Project

- I must approve the project
- Work in teams of 2-3 people depending on the scope of the project; discuss with me
- Types of projects
  - Research: solve a new problem, survey state of the art, extend an existing technique, …
  - Implementation: implement an existing technique and perform empirical/feasibility study
  - Experimentation: apply a technique and evaluate its effectiveness, perform an experiment and interpret its results, …
  - Commercial: download, install, evaluate, and compare several existing commercial tools
Project—Important Dates

- September 22: Initial proposal due by e-mail to Mary Jean, Saswat (saswat@cc), George (Baah@cc), and Raul (raul@cc)
- September 29: Project proposal due (2-pages)
- October 27: Intermediate status report (progress towards project goals, discussion of missing steps, tentative schedule for completion, ≤ 2 pages); need to have significant part done
- November 17: Project papers due
- December 1: Revised project papers due
- December 1,3: Project presentation (everyone prepared for December 1)

Project Suggestions

Analysis projects
1. COBOL to Jimple translator
2. Multi-core Soot
3. Multi-core Data-flow Analysis

Testing projects
4. Test-case generation assistant
5. Implementation of CUTE
6. Guided Test-case Generation

Oracle projects
7. Statistical Models of Program Executions
Project Suggestions

Fault-localization, debugging projects
  8. Context-sensitive Tarantula
  9. Object-sensitive Tarantula
  10. Visualization for Interleavings
  11. Survey of Concurrency Bugs

Other
  13. Literature Search
  14. Individually Designed

Analysis Projects
COBOL to Jimple Translator

Common Business-Oriented Language to Jimple Translator

Motivation
- Most existing software is written in COBOL, but these systems need to be upgraded (rewritten) in newer languages, such as Java
- Transfer of knowledge from research community to COBOL world is hindered by lack of infrastructure for analysis

Goal
- Develop a translator from Cobol to Jimple

How
- Determine what is available for analysis of COBOL programs
- Develop a translator for COBOL to Jimple so that the relationships in the COBOLI programs can be identified using Jimple
- Does not require previous experience with COBOL—knowledge of basic compiler technology is sufficient
- There are tools that build AST (abstract syntax tree) for COBOL. The project will involve familiarizing with this AST and jimple, and then emit jimple code by traversing the AST.
- Then if time permits, you can apply some existing soot analysis to solve a problem that is faced in COBOL program maintenance.
Multi-core Soot

Multi-core Soot
• Is a coarse-grain parallelism in Soot to take advantage of the new multi-core machines
• Motivation
  • Soot is a popular, open-source program analysis system for Java.
  • Currently, during any program analysis, significant time is spent in converting bytecode of each method (typically tens of thousands) into Soot's internal representation, jimple.
  • This conversion, called jimplification, can be done in parallel for each method.

Multi-core Soot

• Goal
  • Implement coarse-grain parallelism in Soot to take advantage of the new multi-core machines
• Requires
  • understanding and modifying large Java systems, basic understanding of parallel programming
Multi-core Soot

- Typically changes will be
  - creating a pool of worker threads; when a method needs to be simplified, a worker thread from the pool is assigned to the task.
  - replacing non-concurrent library classes such as Map, and Lists with their concurrent counterparts in Java's java.util.concurrent package.
- For help with understanding Soot, contact saswat@cc

Multi-core Data-flow Analysis

Multi-core data-flow analysis
- Is a design and implementation of a parallel version of a data-flow analysis problem, similar to reaching definitions
- Goal
  - Design and implement an efficient parallel algorithm to solve this data-flow analysis problem to run on new multi-core machines.
  - Requires: interest in algorithms, understanding of parallel programming
  - Language of implementation: Any
Multi-core Data-flow Analysis

• The dataflow analysis is problem is similar to reaching definition analysis with one difference: New edges are added to the graph as Out sets gets updated during propagation as follows:
  • Besides the standard inputs to the algorithm such as the graph, initial Out's, and Kill's for each node, there is another input F. F is a set of quadruples of nodes (y,u,v,x).
  • An edge from x to y is added if intersection of Out(u) and Out(v) evaluates to non-empty and F contains the quadruple (y,u,v,x).

Multi-core Data-flow Analysis

• Additional information
  • All inputs to your tool will be with respect to the nodes in the graph. You will use an existing tool to generate the inputs.
  • You can either create and manage threads explicitly, or use higher-level tools such as OpenMP or Cilk++.
  • Further references on using Cilk++ for solving similar problems are available.
  • Contact saswat@cc for further clarification
Testing Projects

Test-case Generation Assistant

- Helps a tester to create a test case that covers a target statement (basic block)
- Shows how close to the target was the last execution of the program
- Motivation
  - Automatic test-case generation is difficult but an assistant that could help the developer could reduce the time to get new test cases
Test-case Generation Assistant

- **Goal**
  Design and create such a tool

- **How**
  Create a tool with features such as
  - Ability to show the proximity of execution to target
    - Method calling distance at the call graph
    - Statement distance to target, if target’s method is called
    - If possible, use control-dependence distance
  - A visualization of the covered call graph and distance to target’s method
  - Integrated into Eclipse

