“Quality+ Certification Training Plan and Exam Questions”

Eric M. Pastore

Submitted November 14, 2015

Submitted as a Capstone Project Deliverable in Partial Fulfillment of a Master of Science Degree in Professional Studies at the Rochester Institute of Technology

Table of Contents

1. Needs Assessment

1.1. The foundation of this Training Plan is a graduate Capstone Project with 5 phases: Analysis, Design, Develop, Implement, and Evaluate. The Analysis phase was an exploratory phase designed to identify the major problems in software development. A total of 20 software engineers were interviewed to determine what issues they saw in the software industry. Overall, 46 unique issues were identified and analyzed in the Analysis phase. These issues are identified throughout this document as a percentage of identified issues. Table 1 below lists the related identified issues.

Table 1 – Selected Identified Issues

Issue	Percentage
Technical Debt	10%
Developers Not Trained in Design and Best Practices	8%
Developers Do Not Keep Up With New Methods	5%
Documentation Inadequate	4%
Legacy Systems Difficult to Maintain	3%
Systems Are Unnecessarily Complex	2%
Scalability Drives Complexity	2%

1.2. Based on this analysis, the primary issue to solve is Technical Debt (10%); a term used by several engineers to describe the hacks and shortcuts developers use to complete code projects on time and under budget. Technical Debt is caused by a lack of best practices, and results in code that works but is difficult to understand, maintain, and update. Technical Debt leads to problems on many projects because the code is difficult to understand by the developer or is simply not built well enough to allow for future development. Time saved up front is later wasted fixing problems and relearning solutions rather than building value-added features.

1.3. Many other issues identified were related to a lack of knowledge about best practices. The secondary issue, based on percentage, is that developers are not trained in design and best practices (8%), which results in the type of shortcuts identified in the primary issue of Technical Debt. Many other issues suggest that

developers lack training and knowledge in several related areas, such as an inability (or unwillingness) to keep up with new methods (5%) and create documentation (4%). Based on the 80/20 Rule (MindTools.com), which states that 20% of issues cause 80% of problems, Technical Debt and the related lack of training were chosen as the primary issues to solve.

1.4. To address the primary and secondary issues, there are three possible solutions:

- **Update the Academic Curriculum for IT Students:** Changing the behaviors which lead to Technical Debt must start while future developers are still in school. An updated foundations course, ex. **Software Engineering Foundations**, could teach students how to design applications and focus on coding technique as opposed to learning specific technologies or methodologies. This would develop students ready to code in business environments.
- **Develop an On-the-Job (OJT) Training Certification Program:** Employers could take a lead role in building developer skills by implementing an in-house OJT program. An OJT program would build developer skills using real business examples taken from existing technologies and clients. Developers who pass the program would have hands-on knowledge of best practices in the context of their business environment.
- **Develop a Design** and **Best Practices Certification:** Many IT professionals earn certifications to prove their skills in a certain area, but no IT certifications exist specifically for design and best practices. A professional certification focused on best practices would drive a greater focus on best practices in both industry and academia, leading to better training opportunities to support developers seeking the certification.

1.5. Based on the issues identified in the Capstone Project, the best solution is to develop a design and best practices certification. There are several factors to support this conclusion:

- **Certification Does Not Exist:** Most certifications focus on specific technologies (Oracle's Java or Microsoft's Windows platform) or technical areas (computer repair or networking). Some certifications, such as the **International Software Architecture Qualification Board (iSAQB) Certified Professional for Software Architecture (CPSA)** and **IEEE Computer Society Certified Software Development Associate (CSDA)**

certifications, cover software development skills, but do not focus on best practices specifically. A simple, targeted certification is needed.

- **Standardization:** Best practices are not always standardized across professionals and organizations, even though many code standards exist. Disagreement about best practices and design methodologies impacts software quality because developers do not consistently apply techniques, resulting in inconsistency and inefficiency.
- **Lack of Training Options:** Academic and professional training curricula are usually driven by industry needs. If industry does not realize the need for a greater focus on best practices, then opportunities to learn will be limited. However, if industry requires a professional certification in best practices, then developers will seek certification to enter the job market, which in turn will drive demand for academic and professional training.

1.6. Developers need training in standard design methodologies and best practices and the software industry needs a method to validate developer skills. A professional certification, adopted by the software industry, will provide a standard framework for developer education, training, and skills validation. The Quality+ certification will, in turn, provide this framework and drive the necessary improvements in developer education and training.

2. Business Goals

2.1. A lack of developers' skills in design and best practices drives other issues in industry which drain time and resources. Of the issues identified in the Capstone Project, difficulty of maintaining legacy systems (3%), unnecessarily complex systems (2%), and scalability driving complexity (2%) are all impacted by this skill gap. This skill gap particularly affects project planning, since project managers plan for normal development activities but do not take into account the extra work required to fix code which is not properly designed, written, and/or documented.

