CSE 1325: Object-Oriented Programming

Lecture 09 – Chapter 12
(Using gtkmm)

Introduction to Graphical User Interface (GUI) + Agile Processes

Mr. George F. Rice

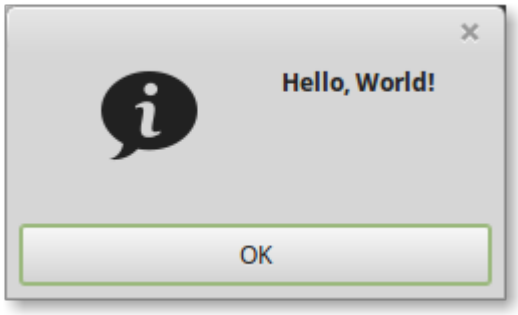
george.rice@uta.edu

Based on material by Bjarne Stroustrup
www.stroustrup.com/Programming

ERB 402
Office Hours:
Tuesday Thursday 11 - 12
Or by appointment

Overview: Intro to GUI and Agile Processes

- Graphical User Interfaces (GUI)
  - History of User Interfaces
  - Principle of Least Astonishment
  - Why GUI?
  - Common Widgets
  - Selecting a GUI Library
- The Façade Pattern
- Agile Processes and Scrum
  - Agile vs Waterfall
  - Four Artifacts
  - Super-Simplified Scrum



A Brief History of User Interfaces

- Early user interfaces were based on paper tape...
  - Paper tape (with 5, 6, 7, or 24 holes per row) date back to 1725 for automating weaving looms
  - First used in 1846 to prepare telegrams for transmission
  - Punch machine separate from reader
  - Excellent confetti generator!

Photo by Bad Germ CC-BY-SA-3.0 Unported

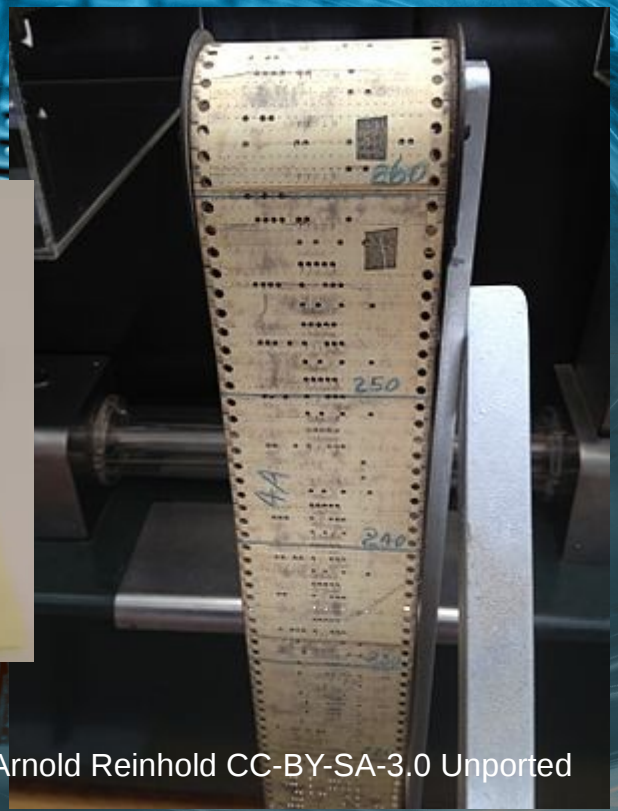
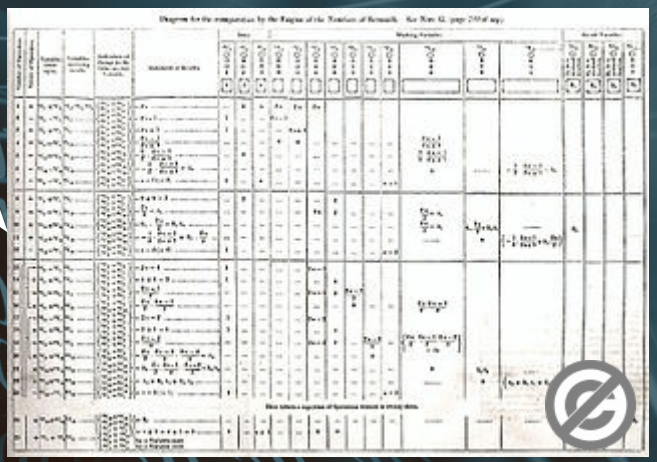


