Developing Production-Ready MATLAB Code

Marta Wilczkowiak

Production Applications in MATLAB

With MathWorks tools, the risk managers can develop algorithms and financial models, and the IT division can quickly deploy the applications. Because we can implement changes in our models and get them into production quickly, we can rapidly respond to new market data and conditions."

Peter W. Schweighofer
UniCredit Bank Austria

How MATLAB code goes in production?

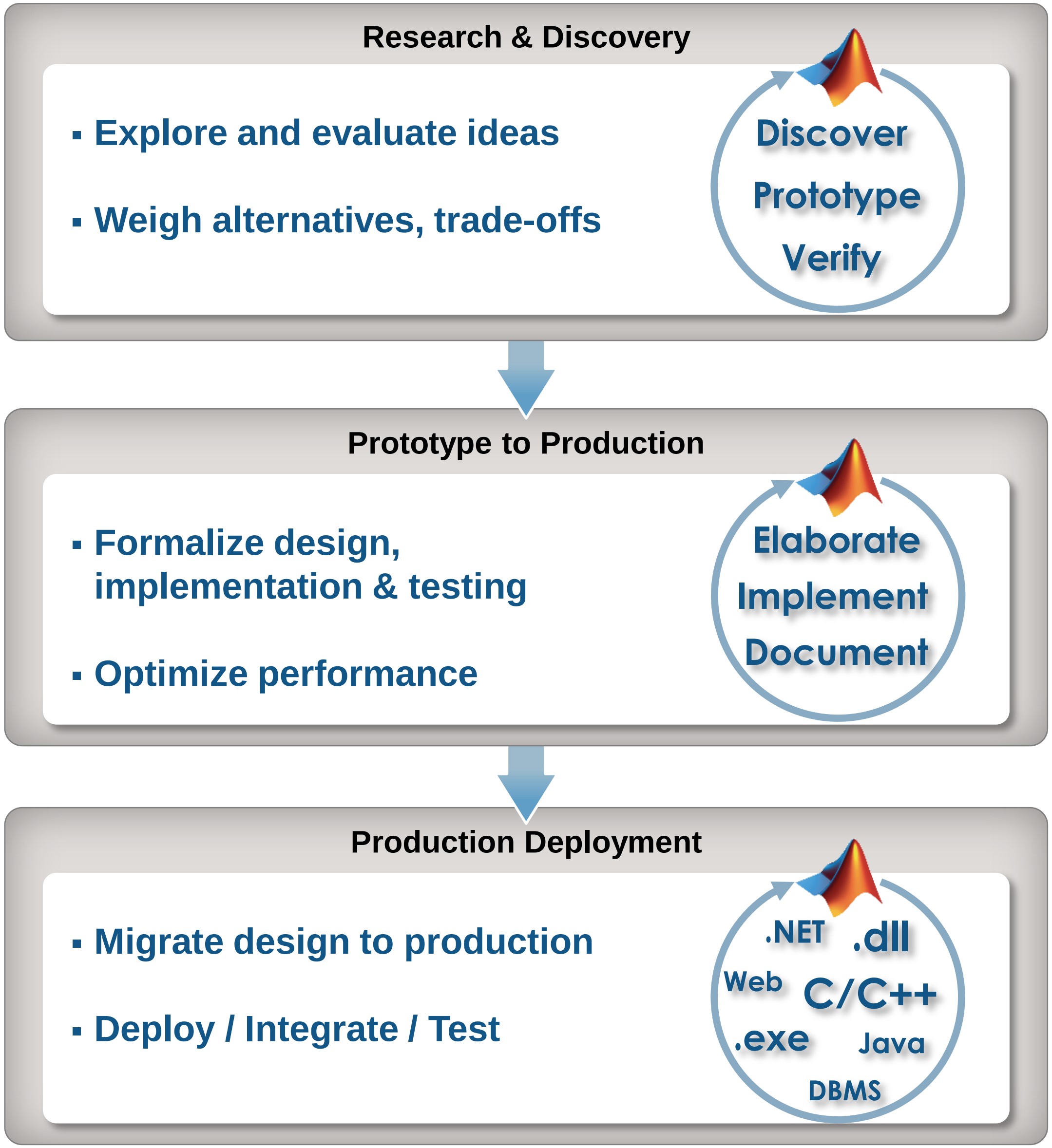
```
a = magic(n);
```

?

Production-Ready Code

- Collaboration
- Quality, e.g. FURPS (HP, IBM)
  - **F**unctionality
  - **U**sability
  - **R**eliability
  - **P**erformance
  - **S**upportability
- Integration

Application Development Process



Research & Discovery

- Explore and evaluate ideas
- Weigh alternatives, trade-offs

Discover

Prototype

Verify

Prototype to Production

- Formalize design, implementation & testing
- Optimize performance

Elaborate

Implement

Document

Production Deployment

- Migrate design to production
- Deploy / Integrate / Test

.NET .dll

Web C/C++

.exe Java

DBMS

Quality and Collaboration across the Development Process

- Architecture & Design
- Implementation
- Performance
- Testing

Design for…

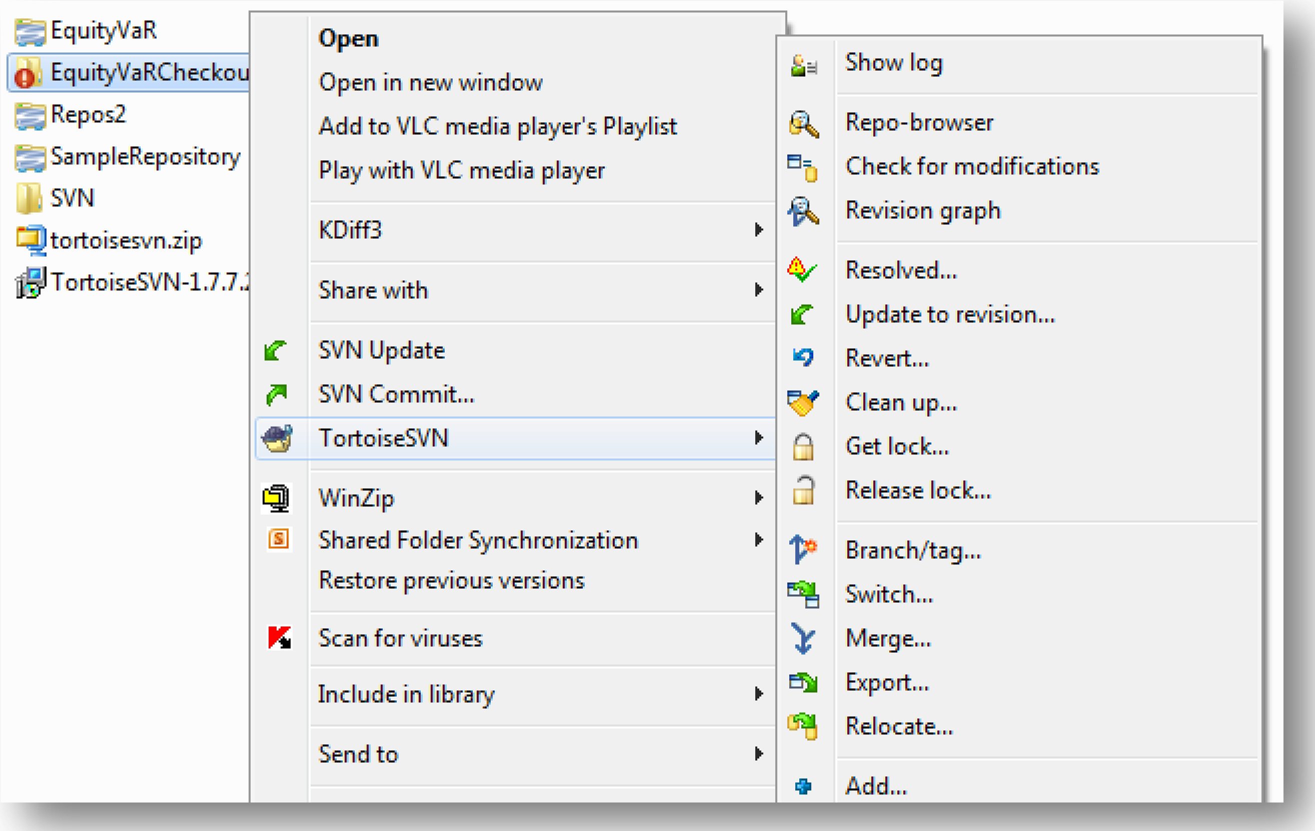
- Modularity
- High cohesion, low coupling

