How it's Made: Behind the Scenes of Building Microsoft Roslyn

Kevin Pilch-Bisson
@Pilchie - kevinpi@microsoft.com
Brian Rasmussen
@Kodehoved - brianras@microsoft.com

Agenda

- Introduction
- Product Planning

- Writing Code

- Performance Testing

- Functional Testing
- Conclusion

Introduction

Roslyn

- Long lead project to provide APIs into compiler
- Lowers the barrier to entry for language tooling
- Why rewrite

History

- “FELT/Gryphon/Lanai”
- Line-by-line translation from C++ to C# and VB.
- Roslyn starts

Challenges

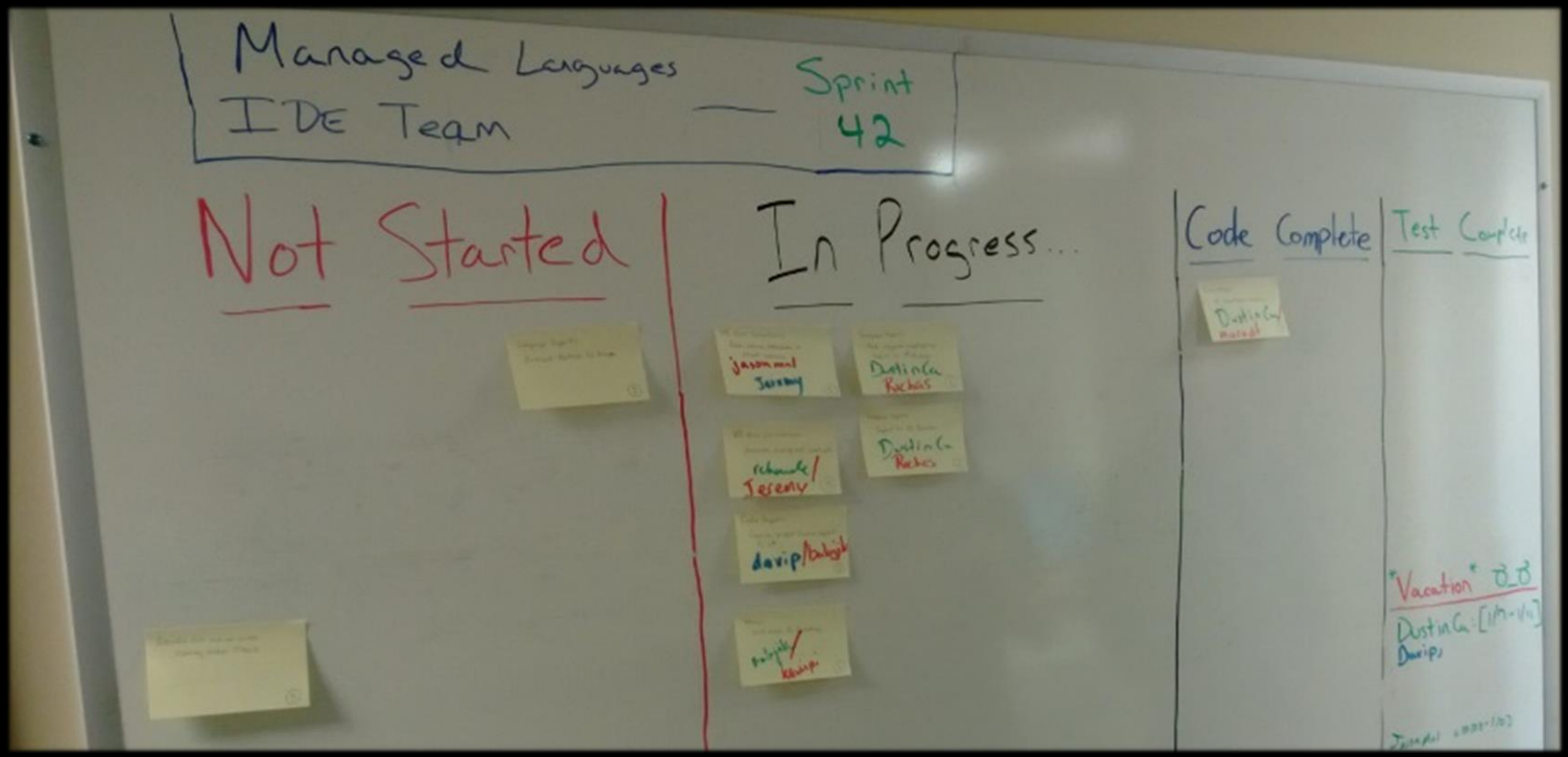
- Extremely high compatibility bar
- Hard performance targets
- Integration with legacy code

