Secure Software Development for the Cloud

Aspen Olmsted, Ph.D.
Associate Professor

About Me

- Associate Professor in School of Computing and Science – Wentworth Institute of Technology
- 30+ years in the Software Industry – Sold software company to publicly traded company in 2005
- Expert Witness
- Created nearly a dozen technical programs for universities such as New York University, College of Charleston, Simmons University, Fisher College
- Created over 41 edx and Coursera courses
- Author of two technical books

Recent Book On Topic

‹packt›

1ST EDITION

Security-Driven Software Development

Learn to analyze and mitigate risks in your software projects

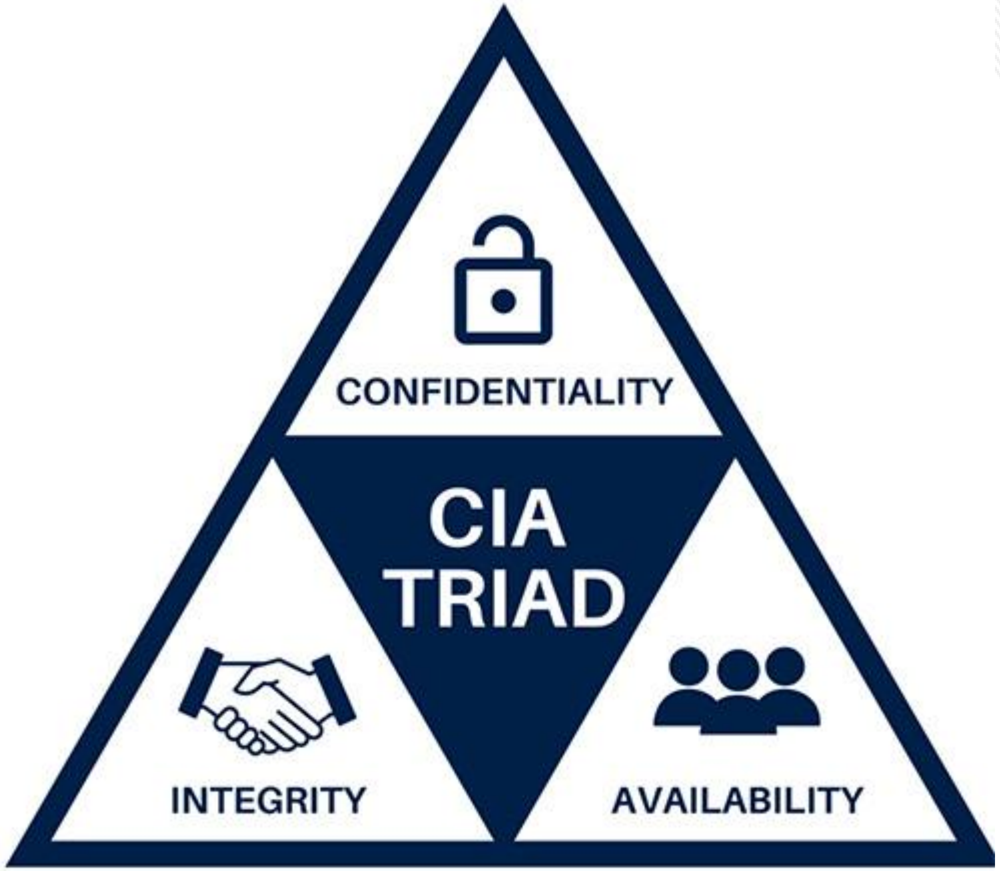
ASPEN OLMSTED

Agenda

- Software Security
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic 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

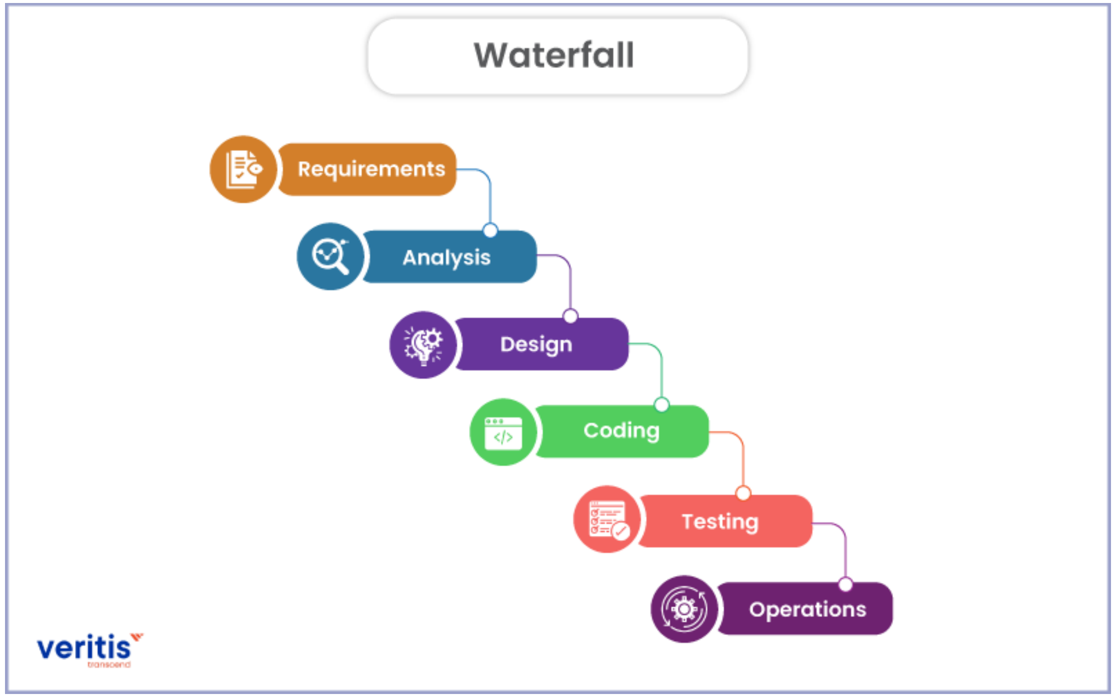
- Security
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Software Development Lifecycles

