Software Development at Microsoft

T.K. Backman
tkback@microsoft.com

Principal Development Lead
Debugging and Tools Group
Windows Engineering Desktop
Microsoft Corporation

Jason Yang
jasony@microsoft.com

Principal Development Lead
Analysis Technologies Team
Windows Engineering Desktop
Microsoft Corporation

The Real World Challenge

Code on a massive scale

Developers on a massive scale

Tight constraints on schedules

What We'll Talk About Today

- Company structure
  - Why the world is not just about developers ☺
- Innovation strategy
  - How we actually improve software over time
- Dynamic tension
  - When people are involved, everything changes
- Development cycles
  - How we build software products in cycles
- Program analysis
  - How we push quality upstream
- Windows engineering system
  - How we build large-scale products

Core Disciplines @ Microsoft

- Total size: ~89,000 employees
- Windows & Office – "perfect org structure"
  - PM – program managers
  - Dev – software developers
  - Test – software developers in test
- Around 1000 PM+Dev+Test feature teams on 100s of products

Windows Division

- Team size: ~10,000 employees
- Sales & marketing
- Project managers / product managers
- 30 feature teams
  - 1500 Devs
  - 1500 Testers
  - 1000 PMs
- Customer support engineers
- Build engineers

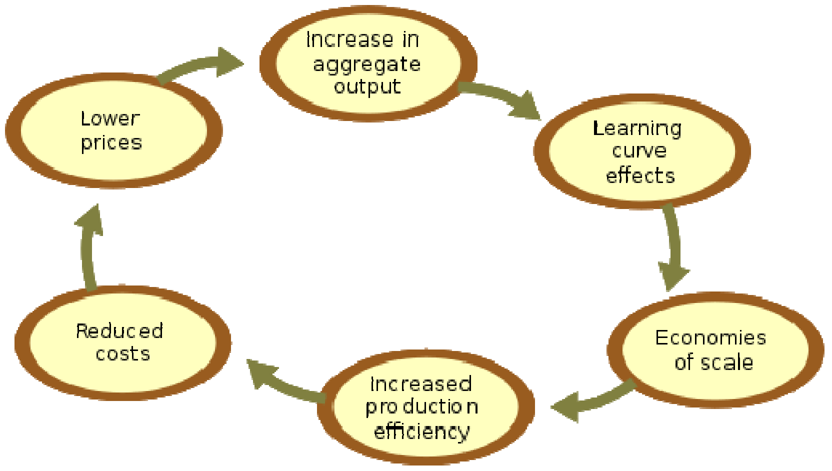
Software– Art or Science?

“I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of *Science*, whatever the matter may be."

– *Lord Kelvin, 1883*

Virtuous Feedback Loops

- “A complex of events that reinforces itself through a feedback loop”



- Once you have measurability and virtuous feedback, you get incremental improvements
- Examples
  - SQM data, usability testing, Windows Error Reporting, static analysis, code coverage, test reports, annual reviews, product reviews

Product Design

- Identify customers and their requirements / problems / values
- Describe compelling **visions** for the product
- Establish **tenets** that act as product themes to support the visions
- Describe the **scenarios** that enable tenets
- Create **features** that embody the solutions
- Iterate features based on virtuous feedback

Dynamic Tension

"The actual process is fluid and evolving…"

- Thought leader: Dev / Test / PM
- Version focus: features vs. bugs
- Design agility: waterfall vs. scrum
- Capacity allocation: design/coding/debugging
- Open source: Cathedral vs. Bazaar

Thought Leaders

- Which form of leadership?
- All teams are organized / led differently
  - PM driven – best for end user visible shipping features / products
  - Dev driven – best for research / highly technical projects
  - Test driven – best for sustaining engineering
- Teams tend to evolve as the products / features mature

Version Focus

- How innovative should we be this time?
  - Focus on features
    - Usually results in new value but weak quality
  - Focus on bugs
    - Usually results in great quality but not interesting
  - Reaching a balance
    - Your customers will tell you which they want

Design Agility

- Scale of feature iteration?
  - Waterfall model
    - Planning occurs upfront years in advance and is often way off base by the end of the project
  - Scrum model
    - Planning occurs every 6 weeks and everything is delivered in small, short sprints with immediate feedback
    - May only work well for smaller features/products
  - Hybrid solutions
    - Planning occurs every 5 months and after each milestone customer feedback is received when major components are completed and integrated

