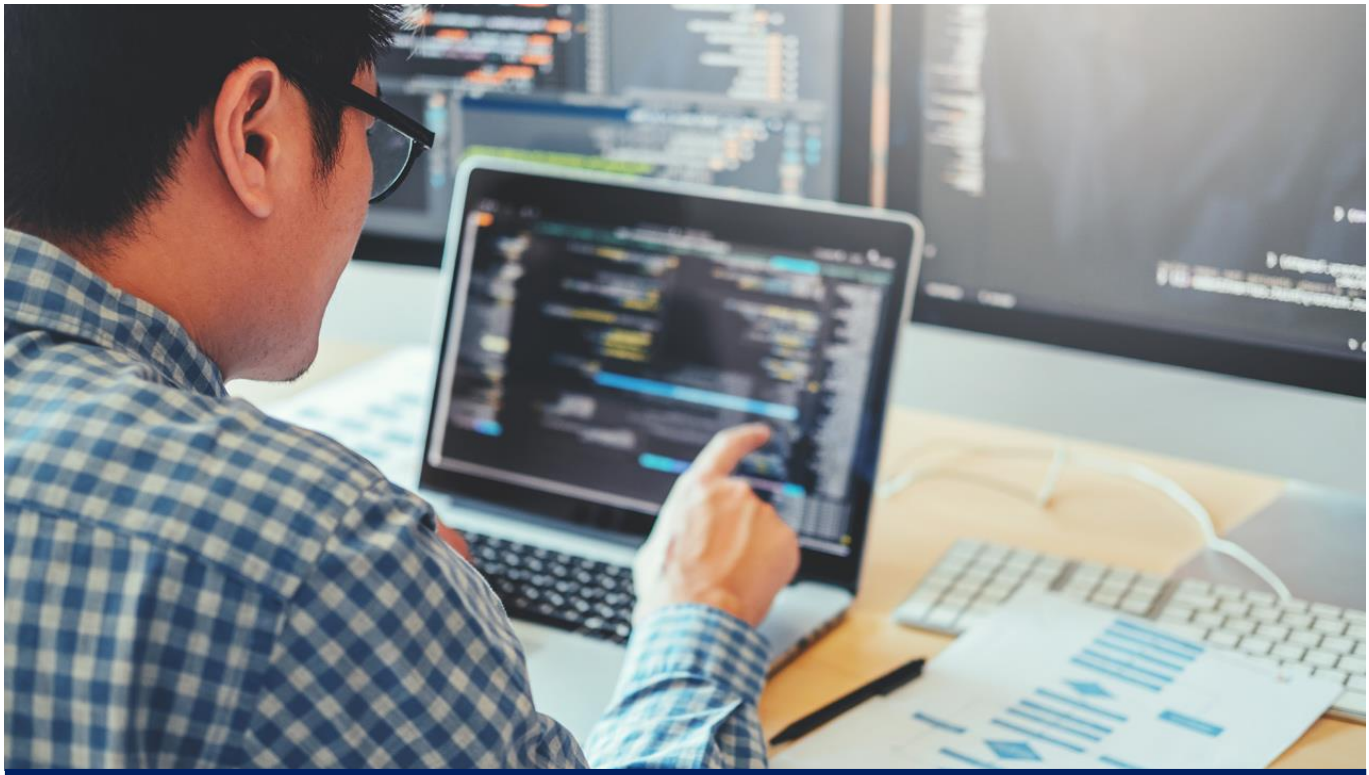


Programmer to Secure Agile Programmer

SKILLSOFT ASPIRE JOURNEY

skillsoft | percipio

Tracks

Track 1: Programmer

In this Skillsoft Aspire track of the Secure Agile Programmer journey, the focus will be on programming standards for secure programmers.

Explore 6 courses | 5h 37m 3s 1 lab | 8h

Track 2: Secure Programmer

In this Skillsoft Aspire track of the Secure Agile Programmer journey, the focus will be on security concepts, vulnerabilities, encryption, attacks and resiliency coding for secure programmers.

