Secure Product Development

Agile Development Basics

Dr. Panayotis Kikiras
INFS133
April 2019

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

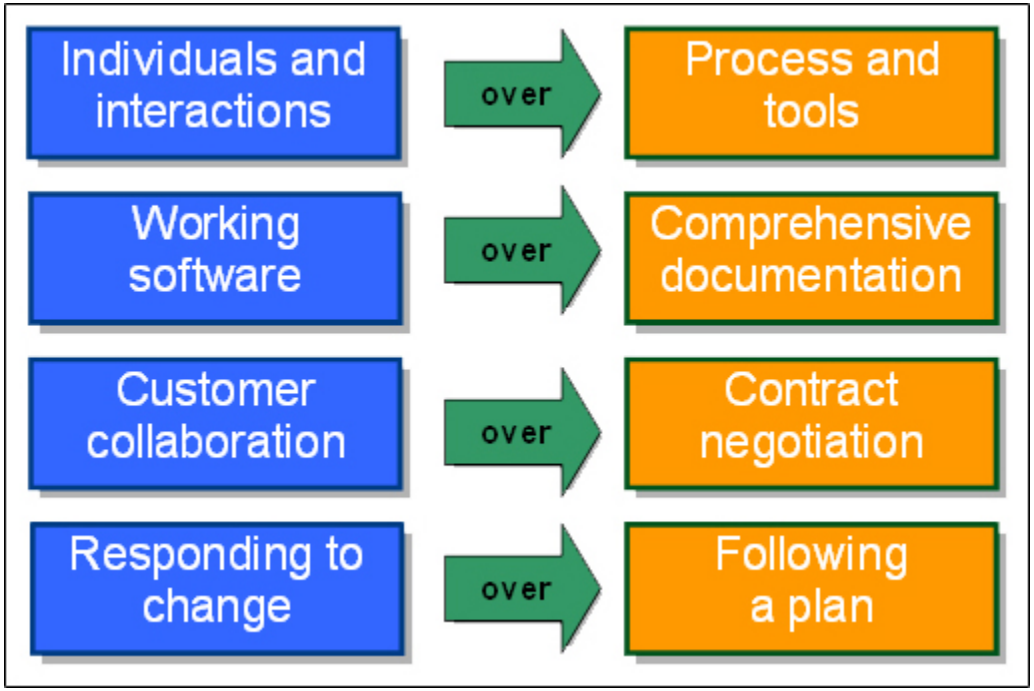
Agile manifesto

- We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - Individuals and interactions over processes and tools
    Working software over comprehensive documentation
    Customer collaboration over contract negotiation
    Responding to change over following a plan
- That is, while there is value in the items on the right, we value the items on the left more.

The principles of agile methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Agile Values



Individuals and interactions over Process and tools

Working software over Comprehensive documentation

Customer collaboration over Contract negotiation

Responding to change over Following a plan

Agile Remarks

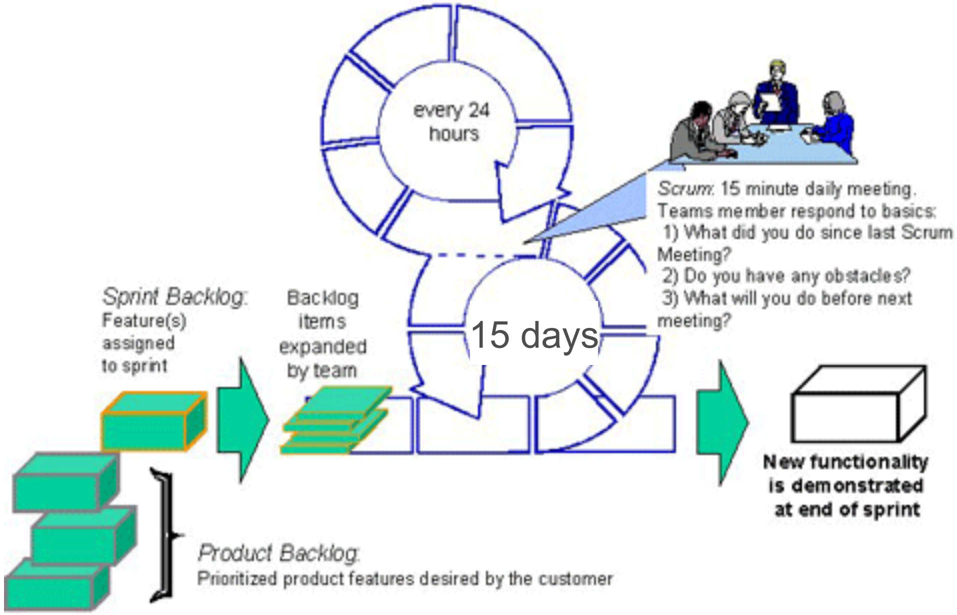
- The professional goal of every software engineer, and every development team, is to deliver the highest possible value to our employers and customers.
  - And yet, our projects fail, or fail to deliver value, at a dismaying rate.
- Though well intentioned, the **upward spiral of process inflation** is culpable for at least some of this failure.
- The principles and values of agile software development were formed as a way
  - to help teams break the cycle of process inflation, and
  - to focus on simple techniques for reaching their goals.

- At the time of this writing there were many agile processes to choose from. These include
  - **SCRUM**,
  - Crystal,
  - Feature Driven Development (FDD),
  - Adaptive Software Development (ADP), and most significantly,
  - Extreme Programming (XP).
  - Others…

Where are we know?