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

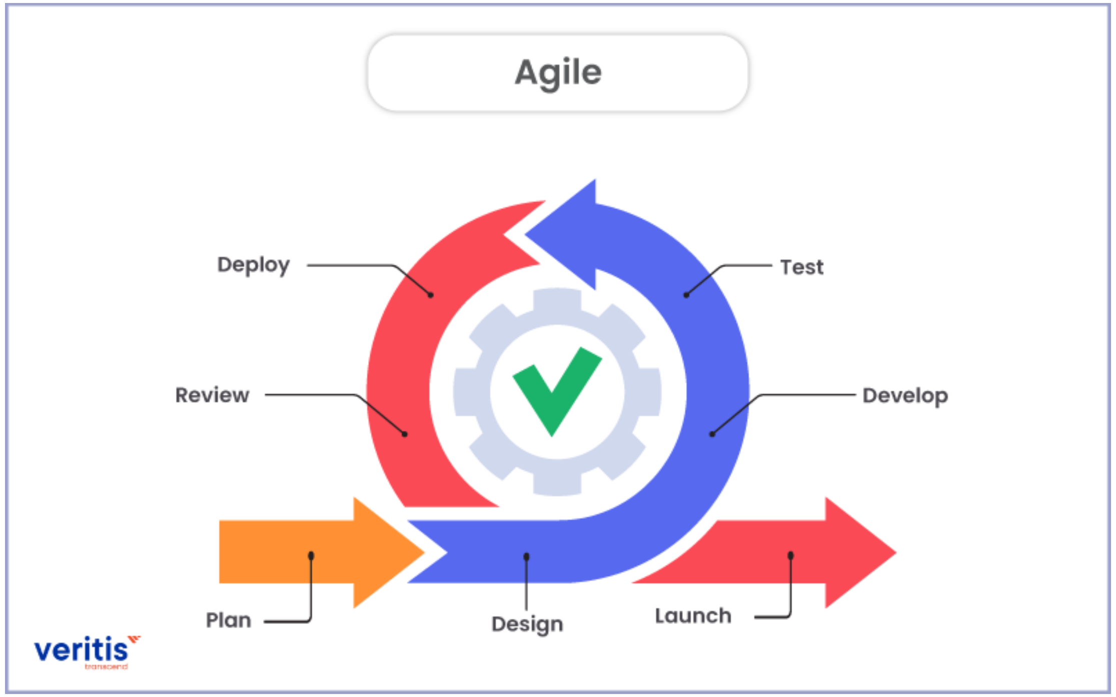
Waterfall

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



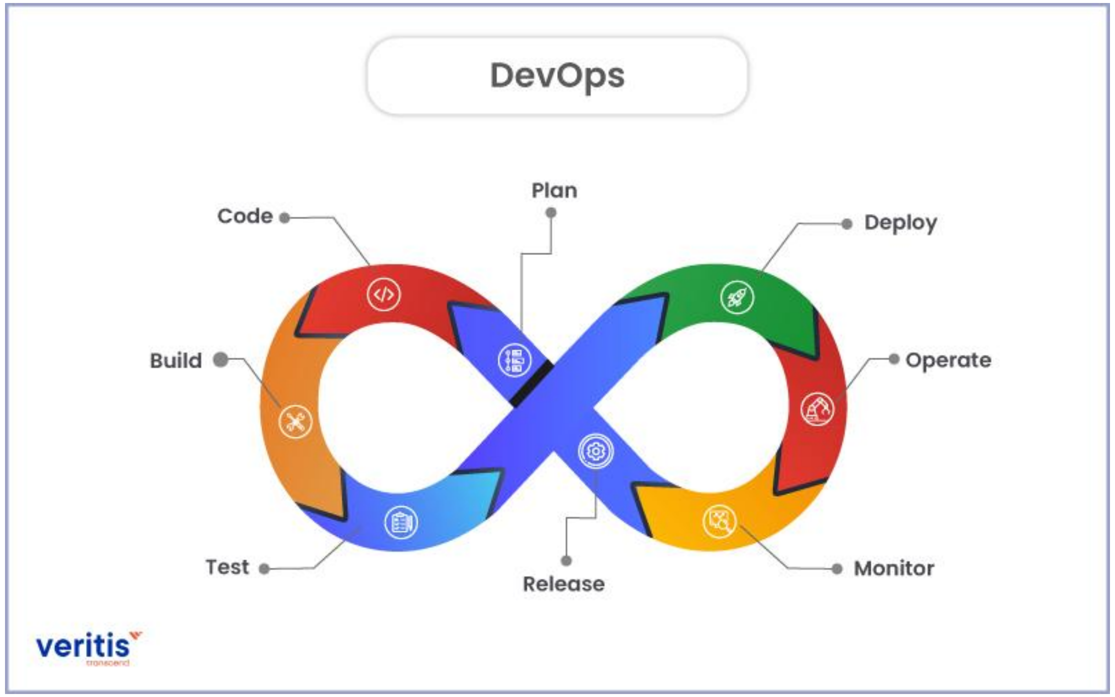
Agile

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



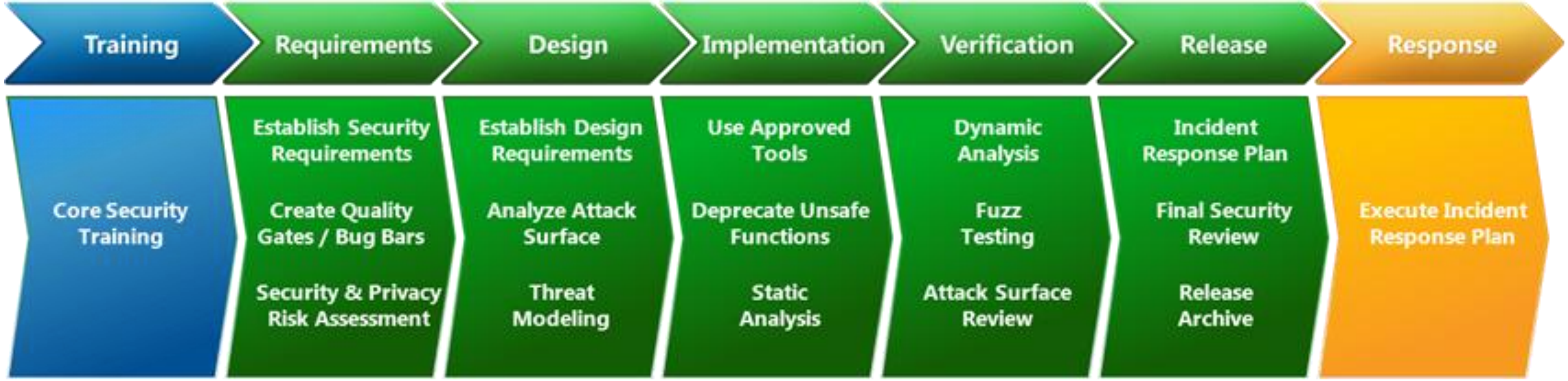
DevOps

- Incremental updates
- Automation of phases



Microsoft Security Development Lifecycle

- Security Steps Added to Phases
- Training and Response



Training	Requirements	Design	Implementation	Verification	Release	Response
Core Security Training	Establish Security Requirements Create Quality Gates / Bug Bars Security & Privacy Risk Assessment	Establish Design Requirements Analyze Attack Surface Threat Modeling	Use Approved Tools Deprecate Unsafe Functions Static Analysis	Dynamic Analysis Fuzz Testing Attack Surface Review	Incident Response Plan Final Security Review Release Archive	Execute Incident Response Plan

