Secure Software Development
Software Security Touchpoints

Objectives

- Explain the concept of a Touchpoint
- List the Software Security Touchpoints (McGraw)
- Identify the most effective security practices to have within software development (McGraw)
- Define the acronym OWASP
- Explain the basic premise for CLASP
- List the 7 best practices for application security according to CLASP
- Explain the difference between the reactive and proactive approaches to software security
- Recognize an example of a bug and a flaw within software
- Explain the economic impact of various security activities

- Define the following terms from last week's reading
  - Single Loss Expectance

Product of the value of an asset X percent of asset lost

  - Annual Rate of Occurrence

The number of threats of a specific type expected to manifest themselves in a year.

  - Annual Loss Expectancy

SLE x ARO

How much risk is there in a given year.

Discussion

- What are the best practices for software development?

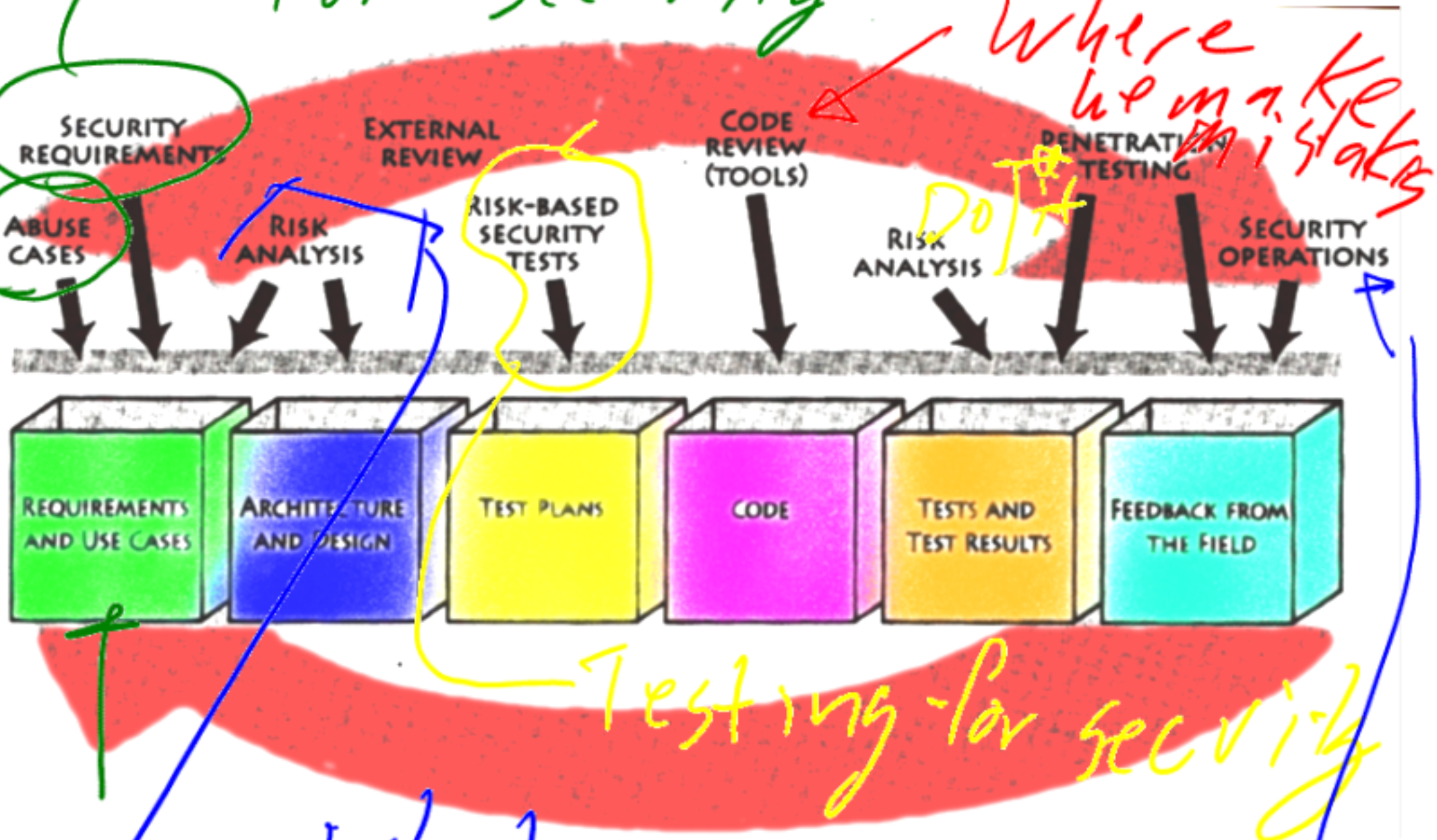
A solid planning phase.
Thourough testing
Solid design
Design/code review

What is a security touchpoint

- A best practice for ensuring software security
  - Based on the concepts of normal software development

A mechanism where a person impacts the security of the system.

