Final Assignment—E-Commerce Website Test Plan

Alicia Piavis

CST313: Software Testing

Robert Key

05/02/2020

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Software Requirements Specification

1.1 Introduction

1.1.1 Purpose

This software requirements specification (SRS) outlines the requirements for the e-commerce website software build. The purpose of the e-commerce website is to allow customers a way to quickly find and purchase what they desire from a range of options. This document refers to version 1.0 of the e-commerce website. The scope of this SRS encompasses the entirety of the initial build of the e-commerce website, including the basic features described in section 4, as well as other non-functional requirements described in section 5.

1.1.2 Document Conventions

This document is broken into 5 sections with headers for Introduction, Overall Description, External Interface Requirements, System Features, and Other Non-Functional Requirements. Sub-headers provide detail regarding the specifics of those sections.

1.1.3 Intended Audience and Reading Suggestions

This SRS is intended for UI/UX designers, software designers, architects, developers, project managers, testers, and other stakeholders involved in the execution of this project. It is suggested that all audience members read the document from start to finish in order to gain a thorough understanding of the purpose and requirements of the e-commerce website.

1.1.4 Product Scope

The goal of the e-commerce website is to allow a user to find what they are looking for quickly and easily. The website will allow a user to search for an item, select a search suggestion, narrow down a search by selecting different filters for an item, view details about an item, add an item to a cart, continue shopping, and then checkout when they are ready. The development of this e-commerce website aligns with the business objective to increase growth and profitability.

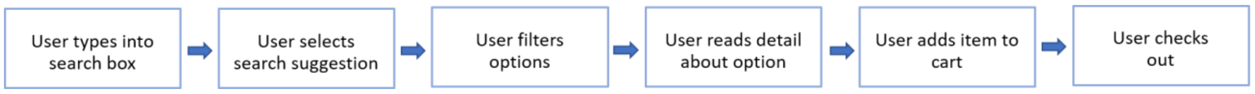
1.2 Overall Description

1.2.1 Product Perspective

The build for the e-commerce website is a new, self-contained product.

1.2.2 Product Functions

A user should be able to search for an item, and narrow down their search by selecting different filters for an item, they should be able to view details about an item, add an item to a cart, continue shopping, and then checkout when they are ready. In addition, there should be different filters for different types of items. In other words, items should be categorized into groupings of similar items. Furthermore, there should be search suggestions when a user starts typing, so that the correct category of item can be retrieved.


User types into search box
User selects search suggestion
User filters options
User reads detail about option
User adds item to cart
User checks out

1.2.3 User Classes and Characteristics

User classes for the e-commerce website include customers, vendors, and administrators. The customer class will need to be able to search select, and purchase items. The vendor class will need to be able to add new items for availability, and update their current inventory of listings. Administrators will need to be able to access all functionalities, including adding, updating, and deleting item listings, as well as assisting users in logging into their accounts and resetting their login information. Administrators should also be able to look up a confirmed order by an order ID number. The primary user class for the e-commerce website is the customer user class.

1.2.4 Operating Environment

The e-commerce website needs to have cross-platform functionality, as well as be mobile responsive. The website should function on all major web browsers, including Chrome, Internet Explorer, Edge, Bing, Yahoo, and Safari. In addition, it should function properly on both Mac and Windows operating systems, as well as iOS and Android.

1.2.5 Design and Implementation Constraints

The e-commerce website needs to be built within 6 months, and must utilize a relational database. In addition, all passwords for user accounts must use encryption. In order to comply with company policy, the website must abide by ADA web accessibility guidelines (ADA, 2007). Furthermore, the system will be maintained by the customer, so the design and build should support modifiability, maintainability, and testability.

1.2.6 User Documentation

In addition to the requested functionalities of the E-Commerce Website, the site will also include documentation aimed at users who need assistance troubleshooting or guidance in creating a user account. Furthermore, the site will have a chat bot, which allows users to request assistance when needed.

1.2.7 Assumptions and Dependencies

Constraints for the project include a budget of $50,000, a timeline of 6 months, and a project team of no more than eight, which includes designers, architects, database administrators, developers, testers, and project managers.

1.3 External Interface Requirements

1.3.1 User Interfaces

Each page of the E-Commerce Website should have the same header, footer, and navbar for consistency. Icons should all be selected from Font Awesome. Each page should contain a back arrow in the same location that allows a user to return to the previous page. All images should include alt tags, and users should be able to tab through a page. In addition, colors should be selected for usability and visual appeal, and fonts should be easily readable. Ads should not draw attention away from the main content of the page, and general layout should be consistent across pages. The home button should be clearly visible on every page. Furthermore, forms should provide users with clear feedback instructing them how to correct errors when they are made.

1.3.2 Hardware Interfaces

The E-Commerce Website should be accessible from all major devices including iOS and Android phones, iPads and other tablets, and both Mac and Microsoft products. The site will use HTTP to make requests and TCP to transfer resources from the web server to the client. The website should also support screen readers.

1.3.3 Software Interfaces

The website should be compatible with iOS, Android, Mac, and Windows operating systems. The build should also include an API that talks to the relational database in order to store and retrieve customer, product, and transactional data. In addition, the site should allow single sign on with options to sign in using Google or Facebook accounts.

1.3.4 Communications Interfaces

Users should be able to receive email notifications upon confirmation of an order, as well as optional newsletters and deal notifications related to their preferences and order history. In additional all personally identifiable information (PII) should be secure, and all passwords should be encrypted.

1.4 System Features

1.4.1 New User Can Create an Account

1.4.1.1 Description and Priority

- Priority: High
- Description: User should be able to create an account. This encompasses part of the basic functionality of the website. A user cannot make a purchase without creating an account.
- Benefit: 9

1.4.1.2 Stimulus/Response Sequences

- User gets prompted on the home page to log in or create an account
- User clicks on register
- User enters form data to create account
- User clicks submit
- User receives positive confirmation that account has been created or clear instructions regarding how to correct their input

1.4.1.3 Functional Requirements

- REQ-1: Website should present user with two buttons on home page (login and register)
- REQ-2: Website should present registration form when user clicks on "register"
- REQ-3: Data from form is validated, sanitized, processed, and sent to the database when a user hits "submit"
- REQ-4: Website presents user with informative messages if input does not meet requirements
- REQ-5: Website provides user with success message when account has been successfully created

1.4.2 Returning User Can Login

1.4.2.1 Description and Priority

- Priority: High
- Description: Returning user should be able to login. This encompasses part of the basic functionality of the website. A user cannot make a purchase without logging in.
- Benefit: 9

1.4.2.2 Stimulus/Response Sequences

- User gets prompted on the home page to log in or create an account
- User clicks on login
- User enters form data to login
- User clicks submit
- User receives positive confirmation that they have been logged in or clear instructions regarding how to correct their input

1.4.2.3 Functional Requirements

- REQ-1: Website should present user with two buttons on home page (login and register)
- REQ-2: Website should present login form when user clicks on “login”
- REQ-3: Data from form is validated, sanitized, processed, and sent to the database when a user hits “submit”
- REQ-4: Website presents user with informative messages if input does not meet requirements
- REQ-5: User credentials are verified against user information stored in the database

- REQ-6: Website provides user with success message when account has been successfully created

1.4.3 User Can Search For An Item

1.4.3.1 Description and Priority

- Priority: High
- Description: User should be able to search for an item. This encompasses part of the basic functionality of the website. A user cannot find an item without being able to search for it.
- Benefit: 8

1.4.3.2 Stimulus/Response Sequences

- User types into the search bar
- User is provided with search suggestions
- User selects a search suggestion
- User is presented with a list of search results
- User selects filters to narrow search results

1.4.3.3 Functional Requirements

- REQ-1: Website should have a search bar
- REQ-2: Website should dynamically present user with search suggestions
- REQ-3: Website should return search results when user selects suggestion
- REQ-4: Website should dynamically generate filters depending on the category that the item falls into
- REQ-5: Website should update search results when filters are selected

1.4.4 User Can View Item Details

1.4.4.1 Description and Priority

- Priority: High
- Description: User should be able to view item details. A user will be hesitant to make a purchase without having the ability to view item details.
- Benefit: 7

1.4.4.2 Stimulus/Response Sequences

- User clicks on an item link from the search results list
- User views item details

1.4.4.3 Functional Requirements

- REQ-1: Search results list has links for each item
- REQ-2: When link is clicked, user is redirected to page that presents item details

1.4.5 User Can Add Item to Cart

1.4.5.1 Description and Priority

- Priority: High
- Description: User should be able to add an item to their virtual cart. This encompasses part of the basic functionality of the website. A user cannot make a purchase without first adding an item to the cart.
- Benefit: 8

1.4.5.2 Stimulus/Response Sequences

- User clicks on button to add item to cart
- User is presented with option to check out or continue shopping

- If user selects “continue shopping”, they are redirected to the search page
- If user selects “checkout”, they are redirected to the checkout page

1.4.5.3 Functional Requirements

- REQ-1: Item detail page should have “add to cart” button
- REQ-2: Website should update cart icon with correct number of items when a user adds an item to their cart
- REQ-3: Website should present user with an option to “continue shopping” or “checkout”
- REQ-4: If user selects “continue shopping”, website redirects them to the search page
- REQ-5: If user selects “checkout”, website redirects them to the checkout page

1.4.6 User Can View Their Cart

1.4.6.1 Description and Priority

- Priority: Medium
- Description: User should be able to view their cart. Most users want to be able to view their cart before committing to a purchase.
- Benefit: 7

1.4.6.2 Stimulus/Response Sequences

- User clicks on the cart icon at the top right of the page
- User views a list of their cart items
- User can edit or delete cart items

1.4.6.3 Functional Requirements

- REQ-1: Cart icon at top right of the page should be a link that redirects the user to the cart view
- REQ-2: The cart view should present the user with a list of the items they added, including a short description of the item, the quantity, and the price
- REQ-3: The cart view should show the user's projected total
- REQ-4: Website should allow a user to delete items from the cart or update the quantity
- REQ-5: Website will refresh cart view after a user updates quantities or deletes items
- REQ-6: Cart view has a "checkout" button

1.4.7 User Can Checkout

1.4.7.1 Description and Priority

- Priority: High
- Description: User should be able to checkout. This encompasses part of the basic functionality of the website. A user cannot make a purchase without checking out and initiating a transaction.
- Benefit: 9

1.4.7.2 Stimulus/Response Sequences

- User clicks on the cart icon
- User clicks on "checkout" button
- User is presented with checkout form
- User enters form data
- User clicks submit

- User receives positive confirmation that order has been placed or clear instructions regarding how to correct their input

1.4.7.3 Functional Requirements

- REQ-1: Website should have "checkout" button on cart view page
- REQ-2: Website should redirect user to checkout page when button is clicked
- REQ-3: Website should present checkout form
- REQ-4: Data from form is validated, sanitized, processed, and sent to the database when a user hits "submit"
- REQ-5: Website presents user with informative messages if input does not meet requirements
- REQ-6: Website provides user with success message when order has been processed

1.5 Other Nonfunctional Requirements

1.5.1 Performance Requirements

New users should be able to create an account in less than 60 seconds. When a user searches for an item, they should receive a list of search results in less than 3 seconds. Once a user hits the "checkout" button, they should be able to complete the checkout form and receive a confirmation message in under 2 minutes. Speed is of utmost importance, as delays may cause a poor user experience and may deter users from returning to the site in the future.