Source control

© Alexandre Buisse / Caters

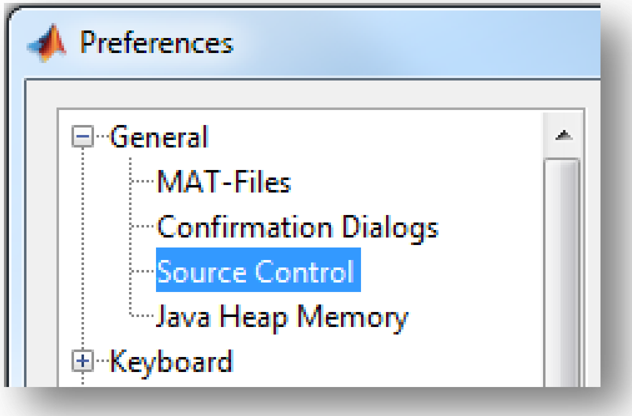
Source Control with MATLAB

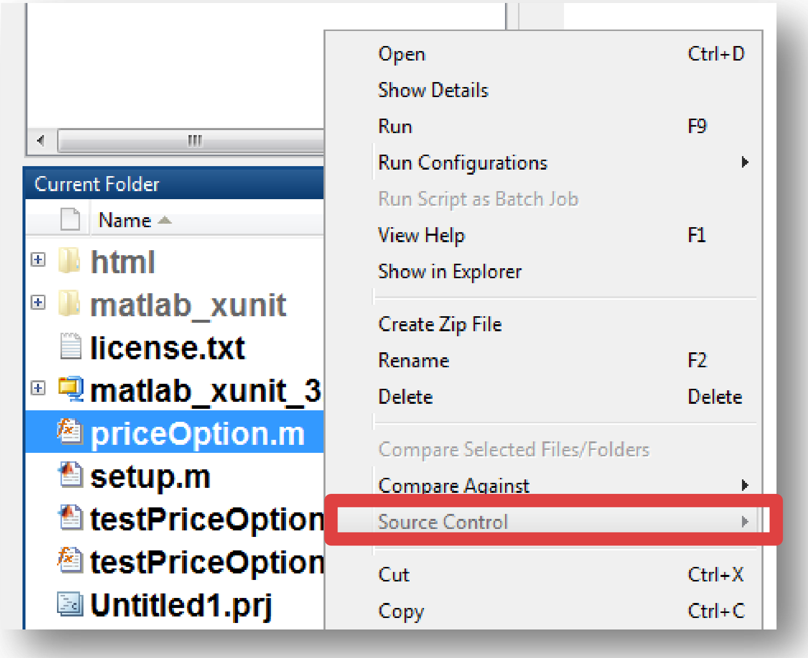
- Any source control solution adequate
- Systems integrate directly into OS shell



Source Control with MATLAB

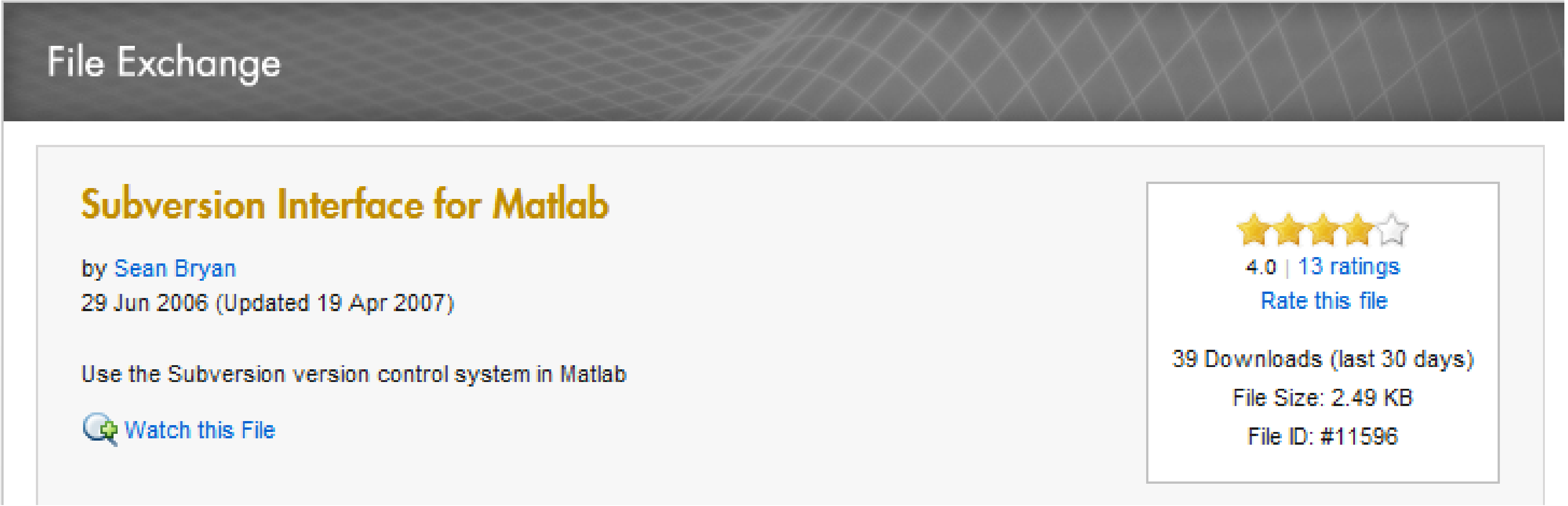
- Any source control solution adequate
- Systems integrate directly into OS shell
- MATLAB MSSCC interface (`verctrl, checkin, checkout`)





