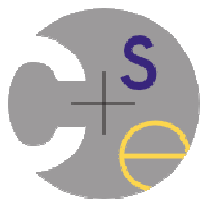


Contract.Requires(amount > 0.0);
Contract.Ensures(Balance == **Contract.OldValue**(Balance) + amount);
Contract.Invariant(Balance > 0.0);

Encouraging Effective Contract Specifications

Todd Schiller, Kellen Donohue,
Forrest Coward, Michael Ernst



University of Washington

Microsoft Code Contracts

```
public class BankAccount {
```

```
    public void Deposit(decimal amount){
```

```
        Contract.Requires(amount > 0.0);
```

```
        Contract.Ensures(Balance == Contract.OldValue(Balance) + amount);
```

```
        ...
```

```
    }
```

```
    ...
```

```
}
```

Precondition

Postcondition

- C# Syntax and Typing
- Run-time Checking
- Static Checking

What contracts do developers write?

What contracts *could* developers write?

How do developers react when they are shown the difference?

How can we use this information to make developers more effective?

Developers use contracts ineffectively

- Most contracts check for missing values, e.g. `!= null`
- Miss aspects of program behavior
- Don't (effectively) use powerful features, e.g., `object invariants`

Introduce tooling to reduce annotation burden

Make suggestions key part of tool ecosystem

Curate best practices.
It's OK to be normative

Goal: Move Developers Toward Using Contracts as Specifications

Contracts as
Assertions

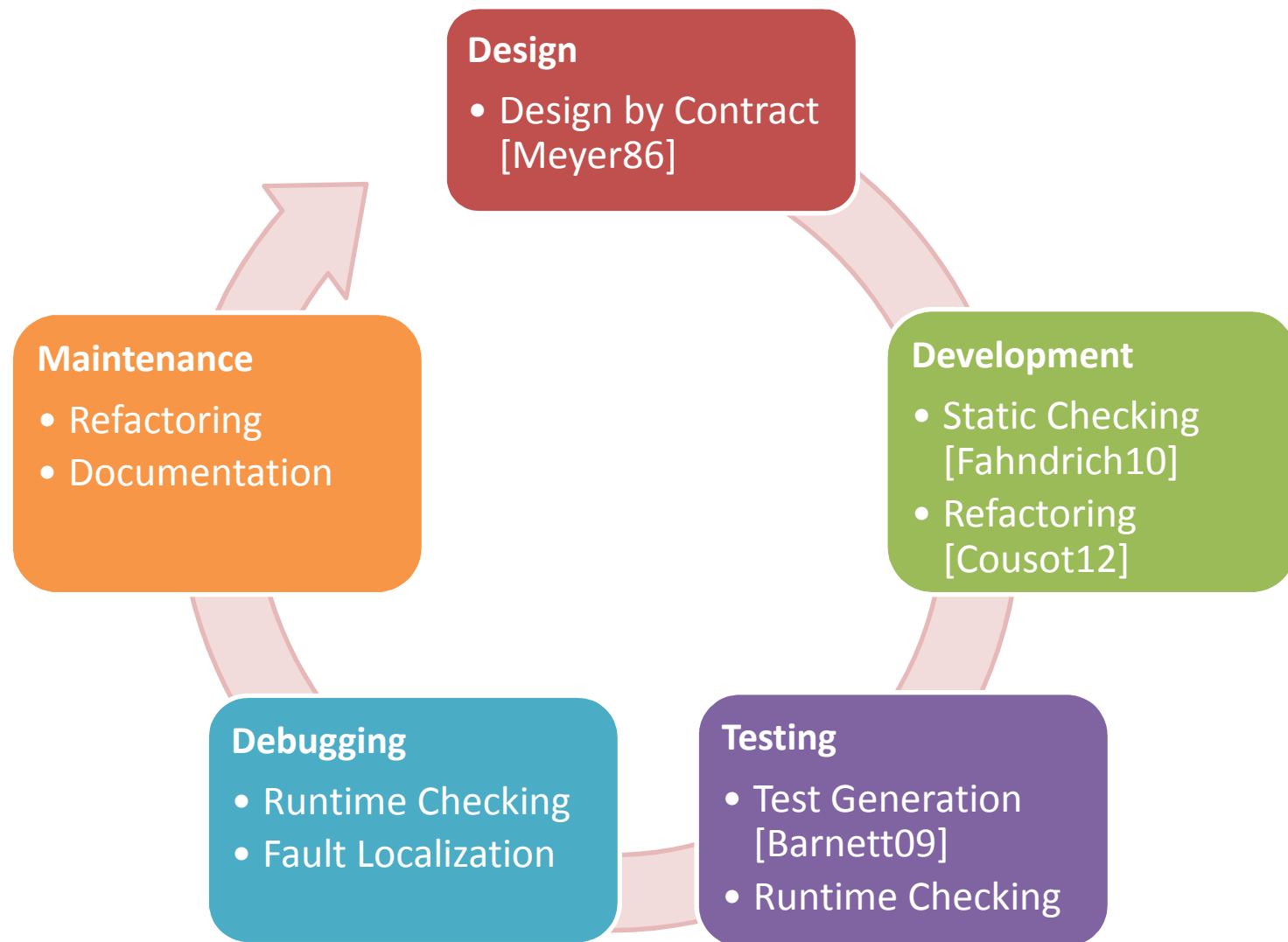


Contracts as
Functional
Specifications

- Assumption Violations

- What program *should* do
- Object Invariants
- Contracts on Interfaces

Effective Contracts Have Many Benefits

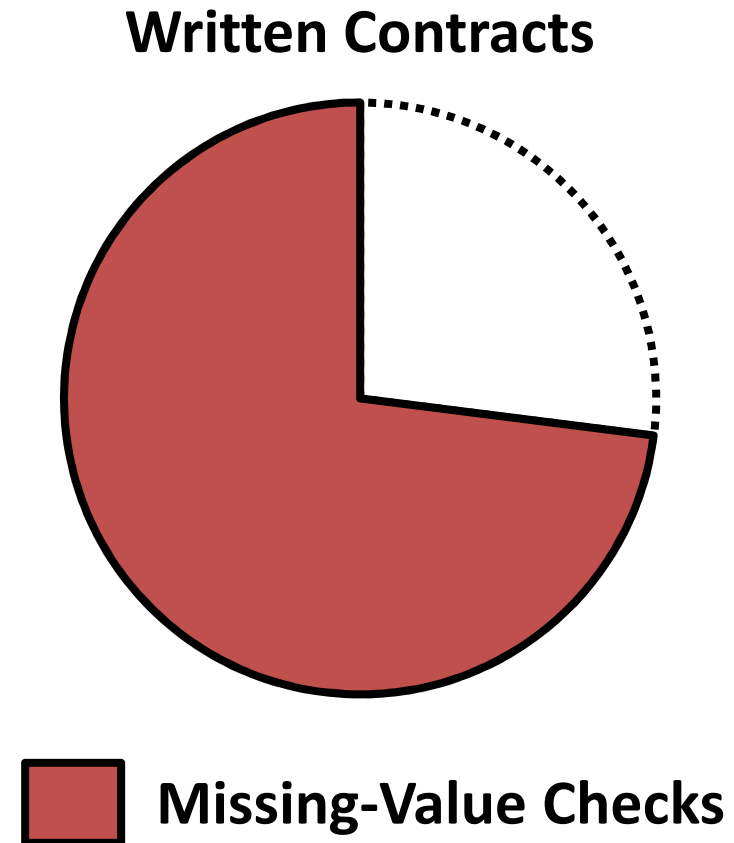


Talk Outline

1. The contracts that developers write
2. The contracts that developers *could* write
3. How developers react when shown the difference

