

experience_△Financial Services



Is Your Web Application Really Secure?

Ken Graf



experience_△Banking

experience_△Capital Markets

experience_△Insurance

Microsoft[®]

What we will discuss today

- Pressures on the application lifecycle
- Why application security defects matter
- How to create 'hacker resistant' business logic in the development environment

Processes changing at Microsoft

"Under the old structure, you wrote the code and then reviewed it for bugs. But that doesn't work anymore. You have to change your processes and techniques from cradle to grave to develop software that can withstand a malicious attack."

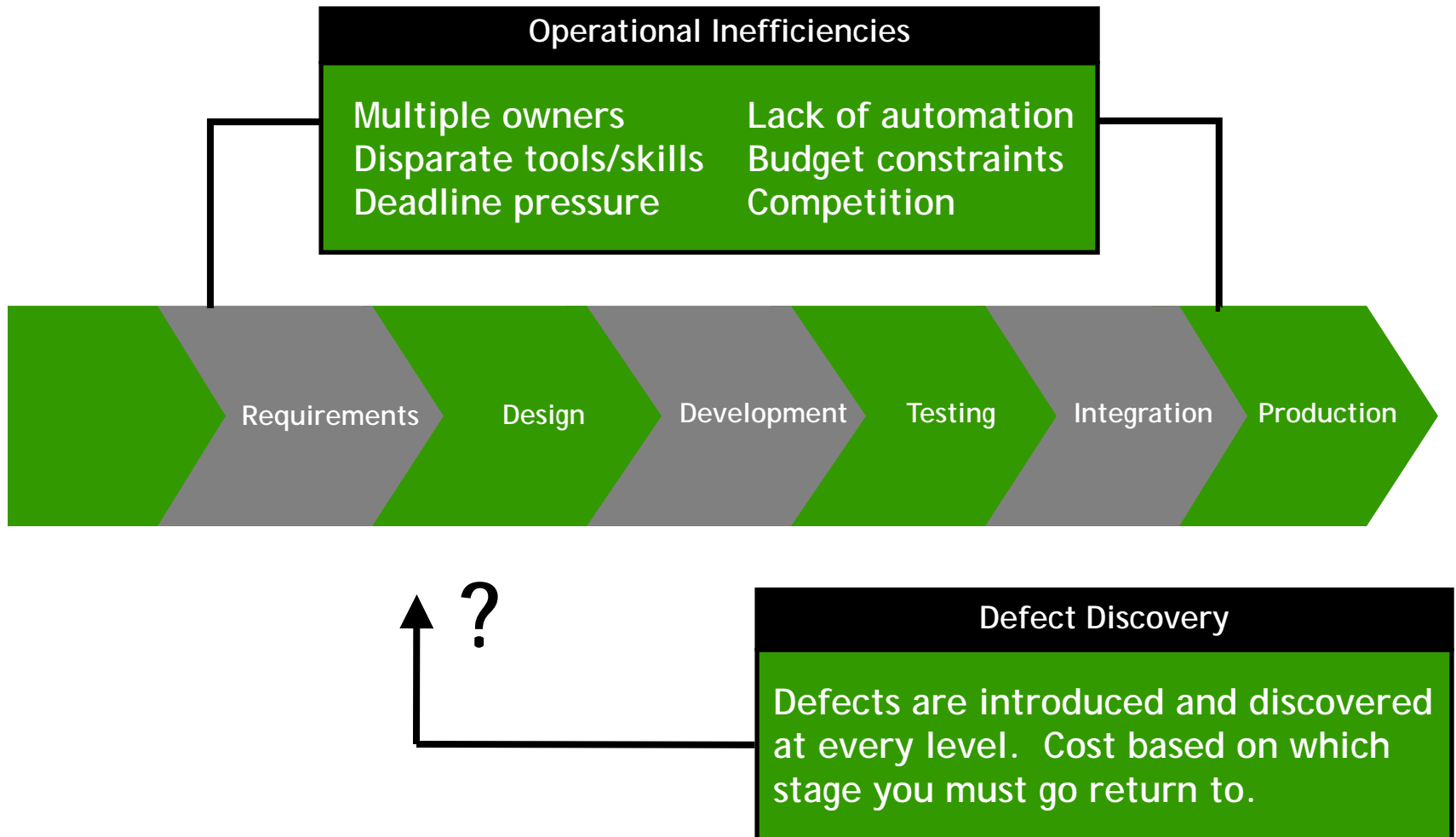
- Michael Howard, Senior Security Program Manager at Microsoft