Software Security Touchpoints



Software Security Touchpoints by effectiveness (McGraw)

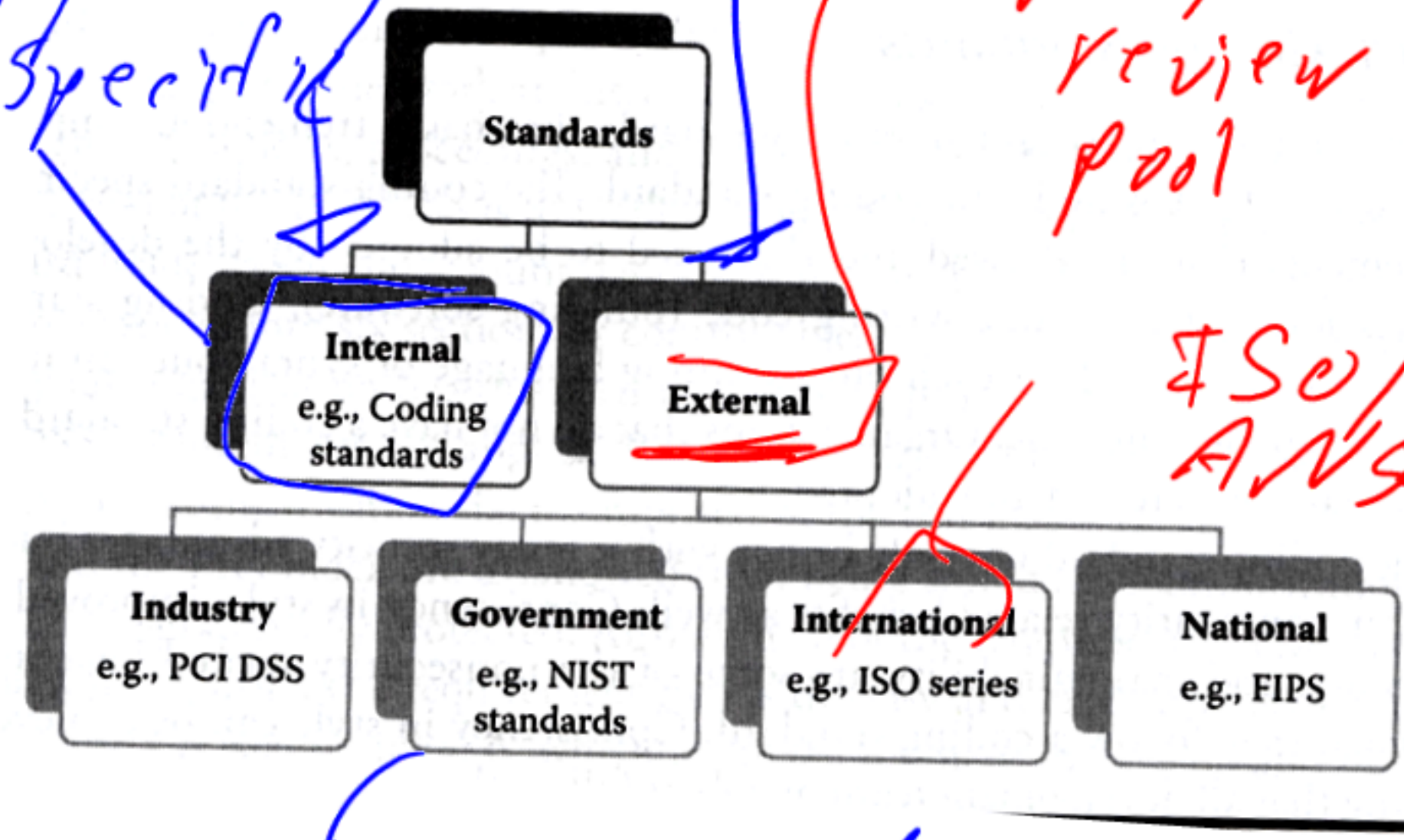
1. Code Review
2. Architectural Risk Analysis
3. Penetration Testing
4. Risk-Based Security testing
5. Abuse Cases
6. Security Requirements
7. Security Operations

Techniques for ensuring secure Sw

Where do we get our security approaches?

