Take Control of Your Code

Software Development Practices for Engineers and Scientists

Steve Eddins, Ph.D.
Software Development Manager
Image Processing, MATLAB Toolbox, MATLAB Math

You're Not a Professional Programmer ...

... but you write a lot of code

```
nColors = 6;
sample_regions = false([size(fabric,1) size(fabric,2) nColors]);

for count = 1:nColors
  sample_regions(:,:,count) = roipoly(fabric,region_coordinates(:,1,cou
                                      region_coordinates(:,2,count));

end

imshow(sample_regions(:,:,2)),title('sample region for red');

%%
% Convert your fabric RGB image into an L*a*b* image using |makecform|
% |applycform|.

cform = makecform('srgb2lab');
lab_fabric = applycform(fabric,cform);

%%
% Calculate the mean 'a*' and 'b*' value for each area that you extracte
% |roipoly|.  These values serve as your color markers in 'a*b*' space.

a = lab_fabric(:,:,2);
b = lab_fabric(:,:,3);
color_markers = repmat(0, [nColors, 2]);

for count = 1:nColors
  color_markers(count,1) = mean2(a(sample_regions(:,:,count)));
  color_markers(count,2) = mean2(b(sample_regions(:,:,count)));
end
```

Engineers and scientists in all disciplines rely heavily on software.
Much of the software is custom code, written in a variety of languages: MATLAB, C++, Java, Fortran, etc.
Applications include data analysis, simulation, embedded controllers, interfacing with lab instruments, etc.
Software projects vary in size from hundred-line MATLAB scripts to applications requiring tens of thousands of lines.

The dilemma: You're not a full-time programmer, and you don't especially want to become one. But you need a lot of software to get your job done. So when software projects stumble (as they so often do), your productivity suffers.

Recognize any of these scenarios?

- You can’t reproduce the results in your year-old paper because the software changed
- Your collaborators keep getting confused because everybody keeps changing the same files
- Everyone’s afraid to touch the critical library written five years ago by a guy who doesn’t work in the lab anymore

Agenda

1. Version control
2. Code quality from the outside – unit testing
3. Code quality from the inside – bad code smells
4. Putting it all together – code refactoring

Use Version Control

Have you ever started making a few "simple" changes, spent a day or so working on them, and then changed your mind? Maybe you realized the changes were misguided, or that they were more complicated than you expected.
You should be using VERSION CONTROL!
A version control system tracks every change made to source code and other documents – what the change was, who made it, and when it was made. It can roll back a file to a previous state, show you what has changed since last month, help you reliably reproduce results, and facilitate collaboration.
With a version control system, you can always revert to the latest version in the repository, or to any previous version.

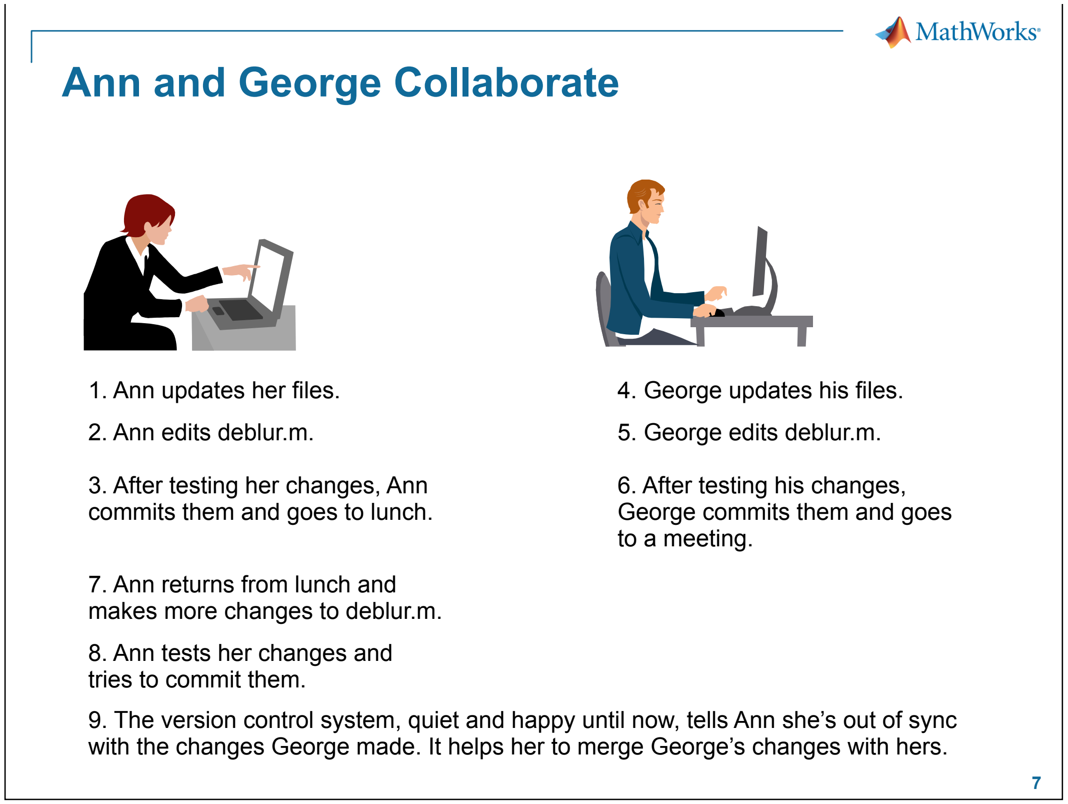
Can you reproduce results from a paper you wrote two years ago? Reproducibility is an important concept in many scientific and engineering disciplines. But reproducing results based on changing software can be difficult or impossible without version control. When you get in the habit of using version control, you'll be able to reproduce past results at will.

Key vocabulary:
Repository – container of the current version and all previous versions of all files associated with a project; managed by the version control system
Working copy – a copy of project files on your own computer that you can modify
Diff – a display of the differences between a file in your working copy and the version of the file in the repository
Commit – submit modified files to the repository
Log – record of developer notes attached to every change

Version Control and Collaboration

“Version control is indispensable on team projects”

“Version control is indispensable on team projects.” [McConnell, Code Complete, page 668]
Version control resolves the thorny question of “who has the latest version of this file?”
Version control shows you at a glance whether anyone else on the team has changed any of the files you have.
Version control lets you easily update your files to the latest versions.
Version control on a server allows anyone (with permission) to access the latest files at any time.



Ann and George Collaborate

1. Ann updates her files.
2. Ann edits deblur.m.
3. After testing her changes, Ann commits them and goes to lunch.
4. George updates his files.
5. George edits deblur.m.
6. After testing his changes, George commits them and goes to a meeting.
7. Ann returns from lunch and makes more changes to deblur.m.
8. Ann tests her changes and tries to commit them.
9. The version control system, quiet and happy until now, tells Ann she's out of sync with the changes George made. It helps her to merge George's changes with hers.

