Summary:

This course is intended for software engineers who would like to learn how increase their productivity by learning about programming tools.

Programming is difficult—some of the problem developers face include

- How can a project be structured so that several developers can work on it concurrently?
- How can the building of a project be automated?
- How can a program be written to make it portable?
- How can a program be prototyped efficiently?
- How can a program be tested and debugged efficiently?
- How can the performance of a program be increased?

Using the right tools can solve these problems. Examples include tools for version control, documentation, program building and configuration, automatic testing, and program analysis.

Our approach will be to introduce a specific problem, show how the tool solves the problem, and the technical principles underlying the tool. We will use actual open-source tools that are freely available, and whose operation is completely transparent.

Description:

Have you ever…

- wasted time trying to merge your code with your project partners code?
- struggled trying to understand what your project partners code was doing?
- spent more time writing the code for reading data than for processing it?
- typed the same commands over and over again to build a project?
- pored over a file looking for certain patterns?
- tested a program but missed an obvious problem?
- spent a day looking for a memory bug you should have found in five minutes?
- needed a program to run 10 times faster and use half the memory?

If so, take this class!

The course will be based on the excellent text Essential Open Source Toolset (ET), by Andreas Zeller and Jens Krinke, 2005, John-Wiley.

Your task in this class will be to read and understand ET. It is short and to-the-point, and well-written. Each chapter includes examples, exercises, and further reading.
Prerequisites:

Option III Software Engineering graduate students should already have the programming background needed to appreciate ET.

Assignments will consist of a written component (e.g., “Your boss hears programming in Perl leads to programs that are $5 \times$ shorter than in C++, and demands you code exclusively in Perl. Tell him 5 dangers of this method.”), as well as implementation.

Implementation will involve taking code distributed with the book, and modifying it in small ways. I expect that even if you are not comfortable with the underlying language, looking at the given code, together with the examples in the book, should be enough for you to do the implementation.

The tools used in the book are open source, and available for Unix/Linux, and Windows (e.g., via Cygwin).

Web site:

All material related to the course is available at www.ece.utexas.edu/~adnan/top. (Weekly reading and homeworks are already posted.)

Format/Evaluation:

I will assign homeworks consisting of exercises from ET. The homeworks will not be programming intensive—their purpose is to make you read ET in an organized manner.

Grades will be assigned solely on the basis of performance on the homework. I will create an account for you to upload homework via Wiki. I will grade the homeworks and return them by US mail.

Schedule:

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1</td>
<td>Version control</td>
<td>diff/patch, RCS, CVS</td>
</tr>
<tr>
<td>Weeks 2–3</td>
<td>Processing input</td>
<td>lex, yacc, antrl</td>
</tr>
<tr>
<td>Weeks 4–5</td>
<td>Building programs</td>
<td>Make, ANT, autoconf, Javadoc</td>
</tr>
<tr>
<td>Weeks 6</td>
<td>Processing</td>
<td>Tcl/Tk, Perl</td>
</tr>
<tr>
<td>Weeks 7–8</td>
<td>Testing, debugging, regression</td>
<td>Dejagnu, Junit, Bugzilla, gdb/ddd, valgrind</td>
</tr>
<tr>
<td>Week 9</td>
<td>Processing analysis</td>
<td>gprof/gcov</td>
</tr>
<tr>
<td>Week 10</td>
<td>IDEs</td>
<td>Eclipse</td>
</tr>
</tbody>
</table>

Relationship to PoP:

I teach another summer topics class, “The Practice of Programming” (PoP), which has been quite popular over the past few years. I developed ToP based on conversations with the students who took PoP. The two classes are meant to be complementary—PoP is more about broader principles, while ToP is focused on specific tools. For most of you, PoP would probably be the better one to take first, but they are self-contained, and you can take ToP first.