**Homework: Test Levels and Test Types**

**Contents:**

[I. Unit Testing 2](#_Toc130850588)

[1. Unit Testing in the Real Life: Testing a Battery 2](#_Toc130850589)

[2. Unit Testing in the Real Life: Testing a Light Bulb 3](#_Toc130850590)

[3. Unit Testing in the Software World: Age Checker 3](#_Toc130850591)

[4. Unit Testing in the Software World: Income Checker 4](#_Toc130850592)

[4.1 Functional tests 4](#_Toc130850593)

[4.2 Non-Functional tests 5](#_Toc130850594)

[II. Integration Testing 6](#_Toc130850595)

[5. Integration Testing in the Real Life: Lighting the Bulb 6](#_Toc130850596)

[6. \* Integration Testing in the Software World: Ads 7](#_Toc130850597)

[7. \* Integration Testing in the Software World: Credit Risk 10](#_Toc130850598)

[III. System Testing 12](#_Toc130850599)

[8. System Testing in the Real Life: Flashlight 12](#_Toc130850600)

[9. System Testing in the Real Life: Digital Scale 13](#_Toc130850601)

[10. System Testing in the Software World: Number Calculator 13](#_Toc130850602)

[IV. Acceptance Testing 16](#_Toc130850603)

[11. Acceptance Testing in the Real Life: Flashlight 16](#_Toc130850604)

[12. Acceptance Testing in the Real Life: Digital Scale 17](#_Toc130850605)

[13. Acceptance Testing in the Software World: Number Calculator 18](#_Toc130850606)

[14. Functional and Non-Functional Tests: Flashlight 18](#_Toc130850607)

[V. ChatGPT extractions 19](#_Toc130850608)

[1. Functional Tests 19](#_Toc130850609)

[2. Non-Functional Tests 19](#_Toc130850610)

[3. Credit Risk Calculator Testing - Automated 20](#_Toc130850611)

[3.1 Python 20](#_Toc130850612)

[3.2 C# 21](#_Toc130850613)

|  |  |  |
| --- | --- | --- |
| **Types**  **Levels** | ***Functional*** | ***Non-Functional*** |
| ***Acceptance*** *(Business, End-user – α, β)* | ● | ● |
| ***System*** *(Thorough, behavior + capabilities, end-user perspective)* | ● | ● |
| ***Integration*** *(Specification, collaboration, internal, external)* | ● | ● |
| ***Unit*** *(developers, isolation)* | ● | ● |

1. Unit Testing

## Unit Testing in the Real Life: Testing a Battery

|  |  |
| --- | --- |
| * Tests in **isolation** | |
| **Test #1** | Check if **physical characteristics** of the battery correspond to its technical **specification**:   * size (dimensions); * shape; * surface; * specific ridges. |
| **Test #2** | Check if **physical condition** of the battery is according to its **specification** – does it have any **deformations**, is it **leaking**, does it produce any **smell**, **heat**, does it have any **substances** on its surface. |
| **Test #3** | Check if the **labels** on the battery – volts (1.5V) and mWh (3300mWh) are correct and responding to its **specification** – with appropriate **equipment** (multimeter/voltmeter/volt amper meter).  Additionally check if the expiration date is in the future. |
| **Test #4** | Check if it is being **produced** in a **durable** manor – does it melt when heated/used as intended, does it deform when small force is exercised. |
| **Test #5** | Check if the battery **is** **charged** – let it fall from a small height onto the table and desk (1-2 cm). If it stays straight, it has a charge, if it falls, it doesn't. |
| *Tests in* ***normal*** *working* ***conditions*** *– with an appropriate device (equipment), e.g. a flashlight* | |
| **Test #6** | Test if the battery **conducts** electricity in the **right** **direction** (when put correctly in the flashlight with +/- in the right places). With multometer may be |
| **Test #7** | Test if the battery **works** in **normal** **conditions**, mentioned in its technical specification – humidity, altitude, temperature.  *“****Check****”* means to test if it heats up, does it melt, does it produce smell, does it cause defect of the devise used for testing, does it produce sparks, etc. |
| *Tests in* ***extreme*** *working* ***conditions*** | |
| **Test #8** | All the same as above, but in **extreme** temperatures, etc. |
| *Tests in* ***extraordinary*** *working* ***conditions*** | |
| **Test #9** | All the same as above, but in **unusual** conditions - fire, pressure, electromagnetic field, water (or other type of liquid), wrapped in different material. |
| *Tests after* ***incidents*** | |
| **Test #10** | All the same as above, but after **dropping** it from different heights, **pressing** it with hard object, **flood** it with different liquids, putting it **incorrectly** in the device. |
| ***Non-functional*** *tests* | |
| **Test #11** | Test the **performance** of the battery in **any of the above** conditions and **periods of time** –how **strongly** it lights, does it **change** in any way – visually and by touch, does it cause **malfunction** of the device, how many **hours** does it work **properly**, what happens when the charge is **running** **out** (low). |