3. Performance Objectives

3.1. Candidates will be able to explain the purpose and use of coding standards, including the business case and data related to defects and bugs. Candidates will also describe which organizations develop code standards, and what information a

standard should contain.

3.2. Candidates will list and explain the basic principles of designing and implementing software that meets established industry best practices and standards, including readability, maintainability, design, automation, documentation, standardization, and Verification and Validation (V&V).

3.3. Candidates will write readable code, including the use of indents, white space, parenthesis, and length limits on functions and classes.

3.4. Candidates will develop maintainable code which uses coding practices able to be intuitively understood by a maintenance developer.

3.5. Candidates will design an application or system architecture which can support future growth and functionality with minimal rework.

3.6. Candidates will consider project needs to select and use appropriate static analysis tools, compiler functions, and other automated tools.

3.7. Given a code block (function, class, etc.), candidates will write informative internal comments in the correct locations and with the correct information.

3.8. Candidates will understand the purpose and use of Computer-Aided Software Engineering (CASE) tools, visual models, and external documents such as design specifications, release notes, and data dictionaries.

3.9. Candidates will use standardization in software development, both in using standard formats and techniques in use in the software development industry and when creating their own.

3.10. Candidates will understand specific quality checks performed as part of overall V&V activities, including static analysis, assertions, and unit testing.

4. Candidate Profile

4.1. The primary candidates for this certification are software developers and engineers in all industries. Table 1 lists the known characteristics of these candidates.

4.2. **Note: While this certification is directed towards software developers and engineers, other important candidates could be project managers and IT functional managers who manage developers.**

Table 2 – Candidate Analysis

Characteristic	Description
Education & Training	Bachelor of Science degree in Computer Science, Information Technology, Software Engineering, or related field. May have some technical training from employer.
Experience	Varied. Entry-level developers could have some internship or co-op experience, while experienced developers could have 5 years of experience or more.
Reason for Certification	Most developers will pursue the certification to meet employer requirements and ensure their marketability. Other developers may be new to the industry, or returning to the industry, and want to validate their professional skills.
Biases/Beliefs	Experienced developers may be set in their ways and unwilling to adopt new methods. Some believe coding is an “art” and that each developer should use their own techniques. Inexperienced developers may hold to a “code before design” mentality from academia. Additional biases may come from management who resist new methods and force certified developers to conform to company culture, even when that culture is in violation of best practices. This may include resistance to certain types of documentation, application design methods, and readability techniques.
Need Gratifiers / Motivators	Developers want to be able to do their jobs with greater speed and efficiency while also reducing effort. They value methods and tools which they perceive to provide these benefits. Developers may also be returning to the field and need validation of and/or an update to their skills. A new certification could provide a needed update to a developer’s resume.
Reading Ability	The reading level will be at high school level or higher for native English speakers. English as a Second Language (ESL) candidates may have a more difficult time comprehending the test questions.
Terms & Topics to Avoid	N/A
Organizational membership	Varied

Characteristic	Description
Geographical Location	Worldwide
Learning Styles	Kinesthetic/Visual. Most prefer to learn using hands-on exercises with real systems. For reference, however, some will view instructional videos and vlogs.

5. Prerequisites

5.1. The list below describes the skills, education, and experience that a candidate will need before taking the Quality+ exam:

- Bachelor of Science degree in Computer Science, Information Technology, Software Engineering, or related field
- No professional experience in software development will be required. Candidates should have a full education in software development and some co-op experience, but will be able to take and pass the exam without being employed as a full-time developer.

6. Delivery Strategies

6.1. The certification will be delivered as a proctored online exam at locations run by third-party certification organizations.

7. Testing and Evaluation

7.1. Certification will be granted upon completion of a 100-question exam, composed of questions of the following types:

- Multiple Choice
- True/False
- Matching
- Fill-in-the-blank from a prepopulated answer bank
- Simulation (Performance-Based)

7.2. Questions will cover the subject areas specified in **Section 10: Certification Outline**.

7.3. Grading will be performed by the system automatically, and the candidate will receive certification immediately after receiving a passing score.

7.4. Candidates will need to complete the exam with a score of 80% to receive certification.

8. Certification Development

8.1. Development of the full exam will require two main resources:

- Instructional Designer/Training Developer – This resource will be responsible to develop the exam according to sound instructional practices.
- Software Development SME – This resource will be responsible to ensure the exam content is technically complete and accurate.

8.2. The certifying body will create a small team each year to review the exam. This review will include the contents of the exam, study materials, and the supporting code standards and reference materials listed in Table 3. If any update is required, the development team will update the training plan, exam, and any supplemental study material.