Some things to note about this sequence:
Ann and George do not send each other their files directly.
They can freely choose to work on their local copy of their files at any time.
Each of them is in the habit of committing their work often.

Pragmatic Programmer Tip #23: Always Use Source Code Control
"Always. Even if you are a single-person team on a one-week project. Even if it's a 'throw-away' prototype. […] Even if we're not working on a project, our day-to-day work is secured in a repository." [Hunt and Thomas, page 88]

If you work with one or more people on the same project ... Use version control.

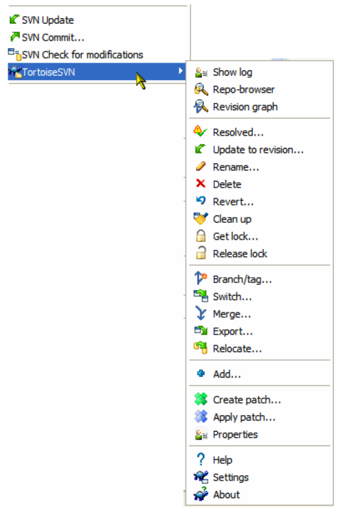
If you write software libraries or applications that other people use ... Use version control.

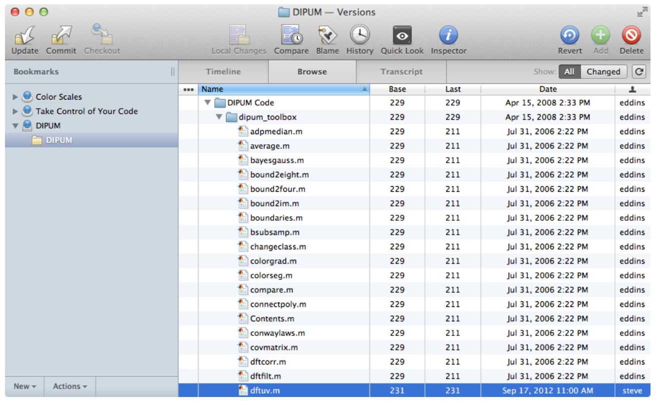
Even if you're working by yourself on "throw-away" code ... Use version control!

Use Subversion

Use Subversion for your version control system.
Subversion is a modern, free, actively maintained, open-source version control system. It works well, and it has easy-to-use clients on all major platforms. The Windows installer, which I have tried, couldn't be easier. There are several good books about Subversion, including one that's free. And there are several commercial and free repository hosting providers.
Compared to CVS (an older, widely used system), Subversion handles directories, file renaming, branching, and binary files more robustly and efficiently.
Obtain Subversion from:
subversion.tigris.org
Other version control system choices:
 CVS
 Commercial offerings such as Perforce
There are many others. For a comprehensive list, see:
en.wikipedia.org/wiki/List_of_revision_control_software
For a comparison of different version control systems, see:
en.wikipedia.org/wiki/Comparison_of_revision_control_software

Clients: TortoiseSVN on Windows, Versions on Mac OS X

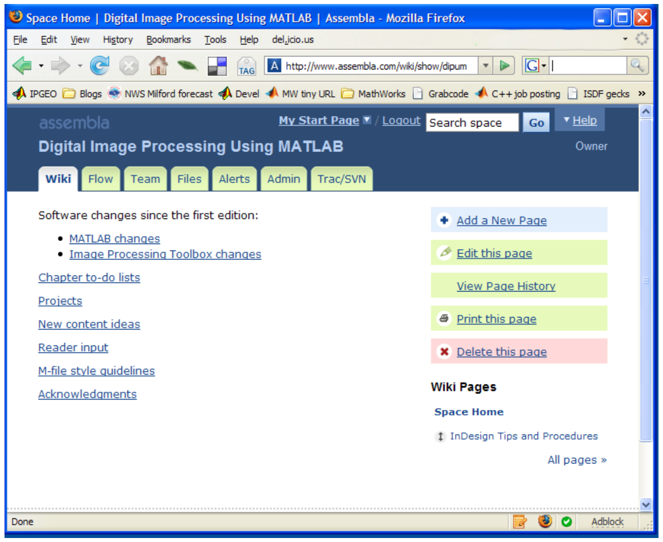




Name	Base	Last	Date	
DIPUM Code	229	229	Apr 15, 2008 2:33 PM	eddins
dipum_toolbox	229	229	Apr 15, 2008 2:33 PM	eddins
adpmedian.m	229	211	Jul 31, 2006 2:22 PM	eddins
average.m	229	211	Jul 31, 2006 2:22 PM	eddins
bayesgauss.m	229	211	Jul 31, 2006 2:22 PM	eddins
bound2eight.m	229	211	Jul 31, 2006 2:22 PM	eddins
bound2four.m	229	211	Jul 31, 2006 2:22 PM	eddins
bound2im.m	229	211	Jul 31, 2006 2:22 PM	eddins
boundaries.m	229	211	Jul 31, 2006 2:22 PM	eddins
bsubsamp.m	229	211	Jul 31, 2006 2:22 PM	eddins
changeclass.m	229	211	Jul 31, 2006 2:22 PM	eddins
colorgrad.m	229	211	Jul 31, 2006 2:22 PM	eddins
colorseg.m	229	211	Jul 31, 2006 2:22 PM	eddins
compare.m	229	211	Jul 31, 2006 2:22 PM	eddins
connectpoly.m	229	211	Jul 31, 2006 2:22 PM	eddins
Contents.m	229	211	Jul 31, 2006 2:22 PM	eddins
conwaylaws.m	229	211	Jul 31, 2006 2:22 PM	eddins
covmatrix.m	229	211	Jul 31, 2006 2:22 PM	eddins
dftcorr.m	229	211	Jul 31, 2006 2:22 PM	eddins
dftfilt.m	229	211	Jul 31, 2006 2:22 PM	eddins
dftuv.m	231	231	Sep 17, 2012 11:00 AM	steve

On all platforms, you can interact with a Subversion repository using the command-line interface. If command-line interfaces make you happy, stop here.
Easy-to-use tools are available on all major platforms.
On Windows, I like TortoiseSVN because it integrates into the Windows Explorer. I've been using TortoiseSVN in my demos. It is available from: tortoisesvn.tigris.org
On Mac OS X, try Versions. (Note: Versions isn't free. A single-user license costs $59 as of 17-Sep-2012.)

