Programmer to Secure Agile Programmer

SKILLSOFT ASPIRE JOURNEY

skillsoft

Głównym wyzwaniem przed którym stają dziś organizacje na całym świecie jest konieczność ciągłego podnoszenia umiejętności i poziomu wiedzy w ślad za gwałtownym rozwojem nowych technologii i zmian na globalnym rynku.

Stały rozwój i podnoszenie kwalifikacji w IT od dawna jest już rzeczą oczywistą, a możliwość zapewnienia wsparcia specjalistom chcącym stale się rozwijać jest jedną z głównych kart przetargowych w walce o pracownika.

Na rynku liczą się dziś ludzie, którzy posiadają konkretne kompetencje i zestaw umiejętności pozwalający im wykonywać zadania efektywnie, a nie Ci z najdłuższym stażem pracy.

Dziś, bardziej niż kiedykolwiek w cenie jest umiejętność budowania ścieżki kariery dla profesjonalistów IT, którzy wciąż chcą się liczyć na rynku pracy.

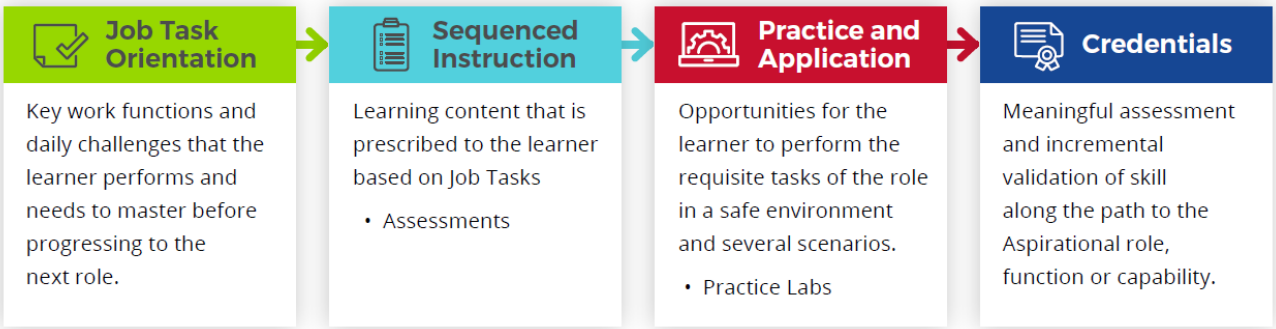
Skillsoft Aspire Journey stanowi odpowiedź na pytanie, jakie szkolenia muszę ukończyć, aby być przygotowanym do mojej wymarzonej pracy. Spośród kilkuset kanałów tematycznych dostępnych na naszej platformie szkoleniowej nasi specjaliści wybrali te, które naszym zdaniem najlepiej wyposażą uczących się w narzędzia potrzebne do realizacji zadań w nowej roli.

Skillsoft Aspire Journey to zestawy szkoleń i ćwiczeń w języku angielskim, które metodycznie, krok po kroku pozwalają specjalistom przejść od poziomu podstawowego do zaawansowanego.

Każda ścieżka zawiera szkolenia, laboratoria wirtualne, video i książki, które pomogą uczącym się osiągnąć pożądane kompetencje poświadczone certyfikatem.

Aspire Journey Model

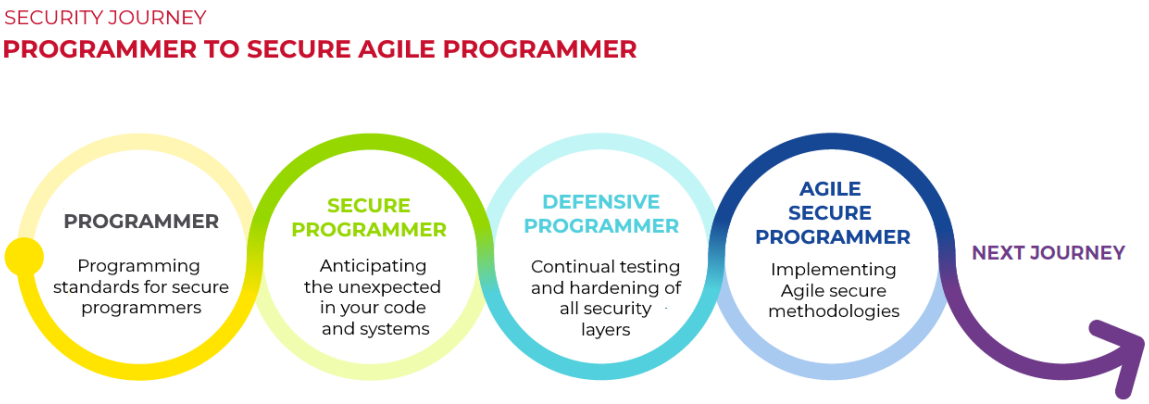
Cała ścieżka opiera się na 4-elementowym cyklu powtarzanym na kolejnych etapach nauki.



1. Określenie kluczowych funkcji i wyzwań, z którymi musi poradzić sobie uczący się w chwili obecnej, jak i tymi, z którymi przyjdzie mu się zmierzyć w nowej pracy.
2. Przejście zaprojektowanych ścieżek w proponowanej kolejności, wykonanie ćwiczeń i zaliczenie testów.
3. Przećwiczenie nowych umiejętności w kontrolowanym środowisku w oparciu o gotowe scenariusze działań. Laboratoria wirtualne Skillsoft
4. Certyfikat – zaliczenie testu końcowego na poziomie co najmniej 70% i uzyskanie certyfikatu potwierdzającego ukończenie danego etapu nauki.

