

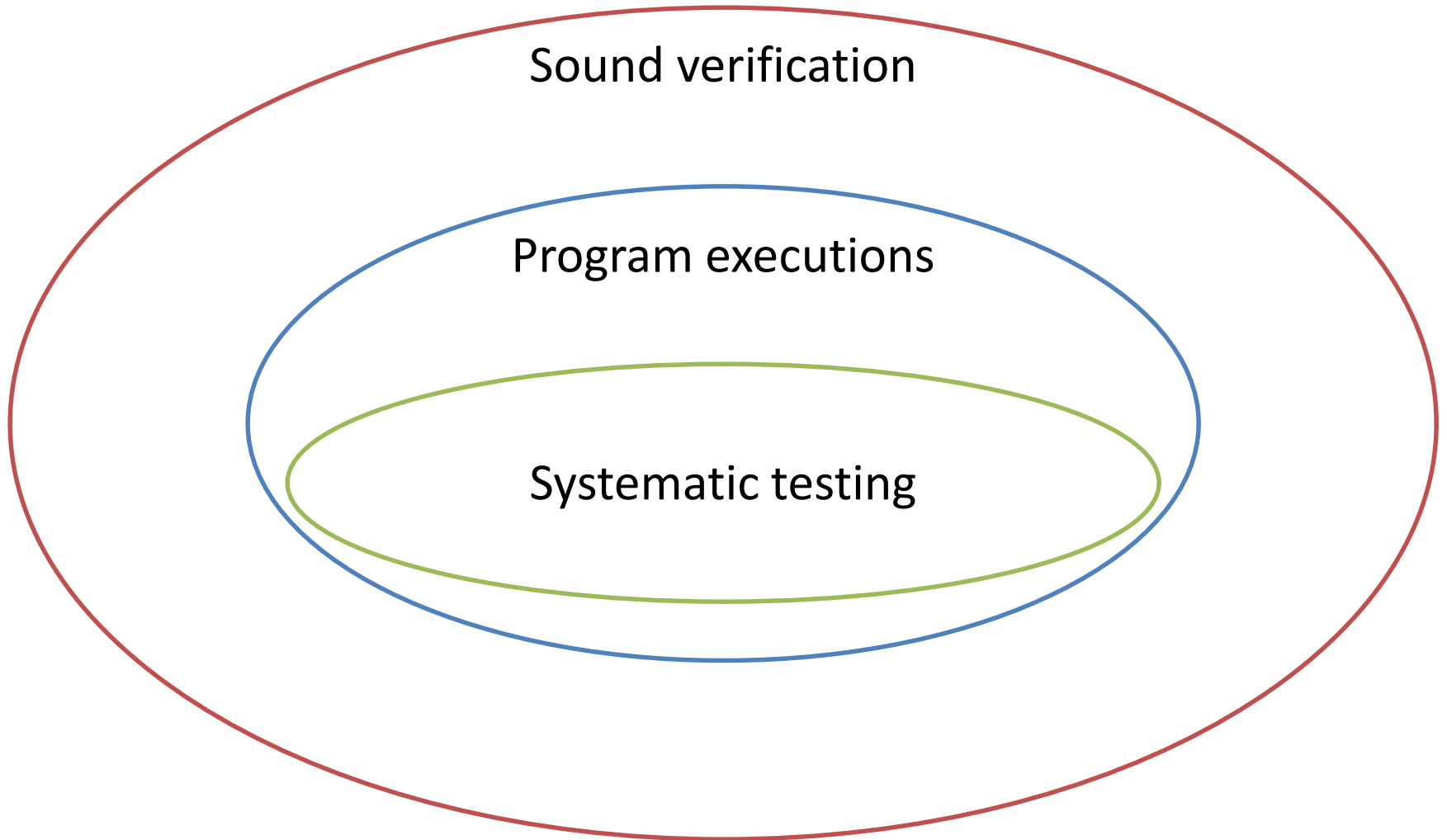
Static Program Analysis Meets Test Case Generation

Lecture 2

Maria Christakis
MPI-SWS, Germany

Textbook:
Static Program Analysis

Verification and testing



Question

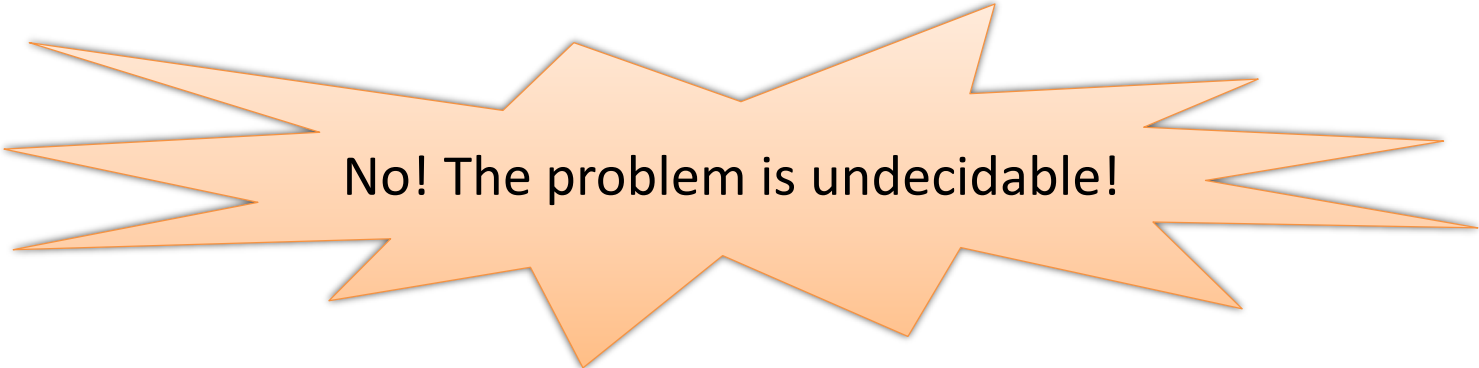
Can we build an **automatic** analyzer that takes as input an **arbitrary program** and an **arbitrary property** such that if the analyzer answers:

- “**Yes**”, then it is certain that **the property holds**
- “**No**”, then it is certain that **the property does not hold**

Question

Can we build an **automatic** analyzer that takes as input an **arbitrary program** and an **arbitrary property** such that if the analyzer answers:

- “**Yes**”, then it is certain that **the property holds**
- “**No**”, then it is certain that **the property does not hold**



No! The problem is undecidable!

New question

Can we build an **automatic** analyzer that takes as input an **arbitrary program** and an **arbitrary property** such that if the analyzer answers:

- “**Yes**”, then it is certain that **the property holds**
- “**No**”, then it is **unknown** if the property holds or not

New question

Can we build an **automatic** analyzer that takes as input an **arbitrary program** and an **arbitrary property** such that if the analyzer answers:

- “**Yes**”, then it is certain that **the property holds**
- “**No**”, then it is **unknown** if the property holds or not

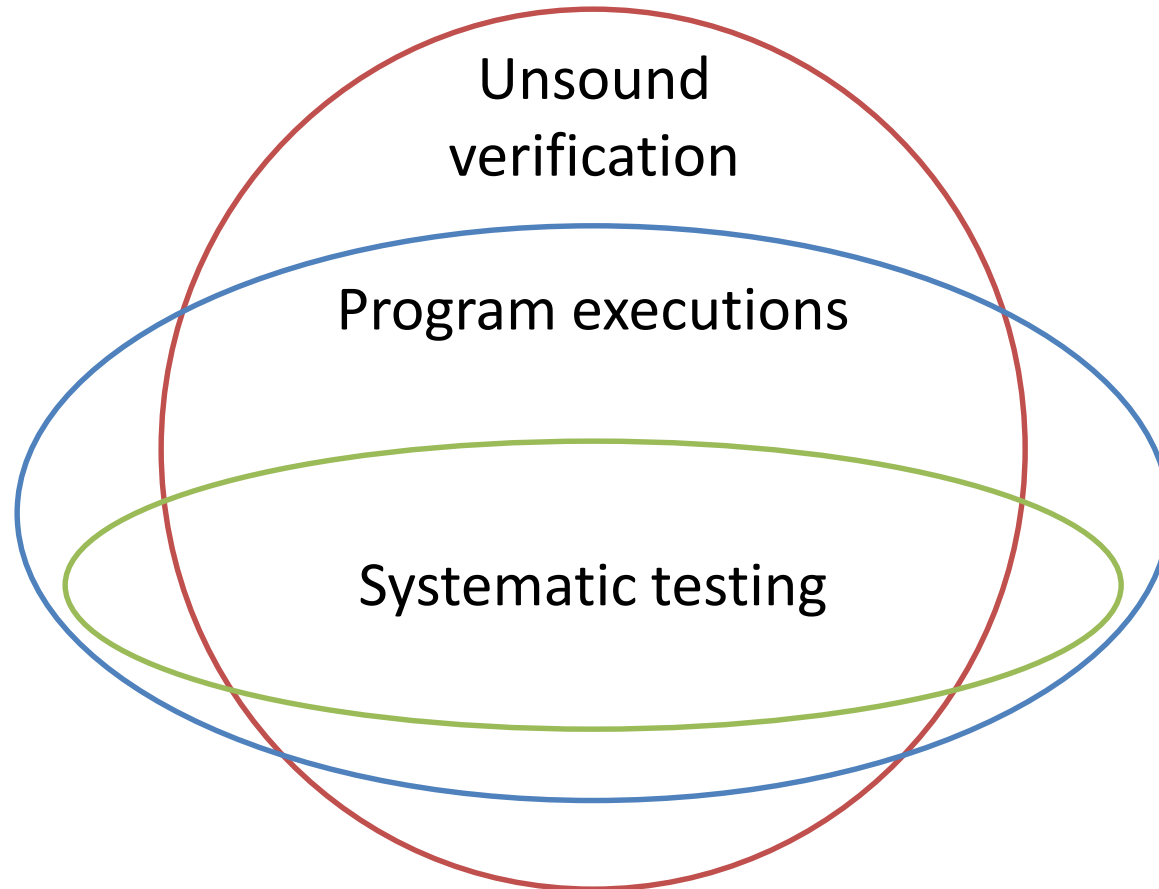


Yes! This problem is trivial!

Static analysis: Challenge

Building a static analyzer that is able to answer “Yes”
for as many programs satisfying the property

Verification and testing



Question: Sound analysis

Can we build an **automatic** analyzer that takes as input an **arbitrary program** and an **arbitrary property** such that if the analyzer answers:

- “**Yes**”, then it is certain that **the property holds**
- “**No**”, then it is **unknown** if the property holds or not

Question: Unsound analysis

Can we build an **automatic** analyzer that takes as input an **arbitrary program** and an **arbitrary property** such that if the analyzer answers:

- “**Yes**”, then it is certain that **the property holds** *ish!*
- “**No**”, then it is **unknown** if the property holds or not

Static analysis: Cool facts

- Can automatically prove interesting properties
 - absence of runtime exceptions, assertions, absence of data races, termination, ...