Source Control with MATLAB

- Any source control solution adequate
- Systems integrate directly into OS shell
- MATLAB MSSCC interface (`verctrl, checkin, checkout`)
- MATLAB Central File Exchange



File Exchange

Subversion Interface for Matlab

by Sean Bryan
29 Jun 2006 (Updated 19 Apr 2007)

Use the Subversion version control system in Matlab

4.0 | 13 ratings

39 Downloads (last 30 days)

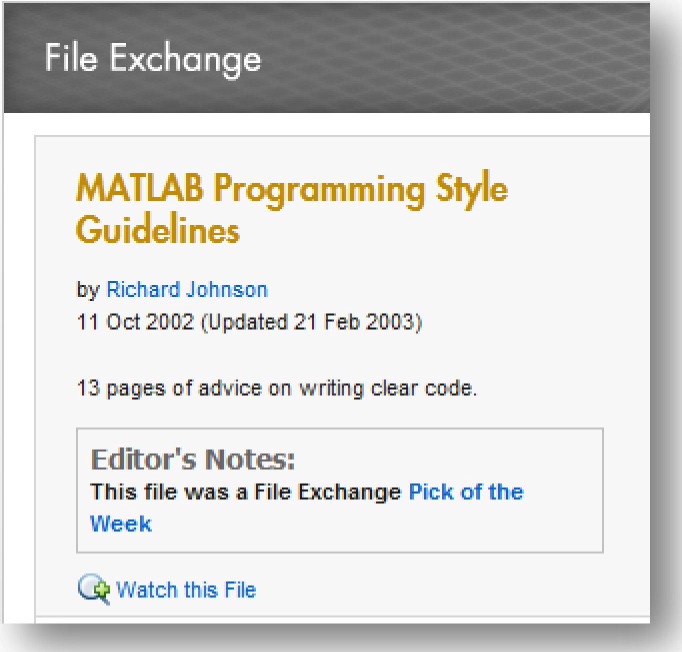
File Size: 2.49 KB

File ID: #11596

Syntax, Safety and Style

Barcelona chair, Ludwig Mies van der Rohe

Style



File Exchange

MATLAB Programming Style Guidelines

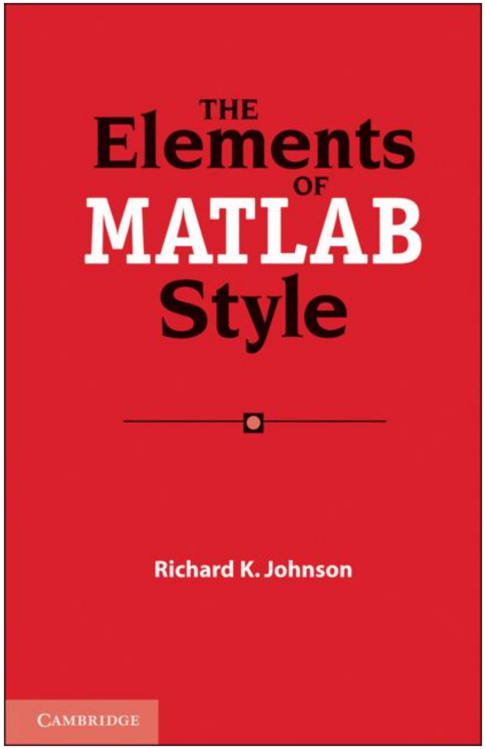
by Richard Johnson

11 Oct 2002 (Updated 21 Feb 2003)

13 pages of advice on writing clear code.

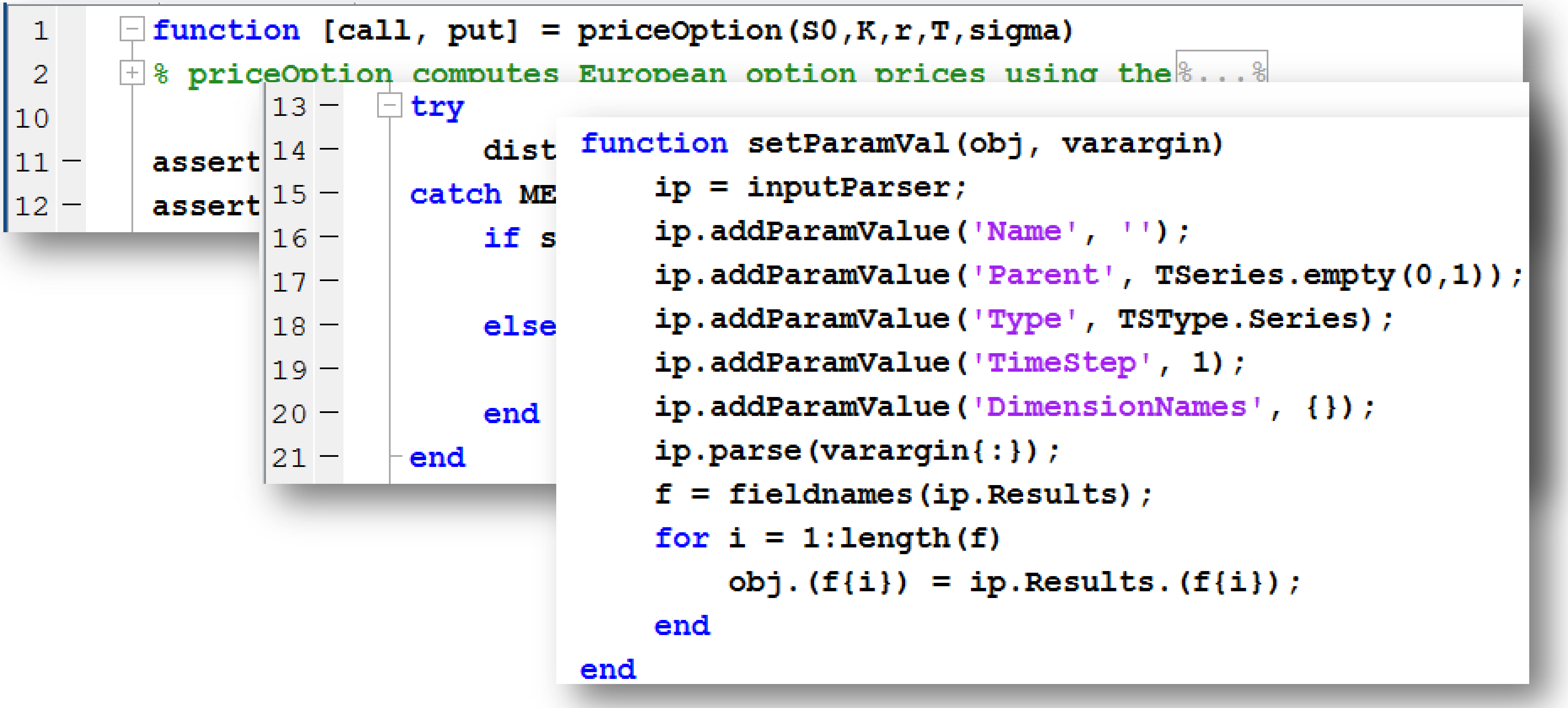
Editor's Notes:
This file was a File Exchange Pick of the Week

Watch this File



Safety (I/0)

