**CMPSC 280**
**Principles of Software Development**
**Fall 2015**

# Laboratory Assignment Eight: Understanding and Releasing Real-World Software

## Introduction

Throughout the semester, you have learned about the phases of the software development lifecycle and both the software tools and the non-cognitive skills that are needed to support the creation of high-quality software in each of these phases. Yet, none of the past laboratory assignments required you to deliver a programming systems product to an external customer. In this laboratory assignment, you will gain experience in working with and understanding a large, real-world software system created by external customers who want to release it to GitHub under a free and open-source license. Additionally, you will use tools, like JDepend and JavaNCSS, to characterize this large, already implemented system. You and your team members will also explore both the issues related to effective technology transfer and the ways to improve the quality of a software system's documentation. Finally, you will learn more about the licensing of free and open-source software.

## Learning About SchemaAnalyst

Many real-world software systems interact with a relational database management system (RDBMS) that hosts a database used for persisting the program's data. The database itself is protected by a schema that governs the types of data values that are allowed into the database. If the schema of the database is not correctly defined, then the database may either become corrupted with incorrect data or it may reject values that should be allowed into the storage system. The Schema-Analyst tool, implemented primarily by Phil McMinn and Chris J. Wright, automatically generates test cases that help a database designer establish a confidence in the correctness of the relational database's schema. Since the creators of SchemaAnalyst want you to help them to release their tool to GitHub, this task is the primary focus of this assignment. In particular, you and your team members will compete against the other two teams to be chosen by the external customers as the team whose private Git repository of SchemaAnalyst will be make publicly available on GitHub.

After you have organized your teams, you should learn more about the fundamental ideas underlying SchemaAnalyst's approach to testing database schemas. You can do this by reading some of the research papers — listed in the "References" section of this assignment — that McMinn, Wright, and Kapfhammer have published about SchemaAnalyst. While it is not necessary for you to understand all of the technical and mathematical details in these papers, a familiarity with the concepts developed by these authors will help your team to successfully complete all of the deliverables required for this project. For instance, an understanding of the basics about SchemaAnalyst will allow you to writing good introductory content for the tool's `README.md` file on its GitHub site. Please see the instructor if you have questions about any of these papers.

## Accessing and Using SchemaAnalyst

Currently, the source code of SchemaAnalyst is stored in a private Git repository hosted by Bit-Bucket. Your team should pick a leader who will then ask the instructor for access to this repository, create a private BitBucket repository for your team to complete this project, and then move all of SchemaAnalyst's source code to your team's private repository; the team leader should create a repository that adheres to the naming convention `SchemaAnalyst-<Team Number>`. Please make sure that you share your team's repository with the instructor and both Phil McMinn and Chris J. Wright, your external customers for this assignment. In addition, your customers have created a preliminary draft of user documentation for SchemaAnalyst and provided this to you in a separate Git repository; please have your team's leader ask the instructor for access to this repository and then use this documentation as a start for the `README.md` file on SchemaAnalyst's main page.

Next, each member of your team should review the source code of SchemaAnalyst's build system provided in the `build.xml` file. Can you learn how to use this system to compile the source code of SchemaAnalyst? While compiling with `ant`, can you find out how many Java classes are inside of SchemaAnalyst? Once you have successfully compiled SchemaAnalyst, you should study the user documentation to learn more about how to run the different parts of this tool. Can each member of your team use SchemaAnalyst to generate a test suite for a sample schema of a relational database? What are the inputs and outputs of SchemaAnalyst when it generates test data?

Of course, it is also important to characterize the quality of a test suite that SchemaAnalyst automatically generates. One way to evaluate tests is to use mutation analysis; please refer to the papers in the "References" section to learn more about how mutation analysis can help you to understand the quality of a test suite. Can you and your team members run the mutation analysis feature provided by SchemaAnalyst? What are the inputs and outputs of SchemaAnalyst when it performs a mutation analysis of a test suite? Overall, how good are the generated test suites?

## Preparing SchemaAnalyst for Public Release

You and your team members should review the existing documentation provided by the customers and develop some ideas for ways in which you can improve it further. For instance, the current documentation does not show the output (i.e., the test suite) that SchemaAnalyst produces when you ask it to generate tests. In addition, there is no detailed description of the purpose and behavior of SchemaAnalyst. Of course, the `README.md` that will display on the front page of your Git site would also benefit from a terminal recording, created with a tool like `asciinema`, that demonstrates the system's key features (to learn more about this tool, you can visit `https://asciinema.org/`). Finally, since SchemaAnalyst contains a commercial library called "General SQL Parser" you will need to interact with the external customers to determine the best way to publish their tool under an appropriate open-source license. To ensure that your team is chosen for the privilege of releasing SchemaAnalyst to GitHub, you may also consider other ways in which you can prepare the customers' tool for release. Please see the instructor if you have questions about these tasks.