Use Web-Based Repository Providers


Space Home | Digital Image Processing Using MATLAB | Assembla - Mozilla Firefox
File Edit View History Bookmarks Tools Help del.icio.us
http://www.assembla.com/wiki/show/dipum
IPGEO Blogs NWS Milford forecast Devel MW tiny URL MathWorks Grabcode C++ job posting ISDF gecks
assembla
My Start Page / Logout
Search space
Go
Help
Owner
Wiki Flow Team Files Alerts Admin Trac/SVN
Add a New Page
Edit this page
View Page History
Print this page
Delete this page
Wiki Pages
Space Home
InDesign Tips and Procedures
All pages »
Done
Adblock

Digital Image Processing Using MATLAB

Software changes since the first edition:

- MATLAB changes
- Image Processing Toolbox changes

Chapter to-do lists

Projects

New content ideas

Reader input

M-file style guidelines

Acknowledgments

Not everyone knows how to set up a publicly accessible Web server that's configured properly to host a Subversion repository. Also, you'd need to learn some administrative details, such as how to control who has read and/or write access to the repository. You'd also have to worry about doing regular backups.
Don't let these concerns stop you. There are services out there that will set everything up for you. Some are commercial, some are free.

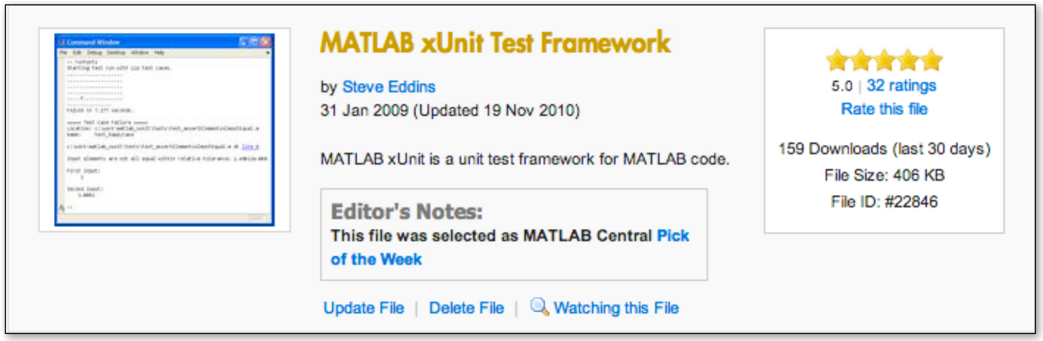
For the book Digital Image Processing Using MATLAB, my coauthors and I have both hosted-projects.com and Assembla.com.

Try a Google search on "subversion hosting" or "software project hosting."

Agenda

1. Version control
2. **Code quality from the outside – unit testing**
3. Code quality from the inside – bad code smells
4. Putting it all together – code refactoring

Code Quality from the Outside — Unit Testing



MATLAB xUnit Test Framework

by Steve Eddins
31 Jan 2009 (Updated 19 Nov 2010)

MATLAB xUnit is a unit test framework for MATLAB code.

Editor's Notes:
This file was selected as MATLAB Central Pick of the Week

Update File | Delete File | Watching this File

5.0 | 32 ratings
Rate this file

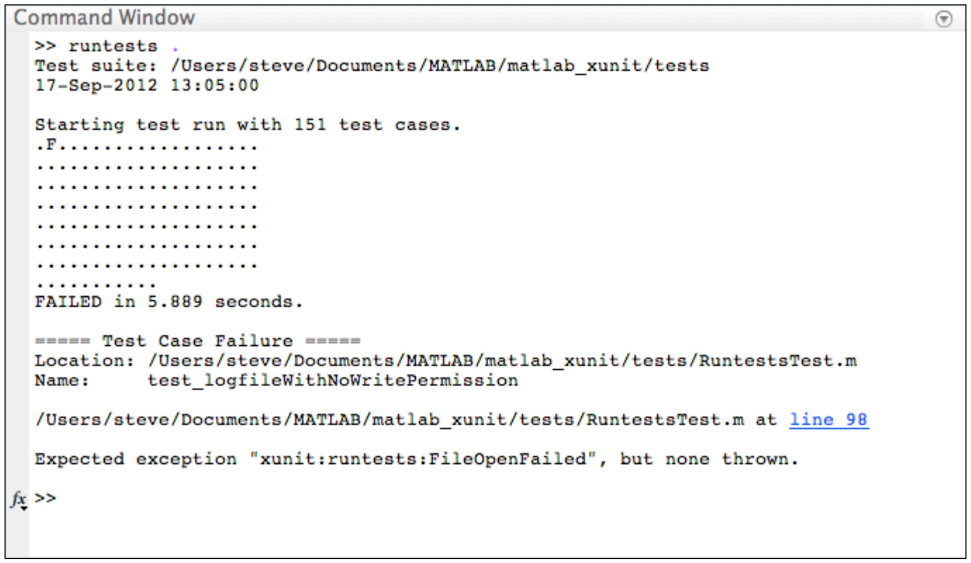
159 Downloads (last 30 days)
File Size: 406 KB
File ID: #22846

“Unit testing” is testing small units of code in isolation. “Small units” can be individual functions or methods, or they can be modules or classes.

“Automated” means, in this context, that all tests can be run with a single command or action, and that the result (pass or fail) is determined automatically, rather than by human inspection of program output.

For individual software developers working without the benefit of a quality assurance group, automated unit testing is the most practical form of testing that they can do on their own.

Writing and Running Tests



Command Window

```
>> runtests
Test suite: /Users/steve/Documents/MATLAB/matlab_xunit/tests
17-Sep-2012 13:05:00

Starting test run with 151 test cases.
.F..................
....................
....................
....................
....................
....................
....................
...........
FAILED in 5.889 seconds.

===== Test Case Failure =====
Location: /Users/steve/Documents/MATLAB/matlab_xunit/tests/RuntestsTest.m
Name:     test_logfileWithNoWritePermission

/Users/steve/Documents/MATLAB/matlab_xunit/tests/RuntestsTest.m at line 98

Expected exception "xunit:runtests:FileOpenFailed", but none thrown.

>>
```

Key elements of a test file containing multiple test cases:

```
function test_suite = testFliplr
initTestSuite;

function testFliplrMatrix
in = magic(3);
assertEqual(fliplr(in), in(:, [3 2 1]));

function testFliplrVector
assertEqual(fliplr([1 4 10]), [10 4 1]);
```

To run all test cases found in the current directory:

```
>> runtests
```

To run all test cases found in a particular file:

```
>> run(test_foobar)
```

Comparing Floating-Point Values

Does 0.1 + 0.2 + 0.3 equal 0.6?