Our Methodology

- More Security Modeling in Distinct Phases
- Closer to the Code
- No Beginning and End – Each new feature/bug fix considers earlier models
- 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
System Model	OCL Constraints/Stereo Types
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

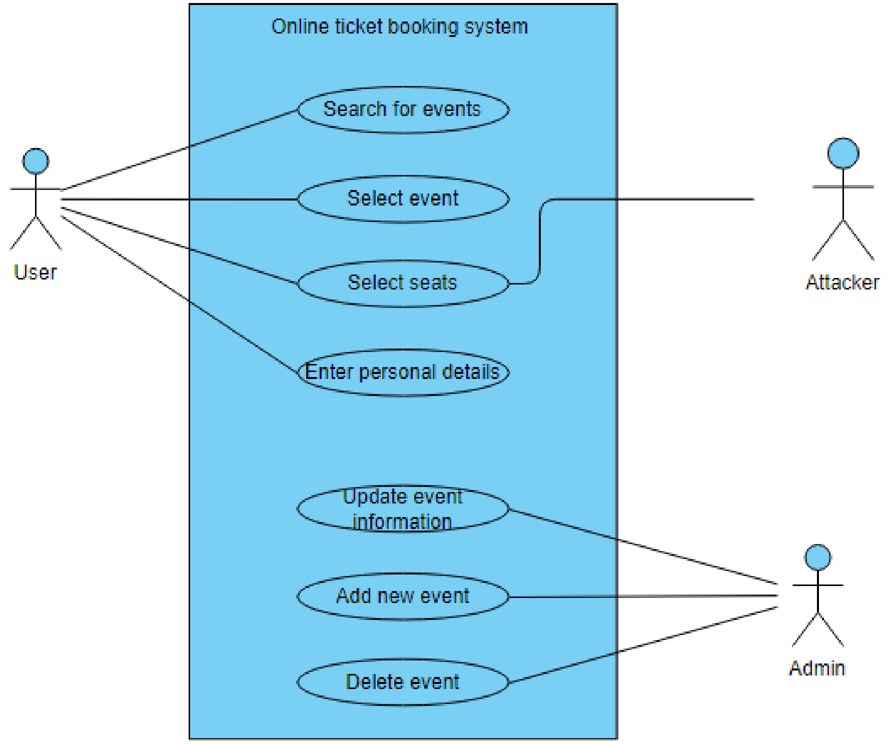
Functional Requirements	Non-Functional Requirements	Constraints
Must allow self service purchases	Must support 50,000 concurrent users	Patron should be able to use an Android phone
Must allow basket of multiple events	Must send e-tickets within 5 minutes of transaction completion	E-tickets must be in 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

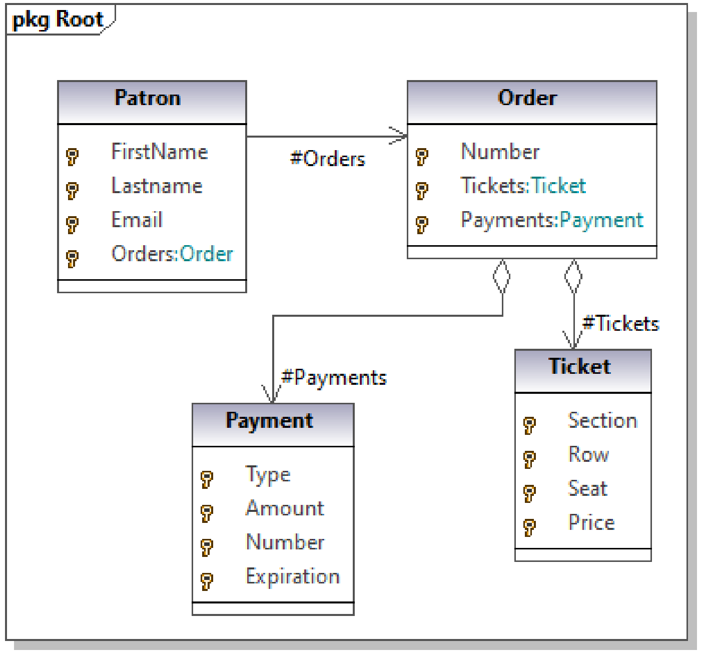
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Object Model