Google Standards organizations

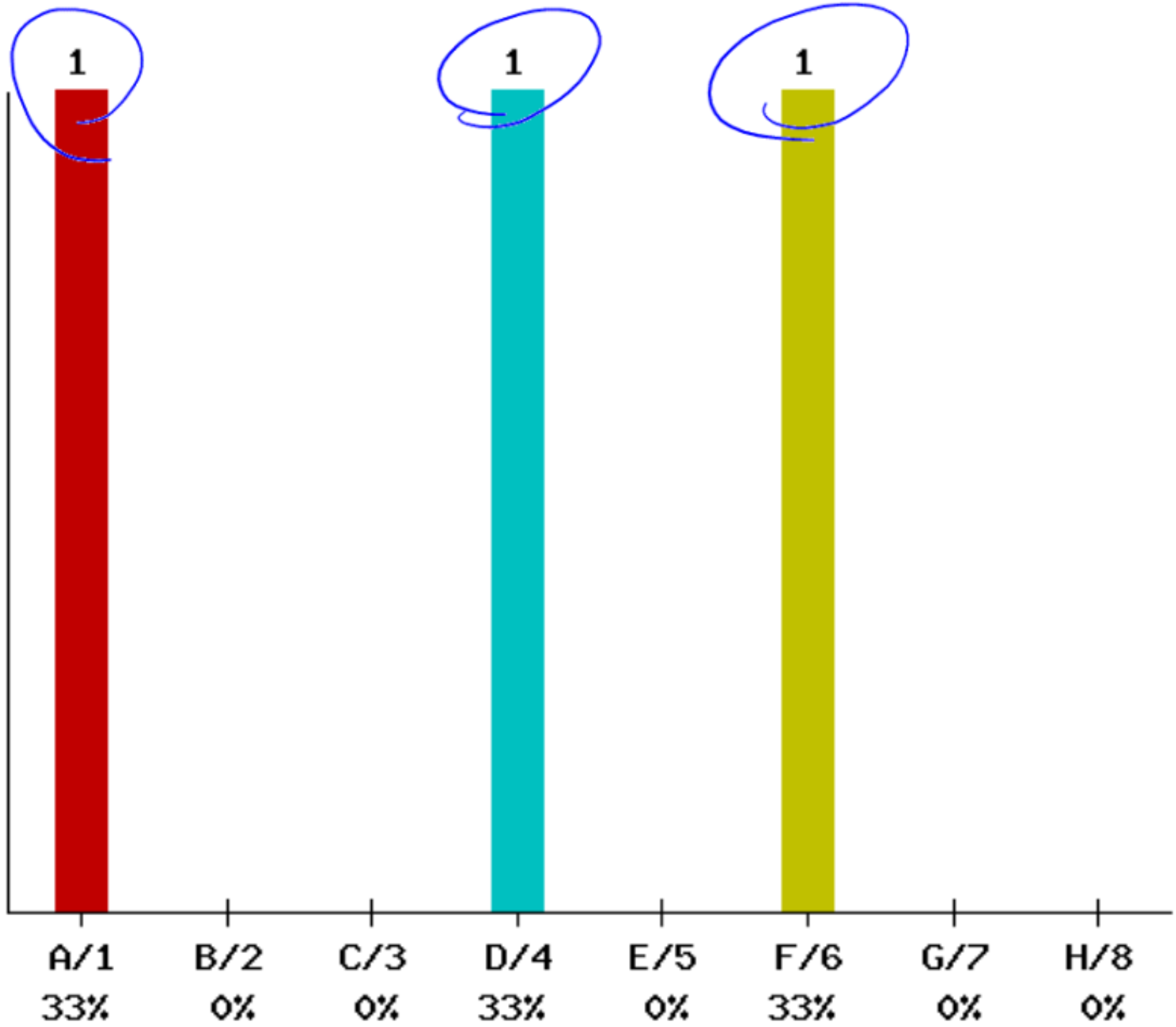
Sources of standards



Time

Trivia

- According to OWASP in 2007, the top web application vulnerability was
  a. Injection flaws
  b. Insecure Communications
  c. Malicious file execution
  d. Cross Site Scripting (XSS)
  e. Buffer overflow
  f. Information leakage and improper error management
  g. What'chu **talkin**' 'bout, **Willis?**


