Non-null References by Default in the Java Modeling Language

Patrice Chalin

Dependable Software Research Group (DSRG)

Computer Science and Software Engineering Department

Concordia University

Montreal, Canada

Specification and Verification of Component-Based Systems (SAVCBS'05)

September 5-6, 2005 at ESEC/FSE 2005, Lisbon, Portugal

Outline

- Introduction
  - Null pointer exceptions
  - Better detecting them statically using JML …
- …

Null Pointer Dereference

- Quite a common symptom of a software bug in C based languages:
  - Java: results in a NullPointerException.

- Static analysis tools
  - Code analysis
  - Detect possible null pointer dereferences.
  - Limited success if given only code.

Code + Specs = Better Detection

- Analysis tools can detect a large proportion of null pointer exceptions when programs are suitable annotated.
- Some tools support a minimal set of annotations (e.g. declaration modifiers)
  - FindBugs: @NonNull
- Our focus: the Java Modeling Language …

Java Modeling Language (JML)

- #1 Behavioral Interface Specification Language (BISL) for Java.
- Supports
  - declaration modifiers, e.g. non_null.
  - Contract-based specification
    - Class invariants
    - Method specifications via pre-, post-conditions
  - More general: BIS

JML Tools

- Various levels of checking (Tool)
  - Run-time assertion checking (JML RAC)
  - Extended static checking (ESC/Java2)
  - (Full) Program verification (LOOP)
- ESC/Java2
- What does JML look like …?

JML Example

```
public abstract class Greeting {
   // …
}
```

- Illustrates JML.
- Illustrates means by which declarations can be constrained to be non-null.

Greeting Class – declaration modifiers

```
private /*@ spec_public non_null */
    String nm;

/*@ public normal_behavior
  @
  @  requires !aNm.equals("");
  @  modifies nm;
  @  ensures nm == aNm;
  @*/
public void set(/*@ non_null @*/ String aNm)
{
   nm = aNm;
}
```

Greeting Class – non-null assertions

```
private /*@ spec_public non_null */
    String nm;

/*@ public normal_behavior
  @  requires aNm != null &&
  @           !aNm.equals("");
  @  modifies nm;
  @  ensures nm == aNm;
  @*/
public void set(String aNm)
{
   nm = aNm;
}
```

Experiences Writing JML

- JML, like Java assumes references can be null.
  - Extra annotation required to constrain as non-null
- Notice that in the previous example:
  - Two reference type declarations.
  - Both constrained to be non-null.

Hypothesis

- After having written JML for a few case studies it seemed that we were writing

  ```
  /*@ non_null @*/
  ```

  very often.

- Study conducted to test hypothesis:
  - *By design, the majority of reference type declarations that are meant to be non-null.*

Study Overview

- Choose a (random) sampling of Java source files from four projects.
- Added non-null annotations (JML) based on our understanding of the design.
- Measured proportion of non-null declarations.

Study

We now explain

- Metrics
- Tool
- Study subjects
- Procedure
- Results
- (Threats to validity)

Study: Metrics (principal ones)

Number of declarations (per file)

- that are of a reference type (*d*)
- specified to be non-null (*m*, hence $m \leq d$).

Study: Main Statistic

- Proportion of reference type declarations that are non-null.
- $x = m / d$

Measuring d, number of ref. decl.

Count:

- Fields
  - Instance
  - Static
- Methods (return types)
- Method parameters.

Excluding: local variables

Measuring m, number of non-null d

- Declaration modifier
  - `/*@ non_null @*/`
- Assertions
  - `o != null` and variants `!(null == o)`, …
  - `o instanceof C`
  - `\fresh(o)`
  - `\nonnullelements(o)`

Measuring m for Fields

```
/*@ non_null @*/ Object o1;

Object o2;
//@ invariant o2 != null;

static Object a[];
//@ static invariant
//@    \nonnullelements(a);
```

Measuring m for Methods & Param.

- Some challenges
  - Multiple specification blocks.
  - Normal and exceptional behavior.

Measuring m for Methods

```
/*@  normal_behavior
  @    requires i == 0
  @    ensures  \result != null
  @          && \result.equals("zero");
  @ also
  @  normal_behavior
  @    requires i > 0;
  @    ensures  \result != null
  @          && \result.equals("positive");
  @ also
  @  exceptional_behavior
  @    requires i < 0;
  @    signals(Exception e) true;
  @*/
/*@ pure @*/ String m(int i) {    ... }
```

[TBC] Measuring m for Parameters

- Similar to previous case …
  - Except that parameter is not counted as non_null if there is a signals only clause.