[Continued from previous slide]

Because the order of operations matters, two mathematically equivalent computational procedures can produce different answers when implemented using floating-point arithmetic. Even identical code can produce different answers on different computers, or when compiled using different compilers, because of optimization variations or differing versions of underlying runtime libraries, such as the basic linear algebra subprograms (BLAS).For testing, then, we should usually avoid exact equality tests when checking floating-point results and instead use some sort of tolerance. There are two types of floating-point tolerance tests commonly used: absolute and relative. Absolute tolerance tests for two values a and b:

$| a - b | \leq T$,

where T is the tolerance. The relative tolerance test is

$|a-b|/\max(|a|,|b|) \leq T$

To understand the difference between these tests, consider the values 10.1 and 11.2 and the tolerance 0.001. The absolute difference between these values is 1.1; given the specified tolerance, they clearly fail the absolute tolerance test. Now, consider the values 134,712.5 and 134,713.6. Their absolute difference is also 1.1, so they also fail the absolute tolerance test. However, they pass the relative tolerance test […].

Specifying a relative tolerance of 10^(-n) is roughly equivalent to saying you expect two values to differ by no more than one unit in the n-th significant digit.

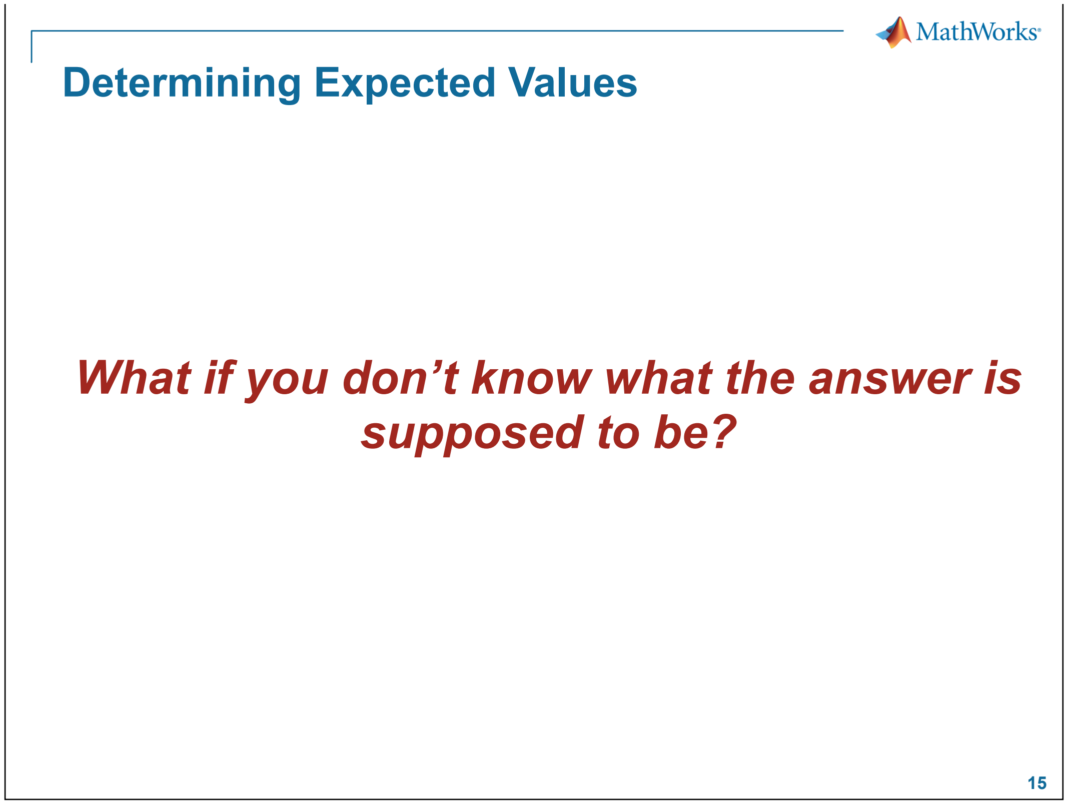
When you compare two vectors, as opposed to two scalars, you should consider whether to apply the tolerance test to each vector element independently or whether to apply it using a vector norm. For example, suppose you compared the two vectors [1 100,000] and [2 100,000] using a relative tolerance of 0.001. If you compare the two vectors elementwise, they clearly fail the tolerance test because the relative difference between 1 and 2 is much higher than the tolerance value. However, another way to compare vectors is using a vector norm, such as L2 […] The two vectors [1 100,000] and [2 100,000] pass this form of relative tolerance test because they differ in the L2-norm sense by only about 1 part in 100,000.

To compare floating-point values, MATLABxUnit provides the functions assertElementsAlmostEqual and assertVectorsAlmostEqual, either of which can use a relative or an absolute tolerance.

When choosing a tolerance, it helps to consider the machine precision, or ε. This quantity is the spacing of floating-point numbers between successive powers of two, relative to the lower power of two. For example, the next floating-point number higher than 1 is 1 + ε, while the next floating-point number higher than 2 is 2 + 2ε.

The MATLAB function eps returns the machine precision. For double-precision floating point, ε is approximately 2.2 · 10^(−16), and for single precision it's approximately 1.2 · 10^(−7).

For computations involving relatively little floating-point arithmetic, a small multiple of εmight be appropriate, such as 100ε. For computations involving coarse approximations, we might use much higher tolerances. The default relative tolerance for the MATLAB xUnit assertion functions is approximately 10^(−8), corresponding to eight significant digits. In general, your choice of tolerance depends on the particular domain and the nature of the your models and approximations.



Determining Expected Values

What if you don’t know what the answer is supposed to be?

The notes below are from S. L. Eddins, “Automated Software Testing for MATLAB,” Computing in Science and Engineering, November/December 2009, pp. 48-54.

As a practical matter, to deal with testing complex models, I recommend that you start by focusing on the “unit” in “unit test.” Even complex simulations comprise simpler computational units that you can individually verify. Verifying individual computational units’ behavior is a helpful step toward verifying overall system behavior. Focusing on the testability of small computational units also encourages the development of modular, more maintainable code.

When it’s infeasible to hand compute expected results—even for small computational units—there are several strategies you can try.

Compare with Alternate Computation Method

The MATLAB fft function computes the discrete Fourier transform (DFT) using a fast algorithm. Some fft test cases compare the function’s output against the output computed directly using the standard DFT mathematical equation. The direct method’s relative slowness doesn’t matter for testing purposes.

Agenda

1. Version control
2. Code quality from the outside – unit testing
3. **Code quality from the inside – bad code smells**
4. Putting it all together – code refactoring

Code Quality — Fundamental Principles

Communicate with people, not computers

Minimize complexity

What a challenging topic! It could be an entire course of study. How can we usefully spend a few minutes talking about it?
Here are a few ideas that are widely accepted, effective, practical, and easy to apply:
 Learn to identify bad code smells
 Keep function complexity down
 Refactor your code regularly
These concepts and techniques can be learned and applied individually, but they also reinforce each other.
Along the way, remember these two fundamental principles that underlie most techniques for assessing and improving the quality of code:
 Write code to communicate effectively with people, not computers
 Write and organize code to minimize complexity
When someone tells you a “rule” for writing code, evaluate it against these principles.