1
1
1
A/1
33%
B/2
0%
C/3
0%
D/4
33%
E/5
0%
F/6
33%
G/7
0%
H/8
0%

METHODOLOGY

Our methodology for the Top 10 2007 was simple: take the MITRE Vulnerability Trends for 2006, and distill the Top 10 *web application security* issues. The ranked results are as follows:

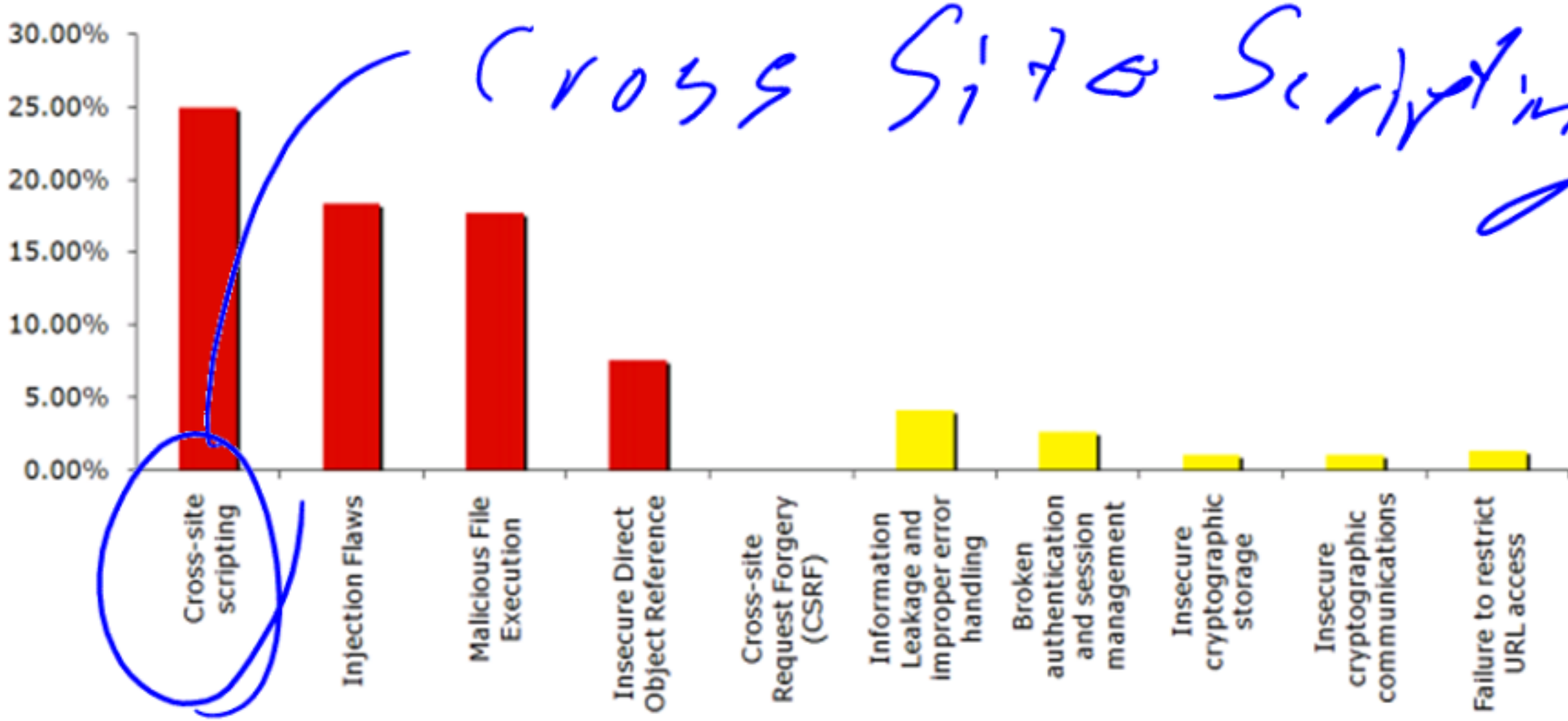


Figure 2: MITRE data on Top 10 web application vulnerabilities for 2006

- Source: OWASP

OWASP 2010 versus 2007

OWASP Top 10 – 2007 (Previous)	OWASP Top 10 – 2010 (New)
A2 – Injection Flaws	A1 – Injection
A1 – Cross Site Scripting (XSS)	A2 – Cross-Site Scripting (XSS)
A7 – Broken Authentication and Session Management	A3 – Broken Authentication and Session Management
A4 – Insecure Direct Object Reference	A4 – Insecure Direct Object References
A5 – Cross Site Request Forgery (CSRF)	A5 – Cross-Site Request Forgery (CSRF)
<was T10 2004 A10 – Insecure Configuration Management>	A6 – Security Misconfiguration (NEW)
A8 – Insecure Cryptographic Storage	A7 – Insecure Cryptographic Storage
A10 – Failure to Restrict URL Access	A8 – Failure to Restrict URL Access
A9 – Insecure Communications	A9 – Insufficient Transport Layer Protection
<not in T10 2007>	A10 – Unvalidated Redirects and Forwards (NEW)
A3 – Malicious File Execution	<dropped from T10 2010>
A6 – Information Leakage and Improper Error Handling	<dropped from T10 2010>

What is CLASP

- The Comprehensive Lightweight Application Security Process
  - Supports the incorporation of security processes into each phase of the software development lifecycle

Set of steps for secure a SW app / system.

CLASP Practices

- Institute awareness programs
- Perform application assessments
- Capture security requirements
- Implement secure development practices
- Build vulnerability remediation procedures
- Define and monitor metrics
- Publish operational security guidelines

CLASP has 30 activities

30 of them.

Reactive versus proactive security

- Reactive + More Common
  - Something goes wrong
  - How can we make this software secure?
- Proactive ← Better
  - We are building this software from scratch?
  - How can we ensure it is secure