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

UML Class Diagrams

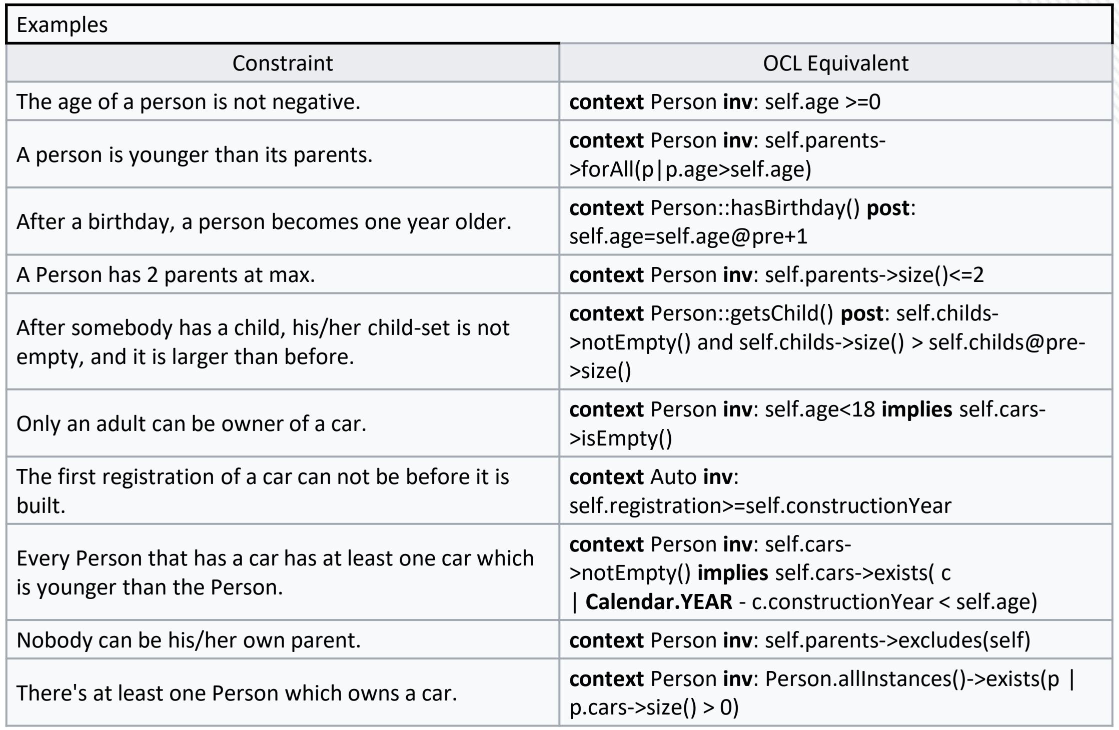
Represents the internal structure of an application



Generated by UModel www.altova.com

Object Constraint Language (OCL)