- Assertions, try/catch, MException, InputParser



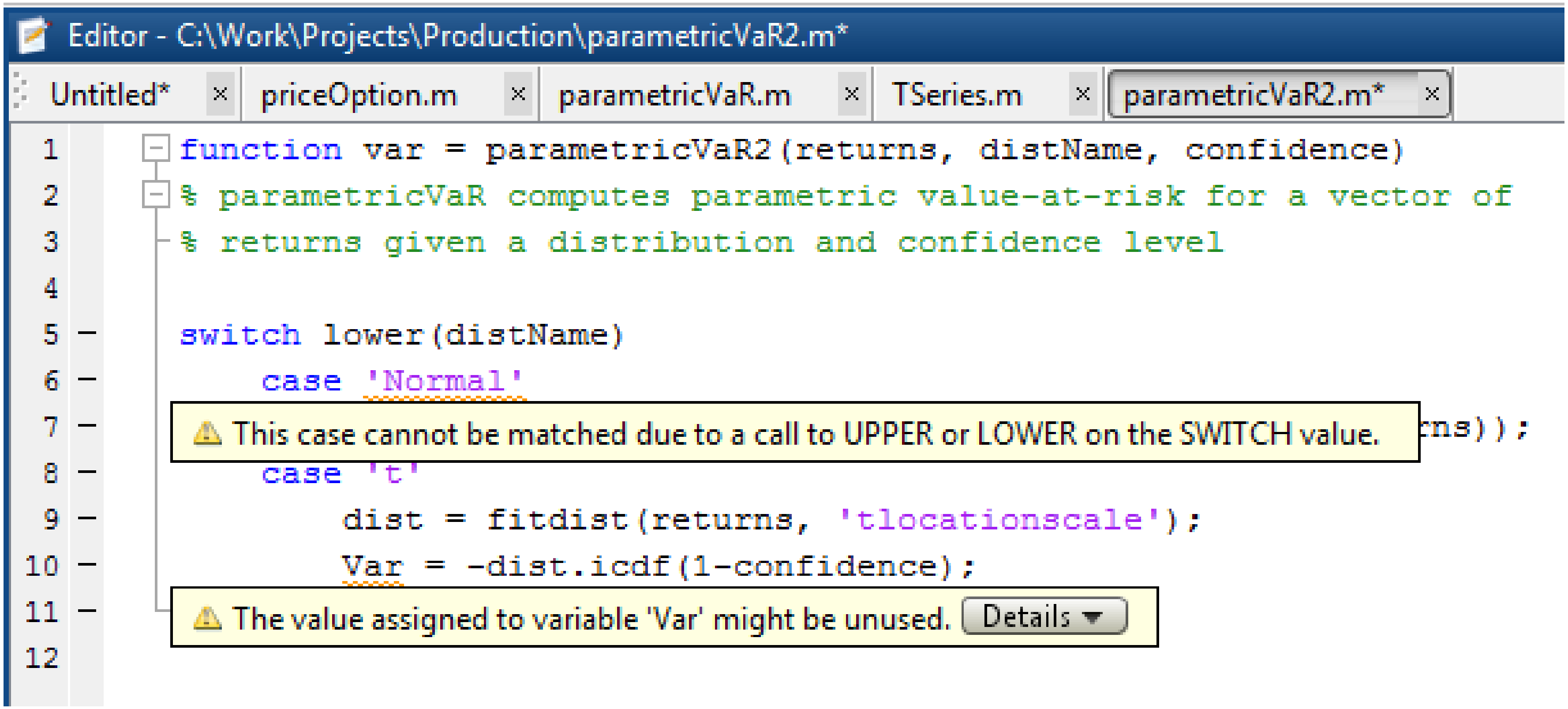
```
function [call, put] = priceOption(S0,K,r,T,sigma)
% priceOption computes European option prices using the%...%

assert
assert
```

```
try
    dist
catch ME
    if s

    else

    end
end
```

```
function setParamVal(obj, varargin)
    ip = inputParser;
    ip.addParamValue('Name', '');
    ip.addParamValue('Parent', TSeries.empty(0,1));
    ip.addParamValue('Type', TSType.Series);
    ip.addParamValue('TimeStep', 1);
    ip.addParamValue('DimensionNames', {});
    ip.parse(varargin{:});
    f = fieldnames(ip.Results);
    for i = 1:length(f)
        obj.(f{i}) = ip.Results.(f{i});
    end
end
```

Syntax

Static Code Analyzer



```
function var = parametricVaR2(returns, distName, confidence)
% parametricVaR computes parametric value-at-risk for a vector of
% returns given a distribution and confidence level

switch lower(distName)
    case 'Normal'
                                                            rns));
    case 't'
        dist = fitdist(returns, 'tlocationscale');
        Var = -dist.icdf(1-confidence);
```

⚠ This case cannot be matched due to a call to UPPER or LOWER on the SWITCH value.

⚠ The value assigned to variable 'Var' might be unused.

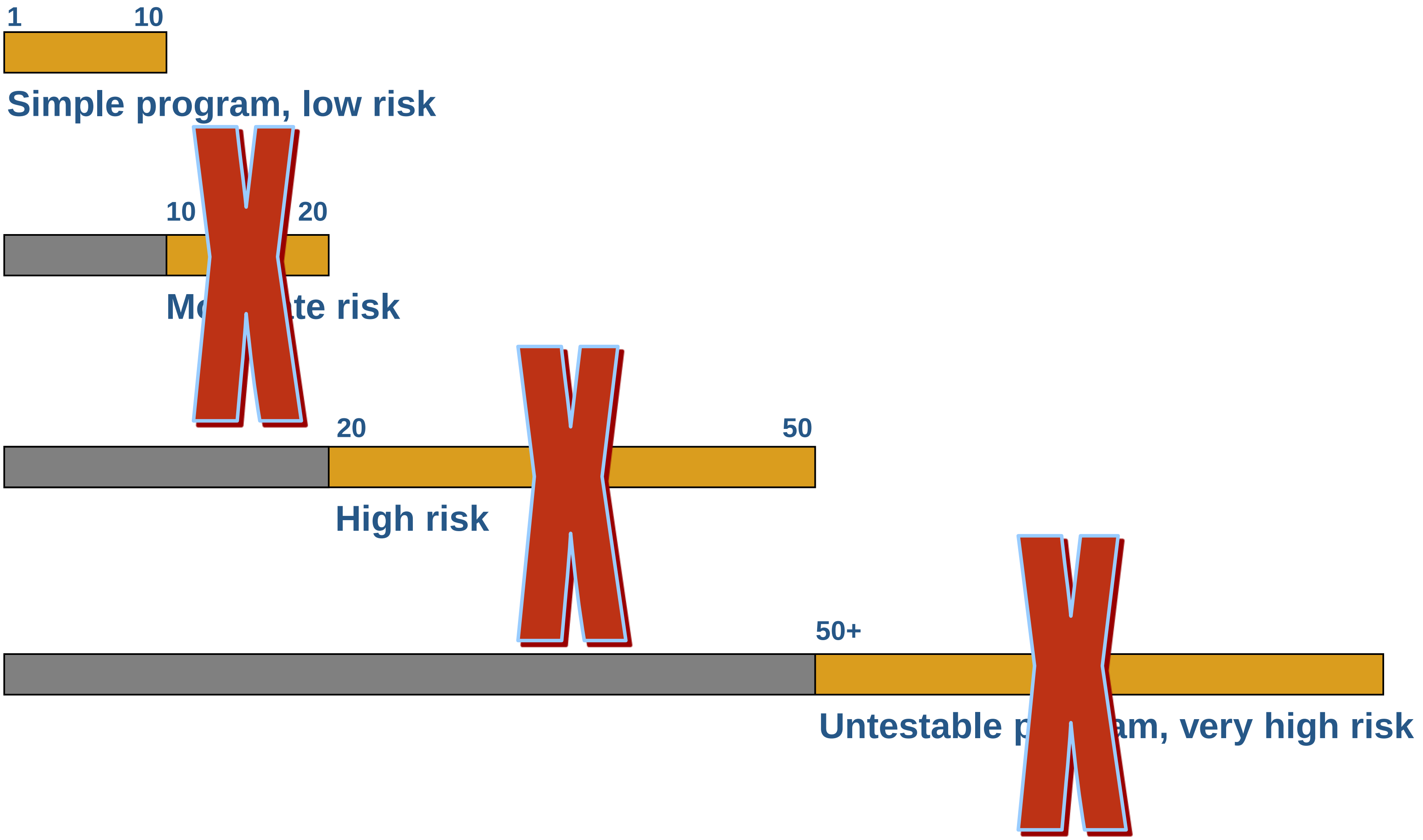
Bad code "smells"