Teams

- 3 “feature teams”
  - Compiler
  - Ecosystem
  - IDE
  - Each has 4-6 dev, 4-6 qa, and 1-2 pm
- Triad model of management

Product Planning

Project Board



Product Backlog

Sprint	Points	Title
\M1\Sprint 40	2	Code Model: Support setting Type property
\M1\Sprint 40	3	Create workspace service to replace IDocumentManager
\M1\Sprint 40	8	Move Roslyn source and bugs to DevDiv2 collection
\M1\Sprint 40	5	Support async/await keywords, (awaitable), and Usage text in completion, quick info and signature help
\M1\Sprint 40	5	Support C# and Visual Basic-specific option pages in VS (except formatting)
\M1\Sprint 40	5	VB Error Corrections: Add missing metadata or project reference
\M1\Sprint 40	3	VB Error Corrections: Spell check for incorrect symbol names
\M1\Sprint 41	3	Extract Method for Async

- Planning and showcase for every sprint
  - Existing VS = good idea of tasks to complete
- 3-week sprints using story points
- Shared backlog across the 3 teams

Parallel Projects

- Drafting
  - Implement feature in a single language
  - Other language in the next sprint
- Build on what we learn
  - Detect risks

What is “Done”?

- Developer and QA person both “sign-off”
- Checklist to guide sign-off
  - Code Coverage >= 80%
  - Reviewed previous versions of VS test coverage and bugs
  - No outstanding Pri-1 bugs

“Bug Jail”

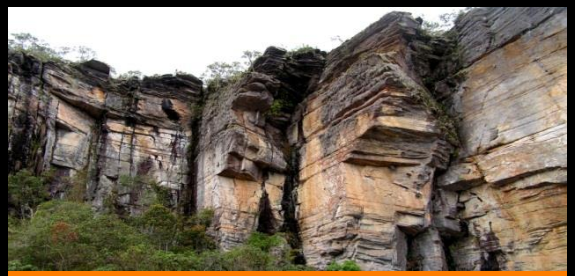
- Jail instead of bug stories
  - Tried different approaches before
- Upper and lower limit
- Fix ~1bug/day = one sprint worth of debt
  - Fewer bugs = better quality
- Bucket for “Important but can wait”

