Software Engineering for Economists

Building Confidence in a Model

- Computational models of socio-economic phenomena are a manifestation of our perceived knowledge about the underlying processes. They key question is how much confidence should we have in a particular model?

- It turns out to be useful to structure such a discussion around three interrelated questions (Adams & al., 2012).

- *Software Engineering* encompasses the tools and methods for defining requirements for designing, programming, testing, and managing software. It is crucial to ensure that the computational implementation is a faithful representation of the original mathematical model (Oberkampf & Roy, 2013). Thus, it is part of the verification step.

Research Example

Running Example

- For the rest of this lecture, we will use a small examples to illustrate ideas of different software engineering tools. However, we will also have a brief look how these tools are applied in the more complex setting of my current research. The online code repository is available online (https://github.com/robustToolbox/package).

Testing

- To see these basic ideas in action, let us check out the testing harness for my current research project (https://github.com/robustToolbox/package/tree/master/development/tests).
- Using bugs to define test cases ensures that they only need to be fixed once.

Profiling

- Now that we have a well designed and tested version of our code, it is time address any performance issues. We will profile our program by measuring the execution time of the program.

- Profiling tools also measure the time spend in each function allowing us to target our development efforts at particularly time-consuming parts of the code.

- Studying the output directly can be rather tedious for large programs. That is when visualization tools turn out very useful. We build on SNAKEVIZ (http://jiffyclub.github.io/snakeviz/).
- For even more advanced visualization, check out *pyprof2calltree* (https://github.com/pwaller/pyprof2calltree/). Tutorial for advanced visualization using *KcacheGrind* (http://bit.ly/1SaXJgM)

Continuous Integration Workflow

- By running the testing harness early and often, bugs are caught closer to their creation. This makes debugging much easier.
- Scalability of research team is improved as basic quality assurance is automated.
- The badges signal to your fellow researchers that we take your responsibilities as a developer of research software serious.

- Reliable work-flow increases own satisfaction.

Best Practices

- Iterative project development with only incremental addition of features. Testing harness ensures that old features are not broken.