- Duplicated code
- Over-commented code
- Overly complex logic flow

McCabe Cyclomatic Complexity

```
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);
    idxGroupedByLevel = [ idxGroupedByLevel, {objIdx(I)} ];
``end
  if (processHoles)
    findHole = ~findHole;
  end
  if ( isempty(holes) && isempty(objs) )
    done = true;
  end
end
```

McCabe Cyclomatic Complexity



1 10

Simple program, low risk

10 20

Moderate risk

20 50

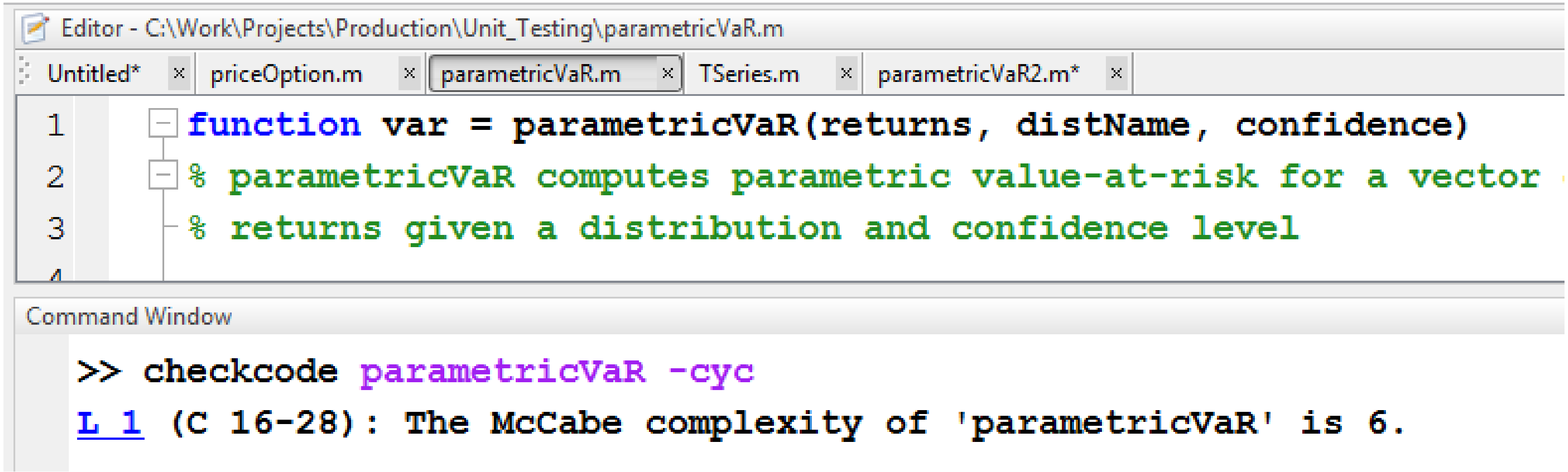
High risk

50+

Untestable program, very high risk

Measure & Improve Code Quality in MATLAB

McCabe Complexity (`checkcode -cyc`)



```
function var = parametricVaR(returns, distName, confidence)
% parametricVaR computes parametric value-at-risk for a vector
% returns given a distribution and confidence level
```

```
>> checkcode parametricVaR -cyc
L 1 (C 16-28): The McCabe complexity of 'parametricVaR' is 6.
```