SCRUM basic principles

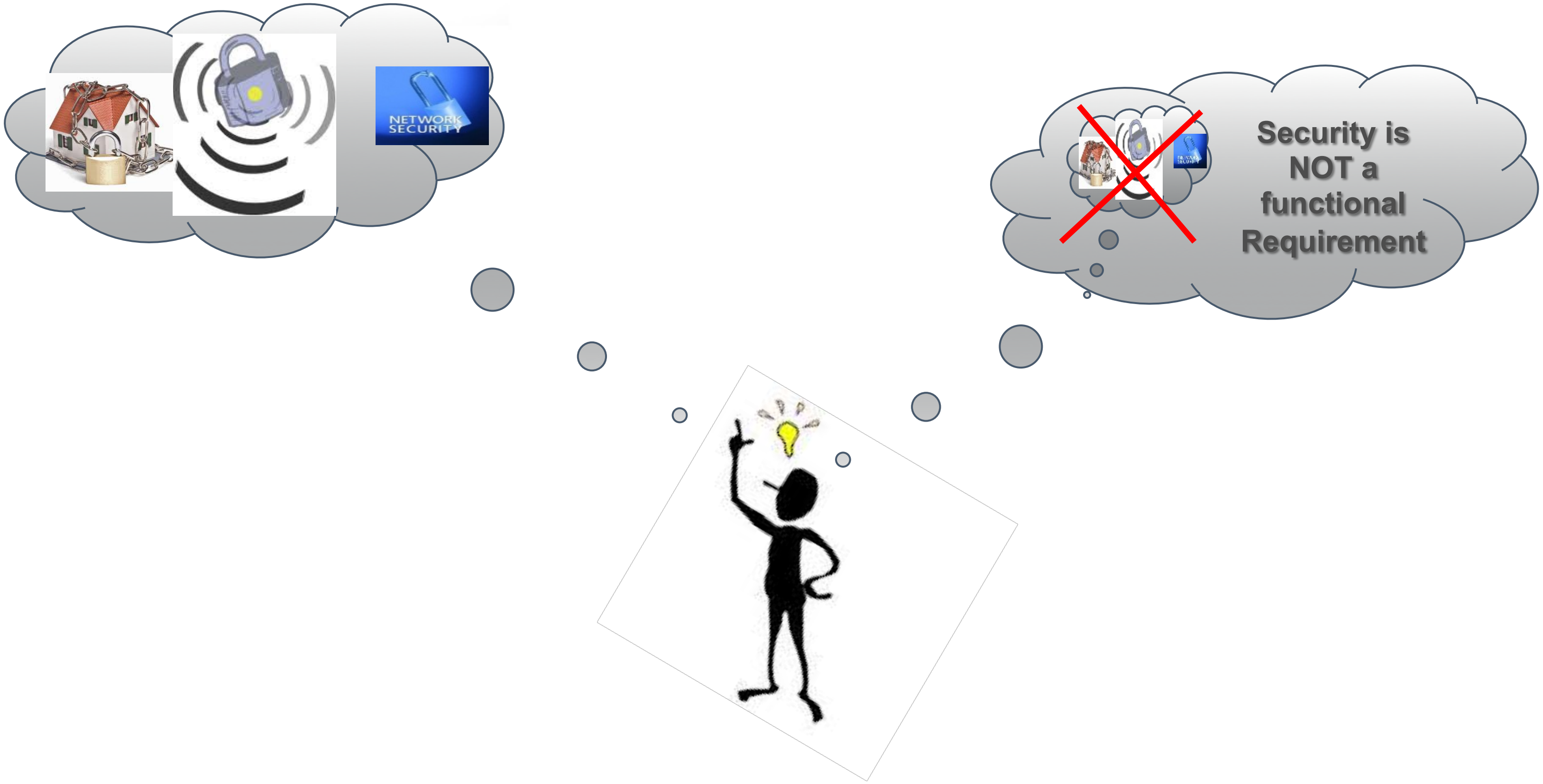
- Early and continuous delivery of valuable software
- Welcome changing requirements, even late in development
- Build projects around motivated individuals and trust them to get the job done.
- Working software as the primary measure of progress
- Continuous attention to technical excellence and good design
- Simplicity—maximizing the amount of work not done
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on, tunes, and adjusts its behavior


every 24 hours
15 days
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
Scrum: 15 minute daily meeting. Teams member respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

Where are we know?

- We trust that our teams are doing their best for security.
  - Do they?
- No specific care in designing for security unless the customer requires that
  - Does it happens now?
- No malicious user stories
- No specific controls for common security flaws

PO view on security

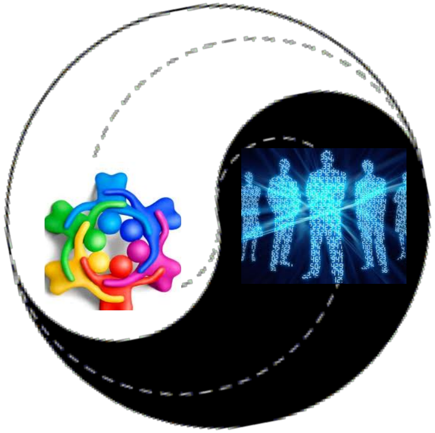


Security is NOT a functional Requirement

Agile vs. Sec Worlds

Agile Teams:

- More responsive to business concerns
- Increasing the frequency of stable releases
- Decreasing the time it takes to deploy new features



Security Teams:

- More aggressive regulatory environment
- Increasing focus on need for security
- Traditional approaches are top-down, document centric

Security in SDLC

Requirements	• Security Requirements
Design	• Security Architecture Review
Implementation	• Secure Code Review
Verification	• Application Vulnerability Testing
Maintenance	• External Application Security Testing