Aspire Journey – Penetration Tester to SpecOps Engineer

Analizując trendy opisujące zachowanie użytkowników na naszych platformach szkoleniowych i współpracując ściśle z naszymi klientami na całym świecie Skillsoft wyselekcjonował najlepsze materiały szkoleniowe i ułożył je w ustrukturalizowaną ścieżkę rozwoju. Ścieżka zawiera około 27 godzin szkoleniowych.



6 courses
4h 40m 19s

- focus on programming standards for secure programmers

6 courses
3h 35m 33s

- focus on security concepts, vulnerabilities, encryption, attacks and resiliency coding for secure programmers

7 courses
4h 55m 27s

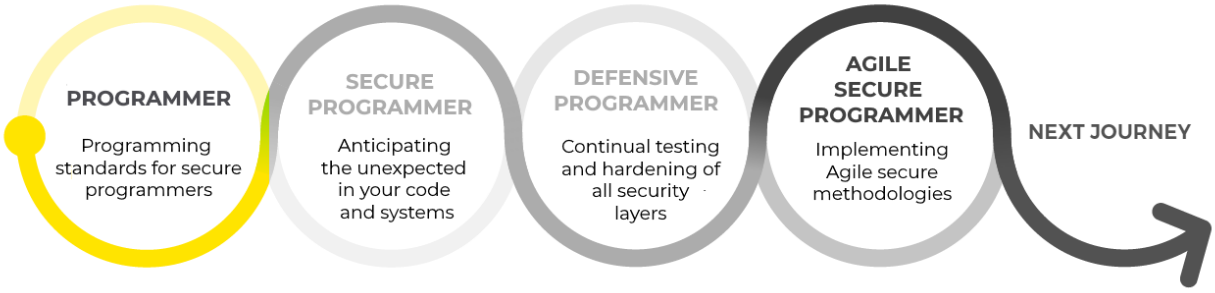
- focus on defensive concepts and techniques, cryptography, code sampling, secure testing, and advanced defensive programmer concepts

5 courses
2h 15m 9s

- focus on secure Agile programming concepts, techniques, modeling, and testing.

SECURITY JOURNEY

PROGRAMMER TO SECURE AGILE PROGRAMMER



Track 1: Programmer (duration: 4h 40m 19s)

 Secure Programmer: Intro to Programming Standards	 Secure Programmer: Software Design Techniques
Objectives: ▪ define basic programming & software engineering concepts ▪ recall IEEE programming standards including general, testing and quality, and maintenance and documentation standards ▪ recall IEEE programming standards including NIST SP 800-27, ISO/IEC 15504 and 24744:2014, and ISO 29110 ▪ recall IEEE and ISO programming standards ▪ identify software requirement types, the FURPS model, and the requirements gathering techniques ▪ identify requirements gathering techniques such as brainstorming, interviews, focus groups, and reverse engineering ▪ describe quality and the change management process ▪ apply the IEEE Std 730 standard for software quality	Objectives: ▪ recognize software design concepts ▪ apply modular design ▪ apply resiliency design ▪ apply architectural design ▪ apply component level design ▪ apply pattern-based design ▪ recognize well designed Java code ▪ recognize well designed Python code ▪ recognize well designed C# code ▪ recognize well designed JavaScript ▪ recognize model-driven design

Secure Programmer: Software Modeling Techniques

Objectives:

- recognize the Unified Modeling Language
- use specific UML diagrams including class, activity, use case, and sequence diagrams
- describe SysML and recognize how it can be used
- use specific SysML diagrams including block definition, internal block, and parametric diagrams

Secure Programmer: Coding Practices

Objectives:

- perform software estimation of resources and time
- apply good coding practices
- recognize bad Java programming
- recognize bad Python programming
- recognize bad C# programming
- recognize bad JavaScript programming
- apply good programming in Java
- apply good programming in Python
- apply good programming in C#
- apply good programming in JavaScript

Secure Programmer: Software Testing

Objectives:

- describe and apply testing methodologies
- apply unit testing
- apply integration testing
- apply regression testing
- apply user acceptance testing
- describe roles and responsibilities in testing
- specific testing methods
- understand test cases and reporting
- apply software metrics
- describe software verification and validation
- describe bug tracking concepts
- use bug tracking methods

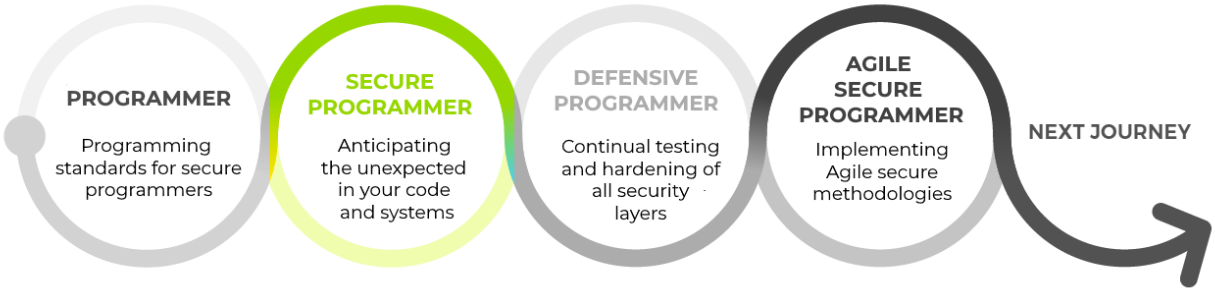
Secure Programmer: Final Exam

Objectives:

- apply architectural design
- apply component level design
- apply good coding practices
- apply good programming in Java
- apply good programming in JavaScript
- apply good programming in Python
- apply integration testing
- apply modular design
- apply pattern based design
- apply regression testing
- apply resiliency design
- apply software metrics
- apply specific UML diagrams including class, activity, use case, and sequence diagrams
- apply testing methodologies
- apply the IEEE Std 730 standard for software quality
- apply unit testing
- apply user acceptance testing
- define basic programming & software engineering concepts
- describe and apply testing methodologies
- describe bug tracking concepts
- describe modular design
- describe resiliency design
- describe software verification and validation
- describes roles and responsibilities in testing
- describe the quality and the change management process
- describe unit testing
- identify architectural design
- identify bad JavaScript programming
- identify IEEE programming standards including general, testing and quality, and maintenance and documentation standards
- identify requirements gathering techniques such as brainstorming, interviews, focus groups, and reverse engineering
- identify software design concepts
- identify software requirement types, the FURPS model, and methods for gathering requirements
- identify the Unified Modeling Language
- identify user acceptance testing
- implement good coding practices
- implement good programming in Java
- implement good programming in JavaScript
- implement good programming in Python
- implement unit testing
- perform software estimation of resources and time
- recall IEEE and ISO programming standards
- recall IEEE programming standards including general, testing and quality, and maintenance and documentation standards
- recall IEEE programming standards including NIST SP 800-27, ISO/IEC 15504 and 24744:2014, and ISO 29110
- recognize bad Java programming
- recognize bad JavaScript programming
- recognize bad Python programming
- recognize component level design
- recognize IEEE and ISO programming standards
- recognize IEEE programming standards including general, testing and quality, and maintenance and documentation standards
- recognize model driven design
- recognize software design concepts
- recognize the Unified Modeling Language
- recognize well designed Java code
- recognize well-designed JavaScript
- recognize well designed Python code
- specific testing methods
- understand test cases and reporting
- use bug tracking methods
- use specific UML diagrams
- use specific UML diagrams including class, activity, use case, and sequence diagrams

SECURITY JOURNEY

PROGRAMMER TO SECURE AGILE PROGRAMMER



Track 2: Secure Programmer (duration: 3h 35m 33s)

 Secure Programmer: Security Concepts	 Secure Programmer: Vulnerabilities
Objectives: ▪ describe security concepts, including the CIA triangle, least privileges, and separation of duties ▪ describe authentication and authorization, including models such as DAC, MAC, RBAC, and ABAC ▪ describe and be able to avoid common programming errors that can undermine security ▪ describe secure programming verification and validation process and techniques	Objectives: ▪ describe specific security vulnerabilities and recognize how to program counter techniques ▪ describe OWASP Top 10 vulnerabilities including SQL injection, broken authentication, and cross-site scripting ▪ describe OWASP Top 10 vulnerabilities including broken access control, security misconfiguration, sensitive data exposure, and insufficient attack protection ▪ describe OWASP Top 10 vulnerabilities including cross-site request forgery, using components with known vulnerabilities, and underprotected APIs ▪ describe and use threat models including STRIDE, PASTA, DREAD, and SQUARE ▪ describe and use CVE vulnerability scoring ▪ implement Java secure coding to combat Rhino Script vulnerability ▪ implement Python secure coding to combat Remote Code Execution Vulnerability ▪ implement C# secure coding to combat SQL Injection Vulnerability ▪ implement JavaScript secure coding to combat SQL Injection Vulnerability ▪ implement Java secure coding to combat SQL Injection Vulnerability ▪ implement Python secure coding to combat a variety of security vulnerabilities ▪ implement C# secure coding to combat common code vulnerabilities ▪ implement JavaScript secure coding to combat Cross Site Scripting attacks ▪ use CVSS scoring for vulnerabilities ▪ use OWASP Zap vulnerability scanner to test web sites for common vulnerabilities ▪ use Vega Vulnerability Scanner to test web sites for common vulnerabilities

Secure Programmer: Encryption

Objectives:

- describe symmetric algorithms including AES, Blowfish, and Serpent
- describe asymmetric algorithms including RSA, ECC, and Diffie-Helman
- describe hashing algorithms such as MD5 and SHA as well as MAC and HMAC

Secure Programmer: Attacks

Objectives:

- code against format string attacks in Java
- code against format string attacks in Python
- code against format string attacks in C#
- code against SQL injection attacks in Java
- code against SQL injection attacks in Python
- code against SQL injection attacks in C#
- code against SQL injection attacks in JavaScript
- code against buffer overflow attacks in Java
- code against buffer overflow attacks in Python
- code against buffer overflow attacks in C#
- code against buffer overflow attacks in JavaScript
- code against cross-site scripting attacks in Java
- code against cross-site scripting attacks in Python
- code against cross-site scripting attacks in C#
- code against cross-site scripting attacks in JavaScript
- code against password cracking attacks in Java
- code against password cracking attacks in Python
- code against password cracking attacks in C#
- code against password cracking attacks in JavaScript

Secure Programmer: Resiliency Coding

Objectives:

- describe the resiliency concepts such as stability, recovery, and defensive coding
- write resilient Java code
- write resilient Python code
- write resilient C# code
- write resilient JavaScript code