Static analysis: Cool facts

- Can **automatically prove** interesting properties
 - absence of runtime exceptions, assertions, absence of data races, termination, ...
- Nicely combines **math and tool building**
 - program semantics, data structures, logic, decision procedures, discrete math, ...

Static analysis: Cool facts

- No concrete inputs required
 - Code is abstractly executed from any point

Static analysis: Cool facts

- No concrete inputs required
 - Code is abstractly executed from any point
- No manual annotations required
 - Loop invariants are automatically inferred

Fun:

What Developers Want and Need
from Program Analysis: An Empirical Study

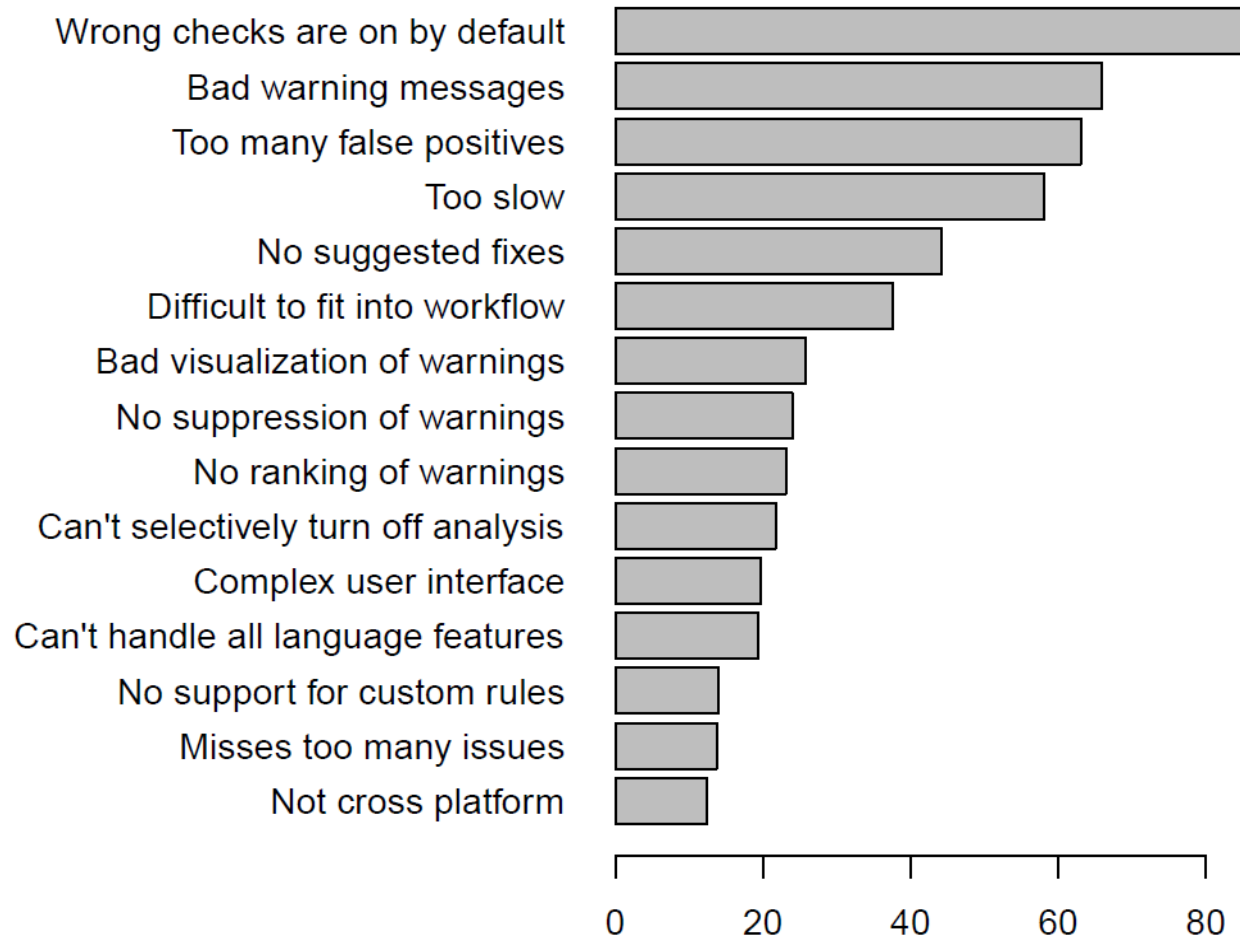
What makes a program analyzer most attractive to developers?

What makes a program analyzer most attractive to developers?

1. What makes program analyzers difficult to use?
2. What functionality should analyzers have?
3. What should their non-functional characteristics be?
4. What code issues occur most in practice?

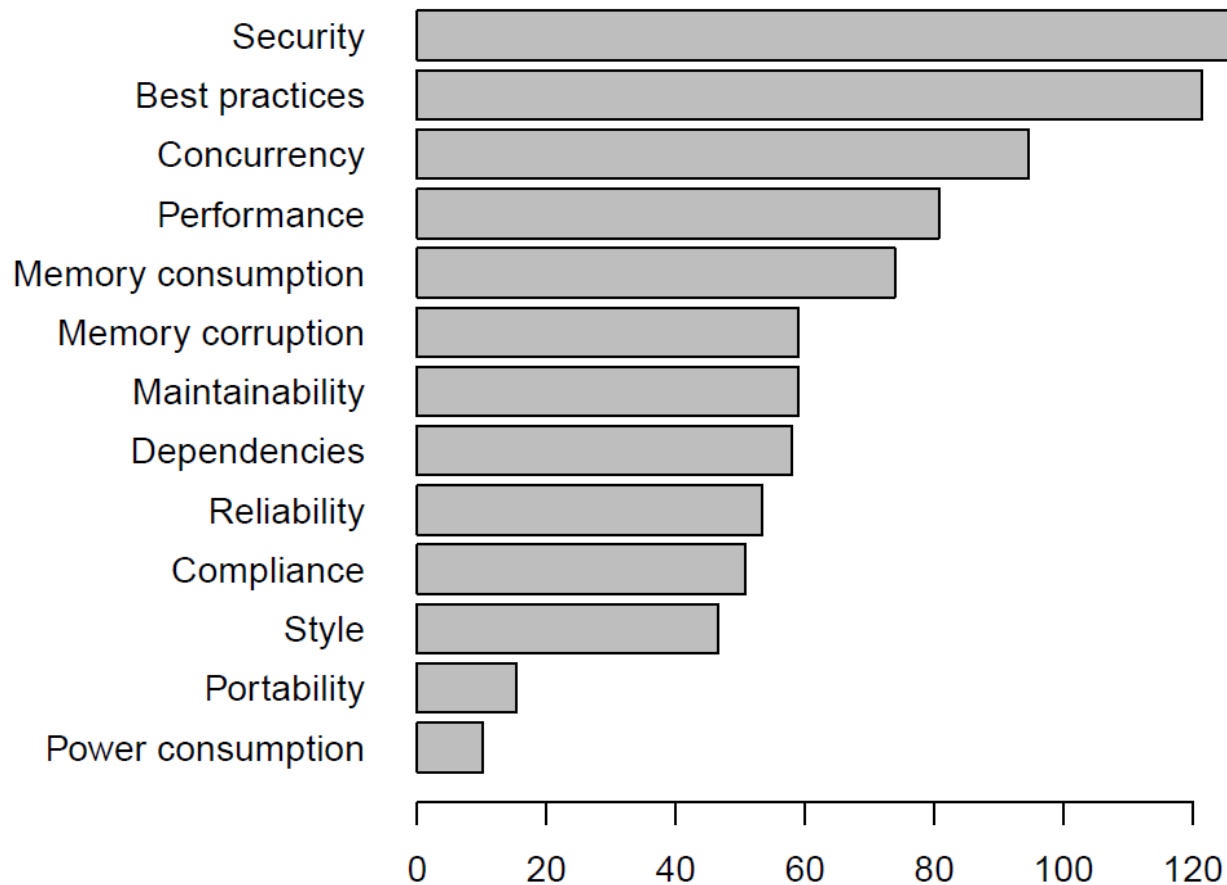
1. What makes program analyzers difficult to use?

Pain Points Using Program Analyzers



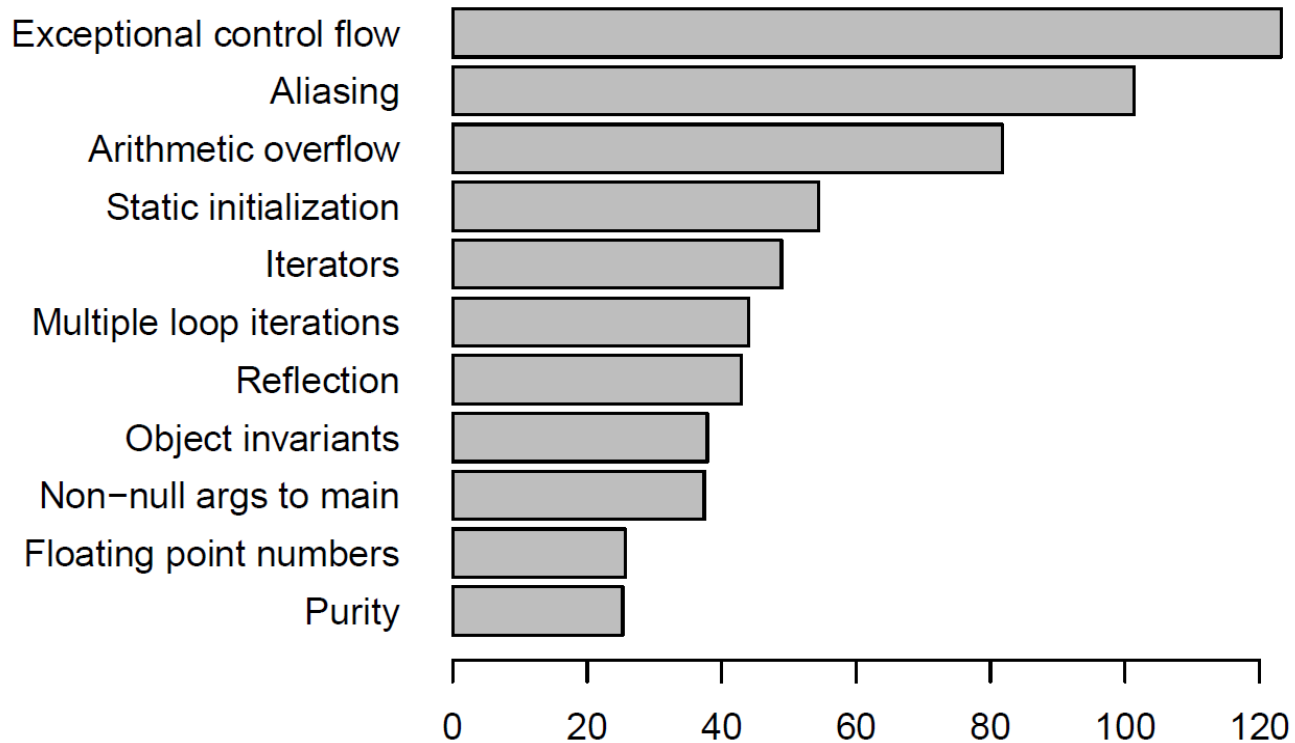
2. What functionality should program analyzers have?

