Functional Area	**Test Name**	**Test Steps**	**Expected Results**	**Actual Results**
Set up of Micro:bit	Set up success	1. Connect Micro:bit using USB. 2. Press the START button to connect Micro:bit via web serial.	All buttons are now clickable except for START and INTERRUPT	*Developer:* expected *Target Audience*: expected *Customer:* expected
	Set up with no Micro:bit connected	1. Press the START button without connecting Micro:bit using USB.	A pop-up should tell you ‘No Paired Serial Devices Available’	*Developer:* expected *Customer:* expected
Unclickable buttons	With code running	1. Connect the Micro:bit using the START button. 2. Press RUN at the top of the screen. 3. Click on one of START, FLASH, RUN, or REBOOT. 4. Click on HELP. 5. Click on INTERRUPT.	After step 3, nothing should happen. After step 4, the duck should appear. After step 5, the code should be interrupted and all buttons except START and INTERRUPT are clickable.	*Developer:* expected *Target Audience*: expected *Customer:* expected
Flashing code to Micro:bit	With an error	1. Connect the Micro:bit using the START button. 2. Type code in the text editor that contains an error. 3. Press the FLASH button.	The code flashes to the connected Micro:bit, but it will only display the error. The helpful duck should appear to give advice on how to deal with the error.	*Developer:* expected *Target Audience*: expected *Customer:* expected but ‘Interestingly this doesn’t

				work on V1 micro:bit. No expectation that it should, given that you didn't have the hardware to test. Works fine on V2.'
	Without an error	1. Connect the Micro:bit using the START button. 2. Type code in the text editor that does not contain an error. 3. Press the FLASH button.	The code should run on the Micro:bit and work as expected.	*Developer:* expected *Target Audience*: expected *Customer:* expected
Editing code	While running on the Micro:bit	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the first 'While loop' example in the Python Language Features tutorial. 4. Press RUN at the top of the screen. 5. Try to edit the code in the editor.	The code should not be able to be edited while running on the Micro:bit	*Developer:* expected *Target Audience*: expected *Customer:* expected but 'I think further work could consider making this state clearer to the user or reconsidering the restriction.'
	Before running for the first time	1. Connect the Micro:bit using the START button.	The code should be able to be changed before running.	*Developer:* expected

		2. Attempt to change the code in the editor.		*Customer:* expected
	After running to completion	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the second 'While loop' example in the Python Languages tutorial. 4. Press RUN at the top of the screen. 5. After the code has been run to completion, try to edit the code in the editor.	The code should be able to be changed after being run to completion.	*Developer:* expected *Customer:* expected
	After running and being interrupted	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the first 'While loop' example in the Python Languages tutorial. 4. Press RUN at the top of the screen. 5. Press INTERRUPT 6. Try to edit the code in the editor.	The code should be able to be changed after running and being interrupted.	*Developer:* expected *Customer:* expected
	After running and terminating with a SyntaxError	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the second 'While	After step 5, you should see the duck and a SyntaxError message. At step 6, the code should be able to be changed.	*Developer:* expected *Customer:* 'This doesn't work for me. … I get a duck with an

		loop' example in the Python Languages tutorial. 4. Change the final line to `print(x` rather than `print(x)`. 5. Press RUN at the top of the screen. 6. Try to edit the code in the editor.		error but cannot edit. But as the test didn't specify that step I may have misunderstood.'
Autocomplete	After typing 'a'	1. Connect the Micro:bit using the START button. 2. Type 'a' into the editor 3. Press tab	After step 2, an autocomplete box should pop up containing `accelerometer` and `audio`. After step 3, `accelerometer` should appear in your editor.	*Developer:* expected *Customer:* expected
	After typing 'T'	1. Connect the Micro:bit using the START button. 2. Type 'T' into the editor 3. Press tab	After step 2, an autocomplete box should pop up containing `True`. After step 3, `True` should appear in your editor.	*Developer:* expected *Customer:* expected
The helpful duck appears	Call on duck without an error	1. Connect the Micro:bit using the START button. 2. Press the HELP button. 3. Press 'My code doesn't do what I want it to do'. 4. Run through a path.	The helpful duck should appear to give advice with whatever is needed.	*Developer:* expected *Target Audience*: expected *Customer:* expected and 'The duck is an excellent exploration of a structured approach to Python debugging and we're interested