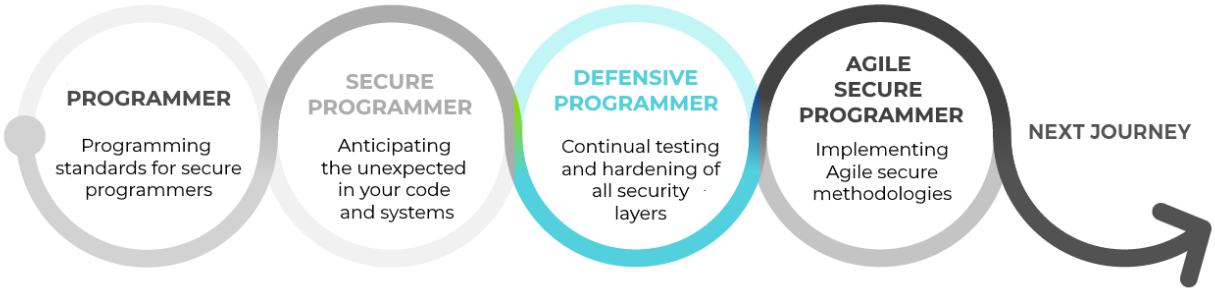
Secure Programmer: Final Exam

Objectives:

- apply C# secure coding to combat common code vulnerabilities
- apply JavaScript secure coding to combat SQL Injection Vulnerability
- code against buffer overflow attacks in C#
- code against buffer overflow attacks in Java
- code against buffer overflow attacks in Java - part 2
- code against buffer overflow attacks in Java - part 3
- code against buffer overflow attacks in JavaScript
- code against buffer overflow attacks in Python
- code against cross-site scripting attacks in C#
- code against cross-site scripting attacks in Java
- code against cross-site scripting attacks in JavaScript
- code against cross-site scripting attacks in JavaScript - part 2
- code against cross-site scripting attacks in Python
- code against format string attacks in C#
- code against format string attacks in Java
- code against format string attacks in Python
- code against password cracking attacks in JavaScript
- code against password cracking attacks in JavaScript - part 2
- code against SQL injection attacks in C#
- code against SQL injection attacks in C# - part 2
- code against SQL injection attacks in Java
- code against SQL injection attacks in Java - part 2
- code against SQL injection attacks in JavaScript
- code against SQL injection attacks in Python
- code against SQL injection attacks in Python - part 2
- describe and be able to avoid common programming errors that can undermine the security
- describe and use CVE vulnerability scoring
- describe and use threat models including STRIDE, PASTA, DREAD, and SQUARE
- describe asymmetric algorithms including RSA, ECC, and Diffie-Helman
- describe authentication and authorization, including models such as DAC, MAC, RBAC, and ABAC
- describe hashing algorithms such as MD5 and SHA as well as MAC and HMAC
- describe OWASP Top 10 vulnerabilities
- describe OWASP Top 10 vulnerabilities including broken access control, security misconfiguration, sensitive data exposure, and insufficient attack protection
- describe OWASP Top 10 vulnerabilities including cross-site request forgery, using components with known vulnerabilities, and underprotected APIs
- describe OWASP Top 10 vulnerabilities including SQL injection, broken authentication, and cross-site scripting
- describe secure programming verification and validation process and techniques
- describe security concepts, including the CIA triangle, least privileges, and separation of duties
- describe specific security vulnerabilities and recognize how to program counter techniques
- describe symmetric algorithms including AES, Blowfish, and Serpent
- describe the resiliency concepts such as stability, recovery, and defensive coding
- identify OWASP Top 10 vulnerabilities including broken access control, security misconfiguration, sensitive data exposure, and insufficient attack protection
- identify OWASP Top 10 vulnerabilities including cross-site request forgery, using components with known vulnerabilities, and underprotected APIs
- identify security concepts, including the CIA triangle, least privileges, and separation of duties
- identify symmetric algorithms including AES, Blowfish, and Serpent
- identify the resiliency concepts such as stability, recovery, and defensive coding
- implement C# secure coding to combat common code vulnerabilities
- implement JavaScript secure coding to combat Cross-Site Scripting attacks
- implement JavaScript secure coding to combat SQL Injection Vulnerability
- implement Java secure coding to combat SQL Injection Vulnerability
- implement Python secure coding to combat a variety of security vulnerabilities
- recognize OWASP Top 10 vulnerabilities including broken access control, security misconfiguration, sensitive data exposure, and insufficient attack protection
- recognize specific security vulnerabilities and recognize how to program counter techniques
- use CVSS scoring for vulnerabilities
- use OWASP Zap vulnerability scanner to test web sites for common vulnerabilities
- use Vega Vulnerability Scanner to test web sites for common vulnerabilities
- write resilient C# code
- write resilient Java code
- write resilient Java code - part 2
- write resilient JavaScript code
- write resilient Python code

SECURITY JOURNEY

PROGRAMMER TO SECURE AGILE PROGRAMMER



Track 3: Defensive Programmer (duration: 4h 55m 27s)

Defensive Programmer: Defensive Concepts

Objectives:

- identify general defensive concepts
- describe the first five CERT Top 10 secure coding practices - Validate input, Heed compiler warnings, Architect and design for security, Keep it simple, and the Default deny
- describe the last five CERT Top 10 secure coding practices - Adhere to the principle of least privilege, Sanitize data sent to other systems, Practice defense in depth, Use effective quality assurance techniques, and Adopt a secure coding standard
- apply defensive coding
- use Open Source Security Testing Methodology Manual concepts
- apply the Flaw Hypothesis Method
- describe the role of Six Sigma in producing better quality, secure programming