## Unit Testing in the Real Life: Testing a Light Bulb

Almost same as above.

|  |  |
| --- | --- |
| Functional tests | |
| **Test #1** | **Power supply testing:** This test verifies that the light bulb receives power and lights up when it's connected to a power source. This test checks the functionality of the bulb's electrical components. |
| **Test #2** | **Light output testing:** This test checks the brightness and quality of the light produced by the bulb. It measures the amount of light output in lumens or lux and verifies that the light produced by the bulb meets the manufacturer's specifications. |
| **Test #3** | **Lifespan testing:** This test checks the bulb's lifespan by turning it on and monitoring how long it takes before it burns out. This test confirms that the bulb can last for an adequate amount of time before requiring replacement. |
| Non-functional tests | |
| **Test #4** | **Visual inspection**: This test visually inspects the bulb for any physical damage, such as cracks or broken filaments. It checks for any visual defects that could affect the bulb's functionality. |
| **Test #5** | **Durability testing:** This test checks the bulb's ability to withstand environmental factors, such as vibrations or temperature changes. It tests the bulb's durability to ensure that it can function properly under various conditions. |

## Unit Testing in the Software World: Age Checker

|  |  |  |
| --- | --- | --- |
| **Down limit** | **Upper limit** | **Expected result** |
| 0 | 12.9 | Child |
| 13 | 19.9 | Teenager |
| 20 | 64.9 | Adult |
| 65 | 149.9 | Elder |
| <0 | | Error |
| ≥150[[1]](#footnote-1) | | Error |

**N.B. In specification it is said “If the age is negative or above 150, returns “error” “, but when 150 is eneterd, Age Checker gives error, which is not the expected result. The specification should say “equal or above 150” or “ 150 inclusively”**

| **№** | **Result** | **Test cases** | **Reason to fail** |
| --- | --- | --- | --- |
| **Test #1** | **Pass** | **Child** cases: 0, 12, 0.1, 11.9, 6, 5.5, 12.4, 11.5, 12.5, 12.6, 12.7, 12.8, 12.9  → Expect true |  |
| **Test #2** | **Pass** | **Teenager** cases: 13, 19, 16, 19.4, 18.5, 19.5, 19.6, 19.7, 19.8, 19.9  **→** Expect **true** |  |
| **Test #3** | **Pass** | **Adult** cases: 20, 64, 63.5, 40, 64.5, 64.6, 64.7, 64.8, 64.9  **→** Expect **true** |  |
| **Test #4** | **Fail** | **Elder** cases: 65, 150, 150.1, 150.2, 150.3, 150.4, 150.5, 150.6, 150.7, 150.8, 150.9, 149.5 **→** Expect **true** | Above 150  /”equal” missing/ |
| **Elder** case: 64.99999999999 **→** Expect **false** | 65 “exclusively” |
| **Test #5** | **Pass** | **Error** cases: negative numbers: -1, -10, -1500, -0.2, -7.3 **→** Expect **true** |  |
| **Test #6** | **Pass** | **Error** cases: **bigger** than 150: 151, 1500, 2654 **→** Expect **true** | Above 150  /”equal” missing/ |
| **Test #7** | **Pass** | Leave the field **empty →** expect to receive a **hint** to enter valid age. I do. |  |
| **Task #1** | Enter **symbols** **different** **from** **digits** – “s”, “%”, “@”, ;”” **→** expect to receive a **hint** to enter valid age. Instead nothing happens. | | No hint |
| **Task #2** | Try to enter **age** with another symbol for **decimals** (instead of [.]) **→** Expect possible  Proposal to allow additional decimal symbol – [,]. | | Not possible |
| **Task #3** | After entering valid age I try to go to **[CHECK AGE]** with **[Tab]** **→** Expect to be **shown** in any way that **[CHECK AGE]** is selected. | | Not possible |

## Unit Testing in the Software World: Income Checker[[2]](#footnote-2)

|  |  |  |
| --- | --- | --- |
| **Down limit** | **Upper limit** | **Expected result** |
| 0 | 999 | Low |
| 1000 | 2999 | Mid |
| 3000 | ∞ | High |
| <0 | | Error |