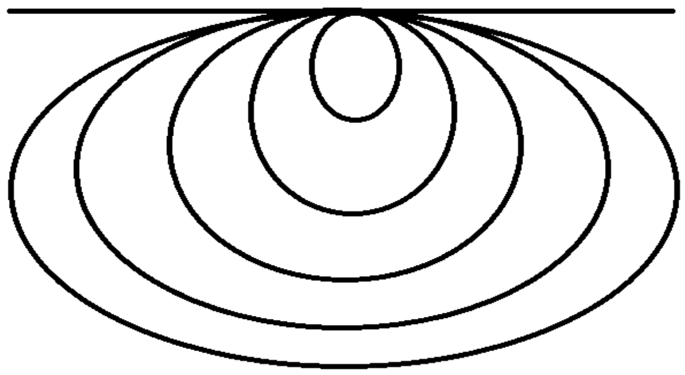
Capacity Allocation

- Where do you spend your time?
  - Design – OOD, factoring, architecture, algorithms
  - Coding – producing source, writing unit tests, TDD
  - Debugging – debuggers, running tests, fixing bugs
- Some typical allocations
  - OOD: 60% design, 20% coding, 20% debugging
  - Classic: 40% design, 20% coding, 40% debugging
  - Agile: 20% design, 30% coding, 50% debugging
- Most sub-teams will vary their approach

Open Source

- Who controls the code?
  - Cathedral – High priest owns the scripture
    - This is the classic one person owns each binary approach used industry-wide by many companies
  - Bazaar – everyone can join in
    - This is the approach used by most non-profit organizations where any can contribute
  - Public vs. private variants
    - It's possible to do "open source" inside a company where it's still private, but jointly developed by all

Concentric Feedback Loops



1. Product cycle – years per release
2. Outer loop – months per milestone
3. Middle loop – days per feature
4. Inner loop – minutes per compilation

Product Cycle

- Years/Release
  - Tools
    - Project – schedule charts for tracking progress
    - Excel spreadsheets – for feature value analysis
    - Internal websites – for document management
    - SQM product data – for customer usage data
    - Customer feedback – qualitative & quantitative data
  - Roles
    - Sales, marketing, Dev/Test/PM, doc, support
  - Deliverables
    - Requirements/pillars/tenets, Beta/RC/RTM bits, packaging, docs/kits, sales/marketing campaigns

Outer Loop

- Months/Milestone
  - Tools
    - Team Foundation Server (TFS) – feature tracking
    - Automated testing – functional tests
    - UX usability testing – live customer tests
    - Product Studio – bug database
  - Roles
    - PM/Test/(Dev)
  - Deliverables
    - Product features, product metrics, quality reports

Middle Loop

- Days/feature
  - Tools
    - Product Studio
    - Unit testing
    - Email discussion
    - Architecture/design/test documents
  - Roles
    - Dev/Test/(PM)
  - Deliverables
    - Bug fixes, code reviews, binaries, test runs

Inner Loop

- Minutes/Run
  - Tools
    - Source Depot – manage code versions
    - Visual Studio – compile/link/run
    - Static analysis – verify written code
    - Unit tests – verify basic functionality
  - Roles
    - Dev/(Test)
  - Deliverables
    - Running code, working tests

Windows Development Toolset

- Visual Studio – write, edit, compile, debug source code
- Team Foundation Server – track product features & tasks
- Source Depot – code changes and source branches
- Product Studio – defect reporting database
- Static analysis – detect code defects at compile time
- TAEF – software unit test framework
- Code coverage – verify completeness of testing
- Application Verifier, Driver Verifier – detect API misuse
- Scalable code search – Windows: 5K binaries,1M functions,100M lines
- Build machines – daily builds on hundreds of source branches
- …

