

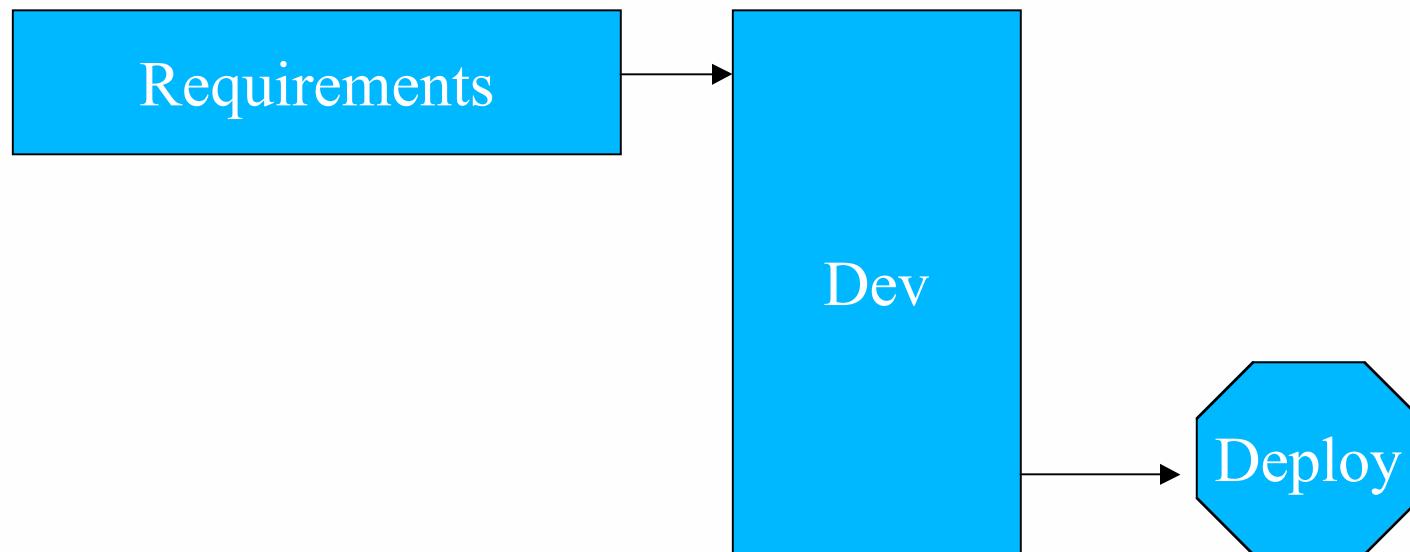
# Towards Application Security Design | Process | Organization

- ***Software Development Process***
- Security Design Process & Artifacts
- Security Team composition
- Future directions



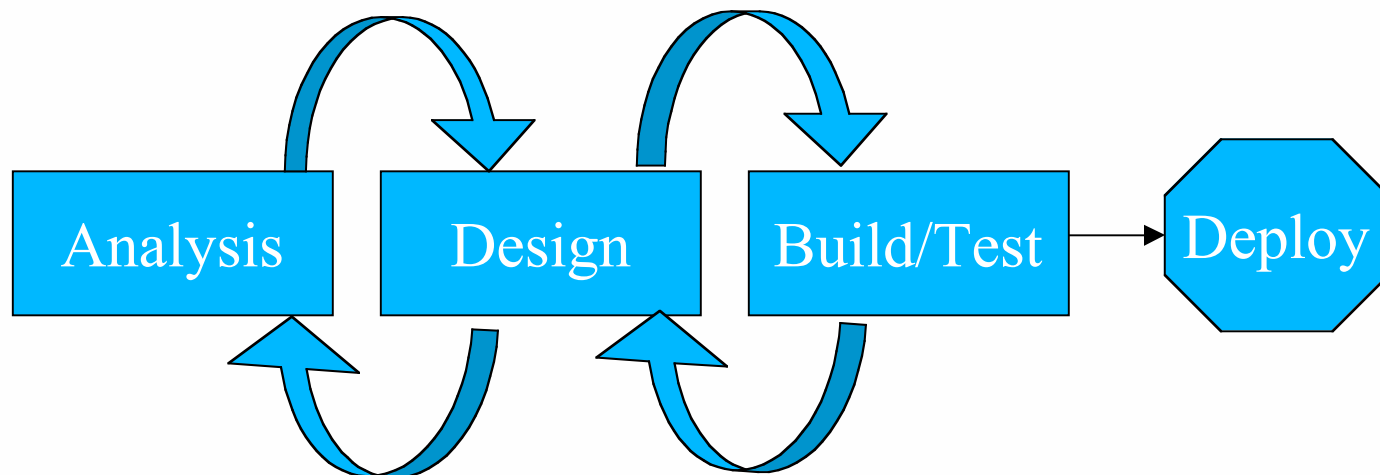
# Software Development Process Evolution