Measuring m for Overriding Methods

- Cannot only examine specification of method.
- Must consider method specification as given in ancestor classes.

Measuring m for Overriding Methods

```
class Greeting {
//@ ensures \result != null;
//@       && !\result.equals(""); public
  abstract /*@ pure @*/ String greeting();
}

class FrenchGreeting extends Greeting {
  //@ also
  //@ ensures \result.equals("Bonjour ");
  public /*@pure non_null*/
  String greeting() { return "Bonjour "; }
}
```

Tool

- Adapted the ISU JML checker
  - Uses tool to parse and type check.
  - Accumulates metrics during/after type checking.

Study Subjects

1. ISU JML checker
2. ESC/Java2
3. Tallying subsystem of Koa
   - Dutch internet voting application
   - Used in 2004 European parlementary elections.
4. Web-based Enterprise Application framework and samples (SoenEA) used at Concordia.

Study Subjects: Encompassing System

	ISU Tools	ESC Tools	SoenEA	Koa e-Voting	Total
# of files	831	455	52	459	1797
LOC (K)	243	124	3	87	457
SLOC (K)	140	75	2	62	278

Study Subjects: Overall Project

	ISU Tools	ESC Tools	SoenEA	Koa	Total
# of files	831	455	52	459	1797
LOC (K)	243	124	3	87	457
SLOC (K)	140	75	2	62	278

Study subject	Checker	ESC/ Java2	SoenEA	Koa TS	Total
# of files	217	216	52	29	514
LOC (K)	86	63	3	10	161
SLOC (K)	58	41	2	4	104

Study Subjects

- Why these?
  - They were partly annotated already.
  - We were familiar or knew someone familiar with the design and implementation.
  - Offer an interesting selection
    - Web apps are a common use for Java.