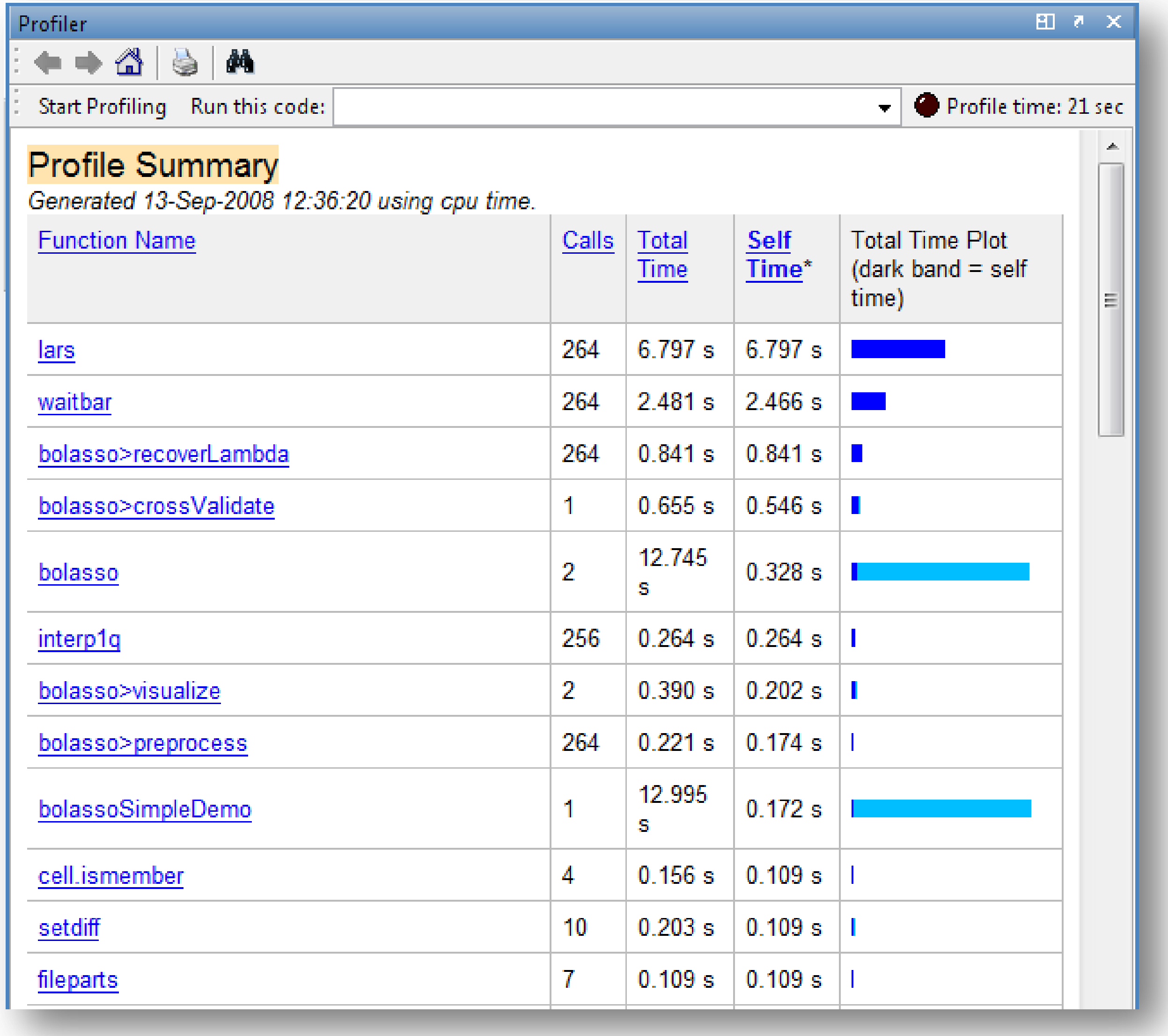
Services

The MathWorks
Training Services

The MathWorks
Consulting Services

Performance Tuning

Performance Profiling



Profile Summary

Generated 13-Sep-2008 12:36:20 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
lars	264	6.797 s	6.797 s	
waitbar	264	2.481 s	2.466 s	
bolasso>recoverLambda	264	0.841 s	0.841 s	
bolasso>crossValidate	1	0.655 s	0.546 s	
bolasso	2	12.745 s	0.328 s	
interp1q	256	0.264 s	0.264 s	
bolasso>visualize	2	0.390 s	0.202 s	
bolasso>preprocess	264	0.221 s	0.174 s	
bolassoSimpleDemo	1	12.995 s	0.172 s	
cell.ismember	4	0.156 s	0.109 s	
setdiff	10	0.203 s	0.109 s	
fileparts	7	0.109 s	0.109 s	

Acceleration Strategies Applied in MATLAB

Option	Technology / Product
1. Best Practices in MATLAB Programming ▪ Improve code (e.g. vectorize) ▪ Identify bottlenecks (e.g., Profiler, Code Analyzer)	▪ MATLAB
2. Better Algorithms ▪ Explore alternative approaches ▪ Leverage built-in optimized libraries	▪ Toolboxes ▪ System Toolboxes
3. More Processors, Cores, or GPUs ▪ Utilize high level parallel constructs (e.g. **parfor**) ▪ Scale to clusters, clouds, and grids	▪ Parallel Computing Toolbox ▪ MATLAB Distributed Computing Server
4. Re-implement in Another Language ▪ Generate code (e.g., C, HDL, MEX) ▪ Run on FPGAs or DSPs	▪ MATLAB Coder ▪ HDL Coder

Remember

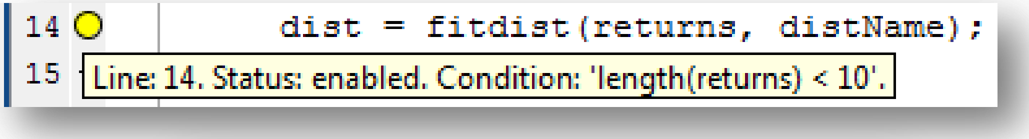
- Make it work
- Make it right
- Make it fast

Kent Beck

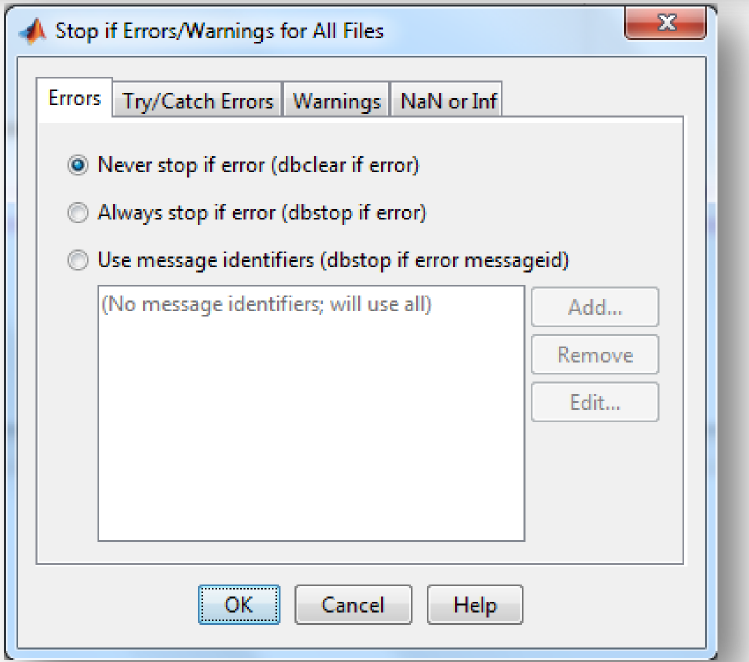
Effective Testing

Improve Code Robustness in MATLAB

Advanced debugging



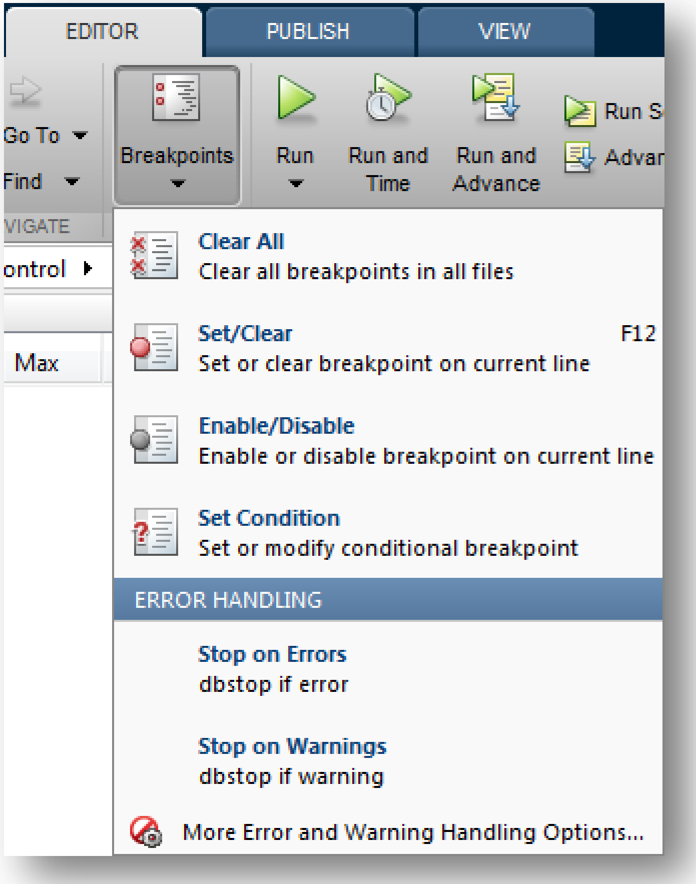
```
14 ○        dist = fitdist(returns, distName);
15 Line: 14. Status: enabled. Condition: 'length(returns) < 10'.
```