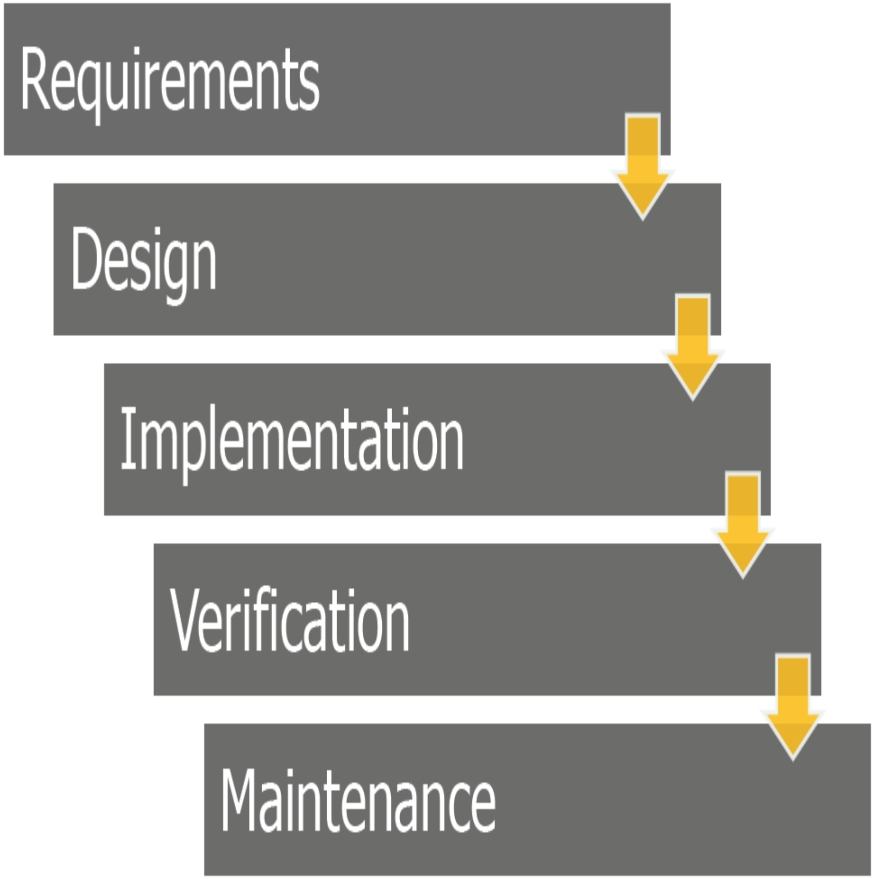
Advantages:

- Well understood process
- Leverages subject matter experts to identify security concerns

Disadvantages:

- Findings from early security reviews are often ignored as "theoretical"
- Costly to go backwards in the development timeline

