Defect Detection at Microsoft – Where the Rubber Meets the Road

Manuvir Das
(and too many others to list)

Program Analysis Group
Center for Software Excellence
Microsoft Corporation

Bottom line

- Defect detection tools based on program analysis are here to stay
- A short story on adoption and deployment
- The target customer is a software developer, not a programmer

Why me?

- Program Analysis group this month
  - Filed 7000+ bugs
  - Automatically added 10,000+ specifications
  - Answered hundreds of emails

  (one future version of one product)
- We are program analysis researchers
  - but we live and breathe deployment & adoption
  - and we feel the pain of the customer

Defect detection tools based on program analysis are here to stay

Program analysis is here today …

- Inter-procedural symbolic evaluation
  - PREfix: filed 3000+ bugs this month
- Inter-procedural path-sensitive dataflow analysis
  - ESP: filed 500+ bugs this month
- Intra-procedural abstract interpretation
  - espX: filed 3000+ bugs this month
- Inter-procedural dataflow analysis
  - SALinfer: added 10,000+ specifications this month

… and will be here tomorrow

- Analysis tools are integrated into and enforced in the development process
  - opening the door for new and better tools
- 3000+ developers are adding specifications
  - opening the door for modular analysis
- Developers are getting access to extensible analysis tools
  - opening the door for domain specific tools

A short story on adoption and deployment

Longhorn today

- Monthly central runs of global analysis tools
  - PREfix, ESP
  - Defects auto-inserted into central bug database
  - Bug caps and RI criteria in place
  - Ranking, filtering, triage, support
  - Release management drives the bugs
  - Message suppression in the defect database

Longhorn today

- Developer desktop use of local analysis tools
  - PREfast, espX
  - Installed and enabled by default for all developers
  - Tools run incrementally with the build
  - Defects produce build breaks and errors
  - RIs are validated and rejected on failure
  - Message suppression in the source code

Longhorn today

- Mandated use of specifications that describe contracts on function interfaces
  - SAL: Standard Annotation Language
  - Focus: buffer overruns, pointer usage
  - Supported by Visual Studio
  - Windows public headers decorated with SAL

Longhorn today

- Central runs + desktop use of automatic specification inference
  - SALinfer
  - Run on special branch, results stored on a server
  - Desktop client queries server and patches code

Forcing functions for change

- Gen 1: Manual Review
  - Too many paths
- Gen 2: Massive Testing
  - Inefficient detection of common patterns
- Gen 3: Global Program Analysis
  - Stale results
- Gen 4: Local Program Analysis
  - Lack of context
- Gen 5: Specifications

The target customer is a software developer, not a programmer

See the big picture

- You are selling an expensive product
  - Time is REAL, in the large and the small
- The customer only cares about the end to end experience
  - Remember Amdahl's Law!
- The customer is always right
  - Understand, then improve and educate

Don’t bother doing this without -

- No-brainer must-haves
  - Defect viewer, docs, champions, partners
- A mechanism for developers to teach the tool
  - Suppression, assertion, assumption
- A willingness to support the tool
- A positive attitude

Understand the customer

- Software developers are time constrained
- Software developers have other options
- Software developers must follow process

- Feel their pain!

Myth 1 – Accuracy matters

- The real measure is Fix Rate
- Centralized: >50%
- Desktop: >75%
- Specification inference
  - Is it much worse than manual insertion?

Myth 2 – Completeness matters

- Complete – find all the bugs
- There will never be a complete analysis
  - Partial specifications
  - Missing code
- Developers want *consistent* analysis
  - Tools should be stable w.r.t. minor code changes
  - Systematic, thorough, tunable program analysis

Myth 3 – Developers dislike specifications

- Control the power of the specifications
- This will work
  - Formalize invariants that are implicit in the code
- This will not work
  - Re-write code in a different language that is amenable to automated analysis
- Think like a developer

Don’t break the shipping code ☺

- Before:

  b = a + 16; Use(b);

- After (correct code):

  __invariant(a); b = a + 16; Use(b);

- After (incorrect code):

  b = __invariant(a) + 16; Use(b);

- Incorrect usage silently breaks the code!

Summary

- Defect detection tools based on program analysis are here to stay
- A short story on adoption and deployment
- The target customer is a software developer, not a programmer

- Where does the rubber meet the road?

Microsoft®

http://www.microsoft.com/cse
http://www.microsoft.com/cse/pa
http://research.microsoft.com/manuvir

Backup slides

Generation 1: Manual Review

- Code reviews, penetration teams
- Lessons:
  - Test all the execution paths
- Environmental changes:
  - Size of codebase starts to increase

Generation 2: Massive Testing

- More testers than developers
- Massive pre-release testing effort

- Lessons:
  - Many bugs follow similar patterns

- Environmental changes:
  - Internet changes exposure of interfaces

Generation 3: Global Analysis

- Centralized use of PREfix
  - Inter-procedural symbolic evaluation
- Lessons:
  - Ease of fix is the critical criterion
  - Late results equal stale results
- Environmental changes:
  - Security becomes Priority 1

Generation 4: Local Analysis

- Desktop use of PREfast
  - Intra-procedural syntactic & dataflow analysis
- Lessons:
  - Lack of context leads to noise
  - Source code lacks information
- Environmental changes:
  - Security becomes Priority 0

Generation 5: Specifications

- Annotations on function interfaces + deep local analysis of function implementations
- Focus: buffer overruns, pointer usage
- Standard Annotation Language (SAL)
  - Will be supported by Visual Studio
  - Will be used to decorate Windows public headers
  - Interface contracts (preconditions/postconditions)
  - Extensions to the type system

SAL Example

Requirement on foo's callers: must pass a buffer that is len elements long

```
void foo(pre elementCount(len) int *buf, int len)
{
```

Assumption made by foo: buf is count elements long

… Local Checker: Do the assumptions imply the requirements?

Requirement on foo: argument buf is len*4 bytes long

```
    memset(buf, 0, len*sizeof(int));
}
```

Requirement on memset's callers: must pass a buffer that is len bytes long

```
int *memset(pre byteCount(len) void *dest, int c, size_t len);
```

Defect Detection Process