

# Integrating security into the J2EE development process

Doing it right from the start

Eelco Klaver ([eklaver@xebia.com](mailto:eklaver@xebia.com))  
J-Spring 2006



# If cars were build like applications...

- Cars would have no airbags, mirrors, seat belts, doors, roll-bars, side-impact bars, or locks, because no-one had asked for them. But they *would* have six cup holders.
- Car design would assume that safety is a function of road design and drivers are considerate.
- Not all components would be bolted together securely and many of them would not be built to tolerate even the slightest abuse.
- Safety tests would assume frontal impact only. Cars would not be tested for stability in emergency maneuvers, brake effectiveness, side impact and resistance to theft.
- The only warning indicator would be large quantities of smoke and flame in the cab.



# Goal of the presentation

- Create an awareness that J2EE security does not solve all security problems.
- Present a number of security activities that should be added to your software development process in order to be able to create secure applications.

# Contents

- J2EE – Security out of the box
- Consequences and impact
- Adding security to the software development process
  - Requirements
  - Design
  - Coding
  - Test
  - Deployment
- Conclusions
- How to introduce secure application development?
- Questions

# J2EE – Security out of the box

- J2EE Specification: security goals

*Transparency:* Application Component Providers should not have to know anything about security to write an application.

- So: Developers do not need to know about security

*Isolation:* The J2EE platform should be able to perform authentication and access control according to instructions established by the Deployer using deployment attributes, and managed by the System Administrator.

- So: Administrators take care of security at the end

# J2EE – Security out of the box

- J2EE security
  - Simplified, unified security model
    - Standard authentication
    - Role-based authorization on URL resources and EJB method level
  - Based on users, groups and roles
  - Access control based on permissions approach
  - Both declarative and programmatic
  - Support single sign on access to application services
- Authorization Model
  - Security roles specified by application developer.
  - Deployer maps roles to identities.

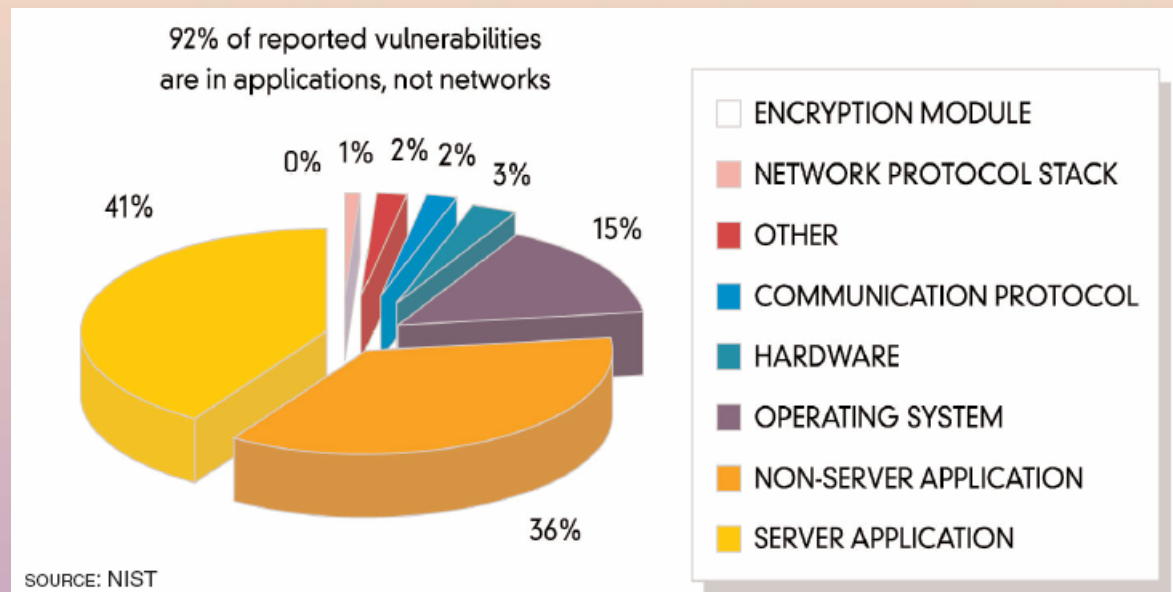
# J2EE – Security is not enough

- But it lacks important features
  - Only access control on web resources and EJB methods
  - No instance-based or segment-based authorization
  - No protection against XSS, injection flaws, DoS, etc.
  - Limited unit test possibilities of J2EE security
  - Standardized audit logging
  - No support for user management API
  - Etc.
- Recommendation
  - Not just administrators, but developers should also know about security



# Current status on application security

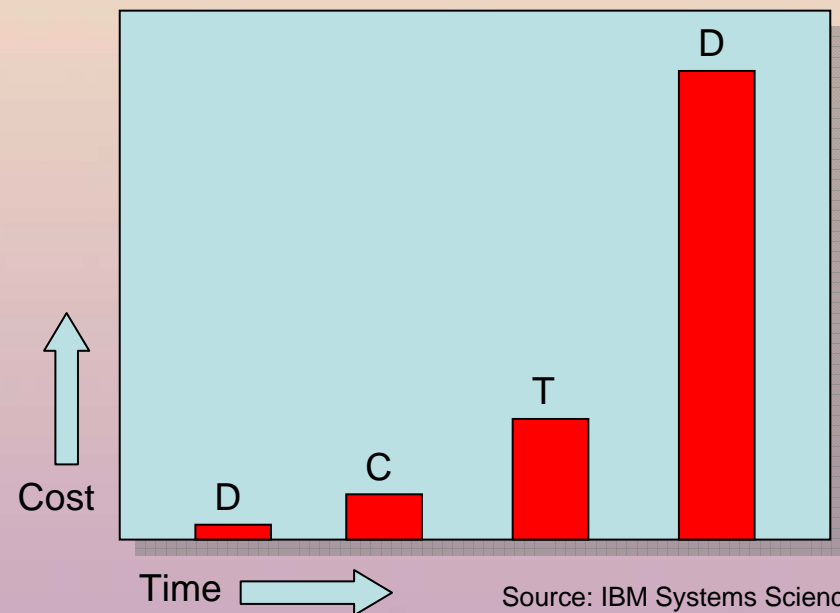
- Most security vulnerabilities are caused by errors in applications.
  - Gartner: > 70%
  - NIST: 92%



- Conclusion
  - Applications are not inherently secure

# Impact of adding security at the end

- Cost increases exponentially with each phase
- Effectiveness decreases
- Longer time to market
- Revenue loss