Waterfall VS. Agile



Requirements

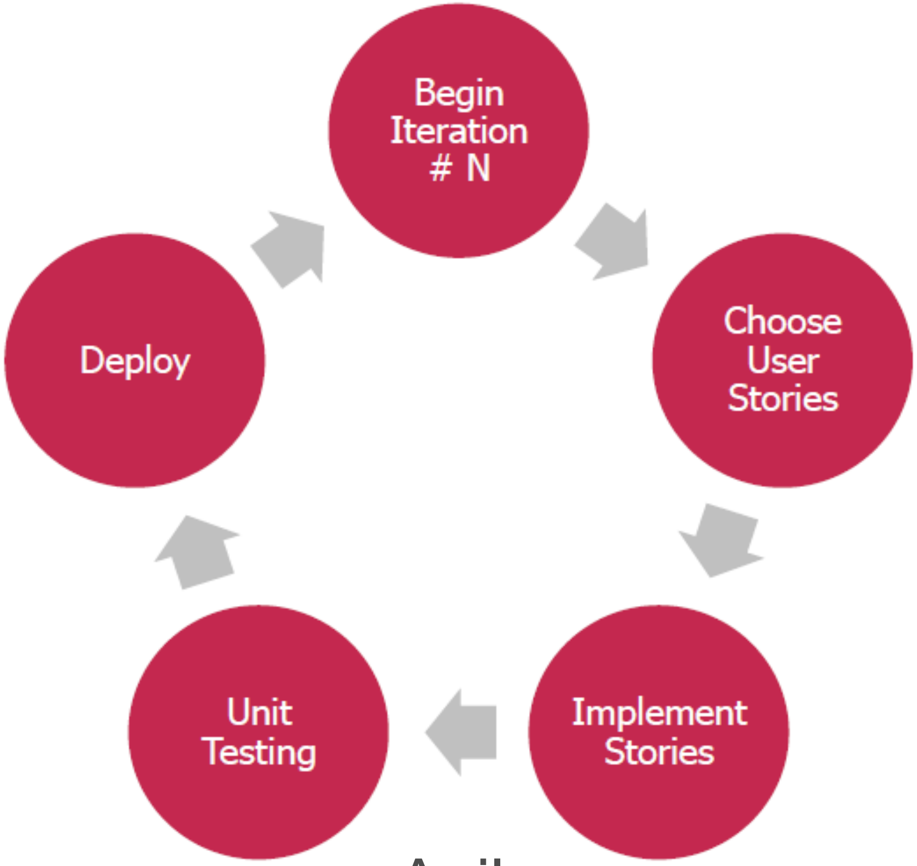
Design

Implementation

Verification

Maintenance

Waterfall



Begin Iteration # N

Choose User Stories

Implement Stories

Unit Testing

Deploy

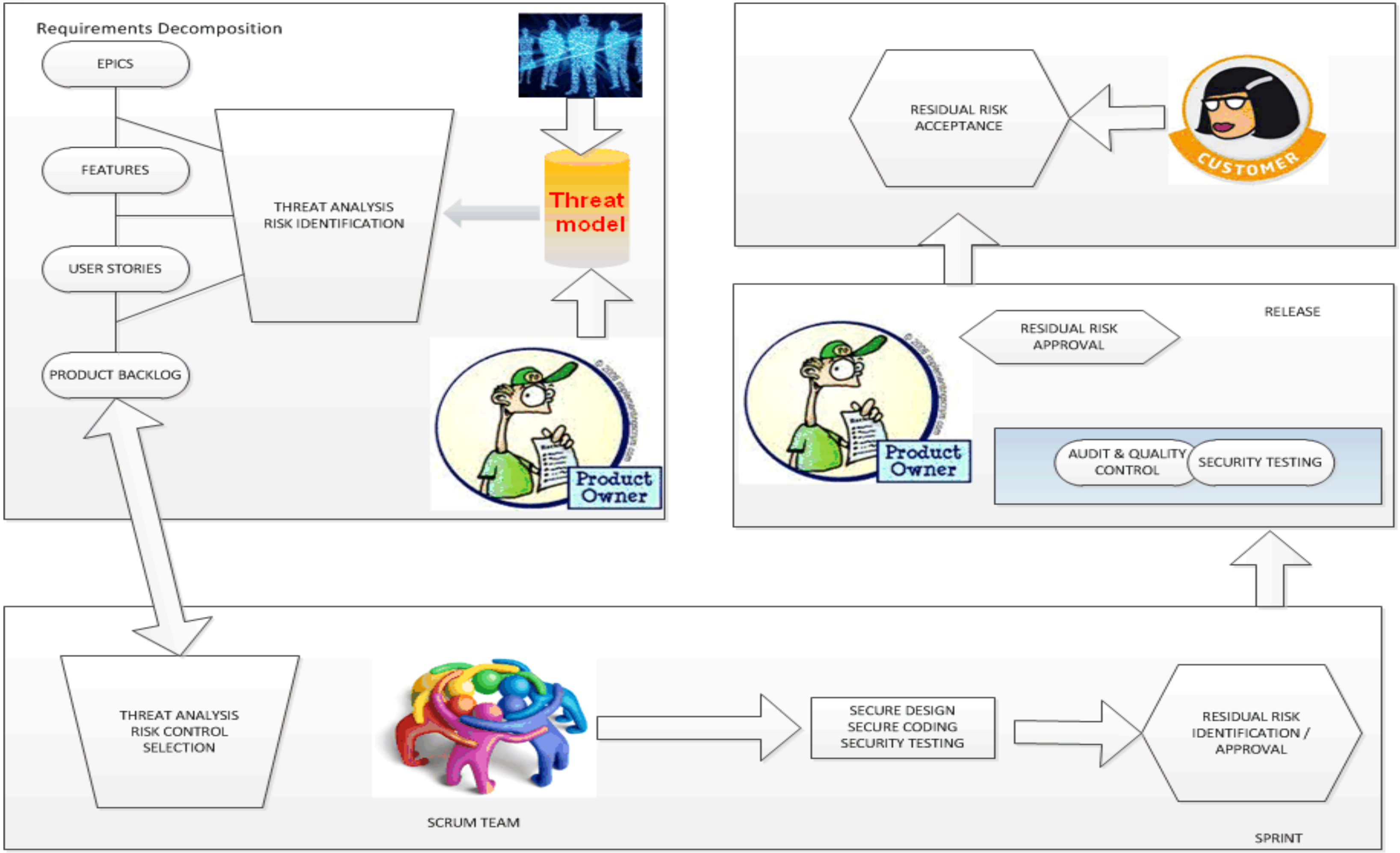
Agile

The Challenge: Lightweight Security Processes

- In SCRUM security processes iterated over and over again (comparing to Waterfall)
- Adjust weight of security processes to distinct scrum controls to keep efforts reasonable

