COMP 587: Software Verification and Validation
Fall 2018

Instructor: Kyle Dewey (kyle.dewey@csun.edu)
Course Web Page: https://kyledewey.github.io/comp587-fall18
Piazza Web Page: http://piazza.com/csun/fall2018/comp587
Office: JD 4427, Extension 4316 (not yet connected)

Course Description (From the Catalog)
An-in depth study of verification and validation strategies and techniques as they apply to the development of quality software. Topics include test planning and management, testing tools, technical reviews, formal methods and the economics of software testing. The relationship of testing to other quality assurance activities as well as the integration of verification and validation into the overall software development process are also discussed.

Learning Objectives
Successful students will be able to:
- Formulate an actionable test plan with ACC (Attributes, Components, Capabilities)
- Test an existing software product using basic industry-standard techniques using their test plan
- Understand and apply advanced testing techniques to existing software
- Use model checking techniques to build confidence in the correctness of software specifications
- Use verification techniques to prove small programs correct

Course Motivation (or a Relevant Rant)
As a society, we have chosen to surround ourselves with software, and to make software essential to daily life. Software controls everything from a car's fuel injector to critical infrastructure. Software makes it easy to build new things cheaply, and to rapidly adapt to change.

However, as a society, we have counterintuitively decided that it's generally *ok* for software to fail, especially if failures are rare. It's ok when unanticipated inputs cause an app to crash. It's ok if memory leaks require me to power cycle a router. It's ok if a race condition causes the traffic lights at an intersection to go to flashing reds. It's ok if a flaw in my encryption scheme allows others to read my email. It's ok if an unchecked array access leaks my social security number. It's ok if a missing bounds check destroys my life's work. It's ok if a lack of input sanitization wastes hundreds of millions of dollars. It's ok if an unexpected scenario kills people.

This sounds ridiculous, and it is ridiculous, but this is reality. Each of the above examples is rooted in real software flaws with real impacts. None of these examples have resulted in widespread outcry for better software. If you think any of the above are decidedly *not ok*, and you want to do something about it, this class is for you.

Course Emphasis and Structure

This course covers a broad spectrum of activities, tools, and technologies which are intended to produce better software. Some of these activities are commonplace and widely employed in industry, whereas others fill particular niches and may be more academic in nature. This course is roughly split into three different pieces:

1. Basic approaches (weeks 1-4). This focuses on approaches which are seen and used almost everywhere in industry. These approaches tend to favor ease of use and overall simplicity over raw power. While these won't find all the problems, they still tend to work well for applications where minor failures are acceptable. These should be used for almost any software project.
2. Advanced approaches (weeks 5-9). This focuses on approaches which can automatically generate large and diverse sets of tests. These require nonstandard background to use, and an overall different way of thinking. Most of these techniques have been used in various industry settings, but with applications where failures are less acceptable (e.g., web browsers, device drivers).
3. Verification-based approaches (weeks 10-15). This focuses on approaches which can find specification-level problems, and can even *prove* that an implementation is devoid of bugs. These can be unwieldy to use, and require potentially years of experience to master. In practice, techniques like these are employed rarely, and only for applications which are intolerant of any failures (e.g., flight control software, space probe subsystems). Much of this area is academic in nature.

Textbook

No textbooks are required. That said, the following books may be of interest to you:

- How Google Tests Software, James Whittaker, Jason Arbon, Jeff Carollo - discusses ACC and a number of basic industrial testing approaches
- Fuzzing: Brute Force Vulnerability Discovery, Michael Sutton, Adam Greene, Pedram Amini - discusses some of the advanced approaches; fairly out of date, but many of the concepts are the same
- Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers, Leslie Lamport - introduction to formalizing and reasoning about software specifications with TLA+, by the creator of TLA+
- Certified Programming with Dependent Types, Adam Chlipala - an in-depth look at dependent type theory and Coq, suitable for proving software implementations correct

Grading

Your grade is based on the following components:

Assignments	30%
Project Proposal / ACC	5%
Project Pull Requests	12%
Project Code Reviews	12%

Project Implementation	30%
Project Report	6%
Project Presentation	5%

Not all assignments will be weighted evenly, nor will you always be given the same amount of time for assignments. Exactly which assignments are assigned depends on how the class progresses. In general, assignments will be submitted through Canvas (https://canvas.csun.edu/). In the event that there is a problem with Canvas, you may email your assignment to me (kyle.dewey@csun.edu), though this should be considered a last resort.

Assignments and Project
Classes are backed by assignments (30% of your grade), which cover V&V in breadth. The bulk of your grade (70%), however, is based on a project, which will allow you to cover a subset of V&V in depth on real software. This breakdown reflects what I want you to take away from this class: adequate high-level knowledge of a variety of approaches, and in-depth knowledge on an area(s) of your choosing.

Details of the project follow, roughly in chronological order:

1. System under test (SUT) selection. You will choose a software project which you want to apply V&V techniques to, hereafter referred to as a SUT. The SUT should be sufficiently large (anything under 1,000 lines of code is almost assuredly too small), and can be in any state of development. Your grade is based **solely** on the V&V aspect, so if you select an incomplete SUT (e.g., it is still in the planning stages), the actual SUT implementation is secondary. You are encouraged to select software you have written in the past, but you must be ok with giving everyone in the class access to this code. You may work in pairs, but more will be expected of such projects.