* 1. Functional tests

|  |  |
| --- | --- |
| **Test #1** | Pass: Low cases: 0, 999, 450; Negative tests: 1000 |
| **Test #2** | Pass: Mid cases: 1000, 2999,1500; Negative tests: 999, 3000 |
| **Test #3** | Pass: High cases:3000, 9999999; Negative cases: 2999 |
| **Test #4** | Pass: Error cases: -5, -5.5 |
| **Test #5** | Fail: 999.6 – Expect “Mid”, receive “Low”; 2999.6 – Expect “High”, receive “Mid”. |
| **Task #1** | Currently there isn’t upper limit for income input. Maybe there should be. |

* 1. Non-Functional tests

|  |  |
| --- | --- |
| **Test #1** | Performance  Generate a large dataset (e.g., 100,000 entries) of random inputs and measure the time it takes for the function to complete the calculations. |
| **Test #2** | Security  Verify that the function does not leak sensitive data, such as the input value or output category, in any way. |
| **Test #3** | Usability  Verify that the function returns an appropriate error message when an invalid input is provided. |
| **Test #4** | Integration  Verify that the function integrates well with other parts of the system, such as the UI or the database. |

1. Integration Testing

## Integration Testing in the Real Life: Lighting the Bulb

Battery **↔** Wire

Light Bulb **↔** Wire

Switch **↔** Wire

|  |  |
| --- | --- |
| **Test #1** | Implement the following circuit, using the provided unit-tested components.  A picture containing shape  Description automatically generated  The bulb should **light**. |
| **Test #2** | Implement the following circuit, using the provided unit-tested components.  Diagram  Description automatically generated  Switch **on** the switch button **→** The bulb should **light**. |
| **Test #3** | Implement the following circuit, using the provided unit-tested components.  Diagram  Description automatically generated  Switch **off** the switch button **→** The bulb should **not** **light**. |
| * Negative tests | |
| **Test #4** | Implement the following circuit, using the provided unit-tested components.  A picture containing shape  Description automatically generated   * The wires aren’t connected properly to the bulb **→** The bulb should **not** **light**. * The wires aren’t connected properly to the battery **→** The bulb should **not** **light**. |
| **Test #5** | Diagram  Description automatically generated   * The wires aren’t connected properly to the bulb **→** The bulb should **not** **light**. * The wires aren’t connected properly to the battery **→** The bulb should **not** **light**. * The wires aren’t connected properly to the switch **→** The bulb should **not** **light**. |

## \* Integration Testing in the Software World: Ads

| **№** | **Description** |
| --- | --- |
| **Test #1** | **Home page:**     1. Test if **[Home]** button redirects me to the **same** page: “**Ads – Home**” **→** Expect **true** 2. Test if **[Login]** button redirects me to the “**Login**” page: “**Ads – Login**” **→** Expect **true** 3. Test if **[Register]** button redirects me to the “**Register**” page: “**Ads – Register**” **→** Expect **true** 4. Test if **buttons for navigation** through **pages** works as expected**: [First], [Previous], [1], … , [Next], [Last].** 5. Test if **buttons for navigation** through **Categories** works as expected**: [Cars], [Jobs]**, etc. 6. Test if **buttons for navigation** through **Towns** works as expected**: [Plovdiv], [Sofia]**, etc. |
| **Test #2** | **Login page:**     1. Test if **[Home]** button redirects me to the **“Ads – Home”** page **→** Expect **true** 2. Test if **red** **[Login]** button redirects me to the **same** page: “**Ads – Login**” **→** Expect **true** 3. Test if **blue** **[Login]** button redirects me to the **User Home** page: “**Ads – Home**” **→** Expect **true** 4. Test if **[Register here]** button redirects me to the “**Register**” page: “**Ads – Register**” **→** Expect **true** 5. Test if **[Register]** button redirects me to the “**Register**” page: “**Ads – Register**” **→** Expect **true** |
| **Test #3** | **User Home page:**     1. Test if **[Home]** button redirects me to the **same** page (User): “**Ads – Home**” **→** Expect **true** 2. Test if **[Logout]** button redirects me to the “**Home**” page (not User): “**Ads – Home**” **→** Expect **true** 3. Test if **buttons for navigation** through **User Ads** work as expected**: [My Ads], [Publish New Ad]** and **[Edit Profile].** 4. Test if **buttons for navigation** through **pages** work as expected**: [First], [Previous], [1], … , [Next], [Last].** 5. Test if **buttons for navigation** through **Categories** work as expected**: [Cars], [Jobs]**, etc. 6. Test if **buttons for navigation** through **Towns** works as expected**: [Plovdiv], [Sofia]**, etc. |

