XPP and User-Interface Design

Applications of Universal Principles of Design in Unmaintained Software

Matthew Ursaki
Computer Science
University of Lethbridge
Lethbridge, Alberta, Canada
ursaki@uleth.ca

1. Introduction

XPPAUT(X-Windows Phase Plane plus Auto) or XPP, is a software tool developed by Bard Ermentrout, an American Mathematician and distinguished professor at the University of Pittsburgh. XPP is a visualization and simulation tool for dynamical systems, and is most notably capable of handling up to 590 differential equations. The software's conception dates back to the mid 1990's, and has seen a number of iterations across multiple platforms. XPP's latest version (8.0) was released January 2016. Existing primarily as a hobby of its creator, the project has not received external funding in its lifetime. Any interfaces once associated with XPP (Rob Clewley's Matlab interface, Jakub Nowacki's Python interface) are no longer maintained or even available. For these reasons, the software suffers a dated user-interface, as well as partial platform dependency. Fortunately, the software and its installation processes are thoroughly documented on the program's website. Additionally, a guidebook, authored by Bard Ermentrout, is available for purchase.

Being a unique and particularly useful teaching, learning, and resource tool, XPP could benefit from a graphical and architectural revision. In its current state, the installation process poses an unnecessarily high barrier to entry, and the software itself is awkward to use.

The main challenge of porting or updating XPP to a new framework is XPP's 'model' and 'view' being very tightly coupled, with nearly no semblance of a 'controller'. X Window System, or X11, is the windowing system responsible for XPP's displays, heavily integrated into the program's functionality. Thus, an architectural rework of XPP would require a fairly complete understanding of the program's functionality.

Without a change in the program's architecture, reworking its interface is unfeasible. As a step towards the task of an architectural rework, this paper will analyze the design principles that XPP's graphical user interface violates and present possible solutions.

2. Background / Related Work

As knowledge and information have become more easily accessible, design and designers have become increasingly specialized. The authors of *Universal Principles of Design*[1] identify three challenges posed by cross-disciplinary design: determining which sources for each discipline are most useful, becoming familiar with terminology related to each discipline, and determining how deep research into each discipline is necessary. *Universal Principles of Design* aims to remedy these challenges by offering laws, guidelines, and considerations that cut across disciplines.

XPP exists at the intersection of Mathematics and Software Engineering, with its creator evidently having a deep understanding of the former, and a less sophisticated understanding of the latter. It is unsurprising that Dr. Ermentrout had not fully considered certain aspects of design when creating XPP.

Universal Principles of Design organizes its contents into five categories, each posed as a question: 1. How can I influence the way a design is perceived? 2. How can I help people learn from a design? 3. How can I enhance the usability of a design? 4. How can I increase the appeal of a design? 5. How can I make better design decisions?

The principles found within each category are not mutually exclusive, and share considerable overlap. For this reason, the majority of my research has been within the categories I deemed most relevant to the improvement of XPP: enhancing usability, helping the user learn from design, and making better design decisions. Rather than addressing each XPP's design flaws by category, however, I will address concerns by principle.

CCS CONCEPTS

• Software Design • User Interfaces • GUI Frameworks
• Legacy Code

3. Design Principle Violations/Considerations

3.1 Aesthetic-Usability Effect (20)

"Aesthetic designs are perceived as easier to user than less-aesthetic designs."

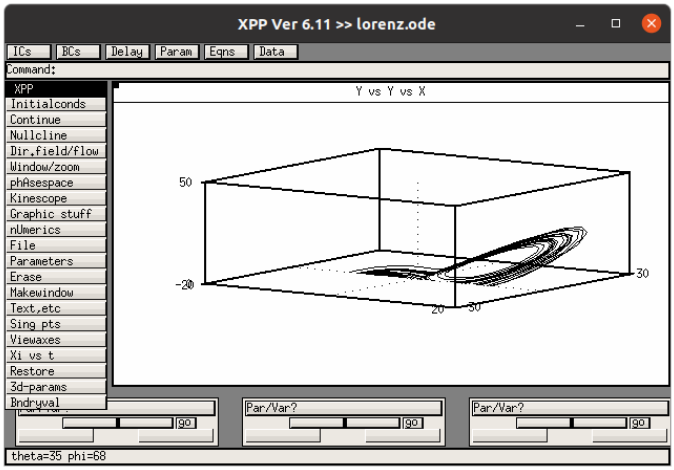
From the opening dialog and throughout XPPS functionality, XPP *appears* old. Though its unexeciting font and grayscale palette have no direct influence on the program's ability to compute and render dynamical systems, the way the program *looks* may foster a negative attitude in some users. Negative perception inevitably stifles the users patience, and lowers their tolerance of the other design problems found within the program.

