SCRUM ALLIANCE®
CERTIFIED SCRUM DEVELOPER®
Learning Objectives

INTRODUCTION

Purpose

The Certified Scrum Developer® course is aimed at software developers (programmers) who are building software in a Scrum environment. The goal is to expose students to the most important tools and techniques that need to be applied in order to build good software in the iterative and incremental fashion that Scrum requires. These ideas are central to the entire field of Agile software development.

The following CSD® Learning Objectives (LOs) are the basis for evaluating and confirming candidate courses.

Scope

The CSD Learning objectives cover the critical skills in coding, testing, refactoring, continuous integration, and other related topics that are needed for successful team software development in the Scrum style.

Prerequisites

The candidates for this course should generally be programmers in a conventional programming language such as Java, C#, Swift, C++, Ruby, and so on. The course involves actual programming in a team environment. Some courses offer student slots to testers, who should have some experience with test automation. Some also accommodate Product Owners, who will help direct the iterations that the programmers and testers experience during the course. This notwithstanding, the course is a programming course, not a survey, overview, or discussion. Actual code will be written, tested, and reviewed by the instructors.

Students should also arrive at the course familiar with:

- Scrum Guide http://scrumguides.org
- Agile Manifesto 4 values and 12 principles http:// www.agilemanifesto.org
- The Scrum Values https://www.scrumalliance.org/why-scrum/corescrum-values-roles

CSD Courses are taught by authorized Scrum Alliance REP® organizations whose courses and instructors have been reviewed and approved by Scrum Alliance®.

Organization of Candidate Courses

Conforming courses may contain more subjects than are covered in the following Learning Objectives (LO), but the LOs are a minimum requirement. For example, a course may also cover Scrum applied to non-software product domains, other non-Scrum-related frameworks, and so on. Conforming courses can be organized in any way; we do not expect courses to be organized according to these LOs or major sections. One LO may be addressed via topics dispersed within different course areas.

REQUIREMENTS FOR SCRUM DEVELOPER CERTIFICATION

To earn a CSD, we require at least five days of formal training taught by a Scrum Alliance Registered Education Provider® (REP) and a Scrum Alliance Authorized Instructor.

There are currently two tracks to achieve CSD certification (detailed below). The courses for both tracks can be taken in any order. Once you complete a track, you will receive notification to accept your CSD license, and the CSD certification will be added to your profile. Your certification is valid for two years.

TRACK 1

TRACK 2

LEARNING OBJECTIVES

By the completion of the CSD course, the learner will be able to perform test-driven development, acceptance test-driven development, refactoring, and continuous integration in a course-provided sandbox situation. The learner will be familiar with the terms and practices involved, and with why these practices are important. They will have taken the first steps toward becoming expert in Scrum-style iterative and incremental development.

Introduction to Scrum (1 Day)

Agile Values - Study of the Agile values highlighted in the Agile Manifesto:

- Define simplicity, communication and feedback (in relation to the *Agile Values* that drive Scrum)
- Describe "individuals and interactions over process and tools."
- Describe "working software over comprehensive documentation."
- Describe "customer collaboration over contract negotiation."
- Explain, using examples, "responding to change over following a plan."

Scrum - Study of Scrum principles and practices, including but not limited to, the following key concepts:

- Define Scrum roles, activities, and artifacts.
- Outline the process of working with a product backlog and a sprint backlog.
- Define a sprint.
- Describe the process of defining "Done."

CSD Technical (3 Days)

Architecture and Design - Study of architecture and design, focusing primarily on the principles that better ensure testability and ease of refactoring, including but not limited to, the following concepts:

- Outline at least three principles of architecture in an Agile environment.
- Design at least one practice on an Agile team.
- Outline at least two principles that enable testability and ease of refactoring.

Collaboration - An in-depth look at the way Agile teams work together. This might include, but not limited to, the following concepts:

- Describe "working together as one team."
- Describe how to "include the customer" in the process.
- Define pair programming.

Test-Driven Development - A study of test-first development, including but not limited to, the following concepts:

- Describe Test-Driven Development (TDD) as a design approach.
- Review the steps of the red-green-refactor cycle.

- Explain, using examples, at least three-unit testing principles and practices.
- Outline five qualities of a good test.
- Describe how to measure test effectiveness.

Refactoring - An introduction to the practice of refactoring, including but not limited to, the following concepts:

- Describe when to refactor.
- Outline refactoring for maintainability.
- Describe refactoring patterns.

Continuous Integration - An introduction to the key practices of continuous integration, including but not limited to, the following key concepts:

- Define a single command build.
- Summarize how to create a build that is automated self-testing and fast.
- Describe the importance of a single-source repository.
- Define increasing visibility and automating deployment.

CSD Glossary

Continuous Integration (Fowler, 2006) - a software development practice where members of a team integrate their work frequently, usually at least daily, and even multiple integrations per day. Each integration is verified by an automated build (including test) to quickly detect errors.

Pair Programming (Williams & Kessler, 2003) - style of programming in which two programmers work side by side at one computer, continually collaborating on the same design, algorithm, code, or test.

DRIVER- types and writes the design

NAVIGATOR- observes the driver and looks for defects

Refactoring (Fowler, 2014) - the process of changing a software system in such a way that it does not change the external structure of the code, only the internal. When you refactor, you are simply improving the design of the code after it has been written.

Single Command Build (Biedenharn, personal communication, October, 2018) - describes the goal of automating the process of turning the code into the fully functional program so that it functions with a single command, thus not requiring a multi-step procedure.

Single Source Repository (Fowler, 2006) - allows teams to keep the main project codebase in one easy-to-access location. This allows for easy version control; and, for any new team members to get up and running easily.

Test-Driven Development (Beck, 2014) - development driven by automated tests.

1. Red: Write a test that doesn't work.
2. Green: Make the test work quickly.
3. Refactor: Clear up all duplication made in process to make the test work.

Unit Testing - testing the smallest units possible to prove what was written is true.

TYPES OF UNIT TESTING FRAMEWORKS:

- Xunit - C# and .Net
- Junit - Java
- Nunit - .Net
- PyUnit - Python
- Cppunit - C++
- Ocunit - Objective C

REFERENCES

Beck, K. (2014). Test-driven development by example. Boston: Addison-Wesley.

Fowler, M., Beck, K., Brant, J., Oodyke, W., & Roberts, D. (2014). REFACTORING: Improving the design of existing code. Reading, MA: Addison-Wesley.

Fowler, M. (2006, May 01). Continuous Integration. Retrieved from martinfowler.com

Williams, L., & Kessler, R. (2003). Pair programming illuminated. Boston, MA: Addison-Wesley.