Code Issues Developers Would Like Detected



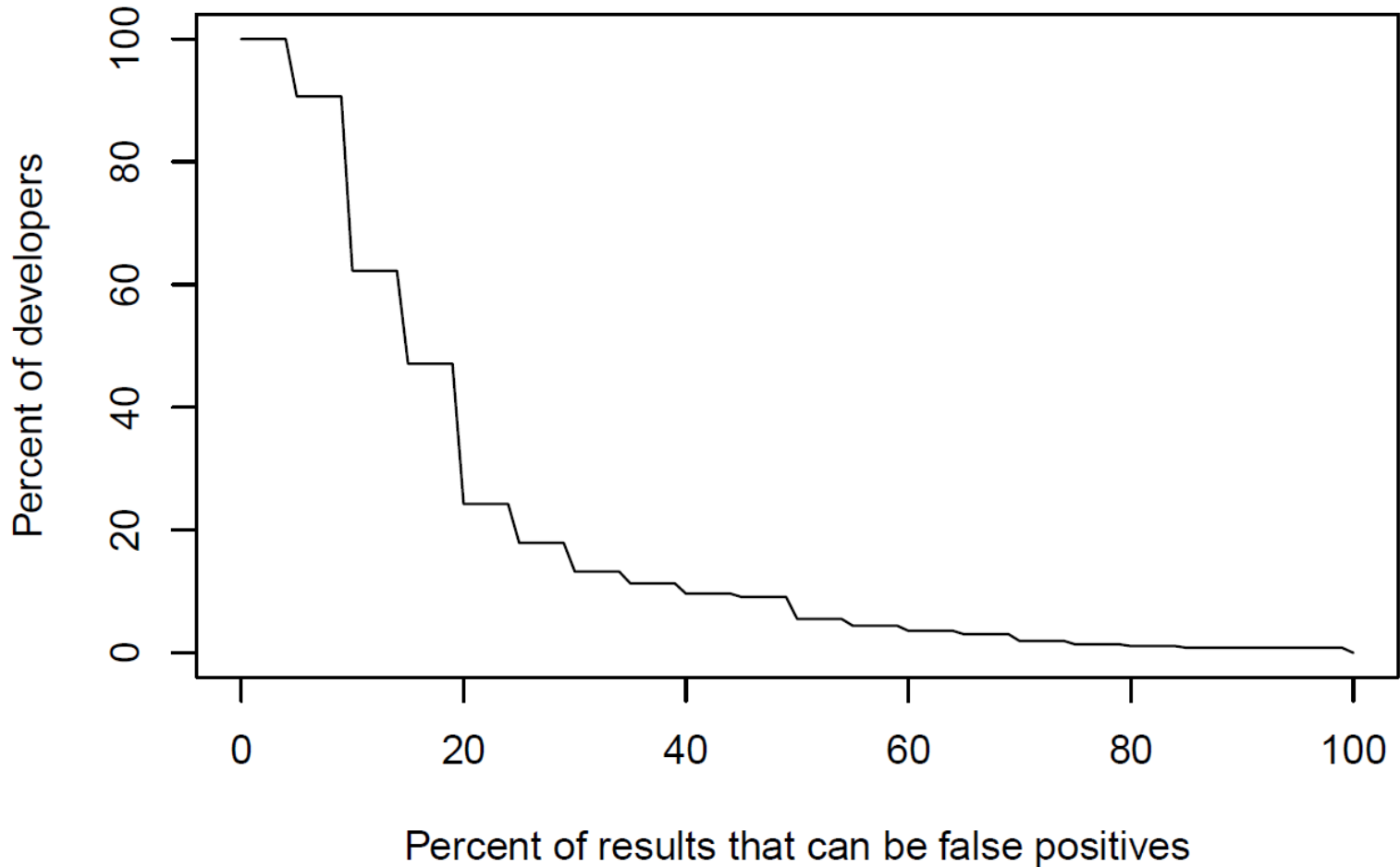
2. What functionality should program analyzers have?

Sources of Unsoundness That Should Not Be Overlooked



3. What should their non-functional characteristics be?

Acceptable False Positive Rate



3. What should their non-functional characteristics be?

- Trade-offs about **analysis time**
 - 57% for more intricate issues
 - 57% for fewer false positives
 - 60% for fewer false negatives
- Trade-offs about **false positives**
 - 50.7% for fewer false negatives

3. What should their non-functional characteristics be?

Program analysis should take a **two-stage** approach, with one stage providing **easy** feedback in the **editor** and another running **overnight** finding **intricate** issues.

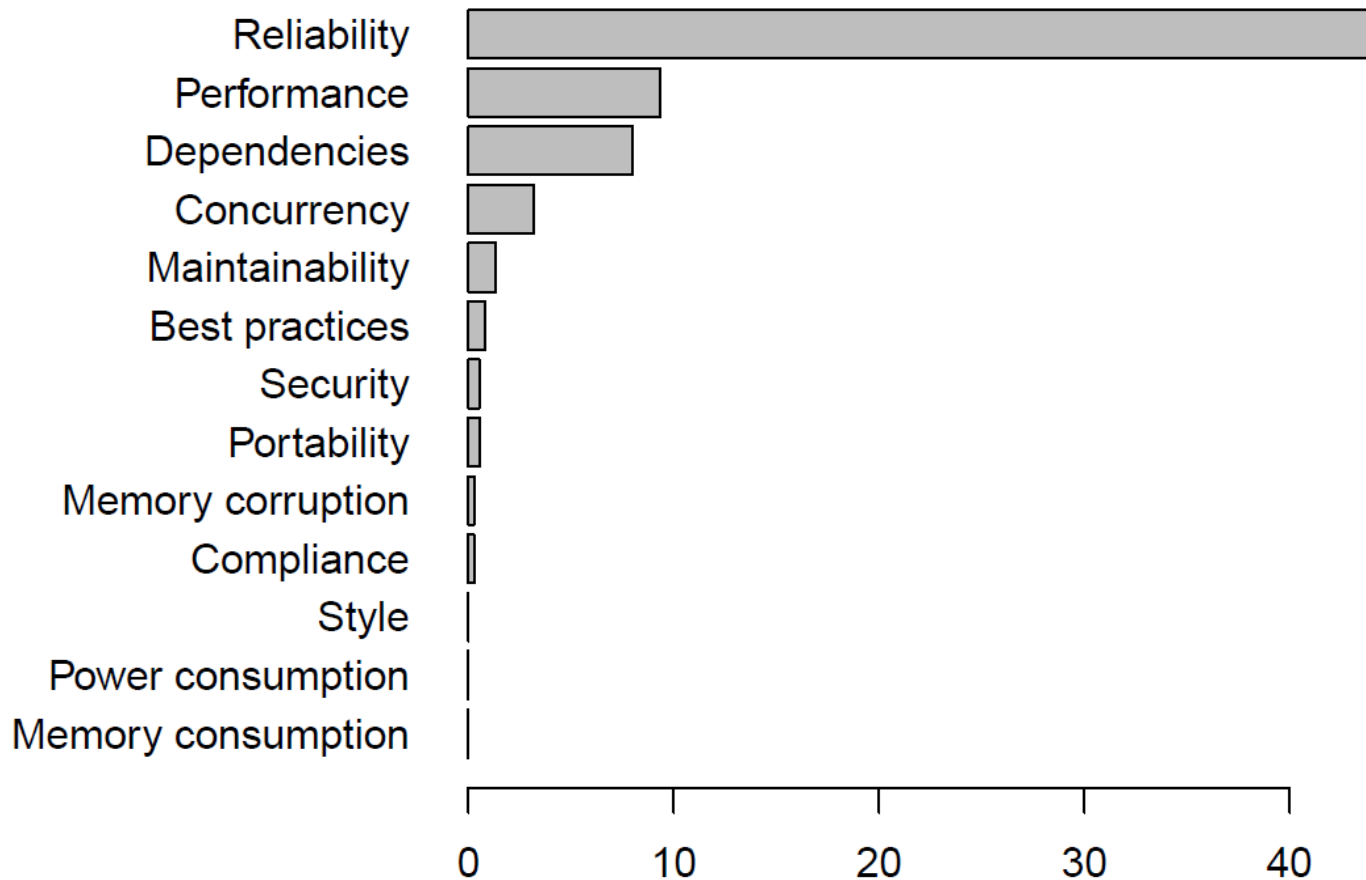
3. What should their non-functional characteristics be?

Program analysis should take a **two-stage** approach, with one stage providing **easy** feedback in the **editor** and another running **overnight** finding **intricate** issues.

“Give me what you can give, fast and accurate (no false positives). Give me the slow stuff later in an hour (it is too good and cheap to not have it). No reasonable change is going to be checked in less than half a day but I do want that style check for that one line fix right away.”