Break

10 minutes

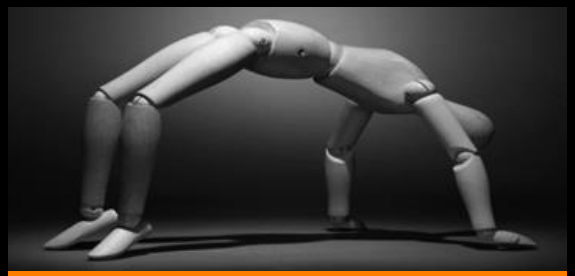
Writing Code

Pillars



Immutable

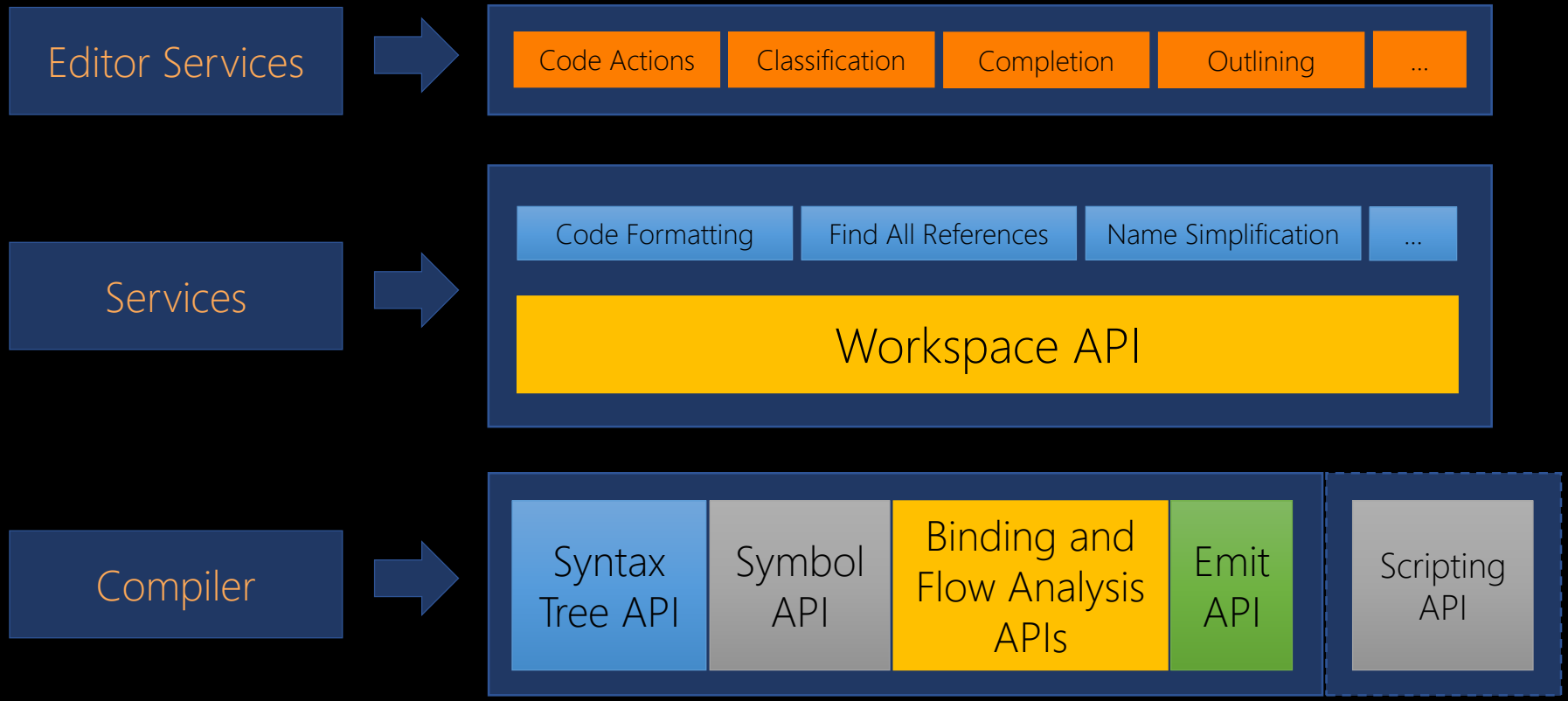
Complete



Resilient

Efficient

API Layers


Editor Services
Code Actions
Classification
Completion
Outlining
...
Services
Code Formatting
Find All References
Name Simplification
...
Workspace API
Compiler
Syntax Tree API
Symbol API
Binding and Flow Analysis APIs
Emit API
Scripting API

Demo

RoslynTestApp

- Ensures parts of Roslyn IDE experience can be hosted outside Visual Studio

- Allows us to write tooling to make development faster
  - Visualizers
  - Bug catching tools
  - Unit test generators

Unit Testing

- Currently around 40,000 unit tests
- Really not “unit” in purist sense
- Very declarative

```
        [Fact]
        public void DoesNotWorkWhenNotInStringLiteral()
        {
            VerifyNoCompletion(@"
class C
{
    static void M(string a)
    {
        throw new System.ArgumentNullException($$);
    }
}");
        }
```

Test.exe

- One stop shop for pre-checkin validation
  - Builds all required code
  - Runs all required tests
  - Has options for “extras” like code coverage, etc
- Designed to take less than 5 minutes to run.
  - Optimize when we go over 5.
  - Haven’t needed to trim what tests we run.

