Senior Design Projects in Compilers / Program Analysis / PL

Template updated Fall 2019

1 Overview

Tools to aid developers in creating correct and efficient software are extremely important for Software Engineers. These tools include compilers, interpreters, and tools to analyze source code for "correctness" or other properties. Such tools draw upon fundamental theoretical concepts, software design principles, and efficient implementation practices. In short, a project to develop software tools will build upon much of the Software Engineering curriculum. This document serves as a guide for senior design projects that develop tools to analyze or manipulate software. These projects take as input a "program" (or part of one), written in a high-level formal language. This is referred to as the "front end" and this portion will be fairly similar (in principle) for any software tool. Depending on the purpose of the software tool, there will be one or more "back ends" used to perform processing on the input program. The front end, and several possible back ends, are further discussed below. Each of these gives several opportunities for project "depth". Several questions are posed that should be addressed either by the project proposal or in the final report.

2 Input language (front end)

Any project proposal for software tools must address the issue of "which programming language(s) are supported", with justification for the choice of language. Does the tool target an existing language? Why was that language chosen? Does the tool target a fragment of an existing language? Does the fragment include enough features to be useful? Or does the tool target a new language? In the case of a new language, a (possibly significant) portion of the project should address the development of the new language. What is unique about the new language? Is it a general purpose language or is it specific to a problem domain? What are the important features of the language? What are the shortcomings of existing languages that the new language overcomes?

2.1 Preprocessor

What (if any) are the features of the language preprocessor? Are comments supported? How is the source code split into modules? Is it possible to include other source files (e.g., `#include` directives)? Is conditional compilation supported (e.g., `#ifdef` directives)? Can the programmer define macros?

2.2 Develop and formalize grammar

The project should specify the grammar for the input language. If the tool targets an existing language, then this should be fairly trivial, as the existing language will already have a specification. If the tool targets a fragment of an existing language, then the grammar serves to indicate "which fragment". If the target is a new language, then development of the language's grammar should be part of the project, and the grammar will be one of the deliverables.

2.3 Develop and formalize type rules

The project should specify the type system used in the input language. Again, for an existing language, this should already be part of the language's specification. If the target language is a fragment of an existing

language, then specifying the type system again serves to indicate "which fragment". If the target is a new language, then development of the language's type system should be part of the project and one of the deliverables. Depending on the nature of the new language, the type system might be simple, or based on one used in another language, or significantly different.

2.4 Runtime environment and supporting libraries

What is the assumed runtime environment of the input language? What is the mechanism used for function/method calls? Are there function pointers? Are there virtual methods? Is recursion supported? How is I/O handled? Is there dynamic memory? Is dynamic memory managed (like Java) or not (like C)? What about other features (especially for a new language) such as multi-threading, thread synchronization, thread/process/network communication?

3 Parser

Most (but not all, see below) software tools read the source code from text files. In that case, a parser is required to read the source code and build an appropriate abstract syntax tree (AST) representation for internal use to aid in further processing.

Is everything going to be written "from scratch" or will the project add to an existing tool? In any case, the parser, along with any design decisions (e.g., what AST structure is used, which type of parser is used) and their justifications will be deliverables.

One alternative to a parser is to use a graphical front-end for source code, as done for example with Scratch. Note that there is still a formal language (with grammar, type rules, etc.) but the GUI front end will effectively build the AST directly and there will not be much need for a parser.

4 Examples of back ends

4.1 Program analysis

The goal of the software is to automatically reason about the input program, or executions of the input program. Different types of analysis are possible, such as identifying (classes of) bugs in the input program, automatically generating test inputs (according to some coverage metric) for use in testing the input program, or even identifying bugs and possible repairs to the input program.

4.2 Build a compiler

The goal of the software is to translate the input program into another language, referred to as the "output language". Presumably this is a lower-level language that can run natively on some system or in a virtual machine, although a "new" language could reasonably be compiled into some other standard language. The output code is generated automatically based on the AST. Optimizations to produce more efficient code (either in terms of code size, or speed, or power consumption) can be applied at various stages.