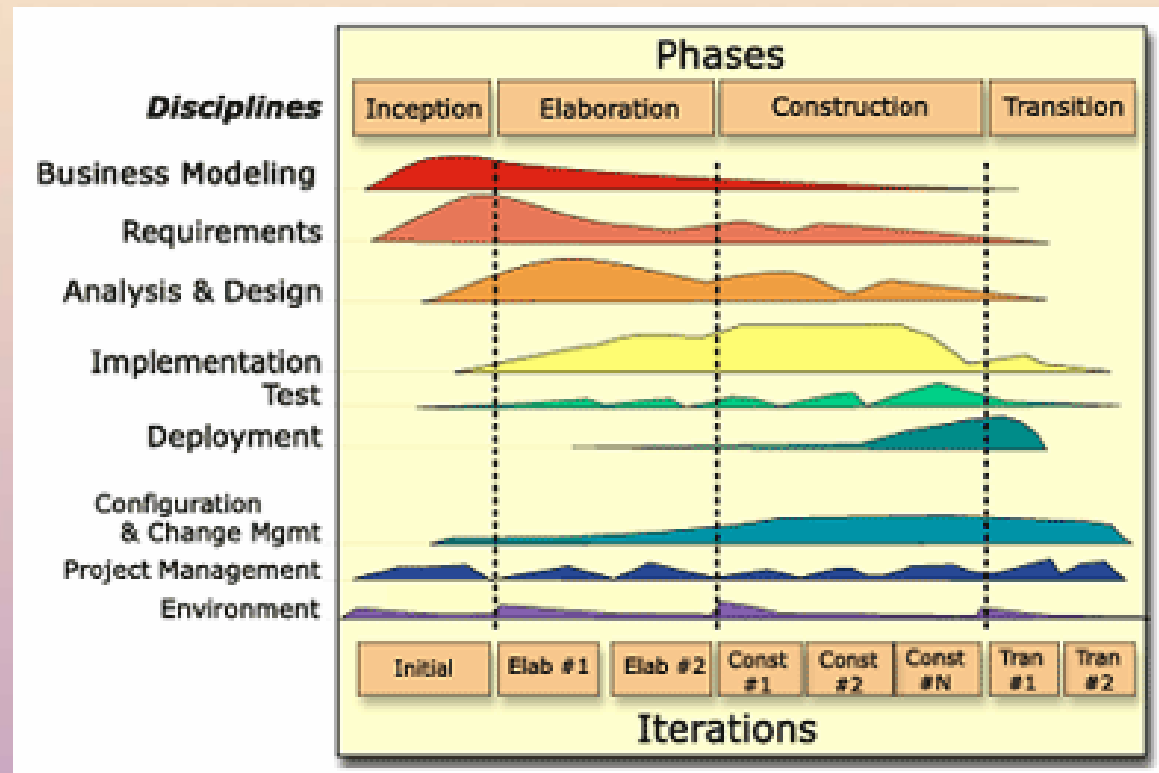


- **Recommendation:**

- Security should be an integral part of the software development process and not an afterthought that is added at the end of the project.

# Software Development Process

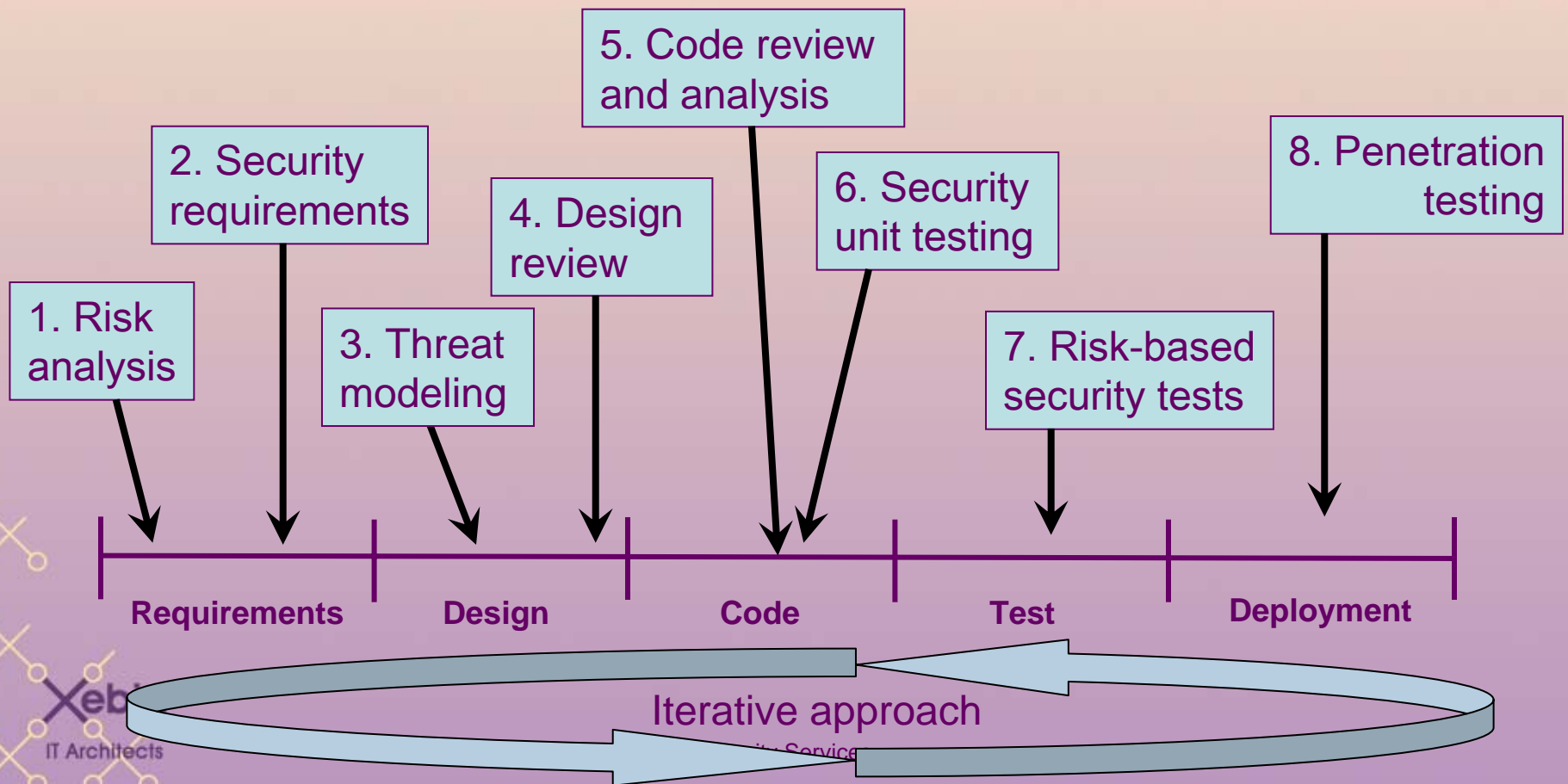
- Requirements
- Design
- Coding
- Test
- Deployment



- Beneficial for all software development methodologies, such as iterative, agile and waterfall models.