What I Wish Someone Would Have Told Me

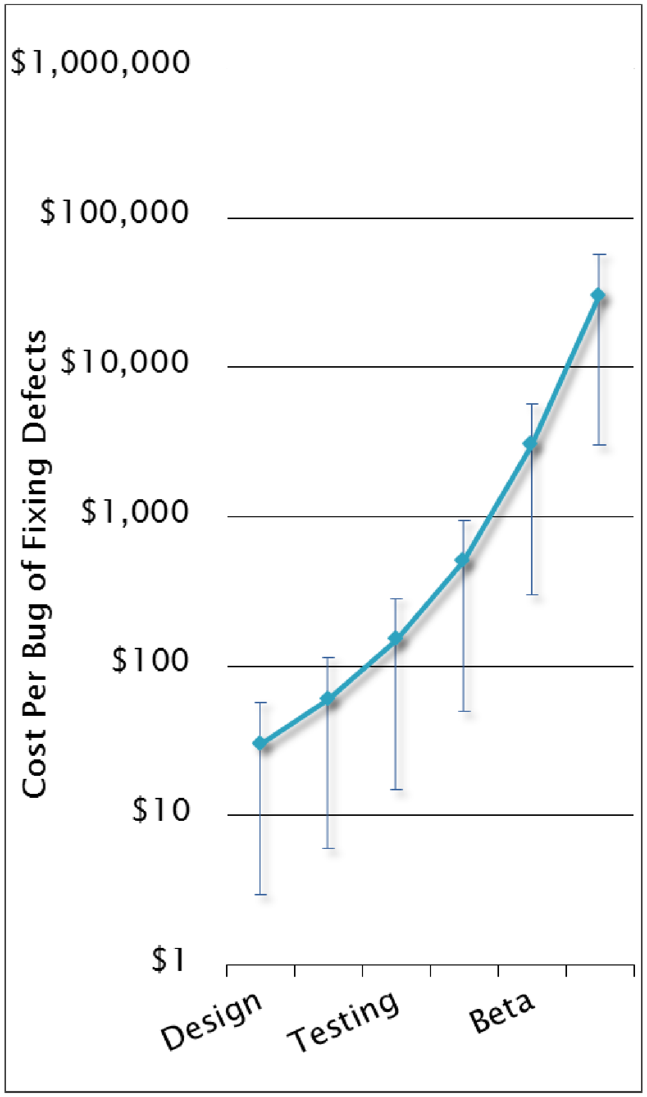
- Actual productive development hours in an 8 hour day are **very, very few**; don't be surprised at the overtime
- You need to learn **20% new technology** per year just to stay even with the rate of change
- Software engineers are always **too optimistic about schedules,** particularly new ones; double or triple your estimates
- Devs stay at a small to medium software company with an **average of 24-30 months**; you will be moving around a lot
- **Revenue per employee** is crucial: <$200K doom; $200k-300k OK; >$300K great
- Be sure you pick a product & company you care deeply about

Good design + analysis tools + sound engineering process

⬇

Significantly fewer code defects

Push Quality Upstream Matters



Microsoft Source Code Annotation Language (SAL)