Most Contracts Just Check for Missing Values

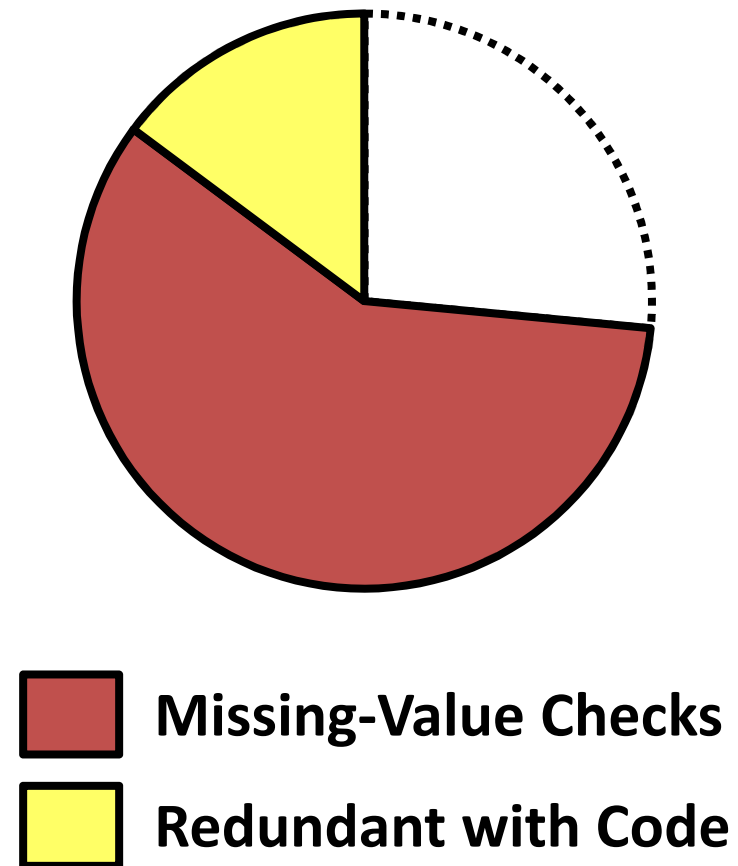
- Subjects: The 90 C# projects with Code Contracts on Ohloh
- Missing-Value: Null, Empty String, Empty Collection



Many Postconditions are Trivially Redundant with the Code

- 25% of contracts are postconditions
- 15% *of postconditions* specify:
 - The value a method returns
 - The value a property is set to

Written Postconditions



Smart Defaults Reduce Annotation Burden

Nullness: Checker Framework [Papi08] for Java assumes parameters and return values are non-null

Tool	Annotations per 1K LOC	
Checker Framework w/ Defaults	1-2 annos.	Defaults cut # of annotations needed in half
Code Contracts	2-5 annos.	

Awkward to override restrictions using Contracts:

`x != null || x == null`

Microsoft Code Contracts

```
public class BankAccount {  
  
    public void Deposit(decimal amount){  
        Contract.Requires(amount > 0.0);  
        Contract.Ensures(Balance == Contract.OldValue(Balance) + amount);  
        ...  
    }  
  
    ...  
}
```

- C# Syntax and Typing
- Run-time Checking
- Static Checking

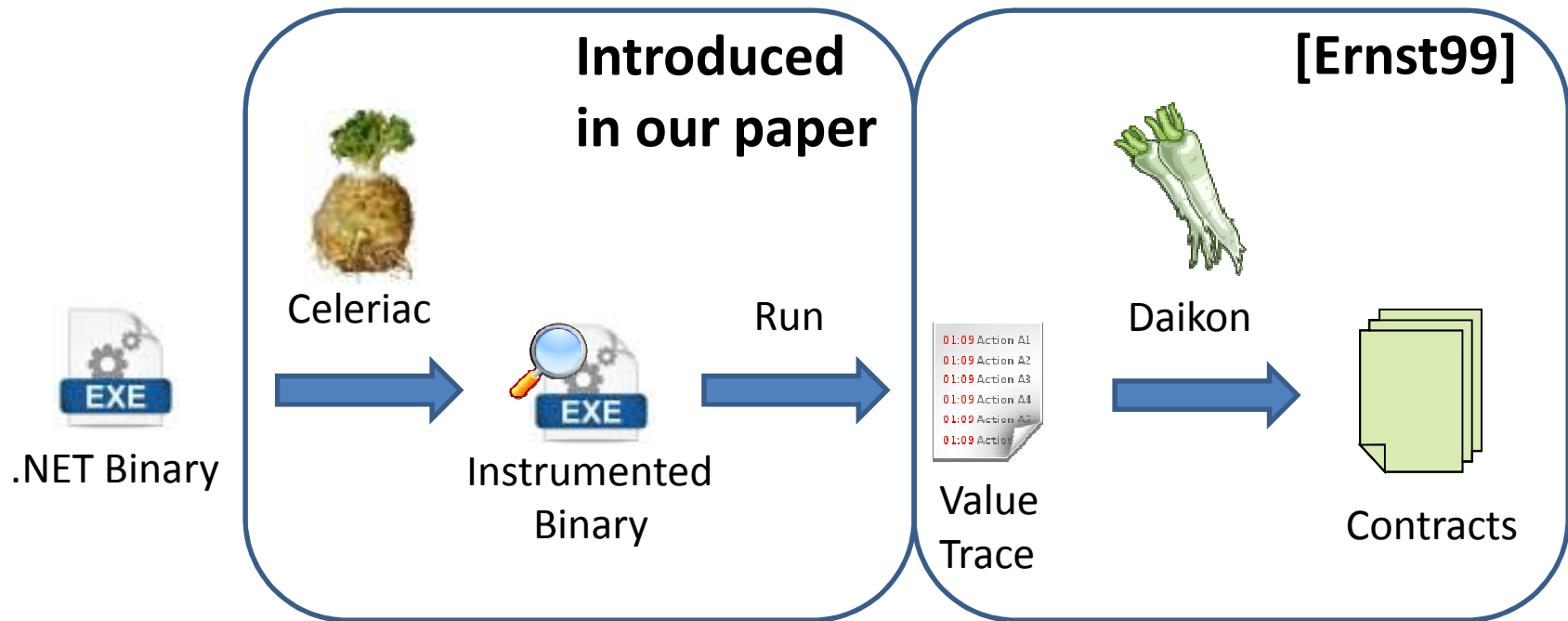
Why Don't Developers Use Functional Specifications? They are Expensive