To XPPs credit, current grayscale palette creates a sense of depth in its buttons, displays, and background, without drawing the users attention elsewhere (Color, 48). The palette also contributes to the program's usability by as many people as possible (Accessibility, 16), considering the possibility that a user may be color blind.

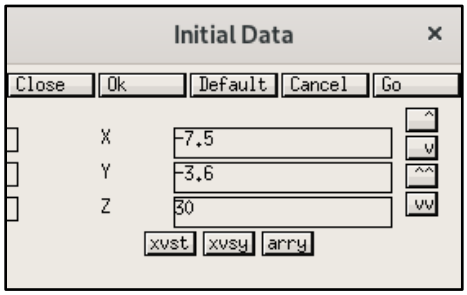
3.2 Consistency (56)

"The usability of a system is improved when similar parts are expressed in similar ways."

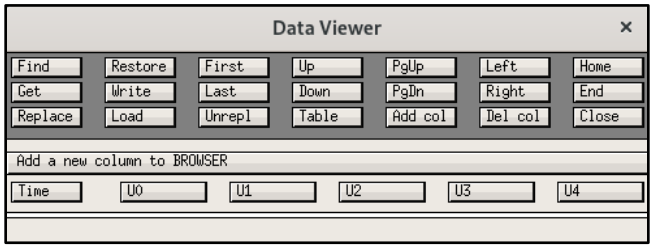
Consistency is particularly important in user interface design because it enables users to efficiently and quickly accomplish relevant tasks, rather than navigating other options or functions. Though XPP is mostly aesthetically and functionally consistent, it suffers from internal consistency; elements of XPPs toolbars and windows are inconsistent. For example: buttons on XPP's horizontal toolbar each produce new windows with additional options.



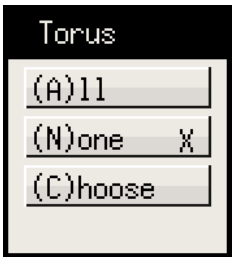
The 'ICs' button produces this window:



And the 'Data' button produces this window:



The 'Close' buttons are found on opposite sides of the two windows. Fortunately, the 'Close' buttons are consistently placed among the other windows produced by the buttons on horizontal toolbar. However, this minor inconsistency is particularly exaggerated in the Data tool by it's sheer amount of options compared to the other tools, confusing the user and slowing down their relevant task. This inconsistency also contributes to another inconsistency cutting across the horizontal and vertical tool bar.



This window is produced by the 'phAsespace' button on the vertical taskbar. Notice the lack of 'Close' option. Instead, to exit this window, the user must press the escape key. This process implies and expects a level of proficiency from the user that is inconsistent with the horizontal toolbar (Control, 64). Furthermore, if the option to press the escape key to close the window exists within the vertical toolbar, it necessarily should exist within the horizontal toolbar.

Additionally, not all of the horizontal toolbar's buttons produce separate windows. Some buttons prompt input on the 'command' prompt above the vertical taskbar, which may not be immediately apparent to the user.

Finally, in one instance, closing a window requires pressing an ambiguously labeled 'Kill' button (File → prt scr → Kill).

3.3 Mapping (152)

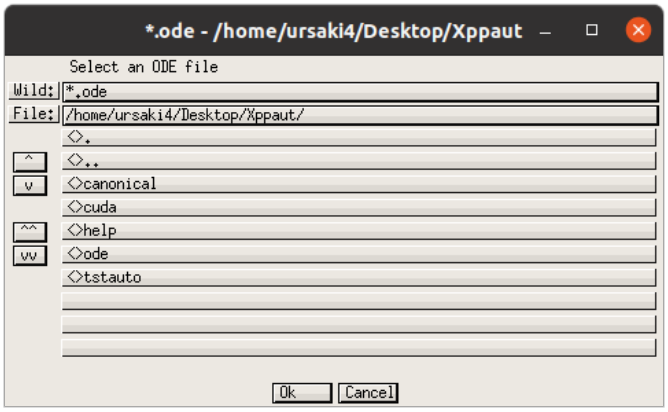
"A relationship between controls and their effects. Good mapping between controls and their effects results in greater ease of use"

Mapping refers to the effect caused by a certain control. A common mapping issue is when the effect of a control has an unexpected outcome.