A number of interesting design decisions can be made here, and a great deal of development may be required, depending on the nature of the output language and the environment. Is the output language and environment some common general computing architecture (such as Java bytecode to run on a JVM, or Intel machine language running in Linux)? Is the output language common (such as Intel machine language) but running in a non-standard environment (an unusual or custom-built OS)? Is the output environment uncommon (unusual processor and operating environment)? Is the output a custom-designed low-level language which is then run in a custom-designed virtual machine?

If the source code is split into modules, how will the compiler build the modules separately? How will the compiled modules be pulled together into a single program?

4.3 Build an interpreter

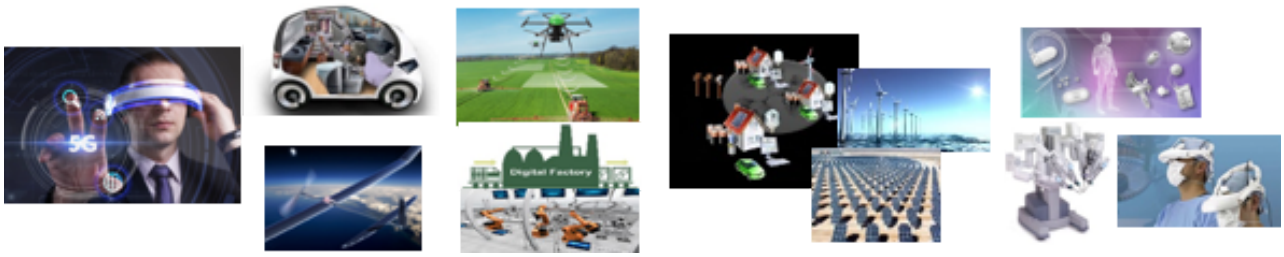
The goal of the software is to directly execute the input program. The interpreter will parse the input and build an appropriate AST, which can then be executed directly. It is also possible to build an intermediate representation first; this is effectively the same as building a compiler that produces the intermediate representation and a virtual machine to execute the intermediate representation, as discussed above.

Note that one possible goal of an interpreter could be to detect bugs (see the "program analysis" discussion above). For example, a C interpreter could check that all array indexes are within the bounds of the array, or that all variables are initialized before they are used (similar to `valgrind`, a memory debugging tool).

5 Testing

It is expected that a collection of input files will be built to use as benchmarks and for automated regression testing. Both *correct* and *incorrect* programs should be included, as the software should contain error detection and error handling mechanisms. Note that automated testing is more difficult with a graphical front end.

CPS Project Template



1. CPS Overview

Tightly integrating sensing, networking, computing, and control with physical systems and processes, cyber-physical systems (CPS) are expected to transform the physical world around us and the way we interact with the physical world, and they will serve as foundations for a wide range of domains such as augmented reality, agriculture, renewable energy, transportation, industrial automation, advanced manufacturing, healthcare, and public safety. CPS is interdisciplinary in nature, and it emcompasses a wide range of topics such as the following:

- Modeling, design, analysis, and implementation of cyber-physical systems
- Dynamic behavior modeling, state machine composition, and concurrent computation
- Sensors and actuators
- Embedded systems and networks
- Feedback control systems
- Analysis and verification techniques, temporal logic, and model checking
- Machine learning

More details on CPS as a discipline can be found from the book Introduction to Embedded Systems: a Cyber-Physical Systems Approach by Lee and Shia.

Canonical applications of CPS can be found in digital and precision agriculture, connected and autonomous vehicles, smart grid, Industry 4.0, Internet of Things (IoT), smart and connected communities, and so on. The broad scope and transformative nature of CPS offer ample opportunities for innovation, thus serving as an exciting context for defining interesting and potentially transformative senior design projects. A senior design project can focus on one or a few inter-related topics of CPS.

2. SE Elements of CPS Senior Design Project

In particular, a software-engineering-oriented CPS senior design project may focus on the software tools and systems of CPS such as those for CPS modeling, design, analysis, or implementation. It is recommended that a CPS project team have expertise in both the cyber aspects (e.g., software engineering and computer engineering) and physical aspects (e.g., electrical engineering) of CPS. However, projects with a software-engineering focus should include sufficient software scope to provide each software engineering student a level of complexity appropriate for senior design. When scoping a project, the following sub-tasks and associated student effort can be used as a rough guide -- not all projects need all components.