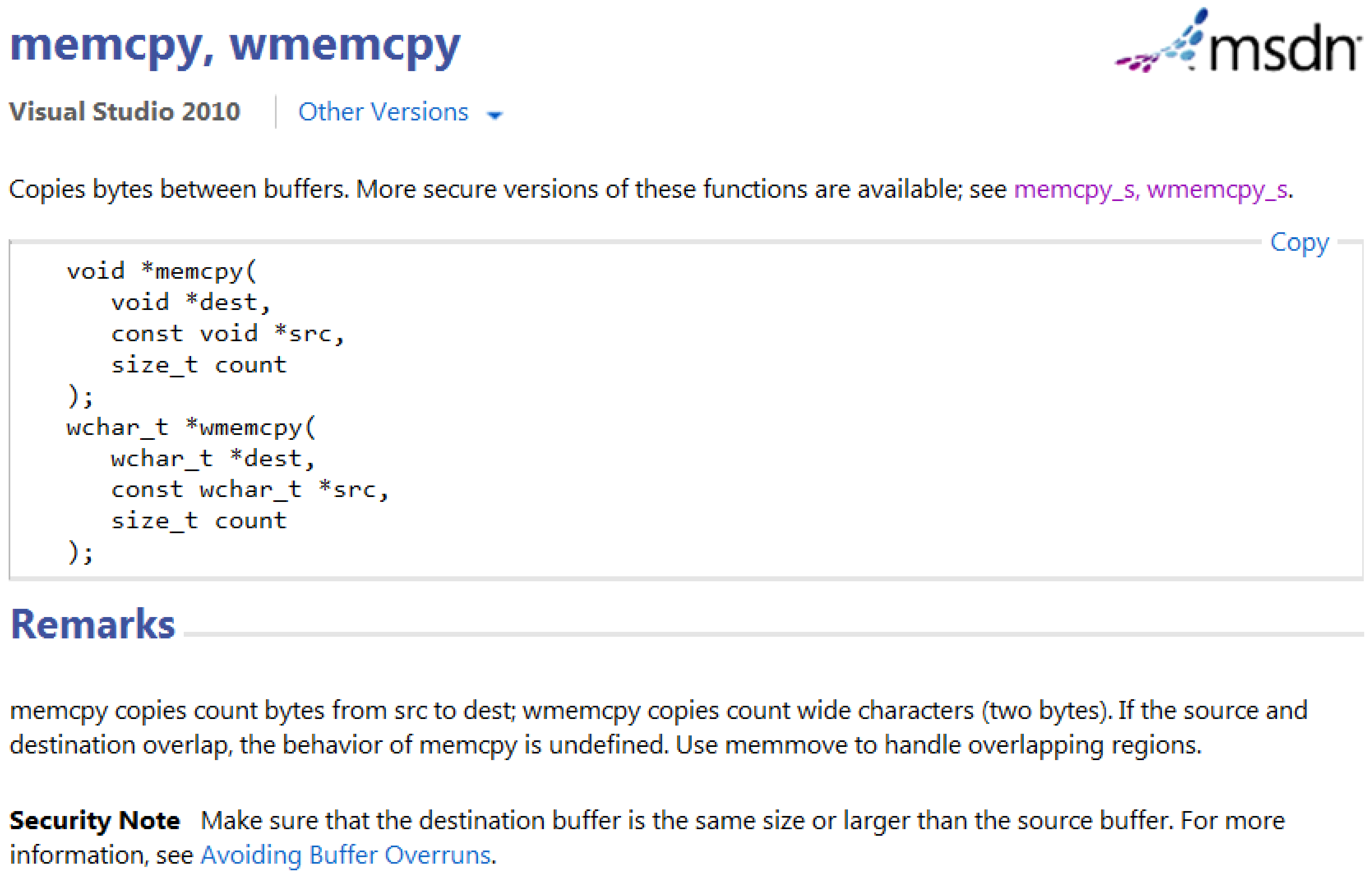
3,631,361 *

* number of annotations in Windows alone

more secure and reliable products

What do These Functions Do?

```
void * memcpy(
    void *dest,
    const void *src,
    size_t count
);

wchar_t *wmemcpy(
    wchar_t *dest,
    const wchar_t *src,
    size_t count
);
```



memcpy, wmemcpy

Visual Studio 2010

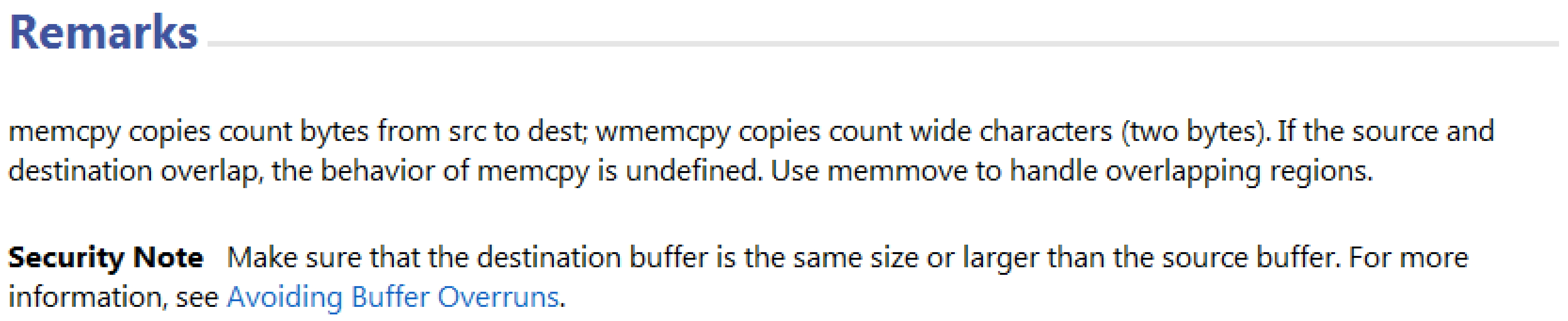
Copies bytes between buffers. More secure versions of these functions are available; see memcpy_s, wmemcpy_s.

```
void *memcpy(
   void *dest,
   const void *src,
   size_t count
);
wchar_t *wmemcpy(
   wchar_t *dest,
   const wchar_t *src,
   size_t count
);
```

Remarks

memcpy copies count bytes from src to dest; wmemcpy copies count wide characters (two bytes). If the source and destination overlap, the behavior of memcpy is undefined. Use memmove to handle overlapping regions.