1.5.2 Safety Requirements

Before a user confirms an order, the website needs to confirm that the user is over age 14. All PII should be protected, and passwords should be encrypted. In addition, sessions should timeout after 5 minutes without activity, unless the user confirms that they want to continue the session.

1.5.3 Security Requirements

All PII should be protected and secure. Credentials should be verified before a user is given access to account information. In addition, passwords should be encrypted. Credit card information should not be stored, and transactions should be processed securely.

1.5.4 Software Quality Attributes

The E-Commerce Website should follow web accessibility guidelines outlined by ADA (n.d.). In addition, the site should be easy to use and navigate. The website should be reliable, as interruptions in service may deter users from returning. The site should be scalable to accommodate for business growth and an increasing userbase. It is important that the site be modifiable, to allow for the addition of new features, and the product should be maintainable and testable, as maintenance will be the responsibility of the customer.

1.5.5 Business Rules

Users should not be allowed to add items to a cart, checkout, or view their account information unless they are logged in. Administrators should have the ability to assist users in resetting credentials. They should also be able to search for order details using a reference number for administrative purposes. Finally, administrators should be able to retrieve a list of all orders, past and present, for reporting purposes.

2. Testing Levels and UML Models

2.1 Introduction

It is imperative to test software throughout the development lifecycle in order to ensure a quality product. When performed effectively, testing can save costs, reduce development time, improve maintainability, improve code quality, identify and correct faults early in the process, increase robustness, and ensure that the customer is receiving a quality product that meets their needs and desires. The four levels of testing are component testing, integration testing, system testing, and acceptance testing (Spillner, Linz, & Schaefer, 2014). Component testing breaks software down into functional units, and tests whether or not these units meet the required specifications. Integration testing occurs when groups of related components are tested together to identify faults in the interfaces and interactions between components (GeeksForGeeks, n.d.). System testing “specifically focuses on testing the functional and non-functional aspects of the software in more comprehensive manner including security, usability, performance and compatibility.” (Suffian et.al., 2016). Acceptance testing ensures that the software meets the customer’s needs, and works in a production-level environment.

2.2 Component Testing

In regards to the E-Commerce Website, component testing will occur for each of the different functionalities illustrated in the use case diagram in Figure 1. For example, the software will have components that carry out each of the different functions: populating search suggestions when a customer starts typing into the search bar, dynamically generating filters once a user selects a search suggestion, displaying the search results as a list, showing item details when a user clicks on a link from the list, allowing a user to add an item to their cart, displaying the cart when a user wants to view the cart, carrying out a transaction when a user needs to checkout, and displaying a confirmation page. There will also be components that handle functionalities linked to other user groups, such as administrators and vendors. Components that might be used for administrators include those that allow an administrator to create a report, pull a report, delete a report, and update a report. There will also be components that allow vendors to add new items to the E-Commerce Website, update item details, and be notified when inventory is low. Components will also include the classes shown in Figure 3. These classes include E-Commerce System Manager, User, Customer, Vendor, Administrator, Report, Item, Cart Item, Cart, Delivery Method, and Invoice. All of these components should be thoroughly tested to ensure that they address the use cases in Figure 1, the activity diagram in Figure 2, the class diagram in Figure 3, and the sequence diagram in Figure 4. Component-level testing will also ensure that the code is maintainable, effective, robust, performant, and lacks faults.

2.3 Integration Testing

The E-Commerce System will also need to go through integration testing. Integration testing allows testers and developers the ability to "discover faults and bugs in the interaction between integrated components" (Ali et.al., 2018). Once all of the components mentioned above are thoroughly tested on their own, test cases will be made to evaluate the collaboration of components. For example, Figure 4 illustrates the classes that are necessary in a sequence of events that allow a customer to place an order. While each of these classes might have passed testing at the component level, the real test is whether or not the control flow and data flow is successful across the interfaces and interactions between components.

2.4 System Testing

After integration tests are complete, the E-Commerce Website will need to go through a series of system tests, which assess whether or not the website as a whole meets functional and non-functional requirements. For example, does the website possess all of the functionalities shown in the activity diagram? Is the E-Commerce website fast, reliable, available, secure, and maintainable? The testers and developers at this point need to view the system from a user perspective, and test the system in an environment that is as close to production-level as possible.

2.5 Acceptance Testing

Finally, the E-Commerce Website will go through acceptance testing. This process will involve customers to ensure that the system provides them with all of the functionality they need to quickly search for and purchase items. Administrators will need to provide feedback regarding whether or not they can quickly and easily access reports, and vendors will need to determine if they can adequately use the website to sell new products, and view and update inventories. Acceptance testing will take place across different operating systems and devices to ensure that all users will have a positive user experience. Performing the four levels of testing (component, integration, system, and acceptance) will support the development team in delivering a high-quality product as efficiently and quickly as possible.

2.6 UML Models

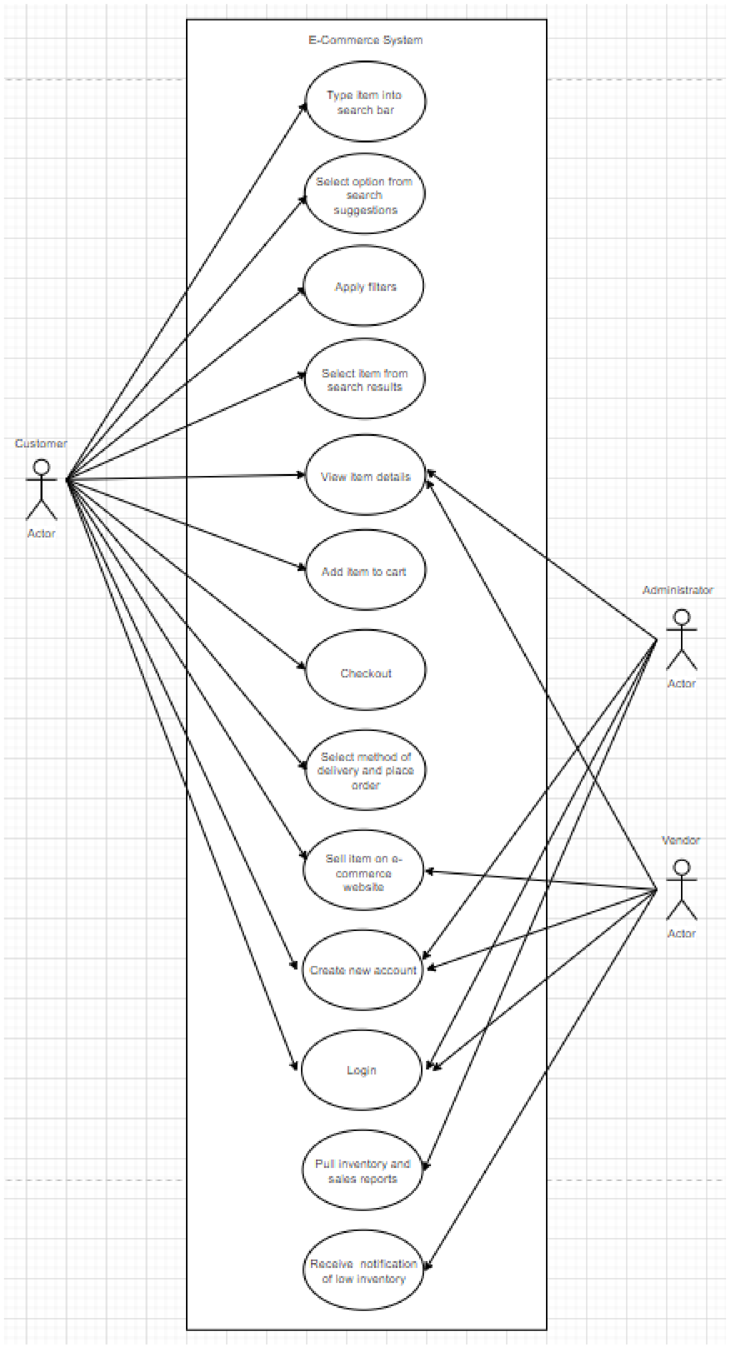

E-Commerce System
Customer
Actor
Administrator
Actor
Vendor
Actor
Type item into search bar
Select option from search suggestions
Apply filters
Select item from search results
View item details
Add item to cart
Checkout
Select method of delivery and place order
Sell item on e-commerce website
Create new account
Login
Pull inventory and sales reports
Receive notification of low inventory