Continuous Integration

- Kick off TFS TeamBuild builds for every checkin
  - Debug, Release, CompilerDogfood
- Rolling builds for
  - VS integration tests, perf
- Nightly builds for
  - Dogfood installs, Perf test runs, Code Coverage
- Sprint end “real-signed” builds

The Squirrel of Shame

API Design

- 2x2-hour design meetings weekly
- Small group of people
  - deliberately scheduled in 8-person conference rooms
- Try to distill principles to guide API creation
- Review existing APIs and changes to ratify them, and then “lock them down”

API Design

- Many competing priorities
  - Ease of use versus performance
  - Level of abstraction
  - Attempting to design for the “Pit of Success”
- Vetting APIs
  - Building our own IDE using them
  - Produce CTPs and try to get others to use the APIs
  - internal partners

Dogfooding

- Microspeak for “eating our own dogfood”
- Pick up new bits daily
  - Build produces installer
- Why dogfood?
  - Subjective feedback early
  - Track performance across broad range of uses
  - Enables telemetry based decisions

Break

15 minutes

Performance Testing

Types of Performance

- Throughput
  - Conceptually simple
  - Pipeline, compiler
  - Profiling is usually useful, but memory usage is important too
- Responsiveness
  - Typing
  - Long running tasks are okay …
    - As long as we don’t wait for them to complete
    - Schedule and cancel tasks in response to input
  - Profiling can be tricky
  - GC affects responsiveness

Interaction Classes

Interaction class	Target	Upper bound	Human perception	UX Transition / Feedback*
Instant	<= 50 ms	100 ms	No noticeable delay.	See result
Fast	50 – 100 ms	200 ms	Minimally noticeable delay. No feedback necessary.	See result
Typical	100 – 300 ms	500 ms	Quick, but too slow to be described as fast. No feedback necessary.	See result
Responsive	300 – 500 ms	1 sec	Not fast, but still feels responsive. No feedback necessary.	See result / wait cursor
Continuous	>500 ms	5 sec	Medium wait, no longer feels responsive. May need feedback.	Wait cursor
Captive	>500 ms	10 sec	Long, but not long enough to do something else. May need feedback.	Progress Dialog + Cancel? or Push to background
Extended	>500ms	>10 sec	Long enough to do something else while waiting. May need feedback.	Progress Dialog + Cancel? or Push to background
Long running	>5s	> 1 minute	Users will certainly do something else.	Progress Dialog + Cancel + Suspend? or Push to background
Extra Long running*	Minutes	Hours	Users will certainly do something else.	Progress Dialog + Cancel + Suspend or Push to background

