

# Static Code Analysis Procedures in the Development Cycle

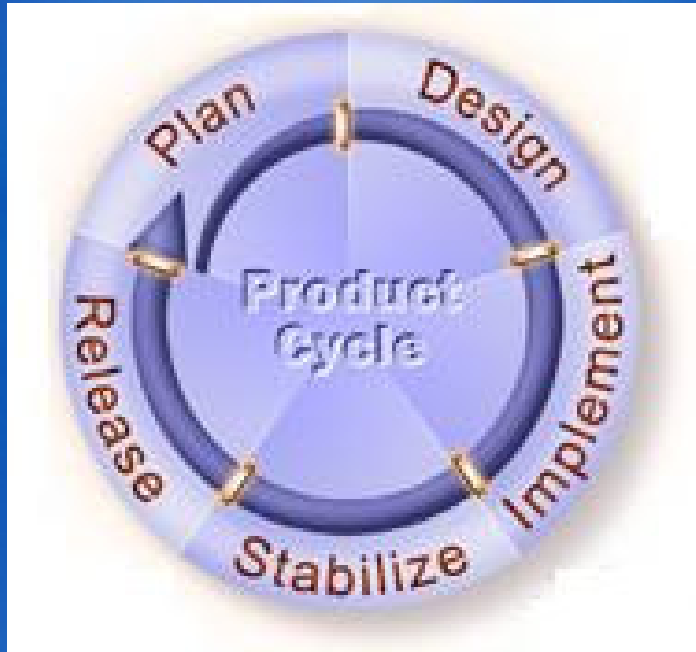
Tools, Technology, and Process in  
Engineering at Microsoft

Mooly Beerli  
Microsoft Haifa R&D Center

# Agenda

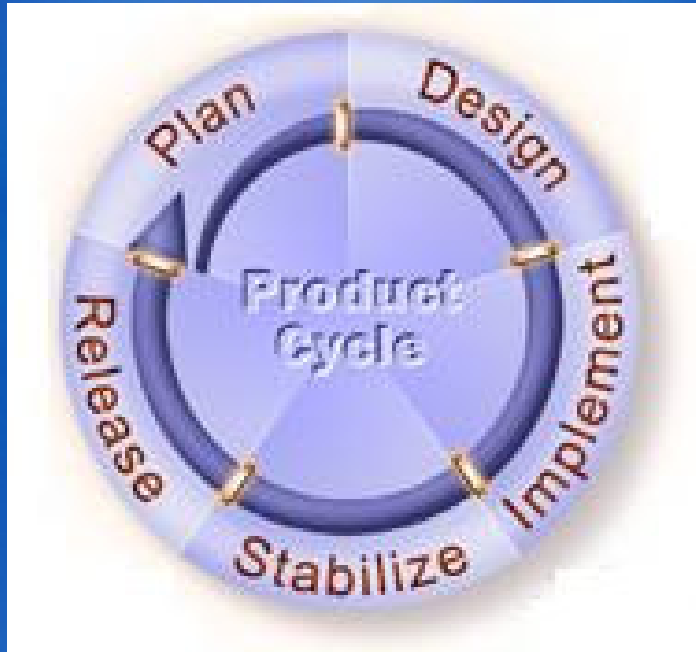
- Static code analysis tools
- PREfix and PREfast
- Integration into the development cycle
- Summary

# A Product's Life Cycle



- Cost of fixing bugs
  - Plan & Design - no code bugs (☺)
  - Implementation – Low cost, just fix the code and check in
  - Stabilize – Medium cost, track the bug, develop the 'right' test case, etc.
  - Release – High cost, reputation, release a hot fix (patch), documentation, publishing, etc.

# A Product's Life Cycle – cont.



- Types of tools
  - Build, Source control
  - Bug tracking (“RAID”)
  - Compilers, Linkers & Debuggers
  - Profiling & Optimization
  - Testing: Coverage, Fault injection, Test case generation, Prioritization, Capture & Replay, ...
  - Localization
  - Run-time checkers/verifiers
  - Static Analysis – this is our focus today.