1. Embedded: The heart of a CPS implementation project will be the embedded software. As such it must be carefully developed and tested to ensure proper behavior. Testing may take place in simulation and/or the real-world.
    a. Develop and test implementation using extension and integration of existing domain-specific embedded software frameworks (1-2 students)
    b. Develop and test implementation build from the ground-up requiring significant modeling and simulation of the cyber-physical system (2-3 students)
2. Modeling/analysis: A critical aspect of CPS that is often overlooked by inexperienced engineers is modeling and analysis. Generating software models for a CPS application requires careful understanding of the target system and methodical application of software engineering approaches to ensure its ability to usefully reflect the physical system for purposes such as verification.
    a. Develop a software model for a specific CPS (2 students)
    b. Apply a software verification technique to a specific CPS (2 students)
3. Interface: Often times CPS systems require specialized methods for operators (or designers in the case of modeling and simulation) to visualize and interface with the system.
    a. Specify, prototype, and solicit user feedback on a basic user interface for the particular software (1 student)
    b. Specify, prototype, and solicit user feedback on a novel user interface with significant custom coding is required beyond use of pre-designed frameworks (2-3 students)
4. Cloud: Many applications of CPS contain a cloud aspect, including the collecting and analyzing of a significant amount of data generated by the
    a. Develop and test a basic collection, storage, and serving system for data from a CPS (1 student)
    b. Develop and test a system including the above plus custom data analysis for a CPS (2-3 students)
5. End-to-end integration is a critical component of prototyping projects. Often times integration is overlooked and becomes a limiter within the project. For software requiring

two or more of the above components, an additional 1-2 students worth of time should be expected in order to account for a properly interfaced and systematically tested integration.

3. Example Project Directions/Ideas

Please find below a few example CPS project directions/ideas with a strong software engineering/systems focus:

1. Prototyping a multi-domain simulation/emulation software system for *connected and automated vehicles* using open-source software systems, such as the OpenAirInterface software system for 5G and the SUMO vehicle mobility simulator.
2. Prototyping a multi-domain simulation/emulation software system for *unmanned aerial vehicle (UAV) swarm* using open-source software systems, such as the OpenAirInterface software system for 5G and the NPS/JSBSim aircraft simulator.
3. Prototyping a wireless networked, heterogeneous robot swarm (e.g., with unmanned aerial/ground/surface vehicles) simulation/emulation system using open-source software platforms, such as the OpenAirInterface software system for 5G, the SUMO vehicle mobility simulator, and the NPS/JSBSim aircraft simulator.
4. Modeling and analyzing the real-time behavior (e.g., real-time communication capacity) of an industrial wireless network, and prototyping the network using open-source software systems such as the OpenAirInterface system for 5G and the Contiki system for IoT.
5. Using Simulink to model and analyze the behavior of connected and automated vehicles or UAV swarms.
6. Model a network of batteryless, energy-harvesting IoT nodes running a distributed key-value store to provide ACID properties.

A Template for Software Engineering Senior Design Projects in Cybersecurity

Overarching Guidelines

We start with some key elements for a good software engineering (SE) senior design project. The list is followed by some project ideas for software analysis and verification tools for cybersecurity.

- *Purpose and Impact*: The project should have a clearly defined purpose and impact. The idea is to identify the value the project provides and who benefits from it and why.
- *Challenging Aspects*: The project should entail software engineering challenges involving at least one of the following aspects: analysis, synthesis, or integration of software artifacts.
- *Plan to Address the Challenges*: The project should include a concrete plan that describes how the challenges will be met so that the project could be successfully completed in the given time frame.
- *Roles and Responsibilities*: After discussing individual strengths and weakness, establish roles for each team member including leader, communicator, and technical roles. Outcome

A software project can be an especially enriching experience and enable you to undertake substantial projects if you build on existing software. Real-world software projects rarely start from scratch, instead they leverage existing software. So, think about how your project can leverage existing software.

Software being pervasive and enabling technology, software engineering projects are ideally suited to be interdisciplinary. A good interdisciplinary should address an important application and at the same time involve challenging aspects as a software engineering project.