Figure 1. Use Case Diagram for E-Commerce Website

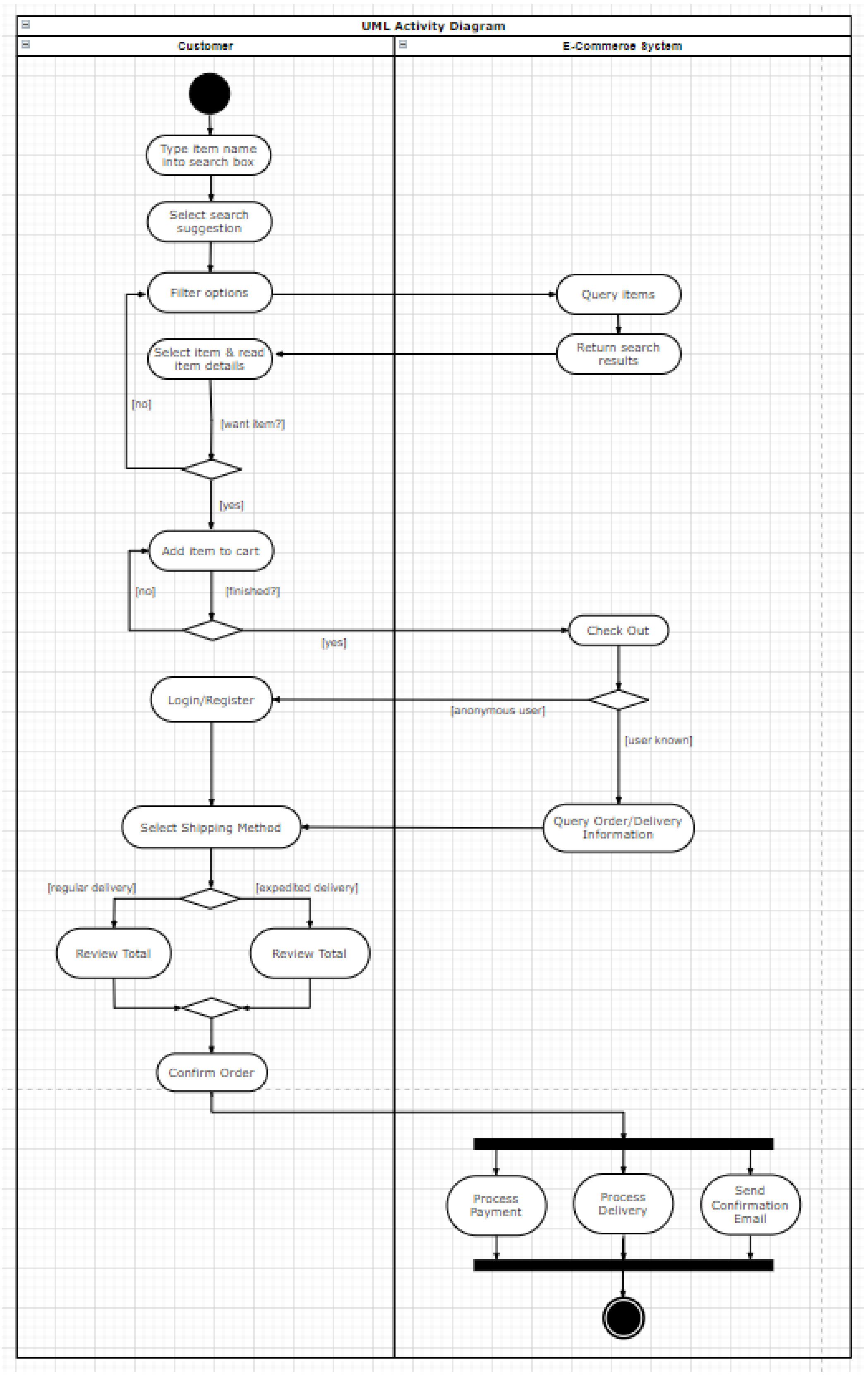


Figure 2. Activity Diagram for E-Commerce Website

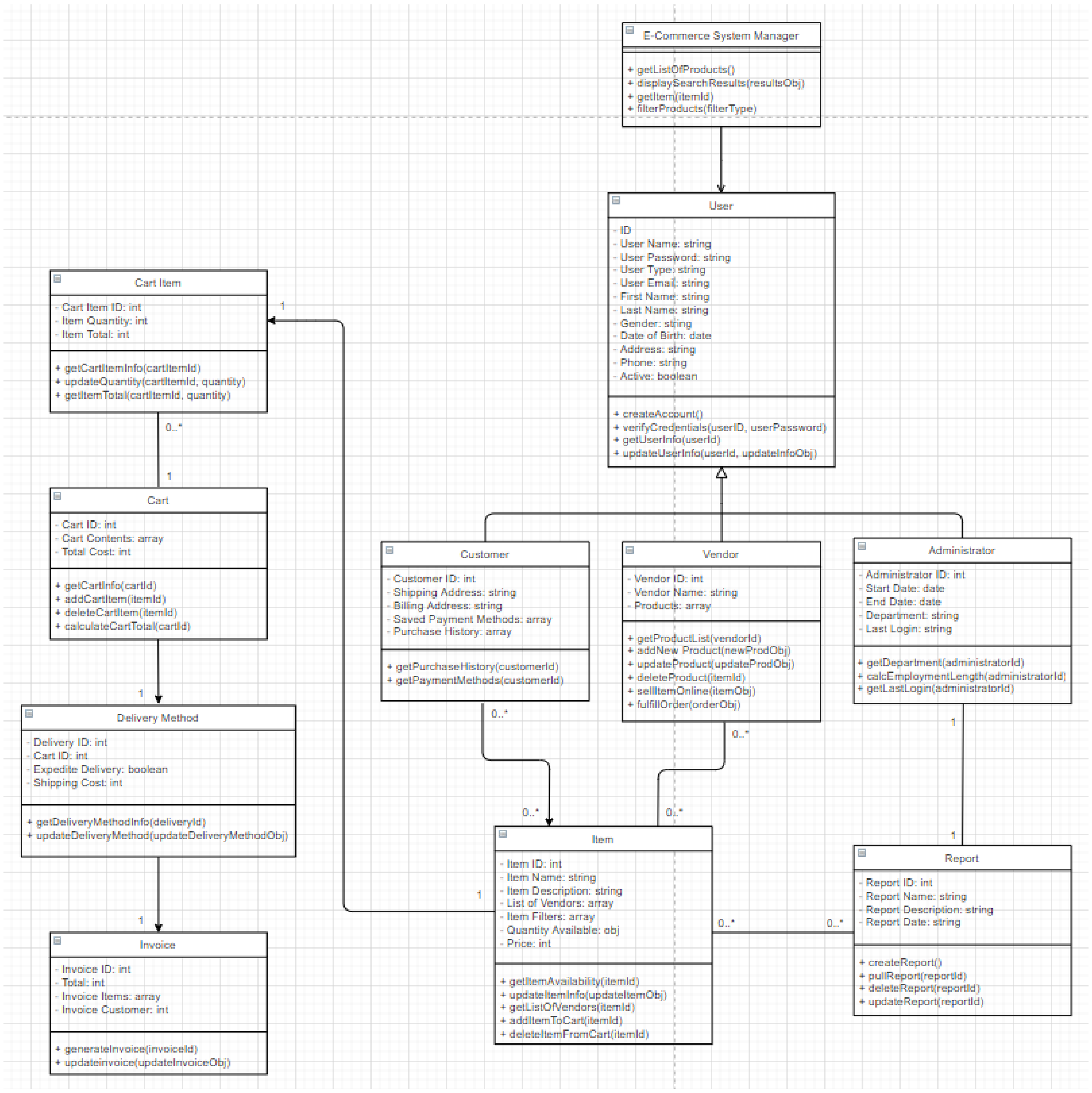


Figure 3. Class Diagram for E-Commerce Website

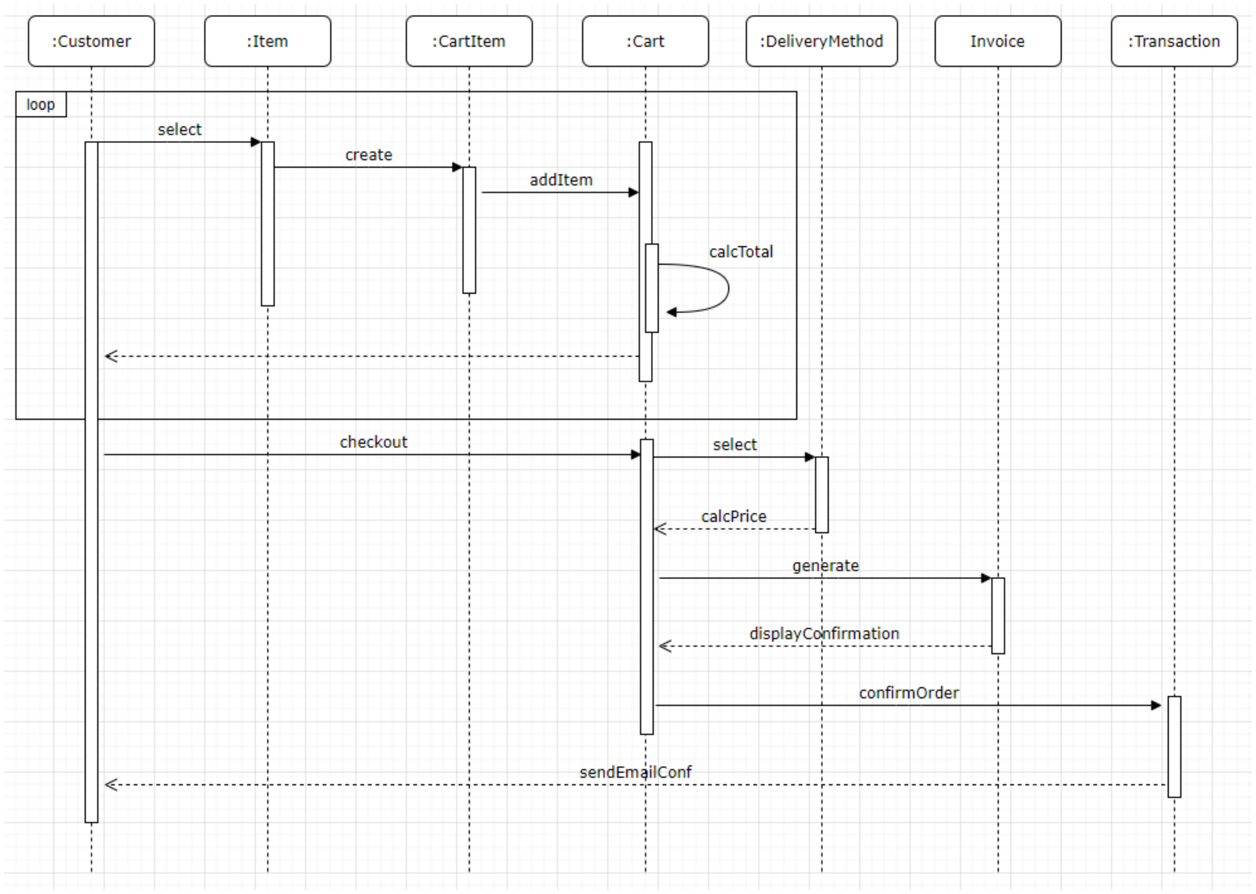


Figure 4. Sequence Diagram for E-Commerce Website

3. Review and Testing Strategies

3.1 Overview of Review Strategies

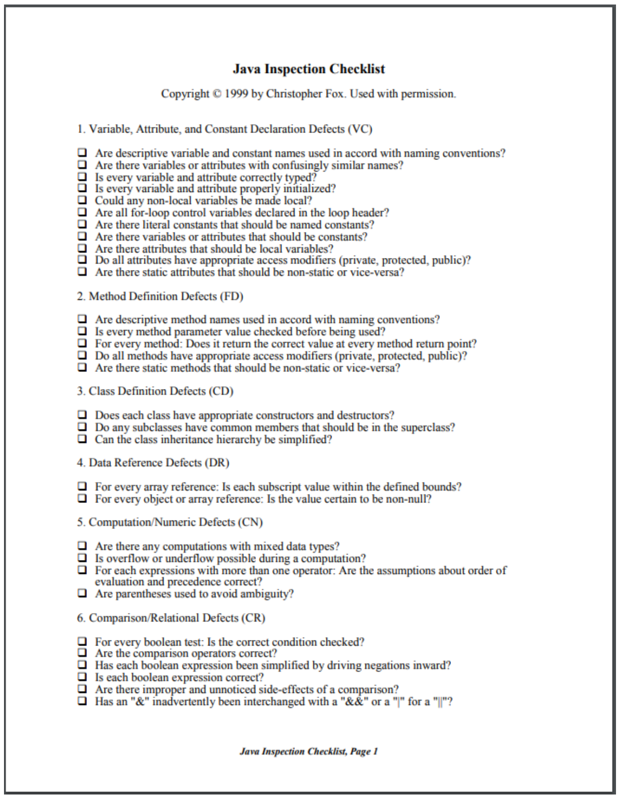
3.1.1 Walkthrough

A walkthrough is an informal review method used to find "defects, ambiguities, and problems in written documents" (Spillner, Linz, & Schaefer, 2014). The author of the code presents to an audience with the goal of education. This strategy involves a meeting, but there is no time limit, and little preparation is involved. Often, the meeting includes walk throughs of different use cases, with the reviewers asking spontaneous questions. Because the review is informal and utilizes spontaneous questions, no template is necessary for documentation. Walk throughs occur in small group settings and require few resources.