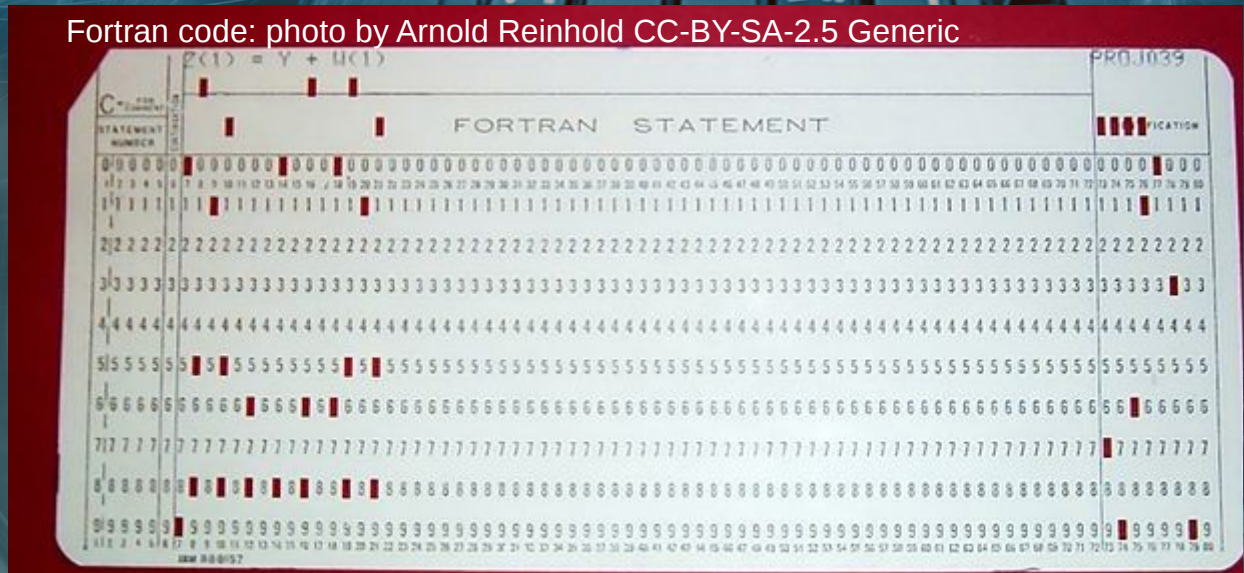
Photo by Arnold Reinhold CC-BY-SA-3.0 Unported

A Brief History of User Interfaces

- … and punch cards aka Hollerith cards
  - Cards with 80 columns, often driving looms and musical instruments, starting in 1832
  - Charles Babbage proposed use of cards for the Circulating Engine Store in the Analytic Engine
    - Lady Ada Lovelace's algorithm to produce Bernoulli numbers was the first computer program... designed for cards
  - Cards were used to calculate the 1890 census



Fortran code: photo by Arnold Reinhold CC-BY-SA-2.5 Generic



A Brief History of User Interfaces

- Digital computers supported physical switches and lights or Nixie tubes on the front panel
  - To boot the computer, just load the program via switches
  - Debug by watching light patterns

