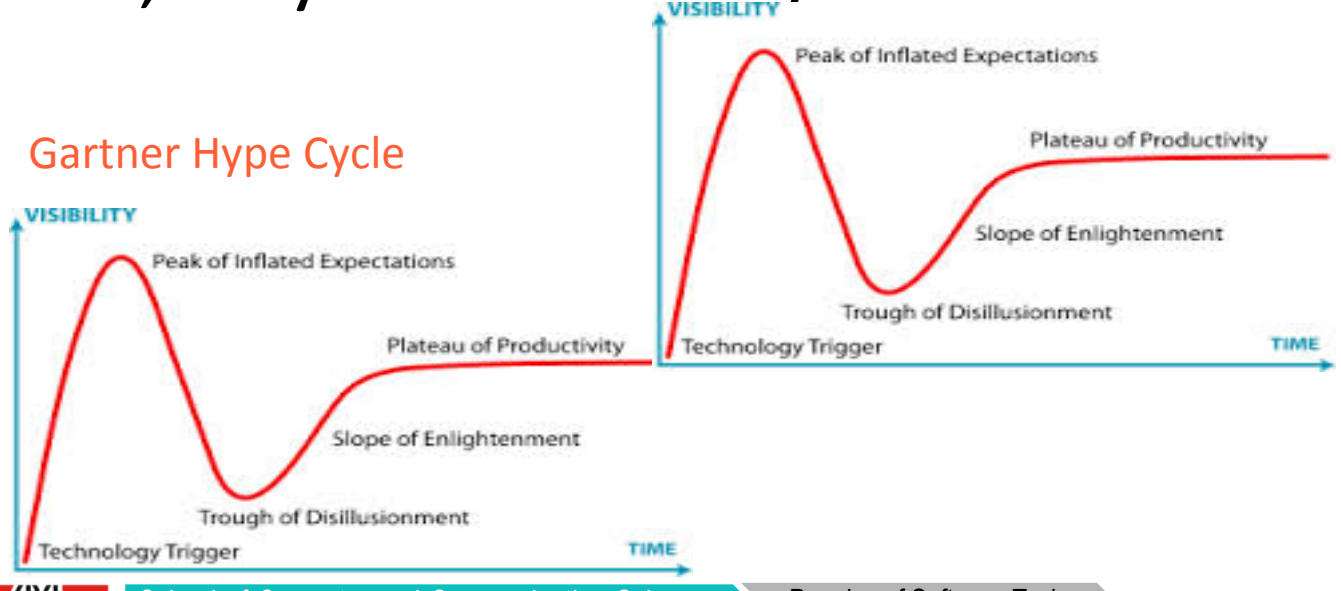
Disclaimer

- My personal opinion
  - Not Microsoft's
  - Not necessarily yours
- Based on my memory and public presentations
  - Left my Microsoft email at Microsoft

Outline

- The early days: software verification
- The revival: software defect detection
- So, why doesn't everyone do it?

Gartner Hype Cycle



1960-70's Research

- Program verification
  - Prove program correct
  - Apply mathematical reasoning to software
- Many interesting and useful results and insights into program semantics and formal methods
- Great skepticism among practitioners and some researchers
  - DeMillo, Lipton, Perlis, *Social processes and proofs of theorems and programs*

1980-90's

- Verification winter
  - Failed to achieve goal of verifying software
  - Few promising avenues of research
  - Fickle funding followed other fads (FFFoF)
  - A few persistent people continued working in area
- Software tools research focused elsewhere
  - HW verification
  - IDEs
  - Program transformation and refinement
  - Languages

2000+

- Suddenly, burst of software engineering research tools
  - Programming languages and formal methods communities
- Why?
  - Y2K?

My Hypothesis

- Shift of emphasis from program verification to bug detection
- Practical success stories
- (later) SAT solving

Historical Aside

- I took sabbatical at Microsoft Research in 1997
  - MSR was 4 years old and growing rapidly
  - Went to see SW development in the "real world"
- Microsoft was the leading software company
  - 2 years after Windows 95
  - Think Google in 2005, Facebook in 2012, …

State of MS Software Development (c. 2000)

- Not very good
  - MS tools were worse than open-source Unix tools
    - SLM, ed, vc
    - No one used Visual Studio
  - No software engineering discipline
    - Total “hero” programmer culture
  - Widespread arrogance
    - eg Aaron Contorer

- Leaders realized they were in trouble
  - Struggling to ship Windows 2K – enterprise software
  - Office could barely crank out a release
    - 2 months of new code in 2 year release cycle
  - Exchange/Outlook barely worked (but still put Lotus Notes out of business??)