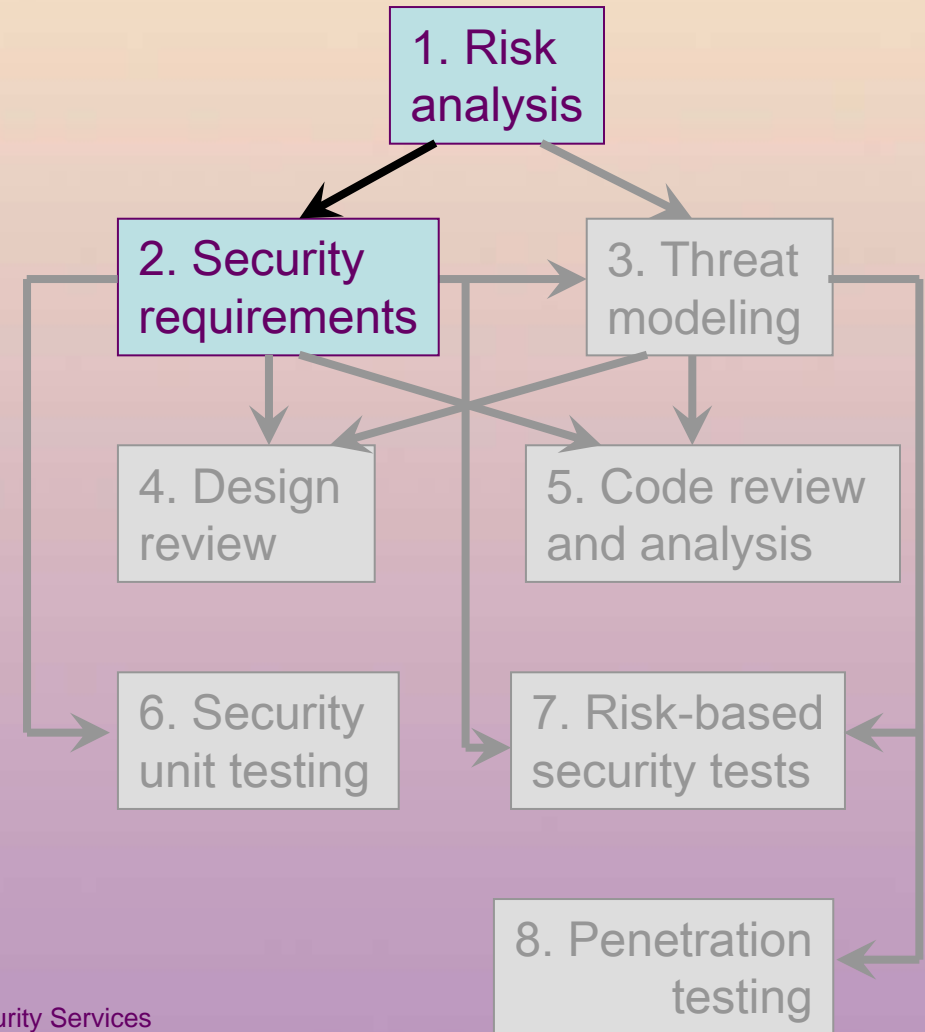
# Software Development Process

- Adding security activities to the development process



# Software Development Process

- **Requirements**
  - Risk Analysis
  - Security requirements
- Design
- Coding
- Test
- Deployment



# Activity 1: Risk Analysis

- Analysis and measurement
  - How much money/resources can be spent on security?
  - Which system components or other aspects should be targeted first?
  - How much improvement is gained by security expenditures
- Risk analysis methods
  - Quantitative:  $ALE = SLE (\$) * ARO$
  - Qualitative: High, Medium, Low
- Define
  - Business goals
  - Business risks in terms of likelihood and impact
  - Business involvement

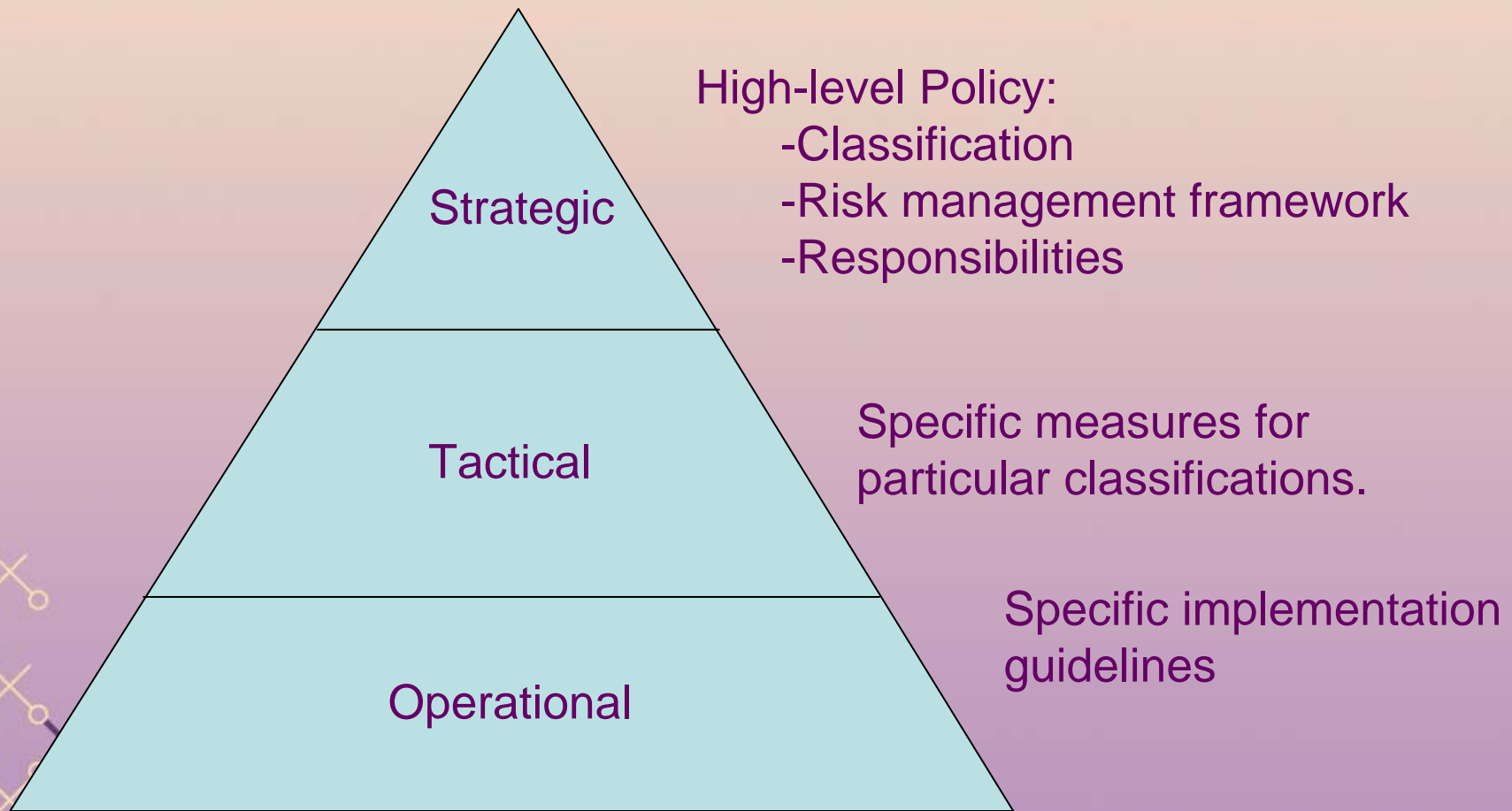


## Activity 2: Security requirements

- Define explicit security requirements and separate them from functional requirements
  - Generic requirements based on security policy
  - Specific requirements based on product-specific threats