Bad Code Smells

- *Duplication*
- *Complex functions*
- *Comments as deodorant*

A bad code smell is a characteristic of code that causes an experienced programmer to pause, wrinkle his or her nose, and think, “There’s a good chance something bad is going to happen here.” The odor metaphor resonates strongly because smell is such a powerful memory cue.
Fowler’s Refactoring includes a catalog of smells in Chapter 3. Read them. Find two or three that make sense to you and learn to sniff for them.
I’ve picked three of the most widely applicable smells to discuss here:
“Don’t Repeat Yourself”
“Complex Functions”
“Comments”

Don’t Repeat Yourself

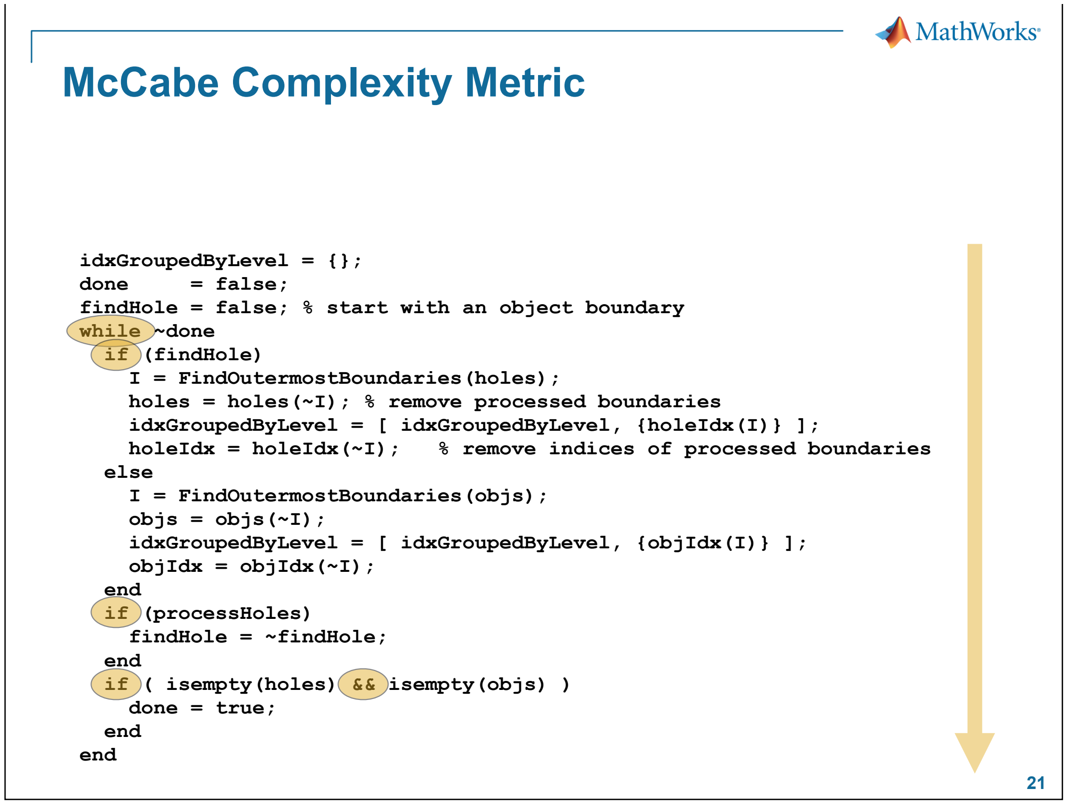
Duplicated code:
“Number one in the stink parade”

Duplicated code is “number one in the stink parade.” [Fowler, page 76] This is such a common theme in the software literature that “don’t repeat yourself” has its own acronym (DRY).
Why? Because anything repeated in two or more places will eventually be wrong in at least one.
Meaningful differences can’t be easily distinguished from accidental ones.

Measuring and Reducing Function Complexity

McCabe complexity - the "magic metric"

We move now from the qualitative to the quantitative. The McCabe complexity (also called cyclomatic complexity) metric assigns a positive integer to each routine (function or method) in a program.
I call it the magic metric because it is:
 Easy to compute
 Easy to understand
 Independent of programming language
 Widely accepted
 Well correlated with program quality
From the Carnegie Mellon Software Engineering Institute (SEI): "Cyclomatic complexity is the most widely used member of a class of static software metrics. Cyclomatic complexity may be considered a broad measure of soundness and confidence for a program. Introduced by Thomas McCabe in 1976, it measures the number of linearly-independent paths through a program module."
www.sei.cmu.edu/str/descriptions/cyclomatic_body.html

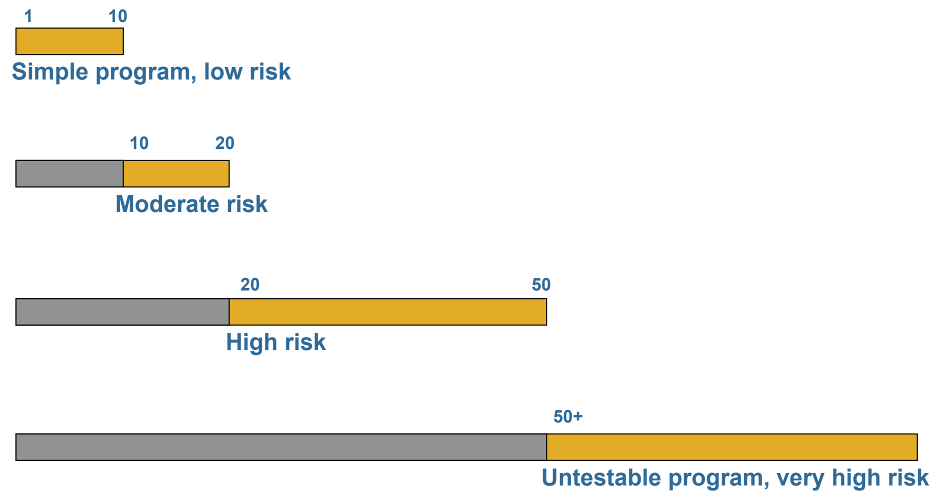


McCabe Complexity Metric

```
idxGroupedByLevel = {};
done      = false;
findHole = false; % start with an object boundary
while ~done
  if (findHole)
    I = FindOutermostBoundaries(holes);
    holes = holes(~I); % remove processed boundaries
    idxGroupedByLevel = [ idxGroupedByLevel, {holeIdx(I)} ];
    holeIdx = holeIdx(~I);   % remove indices of processed boundaries
  else
    I = FindOutermostBoundaries(objs);
    objs = objs(~I);
    idxGroupedByLevel = [ idxGroupedByLevel, {objIdx(I)} ];
    objIdx = objIdx(~I);
  end
  if (processHoles)
    findHole = ~findHole;
  end
  if ( isempty(holes) && isempty(objs) )
    done = true;
  end
end
```

