PROJECT DELIVERABLE 3 STEP2

Team 4 (Sky Blue)

CSCD01

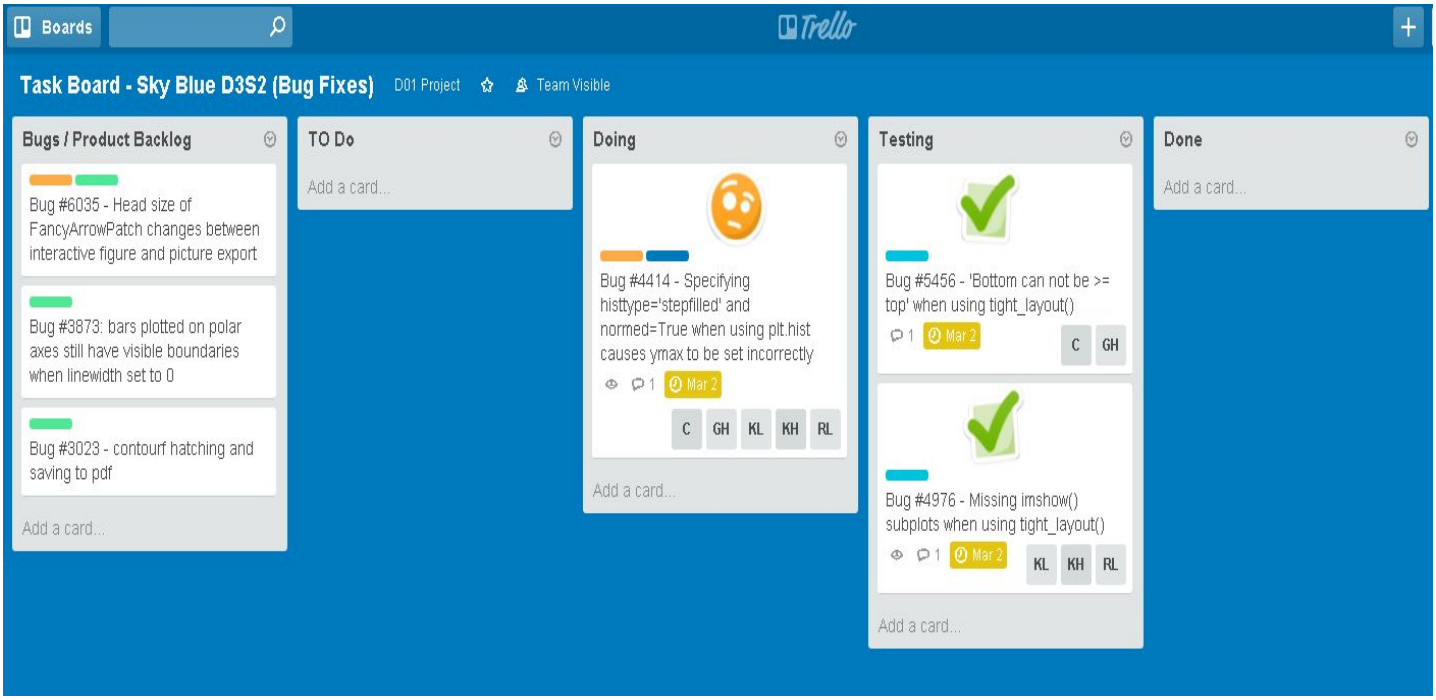
March 1, 2016

Team Members:
Chun Cho
Richard Luo
Yuxuan Hu
Geoffrey Hong
Kai Lin

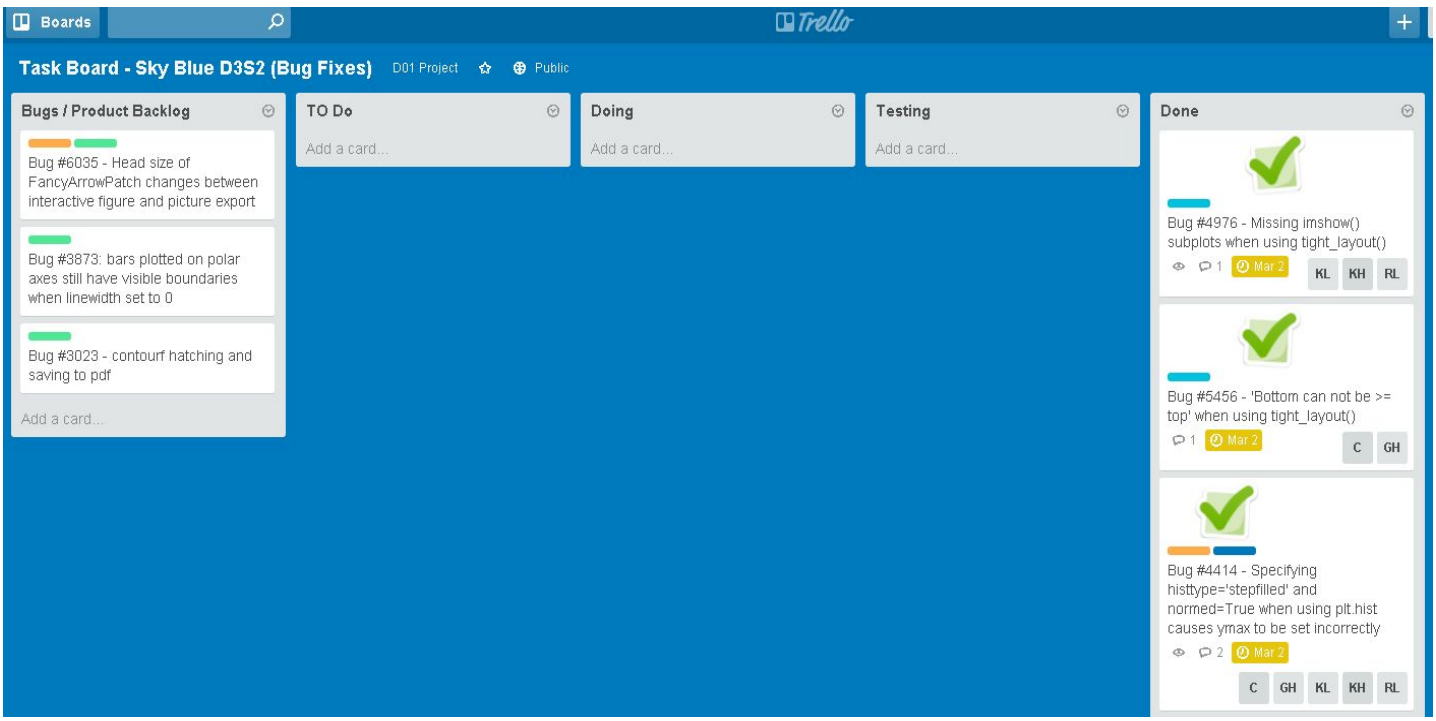
Table of Contents

Project Management

We are using trello for our Project Management.
(https://trello.com/b/dh89g9D0/task-board-sky-blue-d3s2-bug-fixes)
Here is our project task board for our bugs fix screen capped last week.
Member label: **C** : Chun Cho; **GH** : Geoffrey Hong; **KL** : Kai Lin; **RL** : Richard Luo;
KH : Kevin Hu



As of March 1st, 2016, the team has fixed and tested 3 bugs: #4414, #5456, #4976.