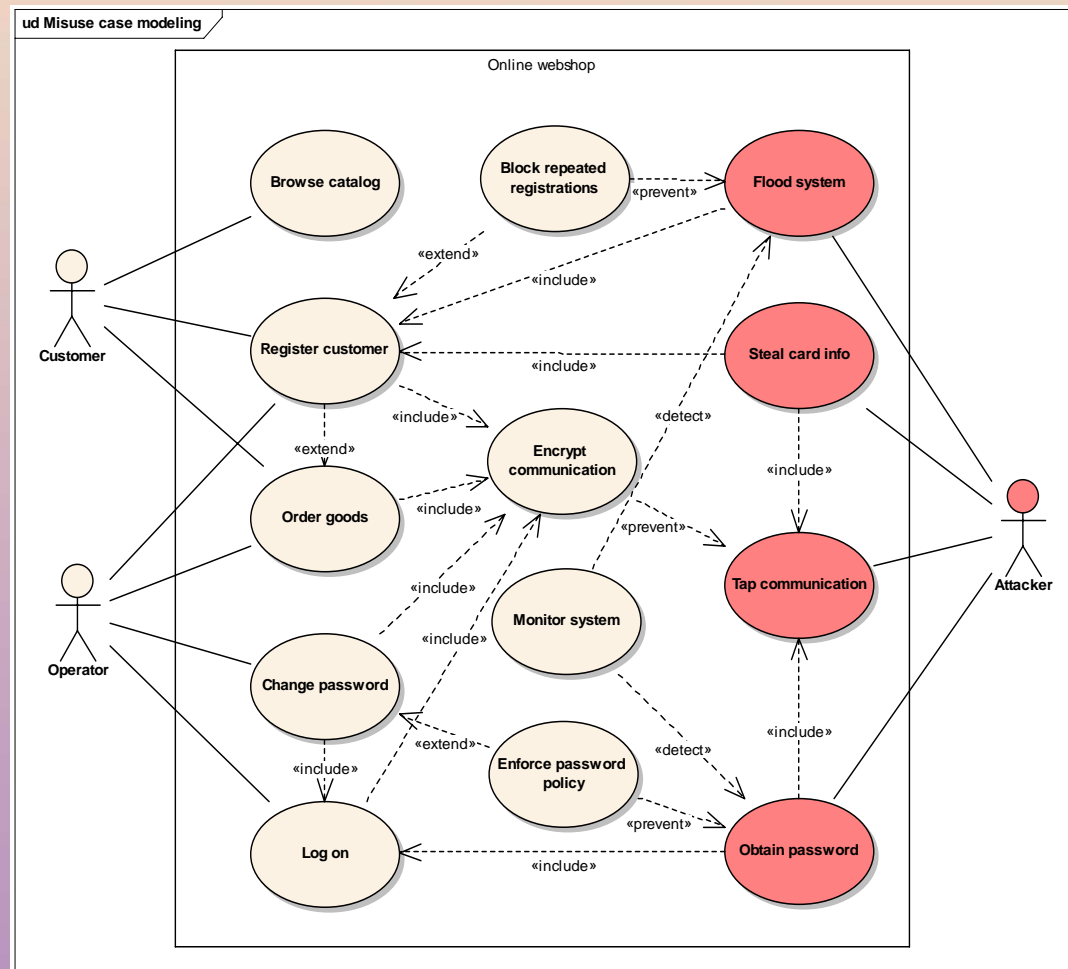
# Activity 2: Security requirements

- Generic security requirements based on security policy



# Activity 2: Security requirements

- Capture specific requirements with use case and misuse case modeling

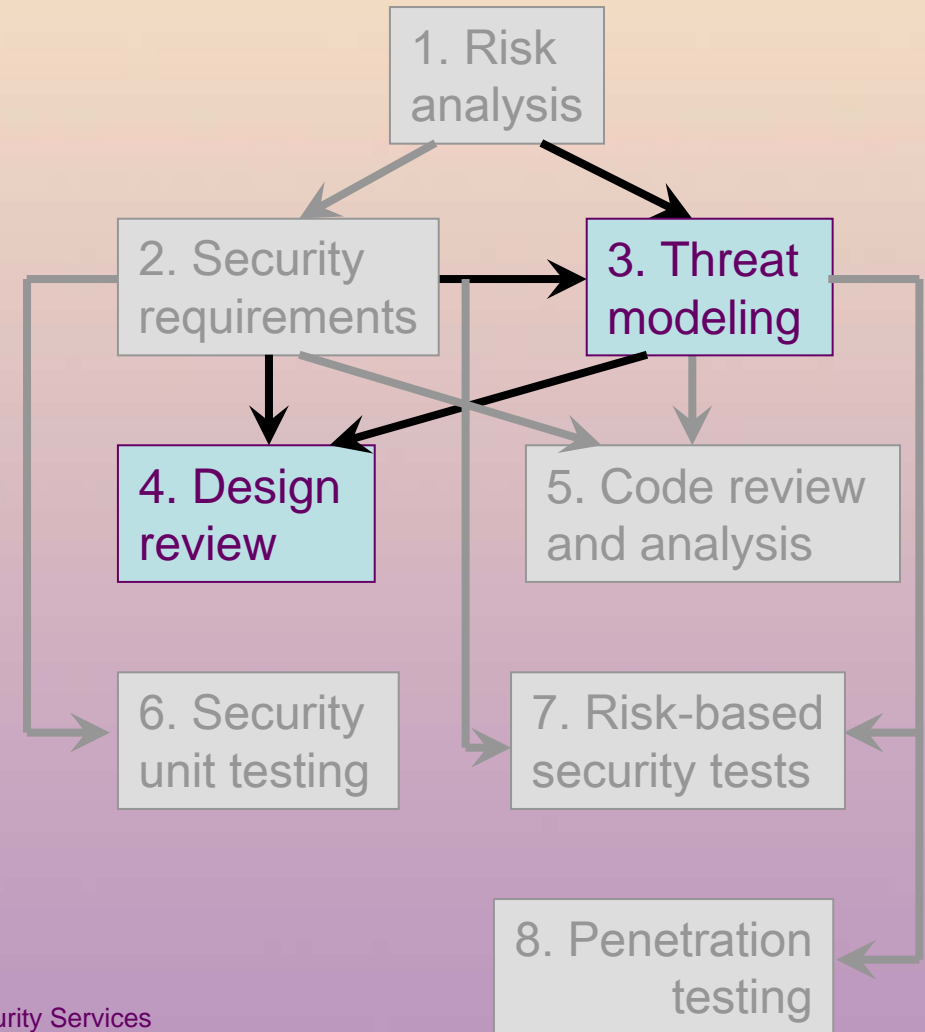


# Requirements best practices

- Make use of risk analysis to get priorities
- Always start with a clear security policy
- Translate policy into explicit and measurable requirements
- Use use cases to define roles and authorizations
  - Easily be mapped to J2EE roles and authorizations
- Use misuse cases to define product specific security requirements