– *Over the waterfall*



# Software Development Process Evolution

- *Iterative development*
  - *Design/prototype hard problems first*



# Common Process Roles

- *Development Processes vary in role definition, but usually include these:*
  - *Stakeholders*
  - *Architect*
  - *Business Analyst/SME*
  - *Developer*
  - *QA*



# Security Team Composition

- Security-specific Team Roles
  - Security Architect: responsible for overall design
  - Security Analyst: responsible for requirements and Misuse Cases
  - Unit Hacker: Unit Hack, suites, and remediation guidelines
  - Application Security Configurator



# What are we developing?

- *What is an enterprise application?*
- *Anatomy of an enterprise application*
  - *Characterized by unique problem and solution sets*
  - *Hybrid of custom code and COTS*
  - *Sharing of data, infrastructure, and services*
  - *Client and server support issues*
  - *Connectivity creates layered and conflicting trust models*



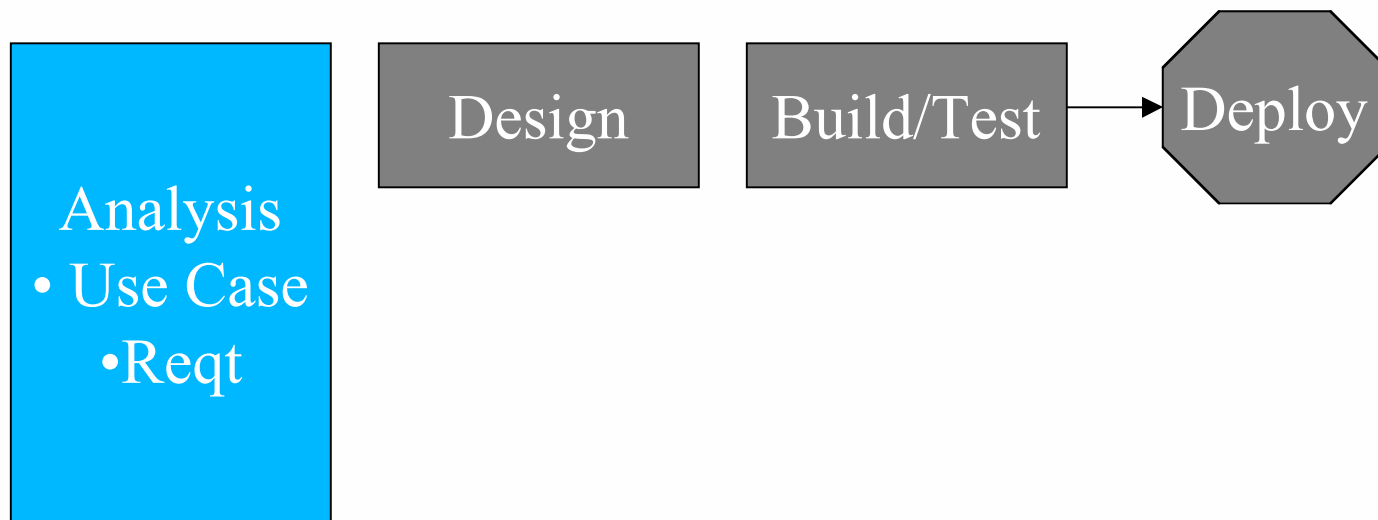
# Security Design Process/Roles

- ***Security specific analysis & design activities***
- ***Holistic cross-domain focus***
- ***Security-centric artifacts which integrate with development process***
- ***Understand when to harvest/adapt & when to roll your own***