According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), “The main objectives of a walkthrough are mutual learning, development of an understanding of the review object, and error detection.”

3.1.2 Inspection

Inspections are very formal. They involve clearly defined roles for individuals, rules, protocols, and checklists (Spillner, Linz, & Schaefer, 2014). The planning phase for inspections revolves around defining objectives. A moderator facilitates the meeting and follows a clear agenda. The goals of an inspection include “finding unclear items and possible defects, measuring review object quality, and improving the quality of the inspection process and the development process” (Spillner, Linz, & Schaefer, 2014). This strategy involves more preparation, and aims to improve the overall development process, in addition to simply reviewing the software for defects (Spillner, Linz, & Schaefer, 2014). A sample inspection template can be seen in Figure 5 (Fox, 1999).



Java Inspection Checklist

Copyright © 1999 by Christopher Fox. Used with permission.

1. Variable, Attribute, and Constant Declaration Defects (VC)

- ❏ Are descriptive variable and constant names used in accord with naming conventions?
- ❏ Are there variables or attributes with confusingly similar names?
- ❏ Is every variable and attribute correctly typed?
- ❏ Is every variable and attribute properly initialized?
- ❏ Could any non-local variables be made local?
- ❏ Are all for-loop control variables declared in the loop header?
- ❏ Are there literal constants that should be named constants?
- ❏ Are there variables or attributes that should be constants?
- ❏ Are there attributes that should be local variables?
- ❏ Do all attributes have appropriate access modifiers (private, protected, public)?
- ❏ Are there static attributes that should be non-static or vice-versa?

2. Method Definition Defects (FD)

- ❏ Are descriptive method names used in accord with naming conventions?
- ❏ Is every method parameter value checked before being used?
- ❏ For every method: Does it return the correct value at every method return point?
- ❏ Do all methods have appropriate access modifiers (private, protected, public)?
- ❏ Are there static methods that should be non-static or vice-versa?

3. Class Definition Defects (CD)

- ❏ Does each class have appropriate constructors and destructors?
- ❏ Do any subclasses have common members that should be in the superclass?
- ❏ Can the class inheritance hierarchy be simplified?

4. Data Reference Defects (DR)

- ❏ For every array reference: Is each subscript value within the defined bounds?
- ❏ For every object or array reference: Is the value certain to be non-null?

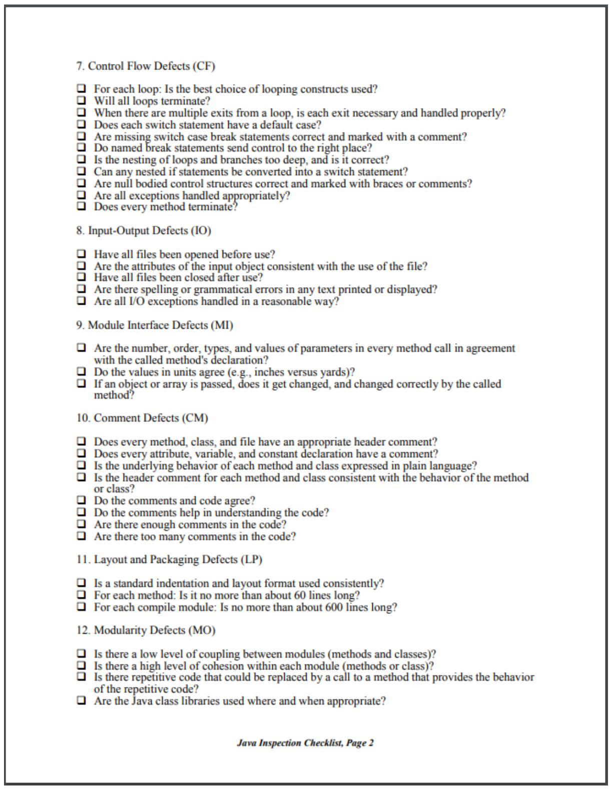
5. Computation/Numeric Defects (CN)

- ❏ Are there any computations with mixed data types?
- ❏ Is overflow or underflow possible during a computation?
- ❏ For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?
- ❏ Are parentheses used to avoid ambiguity?

6. Comparison/Relational Defects (CR)

- ❏ For every boolean test: Is the correct condition checked?
- ❏ Are the comparison operators correct?
- ❏ Has each boolean expression been simplified by driving negations inward?
- ❏ Is each boolean expression correct?
- ❏ Are there improper and unnoticed side-effects of a comparison?
- ❏ Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?

Java Inspection Checklist, Page 1



7. Control Flow Defects (CF)

- ❏ For each loop: Is the best choice of looping constructs used?
- ❏ Will all loops terminate?
- ❏ When there are multiple exits from a loop, is each exit necessary and handled properly?
- ❏ Does each switch statement have a default case?
- ❏ Are missing switch case break statements correct and marked with a comment?
- ❏ Do named break statements send control to the right place?
- ❏ Is the nesting of loops and branches too deep, and is it correct?
- ❏ Can any nested if statements be converted into a switch statement?
- ❏ Are null bodied control structures correct and marked with braces or comments?
- ❏ Are all exceptions handled appropriately?
- ❏ Does every method terminate?

8. Input-Output Defects (IO)

- ❏ Have all files been opened before use?
- ❏ Are the attributes of the input object consistent with the use of the file?
- ❏ Have all files been closed after use?
- ❏ Are there spelling or grammatical errors in any text printed or displayed?
- ❏ Are all I/O exceptions handled in a reasonable way?

9. Module Interface Defects (MI)

- ❏ Are the number, order, types, and values of parameters in every method call in agreement with the called method's declaration?
- ❏ Do the values in units agree (e.g., inches versus yards)?
- ❏ If an object or array is passed, does it get changed, and changed correctly by the called method?

10. Comment Defects (CM)

- ❏ Does every method, class, and file have an appropriate header comment?
- ❏ Does every attribute, variable, and constant declaration have a comment?
- ❏ Is the underlying behavior of each method and class expressed in plain language?
- ❏ Is the header comment for each method and class consistent with the behavior of the method or class?
- ❏ Do the comments and code agree?
- ❏ Do the comments help in understanding the code?
- ❏ Are there enough comments in the code?
- ❏ Are there too many comments in the code?

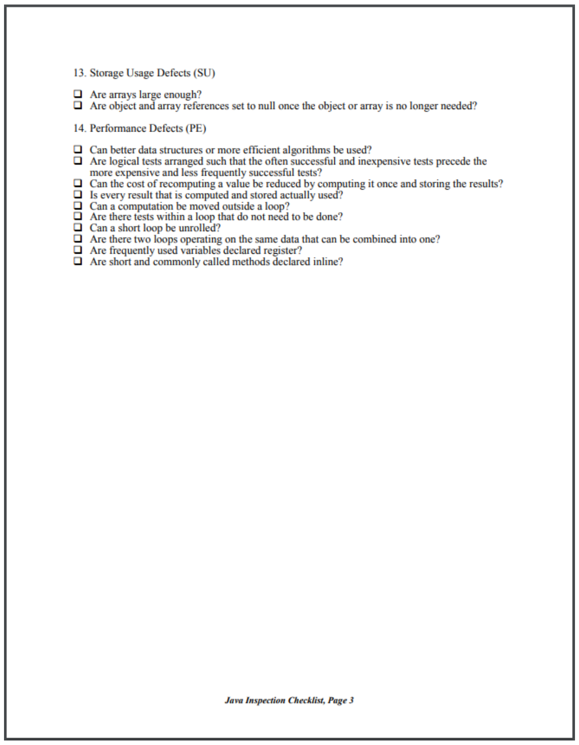
11. Layout and Packaging Defects (LP)

- ❏ Is a standard indentation and layout format used consistently?
- ❏ For each method: Is it no more than about 60 lines long?
- ❏ For each compile module: Is no more than about 600 lines long?

12. Modularity Defects (MO)

- ❏ Is there a low level of coupling between modules (methods and classes)?
- ❏ Is there a high level of cohesion within each module (methods or class)?
- ❏ Is there repetitive code that could be replaced by a call to a method that provides the behavior of the repetitive code?
- ❏ Are the Java class libraries used where and when appropriate?

Java Inspection Checklist, Page 2



13. Storage Usage Defects (SU)

- ❑ Are arrays large enough?
- ❑ Are object and array references set to null once the object or array is no longer needed?

14. Performance Defects (PE)

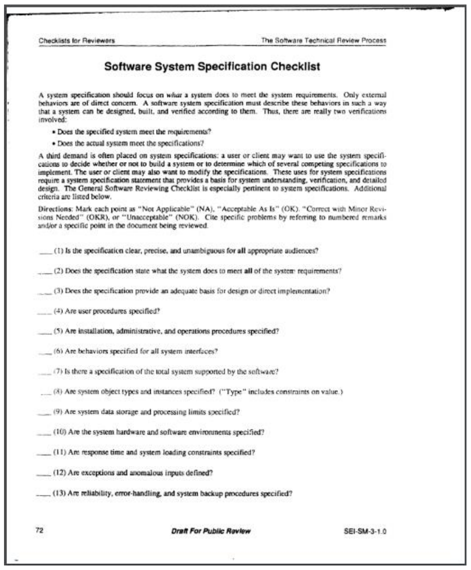
- ❑ Can better data structures or more efficient algorithms be used?
- ❑ Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?
- ❑ Can the cost of recomputing a value be reduced by computing it once and storing the results?
- ❑ Is every result that is computed and stored actually used?
- ❑ Can a computation be moved outside a loop?
- ❑ Are there tests within a loop that do not need to be done?
- ❑ Can a short loop be unrolled?
- ❑ Are there two loops operating on the same data that can be combined into one?
- ❑ Are frequently used variables declared register?
- ❑ Are short and commonly called methods declared inline?