Explore 6 courses | 4h 26m 38s 1 lab | 8h

Track 3: Defensive Programmer

In this Skillsoft Aspire track of the Secure Agile Programmer journey, the focus will be on defensive concepts and techniques, cryptography, code sampling, secure testing, and advanced defensive pro...
View More

Explore 7 courses | 5h 50m 56s 1 lab | 8h

Track 4: Agile Secure Programmer

In this Skillsoft Aspire track of the Secure Agile Programmer journey, the focus will be on secure Agile programming concepts, techniques, modeling, and testing.

Explore 5 courses | 2h 58m 40s 1 lab | 8h

We recommend the following prerequisite skills:

- Familiar with software development concepts
- Familiar with Agile

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

Secure Programmer: Intro to Programming Standards

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

Secure Programmer: Software Design Techniques

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

Secure Programmer: Security Concepts

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

Secure Programmer: Vulnerabilities

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:

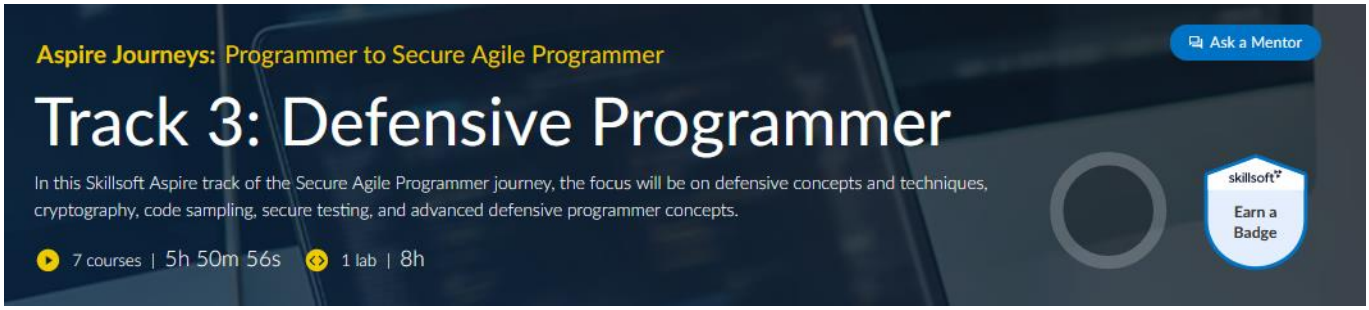
- apply C# secure coding to combat common code vulnerabilities
- apply JavaScript secure coding to combat SQL Injection Vulnerability
- code against buffer overflow attacks in C#
- code against buffer overflow attacks in Java
- code against buffer overflow attacks in Java - part 2
- code against buffer 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

Secure Programmer

Objectives:

- Perform Secure Programmer tasks such as minimizing SQL injection vulnerability, using OWASP Zap application to test an insecure web application, using Python to encrypt a data set and explore vulnerable code that can cause an overrun of buffer's boundary in Java, Python, C# and Javascript. Then, test your skills by answering assessment questions after preventing cross site scripting vulnerability, using Python to brute force a simple password, and creating resilient code in Python and C#.



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

Defensive Programmer

Objectives:

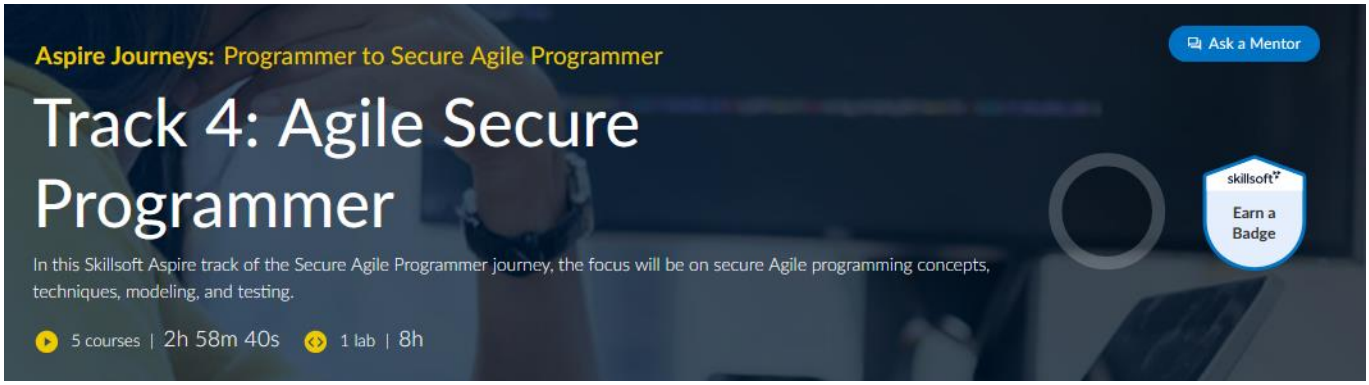
- Perform Defensive Programmer tasks such as implementing Java exception handling and JavaScript validation, and using Python to implement asymetric encryption. Then, test your skills by answering assessment questions after filtering data with Python, performing parameter checking with C#, developing recoverable code in Java and implementing secure testing.

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



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

Secure Agile Programmer

Objectives:

- Perform Secure Agile Programmer tasks such as performing manual and tool-based agile testing, implementing continual security testing and performing manual static code analysis. Then, test your skills by answering assessment questions after performing tool-based static code analysis, implementing bug tracking in Bugzilla and JIRA, and implementing continuous integration techniques.

Business & Leadership for Secure Agile Programmers

Optional

COURSE

Developing a Growth Mindset

2391

COURSE

Getting to the Root of a Problem

932

COURSE

The Essential Role of the Agile Product Owner

277

COURSE

Being an Effective Team Member

1653

COURSE

Improving Your Technical Writing Skills

514

COURSE

Personal Skills for Effective Business Analysis

665

COURSE

Agile Project Planning

1110

COURSE

Encouraging Team Communication and...

