Daniel Coffey

4-2 Milestone: Unit Testing

CS 405 Secure Coding

02/06/2021

For this assignment I had the role of being a senior software developer on a team of software developers who are responsible for a large banking web application. My manager has recently learned about the best practice of creating unit tests for software and wants to see a full example of how it works. A developer on the team started creating a unit test for the std::vector class and managed to get the TextFixure and few tests completed before leaving for his vacation. My manager then tasked me with completing what the other developer started. He also wanted to see several tests provided, both positive and negative.

**Process:**

Looking at the provided code, there were plenty of changes that I needed to make. One step that I needed to make was to create a test to verify adding a single value on an empty collection. To do this step, I needed to ask myself, is the collection empty? If the collection is empty than the size must be zero. To check is the collection is empty, I utilized the ASSERT\_TRUE expression. This is a test that the assertion only succeeds if and only if the Boolean, or in this case the expression, evaluates to equal to true. If this fails, the test will terminate with an error report that includes the exact expression text. The expression set for this was if the collection is equal to the empty member function, meaning if the collection is empty, it will assert true. Next, I utilized ASSERT\_EQ, which assets that two expressions are equal using PartialEq. This function was essentially saying, if the collection is empty, then the collection size must be zero. Then an entry of 1 was added. ASSERT\_FALSE was then used to check if the collection was still empty. ASSERT\_FALSE tests the expression by evaluating it as a Boolean. The assertion only succeeds if the Boolean (expression) is equal to false. If the test fails, there is an error report that includes the exact expression text. I then utilized another ASSERT\_EQ to state if the collection is not empty, then its size must be equal to one. Once this was done, I was able to replicate the exact same thing for creating a test to verify adding five values to a collection. The only thing I did differently here was change the 1 to a 5.

My next task was to create a test to verify that the max size is greater than or equal to the size for 0, 1, 5, and 10 entries. Since 10 was the highest number, I decided to add 1 entry for the test. I added 11 entries and then used ASSERT\_TRUE to verify if the max size is greater than or equal to the size for all the entries. This looked like: ASSERT\_TRUE(collection->max\_size() >= 0); for zero, and the number would change for each entry it was being tested for.

After this, I was tasked with creating a test to verify that the capacity is greater than or equal to the size for 0, 1, 5, and 10 entries. To create the tests for this, I did the exact same thing as comparing to the max size, the only difference Is I changed the max\_size() method to capacity(). I added the 11 entries and then verified that the capacity is greater than or equal to the size for 0, 1, 5, and 10 entries. An example of how this looked for zero entries is: ASSERT\_TRUE(collection->capacity() >= 0);

Next, I created a test to verify that resizing increases the collection. To do this I initialized the entries by adding an entry. My next step was to initialize and declare the previous value to the collection size. I set this as the variable initialSize. Then, I needed to resize the container to hold the elements. For the test, I resized the collection to hold 15 elements. I then utilized ASSERT\_TRUE to verify if the collection size is larger than the initial size.

I then had to create a test to verify if resizing decreases the collection. This was a little challenging for me at first, but then I realized I can just do the opposite of what I previously did. I initialized the entries by adding 15 entries. I then used the variable initialSize again and initialized this variable and declared the previous value to the collection size. The container, being the collection, was then resized to hold one element. I then utilized ASSERT\_TRUE if the collection size is smaller than the initial size.

The next step I took was to create a test to verify that resizing decreases the collection to zero. The first step I took was adding 15 entries (elements). After this I initialized the variable initialSize and declared the previous value to the collection size. The next step was to resize the collection to zero. I then utilized ASSERT\_TRUE to verify that the size has decreased the collection to zero. This looked like: ASSERT\_TRUE(collection->size() == 0);

Next, I created a test to verify that clear erases the collection. I started this off by adding 15 entries. I then used the clear method on the collection. I then utilized the ASSERT\_TRUE function to verify that the collection size has been resized to zero, verifying that clear erases the collection. This looks like: ASSERT\_TRUE(collection->size() == 0); Once this was done, my next step was to create a test to verify that the collection was erased from beginning to the end. The first step was to add all entries, being 15. Once this was done, I used the iterator erase to erase the collection ranging from the beginning to the end. I then utilized the ASSERT\_TRUE method to verify the collection size has been resized to zero. If the collection size is equal to zero, that means the collection is empty and the collection has in fact, been erased.

I then had to create a test to verify the reserve increases the capacity only, but not the size of the collection. This one was a bit complicated for me and I got a little stuck, but I was eventually able to figure it out. I started the same way as I have been, and this was by adding the entries. Once this was done, I created two variables, one for the initial capacity and one for the initial size. Once this was done, I then requested for the container to be reserved to at least 50. After this, I had to verify that the size is equal, but the capacity is larger. I used the ASSERT\_TRUE method here and set the collection size equal to the initial size. Then I used the ASSERT\_TRUE method again and set the collection capacity greater than the initial capacity. I had a hard time differentiating between the capacity and size while working with this, but it wasn’t until I utilized the initial capacity variable where I was able to get this test properly working.

My next step was to create a test to verify that the std::out\_of\_range exception is thrown when calling at() with an index out of bounds. This was different than the rest of the tests so far because this is supposed to be a negative test. To start, I defined a vector with the size of 9 elements. I then expected the out\_of\_range exception to be thrown upon calling my vector with out of bound access. This looked like: EXPECT\_THROW(elements.at(10), std::out\_of\_range);

The last step of this assignment was to create two more unit tests. I wanted to step out of my comfort zone and challenge myself by doing something a little different than what was provided. For the first test, I created a test to add an element to the collection. To start this test, I added entries into the collection. I then used the vector push\_back method to add a new element at the end of the vector, after its current last element. This effectively increases the container size by one, which causes an automatic reallocation of the allocated storage space, if and only if, the new vector size surpasses the current vector capacity. To check this, I utilized the ASSERT\_FALSE method to test if the collection size is equal to five. I have been utilizing ASSERT\_TRUE this whole time, so I wanted to try using ASSERT\_FALSE. The last test involved using the vector assign method. I started by adding five entries to the vector collection. The next step was to use collection->assign(5,100); this adds 5 elements, with 100 for the value. What the vector assign method does is add new contents to the vector, replacing its current contents, and modifying its size accordingly. So, to test for this I had to ASSERT methods, one for true and one for false. The ASSERT\_FALSE method was for if the collection size was equal to zero, checking if it was empty and the ASSERT\_TRUE method was for if the collection size was equal to 5.

Text

Description automatically generated with medium confidence