Java Inspection Checklist, Page 3

Figure 5. Sample Inspection Checklist (Fox, 1999)

3.1.3 Technical Reviews

Technical reviews aim to ensure that test objects meet their specifications (Spillner, Linz, & Schaefer, 2014). While this strategy involves a review meeting, the author typically does not intend. Instead, the reviewers do a considerable amount of preparation work to review the object and provide their feedback. The meeting involves a recorder who takes notes, consolidates feedback from the reviewers, and generates the final document with the results (Spillner, Linz, & Schaefer, 2014). In addition to feedback, these review meetings involve discussions around possible alternatives, as well as possible errors and defects. The reviewers must each have a certain level of technical expertise in order for this process to be successful. Templates for technical reviews can be seen in Figure 6 below (Carnegie Mellon, 1991).



Checklists for Reviewers The Software Technical Review Process

Software System Specification Checklist

A system specification should focus on *what* a system does to meet the system requirements. Only external behaviors are of direct concern. A software system specification must describe these behaviors in such a way that a system can be designed, built, and verified according to them. Thus, there are really two verifications involved:

- Does the specified system meet the requirements?
- Does the actual system meet the specifications?

A third demand is often placed on system specifications: a user or client may want to use the system specifications to decide whether or not to build a system or to determine which of several competing specifications to implement. The user or client may also want to modify the specifications. These uses for system specifications require a system specification statement that provides a basis for system understanding, verification, and detailed design. The General Software Reviewing Checklist is especially pertinent to system specifications. Additional criteria are listed below.

Directions: Mark each point as "Not Applicable" (NA), "Acceptable As Is" (OK), "Correct with Minor Revisions Needed" (OKR), or "Unacceptable" (NOK). Cite specific problems by referring to numbered remarks and/or a specific point in the document being reviewed.

____ (1) Is the specification clear, precise, and unambiguous for all appropriate audiences?

____ (2) Does the specification state what the system does to meet all of the system requirements?

____ (3) Does the specification provide an adequate basis for design or direct implementation?

____ (4) Are user procedures specified?

____ (5) Are installation, administrative, and operations procedures specified?

____ (6) Are behaviors specified for all system interfaces?

____ (7) Is there a specification of the total system supported by the software?

____ (8) Are system object types and instances specified? ("Type" includes constraints on value.)

____ (9) Are system data storage and processing limits specified?

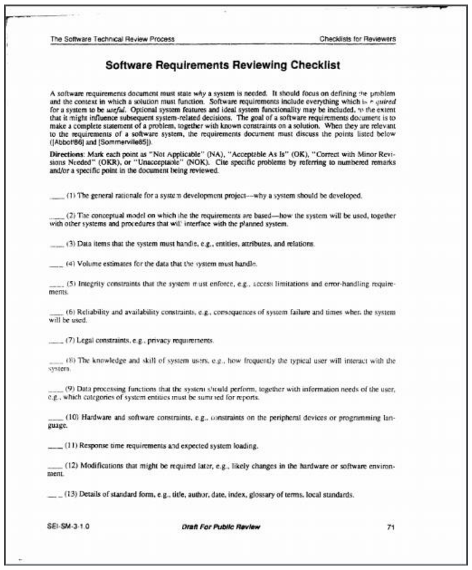
____ (10) Are the system hardware and software environments specified?

____ (11) Are response time and system loading constraints specified?

____ (12) Are exceptions and anomalous inputs defined?

____ (13) Are reliability, error-handling, and system backup procedures specified?

72 *Draft For Public Review* SEI-SM-3-1.0



The Software Technical Review Process Checklists for Reviewers

Software Requirements Reviewing Checklist

A software requirements document must state *why* a system is needed. It should focus on defining the problem and the context in which a solution must function. Software requirements include everything which is *required* for a system to be *useful*. Optional system features and ideal system functionality may be included, to the extent that it might influence subsequent system-related decisions. The goal of a software requirements document is to make a complete statement of a problem, together with known constraints on a solution. When they are relevant to the requirements of a software system, the requirements document must discuss the points listed below ([Abbott86] and [Sommerville85]).

Directions: Mark each point as "Not Applicable" (NA), "Acceptable As Is" (OK), "Correct with Minor Revisions Needed" (OKR), or "Unacceptable" (NOK). Cite specific problems by referring to numbered remarks and/or a specific point in the document being reviewed.

____ (1) The general rationale for a system development project—why a system should be developed.

____ (2) The conceptual model on which the the requirements are based—how the system will be used, together with other systems and procedures that will interface with the planned system.

____ (3) Data items that the system must handle, e.g., entities, attributes, and relations.

____ (4) Volume estimates for the data that the system must handle.

____ (5) Integrity constraints that the system must enforce, e.g., access limitations and error-handling requirements.

____ (6) Reliability and availability constraints, e.g., consequences of system failure and times when the system will be used.

____ (7) Legal constraints, e.g., privacy requirements.

____ (8) The knowledge and skill of system users, e.g., how frequently the typical user will interact with the system.

____ (9) Data processing functions that the system should perform, together with information needs of the user, e.g., which categories of system entities must be summed for reports.

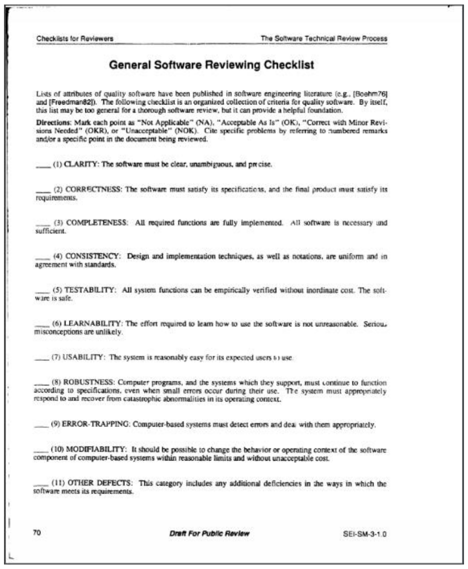
____ (10) Hardware and software constraints, e.g., constraints on the peripheral devices or programming language.

____ (11) Response time requirements and expected system loading.

____ (12) Modifications that might be required later, e.g., likely changes in the hardware or software environment.

____ (13) Details of standard form, e.g., title, author, date, index, glossary of terms, local standards.

SEI-SM-3-1.0 *Draft For Public Review* 71



Checklists for Reviewers The Software Technical Review Process

General Software Reviewing Checklist

Lists of attributes of quality software have been published in software engineering literature (e.g., [Boehm76] and [Freedman82]). The following checklist is an organized collection of criteria for quality software. By itself, this list may be too general for a thorough software review, but it can provide a helpful foundation.

Directions: Mark each point as "Not Applicable" (NA), "Acceptable As Is" (OK), "Correct with Minor Revisions Needed" (OKR), or "Unacceptable" (NOK). Cite specific problems by referring to numbered remarks and/or a specific point in the document being reviewed.

____ (1) CLARITY: The software must be clear, unambiguous, and precise.

____ (2) CORRECTNESS: The software must satisfy its specifications, and the final product must satisfy its requirements.

____ (3) COMPLETENESS: All required functions are fully implemented. All software is necessary and sufficient.

____ (4) CONSISTENCY: Design and implementation techniques, as well as notations, are uniform and in agreement with standards.

____ (5) TESTABILITY: All system functions can be empirically verified without inordinate cost. The software is safe.

____ (6) LEARNABILITY: The effort required to learn how to use the software is not unreasonable. Serious misconceptions are unlikely.

____ (7) USABILITY: The system is reasonably easy for its expected users to use.

____ (8) ROBUSTNESS: Computer programs, and the systems which they support, must continue to function according to specifications, even when small errors occur during their use. The system must appropriately respond to and recover from catastrophic abnormalities in its operating context.

____ (9) ERROR-TRAPPING: Computer-based systems must detect errors and deal with them appropriately.

____ (10) MODIFIABILITY: It should be possible to change the behavior or operating context of the software component of computer-based systems within reasonable limits and without unacceptable cost.

____ (11) OTHER DEFECTS: This category includes any additional deficiencies in the ways in which the software meets its requirements.

70 *Draft For Public Review* SEI-SM-3-1.0

Figure 6. Sample Technical Review Checklists (Carnegie Mellon, 1991)

3.1.4 Informal Reviews

Informal reviews are a lighter version of a technical review. They are author-initiated, and the planning phase involves choosing reviewers and letting them know when to deliver feedback (Spillner, Linz, & Schaefer, 2014). There is typically no meeting involved, but instead, colleagues cross-read each other's comments. In addition, the results are not explicitly documented, so there are no templates used. Examples of informal reviews include pair programming, buddy testing, and code swapping (Spillner, Linz, & Schaefer, 2014). This strategy is common because it is low cost, and little effort is required. In addition, the process encourages discussion and knowledge transfer amongst colleagues.

3.2 Recommended Review Strategies

For the development of the E-Commerce Website, it is recommended to use walkthroughs, technical reviews, and informal reviews. While inspections can be beneficial, they require a considerable amount of time, coordination, and expertise. Walkthroughs, technical reviews, and informal reviews will provide sufficient methods for identifying defects, verifying requirements, and improving code quality, readability, and maintainability while reducing time and cost.

3.3 Static Testing Strategies

3.3.1 Tools

The following tools will be used for static testing of the E-Commerce Website: an IDE or text editor, a compiler, and a linter. These tools will allow the developers and testers to identify syntax violations, cross-reference program elements, check data types, detect undeclared variables, detect dead code, and check interface consistency (Spillner, Linz, & Schaefer, 2014). Aside from detecting errors, the IDE and linter will also ensure that code is more readable and consistent since there will be multiple developers collaborating on the E-Commerce Website. The IDE and compiler will also assist developers in debugging.

3.3.2 Compliance to Conventions and Standards

According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "Compliance to conventions and standards can also be checked with tools. For example, tools can be used to check if a program follows programming regulations and standards." Tools within the IDE and linter mentioned above should be utilized to implement rules that enforce proper coding conventions and standards. In addition, once the programming language is selected, an associated style guide should also be chosen. For example, if the E-Commerce Website uses Java, programmers and testers can utilize the Google Java Style Guide (Google, n.d.) to ensure consistency and compliance.

3.3.3 Data Flow Analysis

Data flow analysis should be used for the purpose of revealing defects in code for the E-Commerce Website. This type of analysis "checks the usage of every single variable" (Spillner, Linz, & Schaefer, 2014). This includes variables that are defined, referenced, and undefined. The data flow analysis will be used to reveal data flow anomalies. “An anomaly is an inconsistency that can lead to failure but does not necessarily do so. An anomaly may be flagged as a risk” (Spillner, Linz, & Schaefer, 2014).

