Competency Evaluation

Software Developer Apprentice

Developed by AlbanyCanCode Inc. Funding Provided by State University of New York and SUNY Schenectady

Rationale

Overview

This document is broken down by the major Work Processes outlined in Appendix A of the Software Developer Apprenticeship Standards. It provides both explanation/justification for various decisions in how to assess apprentices, as well as providing a core written assessment tool. This assessment can be further adapted to different platforms that allow for automated testing of coding sections and online administering of the evaluation. Care has been taken to make the assessment programming language agnostic to account for the many different job placements that apprentices may have.

Workplace Basics

The Workplace Basics standards A1 - A5 would be best evaluated by the employer hosting the apprentice. While not an assessment in and of itself, testimonial from employers would likely be the most accurate indication of employees workplace skills. These standards could also be assessed by standard workforce training certifications that are available, or they could be assessed via an essay prompt.

Defining Projects/Designing Software

Apprentices are asked to respond to several prompts and from them identify key requirements, user actions, and user stories based on the project descriptions. There is no mandatory structure to their response - however, their response should be easy to understand, regardless of their chosen approach.

From the same prompts used for Defining Projects, apprentices are asked to translate those user requirements into technical requirements. These technical requirements should focus on necessary hardware considerations, choosing and justifying appropriate data structures, and planning to mitigate potential security issues.

Writing Software Code

Because different apprentices have likely learned different programming languages, the following questions have been designed to be as language-agnostic as possible. This also means that the apprentice *must declare the language they would like to answer in* before going through this section.

Earlier questions are intended to test fundamental syntax knowledge in the apprentices chosen language, and address Standards D1-D2, D10, D12

Later questions will ask apprentices to write small code samples. These exercises have been chosen to allow apprentices to demonstrate the array of skills expected in Standards D2-D9, D11 in a more practical way that attempting to assess each individually. For example, the Fibonacci Exercise will likely assess:

- Arithmetic Operations
- Variables and Assignment
- Value Comparisons
- Logical Truth
- Loops
- Conditionals
- Custom Functions

Similarly, the Sorting Algorithm exercise will evaluate many of the above topics, as well as determining the apprentices familiarity with other computer science principles such as computational complexity even if they are unfamiliar with the academic definitions.

Questions are designed to be assessed in the order provided in the Assessment section.

In regards to implementing the technical portion of the assessment, apprentices should be allowed time to work on the code in a reasonable development environment. A reasonable development environment should include:

- Common tools, such as code completion and syntax/error highlight
- Ability to test code, modify, and test again
- Access to standard libraries for the language of choice, such as Math libraries

A resource such as https://repl.it/languages would be an example of a reasonable development environment capable of being used for a large range of common programming languages.

Building GUI/Applications

Due to the broad range of possible placements for apprentices, and how much the process of building graphic interfaces varies between different languages, Standard E1 is difficult to assess in this format. It very much depends on the specific software work an apprentice has done (for

example, a developer working in SQL databases may never work in developing a graphic interface in those projects).

Standard E2 is valuable across all project types, and across all levels of skill. Assessment focused on determining how the apprentice approaches testing their applications, and if they are aware of and concerned about the issues that can occur when their application is deployed on different platforms.

Testing, Deployment, and Maintenance

Due to the broad nature of Standards F1-F3, it is difficult to use questions to assess them. That said, Standard F3 can be combined with Standard A5 to assess the apprentices ability to simplify and explain technical concepts. By providing students with a wide range of technical concepts to explain in layman's terms, we can both keep apprentice to a particular level of complexity and avoid having the question misalign entirely with their apprenticeship.

Assessment

Defining Projects/Designing Software

Writing Prompts

Read through the following descriptions of different systems. Based on these descriptions, develop and describe:

a. A list of customer requirements for the system/product.
b. A list of potential actions users may take.
c. A list of technical or system requirements, such as how data will be structured.
d. Any potential hardware requirements or limitations.
e. Potential vulnerabilities in the system you describe.

Your lists do not need to be comprehensive given time constraints, but a recommendation would be 10 items. They should be self-consistent, and proposed technical/system requirements should be realistic. Hardware and vulnerability concerns should be justified in some way.

Tic-Tac-Toe

> A client has asked you to make a Tic-tac-toe game for them. At this time, they haven't specified where they would like the game to be able to run - in the browser or as a desktop application. A game of tic-tac-toe consists of the following:
>
> **1.** The game is played on a grid that's 3 squares by 3 squares.
>
> **2.** You are **X**, the opponent is **O**. Players take turns putting their marks in empty squares.
>
> **3.** The first player to get 3 of her marks in a row (up, down, across, or diagonally) is the winner.