				to see how we can carry it forward and test the ideas with students.’
	Call on duck with NameError, without following a tutorial	1. Connect the Micro:bit using the START button. 2. Change line 5 to say `Display.scroll('Hello, World!')` 3. Press the RUN button.	After step 3, the duck should appear with the error message: `Error on line 5: NameError: name 'Display' isn't defined`.	*Developer:* expected *Customer:* expected
	Call on duck with a SyntaxError, without following a tutorial	1. Connect the Micro:bit using the START button. 2. Change line 5 to say `display.scroll('Hello, World!'` 3. Press the RUN button.	After step 3, the duck should appear with the error message: `Error on line 5: SyntaxError: invalid syntax`.	*Developer:* expected *Customer:* expected
	Call on duck with an ImportError, while following a tutorial	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath a piece of code in the Python Language Features tutorial. 4. Edit the code to make it have an error by replacing the word `microbit` with `Microbit`. 5. Press the RUN button. 6. Navigate through the duck, first pressing ‘An error message is displayed’, telling it nothing is helping until it compares your error with the tutorial.	After step 3, the code should appear in the editor. After step 5, the duck should appear, the error should be displayed, and the line with the error should be highlighted. After step 6, the difference between your code and the tutorial code should be highlighted within the duck’s speech bubble.	*Developer:* expected *Target Audience*: expected *Customer:* expected

	Call on duck with a NameError, while following a tutorial	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the first 'While loop' example in the Python Language Features tutorial. 4. Edit the code to make it have an error by replacing the function `display` with `displa`. 5. Press the RUN button. 6. Navigate through the duck, first pressing 'An error message is displayed', telling it nothing is helping until it compares your error with the tutorial.	After step 3, the code should appear in the editor. After step 5, the duck should appear, the error should be displayed, and the line with the error should be highlighted. After step 6, the difference between your code and the tutorial code should be highlighted within the duck's speech bubble.	*Developer:* expected *Customer:* expected
	Call on duck with a TypeError, while following a tutorial	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the first 'While loop' example in the Python Language Features tutorial. 4. Edit the code to make it have an error by replacing the integer `500` with the string `'500'`. 5. Press the RUN button. 6. Navigate through the duck, telling it nothing is helping until it compares your error with the tutorial.	After step 3, the code should appear in the editor. After step 5, the duck should appear, the error should be displayed, and the line with the error should be highlighted. After step 6, the difference between your code and the tutorial code should be highlighted within the duck's speech bubble.	*Developer:* expected *Customer:* expected
	Make the duck disappear by	1. Press the HELP button. 2. Press the red 'X'.	After step 2, the duck should disappear.	*Developer:* expected

The helpful duck disappears	pressing the red 'X'			*Target Audience*: expected *Customer:* expected
	Make the duck disappear by pressing 'Goodbye' at the end of the flowchart	1. Press the HELP button. 2. Press 'My code doesn't do what I want it to do'. 3. Press 'A list has changed even though I didn't change it'. 4. Press 'Ah, that's it!' 5. Press 'Thanks Duck, bye for now!'	After step 5, the duck should disappear.	*Developer:* expected *Customer:* expected
	After making the duck disappear by pressing the red 'X' halfway through a path, make the duck reappear	1. Press the HELP button. 2. Press 'My code doesn't do what I want it to do'. 3. Press 'A list has changed even though I didn't change it'. 4. Press the red 'X'. 5. Press HELP again.	After step 5, the duck should start back from the beginning of the flowchart. Rather than the slide it was closed from.	*Developer:* expected *Target Audience*: expected *Customer:* expected
Testing code from tutorial	Running code directly from tutorial	1. Connect the Micro:bit using the START button. 2. Press a RUN EXAMPLE from within the tutorial.	After step 2, the example should run on the Micro:bit and behave as expected. The code isn't expected to stay on the Micro:bit when disconnected from the computer.	*Developer:* expected *Target Audience*: expected *Customer:* expected

	Inserting a fragment, then flashing	1. Connect the Micro:bit using the START button. 2. Clear the editor of any code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath a piece of code in the Python Language Features tutorial. 4. Press the FLASH button. 5. Disconnect the Micro:bit.	After step 3, the code fragment should appear in the editor. After step 4, the code should be flashed to the Micro:bit, behave as expected. After step 5, the program should continue to run on the Micro:bit once reconnected.	*Developer:* expected *Customer:* expected

The customer also provided us with some UX suggestions from working through the test cases:

- 'I think it would be natural to allow triggering the duck debugging flow from the yellow highlighted line (if it has already been closed). You can reopen from help, but it feels natural to try to interact with the remaining area of the UI associated with the error.
- Switching tutorial is confusing when the duck is displayed as it seems to have no effect as the updated UI is hidden behind the duck. It would be worth considering closing the duck in that case.'

They further recommended that some possible extensions might include:

- 'A shift in focus to use the side-by-side notebook approach for educators' continuing professional development. The error message tutorial is especially promising for this.
- Practical testing of the interactive debugging approach offered by the duck with students and refinement based on feedback and more real-world evidence of the errors students encounter. Is there too much text to expect students to work through?
- What should the trade-off be between encouraging students to take a systematic approach to debugging their problems (as showcased by the current implementation) and the system giving smart advice, for example by analysing student error messages and code.
- Making further use of WebSerial. Current micro:bit editors generally use WebUSB but its support for serial interaction is limited. This project has been a very helpful testcase for WebSerial.'