Course Objective:
When a program is implemented to provide a concrete representation of an algorithm, the developers of this program are naturally concerned with the correctness and performance of the implementation. Software engineers must ensure that their software systems achieve an appropriate level of quality. Software verification is the process of ensuring that a program meets its intended specification. One technique that can assist during the specification, design, and implementation of a software system is software verification through correctness proof. Software testing, or the process of assessing the functionality and correctness of a program through execution or analysis, is another alternative for verifying a software system.

Software testing is not a “silver bullet” that can guarantee the production of high quality software systems. While a “correct” correctness proof demonstrates that a software system (which exactly meets its specification) will always operate in a given manner, software testing that is not fully exhaustive can only suggest the presence of flaws and cannot prove their absence. It has noted by various researchers and industry professionals that it is impossible to exhaustively (or completely) test an application because: (1) the domain of program inputs is too large, (2) there are too many possible input paths, and (3) design and specification issues are difficult to test. The first and second points present obvious complications and the final point highlights the difficulty of determining if the specification of a problem solution and the design of its implementation are also correct.

The course intends to improve students skill towards testing both small-scale programs and large-scale projects developed in-house or in-industries.

Course Contents:

Lectures
Software Testing: Basic concepts and preliminaries; The Psychology and economics of Program Testing; Theory of Program Testing; Testing Techniques – static and dynamic; Unit testing, Integration testing, System testing, Regression testing, Black-box and White-box techniques, Static Techniques like code inspections, static analysis and dynamic analysis; Static analysis tools; Control flow and Data flow testing, Non-functional Testing; Test case generation from UML models such as Use case, Activity diagram, Sequence and Statechart; Test case planning, test case design, test case selection, test case prioritization, test case execution; Acceptance testing, GUI testing; Software Quality, McCall’s Quality Factors and Criteria, ISO 9126 Quality Characteristic; Maturity Models, CMM, Testing Maturity Model, Test process improvement

Lab Sessions
The lab sessions of the course use open source testing tools (e.g., JUnit Unit Testing Framework, TestNG Integration Testing Framework, MuJava (Mutation Testing), Static Analysis tools – Findbugs, CodePro, PMD, NFR testing tools, GUI testing tools) and handouts/exercises to generate effective test cases from the source code (or software project artifacts)

Reference Books:
2. Software Quality Assurance: From theory to implementation, Daniel Galin, Pearson Education Limited 2004
3. Software Quality Engineering Testing, Quality Assurance, and Quantifiable Improvement, Jeff Tian, IEEE Computer Society