Bugs versus flaws again

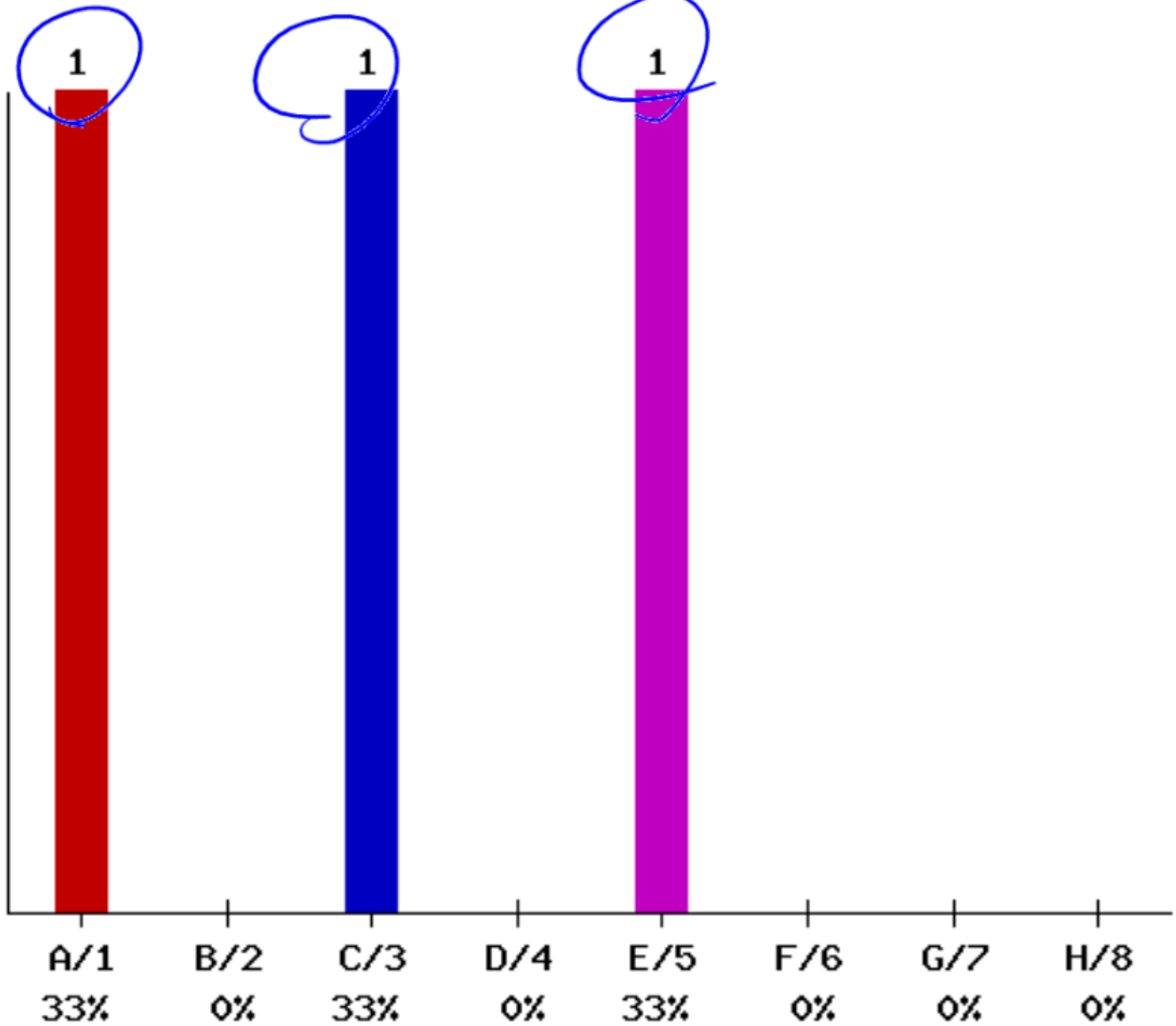
Last week

Bugs	Flaws
Buffer overflow: stack smashing	Method over-riding problems (subclass issues)
Buffer overflow: one-stage attacks	Compartmentalization problems in design
Buffer overflow: string format attacks	Privileged block protection failure (DoPrivilege())
Race conditions: TOCTOU	Error-handling problems (fails open)
Unsafe environment variables	Type safety confusion error
Unsafe system calls (fork(), exec(), system())	Insecure audit log design
Incorrect input validation (black list vs. white list)	Broken or illogical access control (role-based access control [RBAC] over tiers)
	Signing too much code

Trivia

- What is the relative percentage of bugs versus flaws

a. 20% bugs, 80% flaws
b. 30% bugs, 70% flaws
c. 40% bugs, 60% flaws
d. 50% bugs, 50% flaws
e. 60% bugs, 40% flaws
f. 70% bugs, 30% flaws
g. 80% bugs, 20% flaws


1
1
1
A/1
33%
B/2
0%
C/3
33%
D/4
0%
E/5
33%
F/6
0%
G/7
0%
H/8
0%

Flaw Example

```
public class Pass_controlflow_good
{
        public static void main (String[]args){
        try {
        BufferedReader read  new BufferedReader(new
FileReader("Passwords.txt"));
        String adminPass = read.readLine();
        if (args[0].equals(adminPass))
                System.out.println("Access Granted");
        else
                System.out.println("Your password is
not valid, please reenter it.");
        }
        catch(IOException e){
        System.out.println("io error");}
        return;
        }
}
```

plaintext passwords

Bug Example

```
int get_buff(char *user_input) {
  char buff[4];
  int length = strlen(user_input)+1;
  memcpy(buff, user_input, length);
  return length;
}

int main(int argc, char*argv[]){
  get_buff(argv[1]);
  return 0;
}
```