Computing cyclomatic complexity for a routine:
“Add one for the straight path through the routine.
Add one for each of the following keywords, or their equivalents: if while repeat for and or
Add one for each case in a [switch] statement.” [McConnell, page 458]
In MATLAB, don’t forget to count the elseif.
Cyclomatic complexity is easy to compute, but doing it manually gets old very fast. Fortunately, you can find tools that compute it for you.
In MATLAB –
checkcode –cyc foo
For other languages, some teams at The MathWorks use Source Monitor, a free tool that can compute software metrics for C++, C, C#, Java, Delphi, Visual Basic (VB6) or HTML.
www.campwoodsw.com/sourcemonitor.html

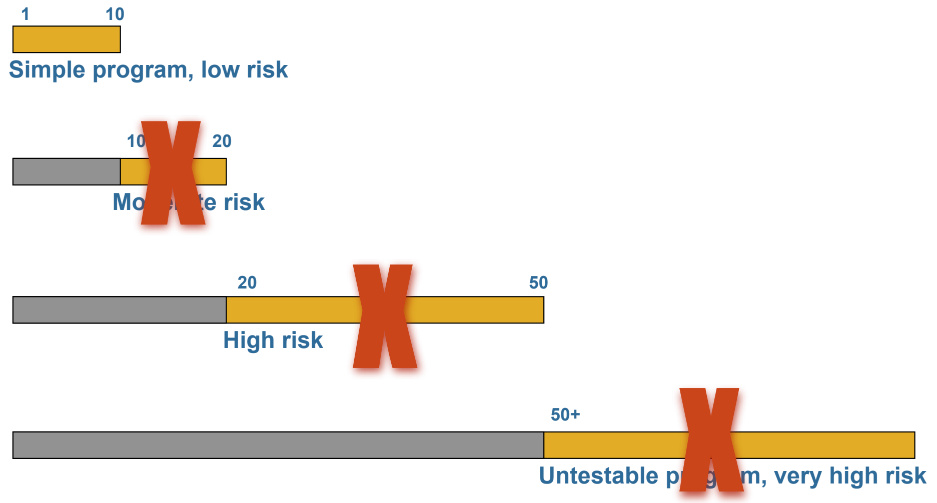
Complexity Correlates with Bug Risk



Source: Software Engineering Institute (SEI)
www.sei.cmu.edu/str/descriptions/cyclomatic.html
Studies:
McCabe, Tom. 1976. “A Complexity Measure.” IEEE Transactions on Software Engineering, SE-2, no. 4 (December): 308-20.
Shen, Vincent Y., et al. 1985. “Identifying Error-Prone Software – An Empirical Study.” IEEE Transactions on Software Engineering, SE-11, no. 4 (April): 317-24.
Ward, William T. 1989. “Software Defect Prevention Using McCabe’s Complexity Metric.” Hewlett-Packard Journal. April, 64-68.
The Ward study reported significantly higher program reliability produced by using the complexity metric at Hewlett-Packard.
At The MathWorks, we have ample anecdotal evidence that high complexity numbers are correlated with problematic code, and that reducing complexity numbers results in code that’s much easier to understand and maintain.
On the Image and Geospatial Team at MathWorks, our presubmission code checklist includes this:

> “Have you checked the McCabe complexity metric? If you're modifying existing code, try to keep the metric from going up for routines you touch. If you're creating new code, aim for the target of a complexity beneath 10 for each new routine.”

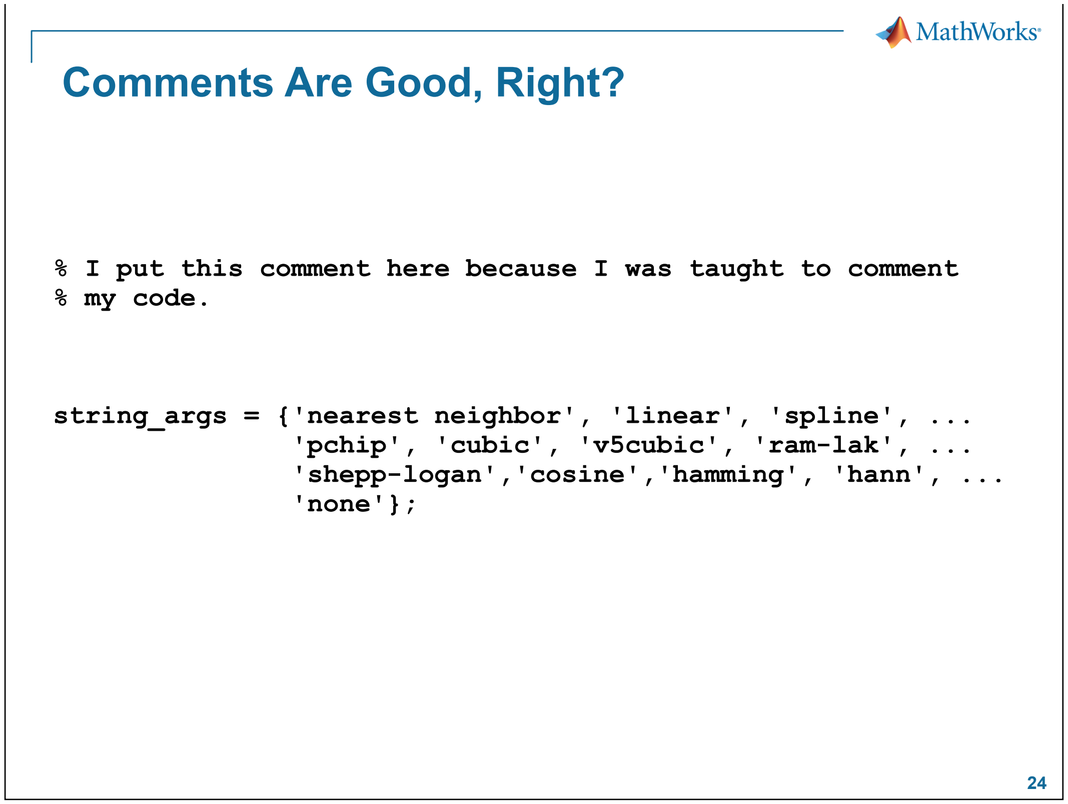
Set Coding Standards Based on Complexity



What coding rules should you follow on your project? If you had to pick just one, make it be this:
No routines allowed with a cyclomatic complexity higher than 10.

Do you supervise people who write code? Insist that they follow this rule.

Why so much focus on the ordinary routine? It is the fundamental unit of code organization in almost every widely-used programming language. Improve your routines, and you improve all of your code.