Table 3 – Code Standards/References

Organization	Standard/Reference
European Committee for Electrotechnical Standardization (CENELEC)	BS EN 62304:2006 Medical Device Software – Software Life-Cycle Processes
European Cooperation for Space Standardization	Space Engineering Software Engineering Handbook.
Food and Drug Administration (FDA)	General Principles of Software Validation; Final Guidance for Industry and FDA Staff
IEEE Computer Society	Software Engineering Body of Knowledge (SWEBOK)
Jet Propulsion Laboratory (JPL)	JPL Institutional Coding Standard for the C Programming Language
Microsoft Developer Network (MSDN)	Coding Techniques
National Aeronautics and Space Administration (NASA)	Software Engineering Handbook

9. Deliverables

9.1. A 100-question exam will be developed to test certification candidates in the necessary skills and knowledge.

9.2. Supplemental training materials should be developed to allow for training and self-study:

- Instructor/Candidate Guides
- Practice Exams
- Self-Study Guides
- Other materials to support online or instructor-led training

10. Certification Outline

10.1. Purpose

10.1.1. Since developers will encounter code standards as part of their careers, candidates will describe existing code standards. Questions in this section could cover which organizations issue the standards and what the standards contain.

10.1.2. Code standards have evolved from a history of defects and issues. Candidates will be able to explain the business reasons for having code standards, along with the positive impact on quality over the course of the software's life cycle.

10.1.3. Organizations have performed several studies relating to defects in software. Questions in this section will test candidates on various statistics related to software quality. For example, unit tests detect, on average, at least one defect in every 10 to 100 lines of code according to NASA's Jet Propulsion Laboratory (JPL).

10.2. Quality Principles

10.2.1. Quality principles include code readability, maintainability, design, automation, documentation, standardization, and V&V. Questions in this section will test candidates on these concepts at a high level. Candidates will know these concepts, the goal of their use, and general areas of application.

10.3. Code Readability

10.3.1. Many style conventions have an impact on the ability of a future developer to read and understand previously written code. These conventions include the use of indents, white space, variable naming, and many others. While specific points can be up for debate, questions in this section emphasize consistency and clarity in written code.

10.3.2. These questions also address function and class length limits, which make functions and classes easier for a developer to read. While different organizations may define their own limits, several standards establish these limits and provide guidelines for their length.

10.4. Code Maintainability

10.4.1. Maintainability, unlike readability, is determined not by appearance but by function. Factors such as the scoping and naming of variables can cause unintended results, and can complicate the work of a maintenance developer. Other factors which affect maintainability are nesting of statements, data type declarations, and hard-coding of values. Candidates will demonstrate understanding of how to write maintainable code.

10.5. Application Design

10.5.1. Many applications and systems have long life cycles and will be updated and improved constantly, from Request for Proposal (RFP) to system retirement. Concepts such as modularity and abstraction deal less with functionality and more with the underlying system structure which makes functionality easier to add and change.

10.5.2. Questions in this section cover design methods which developers can use to ensure that future upgrades are possible and easy, especially when the upgrades will be performed by someone other than the original developer.

10.6. Tools/Automation

10.6.1. Types of Tools

10.6.1.1. Many tools are available to developers to help with coding tasks. Questions in this exam will cover Static Analysis tools, Debugging tools, and the use of compiler warnings.

10.6.2. Static Analysis

10.6.2.1. Static analysis tools are useful to check code for quality issues, security risks, and adherence to code standards. Candidates will demonstrate mastery in the purpose and general use of static analysis tools.

10.7. Internal Documentation

10.7.1. Good documentation is essential for later developers to understand code. Questions in this category will test candidates on the requirements for comments in code. Comments in code can be defined at multiple levels, including application, class, and function levels, and may contain different content depending on comment type.

10.7.2. Along with the need for inline comments, candidates will also demonstrate mastery of the content, placement, density, and formatting of internal documentation.

10.8. External Documentation

10.8.1. Like internal documentation, external documentation is essential for understanding an application. Unlike internal documentation, it will be seen by many other stakeholders besides developers. Overall, external documentation describes the high level design of an application or system.

10.8.2. External documentation spans the software life cycle and includes examples such as the Risk Management Plan, Software Requirements Specification, System Design Specification, and Software Test Plan. Questions in this category cover the different types (or genres) of external documentation, why they exist, and minimum content requirements. Questions also cover CASE tools, which automate the modeling and documentation of many parts of an application or system.

10.9. Code Standardization

10.9.1. Software languages are the building blocks of applications and systems. Developers can use these building blocks in different ways for different projects. There are, however, standards in place which developers should follow, including naming conventions for variables, constants, functions, tables, etc. Even when creating their own solutions, developers should be consistent and adhere to their own standards.