PDP-8 photo by Florian Schäffer CC-BY-SA-4.0

Georg-Johann Lay Gnu FDL 1.2+

PDP = Programmable Data Processor

A Brief History of User Interfaces

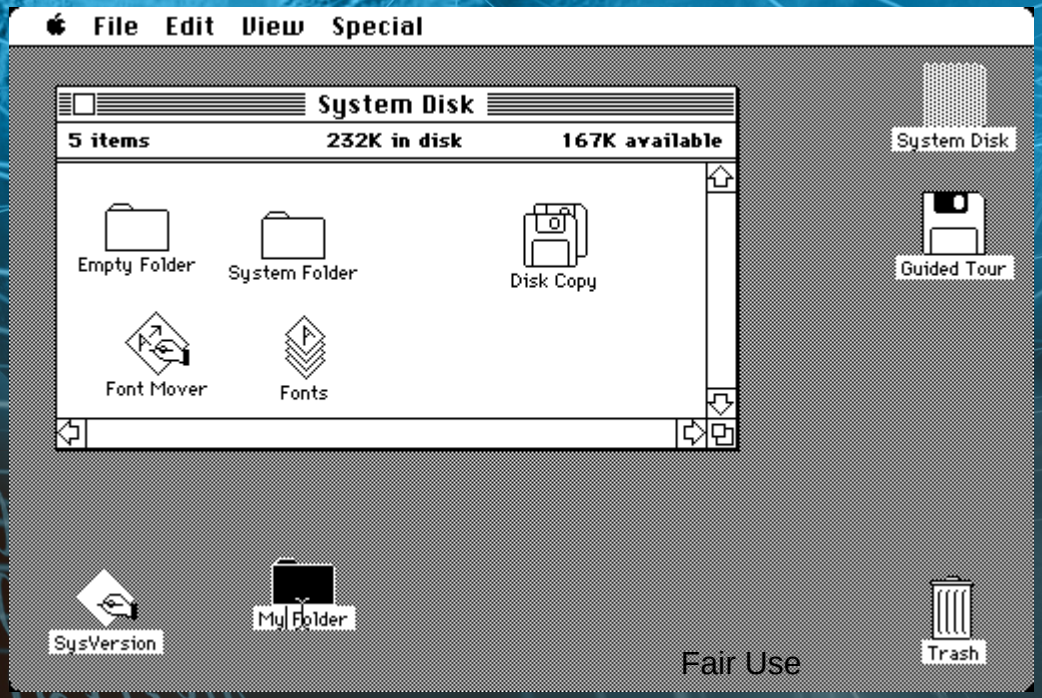
- Keyboards and thermal printers or displays brought the Command Line Interface (CLI)

Dave Fischer CC-BY-SA-3.0

Jason Scott CC-BY-SA-2.0 Generic

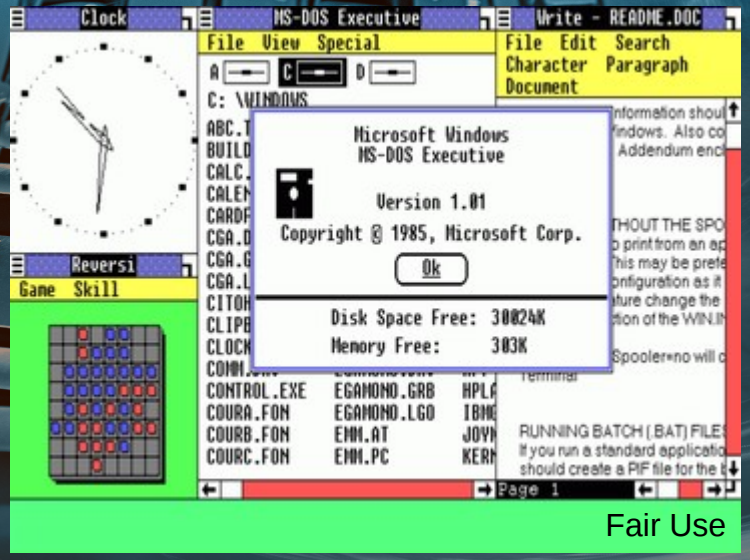
A Brief History of User Interfaces

- In 1984 began the rise of the Graphical User Interface
  (Windows Icon Mouse Pointer, or WIMP)