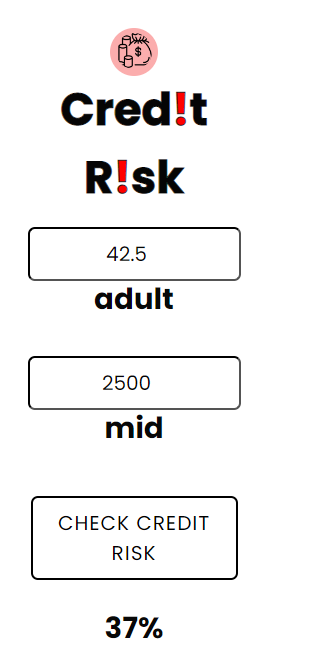
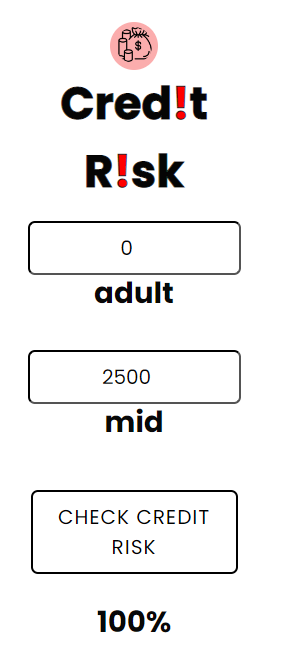
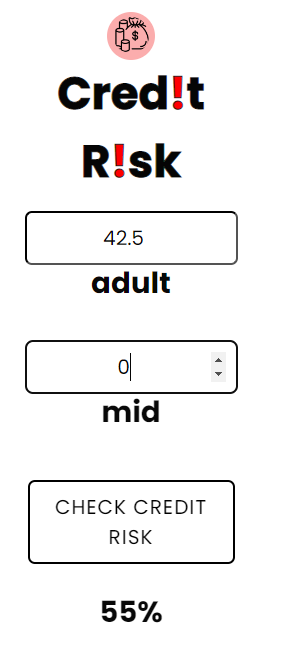
## \* Integration Testing in the Software World: Credit Risk[[3]](#footnote-3)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Age:**  **Income:** | child | teenager | adult | elder | negative |
| Low | Expect **100%** 🡪 **Pass**  Borderchild:0  Borderchild:12.9  Midchild: 6.5  Borderlow:0  Borderlow:999  Midlow: 475 | Expect **80%** 🡪 **Pass**  Borderteenager:13  Borderteenager:19.9  Midteenager:16.5  Borderlow:0  Borderlow:999  Midlow: 475 | Expect **55%** 🡪 **Pass**  Borderadult:20  Borderadult:64.9  Midadult:42.5  Borderlow:0  Borderlow:999  Midlow: 475 | Expect **60%** 🡪 **Pass**  Borderelder:65  Borderelder:149.9  Midelder:115.5  Borderlow:0  Borderlow:999  Midlow: 475 | **Pass**  Borderlow:-5  Borderlow:1000  All Age negatives |
| mid | Expect **100%** 🡪 **Pass**  Borderchild:0  Borderchild:12.9  Midchild: 6.5  Bordermid:1000  Bordermid:2999  Midmid:2000 | Expect **72%** 🡪 **Pass**  Borderteenager:13  Borderteenager:19.9  Midteenager:16.5  Bordermid:1000  Bordermid:2999  Midmid:2000 | Expect **37%** 🡪 **Pass**  Borderadult:20  Borderadult:64.9  Midadult:42.5  Bordermid:1000  Bordermid:2999  Midmid:2000 | Expect **44%** 🡪 **Pass**  Borderelder:65  Borderelder:149.9  Midelder:115.5  Bordermid:1000  Bordermid:2999  Midmid:2000 | **Pass**  Bordermid:999  Bordermid:3000  All Age negatives |
| high | Expect **100%** 🡪 **Pass**  Borderchild:0  Borderchild:12.9  Midchild: 6.5  Borderhigh:3000  Borderhigh:10000  Midhigh:5500 | Expect **64%** 🡪 **Pass**  Borderteenager:13  Borderteenager:19.9  Midteenager:16.5  Borderhigh:3000  Borderhigh:10000  Midhigh:5500 | Expect **19%** 🡪 **Pass**  Borderadult:20  Borderadult:64.9  Midadult:42.5  Borderhigh:3000  Borderhigh:10000  Midhigh:5500 | Expect **28%** 🡪 **Pass**  Borderelder:65  Borderelder:149.9  Midelder:115.5  Borderhigh:3000  Borderhigh:10000  Midhigh:5500 | **Pass**  Borderhigh:2999  Borderhigh:987654  All Age negatives |
| negative | **Pass**  Borderchild:-1  Borderchild:13  All Income negatives | **Pass**  Borderteenager:12.9  Borderteenager:20  All Income negatives | **Pass**  Borderadult:19.9  Borderadult:65  All Income negatives | **Pass**  Borderelder:64.9  Borderelder:150[[4]](#footnote-4)  All Income negatives |  |

**Example:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Test Description** | **Age group** | **Income group** | **Result** | **Pass / Fail** |
| **Test #1** | CreditRisk(age: 35, income: 2300) 🡪 37% | adult 🡪 10% | mid 🡪 30% | 37% | Pass |

**Bug:** When entering 0 in either “**Age**” or “**Income**” field, the **label** below the fields **remains the same**, does not refresh. Nevertheless **the calculation** of the Credit Risk **is** **correct**.