# Analysis Phase

*"A problem, properly stated, is a problem on its way to being solved,"* Buckminster Fuller





# Use Case

- A specific way to capture requirements using actors and actions to show structure and relationships
- Defines both text document and diagram formats
- In Unified Process, Use Cases drive the development process
- Use Case Tools



# Use Case

- How do Use Cases benefit security analysis and design?
  - Breaking down the problem space
  - Contextual relationships
  - System Boundaries
  - Pre & Post conditions
  - Actors/Roles
  - Administrator Use Cases



# Example

- Data classes
  - Public
  - Private
  - Confidential
  - Site Config
- Use Cases
  - Browse Bookstore
  - Login
  - Edit User
  - Purchase Books
  - Admin Site



# Bringing it all together

Use Use Cases requirements, Roles, and Data classification to build access matrix

	Public	Private	Confdl	Site Config
Browse	R			
Login	R	R	R	
Edit User			U, D	
Purchase Books	R	C, R, U	R	
Admin Site				R, U, D



# Mis-Use Cases

- Look at the system from an attacker point of view
- Useful to glean security requirements, create threat models & Unit Hacks
- Discussed in paper by Guttorm Sindre and Andreas Opdahl.
  - More information at:  
[www.ifi.uib.no/conf/refsq2001/papers/p25.pdf](http://www.ifi.uib.no/conf/refsq2001/papers/p25.pdf)



# Mis-Use Cases Elements

- “A *misuse case* is the inverse of a use case, i.e. A function that the system should not allow” -Sindre & Opdahl
- “A *mis-actor* is the inverse of an actor, i.e., an actor that one does not want the system to support, an actor who initiates misuse cases.” -Sindre & Opdahl
- Additional elements
  - Worst Case Threat: end system state if Misuse succeeds
  - Prevention and Detection Guarantees: these guarantees closely resemble a Use Case Post-condition, but encapsulate security-specific concepts of prevention and detection.
  - Stakeholders and Risks: this field gives the security team a place to address what the business risk that is generated by the application.



# Data Model & Classification

- Classifying data assists in security design
- Classification considerations
  - Value: business value of data based on risk assessment
  - Confidentiality
  - Regulatory/industry/legal considerations
  - Roles



# Security Glossary

- Demystify key security terms for development team
- Roll your own or use an industry or vendor standard:
  - **SANS**  
<http://www.sans.org/resources/glossary.php>
  - **MS** <http://www.microsoft.com/security/glossary>





# Security Definitions

- **SANS**

- **Authentication:** *Authentication is the process of confirming the correctness of the claimed identity.*
- **Authorization:** *Authorization is the approval, permission, or empowerment for someone or something to do something.*

