

Software Excellence via Program **Analysis** at Microsoft

Used by permission for 17-654/17-754:
Analysis of Software Artifacts
Jonathan Aldrich, Instructor

Manuvir Das

Center for Software Excellence
Microsoft Corporation

Talking the talk ...

- Program analysis technology can make a huge impact on how software is engineered
- The trick is to properly balance research on new techniques with a focus on deployment
- The Center for Software Excellence (CSE) at Microsoft is doing this (well?) today

... walking the walk

- Program Analysis group in June 2005
 - 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

3

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Context

- The Nail (Windows)
 - Manual processes do not scale to “real” software
- The Hammer (Program Analysis)
 - Automated methods for “searching” programs
- The Carpenter (CSE)
 - A systematic, heavily automated, approach to improving the “quality” of software

4

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

What is program analysis?

- grep == program analysis
- program analysis == grep

- syntax trees, CFGs, instrumentation, alias analysis, dataflow analysis, dependency analysis, binary analysis, automated debugging, fault isolation, testing, symbolic evaluation, model checking, specifications, ...

5

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Roadmap

- (part of) The engineering process today
- (some of) The tools that enable the process
- (a few) Program analyses behind the tools
- (too many) Lessons learned along the way
- (too few) Suggestions for future research

6

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

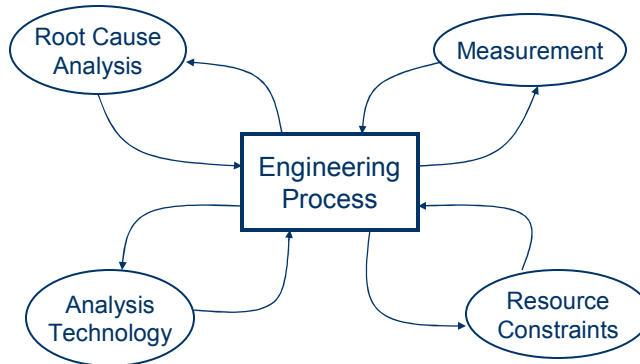
Engineering process

7

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Methodology



8

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Root cause analysis

- Understand important failures in a deep way
 - Every MSRC bulletin
 - Beta release feedback
 - Watson crash reports
 - Self host
 - Bug databases
- Design and adjust the engineering process to ensure that these failures are prevented

9

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Measurement

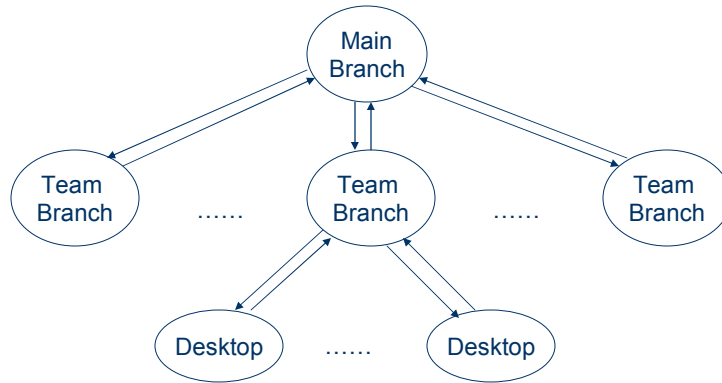
- Measure everything about the process
 - Code quality
 - Code velocity
 - Tools effectiveness
 - Developer productivity
- Tweak the process accordingly

10

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Process – Build Architecture

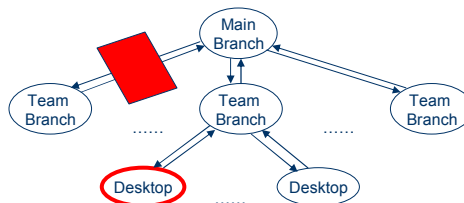


11

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Process – Quality Gates



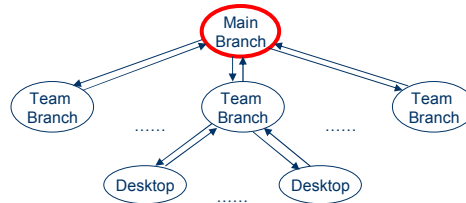
- Lightweight tools
 - run on developer desktop & team level branches
 - issues tracked within the program artifacts
- Enforced by rejection at gate