 ** **

1. System Testing

## System Testing in the Real Life: Flashlight

|  |  |
| --- | --- |
| **Test #1**  **(Hints - Functional)** | **Functional**   1. Test **switch on / switch off the** light – with the **button**; with putting one battery out – does it work as expected 2. Test **battery** replacement – does it work with **different** batteries of one and the same type; 3. Test **bulb** replacement – does it work with **different** bulb of one and the same type; 4. Test battery **duration** – does it last as in the **specification** in different environment **conditions** (temperature, pressure, humidity, altitude); if it has **levels** / **modes** of lighting – test them all; 5. Test the illumination **distance** – does it work in different **environment** conditions; with often changing of different **levels** / **modes** of lighting (if there are such); 6. **Shock** **resistance** test – hit, drop, flood, heat, freeze it and then test if it works (for functional tests) |
| **Test #2**  **(Hints – Non-functional)** | **Non-functional**   1. Test **switch on / switch off the** light – with the **button**; with putting one battery out – how **fast** does it light on / light off, does it start with slightly **lower** light **intensity** which grows over time 2. Test **battery** replacement – does it light **differently** with **different** batteries of one and the same type; 3. Test **bulb** replacement – does it light **differently** with **different** bulb of one and the same type; 4. Test battery **duration** – in different environment **conditions** (temperature, pressure, humidity, altitude); if it has **levels** / **modes** of lighting – test them all – **how** **long** does it last 5. Test the **illumination** **distance** – in different **parts** of the day; in different **environment** conditions; with often changing of different **levels** / **modes** of lighting (if there are such) – **how much** is it vsible 6. **Shock** **resistance** test – hit, drop, flood, heat, freeze it and then test how it works – all of the above tests might take place |
| **Test #3** | **Performance under different conditions:** All of the above – outdoors/indoors, temperature, altitude, electromagnetic foelds, humidity. |
| **Test #4** | **Negative** tests – with wrong batteries, wrong bulb, putting appropriate batteries / bulb in the wrong way; without batteries / bulb **→** expect **not** to work. |
| **Test #5** | Conduct **safety** testing: Ensure that it does not pose any hazards to users.   * **temperature** of the flashlight during **extended** use * checking for any **sharp** edges or points * ensuring that the flashlight does not emit any **harmful** **radiation** or **chemicals** |

## System Testing in the Real Life: Digital Scale

|  |  |
| --- | --- |
| **Test #1** | Test on different surfaces |
| **Test #2** | Test with quick change of different loads |
| **Test #3** | Test with slight jump |
| **Test #4** | Test with overload |
| **Test #5** | Test with load below the down limit |
| **Test #6** | Test with object that has large/small surface |
| **Test #7** | Test with very light (1kg) / heavy (80kg) object in variable positions of the scale |
| **Test #8** | Test it after being upside down, sideways |
| **Test #9** | Test it after unusual pressure has been exercisesd on it |
| **Test #10** | Test in different environmental conditions: temeprature, altitude, pressure, humidity |
| **Test #11** | Test it after being dropped |
| **Test #12** | Test it in various positions |
| **Test #13** | Test with different batteries – type, charge |

## System Testing in the Software World: Number Calculator

|  |  |
| --- | --- |
| **Test #1** | **Positive** test: **Valid** **integers (incl. decimals)** for **all operations (sum, subtract, multiply, divide)**: 🡪 expect **true** |
| **Test #2** | **Positive** test: **Valid** **exponential** numbersfor **all operations (sum, subtract, multiply, divide)**: 🡪 expect **true** |
| **Test #3**  **BUG!** | **Negative** test: **Invalid** **inputs**: 🡪 expect **false**    **Bug**: When entering an **expression** in the field for **number**, I expect to receive a **Result**: *“invalid input”*. Instead I receive a calculation of the first digits entered in the number field.    **Bug**: When entering adecimal number with “,” instead of “.” For decimal separator in the field for **number**, I expect to receive a **Result**: *“invalid input”* OR have the calculation done. Instead I receive a calculation of the first digits entered in the number field. |
| **Test #4** | **Negative** test: **Invalid** **operations**: 🡪 expect **false** (Result**:** *“invalid operation”*) |
| **Test #5**  **BUG!** | **Negative** test: **Division by 0**: 🡪 expect **false[[5]](#footnote-5)**  **Bug:** When **dividing number to 0** I expect to receive a message like**: "Error: division by zero".** Instead I get **“Infinity”.** |
| **Test #6** | Test rounding and precision: Ensure that the calculator rounds and displays results with the correct precision. For example:  1 / 3 = 0.33333 (with the display rounded to a certain number of decimal places) |
| **Test #7** | Test decimal and floating-point calculations: Ensure that the calculator can handle decimal and floating-point calculations accurately. For example:   * 1.5 + 2.5 = 4.0 * 1.1 \* 3 = 3.3 |
| **Test #8**  **BUG!** | Test with big numbers:    **Bug**: **Incorrect** calculation:  FirstNumber: **655500000000000000**  Operation: **+ (sum)**  SecondNumber: **8** should give  Result: **655500000000000008**.  It gives 655500000000000000, therefore we might assume it doesn’t work with big numbers. |