- Rule-based Language to Specify Correctness

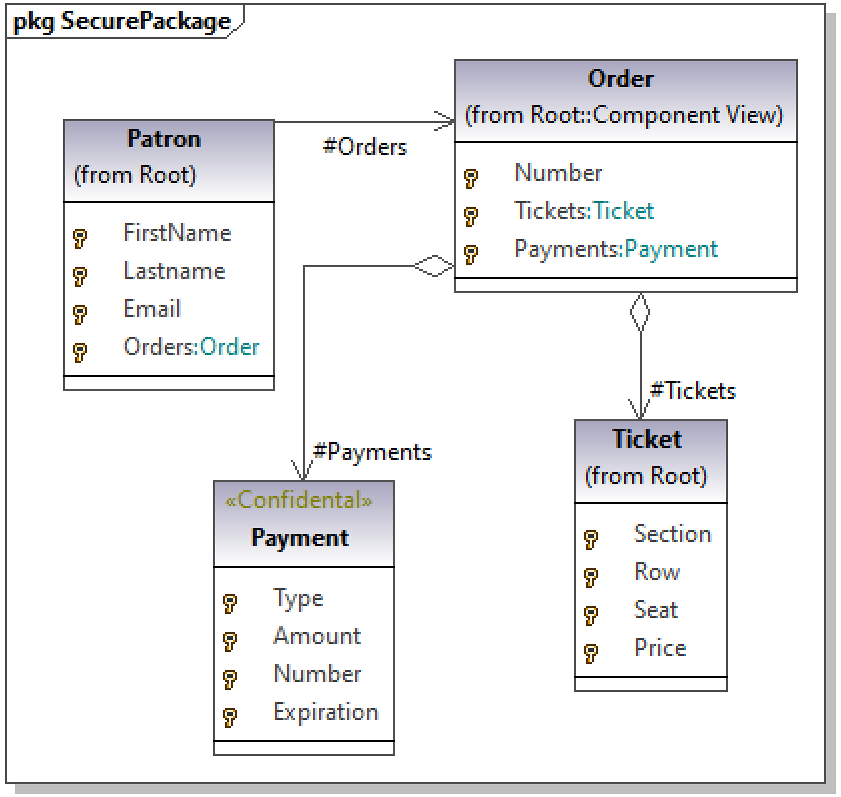


Examples

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



Generated by UModel www.altova.com

Agenda

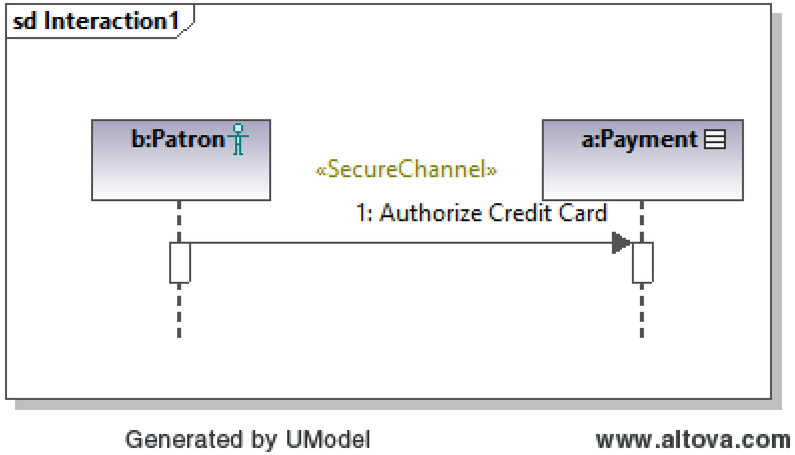
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

Dynamic Model

- Output is
  - Expanded Methods Understanding
  - OCL Constraints
- Tools
  - UML Communication Diagrams
  - UML State Diagrams
  - UML Sequence Diagrams
  - OCL Constraints

Dynamic Models

- Sequence or Communication
- Can use Stereotypes on Classes, Lifelines, Messages
- OCL Pre and Post Conditions on Messages



Agenda

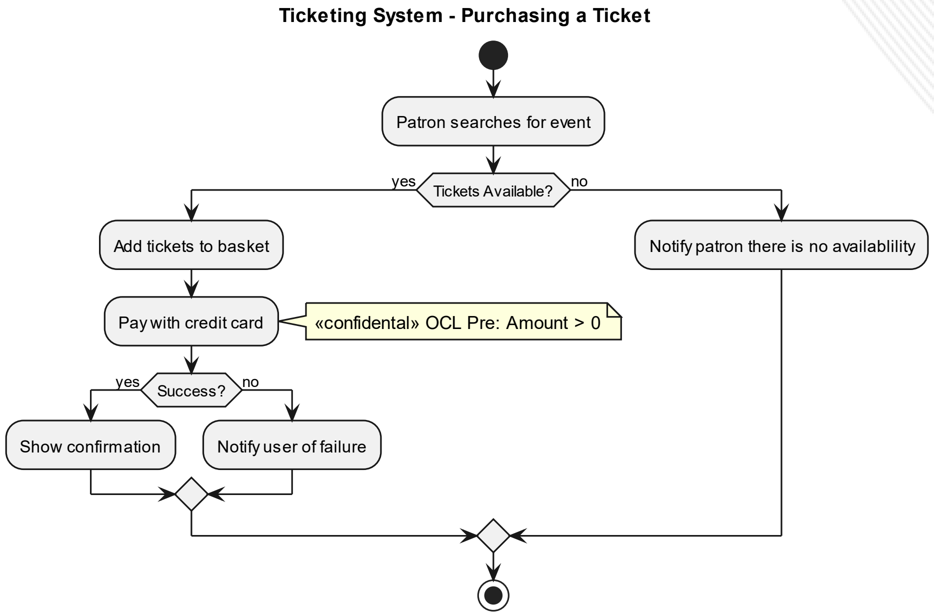
- Software Development Lifecycles
- Functional Model
- Object Model
- Dynamic Model
- System Model
- Threat Model
- Risk Mitigation
- Implementation
- Testing
- Penetration Testing

System Model

- Output is
  - System Partitions
  - Patterns
  - OCL Constraints
  - Stereotypes
  - Actions
  - Components
- Tools
  - UML Sequence Diagrams
  - UML Activity Diagrams
  - UML Component Diagrams
  - UML Deployment Diagrams
  - OCL Constraints

System Models

- Sequence or Activity
- Can use Stereotypes on Classes, Lifelines, Messages & Control Flow
- OCL Pre and Post Conditions on Messages & Control Flow


Ticketing System - Purchasing a Ticket
Patron searches for event
yes
Tickets Available?
no
Add tickets to basket
Notify patron there is no availablility
Pay with credit card
«confidental» OCL Pre: Amount > 0
yes
Success?
no
Show confirmation
Notify user of failure