12

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Process – Automated Bug Filing



- Heavyweight tools
 - run on main branch
 - issues tracked through a central bug database
- Enforced by bug cap

13

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Tools

14

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

QG – Code Coverage via Testing

- Reject code that is not adequately tested
 - Maintain a minimum bar for code coverage
- Code coverage tool – Magellan
- Based on binary analysis - Vulcan

15

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Magellan

- BBCover
 - low overhead instrumentation & collection
 - down to basic block level
- Sleuth
 - coverage visualization, reporting & analysis
- Blender
 - coverage migration
- Scout
 - test prioritization

16

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

QG – Component Integrity

- Reject code that breaks the componentized architecture of the product
 - Control all dependencies across components
- Dependency analysis tool – MaX
- Based on binary analysis - Vulcan

17

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

MaX

- Constructs a graph of dependencies between binaries (DLLs) in the system
 - Obvious : call graph
 - Subtle : registry, RPC, ...
- Compare policy graph and actual graph
- Some discrepancies are treated as errors

18

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Vulcan

- Input – binary code
- Output – program abstractions

- Adapts to level of debug information
- Makes code instrumentation easy
 - think ATOM
- Makes code modification easy
 - link time, post link time, install time, run time

19

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

QG – Formal Specifications

- Reject code with poorly designed and/or insufficiently specified interfaces

- Lightweight specification language – SAL
 - initial focus on memory usage
- All functions must be SAL annotated
- Fully supported in Visual Studio (see MSDN)

20

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

SAL

- A language of contracts between functions
- preconditions
 - Statements that hold at entry to the callee
 - What does a callee expect from its callers?
- postconditions
 - Statements that hold at exit from the callee
 - What does a callee promise its callers?
- Usage example:
`a0 RT func(a1 ... an T par)`
- Buffer sizes, null pointers, memory usage, ...

21

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

SAL Example

- `wcsncpy`
 - precondition: destination must have enough allocated space

```
wchar_t wcsncpy (  
    wchar_t *dest, wchar_t *src, size_t num );
```

```
wchar_t wcsncpy (  
    __pre __writableTo(elementCount(num)) wchar_t *dest,  
    wchar_t *src, size_t num );
```

22

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

SAL Principle

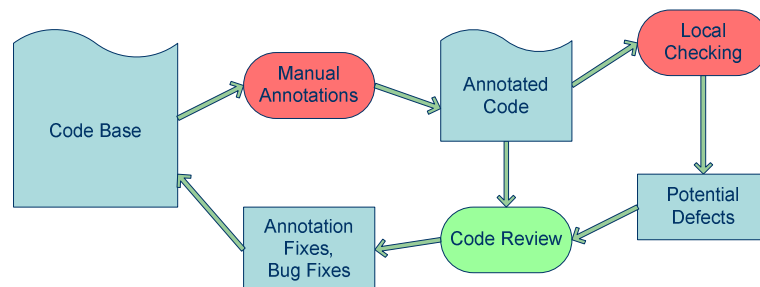
- Control the power of the specifications:
 - Impractical solution: Rewrite code in a different language that is amenable to automated analysis
 - Practical solution: Formalize invariants that are implicit in the code in intuitive notations
 - These invariants often already appear in comments

23

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Defect Detection Process – 1



24

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

QG – Integer Overflow

- Reject code with potential security holes due to unchecked integer arithmetic
- Range specifications + range checker – IO
- Local (intra-procedural) analysis
- Runs on developer desktop as part of regular compilation process

