Wentworth INSTITUTE OF TECHNOLOGY

Secure Software Development

Aspen Olmsted, Ph.D.
Associate Professor

Wentworth INSTITUTE OF TECHNOLOGY | SCHOOL OF COMPUTING & DATA SCIENCE

About Aspen

- Work in Software Development for 35+ years
- Designed and Launched NYU Cyber Fellows Program
- Designed and Launched NYU MicroBachelors Programs
- Managed Certification of Several Universities' CAE programs
- Associate Professor at Wentworth Institute of Technology
- Author of Security-Driven Software Development
- Adjunct Professor in NYU Cyber Fellows
- Expert Witness for Software Development Cases

https://www.linkedin.com/in/aspeno/

Agenda

- Software Security
- Software Development Lifecycles
- Functional Model
- Object Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Software Security

- Set of development practices that protect:
  - Software itself
  - The data processed by the software
  - The network communications
- Not just Malicious Users

Agenda

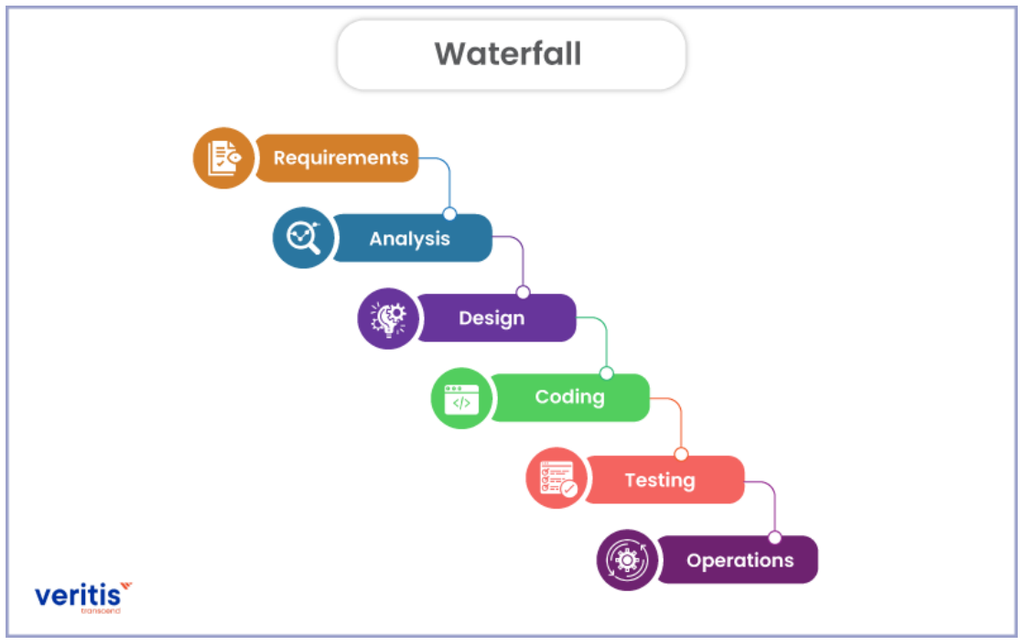
- Security
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Software Development Lifecycles

- There are many SDLCs used to develop software
- We will think about four SDLCs
  - Waterfall
  - Agile
  - DevOps
  - Microsoft Security Development Lifecycle