---

Test-case Generation Assistant

- **Other nice features that could be included**
  - Highlight covered statements and target in Eclipse
  - Show all paths that can lead from last execution to target
  - Values of variables used in branching statements where execution diverged from the target method/statement
  - Show the guidance in Java source code rather than Jimple. The Java compiler, in debug mode, preserves local variable names and associated source code lines. Soot has an option to keep the line number for each Jimple statement.
CUTE (Concolic Unit TEsting)

- Is a tool for automatically generating test-inputs for C programs.
- Goal
  - Implement a version of CUTE
- How
  - Instrument the program as described in http://sp09.pbworks.com/CUTE-Homework to generate a trace for a particular execution.
  - Post-process the trace to compute path-conditions.
  - Use a constraint solver to solve the path conditions to find the test-input.

Implementation of CUTE

- Requires
  - Understanding of symbolic execution and constraint solving
- Additional information
  - This is an implementation that can be a stepping-stone for further research
  - You will need to use a tool to instrument C programs. CIL is one such popular tool.
  - You will need to use a constraint solver to solve path conditions. Yices is a good choice for this.
  - Contact Saswat (saswat@cc) for more information
Guided Test-case Generation

Guided test-case generation

- Finds an input that cause execution to cover a target statement
- Uses machine learning / genetic algorithms to modify input to get “closer” to the target

Guided Test-case Generation

- Goal
  Find new guidance algorithms or strategies (e.g., fitness functions) not present in literature. These algorithms may be simpler than existing techniques

- How
  - Build generator of syntactically valid inputs for subject programs
  - Instrument statements or branches to collect sequence of events (program path) taken by an execution
  - Use control-flow or program-dependence graphs to measure distance to target of current execution(s) that will be mutated
Guided Test-case Generation

- Additional information
  - Search-based Software Test Data Generation: A Survey – McMinn STVR 2004

- Contact raul@cc for more information

Oracle Projects
Statistical Models of Program Executions

Statistical models of program executions
• Addresses the “oracle problem” automatically by identifying deviant behaviors of programs that may cause crashes or performance issues
• Builds a statistical model using observations (e.g., statement coverage) to identify groups of similar characteristics
• May use a “training set” of executions to build the model(s) – machine learning connection

Statistical Models of Program Executions

• Goal
  Investigate the use of one or more statistical models based on execution “events” (e.g., library calls, statements covered) for classifying executions in useful ways (e.g., pass/fail, valid/invalid input file, patterns of usage of program components)
• How
  • Instrument statements or certain library calls
  • Find models that group coverage/count vectors in ways that statistically correlate with passing/failing executions.
Statistical Models of Program Executions

- Example references to read
  - Active Learning for Automatic Classification of Software Behavior - Bowring, Regh, Harrold  ISSTA 04
    - [http://pleuma.cc.gatech.edu/aristotle/Publications/Papers/p398-bowring.pdf](http://pleuma.cc.gatech.edu/aristotle/Publications/Papers/p398-bowring.pdf)
  - The Probabilistic Program Dependence Graph – Baah, Podgurski, Harrold  ISSTA 09
  - Other work by Podgurski:
    - Finding failures by cluster analysis of execution profiles  ICSE 01
    - Automated support for classifying software failure reports  ICSE 03

- Contact raul@cc for more information

Fault-localization, Debugging Projects
Context-sensitive Tarantula

Context-sensitive Tarantula

- Is a fault-localization system
- That computes a suspiciousness value for every (statement, context); these (statement, context) pairs can then be presented in an ordered list to the user for help in locating the fault in the program.

- Motivation
  - Context-sensitivity of the statements could improve the fault-location effectiveness of Tarantula

Context-sensitive Tarantula

- Goal
  Design and create such a context-sensitive fault-localization system for Java programs

- How
  - Develop an instrumenter that will gather the (statement, context) coverage for all statements and contexts; you can use existing instrumenters and change them to gather the information you need
  - Develop a fault-localization tool that will compute the suspiciousness of each (statement, context) for a faulty program and test suite
  - Compare your results to those achieved by a fault-localization system that uses only statement coverage information to compute the suspiciousness
Context-sensitive Tarantula

- Background and additional information
  - Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique
    - [http://pleuma.cc.gatech.edu/aristotle/pdffiles/Jjones/EEOTTAFLT.pdf](http://pleuma.cc.gatech.edu/aristotle/pdffiles/Jjones/EEOTTAFLT.pdf)
  - Visualization of Test Information to Assist Fault Localization
  - Contact Saswat (saswat@cc) for more information

Object-sensitive Tarantula

Object-sensitive Tarantula

- Is a fault-localization system
- That computes a suspiciousness value for every (statement, object); these (statement, object) pairs can then be presented in an ordered list to the user for help in locating the fault in the program.
- Motivation
  - Object-sensitivity of the statements could improve the fault-location effectiveness of Tarantula
Object-sensitive Tarantula