1. Acceptance Testing

## Acceptance Testing in the Real Life: Flashlight

|  |  |
| --- | --- |
| **Test #1** | Does it have logo, overall is the design according to specification |
| **Test #2** | Does it have labels for special certificates |
| **Test #3** | Is it easy to change batteries – are there user friendly hints, are they specific / hard to find |
| **Test #4** | Is it easy to change the bulb – are there user friendly hints, are they specific / hard to find |
| **Test #5** | Is it easy to press the button – is it easy to find it with one hand, does it have distinguishable surface |
| **Test #6** | Is it safe to use – no sharp edges, no leaks, no heat, etc. |
| **Test #7** | Is it ergonomical for holding in one hand |
| **Test #8** | Is it made of sustainable materials – will it last long – body, glass, internal parts |
| **Test #9** | Is it nature friendly – e.g. made of recycled materials |
| **Test #10** | How long does it light (10h, 20h, 30h) |
| **Test #11** | How far does it light (5m, 10, 50m, 100m) |
| **Test #12** | Is it shock/water/heat resistant/proof |
| **Test #13** | How much does it weight (200g,500g) |
| **Test #14** | How big is it (20cm/5cm) |
| **Test #15** | Is it easy to put / hang it from behind on a backpack / clothe |
| **Test #16** | Is it easy to put it straight on a flat surface and stay there without falling |
| **Test #17** | Does it have a timer to switch off after certain amount of time (1h, 2h, 12h) |
| **Test #18** | How fast does it light up / switch off after pressing the switch button (100ms, 1sec, 5sec) |
| **Test #19** | Does it have a reserve way of switching on if button doesn’t work properly |
| **Test #20** | Does it have a SOS signal mode (or any other) |
| **Test #21** | Does it have kids protectition |
| **Test #22** | Does it have low battery indication |
| **Test #23** | Does it have tactile feedback mechanism, such as raised dot or texture to indicate the location of the power button. |

## Acceptance Testing in the Real Life: Digital Scale

|  |  |
| --- | --- |
| **Test #1** | Is the design as intended AND satisfactory |
| **Test #2** | How fast could a couple of people weight themselves one after another |
| **Test #3** | Are the up and down limits appropriate (1kg – 180kg, 0.5kg – 200kg) |
| **Test #4** | Does it have indicators / hints for incorrect use |
| **Test #5** | Does it have labels for special certificates |
| **Test #6** | Is it easy to change batteries – are there user friendly hints, are they specific / hard to find |
| **Test #7** | Is it safe to use – no sharp edges, etc. |
| **Test #8** | Is it made of sustainable materials – will it last long – body, glass, internal parts |
| **Test #9** | Is it nature friendly – e.g. made of recycled materials |
| **Test #10** | How long do the batteries last (1week, 4 weeks, 3 months) |
| **Test #11** | Is it shock/water/heat resistant/proof |
| **Test #12** | How much does it weight (0.5kg, 1kg, 2kg) |
| **Test #13** | How big is it |
| **Test #14** | Does it have a timer to switch off after certain amount of time |
| **Test #15** | How fast does it switch on / switch off after stepping on it, and then – show weight |
| **Test #16** | Does it have kids protectition |
| **Test #17** | Does it have low battery indication |
| **Test #18** | Does it have modes for different materials / substances |
| **Test #19** | Does it show correct weight on different surfaces |
| **Test #20** | Does it show correct weight after being put upside down or sideways |
| **Test #21** | Does it produce funny sounds when shaken |