4. When all 9 squares are full, the game is over. If no player has 3 marks in a row, the game ends in a tie.

Automated Teller Machine

Base your requirements on the following user experience:

"It's payday, and Joseph needs to deposit his paycheck with his bank. He ended up leaving work late today, and can't make it to the bank before it closes. He needs to pay his upcoming bills, so he chooses to use the ATM instead.

He identifies himself to the ATM using his bank card, and then inserts the check into the machine. He confirms the amount of the check is $1000. He then specifies he wants to transfer $100 from checking to saving, and would also like to withdraw $50 in cash from checking. He indicates that we would like to get the current balances, and would like printed record of this transaction. He takes his $50 cash and transaction record which includes the current balances on his accounts."

Programming Knowledge & Skills

1. List and define 3 primitive data types that are consistent across languages.
2. What are some limitations of the different number data types?
3. In <LANGUAGE OF CHOICE>, perform the following conversions/type casts:
    a. Integer to String
    b. String to Integer
    c. Integer to Float
    d. Float to String

4. **Fibonacci Numbers**

The Fibonacci numbers are the numbers in the following integer sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

Starting with the numbers 0 and 1, the next number in the sequence is the sum of the preceding two. For example:

```
0 + 1 = 1 #Third number
1 + 1 = 2 #Fourth number
2 + 1 = 3 #Fifth number
etc...
```

In <LANGUAGE OF CHOICE>, write a program/function that does the following:

a. Prints out the n-th Fibonacci number. For example, the 10th Fibonacci number is 55.
b. Make sure to consider and handle possible errors.
c. Discuss the memory use of the program you wrote. This does not need to be in great detail, but you should address the limitations of your program.

5. **Custom zip() Function**

The zip() function in several programming languages takes two or more lists, arrays, or other iterable objects, and combines them into a new list. With two lists, for example:

```
list1 = ["a", "b"]
list2 = [1, 2]
newList = zip(list1,list2)

print(newList)
>>> [ ["a", 1], ["b", 2] ]
```

a. In <LANGUAGE OF CHOICE> write a custom zip() function which takes only two lists as arguments, and returns the zipped result as shown above.
b. Make sure to consider edge cases, such as the lists having different lengths.

6. A variety of algorithms exist for sorting lists of data. In <LANGUAGE OF CHOICE>, implement a sorting function called ***sort()*** which takes a list of integers and arranges them in ascending order (smallest to largest). An example using Python's sorted() function is below:

```
unsorted_list = [45, 3, 1, 7, 2, 67]
sorted_list = sorted(unsorted_list)

print(sorted_list)
>>> [1, 2, 3, 7, 45, 67]
```

   Assume that the list of numbers to be sorted is a random list of integers from 0 to 1000, with each integer being equally likely. You do not know the length of the list.

   Discuss the performance of the algorithm you wrote, in terms of memory use and performance time.

7. In <LANGUAGE OF CHOICE>:

   a. Write out a small script importing additional libraries, modules or functions. You do not need to refer to actual libraries/modules/etc.

   b. What are some potential concerns when importing external libraries?

Testing and Quality Assurance

1. From your experiences, describe how your workplace handles testing and quality assurance of projects.
   a. Do they follow a particular methodology?
   b. What has your role in the process been?
   c. Why is code testing and quality assurance important?
2. What possible issues are there when supporting an application on different platforms? How can these issues be addressed or handled in an example scenario? This could be different desktop operating systems (if developing desktop apps), different mobile operating systems (if developing mobile apps), or different web browsers (if developing web apps).

Communicating Technical Ideas

1. Select a technical concept from the following list[1]:
    a. RESTful APIs
    b. Relational Databases
    c. Encryption
    d. Machine Learning/AI

    As best you can, describe that concept using minimal technical terminology. For example, describe the concept as if you were discussing this with a client who has no education in programming or computer science. You can assume a basic level of computer knowledge with a common operating system.

2. Describe the software development lifecycle as it applies to the work you have done, such as if your work follows a waterfall or agile methodology, or if it is a hybrid system. Your answer should address potential benefits and shortcomings of the methodology.

[1] This list can be modified to allow for a more diverse pool of apprentice background and job experiences.

Writing Prompt Rubrics

The writing prompt rubrics below run on a 1 - 4 scale, with some categories covering less of the scale due to a more binary demonstrable criteria (‘either they did or they didn’t”).