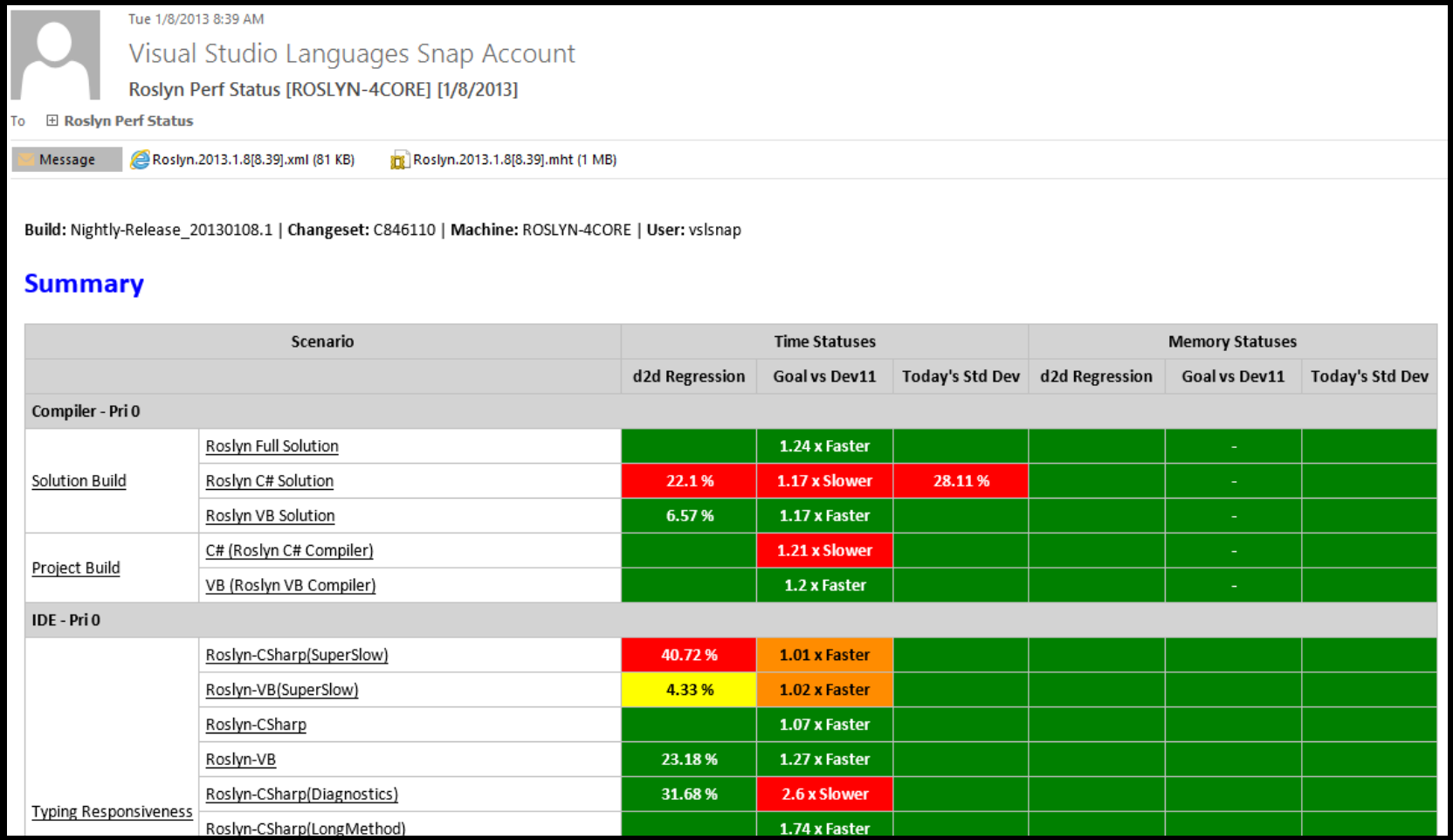
Challenges

- What to measure?
  - Giant test matrix
  - Pick key scenarios
  - Compiler: snippets and end-to-end
  - IDE: typing, navigating, renaming, etc.
- How to measure?
  - Compare against existing VS
  - Compiler: Profiling
  - IDE: Measure responsiveness

Other Issues

- Testing vs. dogfooding
  - Additional coverage – hardware and scenarios
- Robustness
  - VS is complex, a lot can go wrong
  - Fragile tests – rolling test runs
- Partner Projects
  - Use Roslyn in other scenarios

Tracking Performance



Tue 1/8/2013 8:39 AM

Visual Studio Languages Snap Account

Roslyn Perf Status [ROSLYN-4CORE] [1/8/2013]

To ⊞ Roslyn Perf Status

Message Roslyn.2013.1.8[8.39].xml (81 KB) Roslyn.2013.1.8[8.39].mht (1 MB)

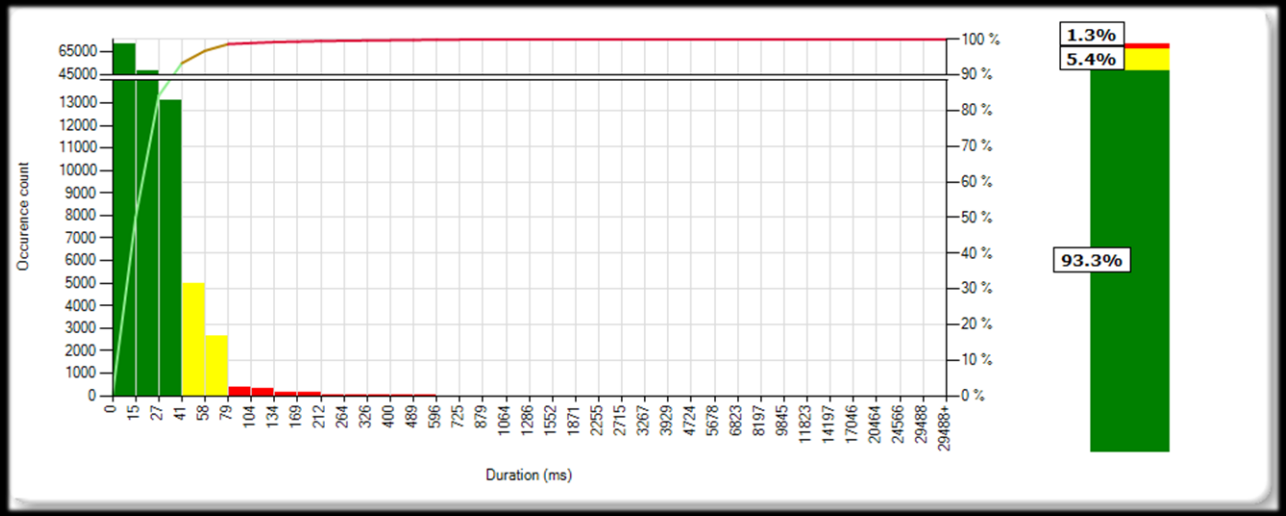
Build: Nightly-Release_20130108.1 | **Changeset:** C846110 | **Machine:** ROSLYN-4CORE | **User:** vslsnap