Apple Macintosh OS 1.0 Desktop

Xerox Star Compound Document



Microsoft Windows 1.01 →

A Brief History of User Interfaces

- GUIs matured over time into “push button software” with rich mouse interaction

Fair Use

Apple Macintosh OS X 10.11 Desktop

Максим Пе CC-BY-SA 4.0 Int

Ubuntu Linux 14.04 Desktop

Microsoft Windows 10 →

Fair Use

A Brief History of User Interfaces

- In 1996, Phillips introduced Voice Dial™, an early Voice User Interface (VUI)

- In 2011, Apple announced the Siri general VUI

VUIs have generally acted as a complement to other user interfaces rather than as a primary interface

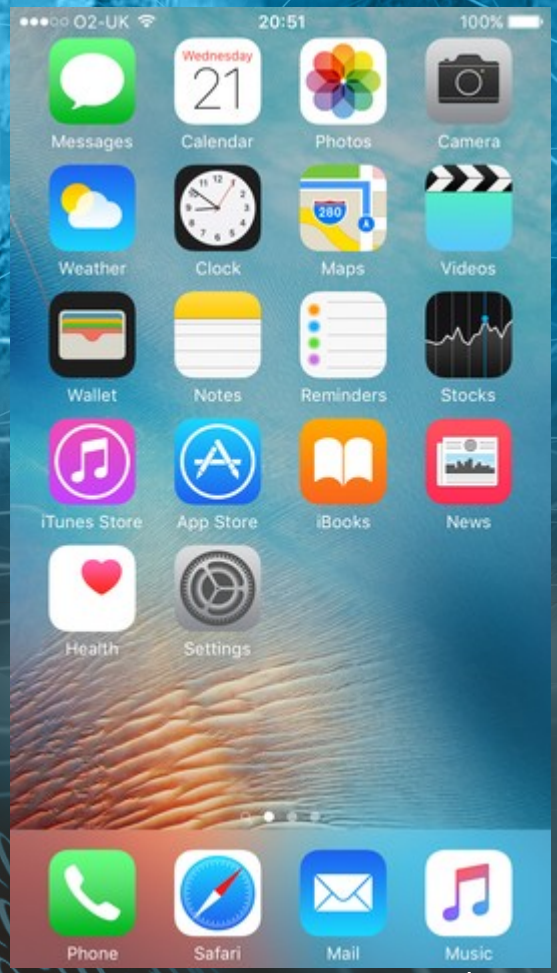
A Brief History of User Interfaces

- In 2006, the Touch Interface...
  (also called the Gesture Interface)