# Software Development Process

- Requirements
- **Design**
  - Threat modeling
  - Design review
- Coding
- Test
- Deployment



# Activity 3: Threat modeling

- Refine the threats identified in the misuse case models
- Threat modeling is a security-based analysis that:
  - Helps understand where the product is most vulnerable
  - Identifies and evaluates the threats to an application
  - Aims to reduce overall security risks
  - Finds assets
  - Uncovers vulnerabilities
  - Helps form the basis of security design specifications
- Link identified technical risks to the defined business goals and risks



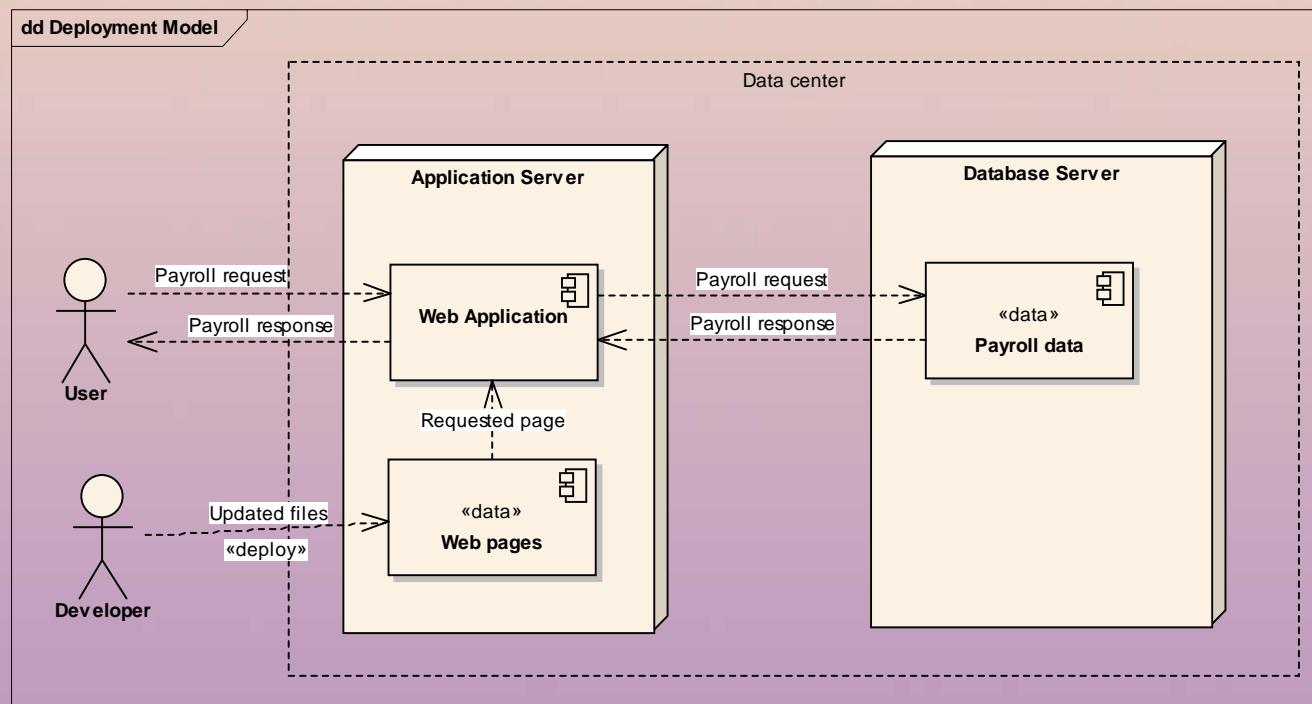
# Activity 3: Threat modeling

- Identify threats with **STRIDE**:
  - Spoofing identity
  - Tampering with data
  - Repudiation
  - Information disclosure
  - Denial of service
  - Elevation of privilege
- Use **DREAD** to rate threats
  - Damage potential
  - Reproducibility
  - Exploitability
  - Affected users
  - Discoverability
- Other threat modeling techniques
  - Attack trees
  - OCTAVE

# Activity 3: Threat modeling

## Step 1

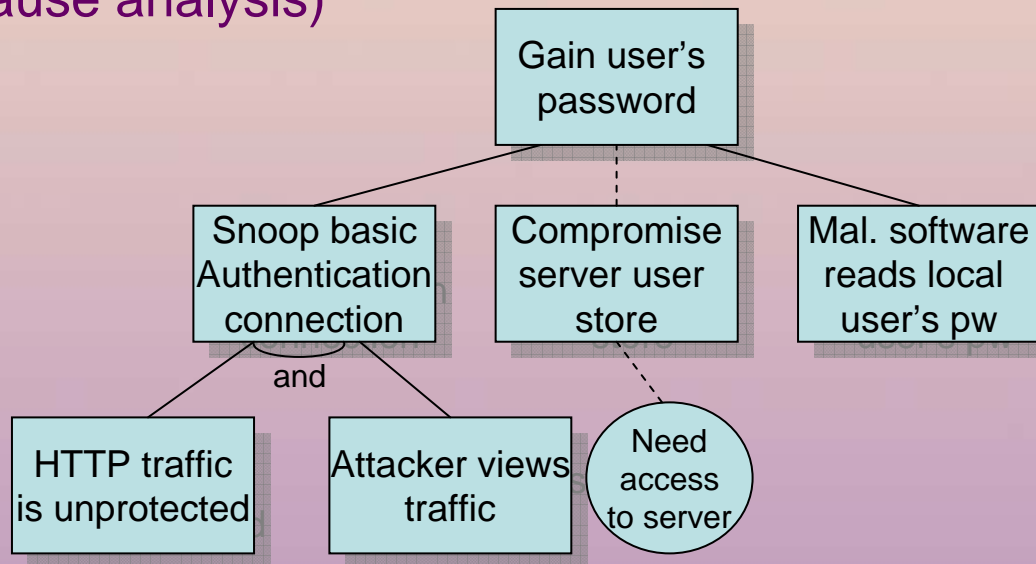
- Decompose application
  - Use DFD of UML deployment diagrams
  - Distinguish (inter)actors, data store, processes and boundaries