Defensive Programmer: Defensive Techniques

Objectives:

- apply exception handling effectively
- describe validation techniques and procedures
- describe reliability, resiliency, and recoverability and how it can be achieved in software engineering
- describe CDI/UDI, why it is important and how it should be done
- apply parameter checking
- use Java exception handling
- use Python exception handling
- use C# exception handling
- use JavaScript exception handling
- use Java validation
- use Python validation
- use C# validation
- use JavaScript validation
- describe component trust including when and how to achieve trust of components
- describe how to reuse code effectively and defensively

Defensive Programmer: Cryptography

Objectives:

- describe basic cryptography concepts, cryptography types, and applications
- describe basic cryptography applications to confidentiality and integrity
- use Java Cryptography
- use Python Cryptography
- use C# Cryptography
- use JavaScript Cryptography

Defensive Programmer: Advanced Concepts

Objectives:

- describe session management techniques and secure session management
- define risk management and be able to apply risk management to software projects
- describe assertive programming and be able to implement assertions
- describe intelligible exceptions and be able to implement meaningful and actionable exception handling

Defensive Programmer: Code Samples

Objectives:

- implement Java filtering
- implement Python filtering
- implement C# filtering
- implement JavaScript filtering
- implement Java resilient code
- implement Python resilient code
- implement C# resilient code
- implement JavaScript resilient code
- implement Java recoverable code
- implement Python recoverable code
- implement C# recoverable code
- implement JavaScript recoverable code
- implement Java parameter checking
- implement Python parameter checking
- implement C# parameter checking
- implement JavaScript parameter checking
- implement validation in Java
- implement validation in Python
- implement validation in C#
- implement validation in JavaScript

Defensive Programmer: Secure Testing

Objectives:

- describe secure testing concepts including unit, integration, and regression testing
- apply secure unit testing including how it is done and who should do it
- apply effective and secure regression testing
- apply secure integration testing including when and who conducts integration testing
- use effective security metrics
- effectively track security bugs

Final Exam: Defensive Programmer

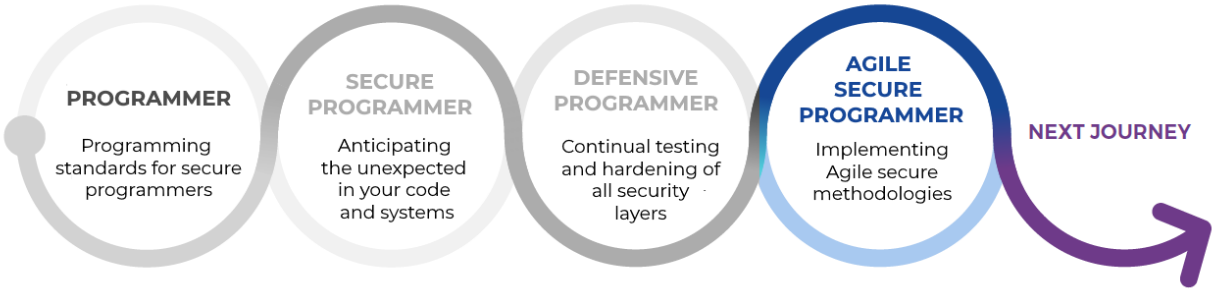
Objectives:

- apply defensive coding
- apply effective and secure regression testing
- apply exception handling effectively
- apply parameter checking
- apply secure integration testing including when and who conducts integration testing
- apply secure unit testing including how it is done and who should do it
- apply the Flaw Hypothesis Method
- define risk management and be able to apply risk management to software projects
- describe assertive programming and be able to implement assertions
- describe basic cryptography applications to confidentiality and integrity
- describe basic cryptography concepts, cryptography types, and applications
- describe CDI/UDI, why it is important and how it should be done
- describe component trust including when and how to achieve the trust of components
- describe how to reuse code effectively and defensively
- describe intelligible exceptions and be able to implement meaningful and actionable exception handling
- describe reliability, resiliency, and recoverability and how it can be achieved in software engineering
- describe secure testing concepts including unit, integration, and regression testing
- describe session management techniques and secure session management
- describe the first five CERT Top 10 secure coding practices - Validate input, Heed compiler warnings, Architect and design for security, Keep it simple, and the Default deny

- describe the last five CERT Top 10 secure coding practices - Adhere to the principle of least privilege, Sanitize data sent to other systems, Practice defense-in-depth, Use effective quality assurance techniques, and Adopt a secure coding standard
- describe the role of Six Sigma in producing better quality, secure programming
- describe validation techniques and procedures
- effectively track security bugs
- identify general defensive concepts
- identify intelligible exceptions
- implement C# filtering
- implement C# parameter checking
- implement C# recoverable code
- implement C# resilient code
- implement Java filtering
- implement Java parameter checking
- implement Java recoverable code
- implement Java resilient code
- implement JavaScript filtering
- implement JavaScript parameter checking
- implement JavaScript recoverable code
- implement JavaScript resilient code
- implement Python filtering
- implement Python parameter checking
- implement Python recoverable code
- implement Python resilient code
- implement secure integration testing including when and who conducts integration testing
- implement validation in C#
- implement validation in Java
- implement validation in JavaScript
- implement validation in Python
- use C# Cryptography
- use C# exception handling
- use C# validation
- use effective security metrics
- use Java Cryptography
- use Java exception handling
- use JavaScript Cryptography
- use JavaScript exception handling
- use JavaScript validation
- use Java validation
- use Open Source Security Testing Methodology Manual concepts
- use Python Cryptography
- use Python exception handling
- use Python validation