Fair Use

A Brief History of User Interfaces

- ...which has also evolved



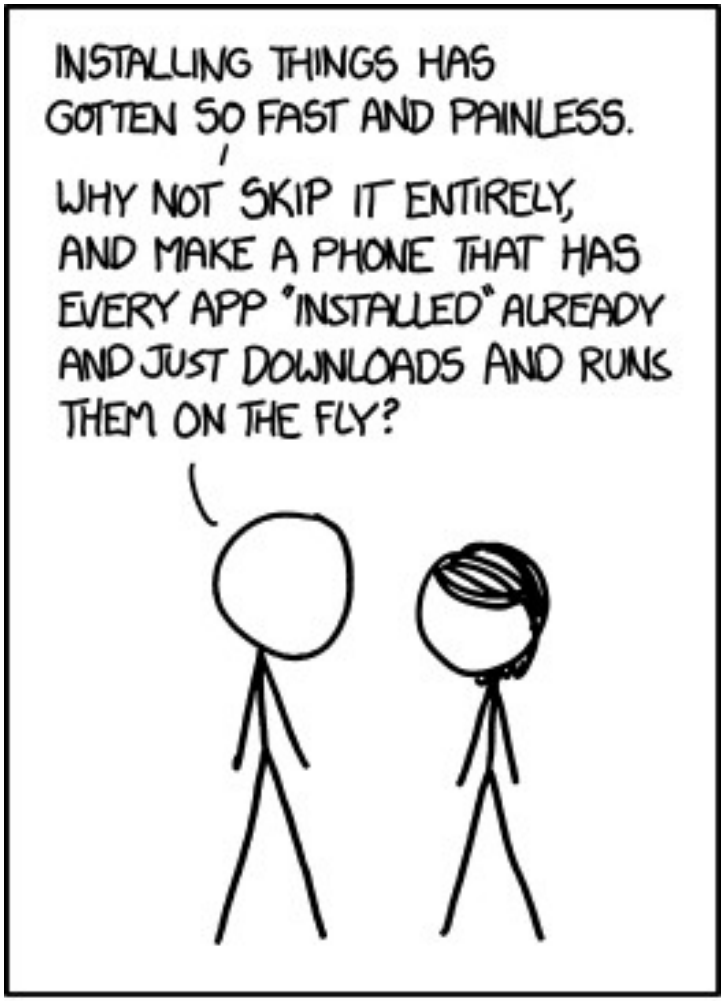
Fair Use

Apple iOS 9 Home Screen

Apache AOSP

Android 7 Nougat Home Screen

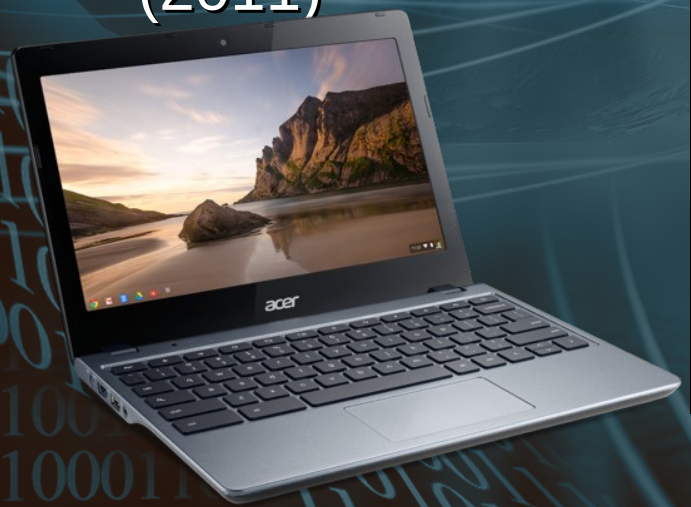
A Brief History of User Interfaces