# Activity 3: Threat modeling

## Step 2

- Determine the threats to the system
  - Use STRIDE to categorize threats
  - Use threat trees to determine how the threat can manifest itself (root cause analysis)



- Rank the threats by decreasing risk, e.g. using DREAD

# Activity 3: Threat modeling

## Step 3

- Choose how to respond to the threats
  - Accept the threat
  - Mitigate the threat
  - Insure the threat
  - Inform the user of the threat

# Activity 3: Threat modeling Tools

The screenshot displays the Threat Analysis & Modeling Tool interface. The main window title is "Threat Analysis & Modeling Tool - C:\Xebia\Security\Components\User Manager\UserManager.atmx". The menu bar includes File, Edit, Threat Model, Analytics, Visualizations, Reports, Tools, and Help. The interface is divided into several panes:

- Threat Model:** A tree view on the left showing the application decomposition. The selected path is "Confidentiality > Unauthorized disclosure of <Create user> using <User Manager Website> by <Functioneel Beheer>".
- Threat Details:** The main right pane shows the configuration for the selected threat:
  - Name:** Unauthorized disclosure of <Create user> using <User Manager Website> by <Functioneel Beheer>
  - Description:** This is an automatically generated threat affecting the <Create user> action supporting the <Create user> usecase. This use case is used to realize the following net data effect:  
Functioneel Beheer < C > User information
  - Details of the action:** Caller: <Functioneel Beheer> <delegating>
  - Call:** Functioneel Beheer Create user User Manager Website
  - Primary Threat Factors:**
    - Unauthorized disclosure of the identity
    - Unauthorized disclosure of the data
  - Risk Measure:** Impact: High, Probability: High, Risk Rating: 9
  - Risk Response:** Response: Reduce
  - Attack Countermeasure:**
    - Forceful Browsing : Implement strong authorization controls
    - HTTP Replay Attack : Implement Secure end-to-end communication
    - LDAP Injection : Perform input validation
    - One-Click Attack : Make every request unique
    - Password Brute Force : Enforce password complexity
- Quick Help:** A pane at the bottom left providing context and examples:
  - The primary threat factors for Confidentiality are the unauthorized disclosure of the executing identity and the unauthorized disclosure of the data.
  - Example 1: Think of voting. The data submitted on ballotxyz is not confidential in itself, it will be viewed and counted by the ballot counters. Nevertheless, it is important to protect the identity of the voter who filled out ballotxyz

# Activity 4: Design reviews

- Manually inspect the design (documentation)
  - Verification of requirements and threat model in design
  - Inspect how security mechanisms are implemented
    - Are J2EE standards being used?
  - Inspect transaction management, error handling etc.
  - Verification against secure design principles
    - Principle of least privilege
    - Define consistent error-handling strategy
    - Fail safely
    - Multiple layers of defense
- No tools available
  - Use checklists

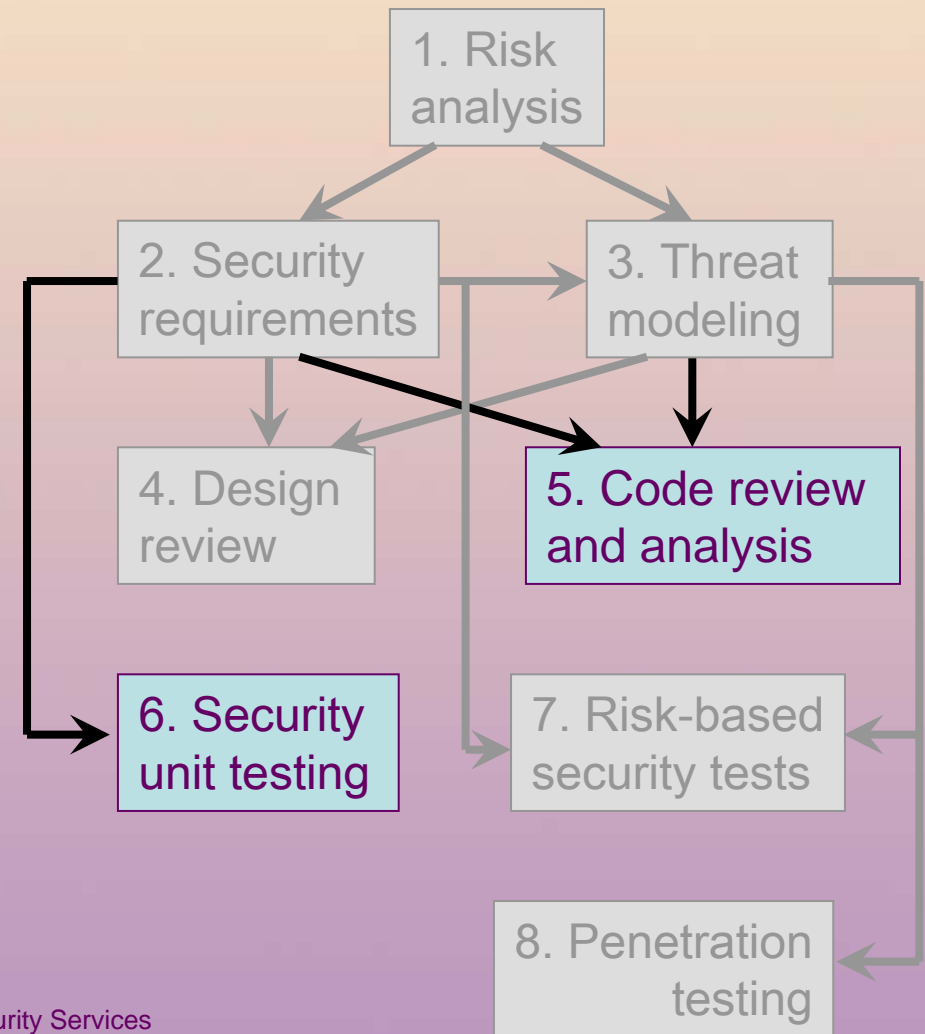


# Design best practices

- Start the design with a threat modeling to identify highest risks
  - Not only identify threats, but also rate them
- Make sure the design is reviewed
  - Apply secure design principles
- Use J2EE security or other known framework

# Software Development Process

- Requirements
- Design
- **Coding**
  - **Code review and analysis**
  - **Security unit testing**
- Test
- Deployment



# Activity 5: Code review and analysis

- Static code analysis
  - Include in automated build
  - Tools: Fortify, CodeAssure, FindBugs, PMD
- Manual code reviews:
  - CIA business requirements
  - OWASP checklist
  - Language specific checklists
  - Industry specific requirements