# Static analysis tools

- Analyze code and detect potential defects
  - Advantages:
    - Not limited by test cases
    - Identify location of defect precisely (easy to fix)
    - Applicable early in the development cycle
    - Puts responsibility on developers
  - Issues
    - Up-front investment
    - Usability and noise
    - Scalability
    - Integration into environment



# Three common questions

- Do these tools find important defects?
  - Yes, definitely – including defects that would cause security bulletins, blue screens, ...
- Is every warning emitted by the tools useful?
  - No, definitely
  - We continue to focus on “noise”, but it won’t go away
- Do these tools find all the defects?
  - No, no, no!

# PRPrefix

- Implemented by MSR PPRC (Microsoft Research, Programmer Productivity Research Center)
- C/C++ defect detection via static analysis
- Powerful inter-procedural analysis
  - Incomplete
  - Useful in practice
- Typically run as part of a centralized build

# *Some Defects PRefix Finds*

## ● **Memory Management**

- Double free
- Freeing pointer to non-allocated memory (stack, global, etc.)
- Freeing pointer in middle of memory block

## ● **Initialization**

- Using uninitialized memory
- Freeing or dereferencing uninitialized pointer

## ● **Bounds violations**

- Overrun (reference beyond end)
- Underflow (reference before start of buffer)
- Failure to validate buffer size

## ● **Resource Leakage**

- Leaking Memory/Resource

## ● **Pointer Management**

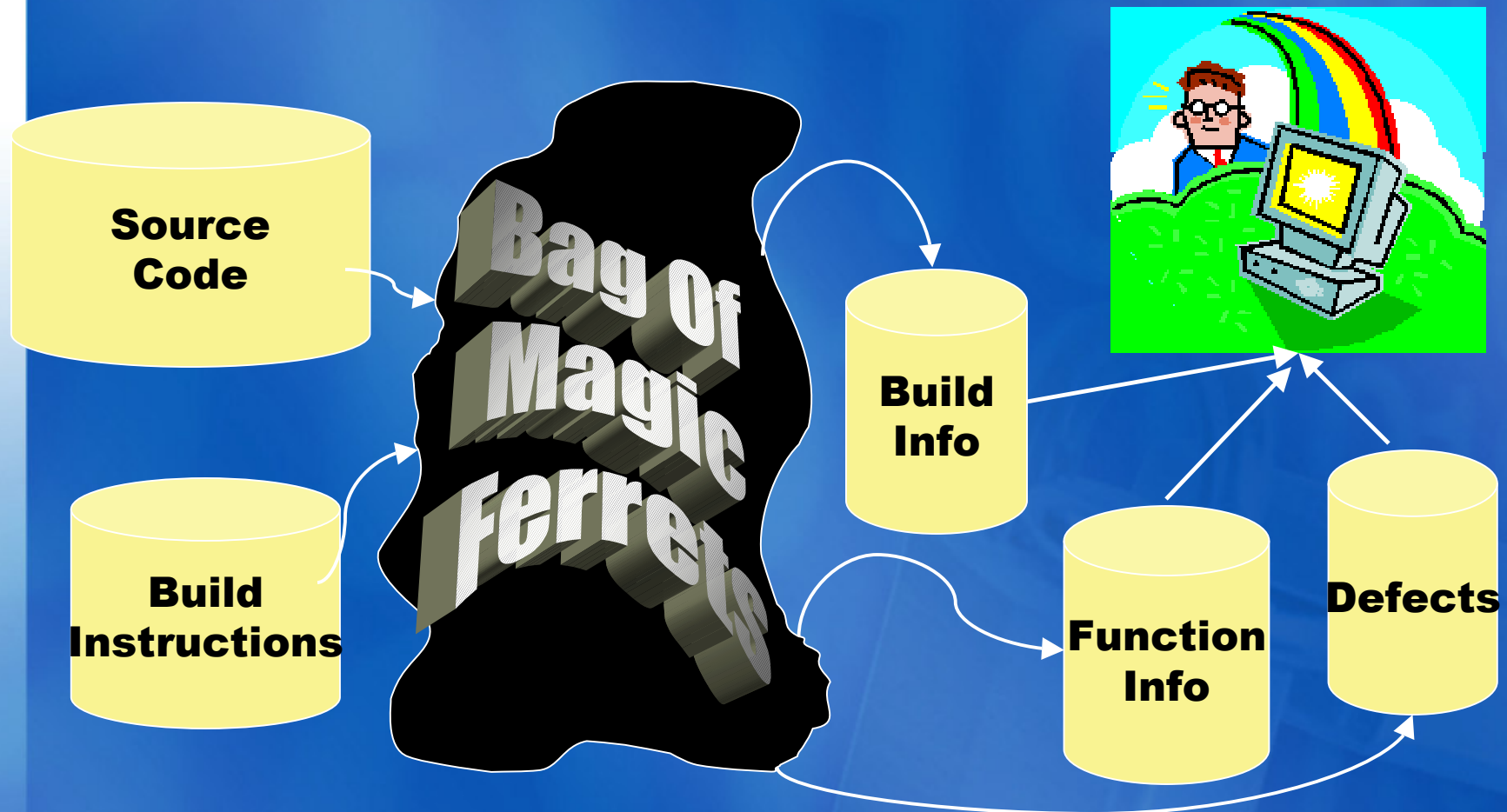
- Dereferencing NULL pointer
- Dereferencing invalid pointer
- Returns pointer to local
- Dereferencing or returning pointer to freed memory