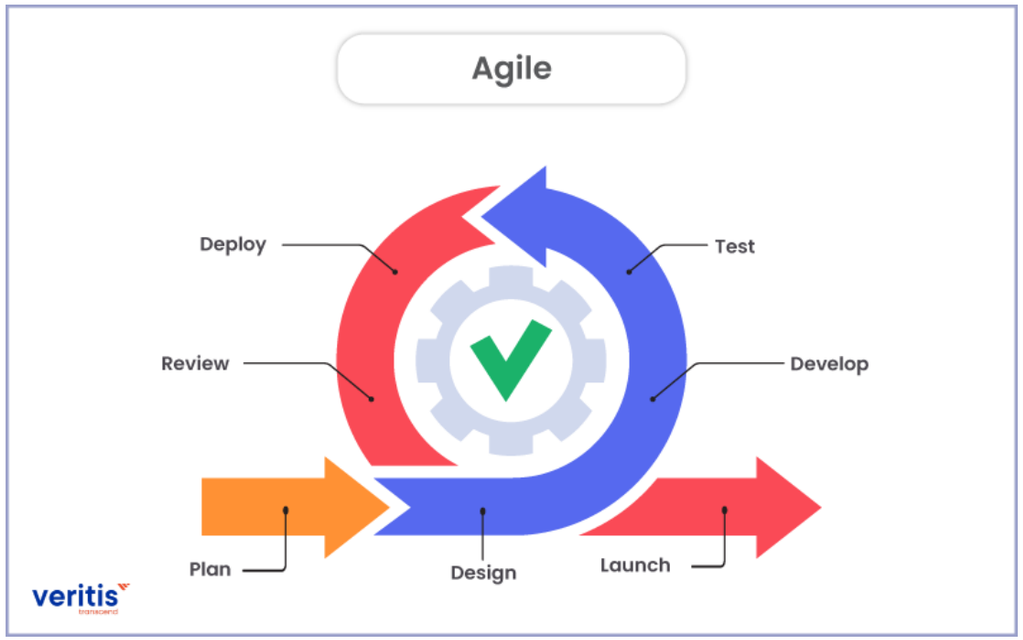
Waterfall

- Well Defined Steps
- Problems may not be discovered until late in the process
- Finished Product