Software Productivity Tools (SPT)

Part of Programmer Productive Tools

- SPT (Jim Larus) – research
  - SLAM, SDV
  - Vault, Fugue (typestate)
  - ESP (scalable program analysis)

- PPT (Amitabh Srivastava) – development and deployment
  - PREfix
  - PREfast
  - FxCop

- Wolfram Schulte's group focused on testing and model-based software development

What We Did Right

- Great hires from many academic disciplines
  - Sriram Rajamani (HW and formal methods)
  - Tom Ball (program analysis)
  - Manuvir Das (program analysis)
  - Manual Fahndrich (programming languages)
  - Rustan Leino (program verification)

- Reached out to academic community
  - Funding support
  - Internships
  - Talk broadly about MS's problems
  - Generate excitement about area in PL and FM communities

Right, cont'd

- Focus on defect detection, not verification
  - Find bugs, not prove their absence
  - Never could get Bill Gates to internalize the distinction
    - "even the most practical man of affairs is usually in the thrall of the ideas of some long-dead economist" – John Maynard Keynes
- Work closely with PPT tools group and MS developers
- Build real software and deploy it
  - Tom and Sriram agonized about spending a year working with Windows on SV

What We Did Wrong

- Missed security entirely
  - Code Red and Nimda (2001)
    - Existential threat to MS
  - We had done no research on buffer overflows
- Missed the big picture
  - Amitabh: tools can drive process change
  - Jim: tools can fundamentally improve software
  - Both wrong

Fast Forward

- Hired software engineers into group
  - Focused on people and process
  - Nagappan and Murphy built series of models that predicted bug density
- Built up theorem proving expertise and moved away from model checking and program analysis

- I got frustrated and started clean-slate project with Galen Hunt (Singularity)
  - Could we build more robust and secure systems with modern languages and tools? (Yes)
- Software tools and engineering research continues at MS and elsewhere

Biggest SWE Successes at MS

- Windows error reporting (Watson)
- Data mining and failure prediction models
- ~~SDV~~
- ~~PREFast~~

Watson

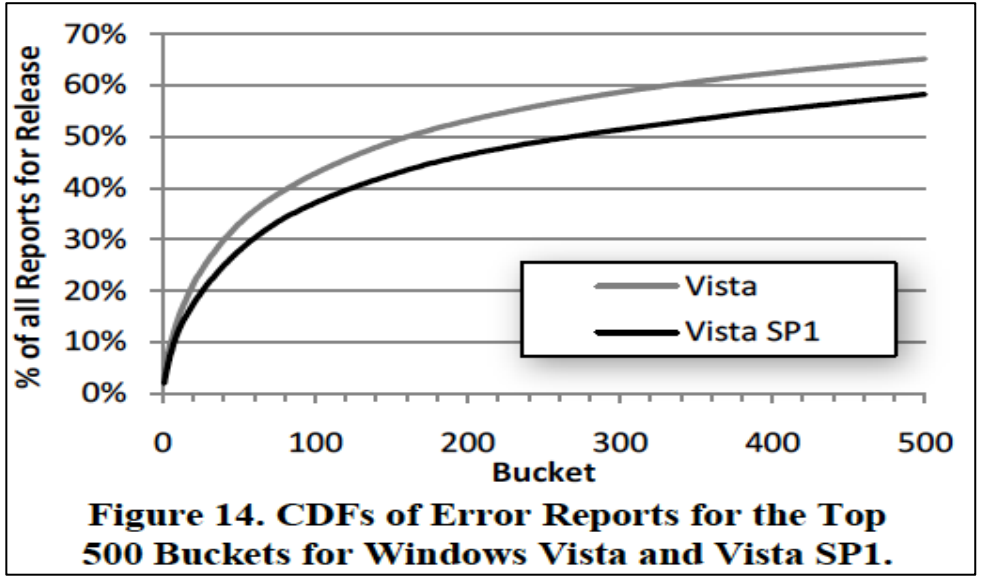


Figure 14. CDFs of Error Reports for the Top 500 Buckets for Windows Vista and Vista SP1.