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

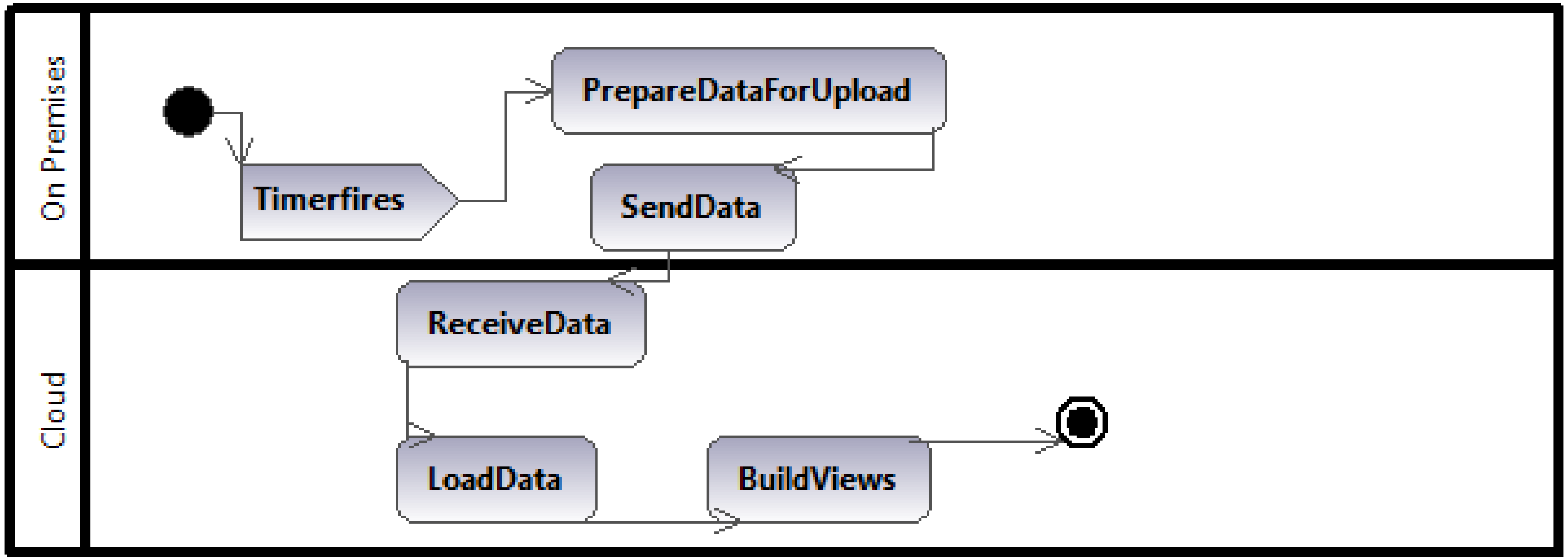
Example Stride Model

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.
- PERTD – 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
  - Data – Vulnerabilities in Data Sources

Daily ETL Upload from On-Premises to Cloud



Daily ETL Download from Cloud to On-Premises

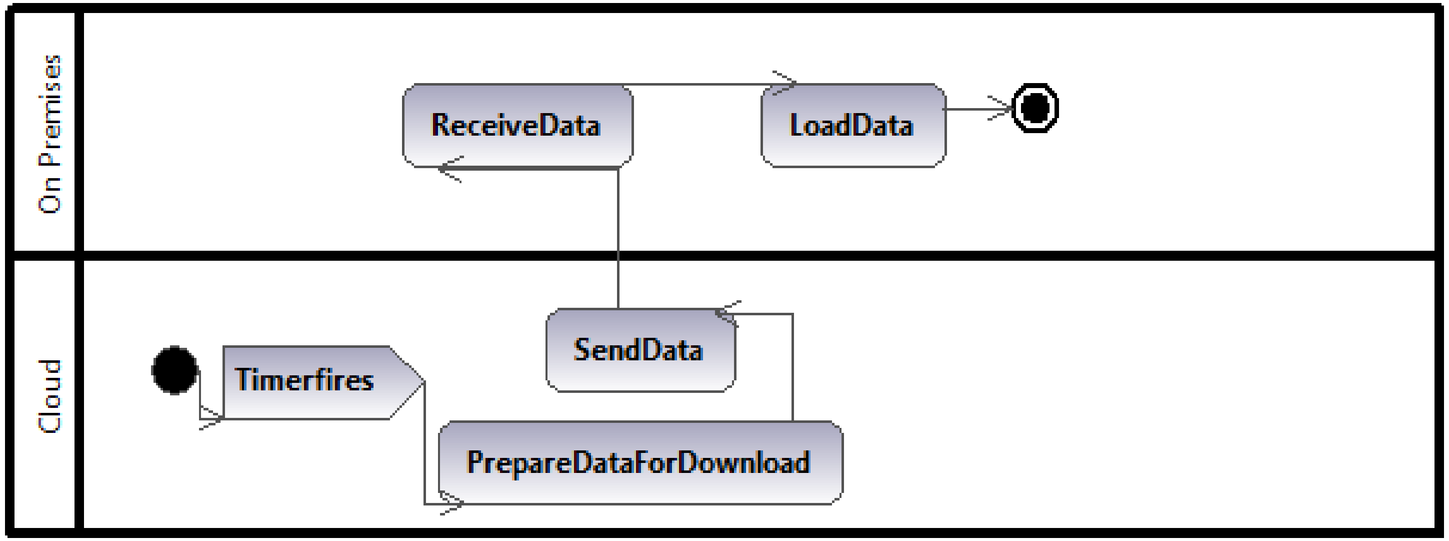


Figure 2 - Download Activity

STRIDE Model of Daily ETL Upload from On-Premises to Cloud

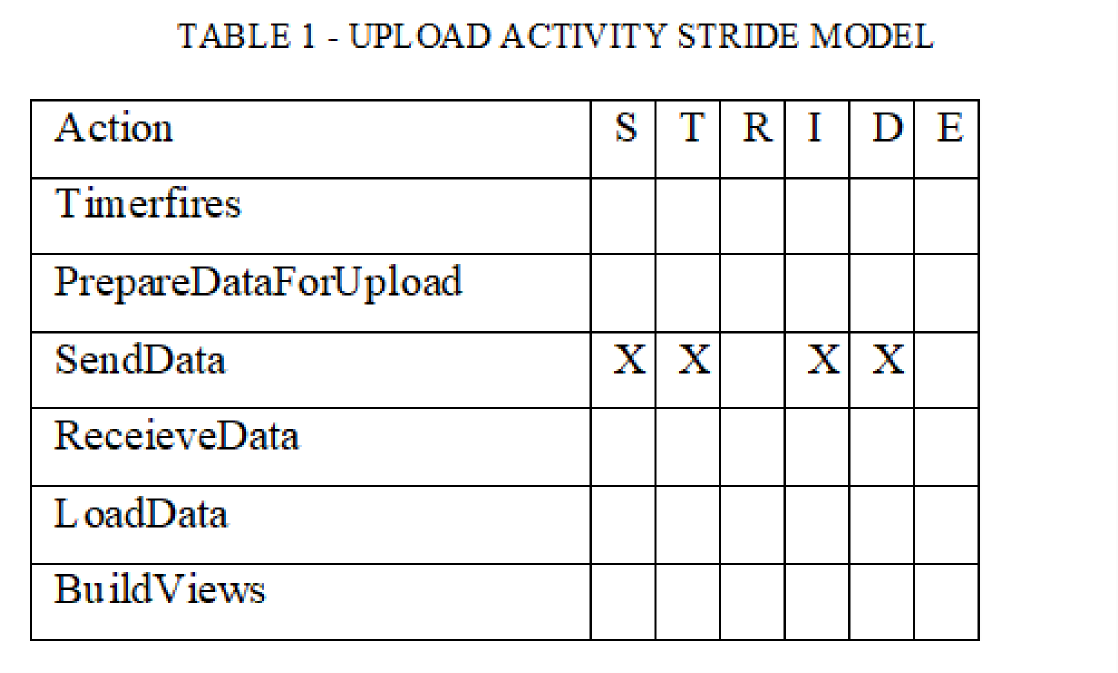


TABLE 1 - UPLOAD ACTIVITY STRIDE MODEL

Action	S	T	R	I	D	E
Timerfires						
PrepareDataForUpload						
SendData	X	X		X	X	
ReceieveData						
LoadData						
BuildViews						