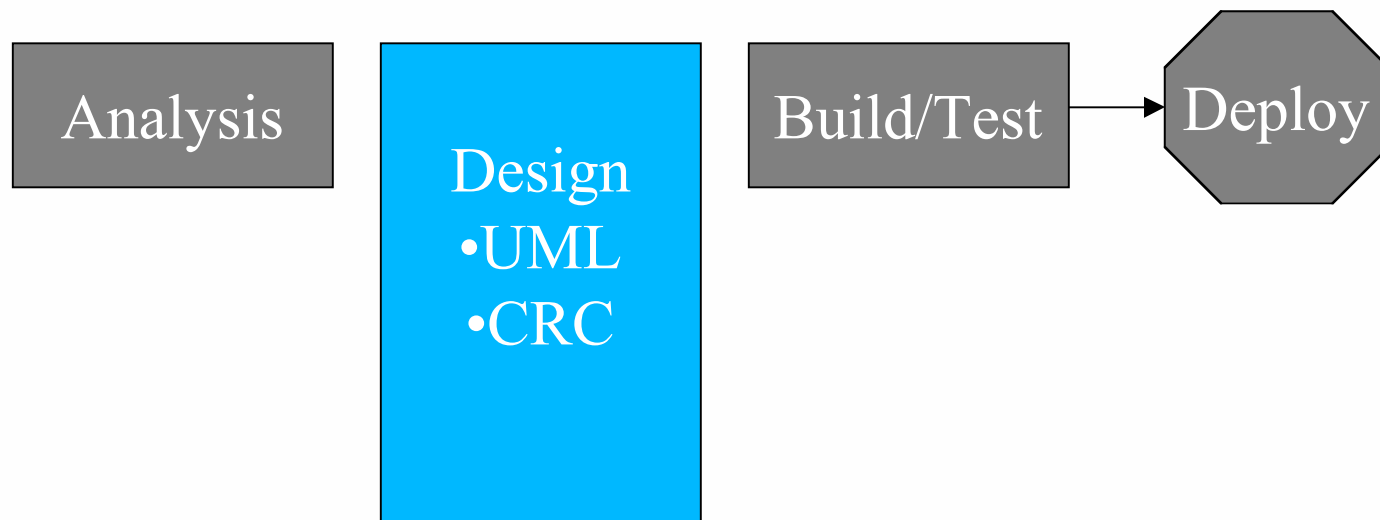
- **MS**

- **authentication (n.)** *The process for verifying that someone or something is who or what it claims to be. In private and public computer networks (including the Internet), authentication is commonly performed through the use of logon password*
- **authorization (n.)** *reference to computing, especially remote computers on a network, the right granted an individual or process to use a system and the data stored on it. Authorization is typically set up by a system administrator and verified by the computer based on some form of user identification, such as a code number or password.*



# Design Phase

Drilling down into design



# Design Phase Participation

Drilling down into design

Tradeoff analysis

– Architectural Options

- Language/frameworks choice
- Design Patterns & Pattern Languages
- Buy/build blend