C / C++

← fixed length

← No check of user input

make failing

Cost of Fixing Defects at Each Stage of Software Development

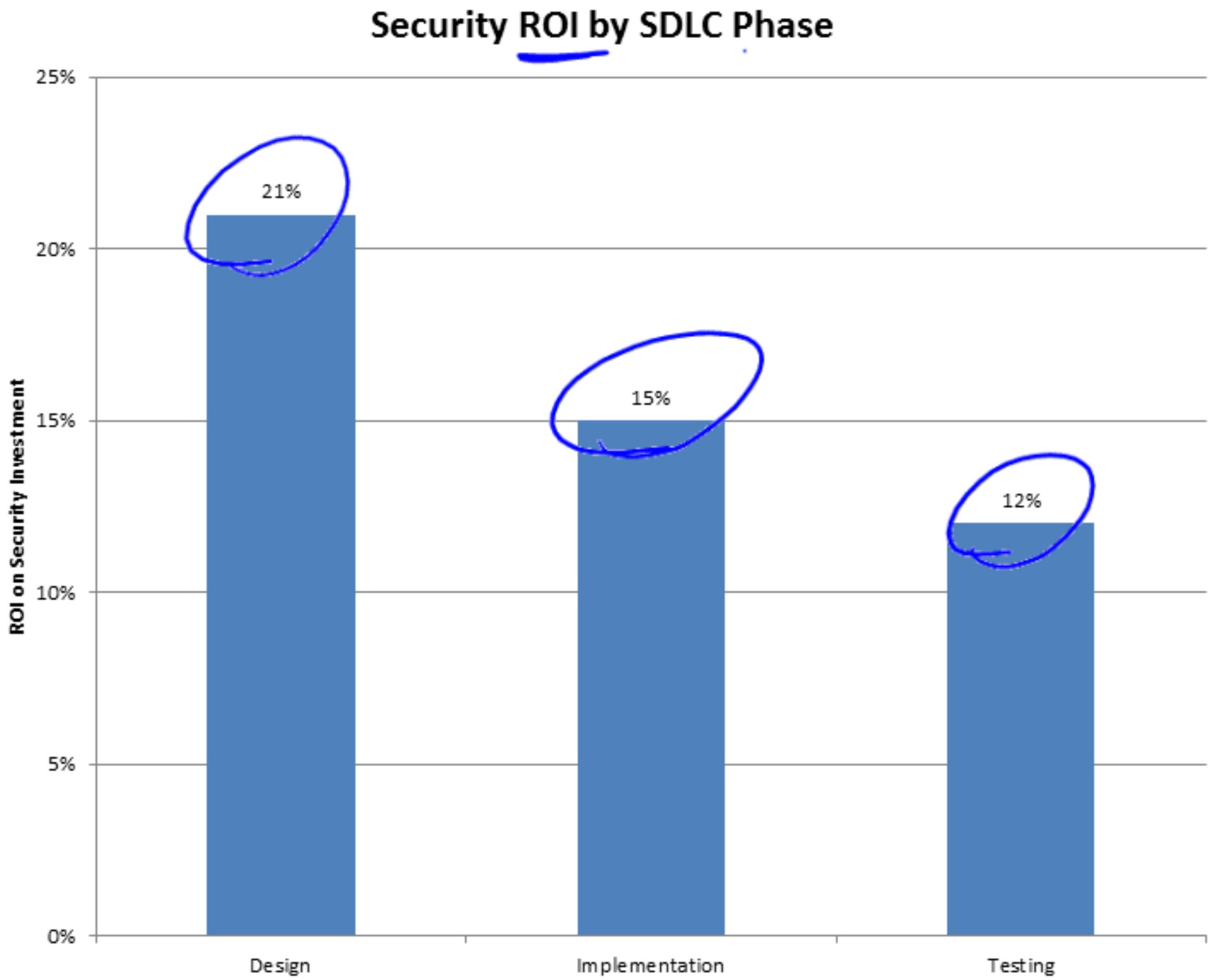
Expensive

Cheap

Expensive to fix securities

Security here

Return on Investment for Security Operations
Secure Business Quarterly, Q4 2001



Security

Code Review With a tool

- Artifact(s)
  - Source code
- Very effective at finding implementation bugs
- Extremely important to do this in C / C++ Development
- Highly efficient with modern tools

Risk Analysis

- Artifact(s)
  - Design
  - Specification

- Identifies problems with core design of the software

Architectural

  - Poor protection of critical data
  - Authentication failures
  - Poor design practices

How is the product built?

Penetration Testing

- Artifact(s)
  - Deployed System
- Can be useful at verifying architectural risk mitigations
- Gives a good understanding of the software in its deployed environment
  - May be more of a badness-o-meter than an actual security metric

Built / Running

Risk Based Security Testing

- Artifact(s)
  - Units
  - Completed System

- Tests that the software can handle abuse cases properly

Abuse Cases

- Artifact(s)
  - Requirements
  - Specification
- Puts the developers into the mindset of an attacker
- Describe the systems behavior when under attack

Security requirements

- Artifact(s)
  - Requirements

- Good security requirements convey required functional security as well as behaviors of the system
  - Generally considered to be constraints on functional requirements in order to achieve security goals