1547

COURSE

Developing and Supporting an Agile Mindset

1069

COURSE

Navigating through Changes and Conflicts in Projects

342

COURSE

Effective Team Communication

1229

Productivity Tools for Secure Agile Programmers

Optional

COURSE

Signing in & Setting up a Team

21

COURSE

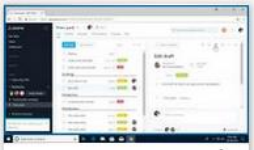
Using the Conversation Tools

20

COURSE

Creating & Managing Projects

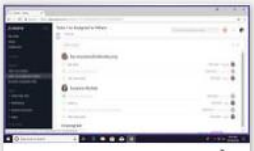
17



COURSE

Finding & Sharing Items

10



COURSE

Running Reports & Configuring Projects

11

COURSE

Signing In & Setting Up

28

COURSE

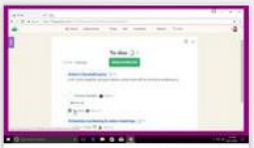
Using the Team Communication Tools

82

COURSE

Setting Up & Tracking Projects

25



COURSE

Managing your Project Tasks & Assets

20

COURSE

Using the Calendar Tools

22

COURSE

Using Basecamp for iOS

21

COURSE

Signing in & Navigating within Spaces

40

COURSE

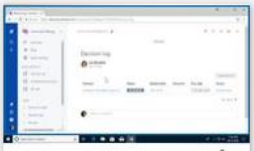
Setting Up & Managing Spaces

33

COURSE

Working with Space

27



COURSE

Working with Team Members

76

COURSE

Configuring Spaces

20

COURSE

Sign-in & Setup

18

COURSE

Communication Tools

23

COURSE

Working with Groups

14

COURSE

Creating, Finding, & Sharing Information

12

COURSE

Configuring Convo

6

COURSE

The Convo iOS App

13

COURSE

Sign-in & Setup

65

COURSE

Creating Teams & Boards

44

COURSE

Managing Cards

22

Productivity Tools for Secure Agile Programmers Continued

COURSE

Finding & Sharing Information

20

COURSE

Setting Up

61

COURSE

Posting & Reacting to Status Updates

47

COURSE

Using Groups

14

COURSE

Collaborating & Communicating

84

COURSE

Configuring Networks

25

COURSE

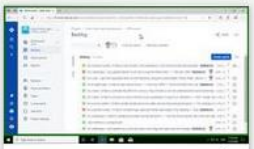
Creating & Setting Up Projects in Jira Cloud

166

COURSE

Configuring & Managing Boards in Jira Cloud

106



COURSE

Planning & Working on a Software Project in Jira...

85

COURSE

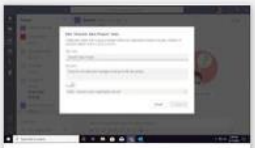
Reporting in Jira Software

83

COURSE

Getting to know the application

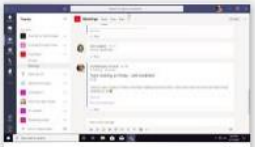
1012



COURSE

Using Teams & Channels

746



COURSE

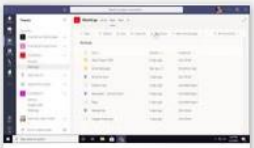
Communicating via the App

731

COURSE

Formatting, Illustrating & Reacting to Messages

543



COURSE

Creating, Finding & Organizing Files

536

COURSE

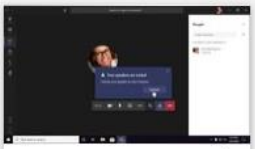
Working with Apps, Tabs & Wiki

465

COURSE

Making calls, Organizing Contacts & Using Voicemail

453



COURSE

Creating, Joining & Managing Meetings

462

Optional Resources

Optional

LAB

Programmer to Secure Agile Programmer Sandbox

HTML / CS / Javascript

1

LAB

Front-End Developer Sandbox

Front-End Development 2020

7

LAB

C Programming Sandbox

C 18

12

Bookshelf

Optional

BOOK

Engineering Safe and Secure Software Systems

2

BOOK

Secure Software Design

BOOK

Software Testing: A Self-Teaching Introduction

2

BOOK

Software Development, Design and Coding: With...

9

BOOK

Software Modeling and Design

12

BOOK

Requirements Modelling and Specification for Service...

BOOK

Writing Great Specifications: Using Specification by...

BOOK

Software Testing: Concepts and Operations

BOOK

Cyber Risk Management: Prioritize Threats, Identify...

BOOK

Asset Attack Vectors: Building Effective...

BOOK

Foundations of Coding: Compression, Encryption,...

2

BOOK

Adaptive, Dynamic, and Resilient Systems

BOOK

Secure and Resilient Software: Requirements, Te...

BOOK

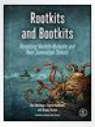
The 7 Qualities of Highly Secure Software

1

BOOK

Core Software Security: Security at the Source

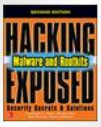
1



BOOK

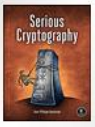
Rootkits and Bootkits: Reversing Modern Malware...

5



BOOK

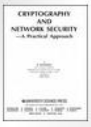
Hacking Exposed Malware and Rootkits: Security...



BOOK

Serious Cryptography: A Practical Introduction to...

3



BOOK

Cryptography and Network Security: A Practical...