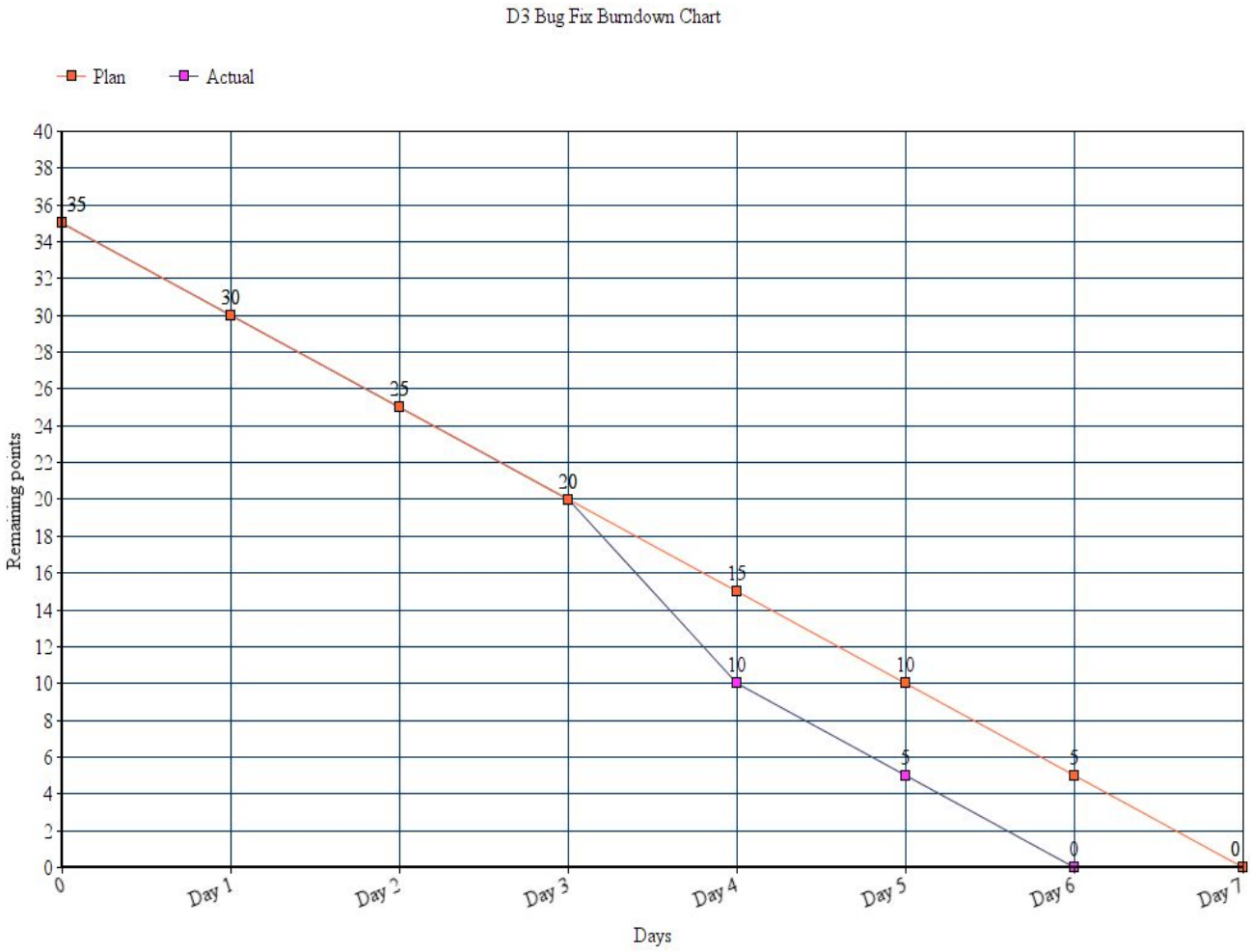


The project burndown chart shows like this. We accomplish fixing three bugs rather than two.

Unit of work : 1 point = 1 day per person, approximately 3 hours.

The plan for the team was to work on Issue #1 (Kevin, Richard, Kai) and Issue #2 (Geoffrey, Chun). Due to tight schedule and other commits during the week of February 22 to March 1, the team planned to spend seven days working on the issues. Members are expected to put in 1 units of work a day, totalling to 35 units of work for the week as a team.

Investigation was planned for the first three days of the week, leaving the other four days for debugging and testing. Fortunately, the team was able to fix Issue #5654 and Issue #4976 by the fourth date, hence leaving time to attempt to fix another issue (Issue #4414).



Instruction for User Acceptance Testing:

The test files for each issue is stored in a folder with the corresponding name, in the same directory as this document. To run each test case:

- Go to the corresponding directory(eg. Issue1_bug#4976_acceptance_test)
- Enter into terminal the corresponding "> python xxx.py" line mentioned in each test case below.

Bug #4976 - Missing imshow() subplots when using tight_layout()

Technical Commentary

Relevant matplotlib source code files:

- matplotlib/lib/matplotlib/tight_layout.py (*modified, line: 199*)

Attempt 1:

In tight_layout, change the following function so that it return positive value

    def _get_left(tight_bbox, axes_bbox):
    def _get_right(tight_bbox, axes_bbox):
    def _get_bottom(tight_bbox, axes_bbox):
    def _get_top(tight_bbox, axes_bbox):

In tight_layout the functions above return the adjustment value of each subplot. But when minimum value of tight_bbox or axes_bbox subtract the maximum value of the other, it might gave out a negative float which cause the bug to occur.
The interaction between subplot and tight_layout will then remove the axes that have overlapped with each other. The main reason this happens is because the above function returns negative float instead of positive.