# Activity 6: Security unit testing

- Testing the smallest unit of an application
  - Similar to normal unit testing, but focus on unhappy flow
  - Testing all possible paths for security breaches
  - Take threat model and misuse cases as input
  - Strive to 100% code coverage in vulnerable components
- Type of vulnerabilities that can be tested
  - Input validation
  - Special characters
  - Exception handling
  - Security components
- Tools
  - JUnit and JCoverage



# Activity 6: Security unit testing

- Example

```
public class AccountValidatorTest extends TestCase {
    ...
    public void testIllegalCharactersInName() {
        account.setName("'");
        validator.validate(account, e);
        assertTrue("' did not cause errors.", e.hasFieldErrors("name"));
        account.setName(";");
        validator.validate(account, e);
        assertTrue("; did not cause errors.", e.hasFieldErrors("name"));
        account.setName("<");
        validator.validate(account, e);
        assertTrue("< did not cause errors.", e.hasFieldErrors("name"));
        account.setName(">");
        validator.validate(account, e);
        assertTrue("> did not cause errors.", e.hasFieldErrors("name"));
    }
    ...
}
```



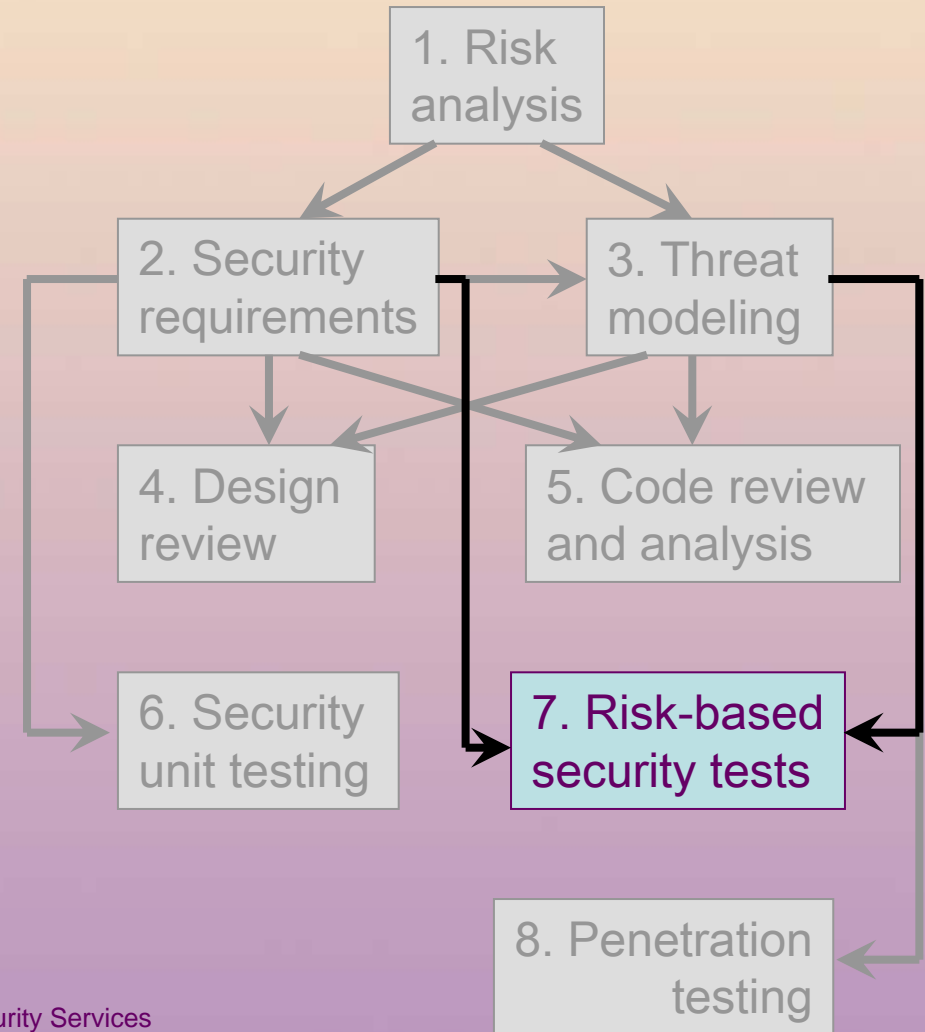
# Code best practices

- Setup code reviews and static code analysis
  - Validate all user input against positive specs.
  - Encode dangerous output to back-end and browser.
  - Never reveal too much information in error messages.
- Make use of existing proven standards and libraries
  - J2EE Container Security.
  - JAAS, JCE, JSSE



# Software Development Process

- Requirements
- Design
- Coding
- **Test**
  - **Security testing**
- Deployment



# Activity 7: Security Testing

- Security Testing is not the same as functional testing
  - Functional security testing
  - Adversarial security testing
  - Performance testing
- Develop security test cases based on
  - Security requirements
  - Misuse case models
  - Threat models
- Automate security tests
  - E.g. JMeter, Cactus, jWebUnit, WATIR
- Flaws feed into defect tracking and root cause analysis

# Activity 7: Security Testing

- Example: Testing HTML injection with jWebUnit

```
public class HTMLInjectionTest extends WebTestCase {  
    ...  
    public void testHTMLInjection() throws Exception {  
        beginAt("/index.html");  
        assertLinkPresentWithText("Enter the Store");  
        clickLinkWithText("Enter the Store");  
        assertFormPresent("searchForm");  
        setFormElement("query", "<a id=\"injection\"  
            href=\"http://www.google.com>Injection</a>");  
        submit();  
  
        // If link present, injection succeeded  
        assertLinkNotPresent("injection");  
    }  
    ...  
}
```



# Activity 7: Security Testing

- Example: test XSS with WATIR

```
def test_XSS_In_Search
  $ie.goto('http://localhost:8080/ispatula/shop/index.do')
  $ie.text_field(:name, 'query').set(
    '<script>window.open
    ("http://localhost:8080/ispatula/help.html")
    </script>')
  $ie.form(:action, /Search.do/).submit
  assert_raises(
    Watir::Exception::NoMatchingWindowFound,
    "Search field is susceptible to XSS") {
    ie2 = Watir::IE.attach(:url,
      "http://localhost:8080/ispatula/help.html")
  }
end
```