## Evaluating SchemaAnalyst's Design and Implementation

In addition to releasing the SchemaAnalyst system, it is important for your team to evaluate the quality of its design. First, your team should use a tool called JDepend; as in a previous assignment, you can learn more about JDepend by visiting `http://clarkware.com/software/JDepend.html`. Before you start the next phase of this assignment please make sure that all of your team members fully understand all of the details about the design quality metrics that JDepend calculates. Your team can use these metrics to better understand the quality of SchemaAnalyst's design and make suggestions, to the external customers, for ways in which the design of SchemaAnalyst could be improved. As one of the deliverables for this assignment, your team should use Markdown to prepare a document that contains formal definitions and equations that describe all of the design quality metrics and then comments on the relevant characteristics of SchemaAnalyst's design.

It is also essential for your team to calculate metrics that characterize the quality of SchemaAnalyst's implementation. As you know from a previous laboratory assignment, JavaNCSS is a software tool that can automatically scan your Java source code and report information about the number of non-commented source code statements and the cyclomatic complexity. If you have not already done so, you can learn more about JavaNCSS by visiting `http://www.kclee.de/clemens/java/javancss/`. After reading the Web site for JavaNCSS, each member of your team should search for papers in the ACM Digital Library, available at `http://dl.acm.org/`, that formally describe the meaning of the metrics calculated by JavaNCSS, such as cyclomatic complexity. As one of the deliverables for this assignment, your team should use Markdown to write formal definitions and equations that fully explain all of the metrics calculated by JavaNCSS.

Once all of the members of your team understand the metrics calculated by JavaNCSS, you should download and install this tool. As you did with JDepend, now you must learn how to run JavaNCSS and then apply it to the source code of the system implemented by your external customers. What does the output of JavaNCSS tell you about the quality of SchemaAnalyst's implementation? Does SchemaAnalyst exhibit good implementation characteristics? How can you use the values of these metrics to better understand and suggest modifications for the implementation of the SchemaAnalyst system? How much of SchemaAnalyst's source code is for the program itself? How much of the source code is devoted to the test suite? Overall, how well is SchemaAnalyst documented? What are ways in which your team can improve SchemaAnalyst's documentation?

Additionally, your external customers want you to consider assessing the quality of the test suite that they have implemented for SchemaAnalyst. For instance, your team can think about using a test coverage analysis tool like JaCoCo to calculate the code coverage of SchemaAnalyst's test suite. Alternatively, you could use a tool like MAJOR to perform a mutation analysis of SchemaAnalyst's tests; students who want to learn more about MAJOR and the concepts underlying mutation analysis are encouraged to read several of the papers in the "References" section and to visit MAJOR's Web site, available at `http://mutation-testing.org/`. At minimum, your team must document the baseline characteristics of the test suite. For instance, how many tests come with SchemaAnalyst? Do all of the test cases pass? Are there tests for all of SchemaAnalyst's packages?

Along with running software tools, like JDepend and JavaNCSS, to automatically assess the quality of SchemaAnalyst's design and implementation, you should also take time, as a team, to conceptualize and visualize the design of your customers' tool. Following the outline of the unified modelling (UML) diagram in Figure 6.22 of SETP, your team should create a high-level package diagram for SchemaAnalyst. Next, your team should pick at least one major "sub-system" in SchemaAnalyst and create a UML class diagram, like Figures 6.18, 6.20, and 6.21, that shows the relationship among the classes in this sub-system. Please use a technical drawing tool to crease these UML diagrams; students should consider using a tool like Mermaid, available at `https://github.com/knsv/mermaid/`, to program these diagrams and make them available in their repository. Finally, your team should prepare an explanation of at least one example of how SchemaAnalyst employs object-oriented principles (i.e., polymorphism, inheritance, and encapsulation).

## Review the Textbooks for Key Ideas

To best ensure your team's success at completing this project, you should review all of the content in Chapters 1–6 of SETP, focusing on Chapter 3's relevant details about the process of developing software. Every member of the team should also review Chapters 1–4 of MMM, reconsidering topics such as the roles in a software development team and the techniques that help developers achieve conceptual integrity in a design. Refreshing your understanding of this content will best ensure that your team works well together when completing this assignment — and help you to prepare for the final project where you will deliver another large-scale system to an external customer.