Security Note Make sure that the destination buffer is the same size or larger than the source buffer. For more information, see Avoiding Buffer Overruns.

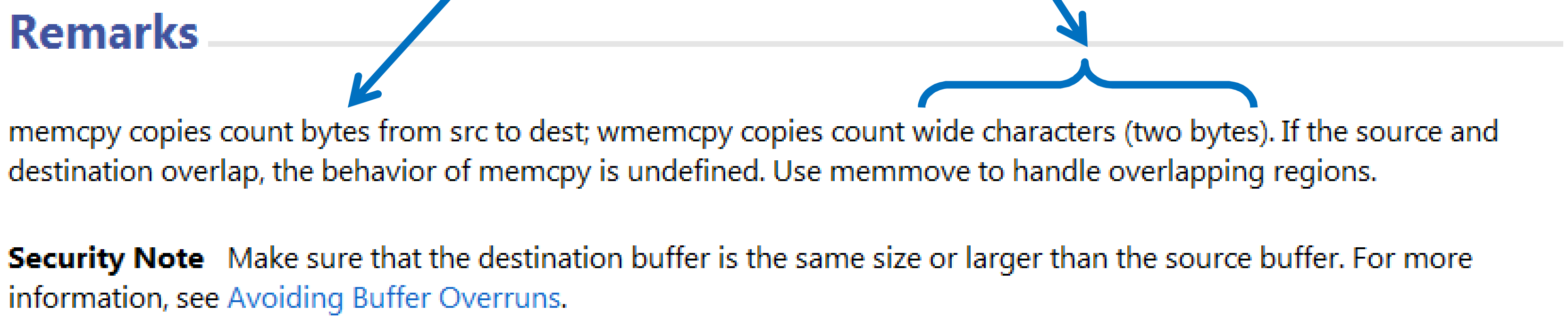


Remarks

memcpy copies count bytes from src to dest; wmemcpy copies count wide characters (two bytes). If the source and destination overlap, the behavior of memcpy is undefined. Use memmove to handle overlapping regions.

Security Note Make sure that the destination buffer is the same size or larger than the source buffer. For more information, see Avoiding Buffer Overruns.

For every buffer API there's usually a wide version.
Many errors are confusing "byte" vs. "element" counts.

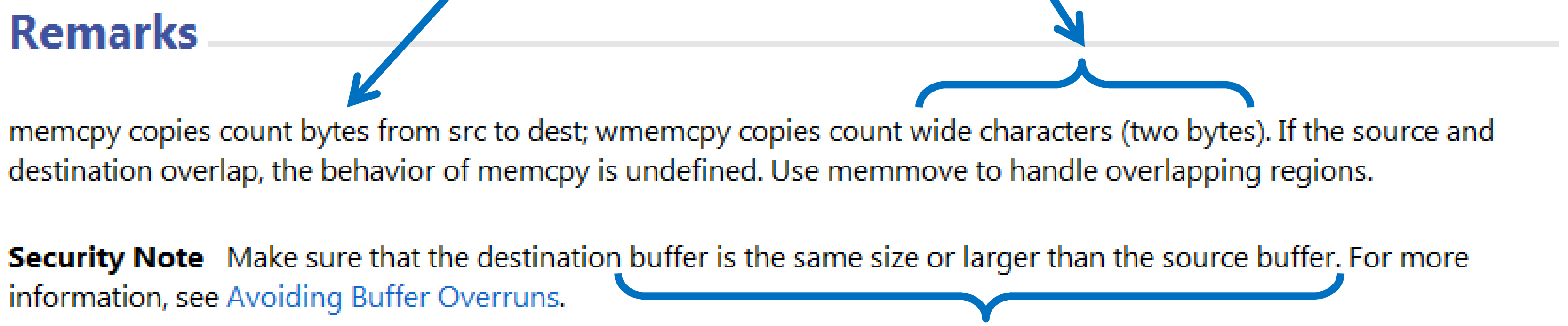


Remarks

memcpy copies count bytes from src to dest; wmemcpy copies count wide characters (two bytes). If the source and destination overlap, the behavior of memcpy is undefined. Use memmove to handle overlapping regions.

Security Note Make sure that the destination buffer is the same size or larger than the source buffer. For more information, see Avoiding Buffer Overruns.

For every buffer API there’s usually a wide version.
Many errors are confusing “byte” vs. “element” counts.



Remarks

memcpy copies count bytes from src to dest; wmemcpy copies count wide characters (two bytes). If the source and destination overlap, the behavior of memcpy is undefined. Use memmove to handle overlapping regions.