25

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

IO

- Enforces correct arithmetic for allocations

```
size1 = ...
size2 = ...
data = MyAlloc(size1+size2);
for (i = 0; i < size1; i++)
    data[i] = ...
```

- Construct an expression tree for every interesting expression in the code
- Ensure that every node in the tree is checked

26

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

QG – Buffer Overruns

- Reject code with potential security holes due to out of bounds buffer accesses
- Buffer size specifications + buffer overrun checker – espX
- Local (intra-procedural) analysis
- Runs on developer desktop as part of regular compilation process

27

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Bootstrap the process

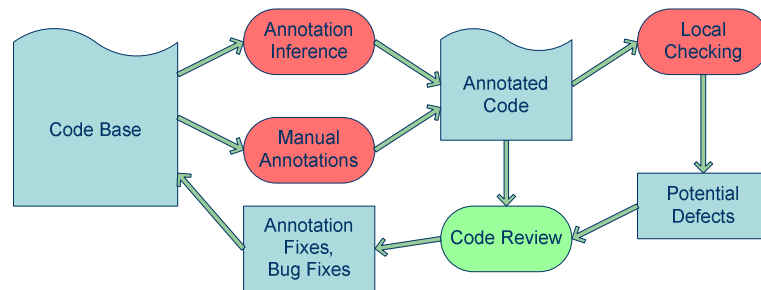
- Combine global and local analysis:
 - Weak global analysis to infer (potentially inaccurate) interface annotations - SALinfer
 - Strong local analysis to identify incorrect code and/or annotations - espX

28

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Defect Detection Process - 2



29

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

SALInfer

```
void work()
{
  int elements[200];
  wrap(elements, 200);
}

void wrap(pre elementCount(len) int *buf,
          int len)
{
  int *buf2 = buf;
  int len2 = len;
  zero(buf2, len2);
}

void zero(pre elementCount(len) int *buf,
          int len)
{
  int i;
  for(i = 0; i <= len; i++)
    buf[i] = 0;
}
```

Track flow of values through the code

1. Finds stack buffer
2. Adds annotation
3. Finds assignments
4. Adds annotation

30

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

espX

```
void work()
{
    int elements[200];
    wrap(elements, 200);
}

void wrap(pre elementCount(len) int *buf,
          int len)
{
    int *buf2 = buf;
    int len2 = len;
    zero(buf2, len2);
}

void zero(pre elementCount(len) int *buf,
          int len)
{
    int i;
    for(i = 0; i <= len; i++)
        buf[i] = 0;
}
```

Building and solving constraints

1. Builds constraints
2. Verifies contract
3. Builds constraints
len = length(buf); i ≤ len
4. Finds overrun
i < length(buf) ? NO!

31

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

QG – Code Correctness

- Reject code with potential crashes due to improper usage of memory
- Pointer usage specifications + memory usage checker – PREfast
- Managed code – PREsharp
- Local (intra-procedural) analysis
- Runs on developer desktop as part of regular compilation process

32

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

ABF – Code Correctness

- Tease out hard to find inter-component bugs that lead to crashes
 - null dereference, un-initialized memory, leaks, ...
 - difficult to find accurately on the desktop
- Inter-procedural symbolic evaluation - PREFIX

33

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

PREFIX

- Bottom-up process on the call graph
- Symbolic evaluation of a fixed number of distinct paths through each function
 - use symbolic state to remove infeasible paths
 - report defects
 - build function models for use by callers
- Solved all the difficult engineering problems for the inter-procedural tools to follow

34

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

ABF – Security

- For every new security issue, map it to a coding defect and root out all other instances
 - Each coding defect is a different pattern, but most can be viewed as finite state properties
- Heavyweight, thorough, property-based inter-procedural analysis - ESP