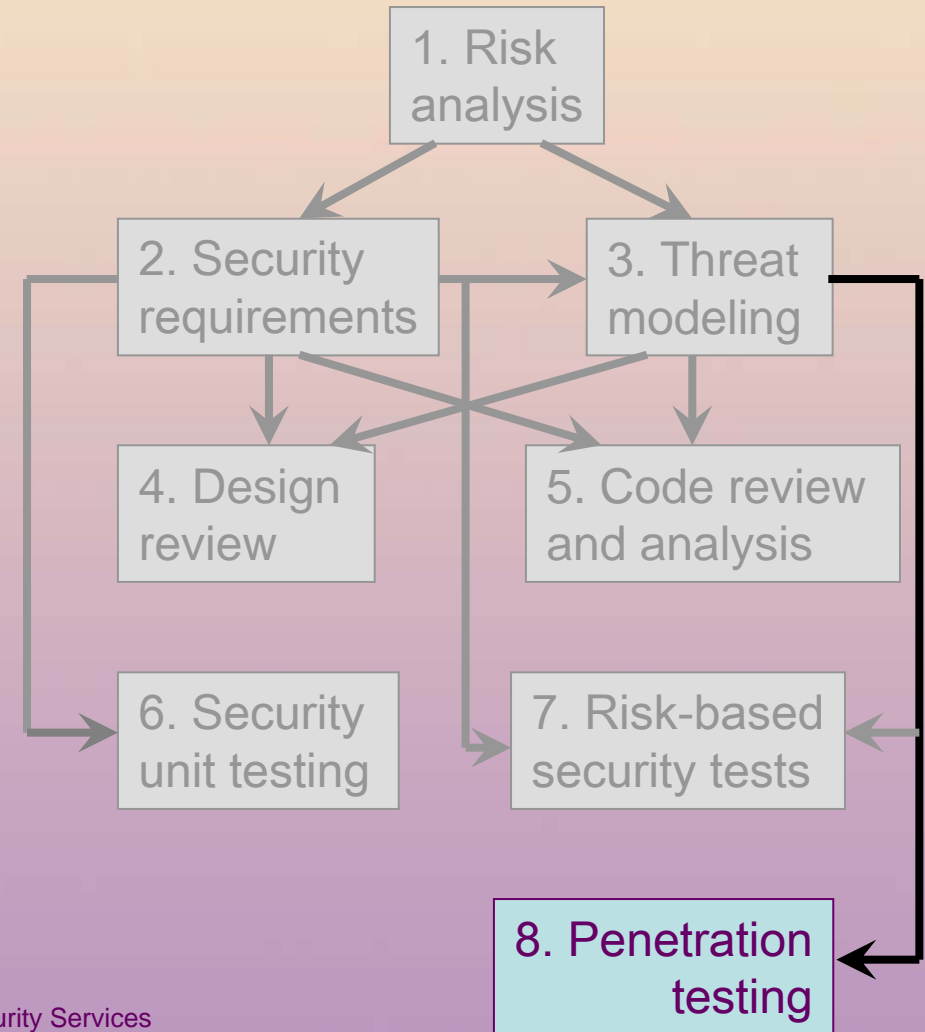
# Test best practices

- Think like an attacker.
  - Submit a variety of invalid data
  - Delete or deny access to files or registry entries
  - Test with an account that is not an administrator account
- Always measure how application performs under load.
- Use threat models to develop security testing strategy
- Automate attacks with scripts and low-level programming languages



# Software Development Process

- Requirements
- Design
- Coding
- Test
- **Deployment**
  - **Penetration testing**



# Activity 8: Penetration testing

- Use to discover configuration and environment problems
  - Can easily be fixed at the end
- Standards are the prerequisite to metrics
  - OWASP
  - OSSTMM
  - OASIS WAS
  - NIST, PCI, others
- Tools
  - WebInspect, AppScan



# Deployment best practices

- Perform penetration tests periodically
  - Tests changes in configuration and environment
  - Tests new vulnerabilities
- Harden your application server.
  - Checklists are provided by major vendors
  - Run application with limited privileges.
  - Automate deployment and hardening process.



# Conclusions

- J2EE security is not enough to secure your applications
- Security should be an integral part of the software development process
  - Risk Analysis
  - Security requirements
  - Threat modeling
  - Design reviews
  - Code review and analysis
  - Security unit testing
  - Security testing
  - Penetration testing



# How to introduce secure application development?

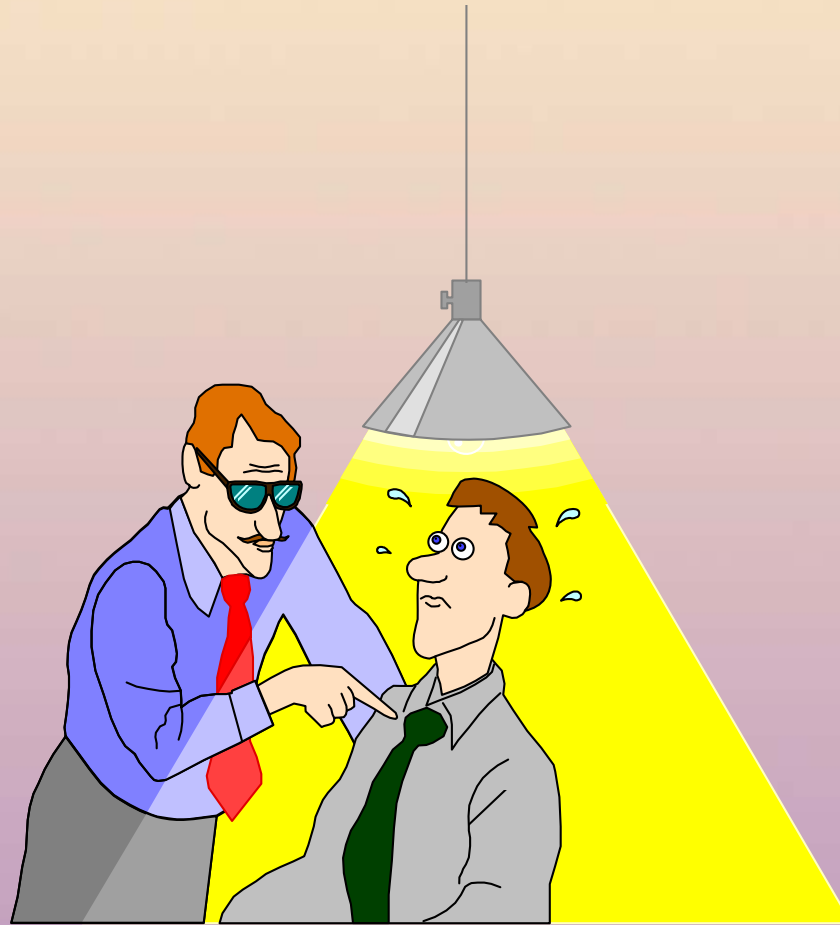
- Investigate current security development process
  - Incrementally add missing security activities starting with highest impact
    - Code and design review against possible vulnerabilities
    - Penetration tests
    - Threat modeling and risk analysis
    - Security requirements and security testing
    - Security unit testing
- Create security awareness and training program for developers

# Build applications more like cars...

- Security as afterthought
- Security as integral part



# Questions



# References

- Capturing Security Requirements through Misuse Cases – Sindre and Opdahl
- jWebUnit – <http://jwebunit.sourceforge.net>
- WATIR – <http://wtr.rubyforge.org>
- OSSTMM – <http://www.osstmm.org>
- OWASP – <http://www.owasp.org>