The current ‘total points’ that an apprentice can earn from the writing prompts is 46, and a minimum of 0. At this time, what would be considered a ‘passing score’ using these rubrics is to-be-determined. Some tuning will be required to hone in on what a fair passing score would be, when combined with the technical portion of the assessment.

Developed by AlbanyCanCode Inc. Funding Provided by State University of New York and SUNY Schenectady

Defining Project Requirements

Automated Teller Machine

Criteria	3	2	1	0
Customer Requirements	Identifies important customer requirements, such as those necessary to perform the key function of the product.	Identifies a good number of meaningful customer requirements, but fails to list key requirement OR Is not detailed in describing key requirements.	Identifies few requirements, or only identifies vague requirements without providing justification.	Fails to identify customer requirements, or incorrectly includes user actions or technical requirements as customer requirements.
Potential User Actions	Identifies a substantial number of potential user actions (approx. 5+).	Most user actions are covered, but the listed actions could be broken down into more specific actions.	Identifies few user actions, only covering a small sections of possible actions OR provides only generic actions for any system.	Fails to identify any user actions, or incorrectly includes customer or technical requirements as user actions.
Technical Requirements	Identifies a substantial number of technical requirements (approx. 8+) for the software. Additional	A substantial number of technical requirements are described, but the technical requirements could be broken down into more detailed/specific needs.	Identifies few requirements, or only identifies vague requirements without providing justification.	Fails to identify customer requirements, or incorrectly includes user actions or customer requirements as technical requirements,
Hardware Considerations		Identifies possible hardware needs, and explains why the need exists as related to the prompt.	Provides possible hardware considerations, but they do not describe why they are applicable to the prompt (if the need is not obvious).	Makes no comment on potential hardware considerations OR Does not justify why there are no/minimal hardware concerns.
Security Considerations		Discusses the potential security concerns of the system, and provides justifications for why they are a concern and potential ways of limiting/resolving them.	Discusses potential concerns, but they are not described in the context of the prompt OR They provide no ideas/examples of how to limit/resolve the concerns.	Fails to recognize any security concerns, OR Listed concerns not relevant to the prompt. OR Does not justify why there are no security concerns.

Max Score: 13

Tic-Tac-Toe

Criteria	3	2	1	0
Customer Requirements	Identifies important customer requirements, such as those necessary to perform the key function of the product.	Identifies a good number of meaningful customer requirements, but fails to list key requirement OR Is not detailed in describing key requirements.	Identifies few requirements, or only identifies vague requirements without providing justification.	Fails to identify customer requirements, or incorrectly includes user actions or technical requirements as customer requirements.
Potential User Actions	Identifies a substantial number of potential user actions (approx. 5+).	Most user actions are covered, but the listed actions could be broken down into more specific actions.	Identifies few user actions, only covering a small sections of possible actions OR provides only generic actions for any system.	Fails to identify any user actions, or incorrectly includes customer or technical requirements as user actions.
Technical Requirements	Identifies a substantial number of technical requirements (approx. 8+) for the software. Additional	A substantial number of technical requirements are described, but the technical requirements could be broken down into more detailed/specific needs.	Identifies few requirements, or only identifies vague requirements without providing justification.	Fails to identify customer requirements, or incorrectly includes user actions or customer requirements as technical requirements,
Hardware Considerations		Identifies possible hardware needs, and explains why the need exists as related to the prompt.	Provides possible hardware considerations, but they do not describe why they are applicable to the prompt (if the need is not obvious).	Makes no comment on potential hardware considerations OR Does not justify why there are no/minimal hardware concerns.
Security Considerations		Discusses the potential security concerns of the system, and provides justifications for why they are a concern and potential ways of limiting/resolving them.	Discusses potential concerns, but they are not described in the context of the prompt OR They provide no ideas/examples of how to limit/resolve the concerns.	Fails to recognize any security concerns, OR Listed concerns not relevant to the prompt. OR Does not justify why there are no security concerns.

Max Score: 13

Criteria	3	2	1	0
How Testing is Handled	If a particular methodology is used, the apprentice describes the process in detail with correct use of industry terminology If no particular methodology is used, the apprentice described how they test code in detail.	Described an approach to testing with mostly correct industry terminology, and includes how they would fit into the QA process as well as it's value.	Can describe an approach to code and software testing, but cannot describe why it is valuable or how they are a part of it.	Has no understanding of the value of software testing, or has no demonstrable experience with the process, that they can describe.
Recognizing Platform Support Issues	The apprentice both identifies the possible issues, as well as clearly describing why those issues occur in the system they choose	Identifies possible platform support, but is vague or provides incomplete descriptions for why it applies to their chosen development scenario.	Can describe platform support issues, but does not describe why or when they occur in any way.	Cannot identify common platform support issues for any software development scenario.
Resolving/Managing Cross-Platform Issues		Answers includes possible ways of resolving common cross-platform issues, dependent on the scenario they chose.	Identifies methods for resolving/managing cross-platform issues, but does not explain how it solves the issue or why	Proposes no methods for resolving cross-platform issues, specific or otherwise.