- **Verbose**, especially involving return / pre-state expressions
 - **Contract.Result**<IEnumerable<TEdge>>()
- **High runtime cost**
 - **Contract.ForAll**(collection, elt => elt > 0)
- **No static checking**
 - dictionary[key] < array.Length

Talk Outline

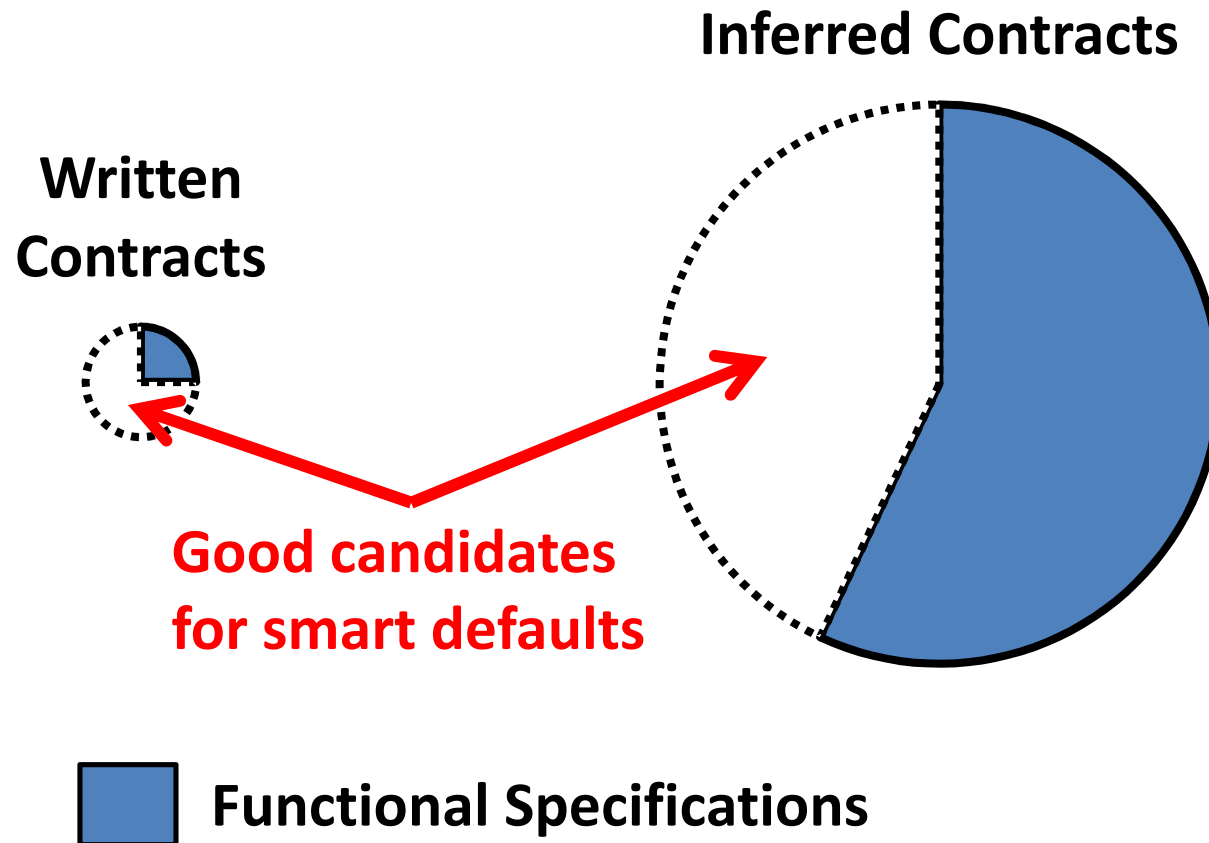
1. The contracts that developers write
2. The contracts that developers *could* write
3. How developers react when shown the difference

