Intro to Methods of Software Engineering Fall 2016

Lecture 4 — October 4, 2016

Patrick Lam *version 1*

Written Communication

"What?!Engineers need to communicate?"

Good communication is always vital to success—

- it's not about being able to calculate derivatives;
- it's not even about being able to write code.

The following skills are also important, but we won't talk about them today.

- active and emphathetic listening; and,
- public (and less-public) speaking [SPCOM 100/223].

How to get better at writing (and any other skill):

practice!
(and get feedback and read others' writing)

"Writing skill" is a misnomer. Your skill depends on your familiarity with the genre.

Important genres for SE:

- code comments;
- commit messages / pull requests / bug reports; and
- emails!

Not so important: English essays, technical reports.

Good technical writing

Jean-luc Doumont proposes that good technical writing should be:

- clear;
- accurate; and
- concise.

Similarly, Derek Rayside proposes:

- clear;
- concise;
- complete; and
- correct.

Good writing is reader-friendly: the writing does not get in your way and you can focus on the ideas in the text.

Before writing, consider the following questions:

- who is the audience?
- what action are you hoping for?

Case study. Pull requests consist of a short message associated with some code that should be merged into the main line of a code repository. For a pull request, the audience is your peers, and you are hoping that they will merge your code.

Here is an example pull request message from a 2014 Capstone Design Project[1].

> Currently if an uncaught exception occurs in a multiobjective algorithm the thread will be terminated but the main thread will continue waiting for solutions indefinitely.
>
> This change allows us to add exceptions to the blocking queue which will be rethrown by the BlockingSolutionIterator, thus terminating the main thread when the solver thread dies.
>
> This change will require modifications to all MultiObjectiveAlgorithms as it changes the interface. multiObjectiveSolve is no longer abstract, instead the abstract template method multiObjectiveSolveImpl should be implemented by the various algorithms.
>
> Reviewers: @AtulanZaman @joseph2625 @mhyee

Note: 1) the use of paragraphs, each with a message; 2) descriptions of: the current state; the effects of the change; and work that will be required as a result of the change; and, 3) code reviews.

Clarity in technical writing. Use the right words and put them in the right order. "Use definite, specific, concrete language." (*Elements of Style #16*; example: "A period of unfavorable weather set in." versus "It rained every day for a week.")

Accuracy. Get the details right. Say who is doing the action (and avoid passive voice: "It was decided to replace the professor with a robot.")

Concision. As *Elements of Style #17* puts it: "Omit needless words", for example "Her story is a strange one." versus "Her story is strange."

[1] `https://github.com/TeamAmalgam/kodkod/pull/37`

Code documentation

Some best practices for code comments:

1. structure your code (just like your writing).
   (avoid 2000-line methods!)
2. choose good names for variables, methods/functions, classes.
   - `i` is totally fine for a loop variable!
   - follow language conventions, e.g. `getX()` versus `x()` (or just allowing direct access to `x`).
   - describe what the thing is/does, e.g. `exceptionQueue()`.
3. in your code, add comments when there are complicated passages (better yet, simplify the code!). One exception: "I tried the obvious thing X but it fails because Y" is a good comment.
4. use good parameter names and include documentation comments for methods and classes, including `@link/@see` references whem appropriate.

```
/**
 * A computational engine for solving relational satisfiability problems.
 * Such a problem is described by a {@link kodkod.ast.Formula formula} in
 * first order relational logic; finite {@link kodkod.instance.Bounds bounds} on
 * the value of each {@link Relation relation} constrained by the formula; and
 * a set of {@link kodkod.engine.config.Options options} specifying, among other global
 * parameters, the length of bitvectors that describe the meaning of
 * {@link IntExpression integer expressions} in the given formula.   The options are
 * usually reused between invocations to the {@linkplain #solve(Formula, Bounds) solve}
 * methods, so they are specified as a part of the {@linkplain KodkodSolver solver} state.
 *
 * <p>
 * A {@link KodkodSolver} takes as input a relational problem and produces a
 * satisfying model or an {@link Instance instance} of it, if one exists.  Some
 * implementation of this interface can also produce a {@link Proof proof} of
 * unsatisfiability, if the given problem has no models.
 * </p>
 *
 * @specfield options: Options
 * @author Emina Torlak
 */
```

5. use unit tests and assertions (more later).