3.3.4 Control Flow Analysis

Control flow analysis will also be used to find defects. Control flow utilizes graphs to represent a sequence of statements to analyze program execution. These diagrams are used to visualize changes in the execution of the program (the control flow) caused by logic such as conditionals and loops. Examples of control flow graphs can be seen in Figure 7 below (GeeksForGeeks, n.d.). According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), “Due to the clarity of the control flow graph, the sequences through the program can easily be understood and possible anomalies can be detected.” Identified anomalies may not cause the program to fail, but they should be addressed to maintain compliance to conventions and standards.

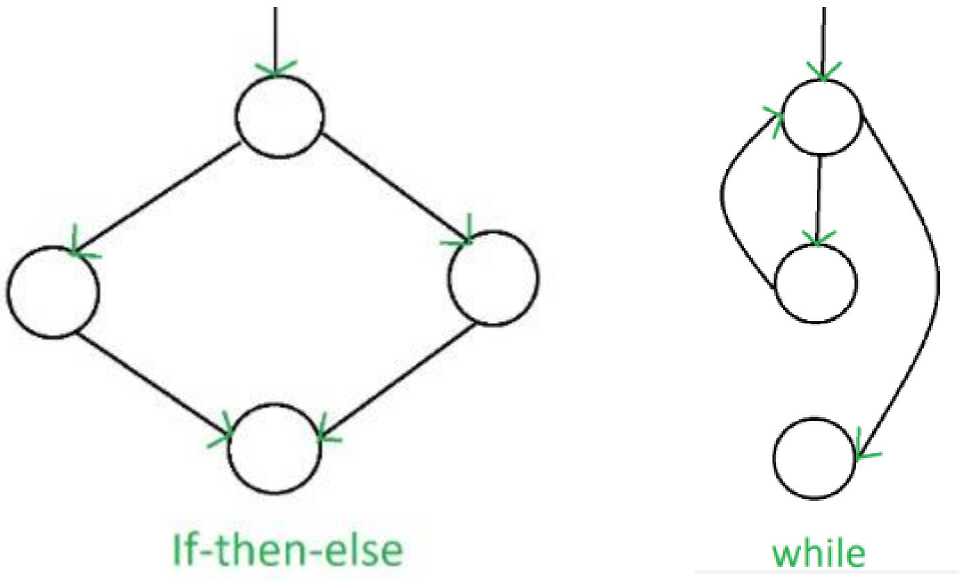


Figure 7. Example Control Flow Graphs (GeeksForGeeks, n.d.)

3.4 Dynamic Testing Strategies

3.4.1 Black Box Testing

Black box testing includes a group of testing techniques that aim to verify that a test object meets its specifications and that expected outputs result from a specified set of inputs. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), the goal of these strategies is "the verification of the functionality of the test object. It is indisputable that the highest priority is that the software work correctly. Thus, black box techniques should always be applied." Therefore, it is imperative that test strategies for the E-Commerce Website include black box techniques. The following methods will be used for testing the E-Commerce Website.

3.4.2 Equivalence Class Partitioning

This approach helps developers and testers systematically generate test cases based on domains for various inputs. "An equivalence class is a set of data values that the tester assumes are processed in the same way by the test object" (Spillner, Linz, & Schaefer, 2014). In other words, this strategy involves identifying the parameters that a function accepts, generating potential valid and invalid inputs for that parameter, creating an equivalence class that represents a group of similar inputs, selecting a representative input for each equivalence class, and then generating test cases that eliminate redundancy as much as possible.

3.4.3 Boundary Value Analysis

Boundary value analysis will be used in conjunction with equivalence class partitioning in order to generate test cases for the E-Commerce Website. This strategy focuses on testing inputs that lie on the boundaries between equivalence classes. "Boundary value analysis delivers a very reasonable addition to the test cases that have been identified by equivalence class partitioning. Faults often appear at the boundaries of equivalence classes. This happens because boundaries are often not defined clearly or are misunderstood" (Spillner, Linz, & Schaefer, 2014).

3.4.4. State Transition Testing

This testing method is beneficial in evaluating how a system reacts to transitions in state. For example, the E-Commerce Website involves a number of objects. These objects change and react to user input. For example, a user's Cart (an object) begins in an empty state. When a user views an item and then selects "Add to Cart", the state of the object changes—it is no longer empty. This testing strategy will be applied to see how events related to a customer searching, selecting, and purchasing an item trigger changes in the state of different objects.

3.4.5 Use-Case-Based Testing

Use-case-based testing will be important in evaluating the functionality of the E-Commerce Website. This strategy involves testing the execution of paths aligned with each of the use cases illustrated in the use case diagram for the E-Commerce Website (in the design documents). For example, one of the test cases describes how a customer needs to be able to narrow down their search by applying filters. This means that at least one test case needs to be generated to evaluate whether or not the website implements that functionality correctly, and whether or not the search results are refreshed when a customer adjusts the filters.

3.4.6 White Box Testing

White box testing is more granular than black box testing and directly involves evaluating the source code. As stated by *Analysis of statement branch and loop coverage in software testing with genetic algorithm* (2017), "White box testing involves executing a program and seeing which parts of it are executed." It is important to include these methods in analysis of the E-Commerce website, since these tests are designed to evaluate branching logic, conditionals, and loops. This ensures that all paths of execution are evaluated in the test environment, so that the first time a line of code is being executed is not in production. The following white box techniques will be used to test the E-Commerce Website.

3.4.7 Statement Testing

Statement testing refers to testing each statement within a test object. The goal is 100% execution of the code, but that is not always possible due to time and budgetary constraints. Statement testing can help identify dead statements (where code exists that cannot be reached by any test case). It is important to track which lines of code have been executed to reduce redundancy and make testing more efficient (Spillner, Linz, & Schaefer, 2014).

3.4.8 Decision/Branch Testing

Decision/branch testing focuses on executing decisions in the code, rather than individual statements. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "In contrast to statement coverage, branch coverage makes it possible to detect missing statements in empty branches." As a result of this, "Decision/branch coverage usually requires the execution of more test cases than statement coverage" (Spillner, Linz, & Schaefer, 2014). This leads to more reliable code. Examples of decisions include IF statements, case statements, and loops. Control flow graphs can also be used in this type of testing to ensure that all test cases are considered.

3.4.9 Testing of Conditions

Condition testing evaluates more complex decisions influenced by multiple conditions in the code. This technique is important so that the results of all complex decisions are considered. Spillner, Linz, & Schaefer (2014) state, "Combinations of logical expressions are especially defect prone. Thus, a comprehensive test is very important."

3.4.10 Experience-Based Testing

Experience-based testing, also known as intuitive-based testing, will be used to supplement black box and white box techniques in the testing of the E-Commerce Website. Intuitive-based tests leverage the skills and experience of developers and testers to generate test cases that might otherwise be overlooked in systematic testing. In this approach, "The test cases are based on where faults have occurred in the past or the tester's ideas of where faults might occur in the future" (Spillner, Linz, & Schaefer, 2014).

4. Test Management Strategy

4.1 Test Teams

In order to achieve efficient development of the E-Commerce Website it is important to consider test teams and an approach to each level of testing. If at all possible, developers on the team should not test their own code. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "because there is a tendency to be blind to our own errors, it is much more efficient to let different people perform testing and development and to organize testing as independently as possible from development." For this reason, designated testers on the development team should perform component testing. In addition, a designated testing team within the project should perform integration testing. These testers may include individuals from the business or IT. System testing should also be performed by a designated team that includes specialists: "Especially in system testing, it is often necessary to extend the test team by adding IT specialists, at least temporarily, to perform work for the test team" (Spillner, Linz, & Schaefer, 2014). This will allow the system to be viewed and tested from multiple perspectives.

4.2 Test Roles

The following test roles should be assigned to individuals in order to conduct testing of the E-Commerce Website: test manager, test designer, test automator, test administrator, and tester (Spillner, Linz, & Schaefer, 2014). The test manager will require experience in software testing, quality management, project management, and personnel management. The test designer should have a skillset that includes test methods and specification, testing, and software engineering. They should also hold a degree in Computer Science. The test automator will require experience in testing, programming, scripting, test tools, and automation. The test administrator needs a skillset involving setting up and supporting test environments, as well as system administration and networking. The tester should have experience following procedures, executing tests, reporting failures, and using test objects and testing tools (Spillner, Linz, & Schaefer, 2014).

4.3 Exit Criteria

Each test case should contain specific exit criteria, which guides testers in determining when a test is considered to be complete. Exit criteria is important because “They prevent tests from ending too early, for example, because of time pressure or because of resource shortages” (Spillner, Linz, & Schaefer, 2014). In other words, exit criteria will help determine when tests for the E-Commerce Website should be started and stopped. By identifying these items up front, testers and developers can rely on criteria to determine when to stop testing: “To make a right decision to stop testing is an arduous resolution, the pre-defined exit criteria can help simplify this process. It is a very important step where all test processes get stopped and this decision is either made by the tester or the whole team together” (Nidagundi & Novickis, 2016). Having clear exit criteria allows testers to have increased confidence in the software.

4.4 Test Estimated Effort

It is important to consider both the costs of testing, as well as the costs of undetected defects, when planning software testing. It is the test manager's responsibility to initiate test effort estimation during the planning phase to ensure that resources are assigned and distributed properly. Estimating test effort for the E-Commerce Website will be conducted by basing estimates on data from former or similar projects, as this method provides the most reliable test effort estimates: "task-driven test effort estimation tends to underestimate the testing effort. Estimating based on experience data of similar projects or typical values usually leads to better results" (Spillner, Linz, & Schaefer, 2014).

4.5 Test and Risk

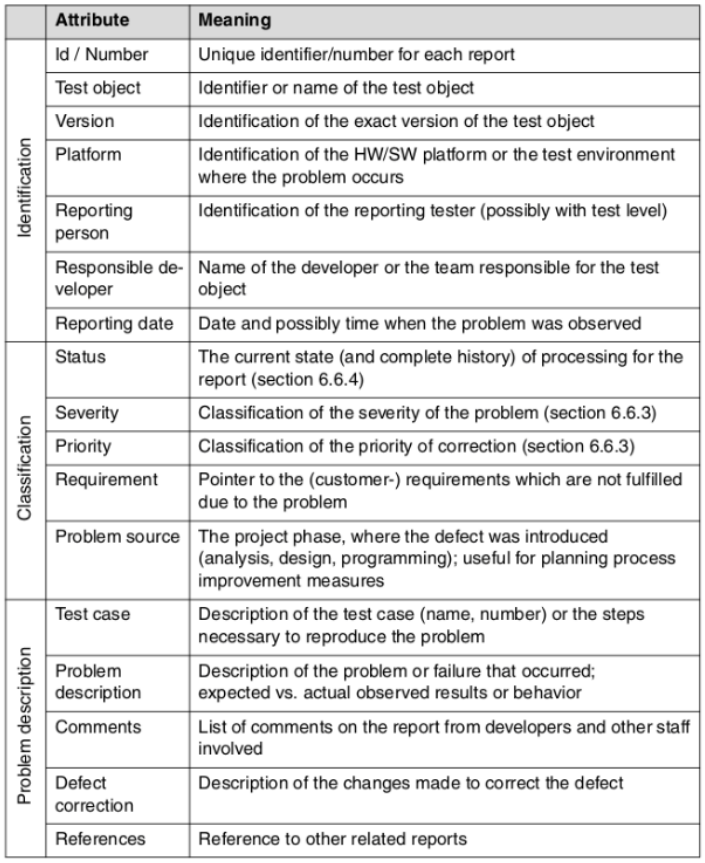
Risk-based prioritization will be implemented in order to ensure that the most significant defects within the E-Commerce Website are revealed as early as possible. This will prevent critical defects from having downstream affects while reducing the time and cost of handling critical defects that make it to production. While test cases can be prioritized based on multiple factors, including frequency of function use, visibility, priority of functional and non-functional requirements from the customer, severity, risk, and complexity, prioritization by risk is one of the best methods for selecting test cases. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "Risk based prioritization of the tests ensures that risky product parts are tested more intensively and earlier than parts with lower risk. Severe problems (causing much corrective work or serious delays) are found as early as possible."