Stop if Errors/Warnings for All Files

- ● Never stop if error (dbclear if error)
- ○ Always stop if error (dbstop if error)
- ○ Use message identifiers (dbstop if error messageid)

(No message identifiers; will use all)



Clear All
Clear all breakpoints in all files

Set/Clear F12
Set or clear breakpoint on current line

Enable/Disable
Enable or disable breakpoint on current line

Set Condition
Set or modify conditional breakpoint

ERROR HANDLING

Stop on Errors
dbstop if error

Stop on Warnings
dbstop if warning

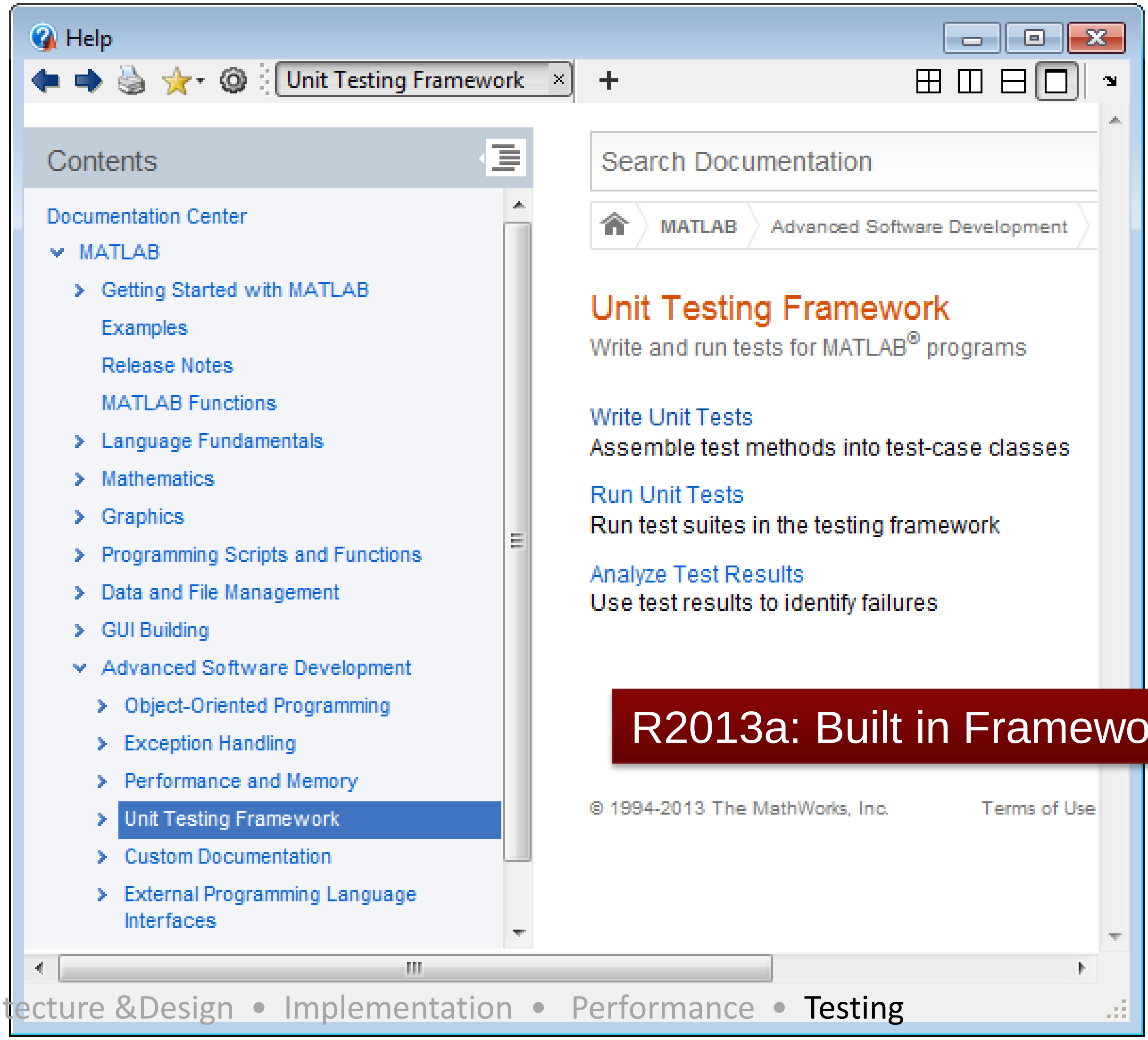
More Error and Warning Handling Options...

matlab.unittest

Write
Verify
Run

Effective Testing in MATLAB

- Automation



Documentation Center
- MATLAB
  - Getting Started with MATLAB
  - Examples
  - Release Notes
  - MATLAB Functions
  - Language Fundamentals
  - Mathematics
  - Graphics
  - Programming Scripts and Functions
  - Data and File Management
  - GUI Building
  - Advanced Software Development
    - Object-Oriented Programming
    - Exception Handling
    - Performance and Memory
    - Unit Testing Framework
    - Custom Documentation
    - External Programming Language Interfaces