Security Note Make sure that the destination buffer is the same size or larger than the source buffer. For more information, see Avoiding Buffer Overruns.

Vital property for avoiding buffer overrun.

SAL Speak

```
void * memcpy(
    _Out_writes_bytes_all_(count) void *dest,
    _In_reads_bytes_(count) const void *src,
    size_t count
);

wchar_t *wmemcpy(
    _Out_writes_all_(count) wchar_t *dest,
    _In_reads_(count) const wchar_t *src,
    size_t count
);
```

- ✓ Captures programmer intent
- ✓ Improves defect detection via tools
- ✓ Extends language types to encode program logic properties

Precondition: function can assume **p** to be non-null when called

```
_Post_ _Notnull_ void * foo(_Pre_ _Notnull_ int *p);
```

Postcondition: function must ensure the return value to be non-null

```
struct buf {
    int n;
    _Field_size_(n) int *data;
};
```

Invariant: property that should be maintained

Automated Program Analysis Tools

Code Correctness
Static tools – PREfix, PREfast, Esp

Detects buffer overrun, null pointer, uninitialized memory, leak, banned API, race condition, deadlock, …

Code Coverage
Code coverage tool – Magellan (based on Vulcan)

Detects code that is not adequately tested

Architecture Layering
Dependency analysis tool – MaX (based on Vulcan)

Detects code that breaks the componentized architecture of product

Accuracy

False positive:
report is not a bug.

vs.

Completeness

False negative:
bug is not reported.

don’t miss any bug + report only real bugs == mission impossible

We need to deal with partial programs and partial specifications.

Any of the inputs could trigger a bug in the program.

- ➢ No false negative—we have to try all of the inputs.
  If we do the inputs in bunches, we’ll have noise.
- ➢ No false positive—we have to try the inputs one by one.
  But the domain of program inputs is infinite.

Dynamic Analysis

Run the program.

Observe program behavior on a single run.

Apply rules to identify deviant behavior.

Example: Application Verifier

vs.

Static Analysis

Simulate many possible runes of the program.

Observe program behavior on a collection of runs.

Apply rules to identify deviant behavior.

Example: PREfast

Local Analysis

Single-function analysis (e.g., PREfast)

Scales well enough to fit in compilers.

Example: unused local variable

```
void foo(int *q) {
    int *r = q;
    *q = 0;
}
```

vs.

Global Analysis

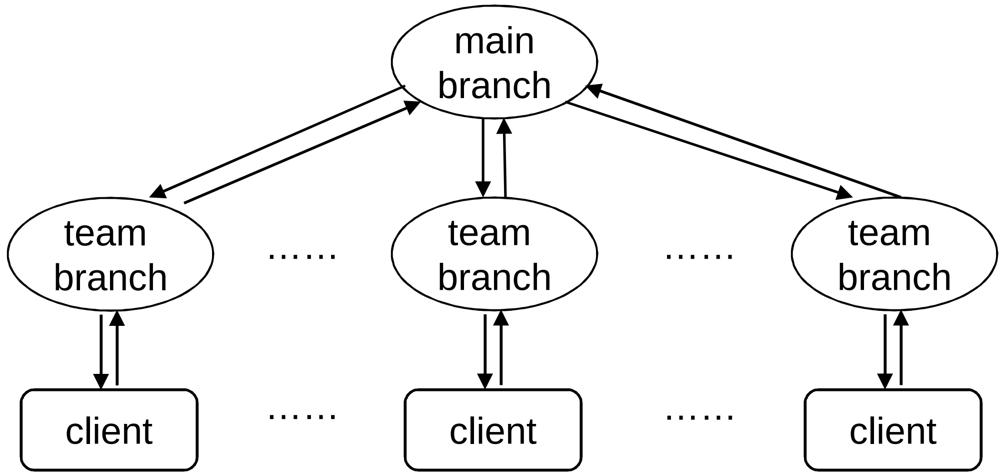
Cross-function analysis (e.g., PREfix)

Can find deeper bugs.