Risk is "the mathematical product of the loss or damage due to failure and the probability (or frequency) of failure resulting in such damage" (Spillner, Linz, & Schaefer, 2014). In order to reduce risk, it is important to implement risk management techniques. Risk management involves identifying, prioritizing, and mitigating risks. Testing can serve as a risk mitigation technique because it "provides information about existing problems and the success or failure of correction" (Spillner, Linz, & Schaefer, 2014). Risk-based testing can be used to generate test cases that address potential areas of risk.

4.6 Incident Reporting

Incident reporting will be used throughout the development and testing of the E-Commerce Website in order to document and manage incidents. Every defect that is determined to be significant and legitimate (not the result of a poorly designed test) should be documented. The template shown in Figure 8 (Spillner, Linz, & Schaefer, 2014) below will be used to document incidents. This will allow the communication of incidents to be consistent, and developers will be able to easily reproduce defects. Homès (2012) stresses the importance of clearly communicating defects: "Determining the impact in an understandable way for developers (data loss, functionality loss, software instability, etc.) and for customers and the hierarchy (impacts in financial terms or in usability terms, noncompliance to requirements, etc.) enables a quick recognition of the anomaly."

The incident report will contain the following information: the tested software, test environment, tester's name, the class that contains the defect, prioritization of the defect, and information relevant to reproducing and locating the defect (Spillner, Linz, & Schaefer, 2014). Testers, developers, customers and users can report incidents. When corrections are made, the incident report should be updated in the database. This will help the project team track and manage incidents.

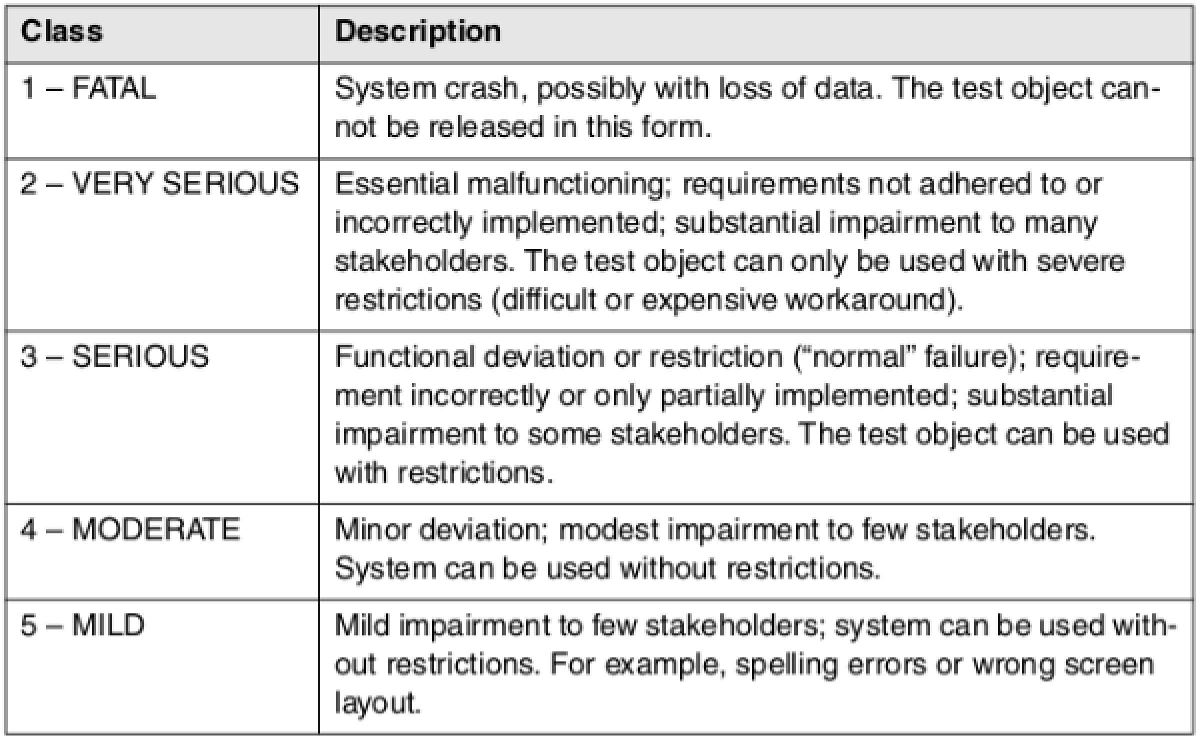


	Attribute	Meaning
Identification	Id / Number	Unique identifier/number for each report
	Test object	Identifier or name of the test object
	Version	Identification of the exact version of the test object
	Platform	Identification of the HW/SW platform or the test environment where the problem occurs
	Reporting person	Identification of the reporting tester (possibly with test level)
	Responsible developer	Name of the developer or the team responsible for the test object
	Reporting date	Date and possibly time when the problem was observed
Classification	Status	The current state (and complete history) of processing for the report (section 6.6.4)
	Severity	Classification of the severity of the problem (section 6.6.3)
	Priority	Classification of the priority of correction (section 6.6.3)
	Requirement	Pointer to the (customer-) requirements which are not fulfilled due to the problem
	Problem source	The project phase, where the defect was introduced (analysis, design, programming); useful for planning process improvement measures
Problem description	Test case	Description of the test case (name, number) or the steps necessary to reproduce the problem
	Problem description	Description of the problem or failure that occurred; expected vs. actual observed results or behavior
	Comments	List of comments on the report from developers and other staff involved
	Defect correction	Description of the changes made to correct the defect
	References	Reference to other related reports

Figure 8. Incident Reporting Template (Spillner, Linz, & Schaefer, 2014)

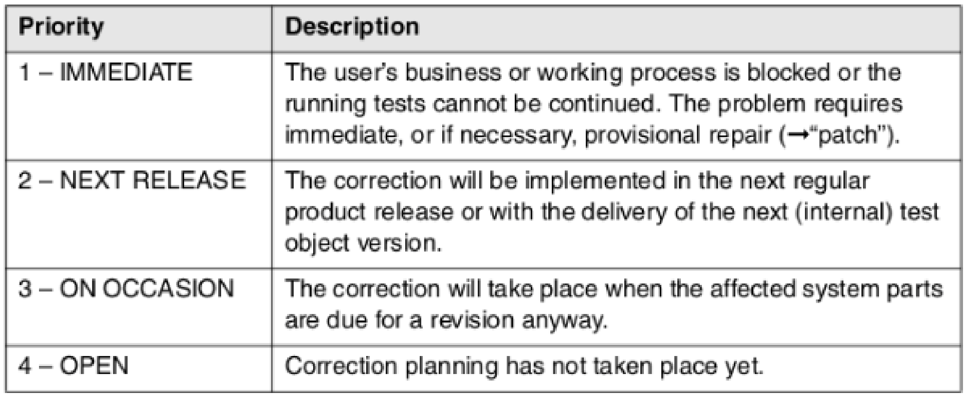
4.7 Defect Classification

Defect classification will be implemented for prioritizing incidents related to the E-Commerce Website. The severity level will reflect the level of impairment caused by the defect. The following 5 levels of severity will be used: 1-Fatal, 2-Very Serious, 3-Serious, 4-Moderate, and 5-Mild. The criteria for these levels can be seen in Figure 9 (Spillner, Linz, & Schaefer, 2014) below. Levels of priority will also be implemented in order to identify how quickly a problem should be addressed. The following four levels of priority will be used: 1-Immediate, 2-Next Release, 3-On Occasion, and 4-Open. The criteria assigned to each of these priority levels can be seen in Figure 10 (Spillner, Linz, & Schaefer, 2014) below.



Class	Description
1 – FATAL	System crash, possibly with loss of data. The test object cannot be released in this form.
2 – VERY SERIOUS	Essential malfunctioning; requirements not adhered to or incorrectly implemented; substantial impairment to many stakeholders. The test object can only be used with severe restrictions (difficult or expensive workaround).
3 – SERIOUS	Functional deviation or restriction ("normal" failure); requirement incorrectly or only partially implemented; substantial impairment to some stakeholders. The test object can be used with restrictions.
4 – MODERATE	Minor deviation; modest impairment to few stakeholders. System can be used without restrictions.
5 – MILD	Mild impairment to few stakeholders; system can be used without restrictions. For example, spelling errors or wrong screen layout.

Figure 9. Severity Levels (Spillner, Linz, & Schaefer, 2014)



Priority	Description
1 – IMMEDIATE	The user's business or working process is blocked or the running tests cannot be continued. The problem requires immediate, or if necessary, provisional repair (→"patch").
2 – NEXT RELEASE	The correction will be implemented in the next regular product release or with the delivery of the next (internal) test object version.
3 – ON OCCASION	The correction will take place when the affected system parts are due for a revision anyway.
4 – OPEN	Correction planning has not taken place yet.

Figure 10. Priority Levels (Spillner, Linz, & Schaefer, 2014)

4.8 Configuration Management

Configuration management will be used throughout the development of the E-Commerce Website in order to track the version history of the project, as well as to allow multiple developers to make contributions without interfering with one another's work. Poor configuration management can lead to a number of avoidable problems such as developers overwriting one another's code, the inability to integrate components because of unknown versions, and difficulty testing because changes to a component are untraceable or testers don't know which test cases belong to which version of a test object (Spillner, Linz, & Schaefer, 2014). These circumstances can be avoided by implementing version management, configuration identification, incident and change status control, and configuration audits. All of these activities can be achieved through the use of standard operating procedures and test management tools: "Modern file version control systems, such as git, mercurial, or svn implement a concept of revision or commit. This concept may be crucial to store program source code, as it allows users to view previous versions of stored files" (Dmitriev, Valter, & Kontsov, 2020).