MATLAB › Advanced Software Development

Unit Testing Framework

Write and run tests for MATLAB® programs

Write Unit Tests
Assemble test methods into test-case classes

Run Unit Tests
Run test suites in the testing framework

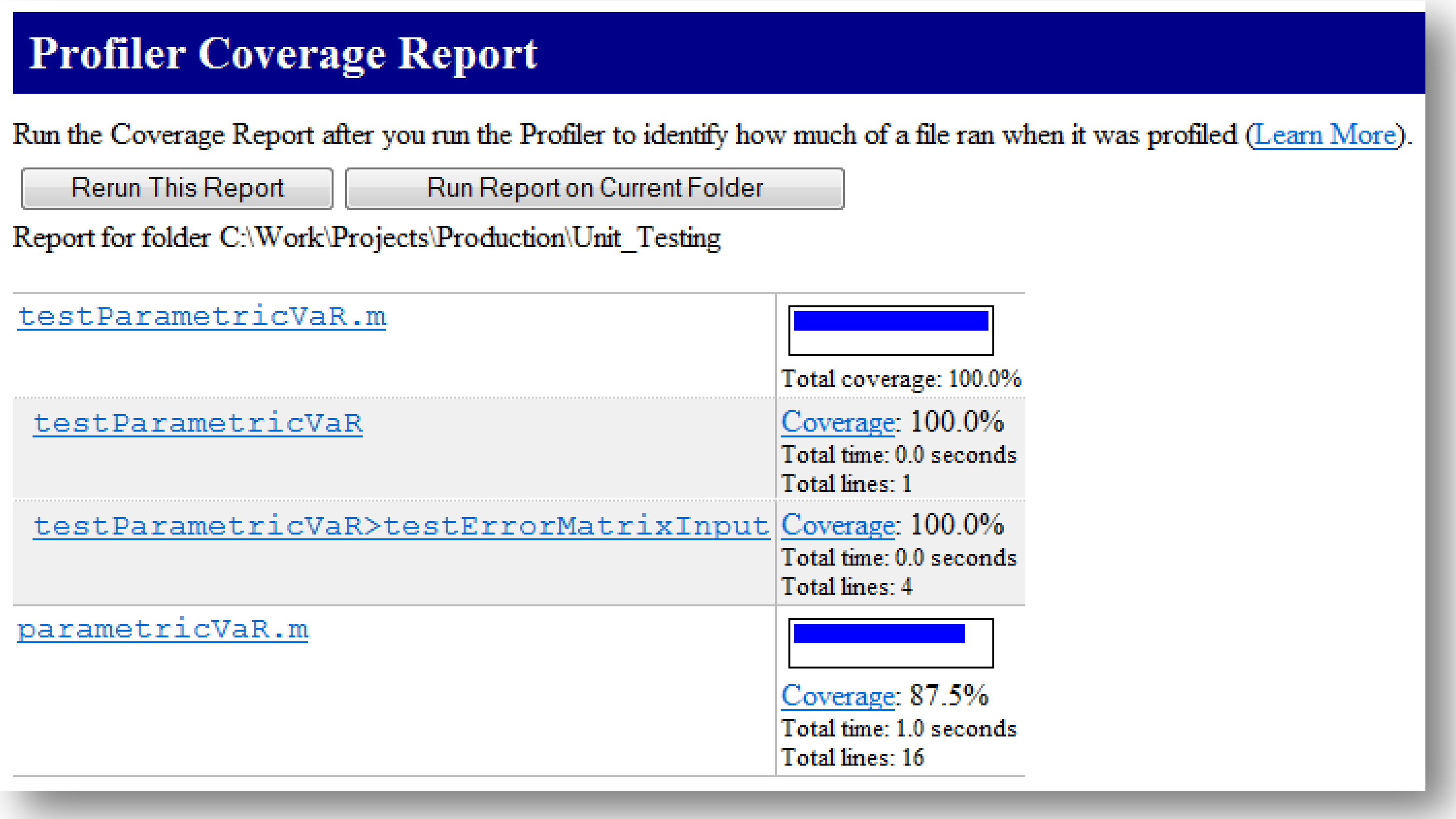
Analyze Test Results
Use test results to identify failures

© 1994-2013 The MathWorks, Inc. Terms of Use

R2013a: Built in Framework

Effective Testing in MATLAB

- Automation
- Coverage



Profiler Coverage Report

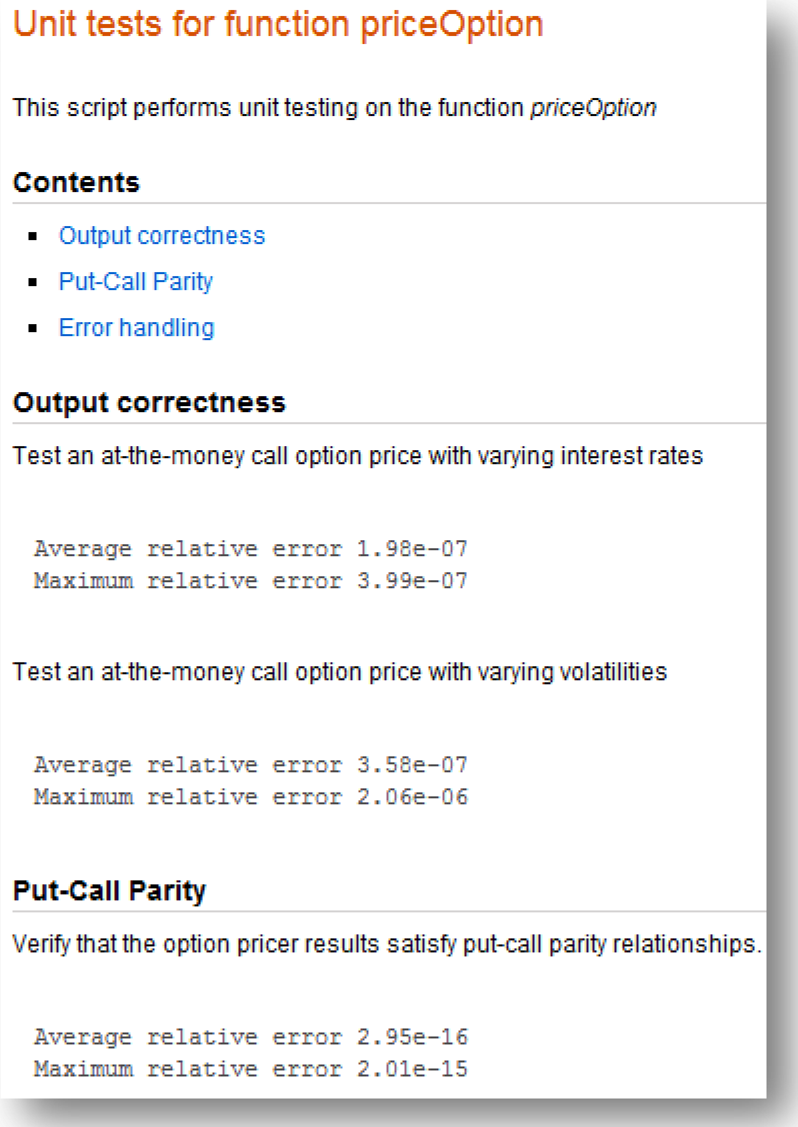
Run the Coverage Report after you run the Profiler to identify how much of a file ran when it was profiled (Learn More).

Report for folder C:\Work\Projects\Production\Unit_Testing

testParametricVaR.m	Total coverage: 100.0%
testParametricVaR	Coverage: 100.0% Total time: 0.0 seconds Total lines: 1
testParametricVaR>testErrorMatrixInput	Coverage: 100.0% Total time: 0.0 seconds Total lines: 4
parametricVaR.m	Coverage: 87.5% Total time: 1.0 seconds Total lines: 16

Effective Testing in MATLAB

- Automation (Harness, eg xUnit)
- Coverage
- Artifact Generation (Documentation)



Unit tests for function priceOption

This script performs unit testing on the function *priceOption*

Contents

- Output correctness
- Put-Call Parity
- Error handling

Output correctness

Test an at-the-money call option price with varying interest rates

```
Average relative error 1.98e-07
Maximum relative error 3.99e-07
```

Test an at-the-money call option price with varying volatilities

```
Average relative error 3.58e-07
Maximum relative error 2.06e-06
```

Put-Call Parity

Verify that the option pricer results satisfy put-call parity relationships.

```
Average relative error 2.95e-16
Maximum relative error 2.01e-15
```

Developing Production-Ready MATLAB Code

- Collaboration
- Quality, e.g. FURPS (HP, IBM)
- Integration