• Goal
  Design and create such a object-sensitive fault-localization system for Java programs

• How
  • Develop an instrumenter that will gather the (statement, object) coverage for all statements and contexts; you can use existing instrumenters and change them to gather the information you need
  • Develop a fault-localization tool that will compute the suspiciousness of each (statement, object) for a faulty program and test suite
  • Compare your results to those achieved by a fault-localization system that uses only statement coverage information to compute the suspiciousness

Object-sensitive Tarantula

• Background and additional information
  • Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique
    • http://pleuma.cc.gatech.edu/aristotle/pdffiles/Jjones/EEOTTAFLT.pdf
  • Visualization of Test Information to Assist Fault Localization
    • http://pleuma.cc.gatech.edu/aristotle/pdffiles/Jjones/VOTITAFL.pdf
  • Contact Saswat (saswat@cc) for more information
Visualization for Interleavings

Visualization for interleavings
- Shows how a shared variable interleaves among multiple threads
- Helps to detect problematic data access patterns that can cause a program failure

• Goal
  • Design and implement a tool that visualizes thread interleavings for multi-threaded Java programs, with the following requirements
    • The tool should show interleavings for a (or all) shared variables.
    • The tool should show problematic data-access patterns within a interleaving.
    • The tool should work for Contest Benchmark programs (small programs, less than 1KLOC).
Visualization for Interleavings

- More information and background
  - Visualization tool for interleavings (we don’t use this tool): Pajé: An Extensible Environment for Visualizing Multi-threaded Programs Executions
  - Selecting shared variables using Soot analysis tool: Static Lock Allocation
  - Definition for problematic data-access patterns: Dynamic detection of atomic-set-serializability violations
  - Contest Benchmark: Towards a framework and a benchmark for testing tools for multi-threaded programs.
  - Contact Sangmin (sangminp@cc) for more details

Survey of Concurrency Bugs

- Is a literature search on concurrency bugs in diverse systems (i.e., thread-based systems, message passing systems)
- Is collecting concurrency bugs from real-world software
Survey of Concurrency Bugs

- **Goal**
  - Classify and compare concurrency bugs for thread-based and message passing systems.
    - Concurrency bugs can be classified as deadlock, atomicity violation, order violation, and others (refer to Shan Lu's paper).
    - The easiest way to find concurrency bugs is to search software bug repositories.
  - Collect, reproduce, and demonstrate concurrency bugs for real-world software.
    - Each software system should be bigger than 10KLOC
    - Collect at least 5 concurrency bugs (total 15 bugs) for each pthread-based, Java-based, and MPI-based system.

Survey of Concurrency Bugs

- **More information and background**
  - Concurrency bug classification on thread-based system: [Learning from Mistakes --- A Comprehensive Study on Real World Concurrency Bug Characteristics](#)
  - Bug survey on MPI programs: [Automated, scalable debugging of MPI programs with Intel® Message Checker](#)
  - Cactus bug repository (that uses MPI): [http://www.cactuscode.org/BugReporting/](http://www.cactuscode.org/BugReporting/)
  - Contact Sangmin (sangminp@cc) for more information
Using Machine-learning Techniques for Software Engineering

Using machine-learning techniques for software engineering

- Helps to efficiently search large space (e.g., searching input space in software testing).
- Helps to model user behavior to guide software testing.
- Helps to understand and visualize program execution (e.g., Probabilistic PDG).

Using Machine-learning Techniques for Analysis and Testing

- Goal
  - Classify the machine learning techniques based on usage in software engineering area.
  - Suggest ideas (and implement prototype) on how to apply machine learning technique to concurrent program testing and debugging.
    - For example, the PPDG can be extended to multi-threaded programs by adding edges representing
Using Machine-learning Techniques for Software Engineering

• Additional information and background
  • Efficient Search: The Current State and Future of Search Based Software Engineering
  • Modeling Web: Measuring and modeling usage and reliability for statistical Web testing
  • Probabilistic PDG: The Probabilistic Program Dependence Graph and Its Application to Fault Diagnosis

Other Projects
Literature Search

• Perform a literature search on a specific topic, to:
  • Assess the state of the art and practice in that topic area
  • Identify open questions and problems
  • (And possibly) lay the groundwork for further research in that area

Literature Search

• How
  • You will select a topic, possibilities include:
    • Techniques based on static and dynamic analysis
    • Dynamic analysis techniques and their use/application
    • Modern trends in software testing and analysis
    • A roadmap of SW testing and analysis
    • Your own topic, after consulting with me
  • You will survey the literature for papers on your topic, and select 8-10 of these for further study
Literature Search

- Your project report will include:
  - An extensive bibliography listing all work in the area, with detailed annotations on the 8-10 you study
  - A list and brief discussion of related commercial/research tools
  - An overview of the papers
  - (most important) An attempt to shed light on the topic area, by imposing some categorization scheme on the work, and/or exposing open problems

Individually Designed

- We can discuss your ideas