STRIDE Model of Daily ETL Download from Cloud to On-Premises

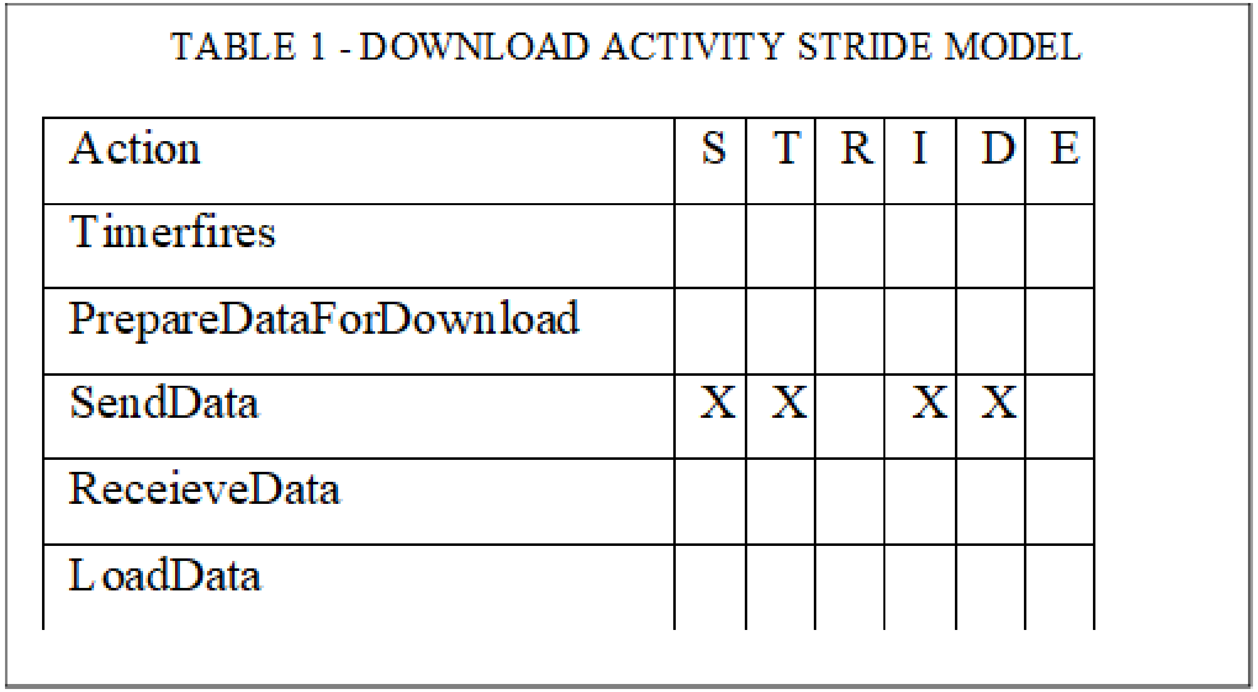


TABLE 1 - DOWNLOAD ACTIVITY STRIDE MODEL

Action	S	T	R	I	D	E
Timerfires						
PrepareDataForDownload						
SendData	X	X		X	X	
ReceieveData						
LoadData						