Inferring Contracts From Runtime Traces with Daikon + Celeriac



Celeriac: code.google.com/p/daikon-dot-net-front-end

There's a Gap Between Written Contracts and Program Behavior



Developer-Written Contracts Miss Aspects of Program Behavior

Object State:

- `this.IsUsable == (this.Reader.GetRefCount != 0)`

Relations:

- `this.programElement.ColumnNumber >= 0`

State update:

- `this.Reader.GetRefCount() >=`
`Contract.OldValue(this.Reader.GetRefCount())`

Talk Outline

1. The contracts that developers write
2. The contracts that developers *could* write
3. How developers react when shown the difference

Case Study Research Question

How do developers decide which contracts to add if contracts can be added with a single click?

Case Study Methodology

Subjects: two developers and their projects

- Sando Code Search: document indexer component
- Mishra RSS Reader: model component

Existing Contracts:

- 28 contracts across 482 methods
- All but 3 were checks for missing values

Task: Developer used interface to insert inferred contracts

C# Contract Discovery

Project To Annotate:

Indexer

Generate

Load

Filters:



Formatting:

☒ C#
☐ Daikon

Method to Annotate:

Sando.Indexer.Documents.ClassDocument:

- [-] Indexer
 - [-] Sando
 - [-] Indexer
 - [-] DocumentIndexer : 59
 - [-] Documents
 - [-] **ClassDocument : 32**
 - ... ClassDocument(Document)
 - ... ClassDocument(ClassElement)
 - ... GetFieldsForLucene : 30
 - ... GetParametersForConstruct
 - [-] CommentDocument : 33
 - [-] Converters
 - [-] DocumentFactory
 - [-] EnumDocument : 30
 - [-] FieldDocument : 30
 - [-] MethodDocument : 31
 - [-] MethodPrototypeDocument : 30
 - [-] PropertyDocument : 30
 - [-] SandoDocument : 32
 - [-] SandoDocumentStringExtension
 - [-] StructDocument : 30
 - [-] IndexFiltering
 - [-] IndexState
 - [-] Metrics
 - [-] Searching

Object Invariants

Type Definition

No XML documentation is available.

No `class` invariant method was found; Insert a contract to create it.

Invariants (32)

Filters (1)

- ▼ this
- ▼ this.GetFieldsForLucene()
- ▼ this.programElement

Case Study Research Question

How do developers decide which contracts to add if contracts can be added with a single click?

Differing Viewpoints to Inserting Contracts

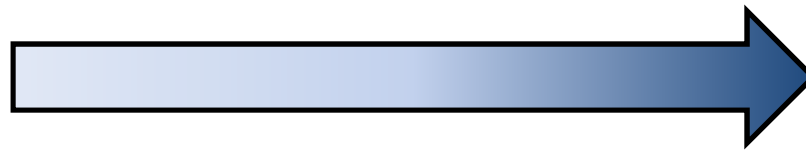
- Sando: in favor of automatically inserting all contracts above some confidence threshold
- Mishra Reader: chose not to insert many valid contracts
 - Avoiding code bloat
 - Fear of runtime overhead
 - Belief that contracts should only be written at module boundaries (public methods)

Suggestions are Beneficial (Up to a Point)

- Tool suggested types of contracts developers would not have thought of
 - e.g.: `Contract.ForAll(collection, elt => elt > 0)`
- Not a perfect substitute for training
 - Sando developer, unaware of object invariant and interface contracts, overlooked tool's suggestions

Training Affects How Contracts Are Used

Contracts as
Assertions



Contracts as
Functional
Specifications

Opportunities to train developers via the tooling itself

- Identifying features that developer is under-utilizing
- Can supplement sound static-checker inference with more expressive inference