4. What code issues occur most in practice?

Live Site Incidents per Category



Takeaways

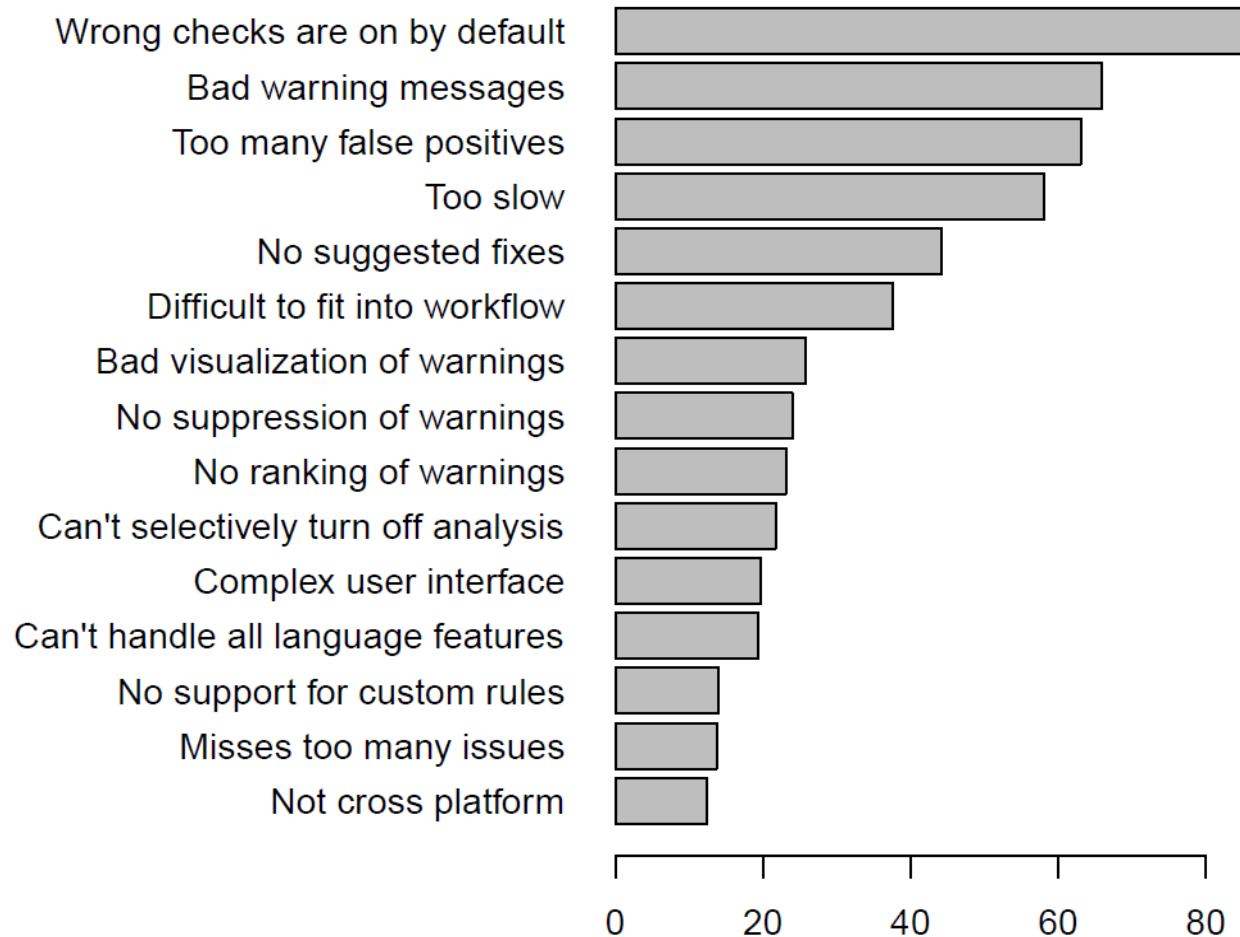
- High false positive rates lead to disuse
- Developers would trade time for higher-quality results
- Program analysis should take a two-stage approach
- Costly bugs in services are mostly related to reliability but developers rank reliability errors low
- Developers do not trust program analyzers to find intricate issues although they want to

Fun:

CFar: A Tool to Increase Communication, Productivity,
and Review Quality in Collaborative Code Reviews

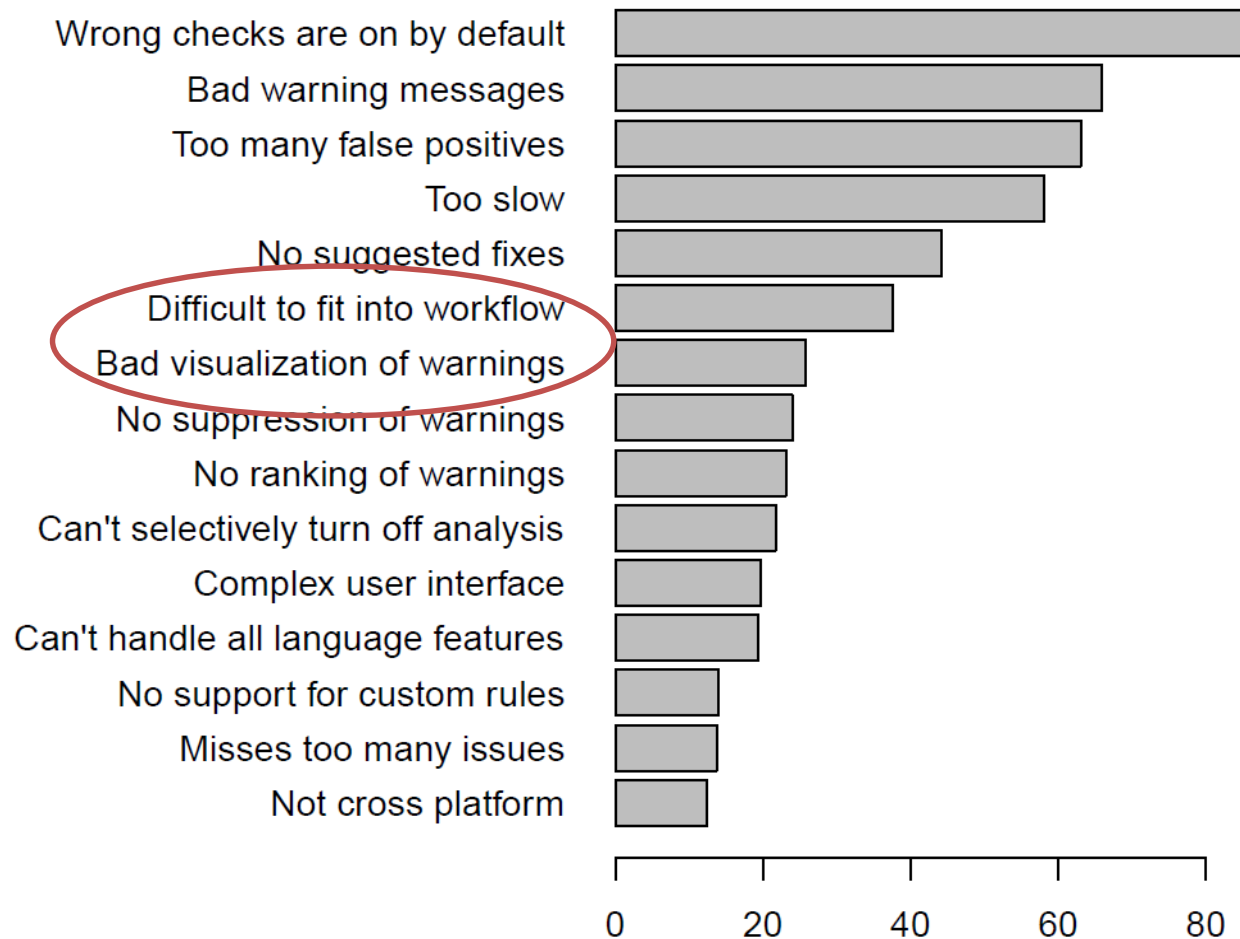
1. What makes program analyzers difficult to use?

Pain Points Using Program Analyzers

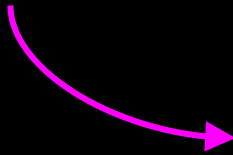
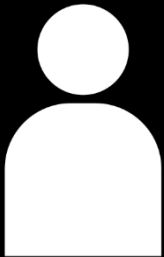


1. What makes program analyzers difficult to use?

Pain Points Using Program Analyzers



What is a code review?



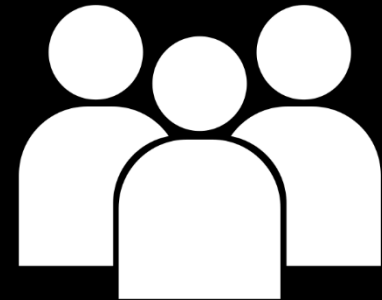
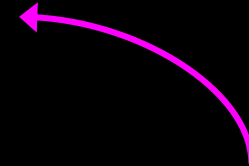
```
def is_scrolling_widget(w):
    return hasattr(w, "get_scrollpos") and hasattr(w,

for w in orig_iter(self):
    if is_scrolling_widget(w):
        return w

def keypress(self, size, key):
    return self._original_widget.keypress(self._original_w

def mouse_event(self, size, event, button, col, row, focus
ow = self._original_widget
ow_size = self._original_widget_size
handled = False
if hasattr(ow, "mouse_event"):
    handled = ow.mouse_event(ow_size, event, button, c

if not handled and hasattr(ow, "set_scrollpos"):
    if button == 4: # Scroll wheel up
        pos = ow.get_scrollpos(ow_size)
        ow.set_scrollpos(pos - 1)
    return True
```



Benefits of Code Reviews

- Discover bugs
- Improve readability and maintainability
- Mental model of the code

Problems with Code Reviews

- Time consuming
 - Over 6 hours per week
 - 100 reviews per week
- Spend effort on *shallow* defects
- Miss other defects

Approach: Automated Code Reviewer

- Insert feedback based on program analysis
- Goal:
 - Increase communication
 - Enhance productivity
 - Reveal more defects

Extend Microsoft CodeFlow

CodeFlow

Using VSTest adapter for nunit-based tests when dynamic coverage is enabled. Got confirmation from vstest team since this is officially supported and helps avoid hangs with nunit-console.exe - CodeFlow / On Premises [Dogfood]

DynamicCodeCoverageRunner.cs

```
215 if (testRunner is VsTestRunner)
216 {
217     throw new QTestException($"Dynamic code coverage for target: {qtestRunnerData.TestBinaryName} is invoked with " +
218                             $"VsTestRunner instead of VsTestRunner140. Only VsTestRunner140 supports code coverage");
219 }
220
221 QTestOptMacros clonedMacros = qtestOptMacros.Clone();
222 //These are specific for the underlying runner, so should not be specified for VsTest runner
223 clonedMacros.QTestAdditionalOptions = "";
224
225 //NUnit runner is treated special. These tests are executed directly using nunit adapter
226 //VsTest team confirmed that there is an official support for VsTest adapter for NUnit.
227 //Using this adapter, instead of launching via nunit-console.exe since it is hanging
228 VsTestRunner140 vstestRunner;
229 if (testRunner is NUnitRunnerBase)
230 {
231     clonedMacros.QTestAdapterPath = $"{QGlobalConstants.QBitsPath}\\{QTestConstants.NUnitVsTestAdapterRelativePath}";
232
233     //We dont populate the original test runner, since results will be collected by the vstest runner itself
234     vstestRunner = new VsTestRunner140(qtestRunnerData, accountInfo, clonedMacros, qtestCmdLineOptions);
235 }
236 else
237 {
238     //Get the command line from the original runner
239     string command = testRunner.Command;
240     string commandArgs = testRunner.GetFinalCommandArgs();
241
242     string qtestCommandFile = Path.ChangeExtension(qtestRunnerData.TestBinaryName,
243     QTestVsTestAdapterConstants.QTestCmdInput);
244     File.WriteAllText(qtestCommandFile, $"{command}{Environment.NewLine}{commandArgs}");
245
246     //Replace the original runner with vstest runner
247     QTestOptMacros clonedMacros = qtestOptMacros.Clone();
248     //Replace the original runner with vstest runner and an adapter path
249     clonedMacros.QTestAdapterPath =
250     $"{QGlobalConstants.QBitsPath}\\{QTestConstants.QTestVsTestAdapterRelativePath}";
251     clonedMacros.QTestAdditionalOptions = "";
252     //These are specific for the underlying runner, so should not be specified for VsTest runner
253     var vstestRunner = new VsTestRunner140(qtestRunnerData, accountInfo, clonedMacros, qtestCmdLineOptions, testRunner);
254 }
255 vstestRunner = new VsTestRunner140(qtestRunnerData, accountInfo, clonedMacros, qtestCmdLineOptions, testRunner);
```

In externally visible method 'DynamicCodeCoverageRunner.InferFrameworkRunnerForCoverage(FrameworkRunner, QTestRunnerData, QTestOptMacros, QTestCmdLineOptions, AccountInfo)', validate parameter 'qtestOptMacros' before using it. (run 'oacr fxcop CloudBuild:retail/target/dbs.qtestlib.dll' for details)

Does the runner-specific logic only apply to NUnit runner or do we have other runners that require specialized logic as well?