<http://www.microsoft.com/indonesia/msdn/sdl.asp>

Typical Challenges for a Financial Service Company

- Well known and attractive site
- Thousands of legacy and new applications
- Application development spread across many business units
- Corporate mandate to build applications securely
- Reduce overall cost of development lifecycle
- Outsourcing >\$1M a year for “ethical hacking”
- Developers had limited knowledge of security issues

Current processes lead to security cost challenges

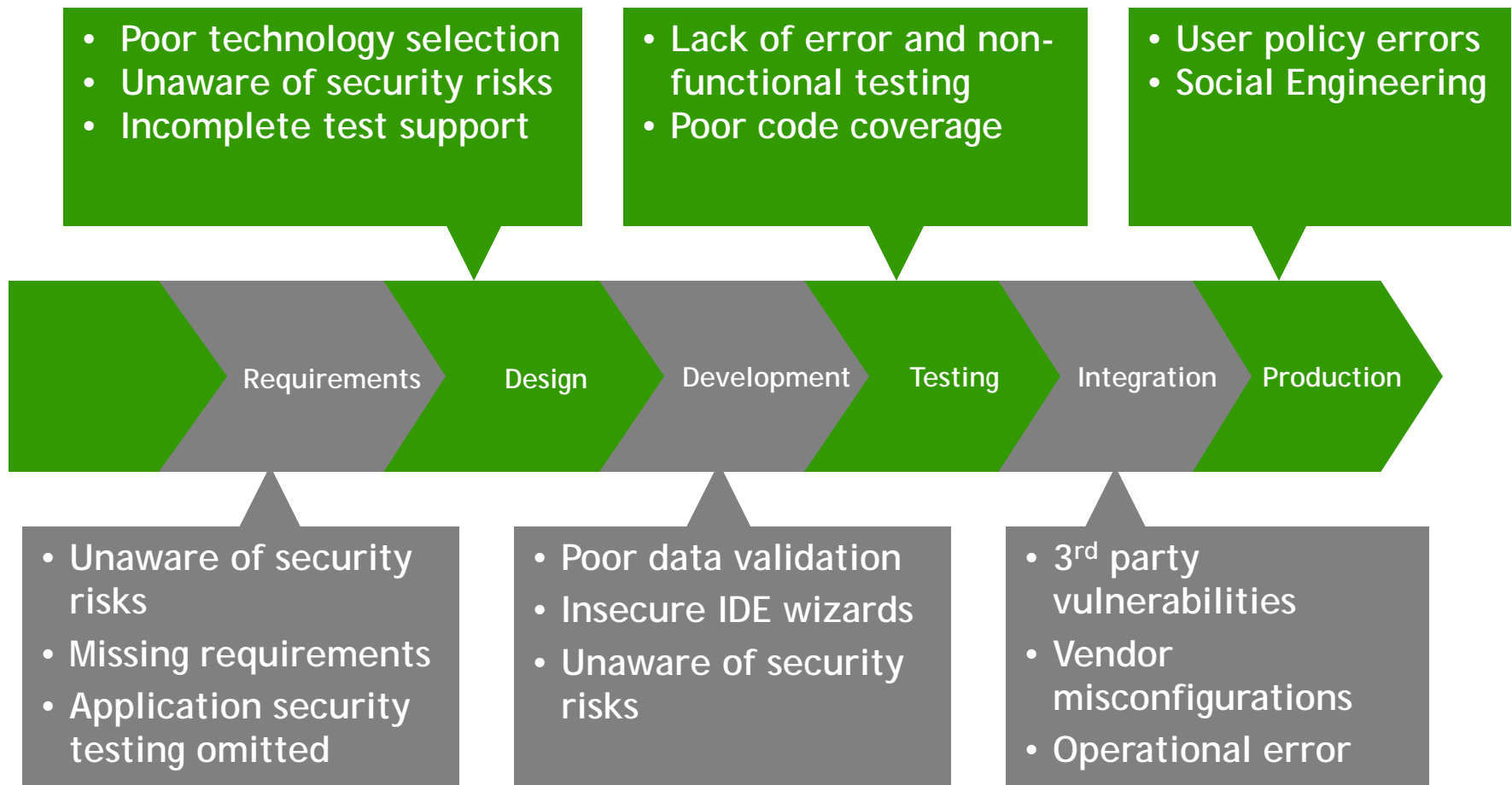


How much do security defects cost?