SECURITY JOURNEY

PROGRAMMER TO SECURE AGILE PROGRAMMER



Track 4: Agile Secure Programmer Engineer (duration: 2h 15m 9s)

Secure Agile Programming: Agile Concepts

Objectives:

- describe iterative software development
- differentiate between Agile and Waterfall
- describe Agile and security concepts
- create a secure Agile culture
- describe Scrum
- describe Lean software
- describe extreme programming
- describe rapid application development
- describe best practices for secure Agile development
- facilitate a secure organizational culture
- describe secure methods for Scrum

Secure Agile Programming: Agile Techniques

Objectives:

- gather Agile requirements
- define Agile techniques including iterative delivery and the use of user stories
- define Agile techniques including the daily standup meeting, pair programming, Scrum events, and planning poker
- describe Agile processes such as the Agile Unified Process and the use of sprints
- create a secure Agile software development lifecycle
- implement Disciplined Agile Delivery
- apply best practices for secure software development

Secure Agile Programming: Agile Modeling

Objectives:

- describe Agile modeling
- apply story-driven modeling
- use secure user stories
- use Specification by Example
- build secure user stories

Secure Agile Programming: Testing

Objectives:

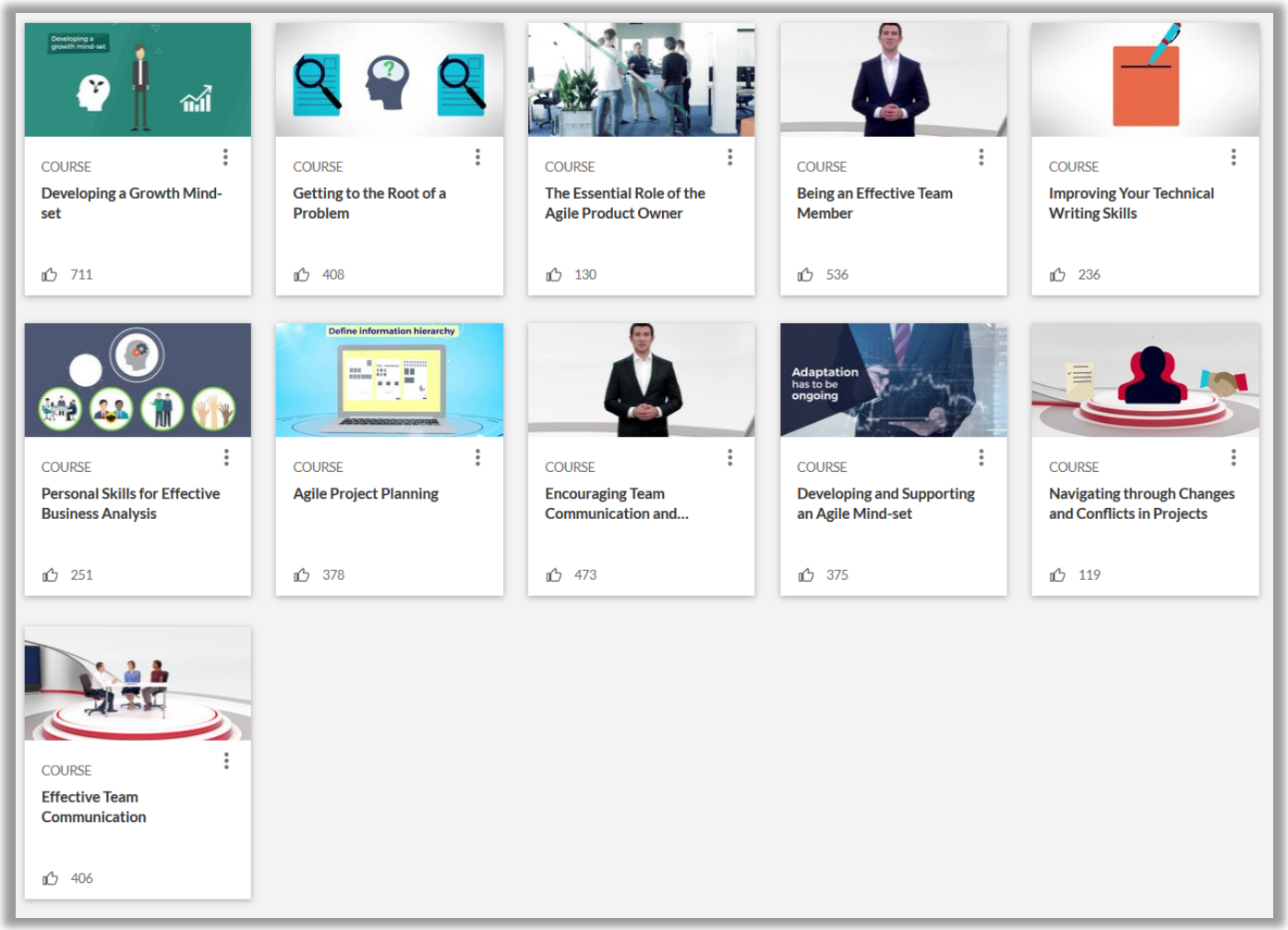
- describe Agile testing
- apply continual security testing
- integrate testing standards into Agile
- apply verification and validation for Agile programming
- integrate metrics into Agile programming
- configure bug tracking in an Agile environment
- conduct static code analysis
- implement continuous integration techniques

Final Exam: Secure Agile Programmer

Objectives:

- apply best practices for secure software development
- apply bug tracking in an Agile environment
- apply continual security testing
- apply story-driven modeling
- apply verification and validation for Agile programming
- build secure user stories
- collect Agile requirements
- compare between Agile and Waterfall
- conduct static code analysis
- configure bug tracking in an Agile environment
- create a secure Agile culture
- create a secure Agile software development lifecycle
- create secure user stories
- create secure user story
- define Agile techniques including iterative delivery and the use of user stories
- define Agile techniques including the daily standup meeting, pair programming, Scrum events, and planning poker
- define Agile techniques including the use of user stories
- describe Agile and security concepts
- describe Agile modeling
- describe Agile modeling
- describe Agile processes such as the Agile Unified Process and the use of sprints
- describe Agile testing
- describe best practices for secure Agile development
- describe extreme programming
- describe iterative software development
- describe Lean software
- describe rapid application development
- describe Scrum
- describe secure methods for Scrum
- differentiate between Agile and Waterfall
- facilitate a secure organizational culture
- gather Agile requirements
- identify Agile methods
- identify Agile modeling
- identify Agile processes such as the Agile Unified Process and the use of sprints
- identify Agile techniques including the daily standup meeting, pair programming, Scrum events, and planning poker
- identify rapid application development
- identify Scrum roles
- identify secure methods for Scrum
- implement Agile testing
- implement a secure Agile software development lifecycle
- implement best practices for secure software development
- implement continual security testing
- implement continuous integration techniques
- implement Disciplined Agile Delivery
- implement metrics into Agile programming
- implement Specification by Example
- implement story-driven modeling
- implement testing standards into Agile
- implement verification and validation for Agile programming
- integrate metrics into Agile programming
- integrate testing standards into Agile
- perform static code analysis
- promote a secure Agile culture
- promote a secure organizational culture
- recall Agile and security concepts
- recognize iterative software development
- recognize Lean software
- use secure user stories
- use Specification by Example

Business and Leadership for Secure Agile Programmers



Productivity Tools for Secure Agile Programmers

COURSE

Signing in & Setting up a Team

10

COURSE

Using the Conversation Tools

8

COURSE

Creating & Managing Projects

8

COURSE

Finding & Sharing Items

5

COURSE

Running Reports & Configuring Projects

5

COURSE

Using Basecamp for iOS

6

COURSE

Signing in & Navigating within Spaces

16

COURSE

Setting Up & Managing Spaces

13

COURSE

Working with Spaces

7

COURSE

Working with Team Members

37

COURSE

Configuring Spaces

7

COURSE

Sign-in & Setup

3

COURSE

Communication Tools

6

COURSE

Working with Groups

3

COURSE

Creating, Finding, & Sharing Information

4

COURSE

Configuring Convo

2

COURSE

The Convo iOS App

3

COURSE

Signing in & Setting Up

30

COURSE

Using Channels

18

COURSE

Private Messaging & Communication Tools

21

COURSE

Creating, Finding & Sharing Information

7

COURSE

Configuring Slack

25

COURSE

Using the iOS App

4

COURSE

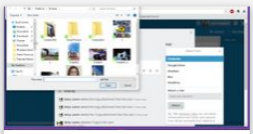
Sign-in & Setup

16

COURSE

Creating Teams & Boards

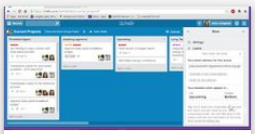
15



COURSE

Managing Cards

5



COURSE

Finding & Sharing Information

9

COURSE

Setting Up

33

COURSE

Posting & Reacting to Status Updates

25

COURSE

Using Groups

6

COURSE

Collaborating & Communicating

42

COURSE

Configuring Networks

16

COURSE

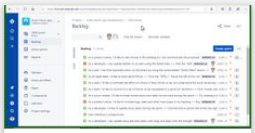
JIRA Cloud: Creating & Setting Up Projects

47

COURSE

JIRA Cloud: Configuring & Managing Boards

27



COURSE

JIRA Cloud: Planning & Working on a Software...

22

COURSE

JIRA Cloud: Reporting in Jira Software

22

COURSE

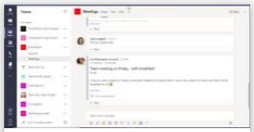
Microsoft Teams: Getting to know the application

691

COURSE

Microsoft Teams: Using Teams & Channels

561



COURSE

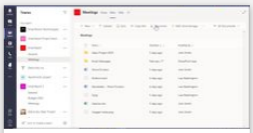
Microsoft Teams: Communicating via the App

510

COURSE

Microsoft Teams: Formatting, Illustrating & Reacting to...

422



COURSE

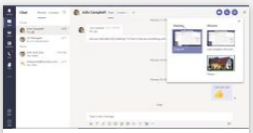
Microsoft Teams: Creating, Finding & Organizing Files

401

COURSE

Microsoft Teams: Working with Apps, Tabs & Wiki

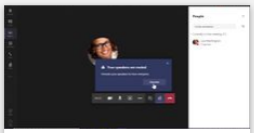
376



COURSE

Microsoft Teams: Making calls, Organizing Contacts ...

371



COURSE

Microsoft Teams: Creating, Joining & Managing Meetings

361

Bookshelf

BOOK

Engineering Safe and Secure Software Systems

2

BOOK

Secure Software Design

BOOK

Software Testing: A Self-Teaching Introduction

1

BOOK

Software Development, Design and Coding: With...

4

BOOK

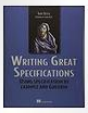
Software Modeling and Design

11

BOOK

Requirements Modelling and Specification for Service...

