Massachusetts Institute of Technology

Robotics: Science and Systems – Spring 2006

Lab Handout: Engineering Meta-Tools

Thursday, February 23^{th}

This handout describes three techniques that we suggest you adopt, to maximize your success and (nearly) minimize your frustration in RSS. The techniques are: *Form Testable Hypotheses*, *Test Hierarchically*, and *Ask Questions*.

Form Testable Hypotheses

Engineering is largely about crafting physical artifacts that behave as desired. In order to desire a particular behavior, one must predict it ahead of time (rather than simply observing it, and seeking to explain it after the fact).

This implies that one should formulate a *mental model* that describes the system one is building, i.e., a mental representation of the system and its behavior under specified conditions. We exhort you to form such a mental model and use it to *form testable hypotheses, and write them down, before you run experiments.* Doing so has a lot of engineering utility, for at least three reasons.

Between you and the system. If the system behaves differently than you predicted, think of this not as a problem, but an opportunity. Chase down the discrepancy and either explain it or correct it.

Think about it: if there's a discrepancy, *either* there is a bug in your system (in which case chasing it down and solving it improves your system), *or* your mental model is flawed (in which case resolving the problem means coming to an improved understanding of your system). In either case, you make progress. Again: discrepancies between your mental model and the system's observed behavior are ideal learning opportunities.

(Obscure aside: bugs and mental model errors can also apply to the means with which you are observing your system, e.g., measurement instruments. So when resolving discrepancies, include in your collection of hypotheses the possibility that your measuring instruments are misconfigured or are being misused.)

Between you and your colleagues. When working as a group, you should seek to form a *shared* mental model with your teammates. Again, when there are discrepancies, you should talk about them, reason them out, and resolve them. When you disagree, there is a great mechanism for resolution: experimentation (see above). Design an experiment as a group, and write a brief description of it in your lab notebooks. Each of you should then make a hypothesis about the outcome, and write it down, again individually. Then run the experiment and observe the results. Repeat until agreement.

As a record of your thought processes. Your lab notebook is the ideal place to record your hypotheses, your experimental designs, the experiment outcomes, and your conclusions from experiments. If you faithfully log these things as they unfold, you will naturally create a record of the evolution of your understanding as your work progresses.

Test Hierarchically

There is a technique for debugging and troubleshooting which we call "hierarchical module testing." Think about every functional module in your system (up to and including the top-level module, the system itself). The module has well-defined inputs, well-defined outputs, and zero or more sub-modules that provide internal functionality. For any module, one can ask, "is this module providing the correct mapping from inputs to outputs?" If the answer is "Yes," then you've validated that module, and you can use it as desired (or turn your attention elsewhere).

However if the answer is "No," then you must debug or troubleshoot the module. Think about it: what could be the cause of the problem? *Either* some sub-module of the module is malfunctioning, *or* all sub-modules are correct, but there is something incorrect in the way the module of interest is supplying inputs to, or interpreting outputs from, one or more sub-modules. This leads immediately to a recursive debugging method: to debug a malfunctioning module, recursively debug any malfunctioning sub-modules, and debug the module's supply of inputs and interpretation of outputs from those sub-modules.

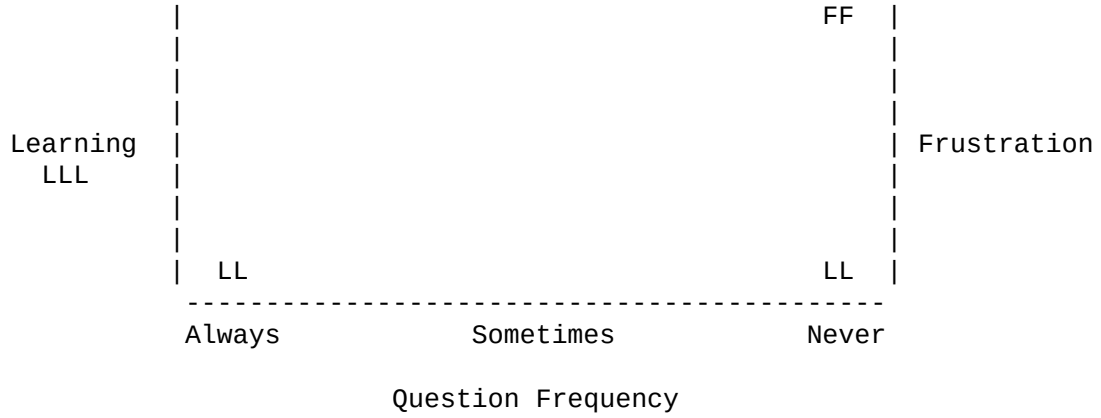
A corollary of this technique is to *test your black boxes*! If the RSS staff (or a vendor or teammate or ...) supplies you with a black box that is claimed to exhibit a certain behavior, then before you rely on it within your system, *test it*. That is, supply it with a set of inputs for which you predict a certain set of outputs, and verify that you indeed observe what you expect to observe.

(Obscure aside: "inputs" means "anything whose state can possibly affect the behavior of your system." This includes external phenomena that may be intermittent, for example bright ambient light in the daytime, which will not generally be present during evening testing.)

Ask Questions

The third technique we urge you to adopt is to *ask questions when you get stuck or when you don't understand something.* We urge all students to ask questions. However, one can ask too many or too few questions. We don't know how to pin down the "right" number of questions, but we can make some observations about the two ends of the spectrum of question-asking frequency.

Suppose you ask questions "always," i.e., whenever you are stuck, even a little bit stuck. In this case, assuming a knowledgeable staff member is around to help, you learn very little and the staff member ends up doing most of your lab for you. On the other hand, suppose you ask questions "never," i.e., you decide to just tough it out whenever you are stuck. This may get you far off-track from the staff's intent in designing the lab, and may leave you extremely frustrated, tired, etc. as well. Note that the onset of frustration may be highly non-linear. Putting it all together, we arrive at the following graph (MKS units omitted):



There is a happy medium somewhere in the middle, which is to ask questions "sometimes," i.e. after giving yourselves a chance to answer the question on your own. We realize that we are being vague here, and that sometimes you don't really have a choice: maybe it's four A.M. and no one is in the hangar or on-line to answer your question. Yet still there is value in thinking about the very process of asking questions.

When you do ask a question, formulate it so as to maximize the chance that we can help you with a useful answer. Think about asking a question as something like filing a bug report: describe the steps you took, including configuration, to set up your experiment; describe the behavior you expected (i.e., your hypothesis), and why; and finally, describe the outcome you observed and its deviation from your expectations. Of course the recipe above doesn't apply to all questions (for example, to many conceptual questions). In any event, we don't mean to discourage you from asking questions. But where the recipe does apply, it can help both students and staff make efficient progress, and can lower everyone's frustration level.