UI Grouping Schemes to Encourage Functional Specifications

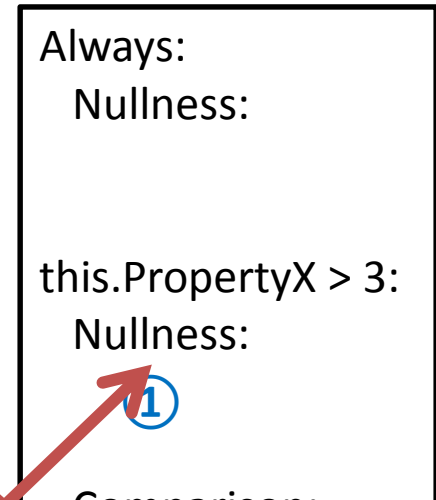
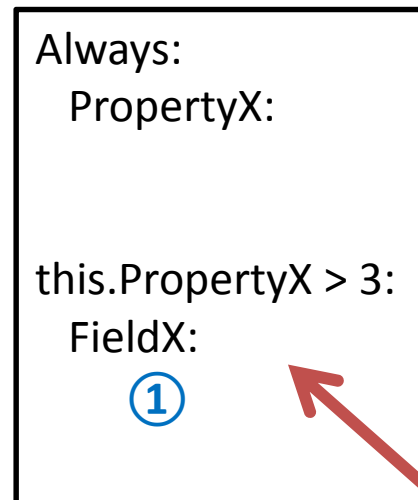
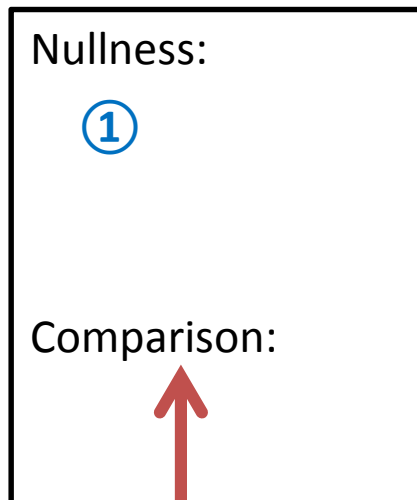
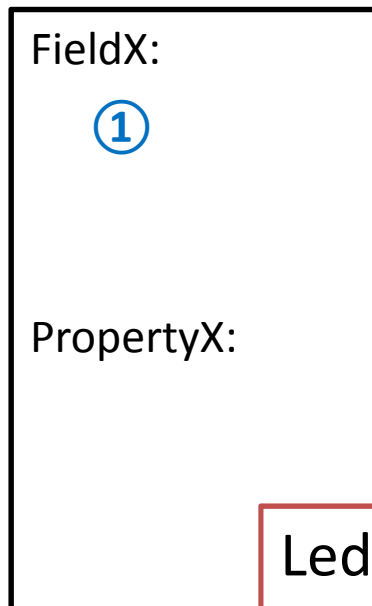
① `(this.PropertyX > 3)` implies `(this.FieldX != null)`

By variable

By kind

By antecedent / var

By antecedent / kind



Led developers to discover kinds of contracts they had not considered before

Grouping by condition did not help the developers reason about implications

Related Work

- Contracts in the Wild:
 - Chalin06: Eiffel programs have a lower proportion of non-null checks, higher proportion of postconditions
 - Estler14: Eiffel, JML, and C# contracts are stable over time; preconditions are larger than postconditions
- Human Factors:
 - Polikarpova09: Daikon finds contracts that developers missed
 - Johnson13: false positives and inadequate presentation prevent uptake of static analysis tools

Conclusion: Both Tooling *and* Training are Required for Usability

- Most check missing values, e.g. `!= null`

Introduce tooling to reduce annotation burden

- Miss aspects of program behavior

Make suggestions key part of tool ecosystem

- Don't (effectively) use powerful features, e.g., object invariants

Curate best practices. It's OK to be normative

Tools and Data: <http://bit.ly/code-contracts>

Lifecycle Not Ideal in Practice

Annotations are too heavy especially the Result/Old syntax is horrid.

The visual studio editor extension is buggy [...] Seeing contracts easily from the call site would be a huge factor in convincing less enthusiastic developers about the benefits.

[The static checker is] too slow, complicated and not expressive enough.

Increased build time is a big problem!

