CS585 Software Verification and Validation
Course Project
Winter 2015

Goals

As the key part of the course work for CS585, you will complete a small course project related with software testing, verification or validation techniques. The main goals of the project are to: 1) let you learn and experience how to independently work on a research topic; 2) build a prototype implementation for the project so that it could potentially converted to a publication or your final Master thesis/project; 3) create a solid project record in your resume; 4) potentially motivate you to create a startup around the idea.

Timeline and Requirements

You may select the topic from the list of given project ideas, or create your own idea. The timeline and requirements about the projects are:

Week 1-2	Choose the research topic
End of Week 2	Decide the final research topic
End of Week 3	Confirm the project scope and requirements with the instructor
Week 4-9	Work on project
Week 6-9	Research presentation[1]
Week 10	Project in-class demo
End of Week 10	Code and documents checking in

[1]During Week 6-9, you will give a short in-class presentation (10 mins) to introduce the research area related with your project, talk about the existing works that have been done, and describe the plan of your own research project. The specific presentation schedule will be decided according to your project topic.

Given the limited time to work on this course project within a single quarter, you are not expected to finish a complete high-quality and publishable research project within the 4-5 weeks, but you need to demonstrate your comprehensive understanding about the related area, and more importantly, the implemented research idea at least with a proof-of-concept prototype.

Research Topic List

1. Automated Test Generation

    1.1. From HTTP API Specification to Integration Tests

    Based on the HTTP APIs specified in the WebController.java (CS580 project), generate the integration test method code automatically.

    Keywords: Web Service, Spring, Code Generation, Code Framework

    1.2. From HTML Page to Selenium Tests

    Based on the widgets used in a HTML page, automatically generate the Selenium (http://www.seleniumhq.org/) test scripts for the page.

    Keywords: GUI Test, Web UI, Selenium, Code Generation

    1.3. From Android UI XML Specification to Unit Tests

    Based on the Android UI XML specification, automatically figure out the widgets to test and generate the unit test code (or code skeleton).

    Keywords: GUI Test, Mobile UI, Android, Code Generation

    1.4. From iOS UI Specification to Unit Tests

    Based on the iOS UI specification, automatically figure out the widgets to test and generate the unit test code (or code skeleton).

    Keywords: GUI Test, Mobile UI, iOS, Code Generation

2. Automated Grading

    2.1. Use Continuous Integration to Support Auto-Grading

    Build a continuous integration pipeline (Jenkins), so that students can check in their code, which will trigger the code to be checked out in Jenkins, followed by running all the grading tests and reporting the scores automatically.

    Keywords: Grading, Education, Continuous Integration, Jenkins

    2.2. Java Annotation-based Automated Grade Reporting

    You can build a new set of Java annotations (e.g., @Test, @Grade), so that students can put these annotation in their assignment classes. Whenever they run the annotated methods, the status and results will be automatically sent to server. This could be used to grade, as well as tracking students' working progress.

    Keywords: Grading, Education, Annotation, Reporting

    2.3. GitHub Repo Clone Detection

    A number of code clone tools are available to detect the code duplications. You can apply these tools to automatically compare two (or a set of) code repositories, and check the code duplication. This could be useful to check code plagiarism.

    Keywords: Grading, Education, Code Clone, Code Duplication

3. Static Code Analysis

3.1. Design Pattern Verification

By loading the source code, automatically detect or verify if certain design patterns have been applied correctly. You need to perform the static code analysis to check the hierarchy, naming, associations, etc.

Keywords: Design Pattern, Static Analysis

4. Mining-based Testing

4.1. Tester Evaluation

Given a GitHub user, automatically analyze all the public projects source code and check the number of test classes, methods being tested, or test coverage, and report whether the user is a good tester or not.

Keywords: Data mining, Big Data, Test Measurement, Test Coverage

4.2. Test Start Finder

In a given GitHub project (large-scale project with a number of contributors), based on the code checkin history related with testing code, find out the best tester in this project.