Securing SCRUM

an IT Sec Approach



Requirements Decomposition

EPICS

FEATURES

USER STORIES

PRODUCT BACKLOG

THREAT ANALYSIS
RISK IDENTIFICATION

Threat
model

Product
Owner

RESIDUAL RISK
ACCEPTANCE

CUSTOMER

RELEASE

RESIDUAL RISK
APPROVAL

Product
Owner

AUDIT & QUALITY
CONTROL

SECURITY TESTING

THREAT ANALYSIS
RISK CONTROL
SELECTION

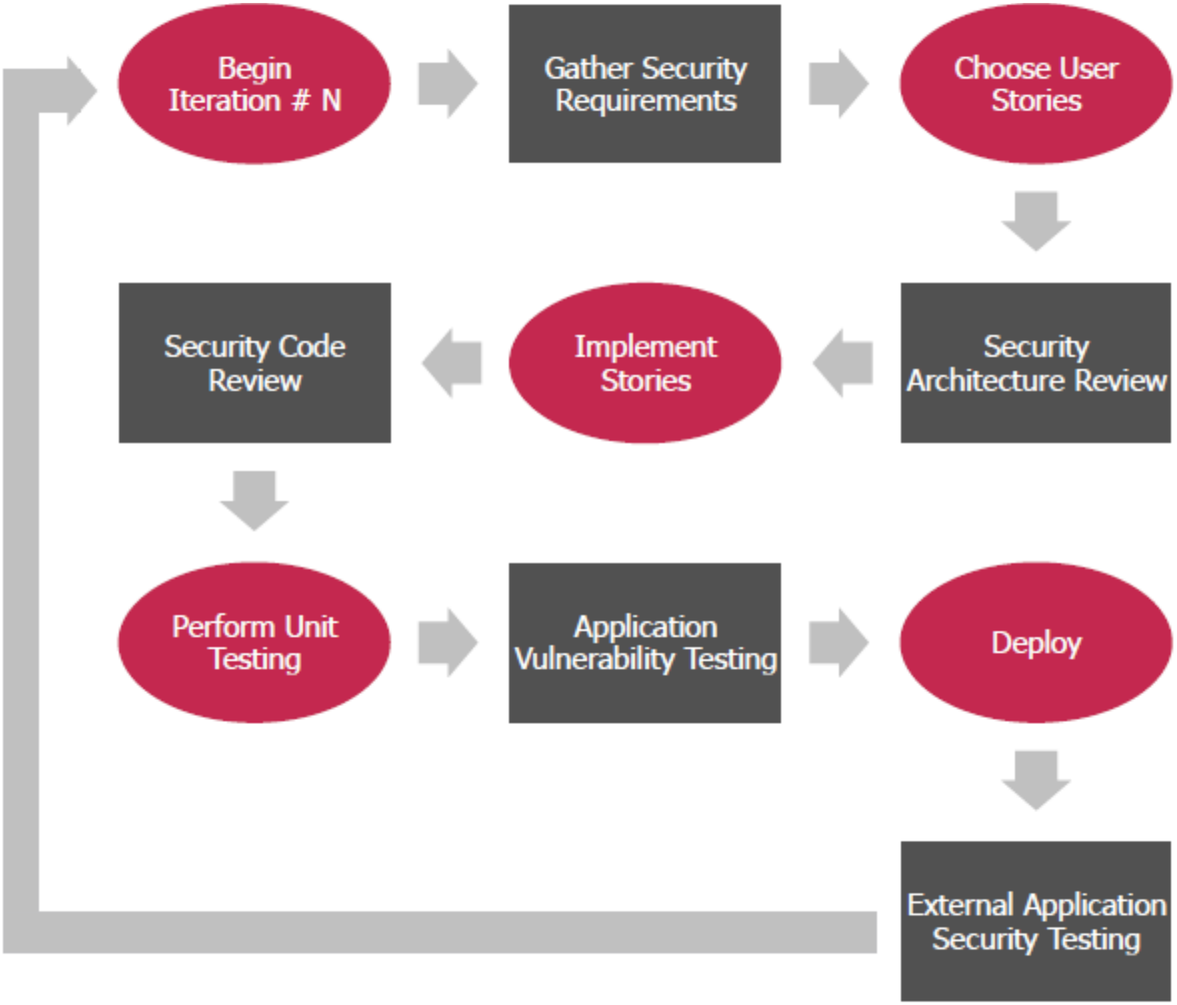
SCRUM TEAM

SECURE DESIGN
SECURE CODING
SECURITY TESTING

RESIDUAL RISK
IDENTIFICATION /
APPROVAL

SPRINT

Traditional Security + Agile Process


Begin Iteration # N
Gather Security Requirements
Choose User Stories
Security Architecture Review
Implement Stories
Security Code Review
Perform Unit Testing
Application Vulnerability Testing
Deploy
External Application Security Testing

Designing a secure product

Designing a Secure Product

- Security by **Design**
  - Attack Surface reduction,
  - Threat Modeling
    - **Octave (**Operationally Critical Threat, Asset, and Vulnerability Evaluation**)**
    - **Microsoft's Security Development Lifecycle Threat Modeling tool**
- Secure by **Default**
- **Clear** Security **requirements** (Customer - Internal)
- **Risk Assessment**
- **Defense in Depth** (applied to software and supporting infrastructure)
- **Compliance with standards** (whenever and if needed)
  - Which?

Security by Design

- Secure by design, in software engineering, means that the software has been designed from the ground up to be secure.
- Malicious practices are taken for granted and care is taken to minimize impact when a security vulnerability is discovered or on invalid user input.

Attack Surface Reduction (ASR)

The Attack Surface Reduction Process

- Look at all of your entry points
  - Network I/O
  - File I/O
- Rank them
  - Authenticated versus anonymous
  - Administrator only versus user
  - Network versus local
  - UDP versus TCP

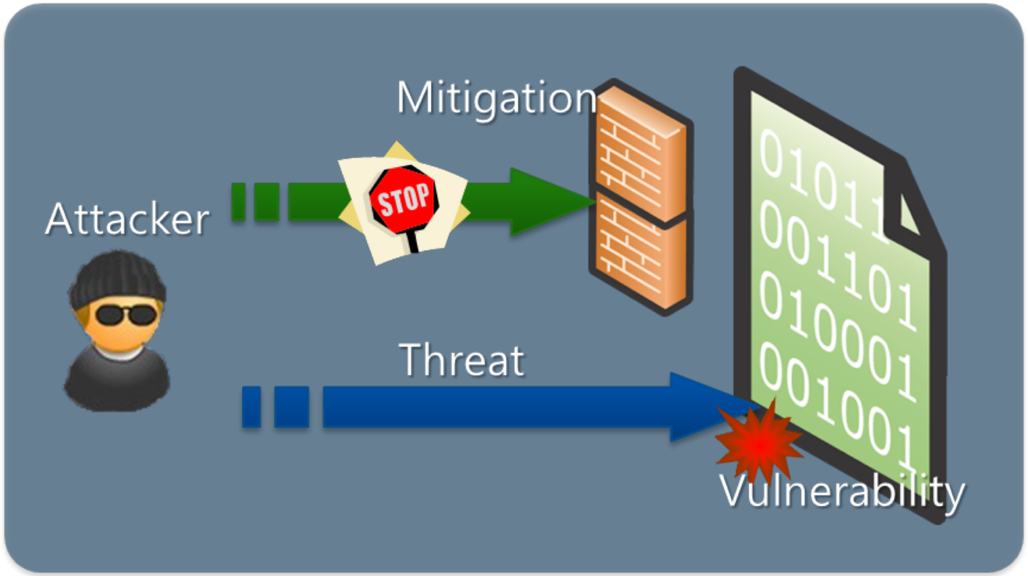
It’s Not Just About Turning Stuff Off!