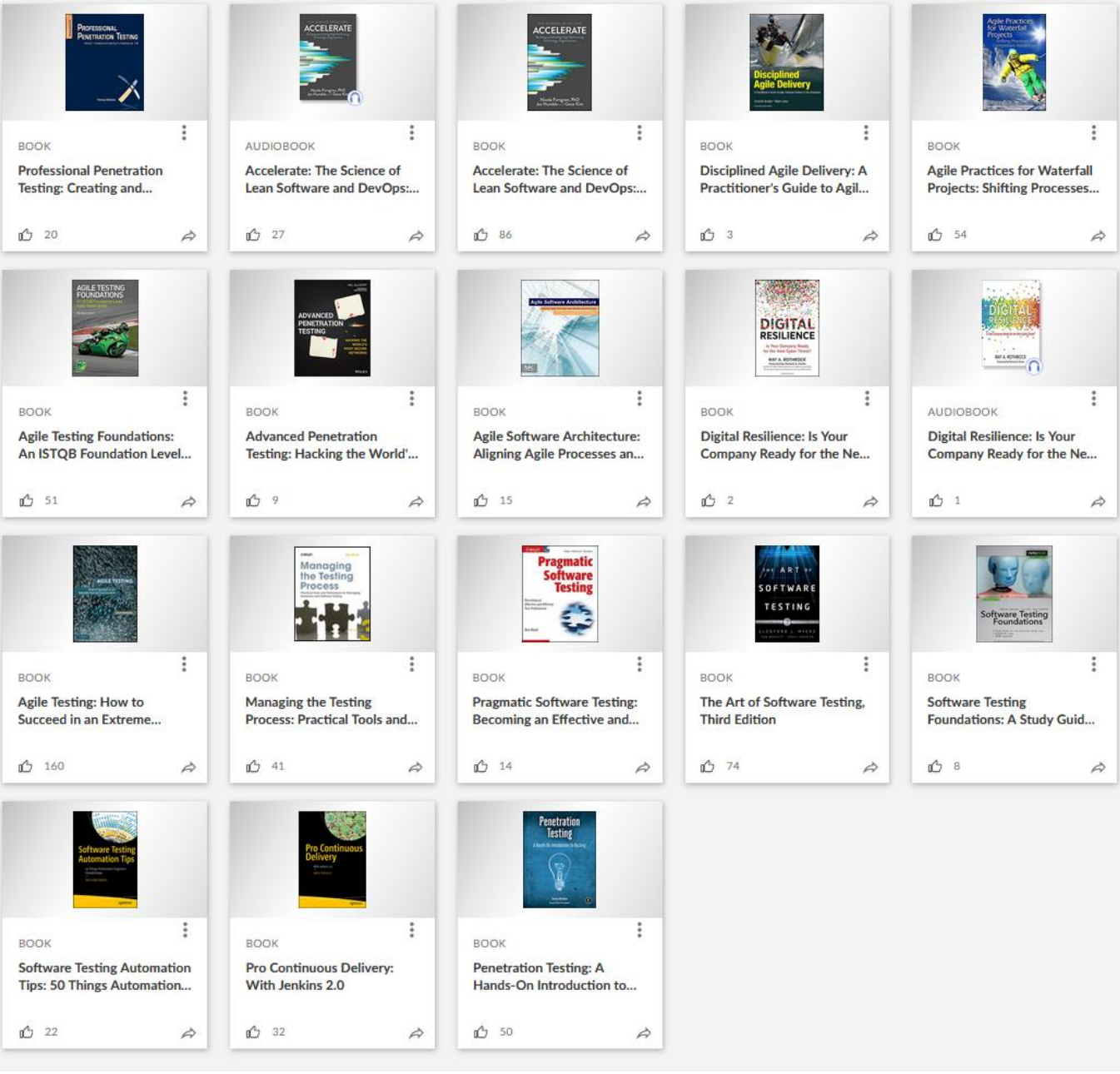
27

BOOK

Quick Start Guide to Penetration Testing: With...

6

Bookshelf Continued



FOLLOW US ON:

www.skilltech.pl

email: biuro@skilltech.pl

tel. +48 22 44 88 827

SkillTech
Technology hired for excellence