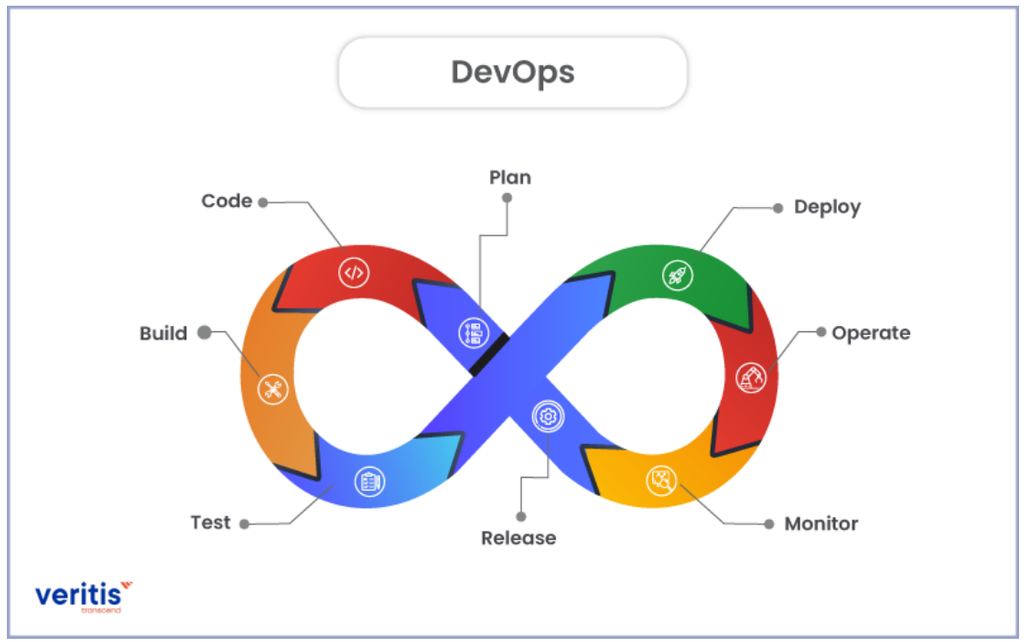
Agile

- No concept of finished product
- One model uses Scrum
- Work put into sprints



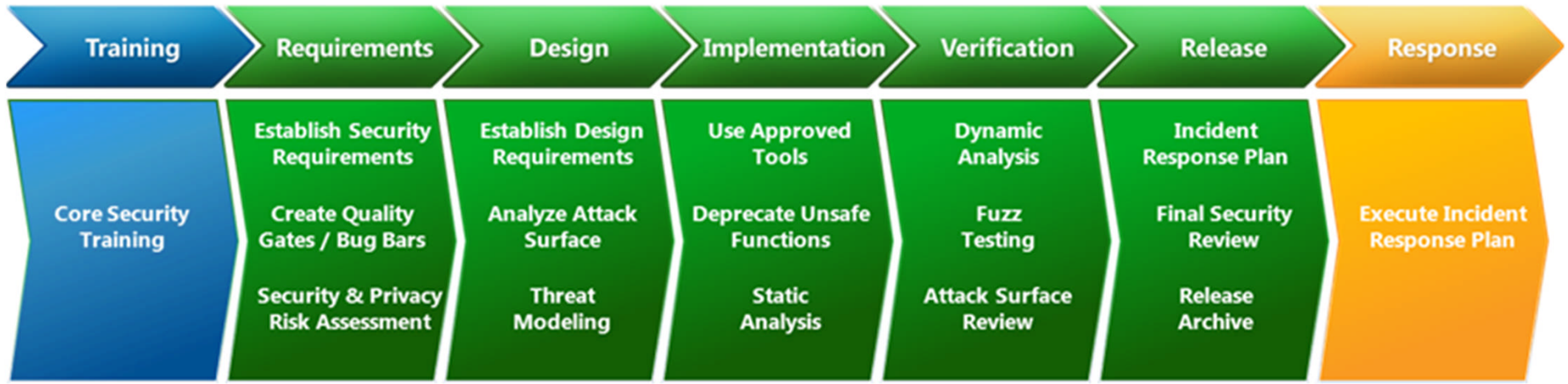
DevOps

- Incremental updates
- Automation of phases



Microsoft Security Development Lifecycle

- Security Steps Added to Phases
- Training and Response



Our Methodology

- More Security Modeling in Distinct Phases
- Closer to the Code
- Automated Code Injection for Mitigation

Our Methodology

Phase	Security Models/Tools
Functional Model	Non-Functional Requirements/Misuse Scenarios/Cases
Object Model	OCL Constraints/Stereo Types
Dynamic/System Model	OCL Pre & Post Constrains/Stereo Types/Patterns
Threat Model	STRIDE/DREAD/PERT Models
Implementation	Training on Know Web Security
Verification	Unit/Integration/System Tests
Penetration Testing	Automated System Scans

Agenda

- Security
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Functional Model

- A model is a simplification
- Output is
  - Functional Requirements
  - Non-Functional Requirements
  - Constraints
- Tools
  - List
  - Textual Scenarios
  - Textual Use-Cases
  - Graphical Use-Cases
  - Textual Mis-Scenarios
  - Textual Misuse Cases
  - Graphical MisUse-Cases

Example Requirements for Event Ticketing Application

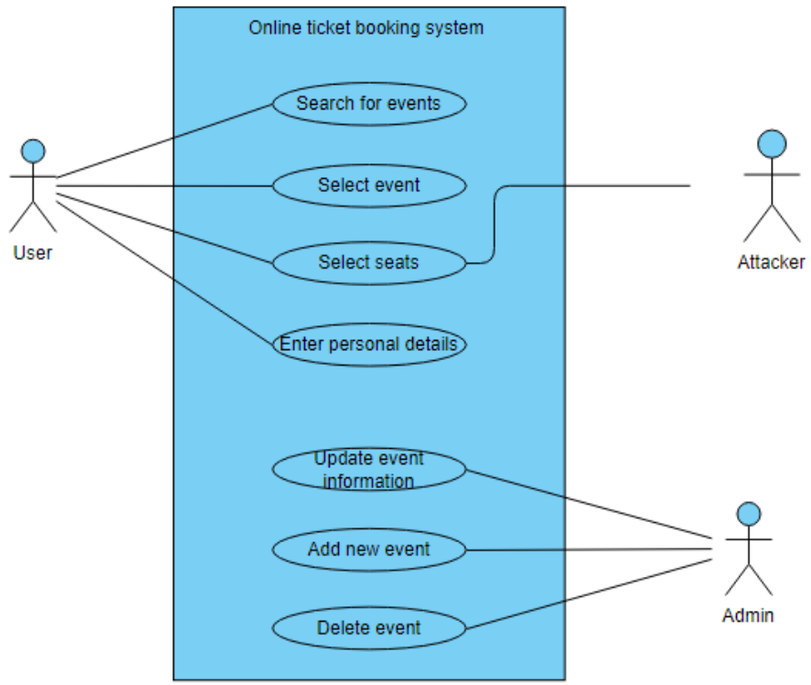
Functional Requirements	Non-Functional Requirements	Constraints
Must allow self service purchases	Must support 50,000 concurrent users	Patron should be able to use an Android phone
Must allow basket of multiple events	Must send e-tickets within 5 minutes of transaction completion	E-tickets must be in pdf format
Venue should be able to control maximum number of tickets per event	Return users must authenticate to reuse previous payment type	
Venue should be able to control available payment types		

Example Misuse Scenario

- Henrietta the Hacker creates several new emails to allow her to purchase more than the allowed tickets. Between each order she uses the browser incognito feature to not have any cookies from previous transaction