## ● **Illegal State**

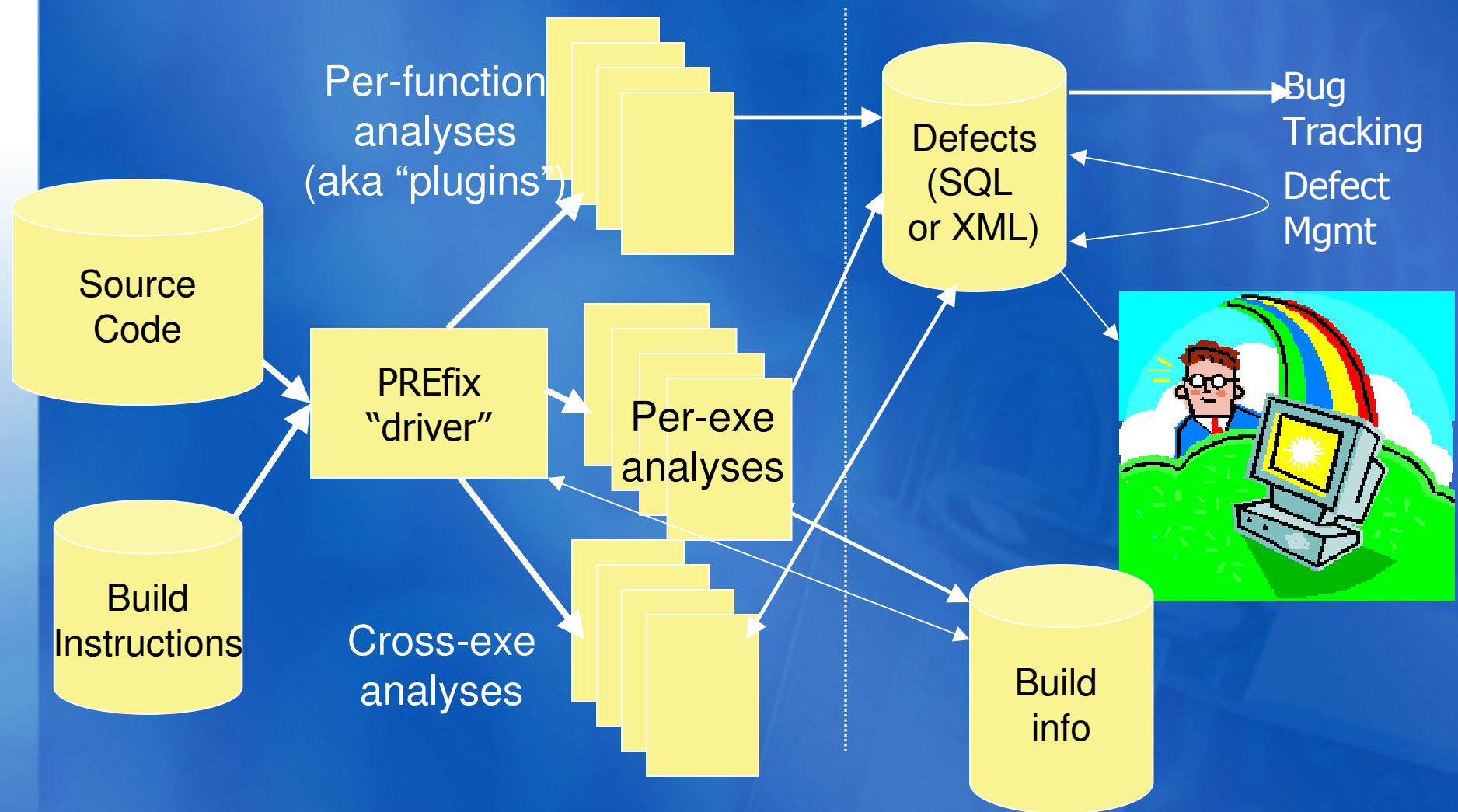
- Resource in illegal state
- Illegal value
- Divide by zero
- Writing to constant string