Higher Attack Surface	Lower Attack Surface
Executing by default	Off by default
Open socket	Closed socket
UDP	TCP
Anonymous access	Authenticated access
Constantly on	Intermittently on
Admin access	User access
Internet access	Local subnet access
SYSTEM	Not SYSTEM!
Uniform defaults	User-chosen settings
Large code	Small code
Weak ACLs	Strong ACLs

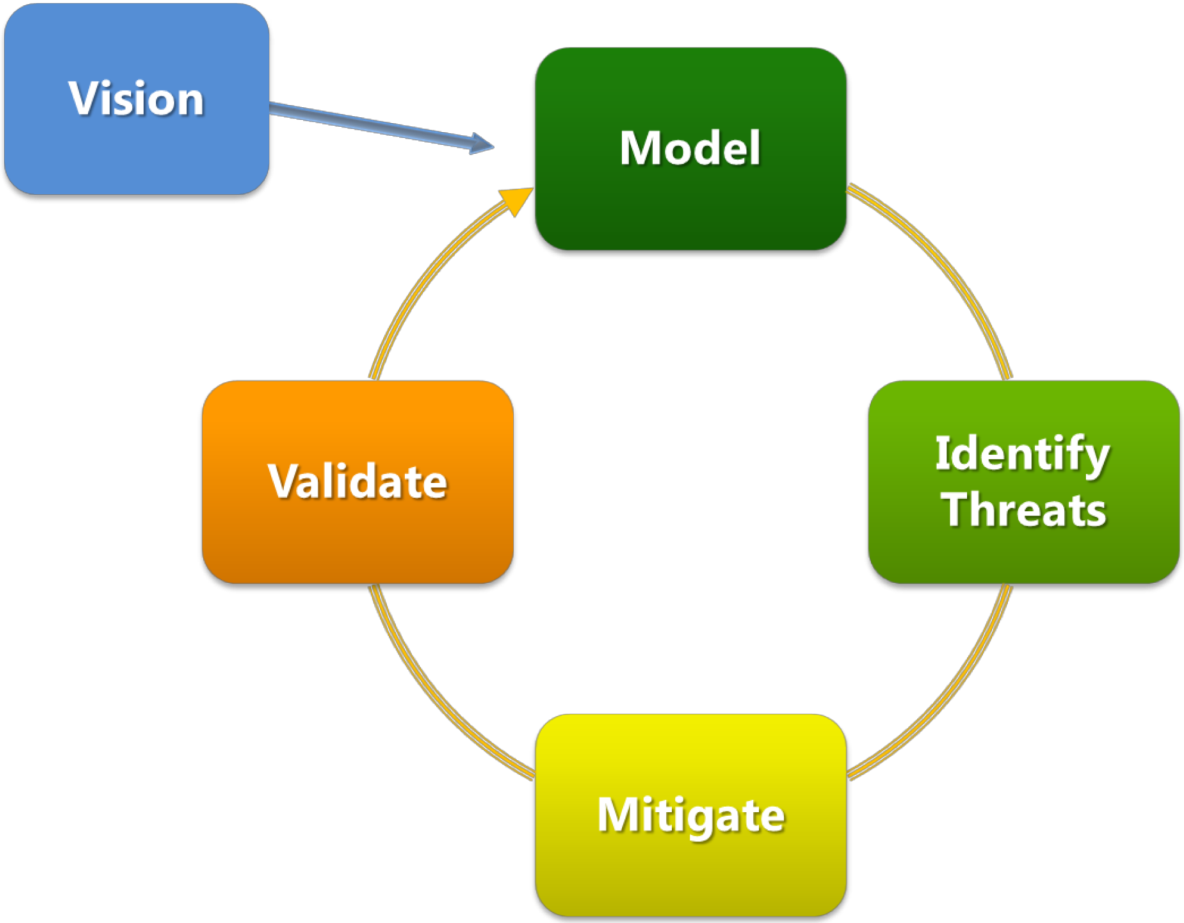
Attack Surface Reduction is as important as trying to get the code right

Threat Modeling

- Threat Analysis
  - Secure software starts with understanding the threats
  - Threats are not vulnerabilities
  - Threats live forever; they are the attacker's goal