Max Score: 8

Communicating Technical Ideas

Technical Concept Explanation

Criteria	3	2	1	0
Use of Technical Terminology (and minimizing 'jargon')	Minimizes the use of 'jargon', and defines any technical terminology they do use in simplified terms first. AND Communicates using analogies, examples, and other rhetorical devices.	Minimal use of jargon, and most jargon is described and explained when it comes upt OR Minimal examples, analogies etc. hinder the reader's ability to interpret the concept	Includes some jargon without explanation or description, or described technical terminology with other technical terms.	Does not define technical terminology they use. OR Defines terms in a fundamentally incorrect way.
Correct Description of Technical Concept	Describes the major principle or key features of the selected technical concept correctly AND Avoids unnecessary details that may confuse others.	Correctly describes the core concept, but includes some small details that may be incorrect without affecting the overall description.	Mostly correct description of the core concept chose, but key aspects of it are explained incorrectly OR Key aspects of the concept are missing entirely, leaving a 'hole' in the understanding of the reader	Describes the technical concept entirely incorrectly.

Max Score: 6

Software Development Lifecycle

Criteria	3	2	1	0
Definition of the Software Development Lifecycle	Describes their workplace methodology in detail AND Describes how that methodology works for a developer in the day-to-day	Described a chosen methodology accurately	Provides a basic definition of a methodology, but does not go beyond the textbook definition to describe how the process works.	Cannot identify or define any methodologies used for managing the software development process.
Strengths of Weaknesses of the Process	If aware of or reflects on several strengths and weaknesses of software development methodologies, as they relate to the developers, the clients, and the product itself. Describes these pros/cons clearly.	Identifies strengths and weaknesses, but does not look at the overall process. For example, they may focus only on the issues a methodology causes for the developers.	Identifies a strength or weakness, but does not make clear why the methodology in question causes that benefit/concern.	Identifies no strengths/weaknesses.

Max Score: 6

Programming Knowledge & Skills Example Solutions

1. Basic Data Types

    a. String or Char
    b. Integer Numbers
    c. Floating Point and/or Fixed Point Numbers

1pt scored for each correct data type listed
Total → 3 points

---

2. What are some limitations of the different number data types?

Answers should include - but are not limited to - some of the following ideas:

- Integer number sizes are limited by the amount of memory that can be allocated to it. For example, an 8-bit unsigned integer is limited to the numbers between 0 and 255
- Trying to store a number outside the range of the allowed integer type results in an *overflow* and can cause errors
- Integers can be either signed or unsigned, and which they are affects the range of allowed numbers (and 8-bit signed integer is limited to -128 to 127
- Floating-point numbers have a limited precision, and are only an approximation of the decimal number they represent
- Doing arithmetic with floating-point numbers can result in errors, such as 0.2 + 0.1 = 0.30000000000000004
- Floating-point numbers are also limited in size based on allowed memory allocation (with a common limit being 3.4028235×10^{38})
- Other correct answers can be accepted at the discretion of the assessor.

2pts for identifying a limitation of integer number types
2pts for identifying a limitation of floating-point number types
Total → 4 points

For the following questions, links have been provided to external, online resources which contain examples of how to perform various actions in the syntax of a large number of different programming languages.

A discussion of possible ways of administering and evaluating the technical section of the assessment is included in the Rationale section of the assessment document itself. Most importantly, answers must be assessed on:

1) Does the program or answer work in the LANGUAGE OF CHOICE?
2) Does the program work for the standard-use case provided in the question?

---

3. In <LANGUAGE OF CHOICE>, perform the following conversions/type casts:
    a. Integer to String
    b. String to Integer
    c. Integer to Float
    d. Float to String

Type-Casting Syntax in Many Languages

1pt for every correct type-cast
Total → 4 points

---

4. Fibonacci Numbers Examples
    a. Fibonacci Sequence Examples in Many Languages
5. Custom zip() Function Examples
    a. Appendix A: Custom Zip() Functions
6. Custom sort() Function Examples
    a. Sort Function Examples in a Variety of Languages
7. Importing Functions & Concerns
    a. Import Syntax in Most Languages

1pt for a function that executes with no errors
3pts for a function that executes the performed function in the standard-use case
Total → 16pts for all coding tasks