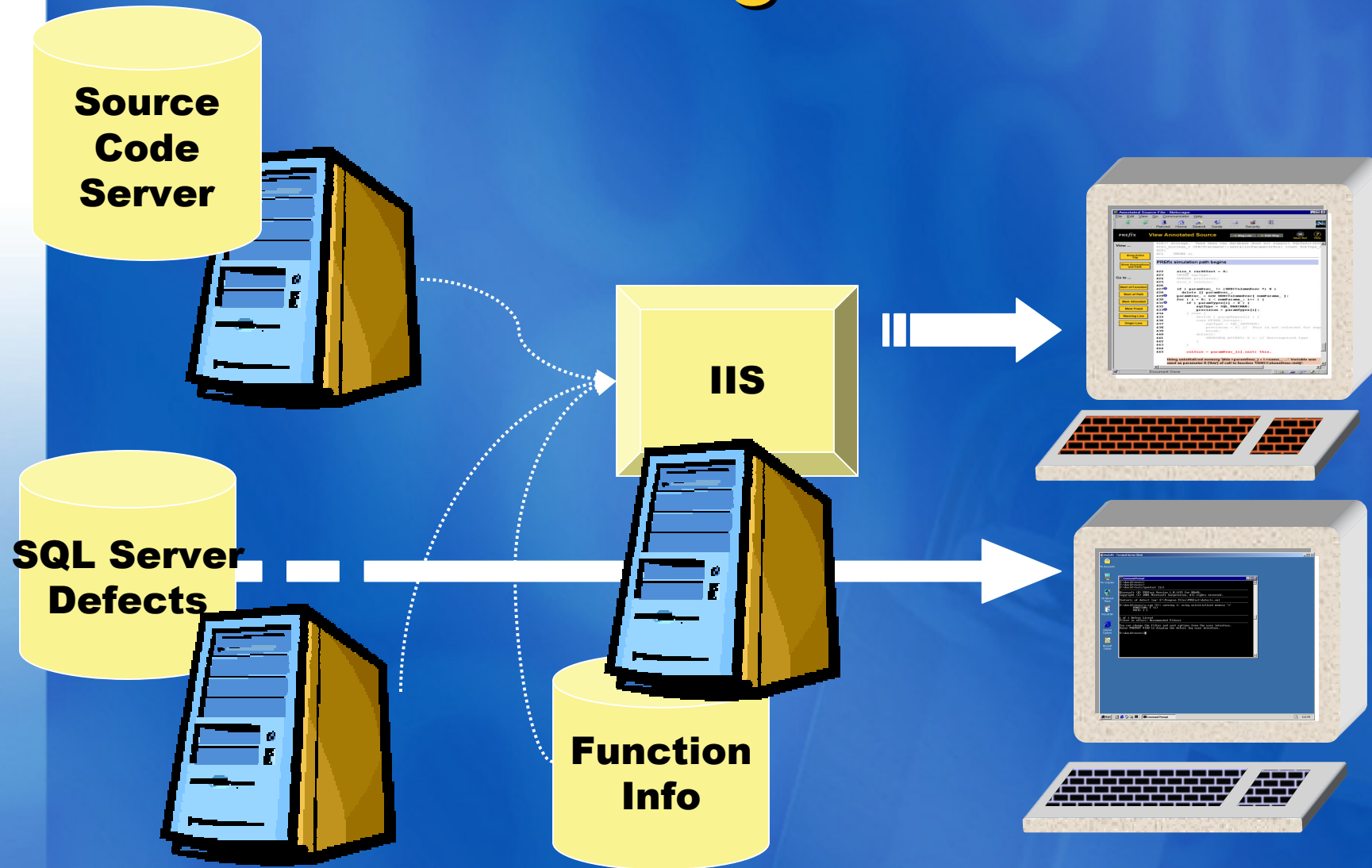
# High-level architecture



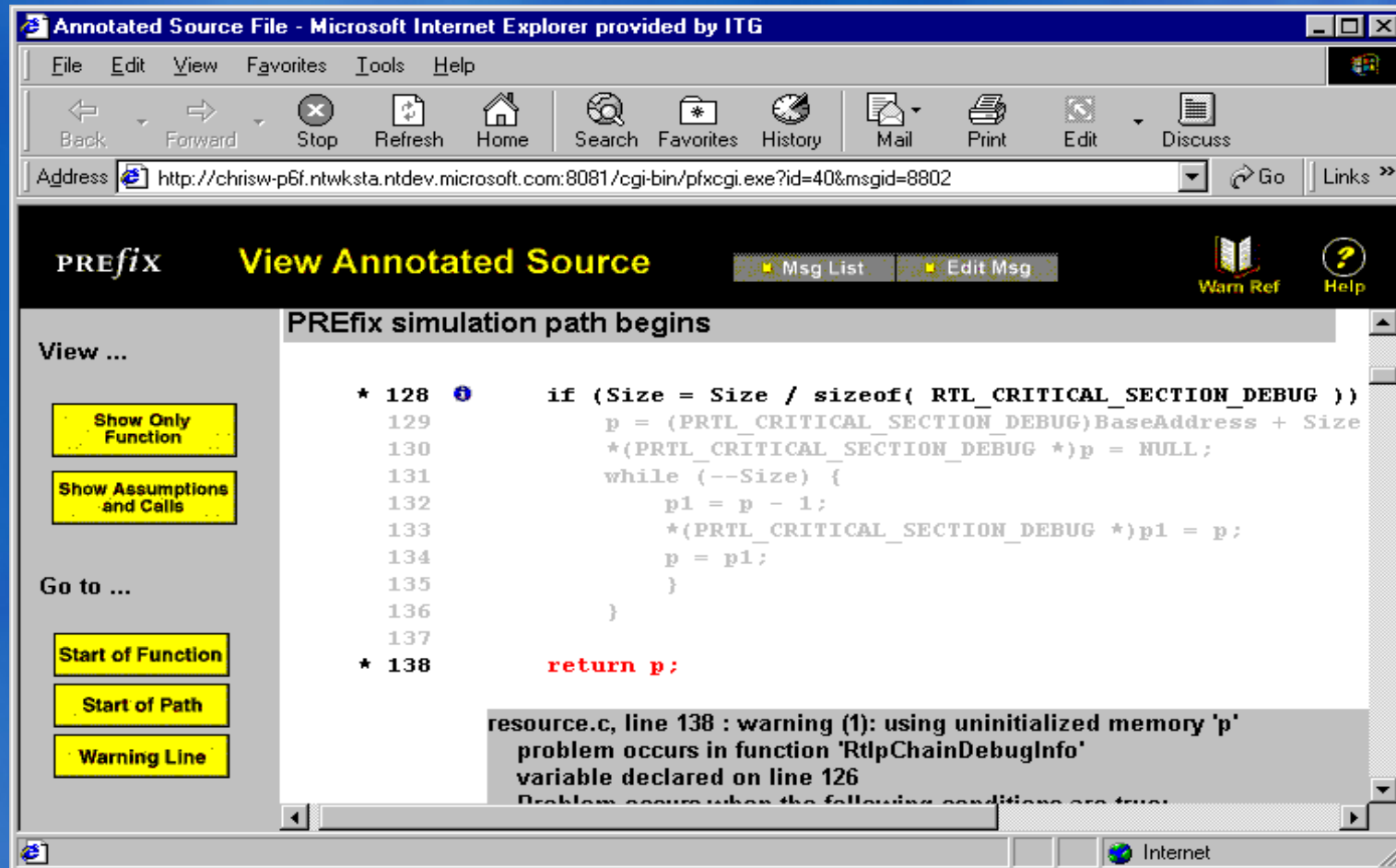
# PREfix Architecture



# PREfix: Viewing Results

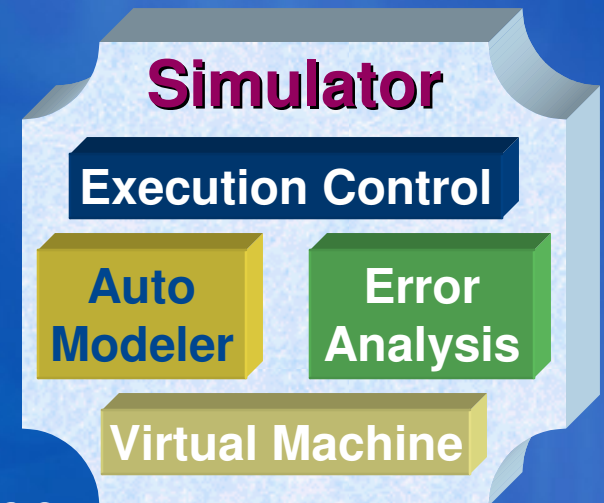


# HTML User Interface



# PREfix Simulator

- Execution control
  - Walks AST parse trees to follow various execution paths
- Virtual machine (VIM)
  - Tracks symbolic state of “virtual computer”
- Auto Modeler
  - Generates behavioral description (*model*) of each function from the virtual machine’s information
- Error analysis
  - Finds and reports defects based on state of VIM





# Analysis is not Complete

- Functions may have huge numbers of paths
- PREfix only explores N paths per function
  - User-configurable, default is 50
  - I.e., we give up on completeness
- Experiments indicate
  - Number of defects grows slowly with more paths
    - E.g., defects for 200 paths = 1.2 \* defects for 50 paths
    - defects for 1000 paths = 1.25 \* defects for 50 paths
  - Analysis time grows linearly with more paths
    - E.g., time for 1000 paths = 20 \* time for 50 paths

# Analysis is not Sound

- Approximations for performance, e.g.
  - Loops: traverse 0 or 1 time and then approximate