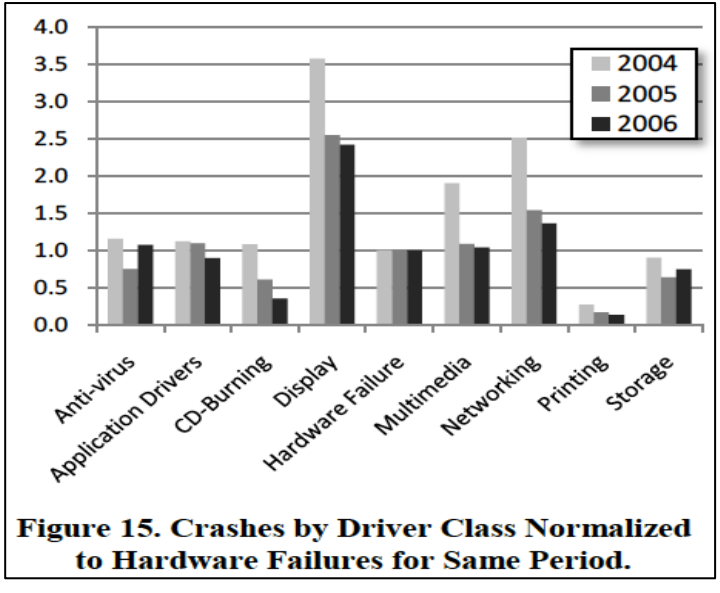


Figure 15. Crashes by Driver Class Normalized to Hardware Failures for Same Period.

Kinshumann, K., et al. (2011). Debugging in the (Very) Large: Ten Years of Implementation and Experience. CACM. **54:** 111-116.

Failure Models

Table 4: Overall model accuracy using different software measures

Model	Precision	Recall
Organizational Structure	86.2%	84.0%
Code Churn	78.6%	79.9%
Code Complexity	79.3%	66.0%
Dependencies	74.4%	69.9%
Code Coverage	83.8%	54.4%
Pre-Release Bugs	73.8%	62.9%

The Influence of Organizational Structure on Software Quality, Nachiappan Nagappan, Brendan Murphy, Victor Basili, International Conference on Software Engineering (ICSE 2008).

Why Isn't Everyone Using Tools?

- Tools are not good enough
  - User-engineering is more important than technical brilliance
- Do not find right bugs (cf WER)
  - Not all defects need to be fixed
- Hard to use
  - Badly trained students cannot write specifications
- Fix manifestation of problem, not problem
  - What is root cause of bugs?
    - Missing, outdated, incorrect knowledge
    - Human fallibility

How Not to Build a Software Tool

- PREFast
- Run for 2+ days on the Windows source
- Dump 50K bugs in the bug database
  - False positive rate > 50%
  - Heuristics prioritize “likely” bugs
- Enormously painful to developers who are busy and are judged on bug counts
  - Little connection to goal of shipping quality software

Which Tool Has 100% of the Bugs it Finds Fixed?

- SAGE (whitebox fuzz tester for security)
  - Patrice Godefroid
- Demonstrates input that cause memory error
  - == potential security problem
  - Cannot minimize importance of bug
- Fits developer workflow
  - Input that causes error
  - Can use standard debugging tools to understand
    - cf traces produced by static analysis tools

Specification is Greek

- Wolfram Schulte's group initial focused on model-driven testing
- Elegant formulation of testing
- No traction at Microsoft
  - Consulting model (researchers wrote specifications)
- Success was specifying Windows interfaces for EU (!) anti-trust settlement
- Same lack of understanding of specifications plagued Spec# and discussions with product groups

Improve Software Development?

- Preventing bugs vs finding them after they occur
- Continuous process improvement (six sigma, etc.)
  - Understand and fix root cause of defects
- Problems are rooted in organization structure, development process, training, discipline
  - These pieces are studied in SWE community
  - Have not been assembled into whole
  - Tools have a role to play to enforce process and assure quality

Would You Eat Here?

Even if the food passed an E. coli test?