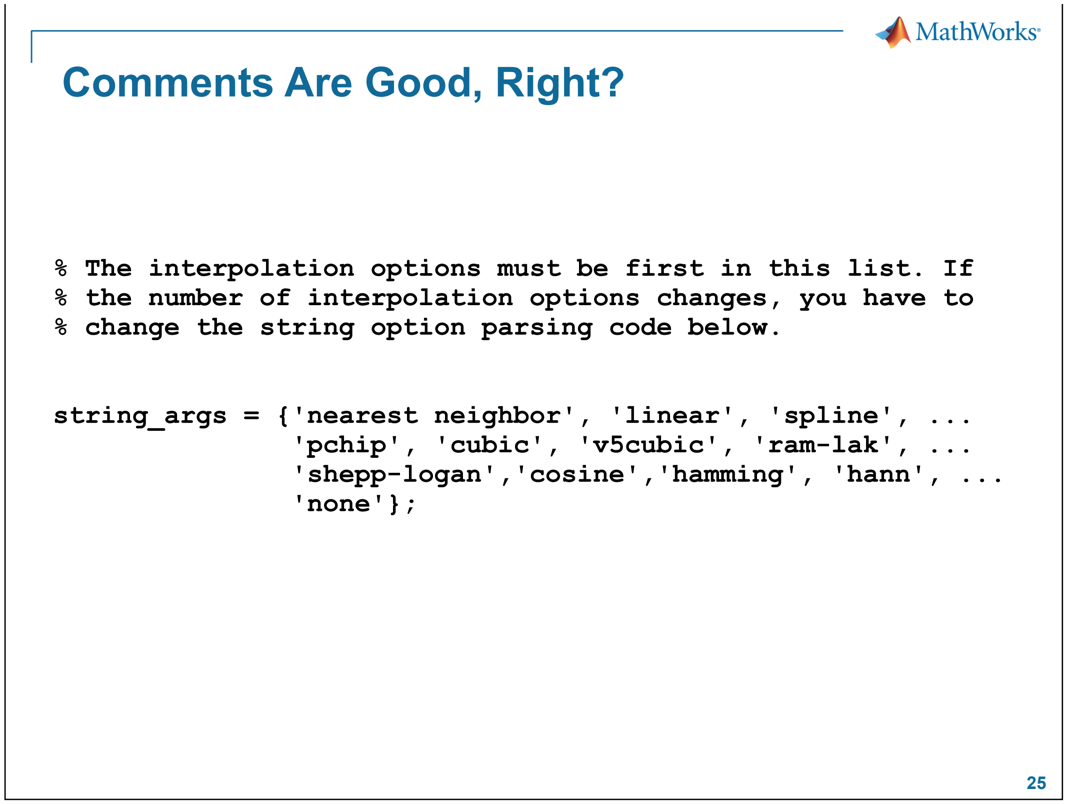


Comments Are Good, Right?

```
% I put this comment here because I was taught to comment
% my code.


string_args = {'nearest neighbor', 'linear', 'spline', ...
               'pchip', 'cubic', 'v5cubic', 'ram-lak', ...
               'shepp-logan','cosine','hamming', 'hann', ...
               'none'};
```

Fowler: “Don’t worry, we aren’t saying that people shouldn’t write comments. In our olfactory analogy, comments aren’t a bad smell; indeed they are a sweet smell. The reason we mention comments here is that comments often are used as a deodorant. It’s surprising how often you look at thickly commented code and notice that the comments are there because the code is bad.” [Fowler, page 87]
McConnell quotes Kernighan and Plauger: “Don’t document bad code – rewrite it.” [McConnell, page 568]

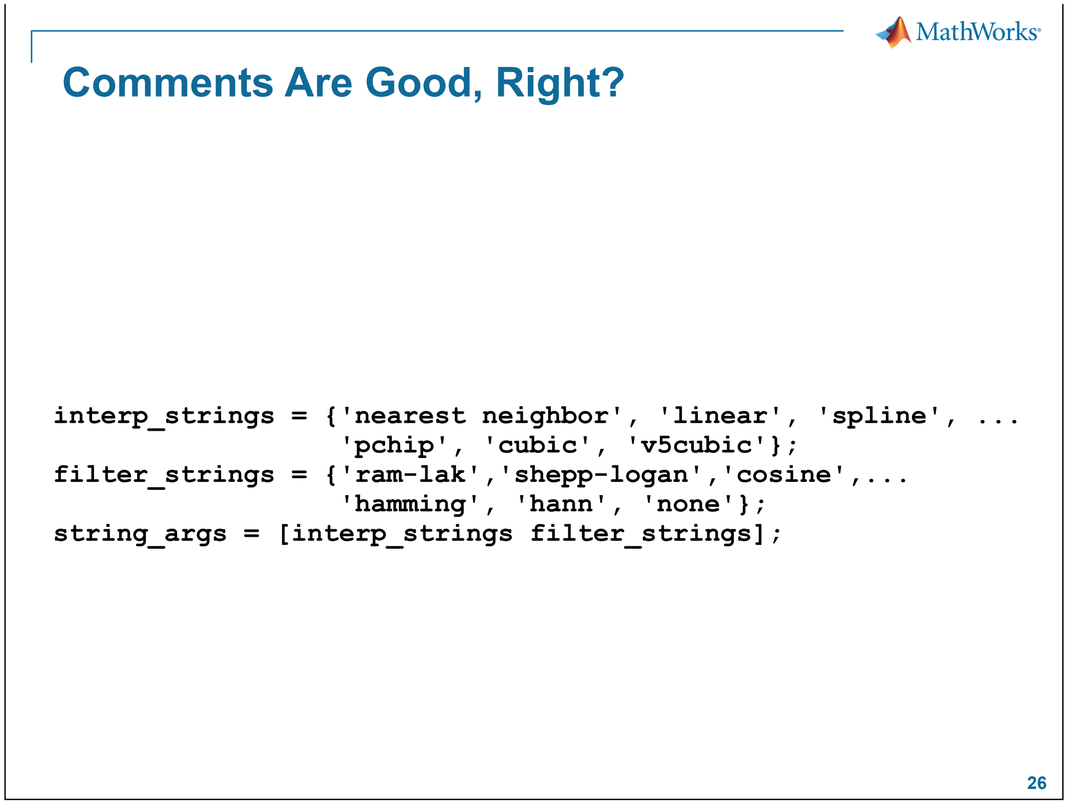


Comments Are Good, Right?

```
% The interpolation options must be first in this list. If
% the number of interpolation options changes, you have to
% change the string option parsing code below.


string_args = {'nearest neighbor', 'linear', 'spline', ...
               'pchip', 'cubic', 'v5cubic', 'ram-lak', ...
               'shepp-logan','cosine','hamming', 'hann', ...
               'none'};
```