  - Recursion: explore cycles “until we’re bored”
- Can’t always find a model for a function call
  - E.g., Function pointers, Virtual functions, 3<sup>rd</sup>-party libraries
- Experiments indicate relatively few incorrect messages due to analysis inaccuracies

# Analysis works well in practice

- Finds enough real defects to be useful
- Noise is low enough that people use it
  - Not just an analysis issue; see below
- Scales well, so works on large code bases

# Sample defect PRefix message

```
void uwmsrsi4(LPCTSTR in) {  
    TCHAR buff[100];  
    _tcsncpy(buff, in, sizeof(buff));  
    /* ... */  
}
```

**TCHAR** is typedef'ed as either char or wchar\_t, depending on whether UNICODE is defined

**\_tcsncpy** expands to either strncpy or wcsncpy

# Sample defect PRefix

```
void uwmsrsi4(LPCTSTR in)
{
    TCHAR buff[100];
    _tcsncpy(buff, in, sizeof(buff));
}
```

**uwmsrsi4.c(10) : warning 51: using number of bytes instead of number of characters for 'buff' used as parameter 1 (dest) of call to 'wcsncpy' size of buffer 'buff' is 200 bytes reference is 399 bytes from start of buffer**

**uwmsrsi4.c(9) : stack variable declared here**

**problem occurs when the following condition is true:**

**uwmsrsi4.c(10) : when 'wcslen(in) >= 200' during call to 'wcsncpy' here**



# PREfast

- Lightweight, “desktop” defect detection
- Simple intra-procedural analyses
- Implemented by MSR PPRC + others
  - Windows devs involved in initial design, implementation
  - Office devs contributed significantly, including OACR environment
  - Extensibility allowed contributions from others
- Key goal: do less, but do it quickly
  - Allow developers to find bugs before check in
  - Extensibility led to very rapid enhancements
- Ties in with key challenges
  - Initial focus on security defects
  - Used as part of security bug bashes

# PREfast “defect description”

- An XML description of each defect, with
  - Brief description (mandatory; everything else is optional)
  - Additional details
  - Effect of the defect
  - Hypothesis about cause (phrased as question)
  - Severity
  - One or more examples (erroneous and corrected code)

# *Some Defects PREfast Finds*

- **Buffer Overrun**
  - Array bounds violations
- **HRESULT**
  - Abuses of the HRESULT type
- **Precedence**
  - Precedence mistakes
- **PREfix-Lite**
  - Uninitialized variables
  - NULL pointers
  - Leaks
- **Typos**
  - Syntax errors in your code