I am not yet totally convinced that [Code Contracts] are ready for prime-time

Subject Projects

Subject Program	Domain	Code Contract Use	Other Quality Tools Used
Labs Framework (11K SLOC)	API exploration framework	Static checking	StyleCop
Mishra Reader (19K SLOC)	RSS reader	Debugging concurrent code	Jetbrains R#
Sando (24K SLOC)	Code search	Early runtime error detection	
Quick Graph (32K SLOC)	Algorithms and data structures	Pex / Testing	Pex

Contract Inserter Interface

Four possible actions:

- Add as contract
- Add as documentation
- Mark as false
- Ignore as implementation detail

Null-checks Can be Expressive

```
public ComplicatedType Foo(. . .){  
    Contract.Ensures(Contract.Result<ComplicatedType>() != null);  
    . . .  
}
```

Types + Contracts Guarantee:

- Methods Signatures + Method Contracts
- Object Invariants

Tool Information

Celeriac: Contract Inference via Runtime Tracing

<https://code.google.com/p/daikon-dot-net-front-end>

Contract Inserter: Visual Studio Add-in

<https://bitbucket.org/fmc3/scout>

Type-State Example: Degenerate Behavior Encoding

```
public class Subscription{  
    public SubscriptionsList SubscriptionsList { get; private set; }  
  
    public void AddItem(Item item) {  
        Contract.Requires(SubscriptionsList != null, "Call Initialize first");  
        ...  
    }
```

All contracts use != null

[InvariantMethod]

```
public void ObjectInvariant(){  
    ...  
}
```

Can't write an invariant

Type-State Example: Application-Specific Property Encoding

```
public class Subscription {  
    public SubscriptionsList SubscriptionsList { get; private set; }  
    public boolean IsInitialized { get; private set; }  
  
    public void AddItem(Item item) {  
        Contract.Requires(IsInitialized, "Call Initialize first");  
        ...  
    }  
  
    [InvariantMethod]  
    public void ObjectInvariant(){  
        Contract.Invariant(!IsInitialized || SubscriptionsList != null);  
        ...  
    }  
}
```

Implications can be tricky
for multiple states

Mishra Reader: Concurrent Debugging via Nullness Checks

Model subcomponent (of MVC architecture) contained just 11 contracts across 80 classes and 360 methods:

- 10 argument non-null preconditions
- 1 invariant: `UnreadCount >= 0`

Pattern Example: Encoding Type-State with Contracts

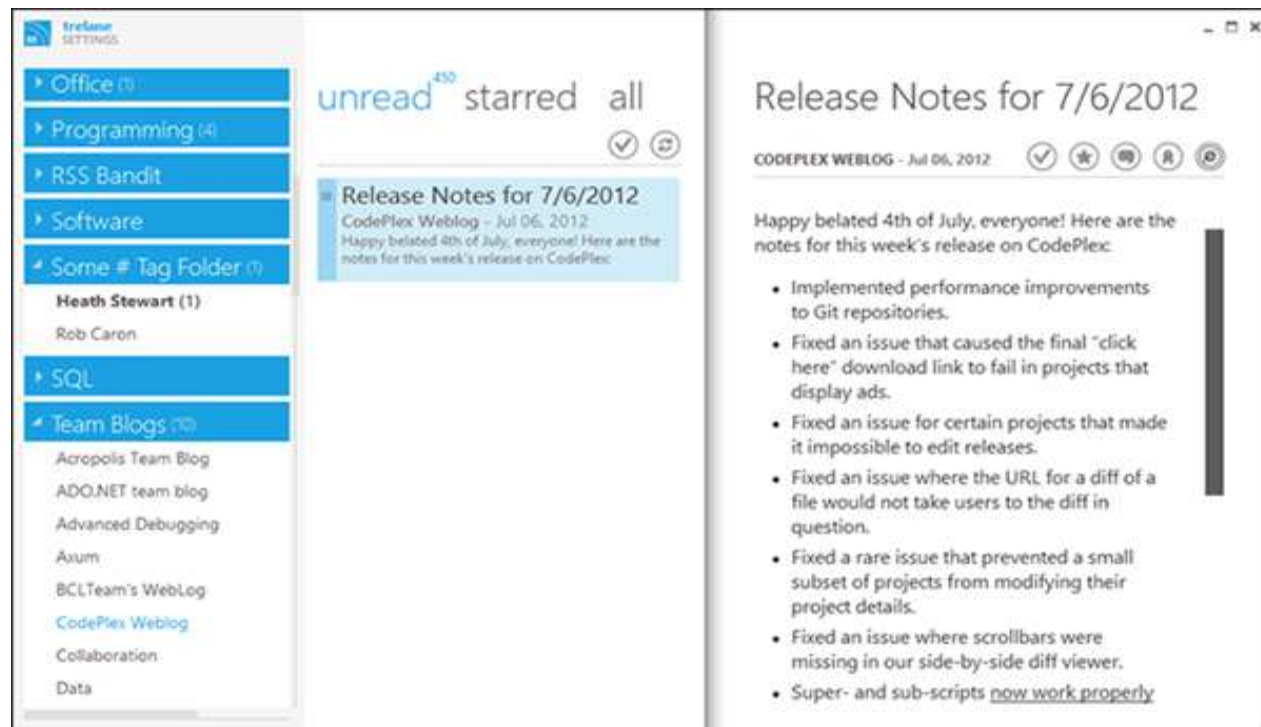
Basic Idea:

- Expose Properties indicating state, e.g., IsOpen
- Contracts contain implications based on state
- Postconditions encode transitions

Observation: only see this pattern in projects that use the static checker

Case Study: Mishra News Reader

Lead developer introduced Contracts to help debug concurrent code



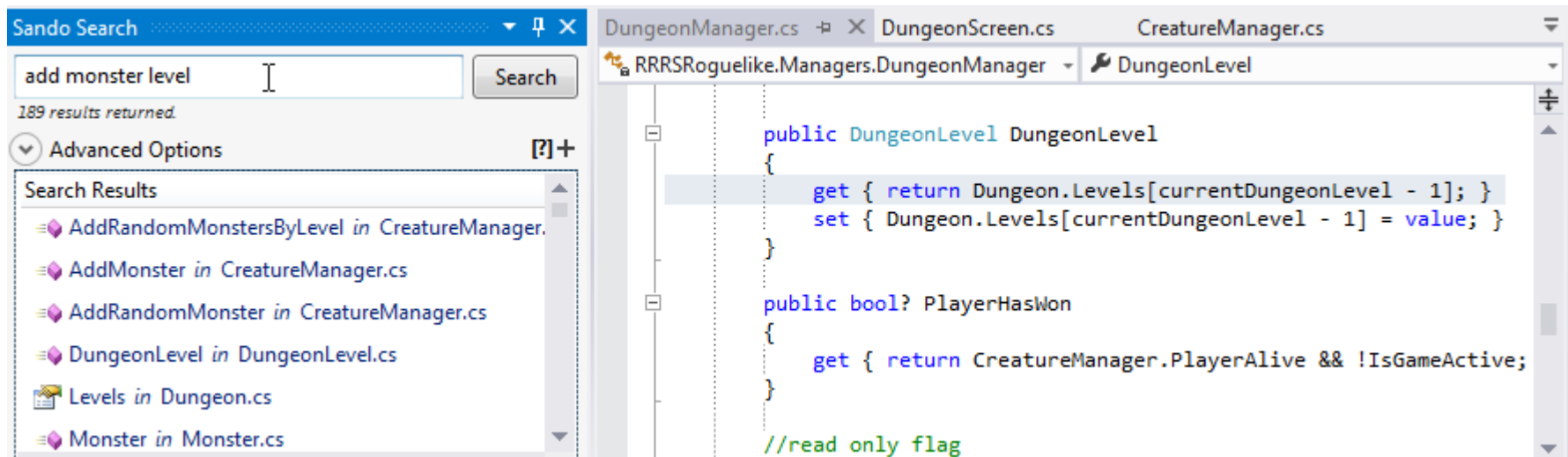
Mishra Reader: Concurrent Debugging via Nullness Checks

Model subcomponent (of MVC architecture) contained just 11 contracts across 80 classes and 360 methods:

- 10 argument non-null preconditions
- 1 invariant: `UnreadCount >= 0`

Case Study: Sando

Introduced Code Contracts after major contributor saw a webinar



Sando: Used Contracts like Assertions

Indexer component contained 17 contracts across 34 classes and 182 methods:

- 12 non-null checks
- 4 non-empty checks
- 1 implication:

`!criteria.SearchByUsageType || criteria.UsageTypes.Count > 0`