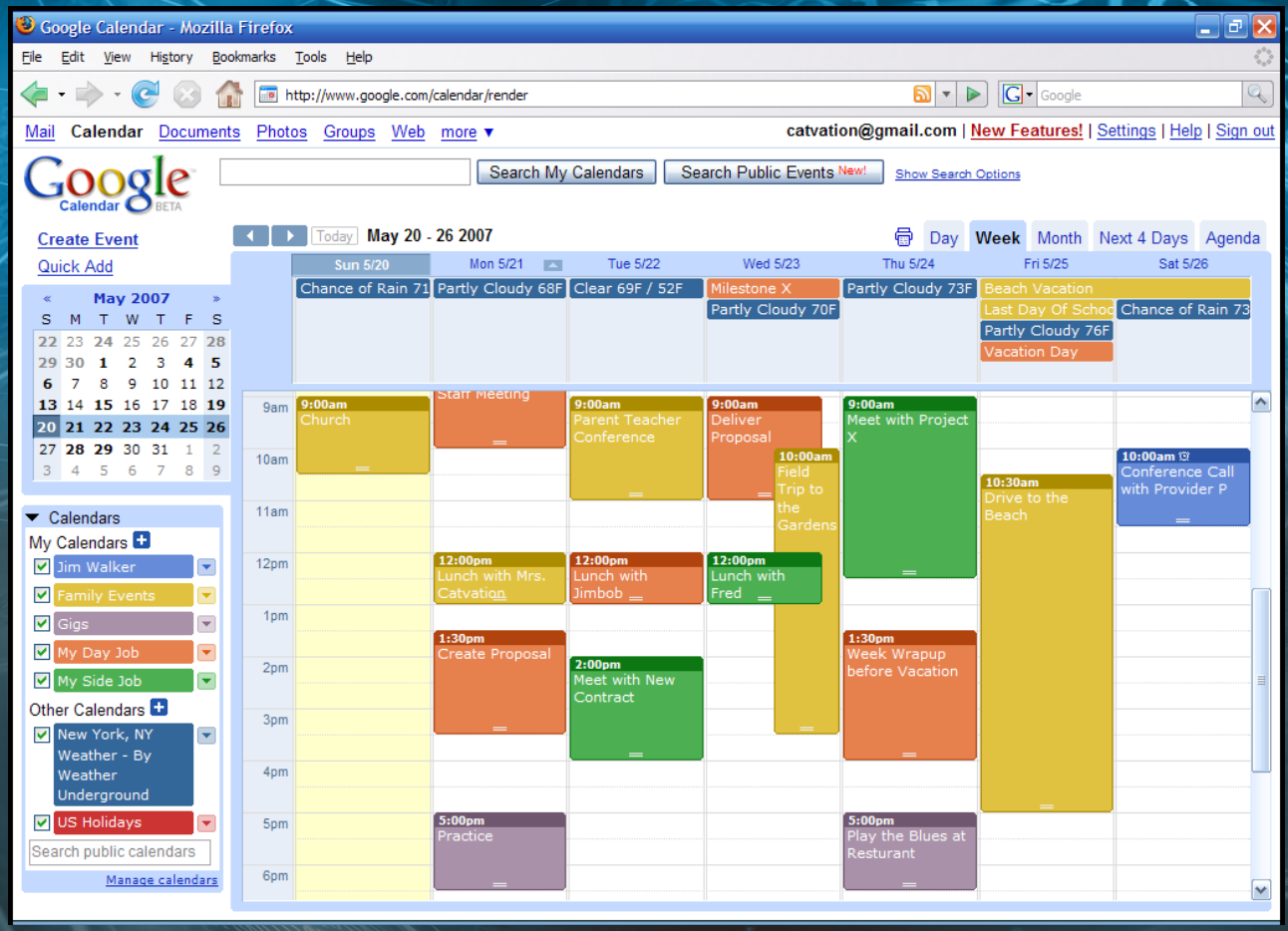


http://xkcd.com/1367/

A Brief History of User Interfaces

- In 1995, with the introduction of JavaScript, web applications began to evolve... slowly
- XMLHttpRequest (1999)
- AJAX (2005)
- HTML 5 and Chromebooks (2011)