Status	File name	Reviewer	Line	Last updated
Active	\\private/DevTools/dbs/QTest/QTestLib/CodeCovRunners/DynamicCodeCoverageRunner.cs	OACR	236	Fri, 1:20 PM
WontFix	\\private/DevTools/dbs/QTest/QTestLib/CodeCovRunners/DynamicCodeCoverageRunner.cs	OACR	726	09/07/2018
ByDesign	/description.txt	OACR	1	09/07/2018

Reviewer Status

- Marge (author)
- ✓ Lisa (required)
- ✓ Bart (required)
- ✓ OACR (required)
- ⌚ Maggie (required)

My Status: Waiting

Show Comments

Type here to filter the list

Status	Participant	Iteration
Unpublished	(0)	
Active	(1)	
Pending	(0)	
Resolved	(0)	

Does the runner-specific logic only apply to Nunit runner or do we have other runners that require specialized logic as well?



Maggie

 Active 

CodeFlow

Using VSTest adapter for nunit-based tests when dynamic coverage is enabled. Got confirmation from vstest team since this is officially supported and helps avoid hangs with nunit-console.exe - CodeFlow / On Premises [Dogfood]

1

DynamicCodeCoverageRunner.cs

```
215 if (testRunner is VsTestRunner)
216 {
217     throw new QTestException($"Dynamic code coverage for target: {qtestRunnerData.TestBinaryName} is invoked with " +
218                             $"VsTestRunner instead of VsTestRunner140. Only VsTestRunner140 supports code coverage");
219 }
220
221 QTestOptMacros clonedMacros = qtestOptMacros.Clone();
222 //These are specific for the underlying runner, so should not be specified for VsTest runner
223 clonedMacros.QTestAdditionalOptions = "";
224
225 //NUnit runner is treated special. These tests are executed directly using nunit adapter
226 //VsTest team confirmed that there is an official support for VsTest adapter for NUnit.
227 //Using this adapter, instead of launching via nunit-console.exe since it is hanging
228 VsTestRunner140 vstestRunner;
229 if (testRunner is NunitRunnerBase)
230 {
231     clonedMacros.QTestAdapterPath = $"{QGlobalConstants.QBitsPath}\\{QTestConstants.NunitVsTestAdapterRelativePath}";
232
233     //We dont populate the original test runner, since results will be collected by the vstest runner itself
234     vstestRunner = new VsTestRunner140(qtestRunnerData, accountInfo, clonedMacros, qtestCmdLineOptions);
235 }
236 else
237 {
238     //Get the command line from the original runner
239     string command = testRunner.Command;
240     string commandArgs = testRunner.GetFinalCommandArgs();
241
242     string qtestCommandFile = Path.ChangeExtension(qtestRunnerData.TestBinaryName,
243     QTestVsTestAdapterConstants.QTestCmdInput);
244     File.WriteAllText(qtestCommandFile, $"{command}{Environment.NewLine}{commandArgs}");
245
246     //Replace the original runner with vstest runner
247     QTestOptMacros clonedMacros = qtestOptMacros.Clone();
248     //Replace the original runner with vstest runner and an adapter path
249     clonedMacros.QTestAdapterPath =
250     $"{QGlobalConstants.QBitsPath}\\{QTestConstants.QTestVsTestAdapterRelativePath}";
251     clonedMacros.QTestAdditionalOptions = "";
252     //These are specific for the underlying runner, so should not be specified for VsTest runner
253     var vstestRunner = new VsTestRunner140(qtestRunnerData, accountInfo, clonedMacros, qtestCmdLineOptions, testRunner);
254 }
255 vstestRunner = new VsTestRunner140(qtestRunnerData, accountInfo, clonedMacros, qtestCmdLineOptions, testRunner);
```

In externally visible method 'DynamicCodeCoverageRunner.InferFrameworkRunnerForCoverage(FrameworkRunner, QTestRunnerData, QTestOptMacros, QTestCmdLineOptions, AccountInfo)', validate parameter 'qtestOptMacros' before using it. (run 'oacr fxcop CloudBuild:retail/target/dbs.qtestlib.dll' for details)

1 1 1 OACR

ByDesign

Does the runner-specific logic only apply to NUnit runner or do we have other runners that require specialized logic as well?

Maggie

Active

Status	File name	Reviewer	Line	Last updated
Active	\\private/DevTools/dbs/QTest/QTestLib/CodeCovRunners/DynamicCodeCoverageRunner.cs	OACR	236	Fri, 1:20 PM
WontFix	\\private/DevTools/dbs/QTest/QTestLib/CodeCovRunners/DynamicCodeCoverageRunner.cs	OACR	726	09/07/2018
ByDesign	/description.txt	OACR	1	09/07/2018

Reviewer Status

- Marge (author)
- ✓ Lisa (required)
- ✓ Bart (required)
- ✓ OACR (required)
- ⌚ Maggie (required)

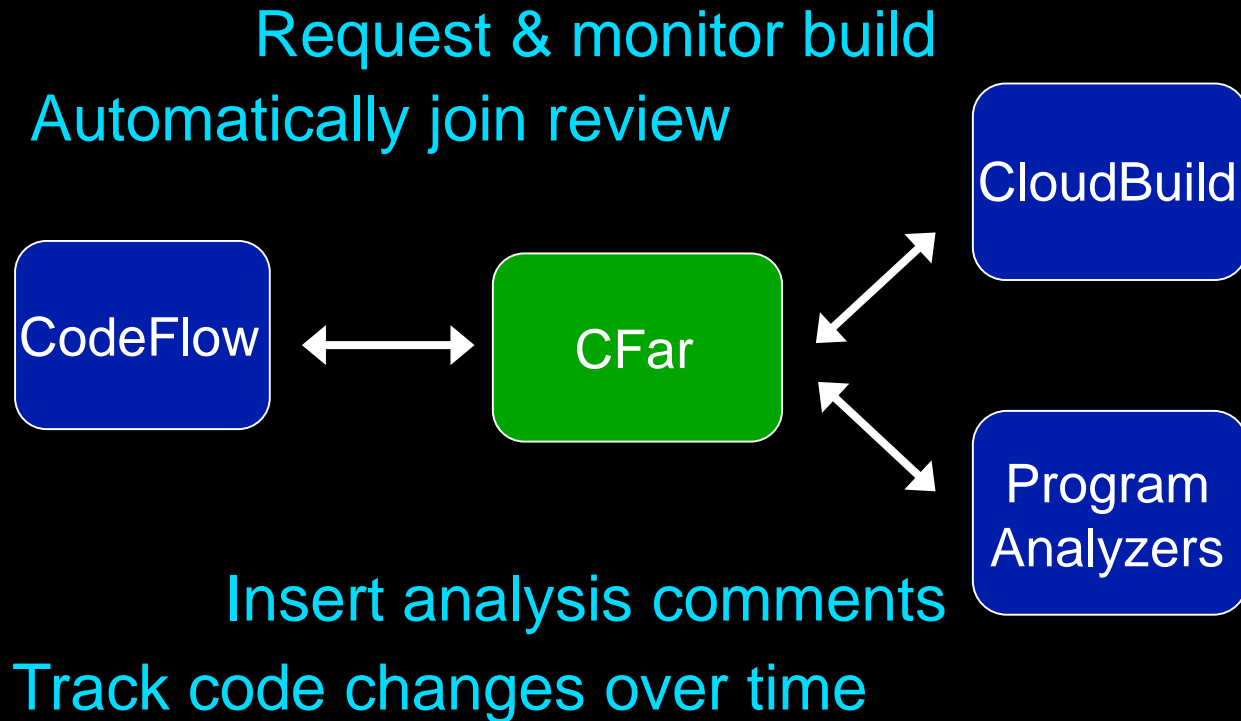
My Status: Waiting

Show Comments

Type here to filter the list

Status	Participant	Iteration
Unpublished	(0)	
Active	(1)	
Pending	(0)	
Resolved	(0)	

CFar Architecture & Features



CodeFlow + CFar

Using VSTest adapter for nunit-based tests when dynamic coverage is enabled. Got confirmation from vstest team since this is officially supported and helps avoid hangs with nunit-console.exe - CodeFlow / On Premises [Dogfood]

DynamicCodeCoverageRunner.cs

```
215 if (testRunner is VsTestRunner)
216 {
217     throw new QTestException($"Dynamic code coverage for target: {qtestRunnerData.TestBinaryName} is invoked with " +
218                             $"VsTestRunner instead of VsTestRunner140. Only VsTestRunner140 supports code coverage");
219 }
220
221 QTestOptMacros clonedMacros = qtestOptMacros.Clone();
222 //These are specific for the underlying runner, so should not be specified for VsTest runner
223 clonedMacros.QTestAdditionalOptions = "";
224
225 //NUnit runner is treated special. These tests are executed directly using nunit adapter
226 //VsTest team confirmed that there is an official support for VsTest adapter for NUnit.
227 //Using this adapter, instead of launching via nunit-console.exe since it is hanging
228 VsTestRunner140 vstestRunner;
229 if (testRunner is NunitRunnerBase)
230 {
231     clonedMacros.QTestAdapterPath = $"{QGlobalConstants.QBitsPath}\\{QTestConstants.NunitVsTestAdapterRelativePath}";
232
233     //We dont populate the original test runner, since results will be collected by the vstest runner itself
234     vstestRunner = new VsTestRunner140(qtestRunnerData, accountInfo, clonedMacros, qtestCmdLineOptions);
235 }
236 else
237 {
238     //Get the command line from the original runner
239     string command = testRunner.Command;
240     string commandArgs = testRunner.GetFinalCommandArgs();
241
242     string qtestCommandFile = Path.ChangeExtension(qtestRunnerData.TestBinaryName,
243         QTestVsTestAdapterConstants.QTestCmdInput);
244     File.WriteAllText(qtestCommandFile, $"{command}{Environment.NewLine}{commandArgs}");
245
246     //Replace the original runner with vstest runner
247     QTestOptMacros clonedMacros = qtestOptMacros.Clone();
248     //Replace the original runner with vstest runner and an adapter path
249     clonedMacros.QTestAdapterPath =
250         $"{QGlobalConstants.QBitsPath}\\{QTestConstants.QTestVsTestAdapterRelativePath}";
251     clonedMacros.QTestAdditionalOptions = "";
252     //These are specific for the underlying runner, so should not be specified for VsTest runner
253     var vstestRunner = new VsTestRunner140(qtestRunnerData, accountInfo, clonedMacros, qtestCmdLineOptions, testRunner);
254 }
255 vstestRunner = new VsTestRunner140(qtestRunnerData, accountInfo, clonedMacros, qtestCmdLineOptions, testRunner);
```

Comment by OACR: In externally visible method 'DynamicCodeCoverageRunner.InferFrameworkRunnerForCoverage(FrameworkRunner, QTestRunnerData, QTestOptMacros, QTestCmdLineOptions, AccountInfo)', validate parameter 'qtestOptMacros' before using it. (run 'oacr fxcop CloudBuild:retail/target/dbs.qtestlib.dll' for details)





Comment by Maggie: Does the runner-specific logic only apply to NUnit runner or do we have other runners that require specialized logic as well?