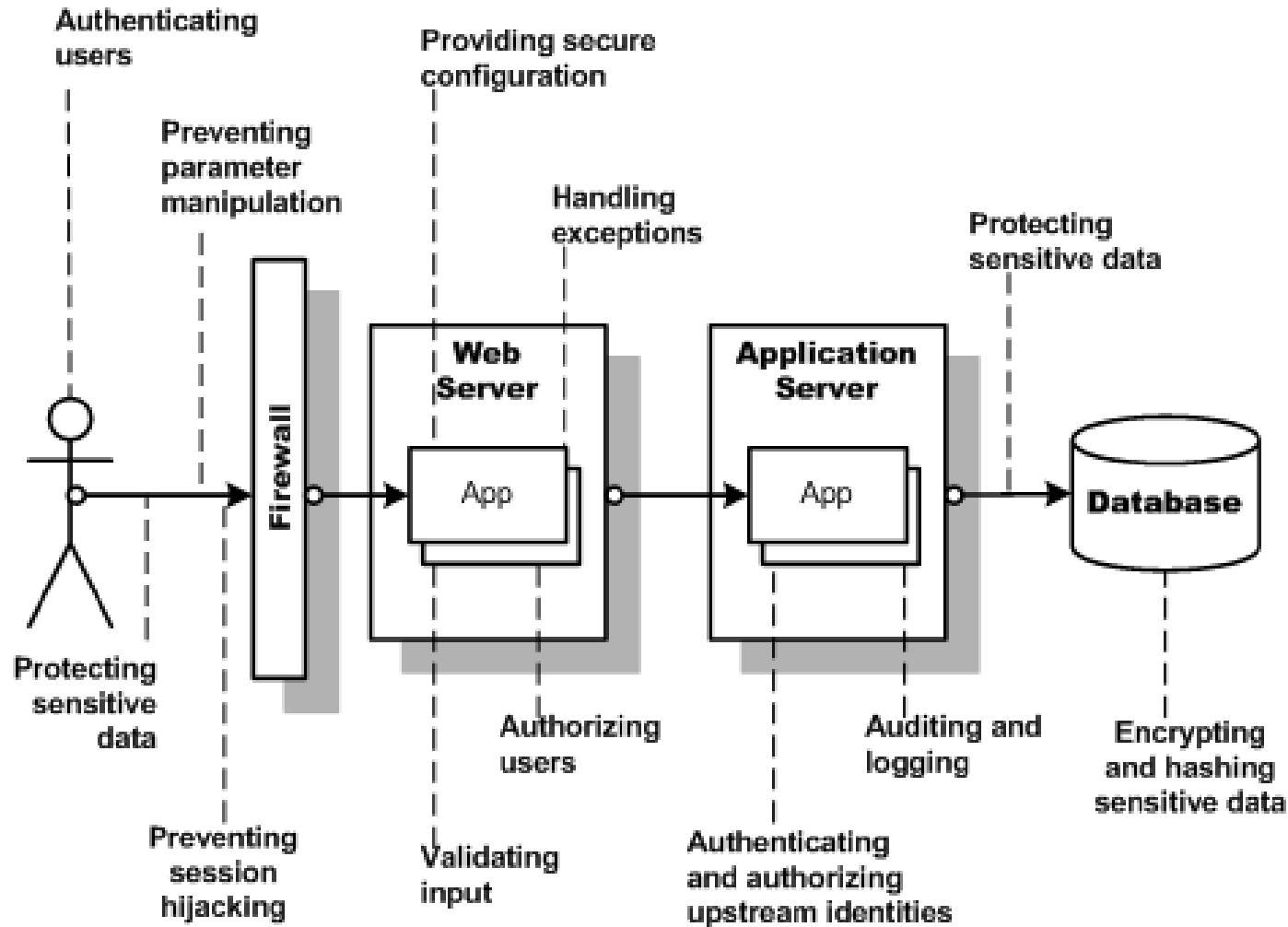
	Found in Design	Found in Coding	Found in Integration	Found in Beta	Found in GA
Design Errors	1x	5x	10x	15x	30x
Coding Errors		1x	10x	20x	30x
Integration Errors			1x	10x	20x

- NIST 2002 Software Quality Study
 - Estimates of relative cost factors of correcting errors as a function of where errors were introduced and found
- A very technical and detailed report on the economics
<http://www.nist.gov/director/prog-ofc/report02-3.pdf>

What are the common causes of defects?