Summary

Scenario		Time Statuses			Memory Statuses		
		d2d Regression	Goal vs Dev11	Today's Std Dev	d2d Regression	Goal vs Dev11	Today's Std Dev
Compiler - Pri 0							
Solution Build	Roslyn Full Solution		1.24 x Faster			-	
	Roslyn C# Solution	22.1 %	1.17 x Slower	28.11 %		-	
	Roslyn VB Solution	6.57 %	1.17 x Faster			-	
Project Build	C# (Roslyn C# Compiler)		1.21 x Slower			-	
	VB (Roslyn VB Compiler)		1.2 x Faster			-	
IDE - Pri 0							
Typing Responsiveness	Roslyn-CSharp(SuperSlow)	40.72 %	1.01 x Faster				
	Roslyn-VB(SuperSlow)	4.33 %	1.02 x Faster				
	Roslyn-CSharp		1.07 x Faster				
	Roslyn-VB	23.18 %	1.27 x Faster				
	Roslyn-CSharp(Diagnostics)	31.68 %	2.6 x Slower				
	Roslyn-CSharp(LongMethod)		1.74 x Faster				

Dogfooding

- Dogfooding expands test coverage
- Getting performance data from dogfooding
- Local diagnostics Window in VS



Test Automation

- Writing tests using Tao
  - Performance specific commands
- Controlling Visual Studio
  - DTE – standard automation model
  - Internal frameworks, e.g. for injecting code
  - SendKeys for simulating typing
- SendKeys is like Portal turrets

Performance Tests


```
<?xml version="1.0" encoding="utf-8" ?>
<TaoTest xmlns="http://microsoft.com/schemas/VSLanguages/TAO">
  <InitTest>
    <StartTarget />
    <OpenProject ProjectFile="$(WorkingDir)\..\..\LegacyTest\RealworldProjects\RoslynSolutions\Roslyn-CSharp.sln"
    <ForceGC/>
  </InitTest>

  <ScenarioList>
    <Scenario Name="Type Responsiveness" Description="Type Responsiveness">
      <DelayTracker Action="Start" TestName="Roslyn-CSharp(Goldilocks)" OutputPath="$(WorkingDir)" />
      <OpenFile FileName="LanguageParser.cs"/>
      <WaitForIdleCPU/>
      <GoToLine LineNumber="9524"/>
      <PlayBackTyping TypingInputFile="PerfTests\goldilocks.input" />
      <DelayTracker Action="Stop" />
    </Scenario>
  </ScenarioList>

  <CleanupScenario>
    <CloseFile SaveFile="false"/>
  </CleanupScenario>

  <CleanupTest>
    <CloseTarget />
  </CleanupTest>
</TaoTest>
```

