirmation of team pairings; *as of Wednesday 28 March the team structure in the preceding table has been confirmed by all*
- precise schedule for code reviews; *as of Wednesday 28 March the code review process and criteria in the preceding tables have been confirmed by all*
- exact definition of regression testing and repository commit process, including review of test/commit script, which does the following;
    - *o* run regression test suite, which includes coverage tests
    - *o* svn commit if all tests pass

  As of Wednesday 28 March the regression test policy is to redirect JUnit stderr output to a log file, with an empty log file indicating that regression tests have all passed; this policy has been confirmed by a majority of class members, and will be fully confiredm on Monday 2 April

Schedule

Based on dicussions during the second class meeting, the following is the large-grain confirmed project schedule:

Week	Task
1	Wednesday: Project planning is complete
2	Monday: Some 405 bugs fixed, minimum 25% tested
3	Monday: All 405 bugs fixed, minimum 50% tested
4	Monday: Minimum 75% tested
5	Monday: 100% tested
	Wednesday: Contacted interested clients
	Friday: Released to clients
6	Add enhancements, respond to client feedback
7	Add enhancements, respond to client feedback
8	Second client release, work on documentation
9	Respond to client feedback, work on documentation
10	Work on documentation, final client release

The percentage measurements for testing is based on code coverage. I.e., "25% minimum tested" means "at least 25% code coverage, as measured by the Cobertura coverage tool".

Grading

Assume a 100 point scale for scoring, with the usual breakdown of 90% and above being an "A" grade, 80-89% a "B", 70-79% a "C", 60-69% a "D", and less than 60% and "F". Everyone in the class starts out with a 100% score. Points are permanently deducted for the following class activities:

Task	**Deduction**
Class Attendance	-4 points for each unexcused absence; coming to class more that 10 minutes late is considered an absence
Commit of Test-Failing Code	-10 points for each instance; although we will have a script that will help avoid this problem, it will likely still be possible to commit failing code, which is why there is a significant penalty for it
Assigned Critical Task	-4 points for each uncompleted critical task; these tasks will be defined weekly by Kaylene and Jira posted, as was done last quarter