Status	File name	Reviewer	Line	Last updated
Active	\\private/DevTools/dbs/QTest/QTestLib/CodeCovRunners/DynamicCodeCoverageRunner.cs	OACR	236	Fri, 1:20 PM
WontFix	\\private/DevTools/dbs/QTest/QTestLib/CodeCovRunners/DynamicCodeCoverageRunner.cs	OACR	726	09/07/201
ByDesign	/description.txt	OACR	1	09/07/201


Reviewer Status: Marge (author), Lisa (required), Bart (required), OACR (required), Maggie (required)

Comment Summary: Show Comments, type here to filter the list, Status Participant Iteration, Unpublished (0), Active (1), Pending (0), Resolved (0)

In externally visible method
'DynamicCodeCoverageRunner.InferFrameworkRun
nerForCoverage(FrameworkRunner,
QTestRunnerData, QTestOptMacros,
QTestCmdLineOptions, AccountInfo)', validate
parameter 'qtestOptMacros' before using it. (run '
oacr fxcop CloudBuild:retail /target dbs.qtestlib.dll '
for details)

  1  1 

OACR

 ByDesign ▾

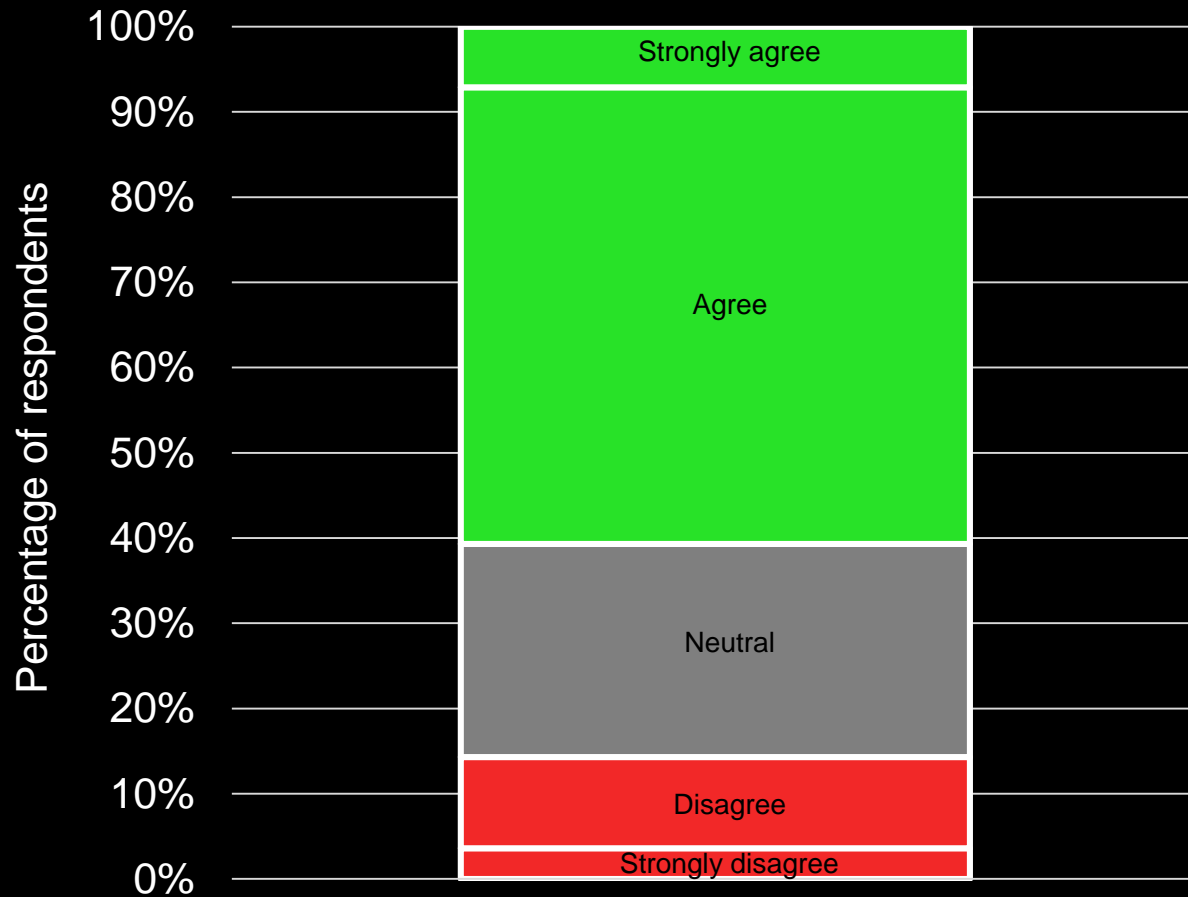
User Study & Field Deployment

- User study
 - 6 developers at Microsoft
 - 1 hour sessions
 - Think aloud
- Field Deployment
 - 98 developers at Microsoft
 - 15 weeks of real-world usage
 - Logged data and emailed survey

Research Questions

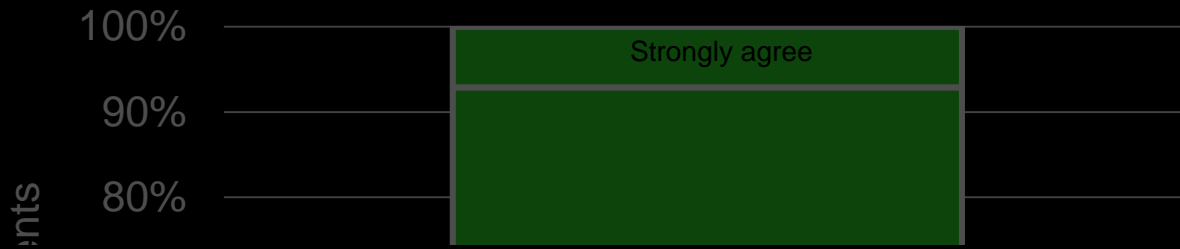
- RQ1: Communication
- RQ2: Productivity
- RQ3: Code quality
- RQ4: Useful

Result: Communication

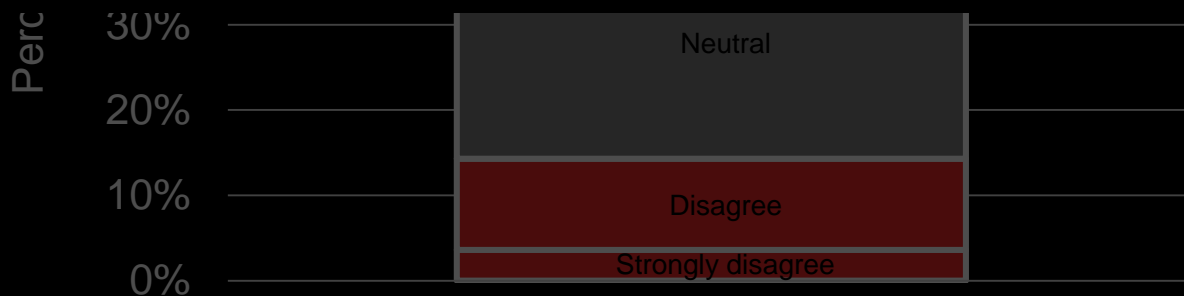


Overall, I feel that the analysis comments enhanced collaboration among developers.

Result: Communication

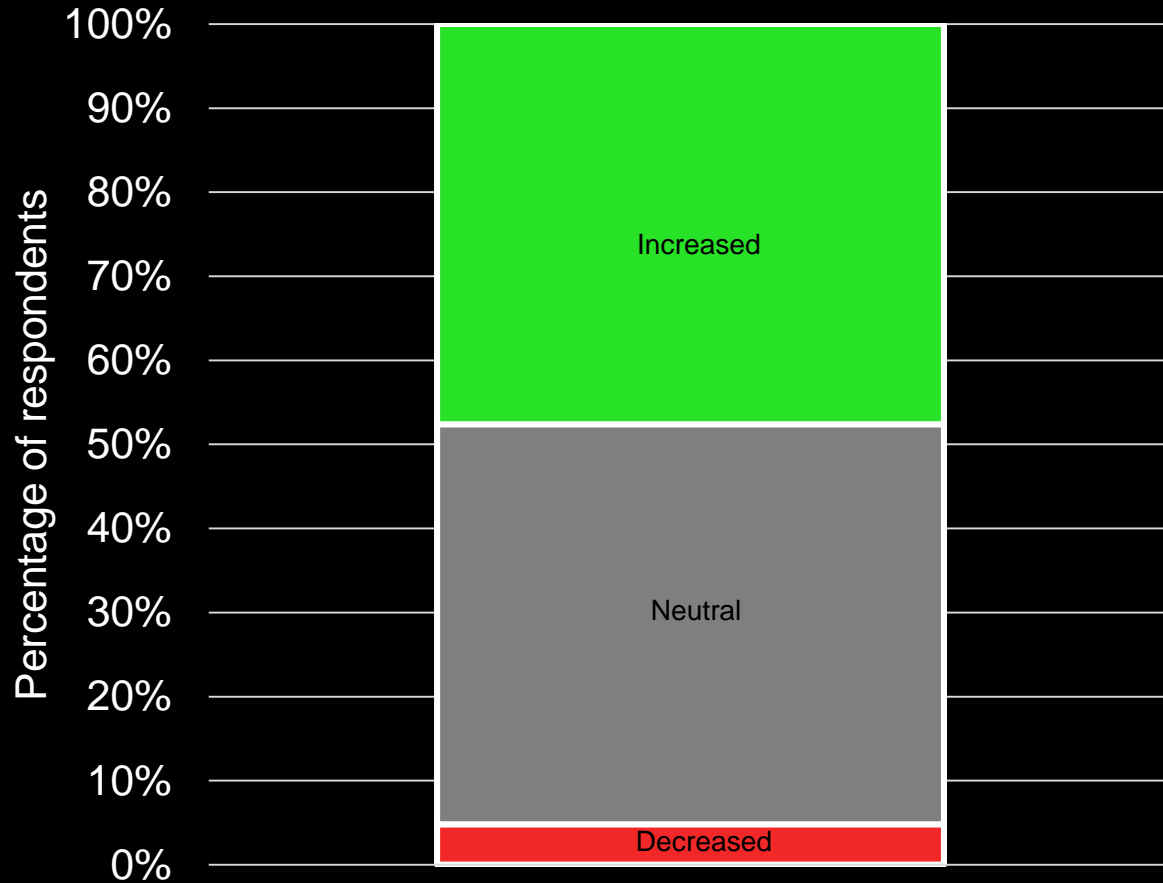


P14: “Having some comments helped start the conversations that might be missed until last minute, so their addition is a net positive.”