Our solution is to change the return value to positive always. However, there are some problem with this solution, it might cause other method that use tight_layout to behave differently. Thus, this solution is not so good.

Attempt 2:

This time, instead of changing all the function, we decide to change the line

    kwargs["wspace"] = hspace / h_axes
    to
    kwargs["wspace"] = **abs**(hspace / h_axes)

so that the adjusted figure will always fits in its axes region without overlapping axes in left, right, top, or bottom coordinate.
With this attempt, it will produce the same image if you move tight_layout() to outer loop.

User Acceptance Test

Test case #1 : Test original issue with tight_layout() inside the loop
Script :
> python test_4x4_2.py
Check image “imshow_4x4_2.png”
Expected Result: The figure produce all 16 image without any image missing
Purpose: Make sure the original issue is solved.

Test case #2 : Test original issue with tight_layout() outside the loop
Script:
>python test_4x4_1.py
Check image “imshow_4x4_1.png”
Expected Result : The figure produce all 16 images without any image missing
Purpose : Make sure the fix does not break whatever was working before.
Note : This scenario works before the fix.

Test case #3 : Test on same rows and cols where tight_layout() should be inside of the loop
Script:
>python test_5x5.py
Check image "imshow_5x5.png"
Expected result : The figure produce all 25 images in "imshow_5x5.png"
Purpose : The fix should work or other different dimension

Test case #4 : Test on different dimension of output where tight_layout() should be inside of the loop
Script:
>python test_3x5.py
Check image "imshow_3x5.png"
Expected result : The figure produce all 15 images in "imshow_3x5.png"
Purpose: The fix should work on other dimension

Regression test on tight_layout() using the existing matplotlib test

Note: To run this test, tester must install mock and nose. After that, they need to rename setup.cfg.template to setup.cfg and change the locak_freetype=True.
> python tests.py matplotlib.tests.test_tightlayout

- Ran 33 tests. OK

Bug #5456 - 'bottom cannot be >= top' when using `tight_layout`

Technical Commentary

Relevant matplotlib source code files:

- matplotlib/lib/matplotlib/tight_layout.py (*modified, line: 170-200*)

Overview of bug

The bug appears when the user sets a y-axis label that exceeds the size of the figure and then calls tight_layout() which causes a ValueError: bottom cannot be >= top and crashes the python program.

In this scenario, bottom and top are the margins(white space) that are set between the figure and the top/bottom of the image. If the y-axis label is set to a long string, it causes the calculations of these margins in tight_layout() to be greater than their max length 0.5, which then raises the ValueError as mentioned before. The size of the figure should be in the range 0.0-1.0 for both x and y lengths. If one of the margins exceed 0.5, then the image produced would have too much empty space on one half. If margins top and bottom exceed 0.5, then the figure will be left with no space to be displayed, which causes the program to crash.

We implemented a fix that uses checks the calculations to control the max size of the margins so that the ValueError will never occur if the y-labels are greater than the size of the figure. Specifically we have added several checks to ensure that after calculating the margins, if they exceed max length, they will default into a safe margin size.

Through investigation, it was discovered that a similar bug happens when the label assigned to the x-axis exceeds the allotted space for the figure. Namely, the error that occurs is ValueError: left cannot be >= right and crashes the python program. We have implemented the same fix to apply for the x-axis as well.

There were several alternative methods to fix this bug such as:

- Clipping the axis's label so that it does not exceed the size of the figure
    - Drawback: Due to the nature of how text is displayed, it is used by many other matplotlib areas. Changing how text interacts with the figure would require extensive regression testing to ensure existing functionalities are not broken.
- Scaling the figure or the label to match each other
    - Drawback: Would need to understand how figures and labels are drawn and scaled, which could be time consuming. If the graph is scale too large,

processing and saving the graph would be expensive as the file associated would be large.

The chosen method mentioned previously was used because it involves minimal amount of risk. The fix could affect anything that uses tight_layout() but the functionality of tight_layout() will remain the same as it was before. This is because the fix specifically targets Figures that uses tight_layout() with axis label that exceed the allotted space. The modifications made to the code does not affect the overall design of matplotlib, nor the architecture.