– Business Value

- Functionality v. Security

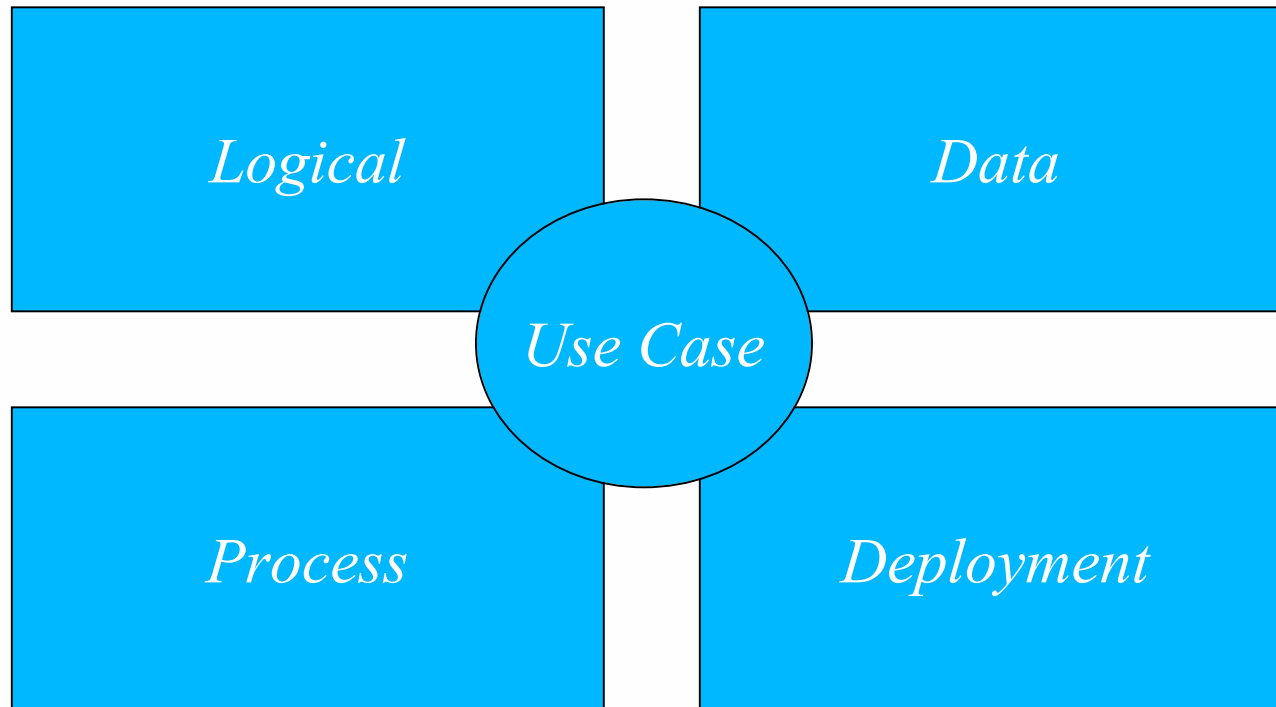
– Usability

- Fundamental conflicts (see upcoming ISB)
- Ignorance/Arrogance paradigm



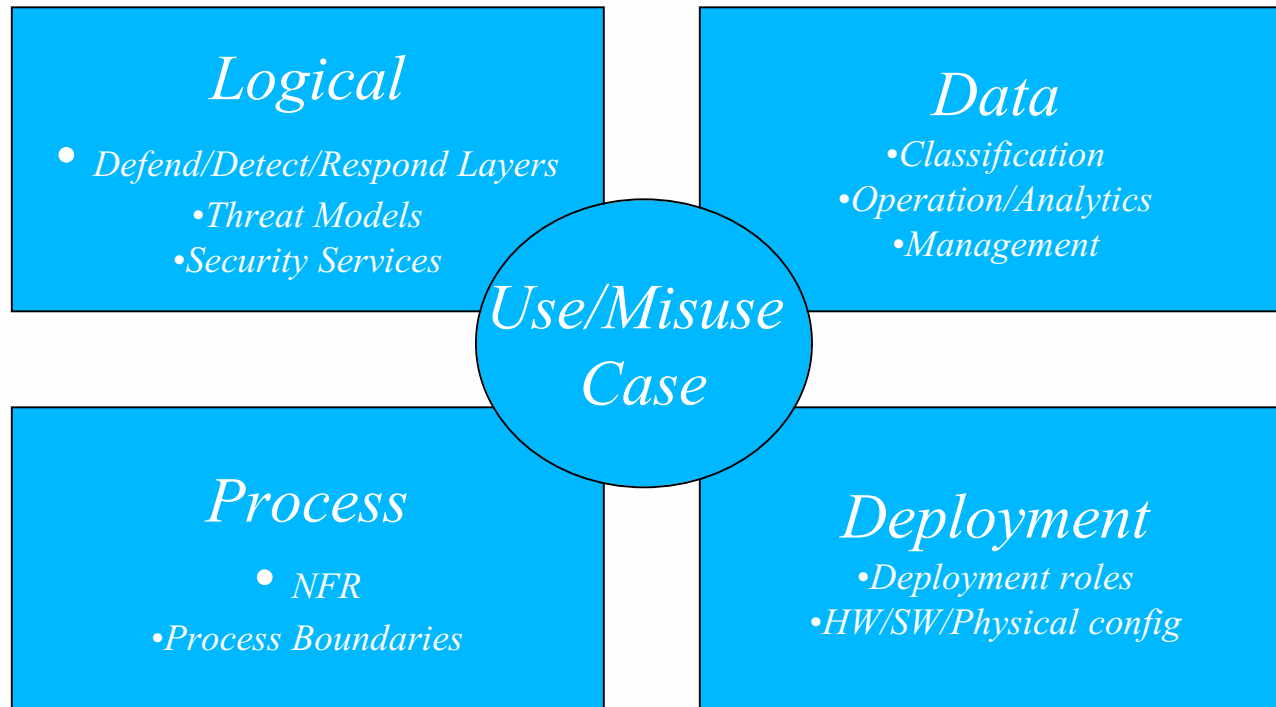
# Building Architecture

4 + 1 Architectural layers



# Building Security Architecture

Architectural layers for holistic security view



# Threat Modeling

- Elaborates on threats in MisUse case analysis
- Focus on distilling:
  - Threat impact level
  - Threat likelihood
  - Mitigation, management, and containment



# Threat Models

- Howard and Leblanc's STRIDE and DREAD
- Identification
  - STRIDE
    - **S**poofing, **T**ampering, **R**epudiation, **I**nformation disclosure, **D**enial of service, and **E**levation of privilege
- Prioritization
  - DREAD
    - **D**amage potential, **R**eproducibility, **E**xploitability, **A**ffected users, and **D**iscoverability