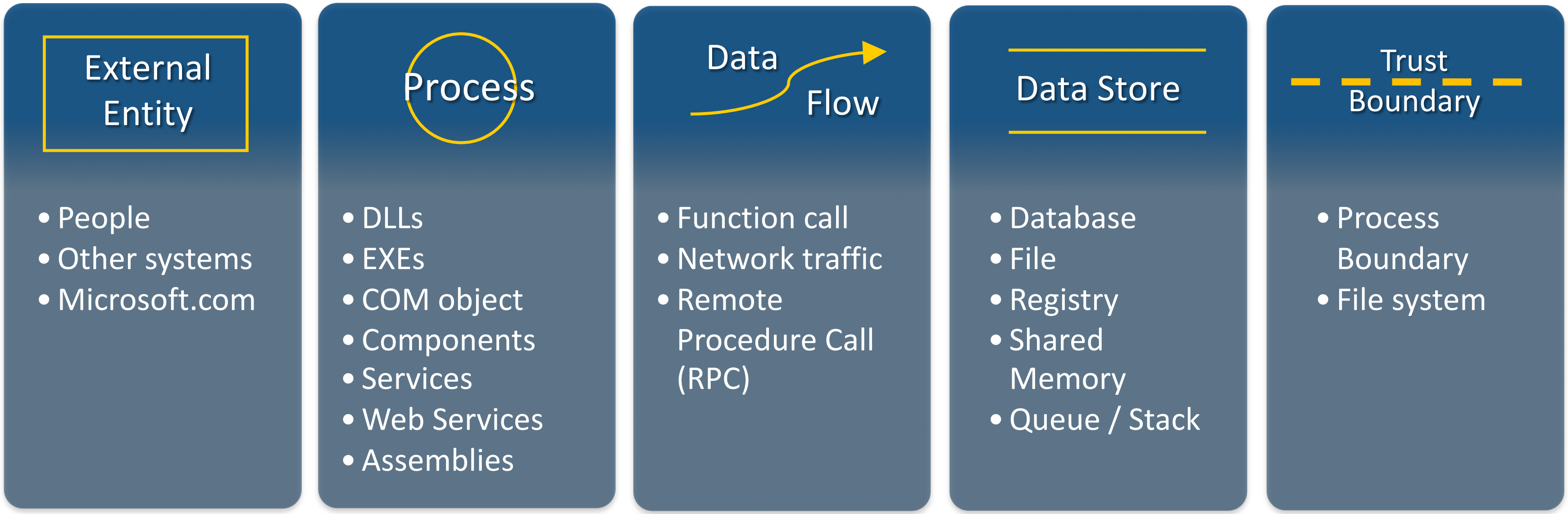
Thread Modeling Process



Thread Modeling Process

Whiteboard Your Architecture

- Start with person, processes, data flows, data stores
    - Unique shape per item
    - Data flows should be one way each
    - Label them with data, not read/write
- Draw attack surfaces/trust boundaries
- Tell a story to see if your picture is ok



External Entity

- People
- Other systems
- Microsoft.com

Process

- DLLs
- EXEs
- COM object
- Components
- Services
- Web Services
- Assemblies

Data Flow

- Function call
- Network traffic
- Remote Procedure Call (RPC)

Data Store

- Database
- File
- Registry
- Shared Memory
- Queue / Stack

Trust Boundary

- Process Boundary
- File system

Find Threats: Use STRIDE per Element

- Start with items connected to dangerous data flows (those crossing boundaries)
- Use the chart to help you think of attacks
- Keep a running list

Spoofing

Tampering

Repudiation

Information Disclosure

Denial of Service

Elevation of Privilege

	S	T	R	I	D	E
External Entity	✓		✓			
Process	✓	✓	✓	✓	✓	✓
Data Store		✓	✓	✓	✓	
Data Flow		✓		✓	✓	

Mitigating Threats

- For each threat, decide how to stop it
  - Redesign and eliminate
  - Use standard threat mitigations
  - Invent new mitigation (not recommended)
  - Accept risk in File a work item in your bug tracking DB
  - Treat threats as bugs, mitigations as features

Validate

- Check threat model diagrams
  - Do they match the design docs or code?

Potential Methodologies - Tools

- Octave (Operationally Critical Threat, Asset, and Vulnerability Evaluation) : is a suite of tools, techniques, and methods for risk-based information security strategic assessment and planning.
    - Free family of tools not automated
    - Part of the US – Cert tool chain.
    - Hard to Implement.
- Microsoft's Security Development Lifecycle (SDL)Threat Modeling tool
    - Based on MS SDL methodology
    - Adopted to Scrum processes
    - Integrated to Visual Studio

Secure by Default

- Security by default, in software, means that the default configuration settings are the most secure settings possible, which are not necessarily the most user friendly settings.
  - Allow only those functionalities that are explicitly need and with the less privileges.

Designing a Secure Product

- Clear Security requirements (Customer - Internal)
  - Difficult at the moment to have customer's requirements
  - Must decide internal baseline security requirements

Designing a Secure Product

- Risk Assessment
  - Part of the Treat Modeling process
- RA Methodologies
  - Microsoft SDL:
    - STRIDE (Identification of threats)
    - DREAD (quantifying, comparing and prioritizing the amount of risk presented by each evaluated threat)
  - Others

Designing a Secure Product

- Defense in Depth:
  - is an information assurance (IA) concept in which multiple layers of security controls (defense) are placed throughout an information technology (IT) system.
  - Its intent is to provide redundancy in the event a security control fails or a vulnerability is exploited which can cover aspects of personnel, procedural, technical and physical for the duration of the system's life cycle.

Designing a Secure Product

- Defense in Depth:
  - is an information assurance (IA) concept in which multiple layers of security controls (defense) are placed throughout an information technology (IT) system.
  - Its intent is to provide redundancy in the event a security control fails or a vulnerability is exploited which can cover aspects of personnel, procedural, technical and physical for the duration of the system's life cycle.

Designing a Secure Product

- **Compliance with standards** (whenever and if needed)
  - Which?
  - Where?
  - When?
- There a lot of different security standards from different bodies
  - ITU has more than 50 ICT related standards (http://www.itu.int/ITU-T/studygroups/com17/ict/part02.html)
  - Same condition in
    - ISO
    - ISA
    - NIST
- We must decide…

Developing Secure products

Developing a Secure Product

- Threat Risk Mitigation
- Adopt and follow Principles
- Education and Training
- Learn from mistakes

- Think like an adversary

Developing a Secure Product

Risk Mitigation Techniques

Threat	Mitigation Feature
Spoofing	Authentication
Tampering	Integrity
Repudiation	Nonrepudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

Developing a Secure Product

- Adopt and follow Principles – Best Practices
- Top 10 Secure Coding Practices
  - Validate input.
  - Heed compiler warnings.
  - Architect and design for security policies.
  - Keep it simple.
  - Default deny.
  - Adhere to the principle of least privilege.
  - Sanitize data sent to other systems.
  - Practice defense in depth.
  - Use effective quality assurance techniques.
  - Adopt a secure coding standard.