User Acceptance Test
Test case #1: Test original issue
>python test_long_ylabel.py
Expected Result: There should be image "long_ylabel.png" and the original bug should be fixed
Purpose: Make sure the original issue is solved

Test case #2: Test original issue with regular ylabel
>python test_regular_ylabel.py
Expected Result: There should be image “regular_ylabel.png” and make sure script runs without any bug
Purpose: Make sure the fix does not break whatever was working before

Test case #3: Test with regular x label to ensure the fix does not break anything
>python test_regular_xlabel.py
Expected Result: check image “regular_xlabel.png” and make sure the script runs without any bug
Purpose: make sure the function still works as intended when the fix is apply to it

Test case #4: test with long x label to ensure that the fix does not break anything
>python test_long_xlabel.py
Expected Result: check image ‘long_xlabel.png” and ensure the script still runs
Purpose: Make sure the fix does not break anything

Regression test on tight_layout() using the existing matplotlib test
Note: To run this test, tester must install mock and nose. After that, they need to rename setup.cfg.template to setup.cfg and change the locak_freetype=True.
> python tests.py matplotlib.tests.test_tightlayout

- Ran 33 tests. OK

Bug #4414 - Specifying histtype='stepfilled' and normed=True when using plt.hist causes ymax to be set incorrect

Technical Commentary

Relevant matplotlib source code files:

- matplotlib/lib/matplotlib/axes/_axes.py (*modified, line: 6240-6254*)

Overview of bug

The bug appears when the user tries to create a histogram by calling matplotlib.pyplot.hist(...) with histtype='stepfilled' and normed=True properties. This causes the y-axis to be incorrect in the image produced.

Based on the previous class definition of hist(), if only histtype='stepfilled' is passed, then normed will default to false and together it causes the histogram to calculate its y-axis limits based on the graphing data. If only normed=True is passed, then histtype will default to 'bar' and together it causes the histogram to scale it's y-axis limits after normalizing graph data. However, when both are provided, the y-axis limits are calculated instead of being scaled, because the previous code does not check if normed=True, which should override the manual calculations of y-axis limits.

The bugfix code involves an if statement in the matplotlib/lib/matplotlib/axes/_axes.py hist(...) method that checks the histtype and normed parameters to sets the y limit accordingly. As mentioned above, if the histtype property were not given, then the algorithm would simply scale the graph as if it were a bargraph. However, this is not the case with histtype = "stepfilled". Since histtype = "stepfilled" is given, it will not scale according to the data, which is why it is imperative to determine the method of scaling before any calculations. The best way to do this is by using an if statement when histtype='stepfilled' to check the value of the normed property, and scale accordingly(i.e. Manually calculate dimensions if data not normalized normed=False vs. Auto scale if data is normalized normed=True).

There were no better alternative methods that could fix the bug as simple as the solution provided. This is because the bug is caused by a very specific missing functionality in the hist() function, that is to check if both parameters histtype and normed are set and how to deal when they are. The fix implemented affects histogram figures created with the hist() function, with parameters histtype='stepfilled' and normed=True. Any other combination of parameters will cause hist() to perform as it has before. The modifications made to the code does not affect the overall design of matplotlib, nor the architecture.

User Acceptance Test

To test these test cases, the tester must install ***seaborn*** through their package manager with either “pip install seaborn” or “conda install seaborn” depending on which one the tester uses.

Test case #1: Test original issue: plt.hist() has set normed=True, histtype="stepfilled"
Script:
>python test_stepfilled_normed.py
Expected Result: There should be image "stepfilled_normed.png" in the same directory and the y-axis last number should be 3. No data should go off screen.
Purpose: Make sure the original issue is solved: y-axis displays the correct numbers when normed=True, histtype="stepfilled"

Test case #2: Test previous working case: plt.hist() has set normed=True
>python test_norm.py
Expected Result: There should be image "normed.png" in the same directory and the y-axis last number should be 3. No data should go off screen.
Purpose: Make sure the fix does not break whatever was working before

Test case #3: Test previous working case: plt.hist() has set histtype="stepfilled"
>python test_stepfilled.py
Expected Result: There should be image "stepfilled.png" in the same directory and the y-axis last number should be 12. No data should go off screen.
Purpose: Make sure the fix does not break whatever was working before