An example of good mapping in software is the mouse's scroll wheel's effect during file browsing. When you use your finger to push up on the scroll wheel, the file browser scrolls 'up' on the screen. When you pull down on the scroll wheel with your finger, the file browser scrolls 'down'. This behaviour is generally expected in file browsing, and is consistent with most people's mental model (154) of casual computing.

Proper scroll wheel mapping, as well as a general desktop users mental model are violated by XPP's opening file browser dialog. The scroll wheel has no effect on viewing which files are accessible within a directory. Furthermore, if the scroll wheel is used in either direction while hovering the cursor over a file, that file is selected for rendering.

While the minimize and maximize buttons work for all of XPPs various windows, none of the close buttons ('x' in the top right corner) work. That is, the close buttons have no effect at all. As discussed in the previous heading, the user must use the inconsistently placed 'close' buttons, or use a keyboard shortcut.



3.4 Chunking (40)

"A technique of combining many units of information into a limited number of units or chunks"

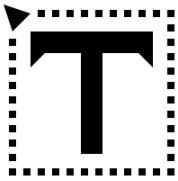
Though efforts have been made to separate XPP's various functions (separate horizontal and vertical toolbars), its chunking is ultimately incomplete.

For example, the 'Continue', 'Erase' and 'Restore' buttons share relevant functionality, yet are arbitrarily place among the other tools on the vertical tool bar. The 'File' button also finds itself out of place on the toolbar, having a purpose more encompassing of the entire program, rather than the other tools that have specific functions regarding the rendering and visualization of .ode files.

3.5 Iconic Representation (132)

"The use of pioctorial images to improve the recognition and recall of signs and controls."

Though XPP engages in some pseudo-iconic representation in its scroll buttons, an inherently visual program would benefit from an iconic aspect to its toolbars. Buttons like 'Kinescope' could be represented by a vector image trace of an actual kinescope, and the 'Text' button could be represented using the familiar "T" vector image.



3.6 Design by Committee (74)

"A design process based on consensus building, group decision making, and extensive iteration."

Though XPP's requirements a relatively straightforward, the consequence of its errors are tolerable, and it has no genuine 'stakeholders', XPP would greatly benefit from a committee of developers for future iterations. Unless future iterations of the program are developed under a strict timeline, the benefit of a diverse design committee would mitigate many of both the design and architectural problems XPP faces today. Leveraging diverse expertise to identify potential flaws and implement best practices will result in a more robust software product. Though this processes is generally slower than 'design by dictator', an agile governance approach will ensure the project's progress.

4. Discussion / Solution

Considering the solutions to the problems observed are essentially implicit in their identification, I will not address by point each design violation. Instead, I will present basic prototypes created using QT Designer which endeavours to address the violations/considerations.

4.1 Main Window

This prototype demonstrates a more up-to-date appearance that will provide the user with a greater sense of familiarity and confidence. Utilizing the QT framework, this prototype rids XPP of its antiquated display provided previously by X11. QT provides CSS stylesheet support. This makes colouring, font selection, and other various visual elements of the program easily accessible.