# Sample PRefast message

- ```
pFunc = (LPFN)GetProcAddress(hModule, "GetCredentials");
if (NULL == pFunc)
{
    rc = GetLastError();
    if (ERROR_PROC_NOT_FOUND == rc)
    {
        goto Exit;
    }
}
rc = (pFunc)(hServer, 0, (LPBYTE*)&pCred);
```

# Sample PRefast message

- ```
pFunc = (LPFN)GetProcAddress(hModule, "GetCredentials");  
if (NULL == pFunc)  
{  
    rc = GetLastError();  
    if (ERROR_PROC_NOT_FOUND == rc)  
    {  
        goto Exit;  
    }  
}  
rc = (pFunc)(hServer, 0, (LPBYTE*)&pCred);
```

exportrrasconfig.cpp(324) : warning 11: Dereferencing **NULL pointer 'pFunc'**.

problem occurs in function 'CheckServer'

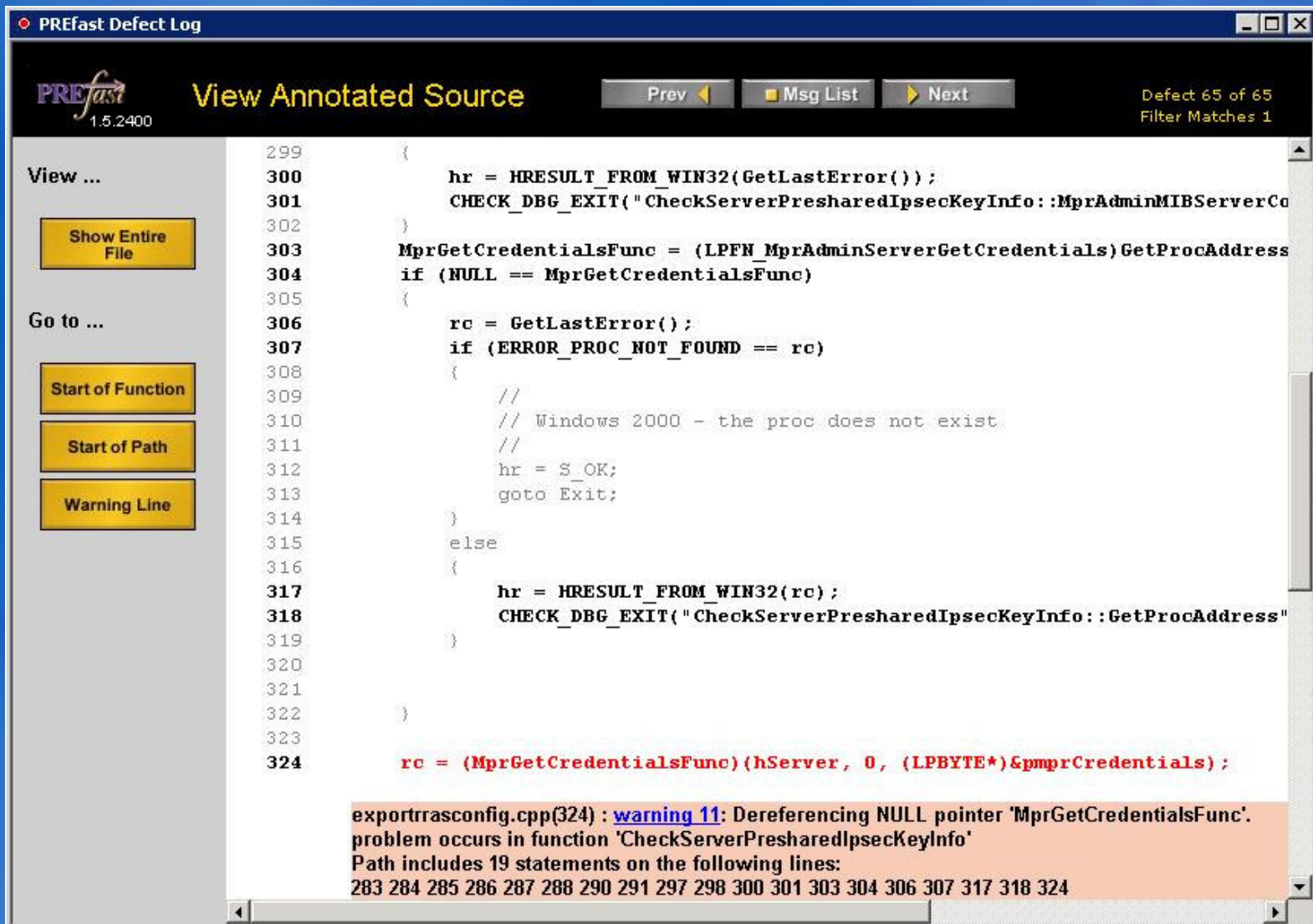
Path includes 19 statements on the following lines:

283 284 285 286 287 288 290 291 297

298 300 301 303 304 306 307 317 318 324



# Sample PREfast message



The screenshot shows the PREfast Defect Log window. The title bar reads "PREfast Defect Log". The window has a dark blue header with the PREfast logo (1.5.2400) on the left, a "View Annotated Source" button in the center, and "Prev", "Msg List", and "Next" buttons on the right. The top right corner displays "Defect 65 of 65" and "Filter Matches 1".

On the left side, there are two sections: "View ..." with a "Show Entire File" button, and "Go to ..." with buttons for "Start of Function", "Start of Path", and "Warning Line".

The main area displays annotated source code from `exportrasconfig.cpp`. The code is as follows:

```
299 {
300     hr = HRESULT_FROM_WIN32(GetLastError());
301     CHECK_DBG_EXIT("CheckServerPresharedIpsecKeyInfo::MprAdminMIBServerCo
302 }
303 MprGetCredentialsFunc = (LPFN_MprAdminServerGetCredentials)GetProcAddress
304 if (NULL == MprGetCredentialsFunc)
305 {
306     rc = GetLastError();
307     if (ERROR_PROC_NOT_FOUND == rc)
308     {
309         //
310         // Windows 2000 - the proc does not exist.
311         //
312         hr = S_OK;
313         goto Exit;
314     }
315     else
316     {
317         hr = HRESULT_FROM_WIN32(rc);
318         CHECK_DBG_EXIT("CheckServerPresharedIpsecKeyInfo::GetProcAddress"
319     }
320 }
321
322
323
324 rc = (MprGetCredentialsFunc)(hServer, 0, (LPBYTE*)&pmprCredentials);
```

At the bottom, a warning message is displayed in an orange box:

`exportrasconfig.cpp(324) : warning 11: Dereferencing NULL pointer 'MprGetCredentialsFunc'. problem occurs in function 'CheckServerPresharedIpsecKeyInfo'`  
Path includes 19 statements on the following lines:  
283 284 285 286 287 288 290 291 297 298 300 301 303 304 306 307 317 318 324

# Noise

- Noise = “messages people don’t care about”
  - (not just “bogus” messages)
- Usually, noise is worse than missing a defect

Too much noise

=> people won’t use the tool

== missing *all* the defects



# Message Prioritization

- Which messages correspond to defects that will actually be fixed?
- “Rank”: a synthetic metric of a message’s “goodness”
  - Better-ranking messages are more likely to identify defects that will actually get fixed
- Multiple dimensions:
  - Severity of consequences
  - Likelihood that message is correct
  - Comprehensibility of message
  - ...

# Noise and history

- Noise naturally increases over time
  - People fix the real defects
- A history mechanism avoids these problems
  - Distinguish newly-occurring messages
  - Goal: avoid re-examining noise messages

# Compare and contrast ...

	Use Model	Analysis	Kinds of defects	“Instant results”
PREfix	Central Build	Cross-function, detailed	~ 40	
PREfast	Desktop	Single-function, superficial	> 100	



# Sample usage: Windows organization

- PREfix: centralized runs
  - Defects filed automatically
  - Roughly monthly from 1/2000-present
  - 30 MLOC – 6 days to complete a run
  - Some teams also run PREfix on their own
- PREfast: run by individual devs/testers
  - Fix before check in
  - Or run against checked-in code

# Summary

- Detecting defects *earlier* in the cycle
  - PREfix: after code is checked in
    - As opposed to during testing or post-release
  - PREfast: before code is checked in
- Static analysis is becoming pervasive
  - PREfix, PREfast's initial successes mean this is no longer a "research" technology
  - Overcoming "noise" is vital
- Technology is encouraging process change

# Questions?