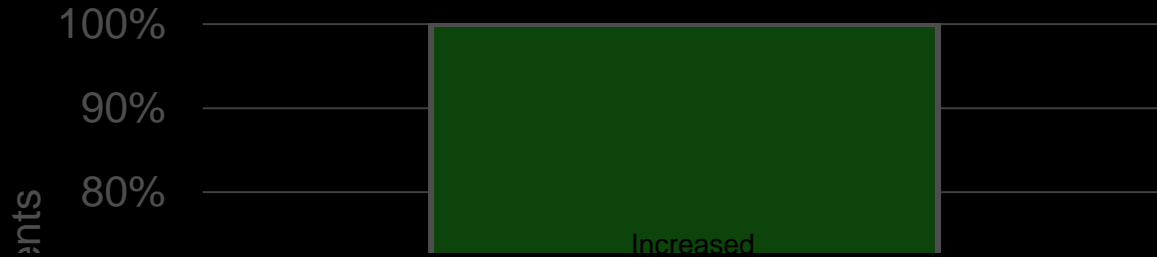
Overall, I feel that the analysis comments enhanced collaboration among developers.

Result: Code Quality



The analysis comments decreased or increased the quality of the code?

Result: Code Quality



98% of analysis comments were addressed



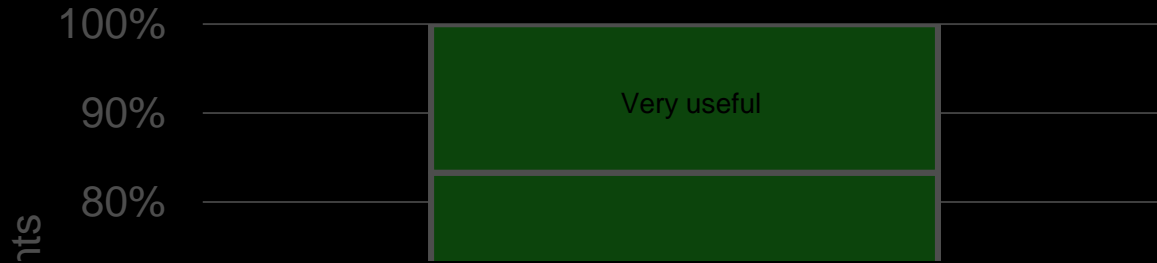
The analysis comments decreased or increased the quality of the code?

Result: Useful

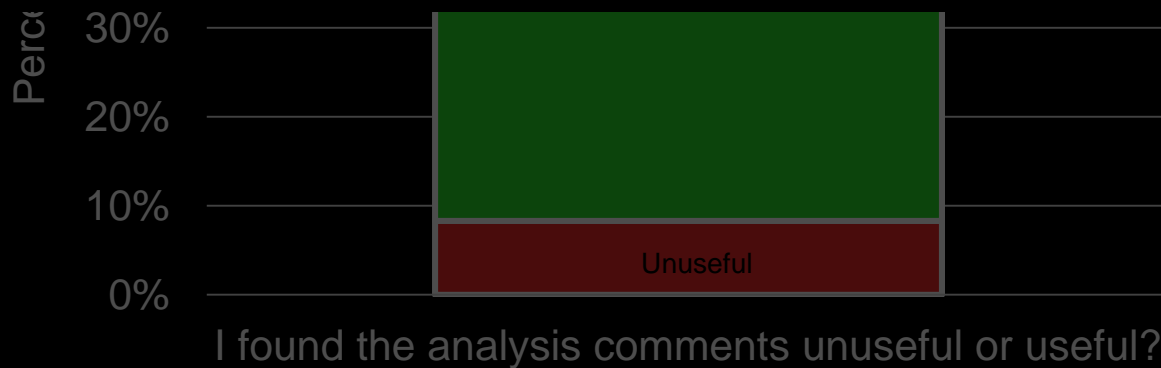


I found the analysis comments unuseful or useful?

Result: Useful



P13: “...It seems like the auto-generated comments I’ve seen are the tip of the iceberg... I’m optimistic that there’s a lot of untapped potential.”



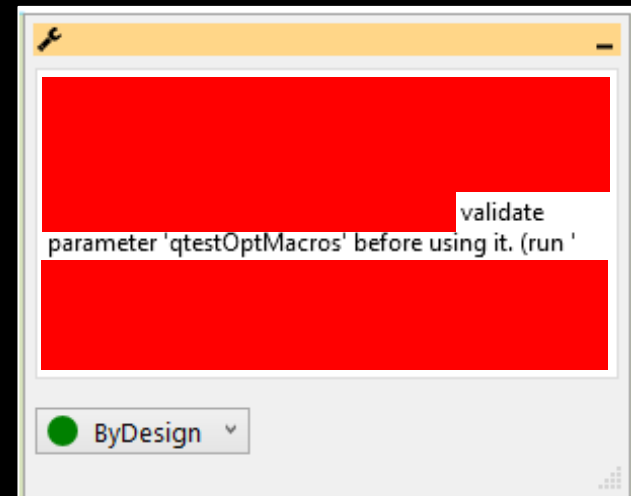
Discussion: Improvements to CFar

- Reduce information overload
 - Simplify the warning messages
 - Provide more filtering features

```
if (testRunner is VsTestRunner)
{
    throw new QTestException($"Dynamic code coverage for
        $"VsTestRunner instead of V
}

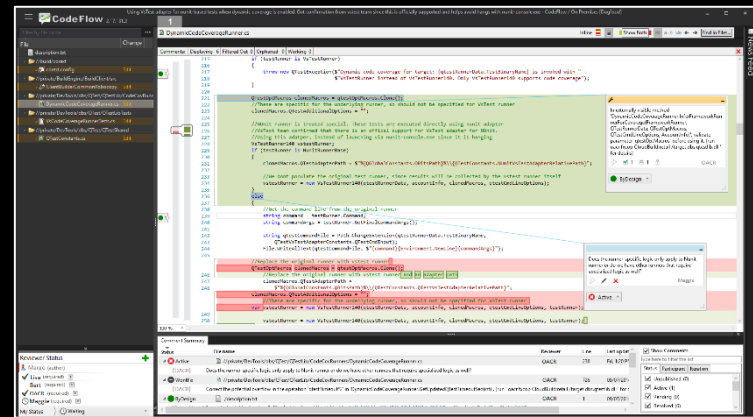
QTestOptMacros clonedMacros = qtestOptMacros.Clone();
//These are specific for the underlying runner, so should
clonedMacros.QTestAdditionalOptions = "";

//NUnit runner is treated special. These tests are execu
//VsTest team confirmed that there is an official support
//Using this adapter, instead of launching via nunit-con
VsTestRunner140 vstestRunner;
if (testRunner is NUnitRunnerBase)
{
```



Takeaways

- CFar: Automated Code Reviewer at Microsoft
 - Designed to improve collaboration
- User Study & Field Deployment
 - Increased communication
 - Increased productivity
 - Improved code quality
 - Found CFar useful



Textbook: Abstract Interpretation*

*Slides inspired by Martin Vechev's course at ETH Zurich

Abstract interpretation: Steps

- Select or define an **abstract domain**
 - based on the type of properties to prove

Abstract interpretation: Steps

- Select or define an **abstract domain**
 - based on the type of properties to prove
- Define sound **abstract transformers**
 - expressing the effect of each expression or statement on the domain

Abstract interpretation: Steps

- Select or define an **abstract domain**
 - based on the type of properties to prove
- Define sound **abstract transformers**
 - expressing the effect of each expression or statement on the domain
- Iterate abstract transformers over the domain
 - until a **fixed point** is reached

Abstract interpretation: Steps

- Select or define an **abstract domain**
 - based on the type of properties to prove
- Define sound **abstract transformers**
 - expressing the effect of each expression or statement on the domain
- Iterate abstract transformers over the domain
 - until a **fixed point** is reached

The fixed point is the over-approximation of the program

Example

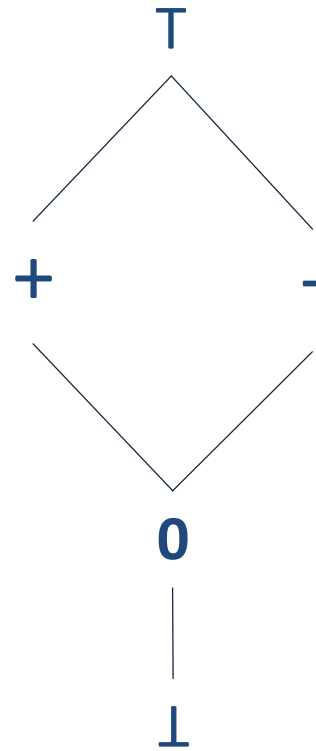
```
foo(int i) {  
  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:       y = y + 1;  
5:       i = i - 1;  
6:       goto 3;  
       }  
  
7:   assert 0 <= x + y;  
}
```


Example: Abstract domain

```
foo(int i) {  
  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```

Example: Abstract domain

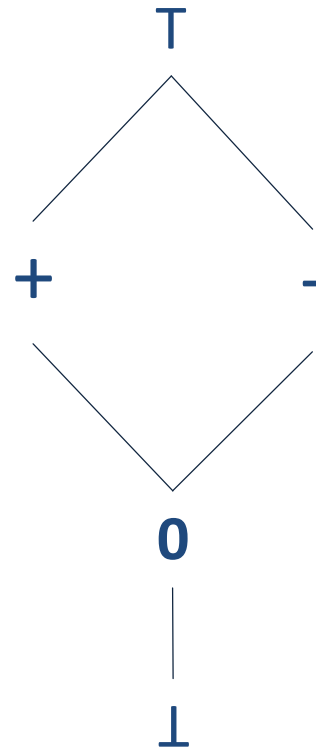
```
foo(int i) {  
  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```



Sign abstract domain

Example: Abstract domain

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= x + y;  
}
```



pc	x	y	i
2	+	⊥	T

Abstract state

Sign abstract domain

Example: Abstract transformers

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	+	T



```
4:   y = y + 1;
```



Example: Abstract transformers

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	+	T



```
4:   y = y + 1;
```



pc	x	y	i
4	T	+	T

Exercise: Abstract transformers

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	-	T



```
4:   y = y + 1;
```



pc	x	y	i
4	T	0	T

Exercise: Abstract transformers

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	-	T



```
4:   y = y + 1;
```



pc	x	y	i
4	T	0	T

Is this transformer sound?

Exercise: Abstract transformers

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	-	T



```
4:   y = y + 1;
```



pc	x	y	i
4	T	0	T

Is this transformer sound?

No!

Exercise: Abstract transformers

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	0	T



```
4:   y = y + 1;
```



pc	x	y	i
4	T	T	T

Exercise: Abstract transformers

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	0	T



```
4:   y = y + 1;
```



pc	x	y	i
4	T	T	T

Is this transformer sound?

Exercise: Abstract transformers

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	0	T



```
4:   y = y + 1;
```



pc	x	y	i
4	T	T	T

Is this transformer sound?

Yes!

Exercise: Abstract transformers

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	0	T