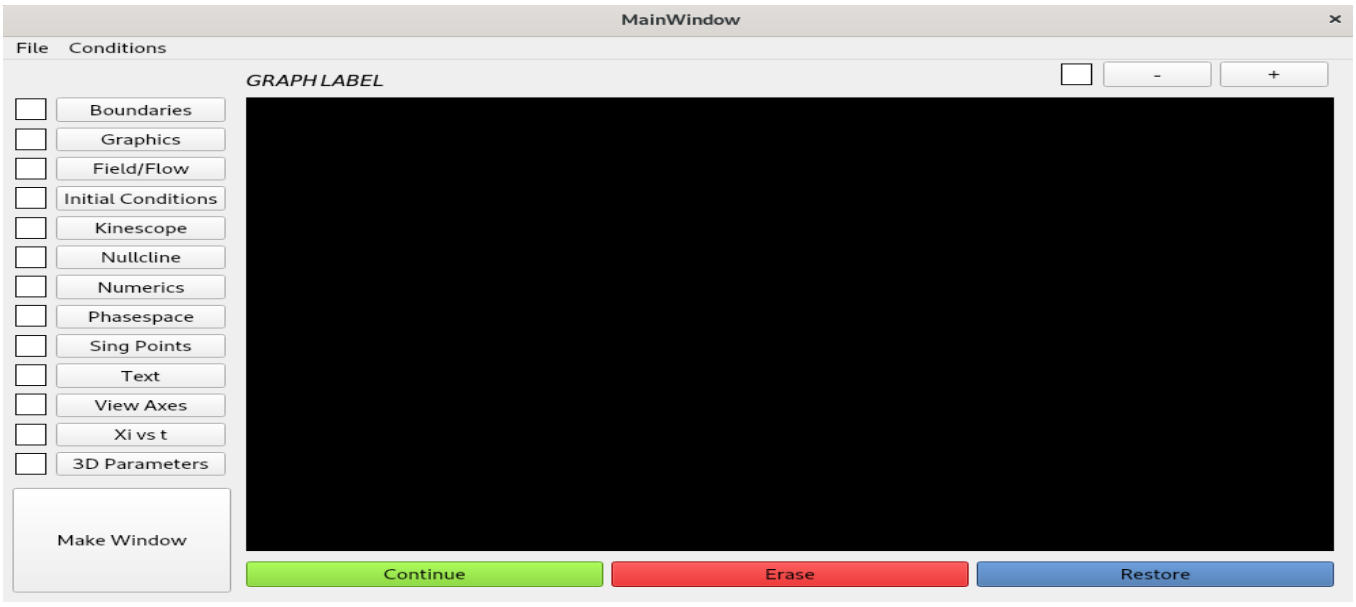
This prototype is not strictly visual. Each button represents an object in the QT project's code. A template for submenus corresponding to each button could be devised in QT to ensure consistency tools.

QT provides developers options on how various aspects of their program interacts with mouse clicks, mouse hovering, keyboard shortcuts, etc., making prior mapping issues easier to address. Future iterations will ideally allow the user increased control over the various submenus found in XPP, supporting keyboard shortcuts, as well as clickable close buttons.

This prototype offers adequate chunking of XPPs functionality. Replacing the horizontal toolbar previously found at the top the window, there is a menu bar. The 'file' button once available on the vertical toolbar is found there. The 'continue', 'erase', and 'restore' functions have been taken from the vertical toolbar and colour-coded beneath the .ode display (which is represented by an empty openGL widget). The 'make window' button has been given its own section and distinct size on the buttom left of the window, as its function is inherently different and separate from the manipulation of the current .ode file. The 'zoom' button is now represented by a plus and minus symbol above the openGL widget.

Some of the buttons have what appears to be white checkboxes beside them: these are placeholders for vector images for skeumorphic/iconic representations of their respective tools. A possible upgrade for this toolbar with be making the the textual representations of the tools collapsible, displaying only the visual representations of each tool.

Though the timeframe for this prototype was relatively short, a delegation to create both horizontal and vertical prototypes was made. Under the direction of the faculty supervisor, I was assigned to create this horizontal prototype, and my project and research partner Kevin Masson was assigned to integrate a vertical prototype into it. This use of design by committee ensured the prototype was developed efficiently, effectively, and timely, with each member contributing their particular area of research to the project.



4.1 Opening Dialog

Rather than forcing the user to use a file browser created using the GUI Framework, future iterations of XPP should use either the file browser native to the operating system, or a simple drag and drop interface.

'Less is more' when engaging with a task as simple as uploading an .ode file. Thus, any future iteration of XPP's opening dialog should be simple, concise, and effortless in terms of user experience.

Conclusion

In spite of it's dated interface and lack of proper engineering, XPP's unique capabilities make the program a viable candidate for an upgrade.

The design principles identified and discussed here provide a basic guide for future development, but should not be considered in anyway comprehensive. Many more principles outlined in *Universal Principles of Design* alone have application in this instance of software development.

Though it is possible to rework this software independently, future development teams should consider building a closer and more collaborative relationship with XPP's creator. If this cannot be realistically achieved, future development teams may need a greater breadth of knowledge, particularly regarding mathematics and dynamical systems.

While there exists other candidates for upgrading XPP's graphical framework, QT seems to be the most favourable, providing thorough documentation, a comprehensive set of widgets, and cross-platform capabilities.

References

[1] Lidwell, William, et al. *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach Through Design*. Rockport Publishers, 2010.