Misuse Cases

- Use and misuse cases are used to validate understanding
- Multiple scenarios are rolled up into generic textual use case and graphical use case models
- Multiple misuse scenarios are rolled up into generic textual misuse case and graphical misuse case models



Agenda

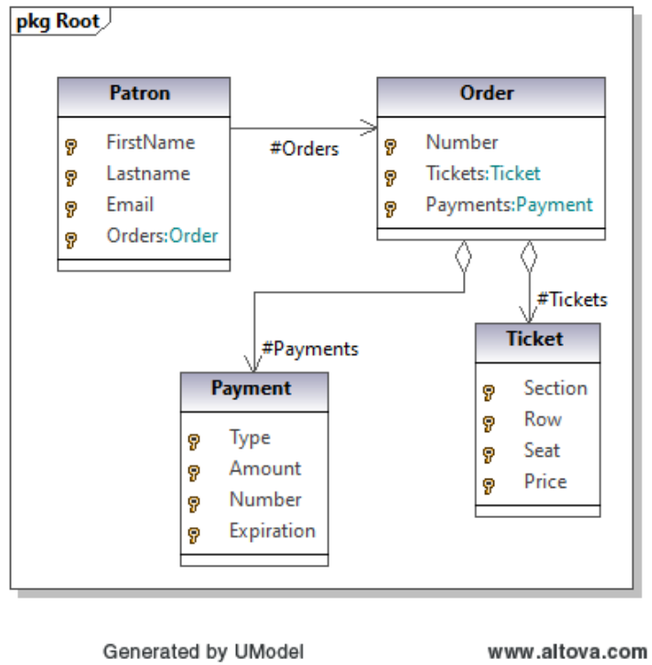
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Object Model

- Output is
  - Object Design
  - OCL Constraints
- Tools
  - UML Class Diagrams
  - OCL Constraints

UML Class Diagrams

Represents the internal structure of an application



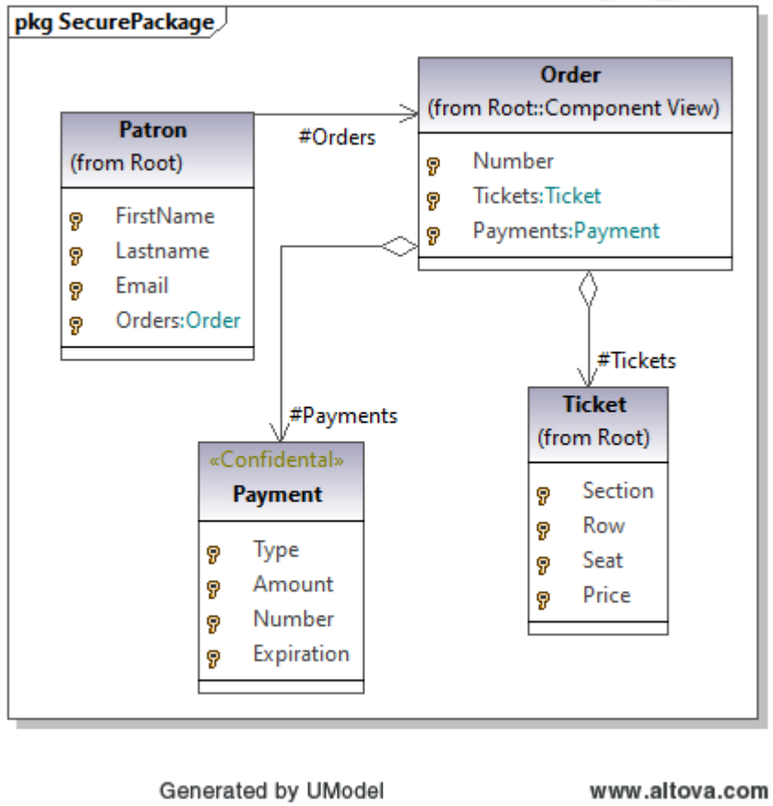
Object Constraint Language (OCL)