Example: null dereference due to broken contract

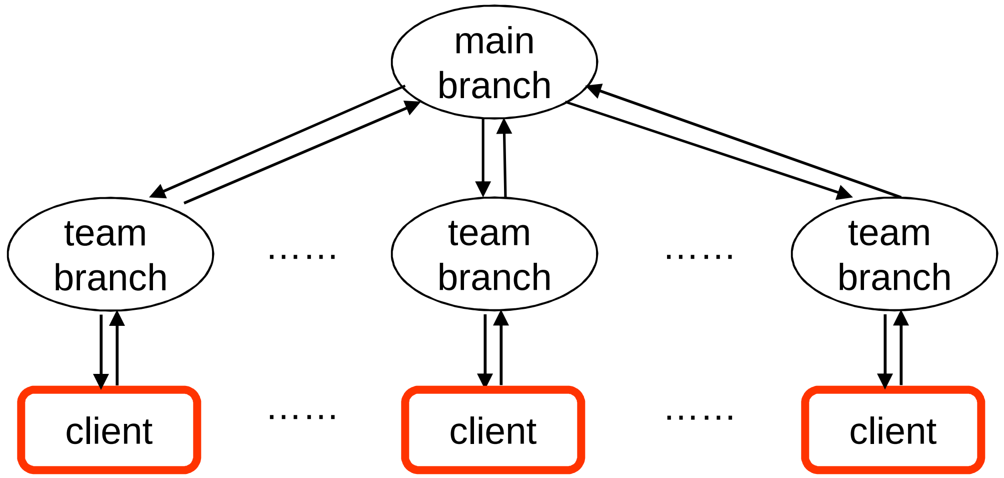
```
void bar(int *q) {
    q = NULL;
    foo(q);
}

void foo(int *p) {
    *p = 1;
}
```

Windows Build Architecture



Forward Integration (FI): code flows from parent to child branch
Reverse Integration (RI): code flows from child to parent branch

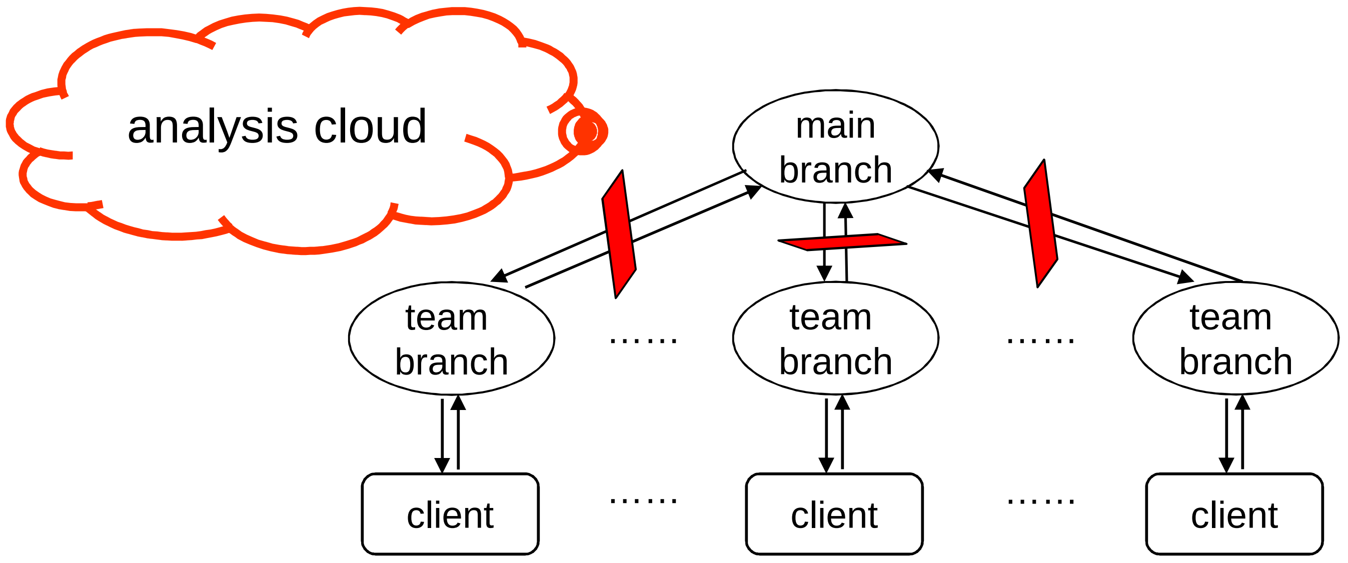
Local Analysis on Developer Desktop



Microsoft Auto Code Review (OACR)

- runs in the background
- intercepts the build commands
- launches light-weight tools like PREfast plugins