# Construction Phase

- Concerned with building, integrating, and testing code
- Iteration
- Use Unit Test tools like Nunit ([www.nunit.org](http://www.nunit.org)) to validate your design assumptions





# Build and Unit Test Process

- Separation of privileges
  - Developer Level
    - Compile
    - Unit test
  - Integration Level
    - Build
    - Configure
    - Deploy
    - Promote



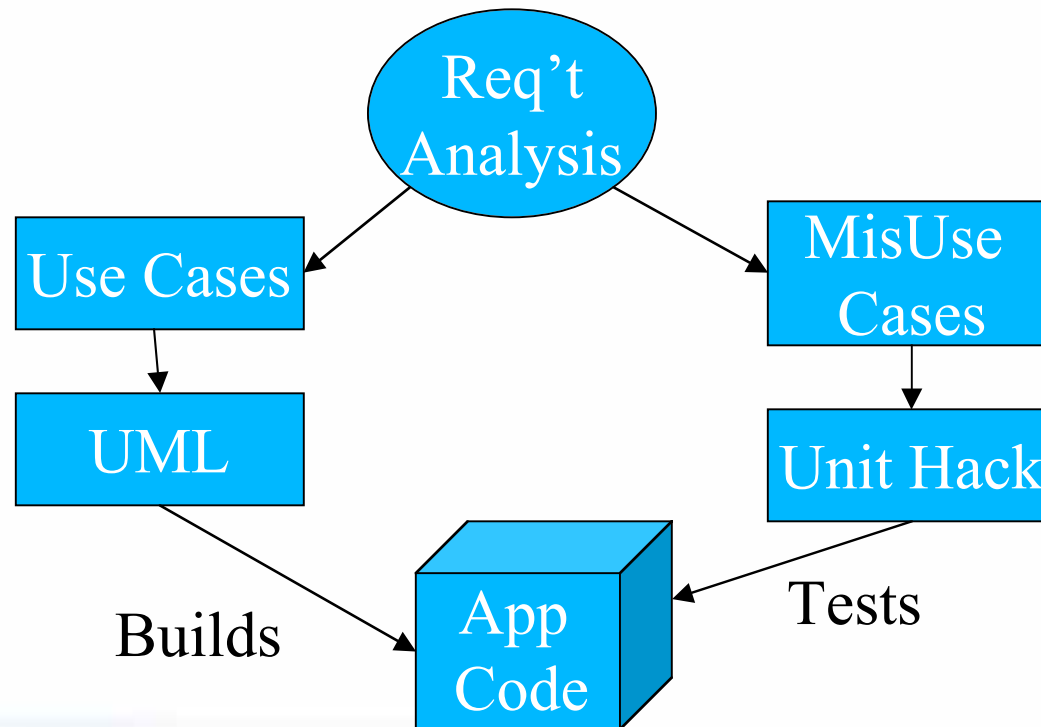
# Unit Hacking

- Unit Hacking Fundamentals
  - Relies on assertion condition (boolean)
    - `Assertion.AssertEquals(foo, bar);`
    - `Assertion.AssertNotNull(foo);`
  - Use same tools as Unit Testing (nunit)
    - <http://sourceforge.net/projects/nunitaddin/>
  - Unit Hacking Mindset
  - Combine domain specific threats with industry best practices
  - Create same benefits for security team as Unit Tests do for BA and Developers



# Unit Hacking

- Where do Unit Hacks fit?



# Unit Hacking in Practice

- Issues with Unit Hacking
- Unit Hack Suites
  - Aim for reuse
  - Identify Hack suites for common patterns such as web apps
  - Provide remediation guidelines where possible
  - Identify attack signatures



# Transition Phase

- Where “security” traditionally begins
- Operational planning
- Monitoring processes
- Incident response planning
- Locking down deployment/config process



# Future Directions

- Continuing evolution of development processes and tools
- Outsource implications
- Aspect Oriented Programming tools for security programming
- More proactive security team involvement in development activities
- Use Cases : <http://alistair.cockburn.us>



# Questions?

- More information and free, monthly architecture newsletter at:  
[www.arctecgroup.net/views.htm](http://www.arctecgroup.net/views.htm)