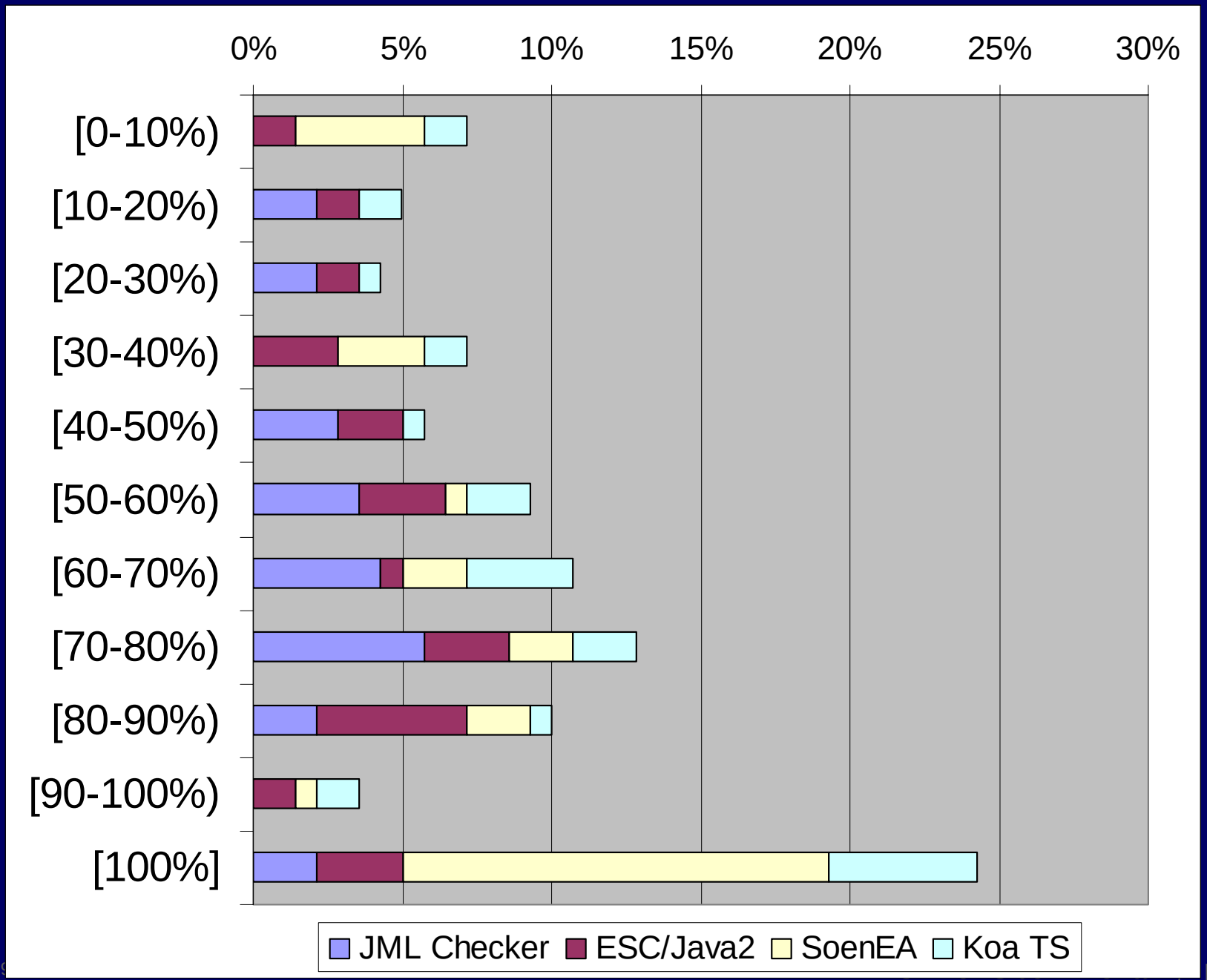
Procedure

- Choose source files:
  - Either used entire component in study, or took a random election of 35 files.
- Annotated source files by adding non-null constraints.
  - Stop annotating once mean – error > 50%.