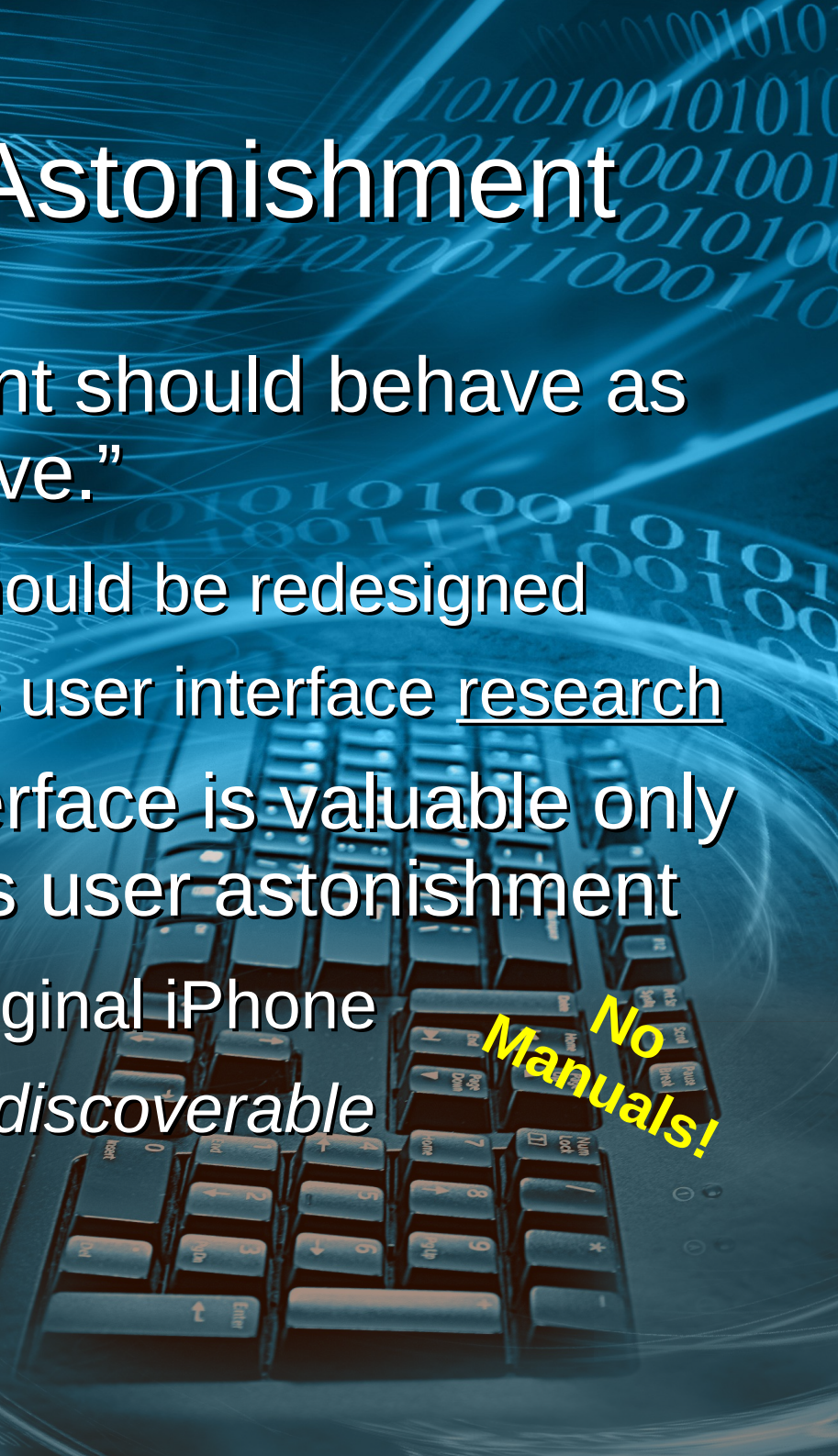




Google Calendar on Firefox Browser

Principle of Least Astonishment

- "A user interface component should behave as the users expect it to behave."
  - If it behaves otherwise, it should be redesigned
  - The natural consequence is user interface research
- Corollary: A novel user interface is valuable only to the extent that it reduces user astonishment
  - A good example was the original iPhone
  - A good interface should be *discoverable*



So... What UI is “Best”?

- Define “best”!
- For “home projects”, write to what you use
  - Usually between GUI, iOS, Android, or web app
  - MVC keeps your “business logic” separate for porting
- For commercial apps, (at least) 3 strategies to start
  - 1 - iOS users will pay for apps – so start there
  - 2 - Use a portable desktop framework (FLTK) and seek sales
  - 3 - Write web apps first (smartphone and tableau layouts) and seek profit from advertisements or subscriptions
- This is a business decision – Your Business May Vary!

Why bother with GUI and graphics?