## Acceptance Testing in the Software World: Number Calculator

|  |  |
| --- | --- |
| **Test #1** | Usability testing: Test the usability of the calculator by conducting user tests with individuals who have varying levels of technical expertise to ensure that the calculator is easy to use and intuitive. |
| **Test #2** | Accessibility testing: Test the accessibility of the calculator by ensuring that it is compatible with assistive technologies such as screen readers and keyboard-only navigation. |
| **Test #3** | Compliance testing: Test the compliance of the calculator with industry standards and regulations, such as the Americans with Disabilities Act (ADA) or General Data Protection Regulation (GDPR). |
| **Test #4** | Localization testing: Test the localization of the calculator by ensuring that it supports different languages and date/time formats. |
| **Test #5** | Performance testing: Test the performance of the calculator by measuring its speed and responsiveness under different loads and scenarios. |
| **Test #6** | Security testing: Test the security of the calculator by ensuring that it is protected against potential security vulnerabilities such as injection attacks or cross-site scripting (XSS) attacks. |
| **Test #7** | User acceptance testing: Test the user acceptance of the calculator by involving end-users in the testing process to ensure that the calculator meets their needs and requirements. |

## Functional and Non-Functional Tests: Flashlight

For this one please see on page 9 - 8: System Testing in the Real Life: Flashlight and on page 11 above: 11 Acceptance Testing in the Real Life: Flashlight

|  |  |
| --- | --- |
| **Functional Tests** | **Non-Functional Tests** |
|  |  |
|  |  |
|  |  |
|  |  |

1. ChatGPT extractions

## Functional Tests

* **Unit** Testing: Testing individual units of code or modules to ensure that each one performs as intended. Example: Testing a function that calculates the average of a set of numbers.
* **Integration** Testing: Testing how different units of code or modules work together. Example: Testing how the database module interacts with the user interface module.
* **System** Testing: Testing the entire software system to ensure that it meets the requirements and performs as intended. Example: Testing a web application to ensure that all the pages and functions work correctly.
* **Acceptance** Testing: Testing the software system to ensure that it meets the end-user's requirements and expectations. Example: Testing a mobile app to ensure that it is easy to use and performs as expected.
* **Regression** Testing: Testing the software system to ensure that changes made to it do not cause any unintended side effects or defects. Example: Testing an updated version of a software system to ensure that it works as expected and does not break any existing functionality.

## Non-Functional Tests

* **Performance** Testing: Testing the software system's performance under various workloads to ensure that it performs well. Example: Stress testing a web application to ensure that it can handle a large number of concurrent users.
* **Security** Testing: Testing the software system's security to ensure that it is secure from external threats. Example: Penetration testing a mobile app to identify any vulnerabilities that could be exploited by attackers.
* **Usability** Testing**:** Testing the software system's ease of use to ensure that it is user-friendly. Example: Conducting a user study to determine how easy a mobile app is to use.
* **Compatibility** Testing: Testing the software system's compatibility with different hardware and software configurations. Example: Testing a web application on different browsers and operating systems.
* **Exploratory** Testing: Testing the software system without a script or plan to find defects or areas for improvement. Example: Testing a mobile app by randomly exploring its features and functions.
* **Localization** Testing: Testing the software system's localization to ensure that it works correctly in different languages and cultures. Example: Testing a software system in different languages and character sets.
* **Accessibility** Testing: Testing the software system's accessibility to ensure that it can be used by people with disabilities. Example: Testing a website to ensure that it can be used by people who are visually impaired.
* **Installation** Testing: Testing the software system's installation process to ensure that it installs correctly and works as intended. Example: Testing the installation of a desktop application to ensure that it installs without any issues.
* **Configuration** Testing: Testing the software system's configuration to ensure that it works correctly in different environments. Example: Testing a web application on different servers and network configurations.
* **Recovery** Testing: Testing the software system's ability to recover from failures or crashes. Example: Testing a mobile app to ensure that it can recover from unexpected errors.
* **Reliability** Testing: Testing the software system's reliability to ensure that it performs consistently and without errors. Example: Testing a software system for a long period of time to ensure that it is reliable.
* **Disaster** **Recovery** Testing: Testing the software system's ability to recover from disasters such as natural disasters or cyber attacks. Example: Testing a backup and recovery system to ensure that it can recover data after a disaster.
* **Endurance** Testing: Testing the software system's ability to handle sustained workloads over an extended period of time. Example: Testing a web application to ensure that it can handle a constant flow of requests over several hours.
* **Stress** Testing: Testing the software system's ability to handle extreme workloads or unexpected events. Example: Testing a software system to see how it performs under high traffic or load