Although not absolutely required for the successful completion of this laboratory assignment, you may also wish to review SETP's material about software documentation (see Section 10.2) and technology transfer (see Section 14.2). Additionally, you may examine Chapter 15 of MMM to learn more about ways in which you can document software. Although this optional content in SETP and MMM reinforces concepts already introduced in past class and laboratory sessions, it may provide further context to support your release a programming systems product on behalf of the external customers. Please see the instructor if you have questions about any of these reading assignments.

## Presentation of Important Insights

As mentioned previously, the external customers will evaluate the work completed by each of the three teams and decide which team has produced the most deliverables of the highest quality. To help the customers to decide which team has done the best work, you are required to prepare a short ten-minute presentation that will "sell" your team's deliverables. The presentation slides should highlight the deliverables that you have completed and the ways in which you think your work adheres to the highest standards of excellence in software engineering. Overall, your presentation must articulate why your efforts will ensure that SchemaAnalyst can be released as a full-featured programming systems product. The team chosen by the external customers will be given permission to, in conjunction with the course instructor, release SchemaAnalyst to the GitHub site.

## Summary of the Required Deliverables

This assignment invites your team to submit one printed version of the following files:

1. A description of and justification for your team's chosen organization, roles, and tool support
2. A complete `README.md` file that fully explains how to understand and use SchemaAnalyst
3. A document that clearly explains the meaning of JDepend's design quality metrics
4. A document that clearly explains the meaning of the metrics calculated by JavaNCSS
5. A full-featured written analysis of the values of all the relevant metrics for the SchemaAnalyst
6. A written analysis of the quality of the test suite provided with SchemaAnalyst
7. Technical diagrams that correctly visualize the design of the SchemaAnalyst system
8. A written explanation of how SchemaAnalyst employs the principles of object-oriented design
9. Modified implementation artifacts for SchemaAnalyst (e.g., build system and source code)
10. The slides of the presentation that you will give on the due date of the assignment

## References

Kinneer, Cody, Gregory M. Kapfhammer, Chris J. Wright, and Phil McMinn (2015). "Automatically evaluating the efficiency of search-based test data generation for relational database schemas," in *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering*.

Wright, Chris J., Gregory M. Kapfhammer, and Phil McMinn (2014). "The impact of equivalent, redundant, and quasi mutants on database schema mutation analysis," in *Proceedings of the 14th International Conference on Quality Software*.

Kapfhammer, Gregory M., Phil McMinn, and Chris J. Wright (2013). "Search-based testing of relational schema integrity constraints across multiple database management systems," in *Proceedings of the 6th International Conference on Software Testing, Verification and Validation*.

Wright, Chris J., Gregory M. Kapfhammer, and Phil McMinn (2013). "Efficient mutation analysis of relational database structure using mutant schemata and parallelisation," in *Proceedings of the 8th International Workshop on Mutation Analysis*.

Just, René, Gregory M. Kapfhammer, and Franz Schweiggert (2012). "Do redundant mutants affect the effectiveness and efficiency of mutation analysis?," in *Proceedings of the 7th International Workshop on Mutation Analysis*.

Kapfhammer, Gregory M. (2012). "Towards a method for reducing the test suites of database applications," in *Poster Compendium of the 5th International Conference on Software Testing, Verification and Validation*.

Just, René and Gregory M. Kapfhammer (2011). "MAJOR: An efficient technique for mutation analysis in a Java compiler," in *Poster Compendium of the 4th International Conference on Software Testing, Verification and Validation*.

Just, René, Gregory M. Kapfhammer, and Franz Schweiggert (2011a). "MAJOR: An efficient and extensible tool for mutation analysis in a Java compiler," in *Proceedings of the 26th International Conference on Automated Software Engineering*.

Just, René, Gregory M. Kapfhammer, and Franz Schweiggert (2011b). "Using conditional mutation to increase the efficiency of mutation analysis," in *Proceedings of the 6th International Workshop on Automation of Software Test*.

Kapfhammer, Gregory M. (2010). "Regression testing," *The Encyclopedia of Software Engineering*. Taylor and Francis – Auerbach Publications.

Kapfhammer, Gregory M. (2004). "Software testing," *The Computer Science Handbook*. CRC Press.