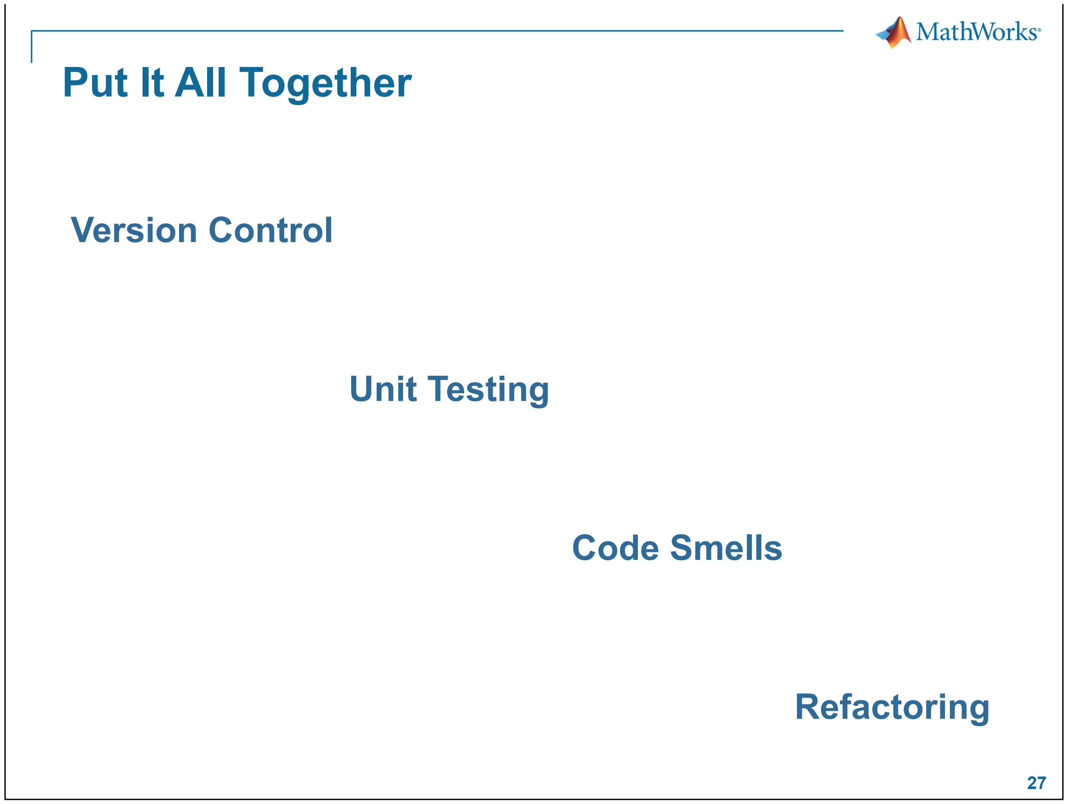
I wrote the comment above when revising the Image Processing Toolbox™ function iradon.m. It’s a classic example of using a comment as a deodorant to cover up the fact that the code is bad.



Comments Are Good, Right?

```
interp_strings = {'nearest neighbor', 'linear', 'spline', ...
                  'pchip', 'cubic', 'v5cubic'};
filter_strings = {'ram-lak','shepp-logan','cosine',...
                  'hamming', 'hann', 'none'};
string_args = [interp_strings filter_strings];
```

Now the code communicates the programmer’s intent. The comment is no longer necessary.



Put It All Together

Version Control

Unit Testing

Code Smells

Refactoring

Case studies:
scantiff.m – Refactoring my own code to keep function complexity under control
patchm.m – Refactoring someone else's code so that I can figure out how to change it safely.
Key refactoring tips:
 Make changes in very small, quick increments.
 Make sure the code still works after each change.
 Check in each small change to version control.
 Drive down function complexity.
 No behavior changes allowed!

General tip for handling "bad" code that someone else left in your lap: If you can't understand the code well enough to make the changes you need to make, start by refactoring the code, using small, cautious steps. Submit each incremental change to version control so that you can easily revert if things start to go wrong.

Code Refactoring

$$\int u\,dv \Rightarrow uv - \int v\,du$$

Do you remember integration by parts in calculus? We were taught that we could transform the expression on the left into the expression on the right without changing its meaning.

To refactor code is to transform it, to alter its structure, without changing its behavior.

Similar to the rule of integration by parts, there are several useful rules for transforming code without changing behavior.

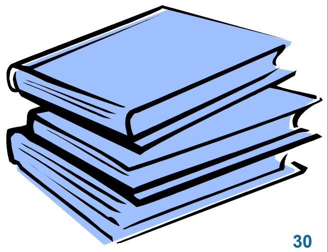
For a comprehensive catalog of refactoring rules, see Martin Fowler’s book Refactoring: Improving the Design of Existing Code.

Agenda

1. Version control
2. Code quality from the outside – unit testing
3. Code quality from the inside – bad code smells
4. Putting it all together – code refactoring

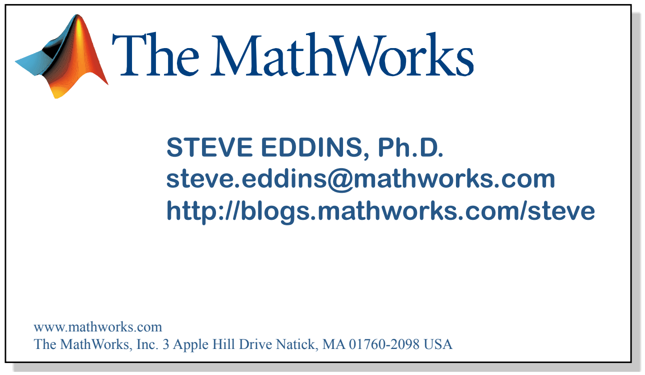
Recommended Reading

- McConnell, *Code Complete: A Practical Handbook of Software Construction*
- Hunt and Thomas, *The Pragmatic Programmer: From Journeyman to Master*
- Fowler, *Refactoring: Improving the Design of Existing Code*
- Wilson, Software Carpenty, software-carpentry.org



There are many excellent books about the craft of software development. I've chosen a very small number to recommend here. These books
 Are language- and environment-agnostic
 Can be read in part or in whole
 Have good bibliographies and recommended reading lists if you want to learn more
Also, these are the books I relied upon most heavily to prepare this presentation. All three are widely read and studied at The MathWorks.
 Steve McConnell. Code Complete: A Practical Handbook of Software Construction, 2nd edition. Microsoft Press. 2004.
 Andrew Hunt and David Thomas. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley. 2000.
 Martin Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley. 1999.

Also, take a look at the course Software Carpentry by Greg Wilson. Course materials are available online at http://www.swc.scipy.org/



The MathWorks

STEVE EDDINS, Ph.D.
steve.eddins@mathworks.com
http://blogs.mathworks.com/steve

www.mathworks.com
The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098 USA