PERTD Model of Daily ETL Upload from On-Premises to Cloud

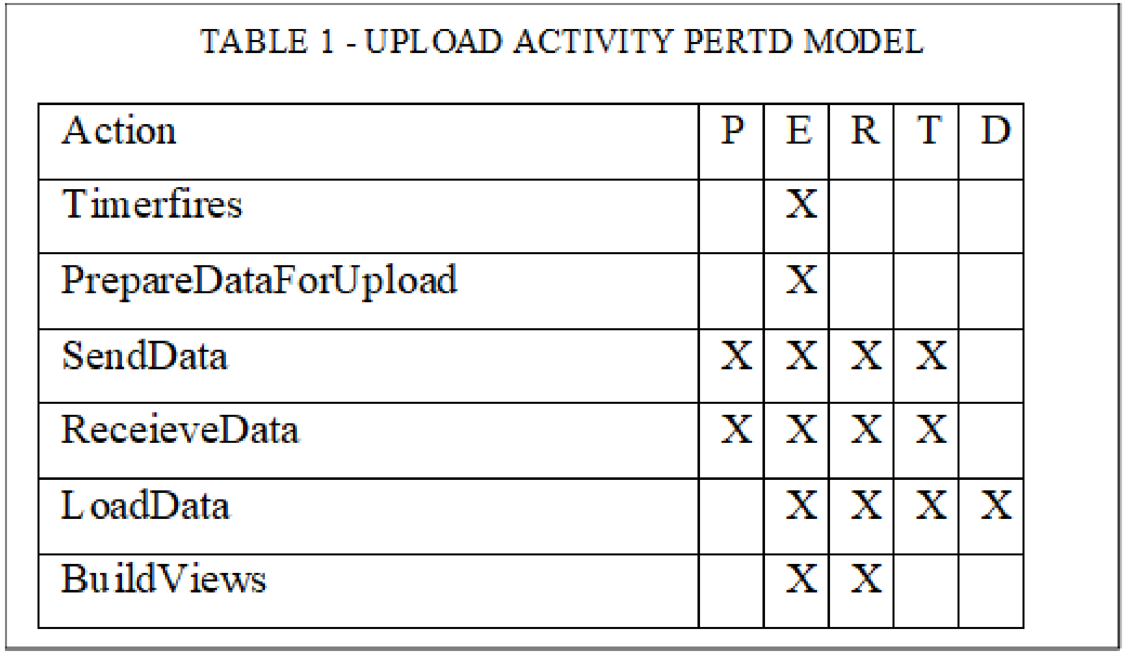


TABLE 1 - UPLOAD ACTIVITY PERTD MODEL

Action	P	E	R	T	D
Timerfires		X			
PrepareDataForUpload		X			
SendData	X	X	X	X	
ReceieveData	X	X	X	X	
LoadData		X	X	X	X
BuildViews		X	X		

PERTD Model of Daily ETL Download from Cloud to On-Premises

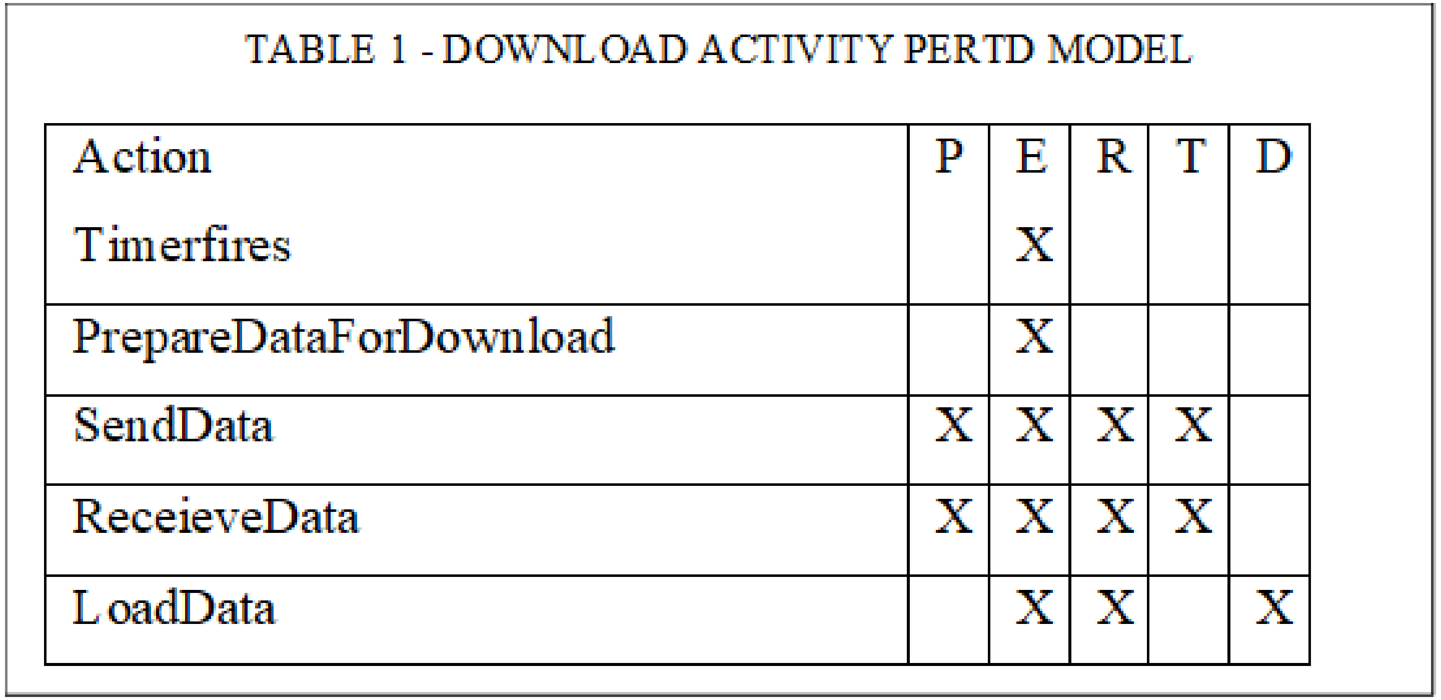


TABLE 1 - DOWNLOAD ACTIVITY PERTD MODEL

Action	P	E	R	T	D
Timerfires		X			
PrepareDataForDownload		X			
SendData	X	X	X	X	
ReceieveData	X	X	X	X	
LoadData		X	X		X

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

CRIRTA – Threat Modeling for Systems for Database Systems

- C - Column Confidentiality
- R - Row Confidentiality
- I - Column Inference
- R - Relationship Correctness
- T - Table Correctness
- A - Availability

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
  - Database Security
  - Standard Web Security
  - Buffer Overflows
- Can be added as stereotypes in earlier models
- Could be generated from XMI or a similar version of model

PERTD Mitigation Strategies

- PERTD – Distributed System Model
  - Partition – Vulnerable to network partition failure – Snapshots, Freshness prioritization
  - Execution – Vulnerable to execution failure - Snapshots , Freshness prioritization
  - Requisite – Vulnerable to previous action failure - Snapshots , Freshness prioritization
  - Time – Vulnerable to execution timing - Snapshots , Freshness prioritization
  - Data – Vulnerabilities in Data Sources – Multisource JSON Schema Validation Stage

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

Testing PERTD

- Regression Tests – Test non-functional requirements
  - PERTD – Distributed System Model
    - Partition – Vulnerable to network partition failure – Isolate hosts during execution
    - Execution – Vulnerable to execution failure – Pollute execution to force failure
    - Requisite – Vulnerable to previous action failure – Pollute requisite test
    - Time – Vulnerable to execution timing – Pollute to slow execution
    - Data – Vulnerabilities in Data Sources – Inject dirty records in some systems

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?