35

Analysis of Software Artifacts

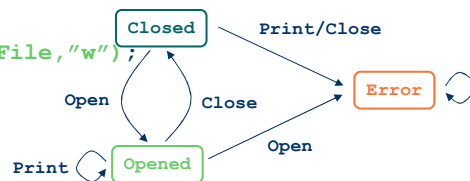
Manuvir Das, Microsoft Corporation

Property-based analysis

```
void main ()
{
  if (dump)
    Open fopen(dumpFile, "w");

  if (p)
    x = 0;
  else
    x = 1;

  if (dump)
    Close fclose(fil);
}
```



36

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

ESP

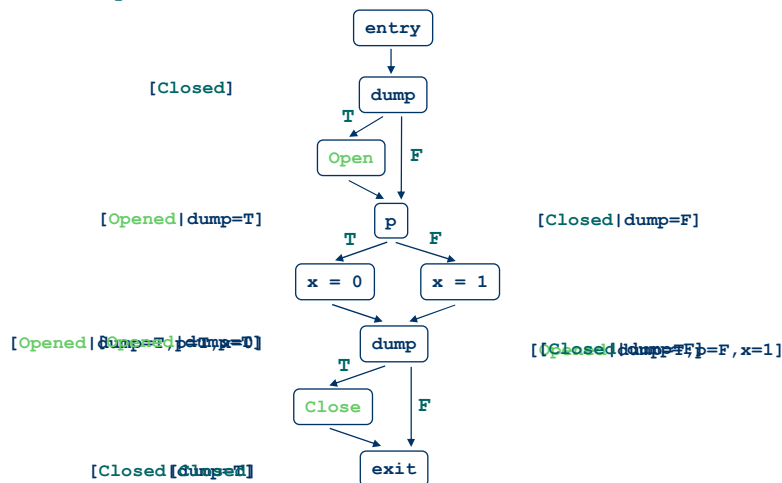
- Symbolically evaluate the program
 - track FSA state and execution state
- At control flow branch points:
 - Execution state implies branch direction?
 - Yes: process appropriate branch
 - No: split state and process both branches
- At control flow merge points:
 - States agree on property FSA state?
 - Yes: merge states
 - No: process states separately

37

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Example



Lessons

39

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Forcing functions for change

40

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

- 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

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

41

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

JA: edited to swap soundness/completeness,
for consistency with course usage

Myth 1 – Completeness matters

Complete == find only real bugs

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

42

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Myth 2 – Soundness matters

Sound == find all the bugs

- There will never be a sound analysis
 - Partial specifications
 - Missing code
- Developers want *consistent* analysis
 - Tools should be stable w.r.t. minor code changes
 - Systematic, thorough, tunable program analysis

43

Myth 3 – Developers only fix real bugs

- Key attributes of a “fixable” bug
 - Easy to fix
 - Easy to verify
 - Unlikely to introduce a regression
- Simple tools can be very effective

44

Myth 4 – Developers hate 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

45

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Myth 5 – Whiteboards are useful

- Whiteboards have held back defect detection
- The most useful analyses and tools mimic the thinking of the developer
 - e.g. do developers consider every possible interleaving when writing threaded code? No!

46

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Myth 6 – Theory is useless

- Fundamental ideas have been crucial
 - Hoare logic
 - Abstract interpretation
 - Context-sensitive analysis with summaries
 - Alias analysis

47

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Don't break the shipping code 😊

- `__invariant()` is an annotation macro
 - generates code in the tools build, noop in the real build
- 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!

48

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Summary

- Program analysis technology can make a huge impact on how software is developed
- The trick is to properly balance research on new techniques with a focus on deployment
- The Center for Software Excellence (CSE) at Microsoft is doing this (well?) today

49

Analysis of Software Artifacts

Manuvir Das, Microsoft Corporation

Microsoft[®]

<http://www.microsoft.com/cse>

<http://research.microsoft.com/manuvir>

© 2005 Microsoft Corporation. All rights reserved.
This presentation is for informational purposes only.
MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.