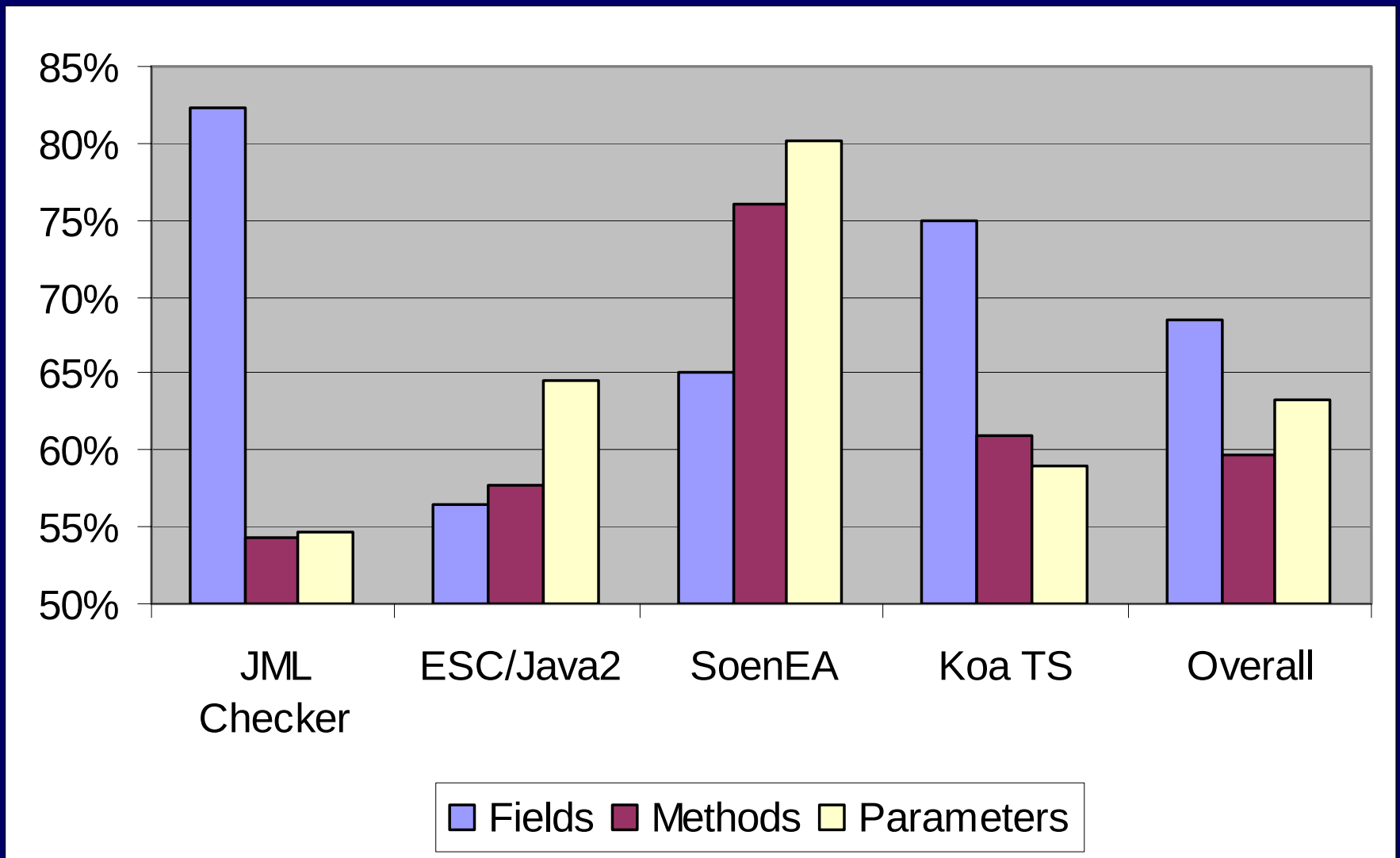
Results: Summary

	JML Checker	ESC/ Java2	Soen-EA	Koa TS	Sum or Av.
n	35	35	41	29	140
N	217	216	41	29	503
$\sum d_i$	376	807	231	564	1978
$\sum m_i$	210	499	177	368	1254
$\sum d_i / \sum m_i$	56%	62%	*77%*	65%	63%
mean $(\underline{x})$	59%	60%	*72%*	64%	64%
std.dev.(s)	0.24	0.31	0.37	0.32	-
E (α=5%)	7.4%	9.3%	-	-	-
μ_{min}	52%	51%	72%	64%	60%

Results: % of files having a value for x in given range



Mean of x by kind of declaration



Threats to (Internal) Validity

- Main threat: we were wrong in annotating a declaration as non-null.
- Mitigation: we were conservative in our annotation exercise.
- Solution: run ESC/Java2 ESC/Java2 on the study subjects to validate our specifications. (50% done)

Improving JML

- If well over 50% of decl. are non-null, why not adopt non-null as the default?

Advantages:

- Non-null is a safer default.
- Without any effort, unannotated source files would be over 50% correct.
- Much less effort to correct remaining declarations by marking them *nullable*.

Language design goals of JML

- Adhere to the semantics of Java to the extent possible.
- Do not surprise Java developers when semantics differ …
  - i.e. Give some explicit indication

JML: Declaration Modifiers

- Class modifier warning developer of different default:
  - `/*@ non_null_by_default @*/`
  - Un annotated declarations would be implicitly declared `non_null`.

- To override default:
  - `/*@ nullable @*/`

JML Tool Help

- Help developers learn about the new default.
- JML checker to warn if class default is not explicitly specified as either:
  - `non_null_by_default`
  - `nullable_by_default`

Feature Support

- JML checker and RAC:
  - 97% complete
  - (keyword change from “null” to “nullable”)
- ESC/Java2
  - 40% complete

Other Languages / Tools ...

- Supporting non-null types and/or annotations.

Splint

- Static checker for C.
- Mainly supports annotation pragmas.
- Default: non-null.
- Declaration modifier: @null.

Nice

- Essentially an enriched variant of Java.
- Supporting
  - parametric types,
  - multi-methods, and
  - contracts
  - among other features
- Default: non-null.
- ?*T* denotes the type of nullable *T*.

ECMA Eiffel Standard (2005)

- Introduces notion of
  - Attached types.
  - Detachable types.

- Default: non-null (attached).
- ?*T* denotes a detachable *T*.

Spec#

- A superset of C# supporting:
  - contracts,
  - checked exceptions and
  - non-null types.
- For backwards compatibility: default is nullable types.
- *T*! denotes non-null type of *T*.

Summary

- Null pointer exceptions can practically be eliminated from properly JML annotated Java code.
- Study: over 50% of decl. non-null in Java.
- JML:
  - New declaration modifiers to support
  - New default of non-null
- Others also believe this is a good idea.

Thank you

- Questions / Remarks?

Greeting Class (part 2)

```
//@ ensures
  \result.equals(greeting()+nm);
public /*@ pure non_null @*/ String
  welcome() {
    return greeting() + nm;
}

//@ ensures \result != null;
//@       && !\result.equals("");
  public abstract /*@ pure @*/ String
  greeting();
```