- Rule-based Language to Specify Correctness

Examples	
Constraint	OCL Equivalent
The age of a person is not negative.	context Person inv: self.age >=0
A person is younger than its parents.	context Person inv: self.parents->forAll(p|p.age>self.age)
After a birthday, a person becomes one year older.	context Person::hasBirthday() post: self.age=self.age@pre+1
A Person has 2 parents at max.	context Person inv: self.parents->size()<=2
After somebody has a child, his/her child-set is not empty, and it is larger than before.	context Person::getsChild() post: self.childs->notEmpty() and self.childs->size() > self.childs@pre->size()
Only an adult can be owner of a car.	context Person inv: self.age<18 implies self.cars->isEmpty()
The first registration of a car can not be before it is built.	context Auto inv: self.registration>=self.constructionYear
Every Person that has a car has at least one car which is younger than the Person.	context Person inv: self.cars->notEmpty() implies self.cars->exists(c | Calendar.YEAR - c.constructionYear < self.age)
Nobody can be his/her own parent.	context Person inv: self.parents->excludes(self)
There's at least one Person which owns a car.	context Person inv: Person.allInstances()->exists(p | p.cars->size() > 0)

Stereotypes

- Add meaning to UML entities and attributes



Agenda

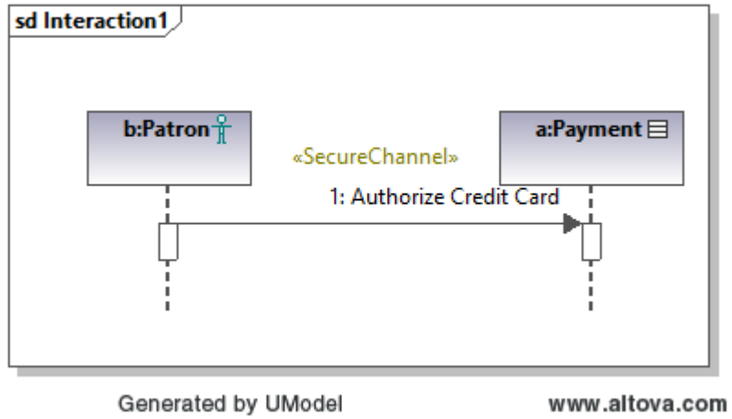
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Dynamic Model

- Output is
  - Methods
  - OCL Pre/Post Constraints
  - Synchronous vs Asynchronous Messages
- Tools
  - UML State Charts
  - UML Communication Diagrams
  - UML Sequence Diagrams

Sequence Diagrams

- Can use Stereotypes on Classes, Lifelines, Messages
- OCL Pre and Post Conditions on Messages
- Arrows represent Synchronous vs Asynchronous



Agenda

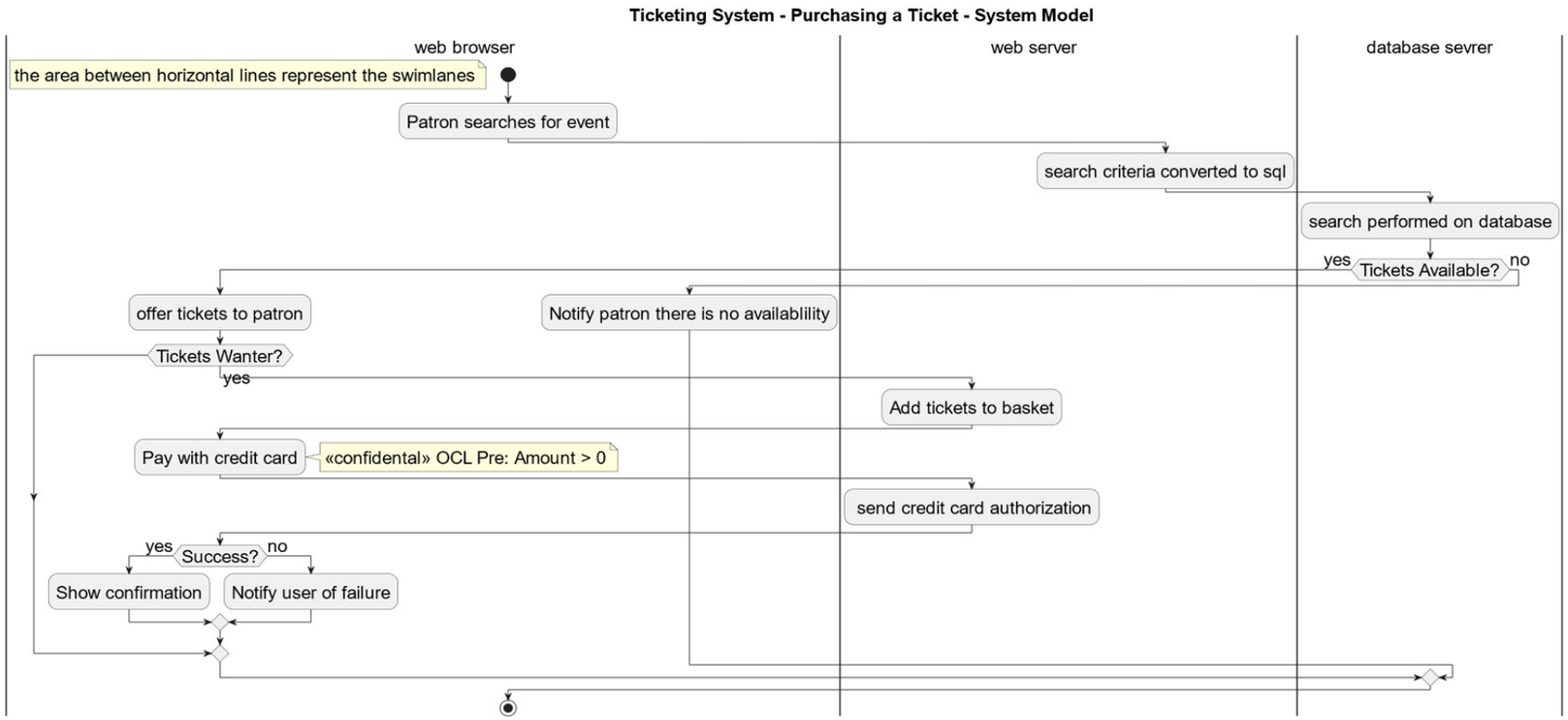
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