What are the typical threats applications face



Sample XSS and SQL attacks



How should you address application security threats?

- Testing
- Education
- Process
- Principals
- Tools

Testing

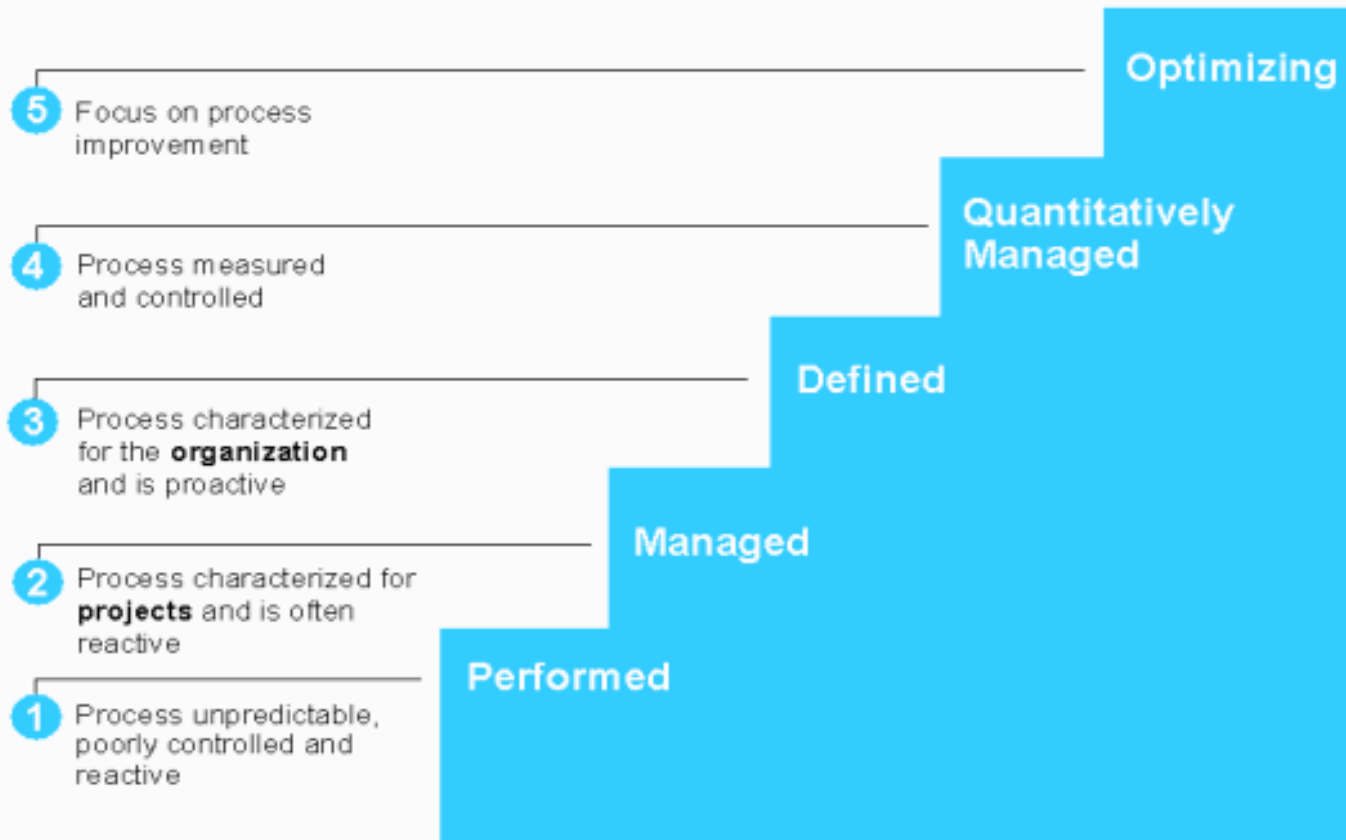
- Write tests first than the code.
- Application testing should be capable of being run on demand, including production.
- Development teams *must* know the proper use of the test framework.
- Security testing *must* be considered at the beginning of the process as part of the requirements and design stages.
- Automate testing.