10.10. Verification and Validation (V&V)

10.10.1. V&V activities are an important aspect of any major software engineering project. Many of these activities are carried out by specialized test personnel, but individual developers play a crucial role. Candidates will understand how static analysis, traceability analysis, and unit testing are performed and their effect on software quality. Candidates will also describe the use of assertions as part of unit testing.

11. Appendix I: Sample Exam Questions (Correct Answers in Red)

11.1. Which organizations develop and publish code standards?

11.1.1. Microsoft

11.1.2. National Aeronautics and Space Agency (NASA)

11.1.3. Food and Drug Administration (FDA)

11.1.4. All of the Above

11.2. Given the following example, determine the best resource to use for reference. You are coding a new module for an application, but do not know how you should format your functions (spacing, curly braces, indents, etc.). How do you decide which formatting to use?

11.2.1. Look at the other modules and use the same formatting

11.2.2. Refer to company code standards and project-specific standards

11.2.3. Refer to FDA and ISO Specifications

11.2.4. Use formatting consistent with your last project

11.3. A function should never be longer than a printed sheet of paper, or 60 lines.

11.3.1. True

11.3.2. False

11.4. A variable is needed for a single function and is not used anywhere else in the application. What is the optimal scope for the variable? Drag and drop the variable to the best location for the variable in the code below.

```
/**
 * A sample of a ploymorphic method.
 * @author Developer
 *
 */
2
public class CreateASet {

    public static void main(String[] args){
        String[] words = {"A", "B", "B", "D", "C", "A"};
3
        System.out.println( "original: " + Arrays.toString(words));
        System.out.println( "as a set: " + Arrays.toString(makeSet(words)));

        Rectangle[] rectList = {new Rectangle(), new Rectangle(),
                new Rectangle(0, 1, 2, 3), new Rectangle(0, 1, 2, 3)};
        System.out.println( "original: " + Arrays.toString(rectList));
        System.out.println( "as a set: " + Arrays.toString(makeSet(rectList)));


        Object[] mixed = {"A", "C", "A", "B", new Rectangle(),
                new Rectangle(), "A", new Rectangle(0, 1, 2, 3), "D"};
        System.out.println( "original: " + Arrays.toString(mixed));
        System.out.println( "as a set: " + Arrays.toString(makeSet(mixed)));
    }

    public static Object[] makeSet(Object[] data){
        assert data != null : "Failed precondition makeSet. parameter cannot be null";
        assert noNulls(data) : "Failed precondition makeSet. no elements of parameter can be null";
        Object[] result = new Object[data.length];
        int numUnique = 0;
4
        boolean found;
        int indexInResult;
        for(int i = 0; i < data.length; i++){
            // maybe should break this out into another method
            indexInResult = 0;
            found = false;
            while(!found && indexInResult < numUnique){
                found = data[i].equals(result[indexInResult]);
                indexInResult++;
            }
            if( ! found ){
                result[numUnique] = data[i];
                numUnique++;
            }
        }
        Object[] result2 = new Object[numUnique];
        System.arraycopy(result, 0, result2, 0, numUnique);
        return result2;
    }

    // pre: data != null
    // return true if all elements of data are non null,
    // false otherwise
    private static boolean noNulls(Object[] data){
        assert data != null : "Failed precondition makeSet. parameter cannot be null";
1
        int i = 0;
        while( good && i < data.length ){
            good = data[i] != null;
            i++;
        }
        return good;
    }

}
```

boolean good = true;

11.4.1. Location 1

11.4.2. Location 2

11.4.3. Location 3

11.4.4. Location 4

11.5. Given the following example, choose the best strategy: Your company is launching a web application which will exist for the foreseeable future and require many updates. How should you design this application?

11.5.1. A single module to make the code easier to see at a glance

11.5.2. One module handling the user interface and another module handling all other functionality

11.5.3. Multiple code modules each handling a single discrete task

11.5.4. Application design is not important

11.6. Most compiler warnings can be disabled because they do not affect code functionality.

11.6.1. True

11.6.2. False

11.7. Internal documentation (code comments) should be declared in which locations in the source code?

11.7.1. Functions

11.7.2. Classes

11.7.3. Variables

11.7.4. All of the Above

11.8. Given the list of project deliverables below, rank them in the order they are created in a project according to a typical software life cycle.

11.8.1. (1) Software Quality Assurance Plan

11.8.2. (2) Traceability Analysis

11.8.3. (3) Software Design Specification

11.8.4. (4) User Help Documentation

11.8.5. (5) Test Procedure and Test Cases

11.8.6. (6) Final Test Report

11.8.7. (7) Release Notes

11.9. When developing code, innovation is key. New ways of developing code are always better than current methods.

11.9.1. True

11.9.2. False

11.10. How many assertions, on average, should a developer build into each function?

11.10.1. None

11.10.2. 2

11.10.3. 10

11.10.4. 4