System Model

- Output is
  - System Partitions
  - Patterns
  - Pre/Post Constraints
- Tools
  - UML Sequence Diagrams
  - UML Activity Diagrams
  - OCL Constraints

Activity Diagram

- Can use Stereotypes on Activities, Messages
- OCL Pre and Post Conditions on Messages



Agenda

- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Threat	Desired Property
Spoofing	Authenticity
Tampering	Integrity
Repudiation	Non-repudiability
Information disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

The STRIDE model is an approach to threat modeling to identify potential vulnerabilities and threats

Example Stride Model

Function	S	T	R	I	D	E
Login	X	X	X	X	X	X
Event Selection					X	
Seat Selection					X	X
Payment	X	X	X		X	
Print at Home	X	X			X	

Other Models

- DREAD – Similar to STRIDE but uses quantitative value
  - Damage: Understand the potential damage a particular threat is capable of causing.
  - Reproducibility: Identify how easy it is to replicate an attack.
  - Exploitability: Analyze the system's vulnerabilities to ascertain susceptibility to cyberattacks.
  - Affected Users: Calculate how many users would be affected by a cyberattack.
  - Discoverability: Determine how easy it is to discover vulnerable points in the system infrastructure.
- PERT – Distributed System Model
  - Partition – Vulnerable to network partition failure
  - Execution – Vulnerable to execution failure
  - Requisite – Vulnerable to previous action failure
  - Time – Vulnerable to execution timing

Other Models

- CRIRTA – Threat Modeling for Systems for Database Systems
  - Reproducibility: Identify how easy it is to replicate an attack.
  - Exploitability: Analyze the system's vulnerabilities to ascertain susceptibility to cyberattacks.
  - Affected Users: Calculate how many users would be affected by a cyberattack.
  - Discoverability: Determine how easy it is to discover vulnerable points in the system infrastructure.
- BIRFS – Threat Modeling for Systems that utilize AI/ML Algorithms
  - B- potential biases in output
  - I - input is outside the domain of control.
  - R - output result does not deviate from a reasonable range
  - F - forensics or logging to defend results
  - S - Sensitive or private data needs to be protected

Agenda

- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Mitigation Strategies

- Some standard mitigation strategies
  - Logging
  - Redundancy
  - Authentication
  - Authorization
- Can be added as stereo types in earlier models
- Could can be generated from XMI or similar version of model

Agenda

- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Implementation

- Train on standard web vulnerabilities
  - OWASP TOP 10
  - SQL Injection
  - Command Injection
  - XSS
  - Request Forgery

Agenda

- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Test Types

- Unit Tests – Test classes, methods
- Integration Tests – Test subsystems with Mocks and Stubbs
- Regression Tests – Test non-functional requirements
- System Tests - Test functional requirements

Agenda

- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Penetration Testing

- Should be performed by separate team from developers
- Output – Report
- Tools
  - Open-source intelligence
  - Nikita – Open-Source scanner for known vulnerabilities
  - Vega – Open-Source web scanner that can run as proxy or scanner

Questions?