Education

- Have everyone understand that application security relies on, *but is separate from*, network and host security.
- Development teams *must* know the proper use of the development framework.
- Education is ongoing, at least annual review.
- Implement dashboards of organizational wide metrics pertaining to application security certification and status.



The Maturity Levels



CMMI Combined Tutorial Feb 16, 2004

52

What is your Application Security Maturity?

Optimizing	<ul style="list-style-type: none"> • Striving for continuous process improvement
Quantitatively Managed	<ul style="list-style-type: none"> • Security reviews completed in all process stages • Published organizational standards for secure development
Defined	<ul style="list-style-type: none"> • Early stage security reviews • Security awareness is provided for all process stages • Predetermined policy is enforced for the entire organization
Managed	<ul style="list-style-type: none"> • Audits • Use of automated scanning tools prior to deployment • Organizational policy statement (exceptions permitted)
Performed	<ul style="list-style-type: none"> • Infrastructure penetration tests • Application policy statement (limited enforcement)

Why Principles?

- You can not cover all situations.
- Principles are basic rules from which more specific and targeted rules can be derived.
- Understanding secure web programming principles enables a reasonable programmer to find the right (=secure) solution for a new problem.
- Can be applied independently of IDE, languages, and tools in use at your site.

Principal #1 - All Input is Evil

- What is (web application) “Input” ?
- Anything HTTP:
 - Method
 - Path
 - Query parameters
 - HTTP headers (e.g. User-Agent, Range)
 - Body parameters
- SSL client data (certificate data)
- Use the .NET validation classes

Principal #2 - Use a Positive Security Model

- Positive security
 - Define exactly what the application is willing to accept and process.
 - Accepting only what the application can process prevents false positives.
- Negative security
 - Define what is hostile or not acceptable.
 - Addresses current (known) attacks and must be updated to handle future (unknown) attacks.
 - Broad (loose) definitions cause false negatives.

Principal #3 - Validate in a Trusted Environment

- Do you trust the application clients?
 - Hint: all input is evil...
- Do you trust your client's machine/browser?
- If it's worth doing, it's worth doing yourself (on your own machine).
- Client side validation is good for user experience. Security-wise, it is meaningless.

Principal #4 - Defensive Programming

- Applications are built in layers.
- Applications use 3rd party code (3rd party = the next department).
- Each layer has security responsibilities:
 - Do not trust the previous layer. Validate your input.
 - Do not trust libraries/internal data. Validate the data (indirect input).
 - Do not provide the next layer with “bad” data – contain the problem.
 - Halt the system as soon as questionable/inconsistent data is detected.
 - Exit gracefully, clean behind you. Don’t leave the system in an undefined state.
- Coarse vs. Fine Grain Validation

Principal #5 - Don't Reinvent the Wheel

- A lot of security (and programming) facilities are available with your programming environment, especially with Microsoft ASP.NET.
- Usually, this is (well) debugged code, with (some) security in mind.
- Get to know your programming environment/libraries. Use what you can.
- Use well, avoid pitfalls, and keep track of security issues as they get published.

Principal #6 - Eliminate Security by Obscurity

- Assume the user can guess/reconstruct your application information.
 - Database structure, field names, data types, etc...
 - Information contained in cookies or tokens.
 - 3rd party products used in the application.
 - Location of application files.
- Acknowledge that hostile parties will have access to your application.
 - Ex-employees.
 - Outsourced developers.
 - Customers and Partners (via service agreements).
 - Competitors (via all of the above).

Principals Review

- Trust nothing – All Input is Evil
- Use a positive security model
- Validate in a trusted environment
- Defensive programming
- Use existing infrastructure
- Eliminate security by obscurity

Security Testing Demonstration



How do we make it happen?

- Improve security awareness.
- Security testing integrated into development process.
- Categorize application risk and liability.
- Security reviews should be part of requirements, design, and development stages.
- Give developers access to automated tools.
- Zero tolerance enforcement.

Questions & Answers