Results

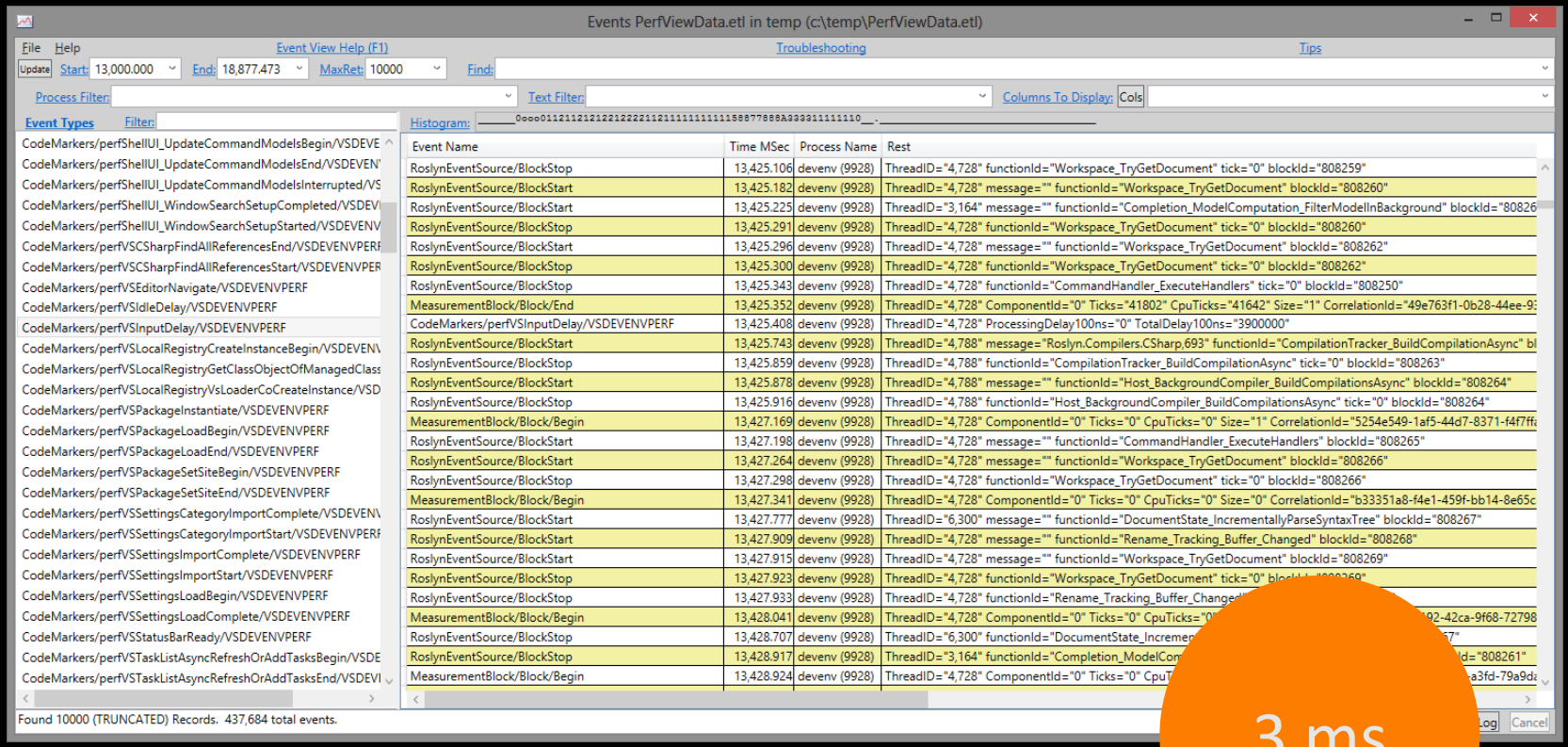
	Native Baseline	Roslyn Result	Roslyn v/s Native
Keystrokes	278	278	-1x | 0%
Average Delay	16.77 ms	7.69 ms	-2.18x | -54.14%
Keystrokes Greater than 35 ms	31	8	-3.88x | -74.19%
Keystrokes Greater than 50 ms	11	5	-2.2x | -54.55%
Keystrokes Greater than 100 ms	3		
Keystrokes Greater than 200 ms			
Percentage of Keystrokes Greater than 35 ms	11.15 %	2.88 %	-3.87x | -74.17%
Percentage of Keystrokes Greater than 50 ms	3.96 %	1.8 %	-2.2x | -54.55%
Percentage of Keystrokes Greater than 100 ms	1.08 %		
Percentage of Keystrokes Greater than 200 ms			
50th Percentile Delay	15 ms		
66th Percentile Delay	16 ms		
95th Percentile Delay	47 ms	31 ms	-1.52x | -34.04%
97th Percentile Delay	62 ms	32 ms	-1.94x | -48.39%
98th Percentile Delay	63 ms	47 ms	-1.34x | -25.4%
99th Percentile Delay	94 ms	62 ms	-1.52x | -34.04%
100th Percentile Delay	172 ms	78 ms	-2.21x | -54.65%

Monitoring

- Peeking inside the Black Box
- Event Tracing for Windows (ETW)
  - Low overhead
  - Used by Windows, GC, TPL, etc.
- Custom event sources in .NET 4.5
- PerfView
  - Analyze ETW output
  - Very powerful, steep learning curve

Demo

Information Overload



Correlating Events

- Domain Specific knowledge about typing
- What happened when this key was typed

C:\temp\Top_Typing_Delays.html C:\temp\Top_Typing_Delay...