Developing a Secure Product

- Education and Training
- At a minimum, train all Product Owners.
- Scrum team autonomy: Trust, but verify.
- Train two persons in every team to act as the “security conscience”.
- Repeat training periodically adjust to new threats.

Developing a Secure Product

- Learn from mistakes
  - Use Scrum Controls to propagate lessons learned
    - Scrum of Scrums
    - Retrospectives

Developing a Secure Product

- Think like an adversary

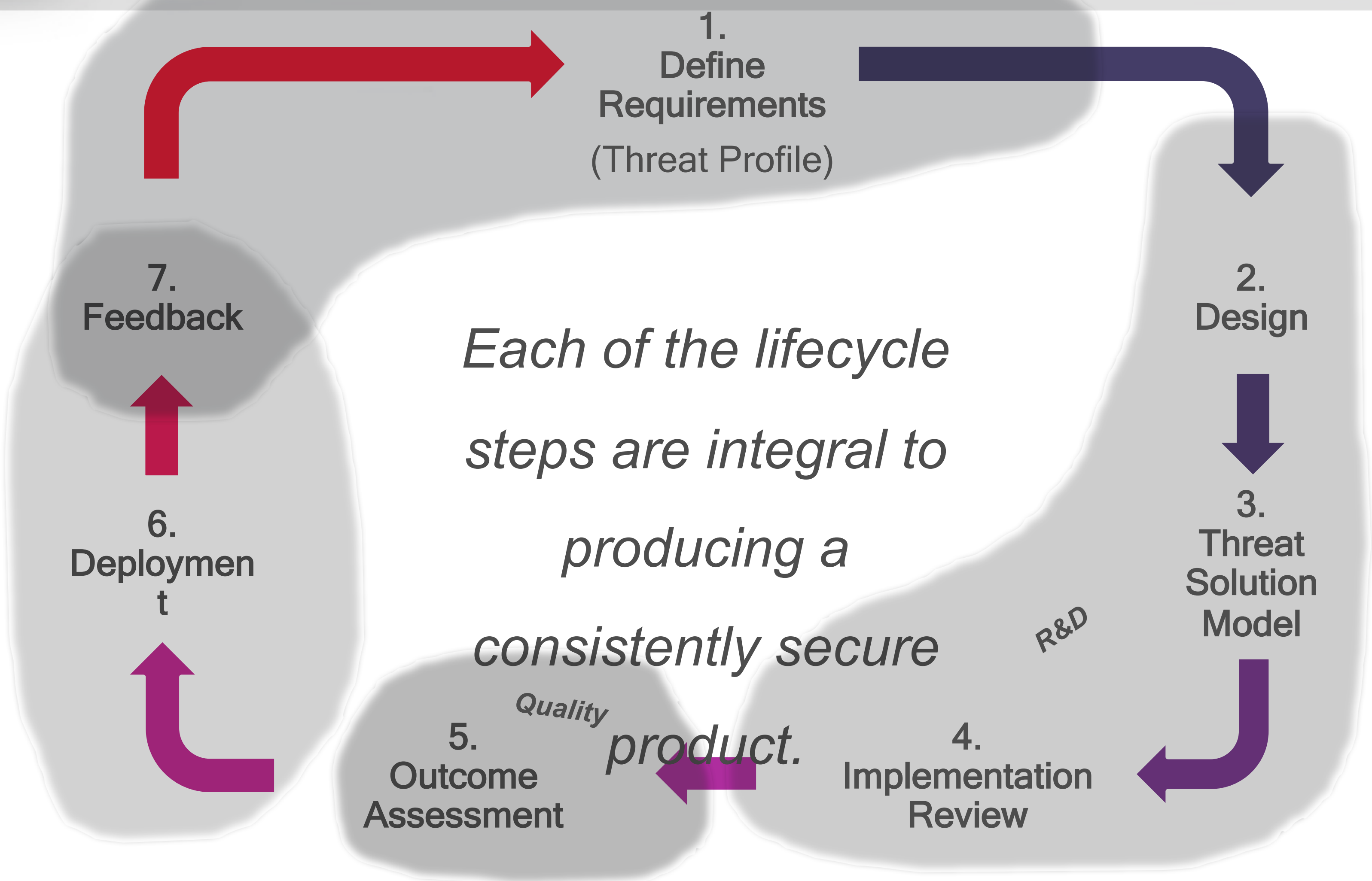
Releasing a Secure Product

- Allocate time for Security Testing
  - Test for common flaws
  - Security Code reviews
  - Infrastructure and software penetration testing.
- Security in **Deployment**
- **Think like an adversary**

Maintaining a Secure Product

- Fix security issues correctly
- Infrastructure and software penetration testing
- Adjust to changes of the supporting infrastructure (patches to OS, libs, etc.)

Product Security Lifecycle



1. Define Requirements (Threat Profile)

2. Design

3. Threat Solution Model

R&D

4. Implementation Review

Quality

5. Outcome Assessment

6. Deploymen t

7. Feedback

Each of the lifecycle steps are integral to producing a consistently secure product.

Immediate steps to current Product's line

- Develop Threat Model Product Owner
- Identify Risk Product Owner
- Plan evil Use Cases Product Owner
- Develop risk mitigation controls
- Calculate residual Risk
- Outcome Assessment Product Owner I ♥ Auditing
- Feedback CUSTOMER
- Continues Improvement Product Owner I ♥ Auditing

Next Steps

- Educate/Train the PO’s
- Develop/Adopt a Threat analysis model
- Create process to map threat model to User stories
- Create Unit Security Tests
- Identify and Use standard security controls

Next Steps

- Develop procedures for the correct use of secure coding standards
- Develop/Adopt a Threat analysis model
- Provide security training to developers (security awareness and proper use of controls)
- Leverage Security experts
- Appoint Security Officers within SCRUM teams.

Discussion