Project Ideas: Software Analysis and Verification Tools for Cybersecurity

This is an important area of cybersecurity pursued by industry, the Defense Advanced Research Project Agency (DARPA), the Department of Homeland Security (DHS), and the National Security Administration (NSA). It is an area that offers many opportunities for senior design projects in software engineering. Here are three broad categories of tool projects. We have included an exemplary tool for each category. The published articles and short demos of these tools can help you shaping your senior design projects.

Integration Projects *Build novel tools by integrating available components. The Mockingbird tool presented at the 2019 International Conference on Software Engineering (ICSE 2019).*

Mockingbird: It combines static and dynamic analyses to yield an efficient and scalable approach to analyze large Java software. The framework is an innovative integration of existing static and dynamic analysis tools and a newly developed component called the Object Mocker that enables the integration. The static analyzers are used to extract potentially vulnerable parts from large software. Targeted dynamic analysis is used to analyze just the potentially vulnerable parts to check whether the vulnerability can actually be exploited.

Link to Mockingbird paper: https://ben-holland.com/papers/Mockingbird.pdf

Synthesis Projects *Synthesize new program analysis tools for specific application domains.*

Interactive visualization toolbox to detect sophisticated android malware: Detecting zero-day sophisticated malware is like searching for a needle in the haystack, not knowing what the needle looks like. This paper describes Android Malicious Flow Visualization Toolbox that empowers a human analyst to detect such malware. Detecting sophisticated malware requires systematic exploration of the code to identify potentially malignant code, conceiving plausible malware hypotheses, and gathering evidence from the code to prove or refute each hypothesis. We describe interactive visualizations of program artifacts to understand and analyze

complex Android semantics used by an app. The toolbox incorporates visualization capabilities that work together cohesively and provides a mechanism to easily add new capabilities.

Link to Paper: https://ben-holland.com/papers/Interactive_Visualization_Toolbox_to_Detect_Sophisticated_Android_Malware.pdf

Analysis Projects *The analysis projects can go in different directions: create benchmarks for assessing software analysis and verification tools.*

PScout: Analyzing the Android Permission Specification: Modern smartphone operating systems (OSs) have been developed with a greater emphasis on security and protecting privacy. One of the mechanisms these systems use to protect users is a permission system, which requires developers to declare what sensitive resources their applications will use, has users agree with this request when they install the application and constrains the application to the requested resources during runtime. As these permission systems become more common, questions have risen about their design and implementation. In this paper, we perform an analysis of the permission system of the Android smartphone OS in an attempt to begin answering some of these questions. Because the documentation of Android's permission system is incomplete and because we wanted to be able to analyze several versions of Android, we developed PScout, a tool that extracts the permission specification from the Android OS source code using static analysis. PScout overcomes several challenges, such as scalability due to Android's large code base, accounting for permission enforcement across processes due to Android's use of IPC, and abstracting Android's diverse permission checking mechanisms into a single primitive for analysis.

Link to Paper: https://security.csl.toronto.edu/papers/PScout-CCS2012-web.pdf

Software Resource for Cybersecurity and Interdisciplinary Projects

We have created software to help you in undertaking substantial cybersecurity as well as interdisciplinary projects. The software is available at: https://github.com/kcsl/chpg

This software resource is built around the "*mathematics of connection*." Newton and Leibnitz developed the "*mathematics of motion*" that has taken us from the steam engine to the spacecraft and the computer and ushered the era of systems with massive connections such as software, internet, and social networks. It has enabled us to understand biological artifacts such the genome as a system of massive connections. We need the "mathematics of connection" to advance the technologies for systems with massive connections. The software resource provides the technology to store, analyze, and operate on systems of massive connections.

This software resource incorporates the following:

1. A graph database to store a system of connections such as the software or the bio structures
2. A graph programming language to understand, analyze, and operate on systems of massive connections
3. A capability to define a schema to capture the semantics of connections – a readymade schema is provided for Java and C. The schema is extensible and you can extend it to other programming languages. You can also import other schemas, for example, a schema for protein structures
4. A capability to transform C or Java software and import into a graph database
5. A capability to visualize and interact with connections
6. A capability to create interactive educational modules for the technology you have developed using this resource