TOTAL OVERALL: 37 points

Appendix A: Custom Zip() Functions (WORK IN PROGRESS)

Python:

```
def zip(list1, list2):
 result = []
 shorterList = list1 if len(list1) < len(list2) else list2
 for i in range(len(shorterList)):
   result.append([list1[i],list2[i]])

 return result
```

JavaScript:

```
function zip(list1, list2) {
 result = [];
 shorterList = list1.length < list2.length ? list1 : list2;

 for (let i=0; i< shorterList.length; i++) {
   result.push([list1[i],list2[i]])
 }
return result;
}
```

C++:
StackOverflow Solution (to be tested)

Java:
StackOverflow Solution (to be tested)
Solution Using Non-Standard Libraries

SOFTWARE DEVELOPER
(Competency-Based)

APPENDIX A
D.O.T. CODE 030.062-010
O*NET CODE 15-1131.00

Competency/performance-based apprenticeship occupations are premised on attainment of demonstrated, observable and measurable competencies in lieu of meeting time-based work experience and on-the-job learning requirements. In competency/performance-based occupations apprentices may accelerate the rate of competency achievement or take additional time beyond the approximate time of completion.

This training outline is the current standard for Work Processes and Related Instruction. Changes in technology, regulations, and safety/health issues may result in the need for additional on-the-job or classroom learning.

Potential Job Titles: Computer Programmer, Software Coder, Web Programmer.

Work Processes

A. Workplace Basics

1. Describe workplace structure.
2. Describe workplace policies and procedures; general and Information Technology (IT) -related.
3. Demonstrate an understanding of general ideas regarding workplace ethics, interpersonal communication, and personal safety.
4. Demonstrate efficient basic task/time management, status reporting, work order updates, team participation.
5. Demonstrate ability to communicate technical ideas/concepts when assisting users unfamiliar with IT jargon.

B. Defining Projects

1. Identify and record customer/user/stakeholder requirements.
2. Compile set of potential actions users may take in a system.

3. Compose requirement specifications (if applicable).

C. Designing Software

1. Identify and record requirements.
2. Recognize computer hardware and demonstrate knowledge of same.
3. Identify potential security threats and other vulnerabilities which may arise.

D. Writing Software Code

1. Set up programming environment to be used with selected software, such as: C++, Java, Python, etc.
2. Input and store data; name and define 4 main data types: a) string; b) integer; c) floating-point number; and d) Boolean.
3. Convert data from one type to another.
4. Recognize and correct errors.
5. Perform operations, such as: arithmetic, assign values, compare values, find truth (logic), set order.
6. Make lists: write, change, fix, set, retrieve.
7. Construct branch choices.
8. Write loops, including breaks and skips.
9. Create and store custom functions.
10. Import functions (if applicable).
11. Write various types of sorting algorithms.
12. Import pre-defined functionality into programs.

E. Building Graphical Interfaces and Applications

1. Write interface code to produce display messages, gather entries, check boxes, And add images to applications.
2. Develop applications; tests and deploy applications on different operating systems (if applicable).

F. Testing, Deployment, and Maintenance

1. Develop new software.
2. Troubleshoot programs.
3. Train end users (if applicable).

Total Hours 1,000-2,000

Apprentices in Competency-Based Programs shall participate in no fewer than 1,000 documented hours of on-the-job training, and until they have demonstrated a competency for each skill in the Work Processes.

Apprenticeship work processes are applicable only to training curricula for apprentices in approved programs. Apprenticeship work processes have no impact on classification determinations under Article 8 or 9 of the Labor Law. For guidance regarding classification for purposes of Article 8 or 9 of the Labor Law, please refer to http://www.labor.state.ny.us/workerprotection/publicwork/PDFs/Article8FAQS.pdf.

SOFTWARE DEVELOPER
(Competency-Based)

APPENDIX B

RELATED INSTRUCTION

Safety/Health/Environment
General Workplace Safety
First Aid & CPR (minimum 6.5 hours every 3 years)
Right-to-Know/ Safety Data Sheets (SDS)
Sexual Harassment Prevention Training (minimum 3 hours)

Computer and Network Components and Operations
Hardware
Peripherals
Software installation
Operating Systems, e.g., Microsoft, MacOS
Troubleshooting
Programming Languages, e.g., Java, C++, Python
Cybersecurity

Professional Development
Technical Support Communication
Time Management
Basic Project Management
Team and Supervisor Communication Skills
Customer Service Fundamentals

Other Courses as Necessary

At least 144 hours of Related Instruction per year must be available for the apprentice at the time of his/her indenture.