```
4:  y = y + 1;
```



pc	x	y	i
4	T	T	T

Is this transformer precise?

Exercise: Abstract transformers

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
4	T	0	T



```
4:   y = y + 1;
```



pc	x	y	i
4	T	T	T

Is this transformer precise?

No!

Abstract transformers

- Abstract transformers are easily sound and imprecise
 - by always returning \top
- Defining the most precise transformer is not always possible
 - called best transformer
- Precision may be sacrificed for performance reasons

Example: Fixed point

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
1	⊥	⊥	⊥



int i



Example: Fixed point

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
1	⊥	⊥	⊥



int i



pc	x	y	i
1	⊥	⊥	T

Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
1	⊥	⊥	T



```
1:  int x = 5;
```



Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
1	⊥	⊥	T



```
1:   int x = 5;
```



pc	x	y	i
2	+	⊥	T

Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
2	+	⊥	T



```
2:   int y = 7;
```



Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
2	+	⊥	T



```
2:  int y = 7;
```



pc	x	y	i
3	+	+	T

Example: Fixed point

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
3	+	+	T



```
3:  if (i >= 0)
```



Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
3	+	+	T



3: if (i >= 0)



pc	x	y	i
4	+	+	+

pc	x	y	i
7	+	+	-

Example: Fixed point

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
4	+	+	+



```
4:    y = y + 1;
```



Example: Fixed point

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
4	+	+	+



```
4:    y = y + 1;
```



pc	x	y	i
5	+	+	+

Example: Fixed point

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
5	+	+	+



```
5:    i = i - 1;
```



Example: Fixed point

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
5	+	+	+



```
5:    i = i - 1;
```



pc	x	y	i
6	+	+	T

Example: Fixed point

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x + y;  
}
```

pc	x	y	i
6	+	+	T



6: goto 3;



Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
6	+	+	T



6: goto 3;



pc	x	y	i
3	+	+	T

Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Exercise: Property

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Is the property verified?

Exercise: Property

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x + y;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Is the property verified?

Yes! The domain is precise enough!

Exercise: Property

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= x - y;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Does the property hold?

Exercise: Property

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x - y;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Does the property hold?

No!

Exercise: Property

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x - y;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Is the property verified?

Exercise: Property

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= x - y;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Is the property verified?

No! The domain is sound!

Exercise: Property

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Does the property hold?

Exercise: Property

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Does the property hold?

Yes!

Exercise: Property

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Is the property verified?

Exercise: Property

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
1	⊥	⊥	T

pc	x	y	i
2	+	⊥	T

pc	x	y	i
3	+	+	T

pc	x	y	i
4	+	+	+

pc	x	y	i
5	+	+	+

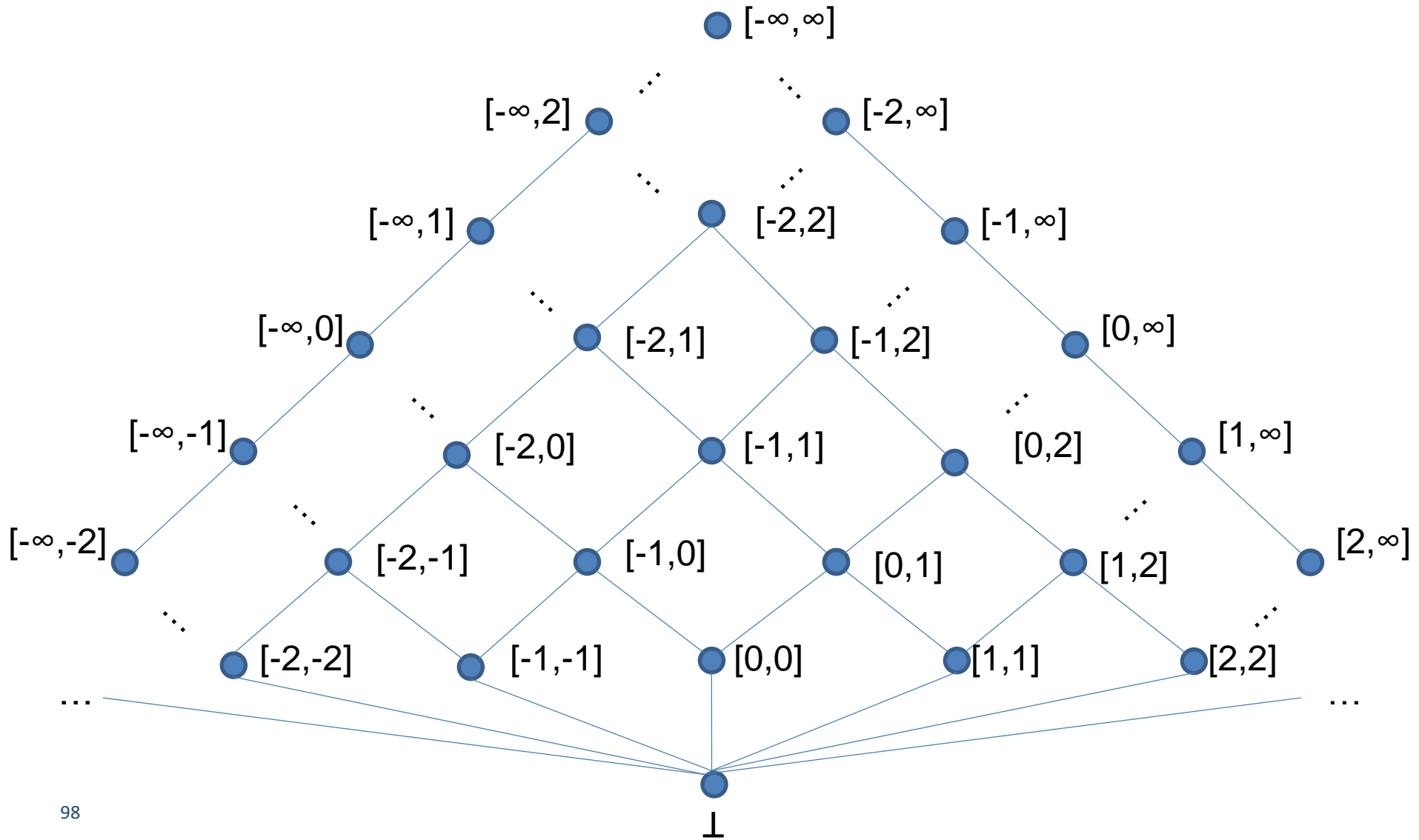
pc	x	y	i
6	+	+	T

pc	x	y	i
7	+	+	-

Is the property verified?

No! The domain is too imprecise!

A more precise abstract domain



Example: Abstract domain

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
5	$[-2, \infty]$	$[1, 7]$	$[1, 2]$

Abstract state

Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
1	⊥	⊥	⊥



int i



Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
1	\perp	\perp	\perp



int i



pc	x	y	i
1	\perp	\perp	$[-\infty, \infty]$

Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
1	\perp	\perp	$[-\infty, \infty]$



```
1:   int x = 5;
```



Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
1	\perp	\perp	$[-\infty, \infty]$



```
1:  int x = 5;
```



pc	x	y	i
2	[5,5]	\perp	$[-\infty, \infty]$

Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
2	[5,5]	⊥	$[-\infty, \infty]$



```
2:   int y = 7;
```



Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
2	[5,5]	\perp	$[-\infty, \infty]$



```
2:   int y = 7;
```



pc	x	y	i
3	[5,5]	[7,7]	$[-\infty, \infty]$

Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
3	[5,5]	[7,7]	$[-\infty, \infty]$



```
3:   if (i >= 0)
```



Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
3	[5,5]	[7,7]	$[-\infty, \infty]$



```
3:  if (i >= 0)
```



pc	x	y	i
4	[5,5]	[7,7]	$[0, \infty]$

pc	x	y	i
7	[5,5]	[7,7]	$[-\infty, -1]$

Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
4	[5,5]	[7,7]	[0,∞]



```
4:   y = y + 1;
```



Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
4	[5,5]	[7,7]	[0,∞]



```
4:   y = y + 1;
```



pc	x	y	i
5	[5,5]	[8,8]	[0,∞]

Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
5	[5,5]	[8,8]	[0,∞]



```
5:    i = i - 1;
```



Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
5	[5,5]	[8,8]	[0,∞]



```
5:   i = i - 1;
```



pc	x	y	i
6	[5,5]	[8,8]	[-1,∞]

Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
6	[5,5]	[8,8]	[-1,∞]



```
6:  goto 3;
```



Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
6	[5,5]	[8,8]	[-1,∞]



```
6:   goto 3;
```



pc	x	y	i
3	[5,5]	[8,8]	[-1,∞]

Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
3	[5,5]	[8,8]	[-1,∞]

pc	x	y	i
3	[5,5]	[7,7]	[-∞,∞]



JOIN



Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
3	[5,5]	[8,8]	[-1,∞]

pc	x	y	i
3	[5,5]	[7,7]	[-∞,∞]



JOIN



pc	x	y	i
3	[5,5]	[7,8]	[-∞,∞]

Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
3	[5,5]	[7,8]	$[-\infty, \infty]$



```
3:   if (i >= 0)
```



Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
3	[5,5]	[7,8]	$[-\infty, \infty]$



```
3:  if (i >= 0)
```



pc	x	y	i
4	[5,5]	[7,8]	[0,∞]

pc	x	y	i
7	[5,5]	[7,8]	$[-\infty, -1]$

Example: Fixed point

```
foo(int i) {  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
4	[5,5]	[7,8]	[0,∞]



```
4:  y = y + 1;
```



This will not terminate!

Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
3	[5,5]	[8,8]	[-1,∞]

pc	x	y	i
3	[5,5]	[7,7]	[-∞,∞]



~~JOIN WIDEN~~



Example: Fixed point

```
foo(int i) {  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
   }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
3	[5,5]	[8,8]	[-1,∞]

pc	x	y	i
3	[5,5]	[7,7]	[-∞,∞]



~~JOIN WIDEN~~



pc	x	y	i
3	[5,5]	[7,∞]	[-∞,∞]

Example: Fixed point

```
foo(int i) {  
  
1:  int x = 5;  
2:  int y = 7;  
  
3:  if (i >= 0) {  
4:    y = y + 1;  
5:    i = i - 1;  
6:    goto 3;  
    }  
  
7:  assert 0 <= y - x;  
}
```

pc	x	y	i
7	[5,5]	[7,∞]	[-∞,-1]

Exercise: Property

```
foo(int i) {  
  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
7	[5,5]	[7,∞]	[-∞,-1]

Is the property verified?

Exercise: Property

```
foo(int i) {  
  
1:   int x = 5;  
2:   int y = 7;  
  
3:   if (i >= 0) {  
4:     y = y + 1;  
5:     i = i - 1;  
6:     goto 3;  
    }  
  
7:   assert 0 <= y - x;  
}
```

pc	x	y	i
7	[5,5]	[7,∞]	[-∞,-1]

Is the property verified?

Yes! The domain is precise enough!

Static Program Analysis Meets Test Case Generation

Lecture 2

Maria Christakis
MPI-SWS, Germany