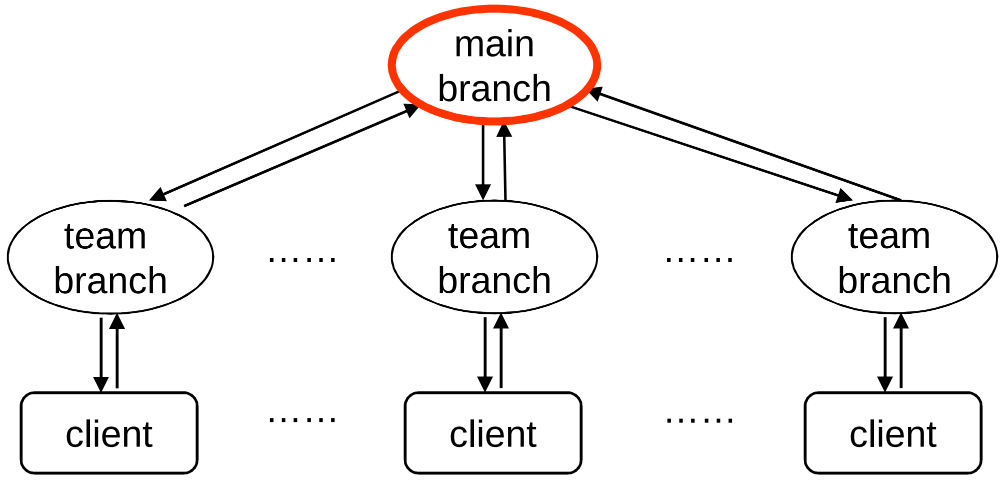
Quality Gates



Quality Gates (static analysis “minimum bar”)

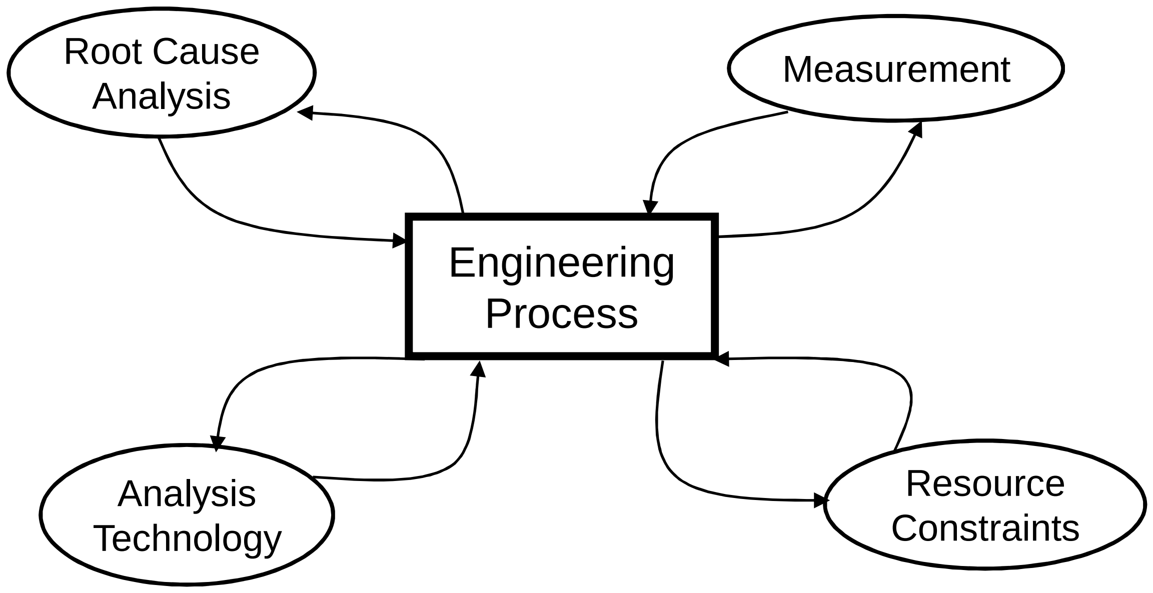
- Enforced by rejection at gate
- Bugs found in quality gates block reverse integration (RI)

Global Analysis via Central Runs



Heavy-weight tools like PREfix run on main branch

Methodology



Understand important failures in a deep way

Measure everything about the process

Use feedback to improve the engineering process

Bottom Line Results

- From Microsoft annual report
  - Years in business – since 1975
  - Annual revenue – $62.484 B
  - Profit margins – 30.84%
  - Balance sheet – $39.98 B
  - Revenue/employee: $700K

Questions?