## Credit Risk Calculator Testing - Automated

* 1. Python
* **Code**

def calculate\_credit\_risk(age, monthly\_income):

    age\_risk = 0

    income\_risk = 0

    age\_category = AgeChecker(age)

    if age\_category == "child":

        age\_risk = 1

    elif age\_category == "teenager":

        age\_risk = 0.6

    elif age\_category == "adult":

        age\_risk = 0.1

    elif age\_category == "elder":

        age\_risk = 0.2

    income\_category = IncomeChecker(monthly\_income)

    if income\_category == "low":

        income\_risk = 0.5

    elif income\_category == "mid":

        income\_risk = 0.3

    elif income\_category == "high":

        income\_risk = 0.1

    credit\_risk = age\_risk + income\_risk - (age\_risk \* income\_risk)

    return credit\_risk if credit\_risk >= 0 else "error"

* **Tests**

import unittest

from credit\_risk import calculate\_credit\_risk

class TestCreditRisk(unittest.TestCase):

    def test\_calculate\_credit\_risk\_teenager\_low(self):

        self.assertAlmostEqual(calculate\_credit\_risk(14, 700), 0.8, delta=0.001)

    def test\_calculate\_credit\_risk\_elder\_mid(self):

        self.assertAlmostEqual(calculate\_credit\_risk(85, 1600), 0.44, delta=0.001)

    def test\_calculate\_credit\_risk\_adult\_high(self):

        self.assertAlmostEqual(calculate\_credit\_risk(30, 3500), 0.19, delta=0.001)

    def test\_calculate\_credit\_risk\_negative\_income(self):

        self.assertRaises(ValueError, calculate\_credit\_risk, 20, -50)

    def test\_calculate\_credit\_risk\_error\_age(self):

        self.assertRaises(ValueError, calculate\_credit\_risk, 1000, 1000)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

* 1. C#
* **Code**

public static double CalculateCreditRisk(int age, double monthlyIncome)

{

    double ageRisk = 0;

    double incomeRisk = 0;

    string ageCategory = AgeChecker(age);

    switch (ageCategory)

    {

        case "child":

            ageRisk = 1;

            break;

        case "teenager":

            ageRisk = 0.6;

            break;

        case "adult":

            ageRisk = 0.1;

            break;

        case "elder":

            ageRisk = 0.2;

            break;

    }

    string incomeCategory = IncomeChecker(monthlyIncome);

    switch (incomeCategory)

    {

        case "low":

            incomeRisk = 0.5;

            break;

        case "mid":

            incomeRisk = 0.3;

            break;

        case "high":

            incomeRisk = 0.1;

            break;

    }

    double creditRisk = ageRisk + incomeRisk - (ageRisk \* incomeRisk);

    return (creditRisk >= 0) ? creditRisk : double.NaN;

}

* **Tests**

[TestClass]

public class CreditRiskCalculatorTests

{

    [TestMethod]

    public void TestCreditRiskForTeenagerWithLowIncome()

    {

        // Arrange

        int age = 14;

        decimal monthlyIncome = 700;

        // Act

        decimal creditRisk = CreditRiskCalculator.CalculateCreditRisk(age, monthlyIncome);

        // Assert

        Assert.AreEqual(0.8m, creditRisk);

    }

    [TestMethod]

    public void TestCreditRiskForElderWithMidIncome()

    {

        // Arrange

        int age = 85;

        decimal monthlyIncome = 1600;

        // Act

        decimal creditRisk = CreditRiskCalculator.CalculateCreditRisk(age, monthlyIncome);

        // Assert

        Assert.AreEqual(0.44m, creditRisk);

    }

    [TestMethod]

    public void TestCreditRiskForAdultWithHighIncome()

    {

        // Arrange

        int age = 30;

        decimal monthlyIncome = 3500;

        // Act

        decimal creditRisk = CreditRiskCalculator.CalculateCreditRisk(age, monthlyIncome);

        // Assert

        Assert.AreEqual(0.19m, creditRisk);

    }

    [TestMethod]

    public void TestCreditRiskForTeenagerWithNegativeIncome()

    {

        // Arrange

        int age = 20;

        decimal monthlyIncome = -50;

        // Act

        decimal creditRisk = CreditRiskCalculator.CalculateCreditRisk(age, monthlyIncome);

        // Assert

        Assert.AreEqual(-1m, creditRisk);

    }

}

1. We assume it is supposed to be “equal or above 150” in the specification, since we have a hint later on that says this Age Checker is unit-tested. [↑](#footnote-ref-1)
2. The site allowsyou to enter income with 1 decimal, but rounds it down, which is not the usual case and isn’t mentioned anywhere. [↑](#footnote-ref-2)
3. If we assume unit-tests for each checker have been done, border cases could be excluded. [↑](#footnote-ref-3)
4. Please see 3 notes regarding border 150 of Unit Testing in the Software World: Age Checker on page 3 [↑](#footnote-ref-4)
5. Depending of the context and the requirements of the calculator [↑](#footnote-ref-5)