2



BOOK

Writing Great Specifications: Using Specification by...

1

BOOK

Software Testing: Concepts and Operations

22

BOOK

Cyber Risk Management: Prioritize Threats, Identify...

1

BOOK

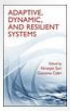
Asset Attack Vectors: Building Effective...

1

BOOK

Foundations of Coding: Compression, Encryption,...

2



BOOK

Adaptive, Dynamic, and Resilient Systems

BOOK

Secure and Resilient Software: Requirements, Te...

BOOK

The 7 Qualities of Highly Secure Software

1

BOOK

Core Software Security: Security at the Source

1

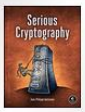
BOOK

Rootkits and Bootkits: Reversing Modern Malware...

2

BOOK

Hacking Exposed Malware and Rootkits: Security...



BOOK

Serious Cryptography: A Practical Introduction to...

1

BOOK

Cryptography and Network Security: A Practical...

21

BOOK

Quick Start Guide to Penetration Testing: With...

4

BOOK

Professional Penetration Testing: Creating and...

18

BOOK

Accelerate: The Science of Lean Software and DevOps:...

11

BOOK

Accelerate: The Science of Lean Software and DevOps:...

27

BOOK

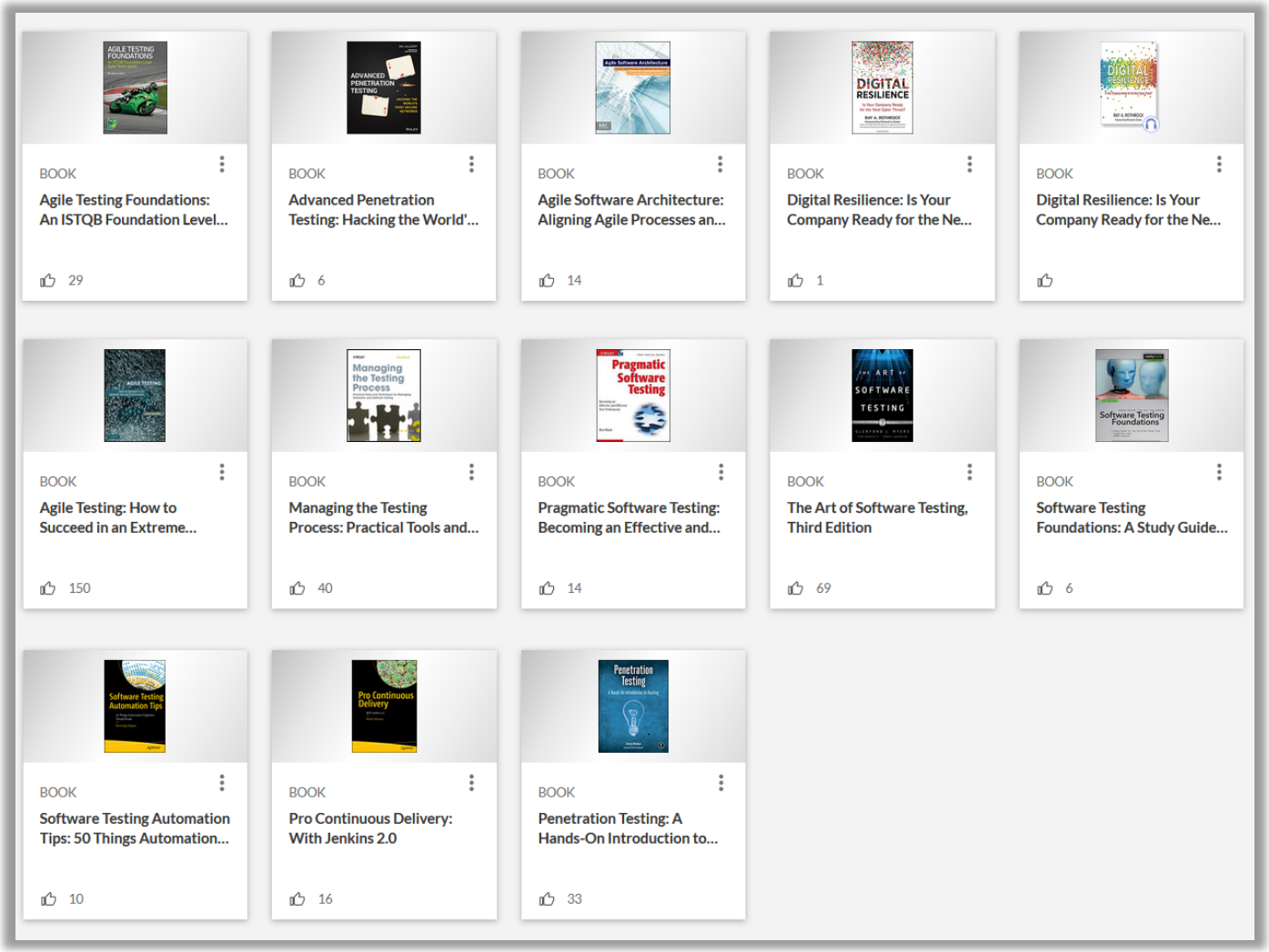
Disciplined Agile Delivery: A Practitioner's Guide to Agil...

2

BOOK

Agile Practices for Waterfall Projects: Shifting Processes...

51



FOLLOW US ON:

www.skilltech.pl

email: biuro@skilltech.pl

tel. +48 22 44 88 827

SkillTech
Technology hired for excellence