2. Project proposal (5%). You will propose exactly what you plan to do with the SUT, from a V&V perspective. For **all projects**, this **must** include aspects related to details covered in the first four weeks of class, with an included ACC plan being a hard requirement. It is also expected to include something from weeks 5-9 (e.g., testing with an advanced technique), and potentially something from weeks 10-15, too. I will tell you if your proposal needs more or less; in general, it's better to include too much and have me tell you to scale down. You may propose to use V&V techniques which aren't covered in the class, and this is even encouraged.

3. Project pull requests (12% total). You are required to submit 12 pull requests between weeks 4-15, one per week, each worth 1%. Each one of these should be relatively small (no greater than 200 lines of code, ideally between 20-100 lines of code). It is expected that you will need to make code changes outside of pull requests. Pull requests are intended for reviews (more on that in a bit), to ensure steady project progress is made, and to catch surprises early.

4. Project code reviews (12% total). You are required to perform 12 code reviews between weeks 5-16, one per week, each worth 1%. The code being reviewed is from previously-submitted pull requests.

5. Project implementation (30%). Your final implementation of the project. It is expected that this will be larger than the sum of your pull requests (i.e., not all the code you need to write can be reviewed). This grade is based on whether or not your implementation actually implements what you proposed, with fully-conforming implementations receiving full credit. It is important that there are **no surprises** here; projects which do not conform to proposals will be penalized, irrespective of reason. If you are worried that your project will not conform to the proposal, **let me know before this point** and we can adjust the proposal without penalty. It is somewhat expected that proposals may be too broad in scope, and oftentimes we won't be able to tell until we really dive into things.

6. Project report (6%). You will write background information on the techniques you used, how you applied them, and what you've learned from the application.

7. Project presentation (5%). You will give an oral presentation on your project, detailing the same sort of information in the report.

Plus/minus grading is used, according to the scale below:

If your score is >=...	...you will receive...
92.5	A
89.5	A-
86.5	B+
82.5	B
79.5	B-
76.5	C+
72.5	C
69.5	C-
66.5	D+
62.5	D
59.5	D-
0	F

Plagiarism and Academic Honesty

While collaboration is allowed on assignments, you are responsible for all of your own work. You may **not** take code from online sources and submit it as your own. If you do find code online which you wish to include in a solution, you must **cite it**. Any violations can result in a failing grade for the assignment, or potentially failing the course for egregious cases. A report will also be made to the Dean of Academic Affairs. Students who repeatedly violate this policy across multiple courses may be suspended or even expelled.

Attendance
In the first week of class, I will take attendance. If you miss both sessions in the first week and have not made alternative arrangements with me, you must drop the class, as per University policy (http://catalog.csun.edu/policies/attendance-class-attendance/). After the first week I will not take attendance, though you are strongly encouraged to attend.

Communication
- Piazza is strongly preferred (allows for private messages, anonymous posting, and class-wide public posting)
- Email is a fallback in case Piazza isn't working
- Do **not** use Canvas' messaging (very easy for me to miss messages)

Late Policy - Assignments
Unless prior arrangements have been made, for each day an assignment is late, it will be deducted by 10%. Assignments that are submitted more than 9 days late will not receive any credit.

Late Policy - Other Components
Unless prior arrangements have been made, pull requests and code reviews must be submitted by Sunday at 11:59 PM. Late submissions will not be accepted. Similarly, late submissions will not be accepted for the final project implementation and report.

Class Feedback
I am open to any questions / comments / concerns / complaints you have about the class. If there is something relevant you want covered, I can push to make this happen. I operate off of your feedback, and no feedback tells me "everything is ok". This is the first time I'm teaching this course, and it is the first time the course has had this particular structure, so I'm anticipating that it won't all be smooth sailing.

---Class Schedule and List of Topics on Next Page---

Class Schedule and List of Topics (Subject to Change)

Week	Monday	Wednesday
1	8/27: Introduction, motivation, project information	8/29: project information, ACC-based test planning
2	~~9/3~~: Labor day (no class)	9/5: ACC + project, code reviews
3	9/10: Project selection deadline, linters	9/12: Unit, integration testing. Automated testing, testability
4	9/17: Testability, mocking	9/19: Measuring testing effectiveness: test coverage
5	9/24: Measuring testing effectiveness: mutation analysis	9/26: Unsound bug-finding tools (e.g., FindBugs)
6	10/1: Test generation: introduction (basic concepts + loops)	10/3: Test generation: grammar-based, grammar and AST review
7	10/8: Test generation: grammar-based, implementation	10/10: Test generation: grammar-based, thinking in grammars
8	10/15: Property-based testing: introduction + relationship	10/17: More property-based testing, concolic execution introduction
9	10/22: Concolic execution: discussion	10/24: Concolic execution: in practice
10	10/29: Model checking: LTL	10/31: Model checking: LTL
11	11/5: Model checking: SPIN/TLA+	11/7: Model checking: SPIN/TLA+
12	~~11/12~~: Veteran's day (no class)	11/14: Model checking spillover; verification introduction
13	11/19: Verification: Dafny	11/21: Verification: Dafny
14	11/26: Verification: Dafny overspill, intro to dependent types	11/28: Verification: dependent types and Coq
15	12/3: Verification: Coq	12/5: Verification: Coq
16	12/10: Project presentations	~~12/12~~: Semester over (no class)