Top Typing Delays

Back

Character	Span	Reported Processing Delay	Calculated Total Delay	EE Suspend Delays	Percent
'u'	12848.494 13389.946	547.00 ms	541.45 ms	534.79 ms	98.8 %
' '	1563.230 1659.709	94.00 ms	96.48 ms	19.02 ms	19.7 %
'f'	1268.796 1315.526	47.00 ms	46.73 ms	6.92 ms	14.8 %
'a'	1355.273 1391.682	47.00 ms	36.41 ms	16.11 ms	44.3 %
' '	13433.300 13468.766	47.00 ms	35.47 ms	0.00 ms	0.0 %
'k'	1899.390 1933.849	31.00 ms	34.46 ms	29.82 ms	86.5 %
'd'	8387.499 8417.701	31.00 ms	30.20 ms	25.56 ms	84.6 %
'l'	4349.862 4373.754	32.00 ms	23.89 ms	18.47 ms	77.3 %
' '	6363.371 6386.710	16.00 ms	23.34 ms	18.60 ms	79.7 %
'l'	6200.904 6223.589	31.00 ms	22.68 ms	16.40 ms	72.3 %
'l'	13398.276 13419.573	16.00 ms	21.30 ms	16.78 ms	78.8 %
'l'	2525.048 2544.440	16.00 ms	19.39 ms	13.00 ms	67.1 %
'j'	5827.393 5846.658	15.00 ms	19.27 ms	14.58 ms	75.7 %

Performance in a GC World

- GC basics
  - Short vs long lived
  - Cheap vs expensive
- Types and allocations
  - Reference types
  - Value types
  - Overhead
  - Locality
- Immutability
  - Create new objects vs changing state
- Allocation rate
  - Frequency of GCs
- Heap size
  - Cost of GC

Reducing Allocations

- Using the right types
  - Compiler uses many structs
- Caching vs computing
  - Difficult – still tuning
  - Don’t compute large data structures again and again
  - Cache small, frequently created objects, e.g. Task.FromResult
- Strings
  - Many duplicates – completely or partially
  - Avoid boxing, e.g. Enum.ToString

Performance Takeaways

- Instrumentation is key to understanding what is going on
- Get data automatically from dogfooding
- Select and prioritize scenarios
- If GC is a problem: Reduce allocations

Break

15 minutes

Functional Testing

Integration Testing

- Bare minimum of tests to ensure pieces fit together
- Host independent test framework
  - XML based configuration for tests based on Actions
  - Multiple implementations of Actions for multiple hosts

Demo

LegacyTest

- 80,000 small C# and VB files
- Built up over the history of the compilers to test each language feature
- Not part of automated testing
  - But unit tests are added for each discovered regression

Walkthroughs

- Introduce people sections of the API
  - How to Write a Quick Fix (CSharp).docx
- Run once per sprint
  - “Overall Goodness Factor”
- Rotate through different people

Automated Breadth Testing

- Perform some action on a large set of input
  - LegacyTest, CodePlex, etc
- Retype every file
  - Compare results of incremental parsing with full parsing
- Invoke Find All Reference on every identifier
- “Oracle” based testing
  - Automate VS2012 and compare against Roslyn

Bug Bashes

- Team-wide contest to find all the bugs in newly completed features
- Held once per-sprint
- Helps find test holes, and issues with test methodology that lead to them

App Building

- QA team is encouraged to build “side projects” that use the Roslyn APIs
  - “VB-Basic/TurboC#”
  - Various code actions
  - Source code indexing to HTML with links
- Helps vet APIs for a broader range of scenarios than just compiler + VS IDE

Conclusions

Challenges

- Extremely high compatibility bar
- Hard performance targets
- Integration with legacy code

Strategies

- Make it easy to write tests
- Run tests often
- Enable dogfooding
- Collect telemetry

Takeaways

- Test, test, test
- Telemetry
- Invest in infrastructure
- Focus on SMART perf goals

Questions?

Resources

- http://msdn.com/roslyn
- #RoslynCTP on twitter.
- kevinpi@microsoft.com
  - @Pilchie
- brianras@microsoft.com
  - @Kodehoved