5. Test Tools

5.1 Introduction

Test tools can ease the burden of manual software testing. Because the E-Commerce Website is a new build, and a fairly complex system, test tools should be acquired and utilized to increase test efficiency and reliability, reduce manual testing, and achieve what some manual tests cannot, such as load and performance testing (Spillner, Linz, & Schaefer, 2014). Before selecting test tools, a cost-benefit analysis should be conducted. The phases of testing for the E-Commerce Website may involve tools for test management and control, test specification, static testing, dynamic testing, and non-functional testing.

5.2 Test Management and Control Tools

Tools that support test management and control allow testers and project managers to document, prioritize, list, and maintain test cases (Spillner, Linz, & Schaefer, 2014). A test management tool that may be considered for development of the E-Commerce Website is Jira. Test management tools can be used to ensure that there are test cases to address each software requirement. A test tool with this functionality is considered a requirements management tool. Test execution tools are another form of test management tool that execute test scripts automatically and document the results. Incident management tools track software defects and their resolutions. Configuration management tools track versions and builds that will be tested. Lastly, tool integration can be used to combine multiple test management tools into one (Spillner, Linz, & Schaefer, 2014). A cost-benefit analysis specific to the E-Commerce Website should be used to determine which tools will be the most beneficial to the project.

5.3 Test Specification Tools

Tools for the specification phase of testing should also be considered regarding the E-Commerce Website. These tools help test designers generate test data. There are four types of test generators: database-based test generators, code-based test generators, interface-based test generators, and specification-based test generators. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), database-based test generators "process database schemas and are able to produce test databases from these schemas." These tools help generate test data by filtering through databases. An example of such a tool is DatProf, which supports Windows operating systems and generates synthetic data using most common database technologies (Software Testing Help, 2020). Code-based test generators use the source code to generate test data. While these tools can be helpful, they cannot detect faults caused by missing code. In addition, Spillner, Linz, & Schaefer (2014) state, "The use of code as a test basis for testing the code itself is in general a very poor foundation." However, code-based test generators can be helpful for regression testing. Interface-based test generators identify parameter domains, which can then be used for equivalence class partitioning and boundary value analysis to generate test cases (Spillner, Linz, & Schaefer, 2014). They are also helpful when testing API's and GUI's. Specification-based test generators synthesize test data from formal specifications or models.

5.4 Static Testing Tools

Static testing tools should be applied directly to the source code of the E-Commerce Website. Because they support the evaluation of source code, they can help identify defects early in the development process, before code is even executed. Review tools are a form of static testing that serve like checklists to help testers plan, execute, and evaluate code reviews (Spillner, Linz, & Schaefer, 2014). Static analyzers can also be used to identify areas in the code that are complex, risky, inconsistent, defect-prone, violate programming standards, or pose portability issues. These areas can then be refactored, and or prioritized for further testing. Model checkers may also be considered in order to analyze the UML models to check for "missing states, state transitions, and other inconsistencies in a model" (Spillner, Linz, & Schaefer, 2014). An example of a static analysis tool is Polyspace. This tool, released in 2014 consists of two components: Polyspace Code Prover and Polyspace Bug Finder. "The former identifies every code instruction by applying all possible values of variables, while the later is used to perform static analysis" (Khalid, 2017).

5.5 Dynamic Testing Tools

Utilizing dynamic testing tools during development of the E-Commerce Website will help reduce the mechanical work involved with manual test execution. These tools provide the test object with input data, and record the output. Examples of dynamic testing tools are debuggers, test drivers and test frameworks, simulators, test robots, comparators, dynamic analyzers, and tools for coverage analysis (Spillner, Linz, & Schaefer, 2014). Debuggers can be found in popular text editors and IDE's. They allow testers and developers to execute a program one line at a time. A program can also be paused mid-execution to observe or manipulate variables. Test drivers and test frameworks are often used for component and integration tests, and are designed for specific programming languages and environments (Spillner, Linz, & Schaefer, 2014). They provide control over the test object. Simulators mirror the real system environment, but are used when the tests cannot be executed there. Test robots log manual inputs such as keyboard strokes or mouse clicks, store them as a script, and then can execute the script in order to replay that sequence of inputs in an automated fashion. These tools are especially helpful for regression testing. Comparators identify deviations from expected results. Dynamic analyzers evaluate memory usage and allocation, as well as identify "memory leaks, wrong pointer allocation, or pointer arithmetic problems" (Spillner, Linz, & Schaefer, 2014). Finally, coverage analysis tools track statement and branch coverage to ensure that as much code as possible is executed during testing. An example of a dynamic analysis tool is VectorCast. "VectorCAST ICover is used to test code coverage of source code through dynamic analysis" (Khalid, 2017).

5.6 Non-Functional Testing Tools

Non-functional testing tools should also be considered for assessing the quality attributes of the E-Commerce Website. Often, these attributes can be difficult to evaluate using manual methods alone. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "Load test tools generate a synthetic load (i.e., parallel data-base queries, user transactions, or network traffic." This allows testers to determine how a system will behave under more realistic conditions. Performance tests can also reveal the response time of a system. Load and performance tools are especially helpful when a system expects to experience high traffic, or when it needs to handle large numbers of parallel requests (Spillner, Linz, & Schaefer, 2014). They can also be helpful in identifying bottlenecks within the system. An example of a tool used for performance and load testing is Apache JMeter (Khandelwal, 2019). Aside from performance and load testing tools, other non-functional testing tools include those for testing security and those for assessing data quality. It is important to identify security vulnerabilities early on in development. As Benchmark Requirements for Assessing Software Security Vulnerability Testing Tools (2018) states, "Developers cannot afford to believe that their security requirements during development are perfect and impenetrable, no matter how thorough their precautions might be." In addition to testing for security vulnerabilities, data quality assessment tools can ensure that data both before and after migration or conversion is correct and complete. Through the use of these testing tools, developers, testers, and project managers can produce an E-Commerce Website that is robust and reliable.

6. Resources

ADA. (2007). Website Accessibility Under Title II of the ADA. Retrieved April 5, 2020 from https://www.ada.gov/pcatoolkit/chap5toolkit.htm

Ali, S., Imran, M., Hafeez, Y., Abbasi, T. R., Haider, W., & Salam, A. (2018). Improving Component Based Software Integration Testing Using Data Mining Technique. 2018 12th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), Mathematics, Actuarial Science, Computer Science and Statistics (MACS), 2018 12th International Conference On, 1–6. https://doi-org.proxy- library.ashford.edu/10.1109/MACS.2018.8628368

Bahaweres, R. B., Zawawi, K., Khairani, D., & Hakiem, N. (2017). Analysis of statement branch and loop coverage in software testing with genetic algorithm. 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), Electrical Engineering, Computer Science and Informatics (EECSI), 2017 4th International Conference On, 1–6. https://doi-org.proxy-library.ashford.edu/10.1109/EECSI.2017.8239088

Carnegie Mellon Software Engineering Institute. (1991). The Software Technical Review Process. Retrieved April 18, 2020 from https://apps.dtic.mil/dtic/tr/fulltext/u2/a236122.pdf

Dmitriev, S. O., Valter, D. A., & Kontsov, A. M. (2020). System for Efficient Storage and Version Control of Arbitrary File Collections. 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2020 IEEE Conference Of, 295–298. https://doi-org.proxy-library.ashford.edu/10.1109/EIConRus49466.2020.9038922

Eriksson, U. (2012, April 5). Functional vs non functional requirements ReQtest. https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

Font Awesome. (n.d.) Font Awesome Icons. Retrieved April 6, 2020 from https://fontawesome.com/icons?d=gallery

Fox, C. (1999). Java Inspection Checklist. Retrieved April 18, 2020 from http://www.cs.toronto.edu/~sme/CSC444F/handouts/java_checklist.pdf

GeeksForGeeks.com. (n.d.). Software Engineering: Control Flow Graph (CFG). Retrieved April 18, 2020 from https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/

GeeksForGeeks.com. (n.d.). Software Engineering: Integration Testing. Retrieved April 11, 2020 from https://www.geeksforgeeks.org/software-engineering-integration-testing/

Google. (n.d.). Google Java Style Guide. Retrieved April 18, 2020 from https://google.github.io/styleguide/javaguide.html

Greene, V. (2018, April 8). The top 5 functional requirements for eCommerce websites Acro Blog. https://blog.acromedia.com/the-top-5-functional-requirements-for-ecommerce-websites

Homès, B. (2012). Fundamentals of Software Testing, John Wiley & Sons, Incorporated. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ashford-ebooks/detail.action?docID=1120766.

Created from ashford-ebooks on 2020-04-25 14:33:08.

Khalid, R. (2017). Towards an automated tool for software testing and analysis. 2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Applied Sciences and Technology (IBCAST), 2017 14th International Bhurban Conference On, 461–465. https://doi-org.proxy-library.ashford.edu/10.1109/IBCAST.2017.7868094

Khandelwal, A. (2019). Top 10 Non -Functional Testing Tools | Pros and Cons. Retrieved May 5, 2020 from https://testinggenez.com/non-functional-testing-tools/

Kumar, J. (2015). The e-commerce problem statement. In Apache Solr patterns. Packt Publishing. https://bit.ly/2Rn8inE

M. Parizi, R., Qian, K., Shahriar, H., Wu, F., & Tao, L. (2018). Benchmark Requirements for Assessing Software Security Vulnerability Testing Tools. 2018 IEEE 42nd Annual Computer Software and

Applications Conference (COMPSAC), Computer Software and Applications Conference (COMPSAC), 2018 IEEE 42nd Annual, COMPSAC, 01, 825–826. https://doi-org.proxy-library.ashford.edu/10.1109/COMPSAC.2018.00139

Mohamed Suffian, M. D., Fairul Rizal, F., Loo, F. A., Aman, N. F., & Bajuri, N. (2016). Software capability rating using system testing scores. 2016 IEEE Conference on Open Systems (ICOS), Open Systems (ICOS), 2016 IEEE Conference On, 105–110. https://doi-org.proxy-library.ashford.edu/10.1109/ICOS.2016.7881997

Nidagundi, P., & Novickis, L. (2016). Introduction to Lean Canvas Transformation Models and Metrics in Software Testing. Applied Computer Systems, 19(1), 30.

Software Testing Help. (2020). Top 10 Best Test Data Generation Tools in 2020. Retrieved May 5, 2020 from https://www.softwaretestinghelp.com/test-data-generation-tools/.

Spillner, A., Linz, T., & Schaefer, H. (2014). Software testing foundations: A study guide for the certified tester exam (4th ed.). Rocky Nook.

Wiegers, K. E. (1999). Software requirements specifications for <project> [Template]. https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc

Appendix A: Glossary

ADA- Americans with Disabilities Act of 1990

HTTP- hypertext transfer protocol

PII- Personally Identifiable Information

SRS- Software Requirements Specification

TCP- transmission control protocol

UI- User Interface

UX- User Experience