Keywords: Data mining, Big Data, Test Measurement, Test Coverage

5. Mobile Testing

5.1. Android Test Bed using Genymotion

Android device testing is hard due to the large number of manufactures and versions. You can build a test bed that runs different versions of the Android, so that developers can send their APK to all the versions to run and test. You can use the virtual Android Genymotion to create the different versions to simulate different devices.

Keywords: Android Testing, Genymotino, Virtual Simulator, Test Bed

5.2. Mobile GUI User Behavior Checker

It is essential to understand how users use your app in your mobile UI. You can develop a code framework (or code annotations) to enable developers to insert certain markers in the mobile app to automatically track user's behavior (e.g., user clicks on button A, then button B, but did not click on button C).

Keywords: Mobile GUI, User Interaction, Automated Monitoring

6. Web Testing

6.1. Web GUI User Behavior Checker

Similar to 5.2, but being applied to web UI, such as HTML.

Keywords: Web GUI, User Interaction, Automated Monitoring

6.2. Browser Compatibility Checker

Checking the compatibility of your web UI is important. One way you can do is to automatically load the web page in different types of browsers, and capture a screenshot of the page shown in the browser. By comparing if the screenshot images are the same, we can tell if the UI is displayed in the same way.

Keywords: Web UI, Compatibility Test, Automated Reporting, Screenshot Capture

6.3. Web UI Reporting

Every time you check in a HTML code to the repo, we automatically run your web page and capture a screenshot and send it to you, so that you can see the current status of your page. This could be connected with 6.2

Keywords: Web UI, Automated Reporting, Screenshot Capture

7. Performance Testing/Validation

7.1. Generic Monitor/Alarm Web Service

Build a distributed web service which can automatically check the HTTP API status of a certain target web service continuously, so that the alarm can be triggered whenever the target service stops working.

Keywords: Web Service, Health Check, Automated Reporting/Alarm

7.2. Annotation-Based HTTP API Latency Metrics and Auto Alarm

In CS580, we used CloudWatch to add performance metrics to track the processing time of each HTTP API. We can build a Java annotation that developers can apply in their HTTP API methods, so that the latency, processing time information can be reported automatically.

Keywords: Web Service, Metrics/Alarms, Performance Monitoring

7.3. QoS-Based Unit Testing

Traditional unit test is based on assertions of values and status, rather than performance (e.g., Quality-Of-Service metrics such as latency, throughput, error rate). You can build a new library or framework that can enable developers to assert QoS attributes and verify the performance automatically.

Keywords: Web Service, QoS, Metrics, Automated Reporting

8. Software Security

8.1. DDoS Attack Service

Build a simple DDoS service to simulate small-scale attack

Keywords: DDoS, Security

8.2. DDoS Identification & Recovery

Add mechanisms in your web service to automatically detect DDoS attacks, and try to recover from it.

Keywords: DDoS, Security

9. Mutation-based Testing

9.1. Automated Test Quality Evaluation

You can build a tool to automatically check in some bad code to the repo and trigger the tests, in order to check whether the existing unit tests can catch the errors.

Keywords: Mutation-based Testing, Test Quality

10. GitHub-based Testing

10.1. Automated Developer Feedback on Test

For each Git push, based on the code coverage, the number of test methods/classes, give developers immediate feedback on the sufficiency and quality of the test.

Keywords: GitHub, Test Feedback, Test Quality and Measurement

11. Model-based Testing

11.1. Test Generation from System Models

Generate test cases from UML models.

Keywords: Models, UML, Code Generation, Test Generation

11.2. High-level Test Model Specification

Build a high-level modeling language to describe the test, so that the low-level test cases and code can be generated.

Keywords: Model-Driven, Test Generation, Code Generation

Please let me know if you have any questions about these topics. Also, feel free to think about and come up with your own ideas!