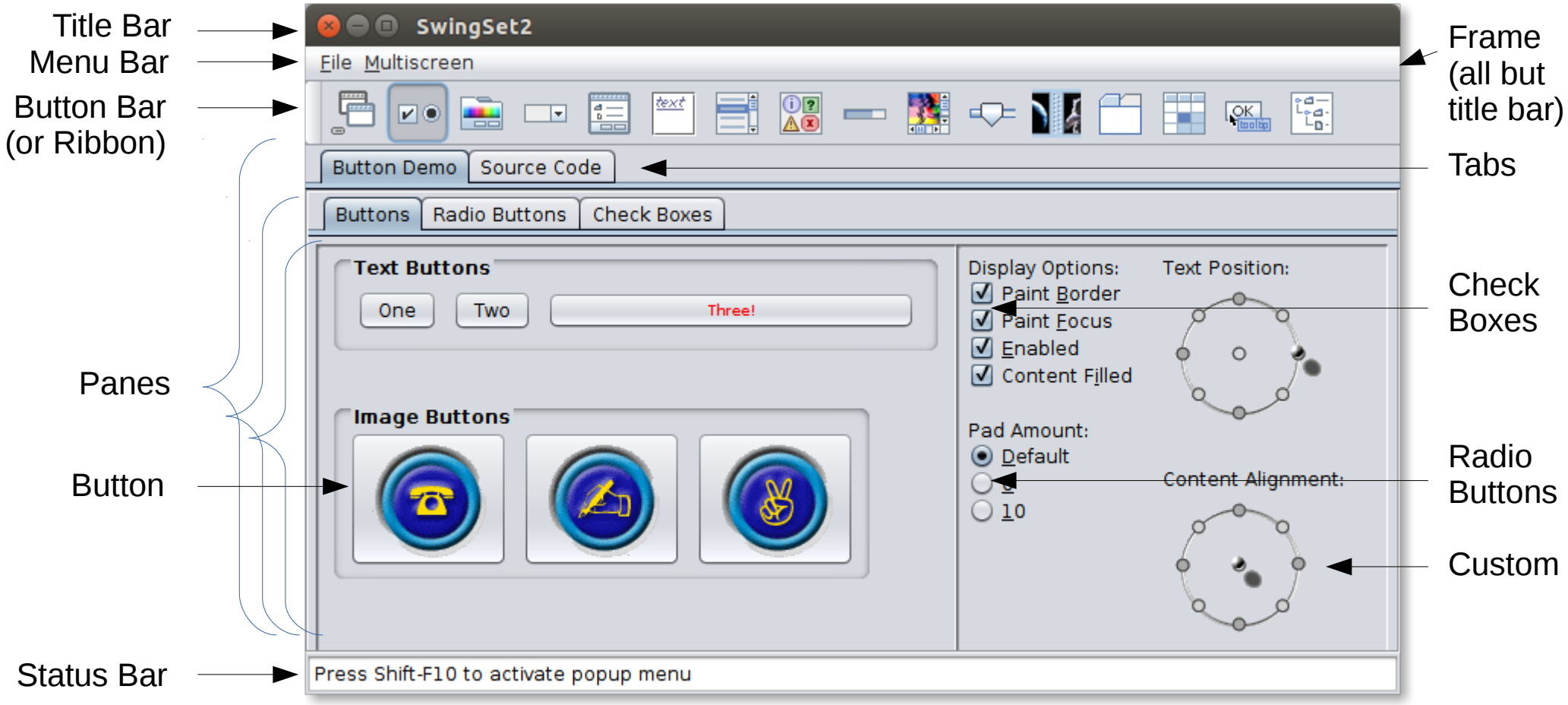
- It’s very common
  - If you write conventional PC applications, you’ll have to do it
- It’s useful
  - Instant feedback
  - Graphing functions
  - Displaying results
- It can illustrate some generally useful concepts and techniques

Why bother with graphics and GUI?

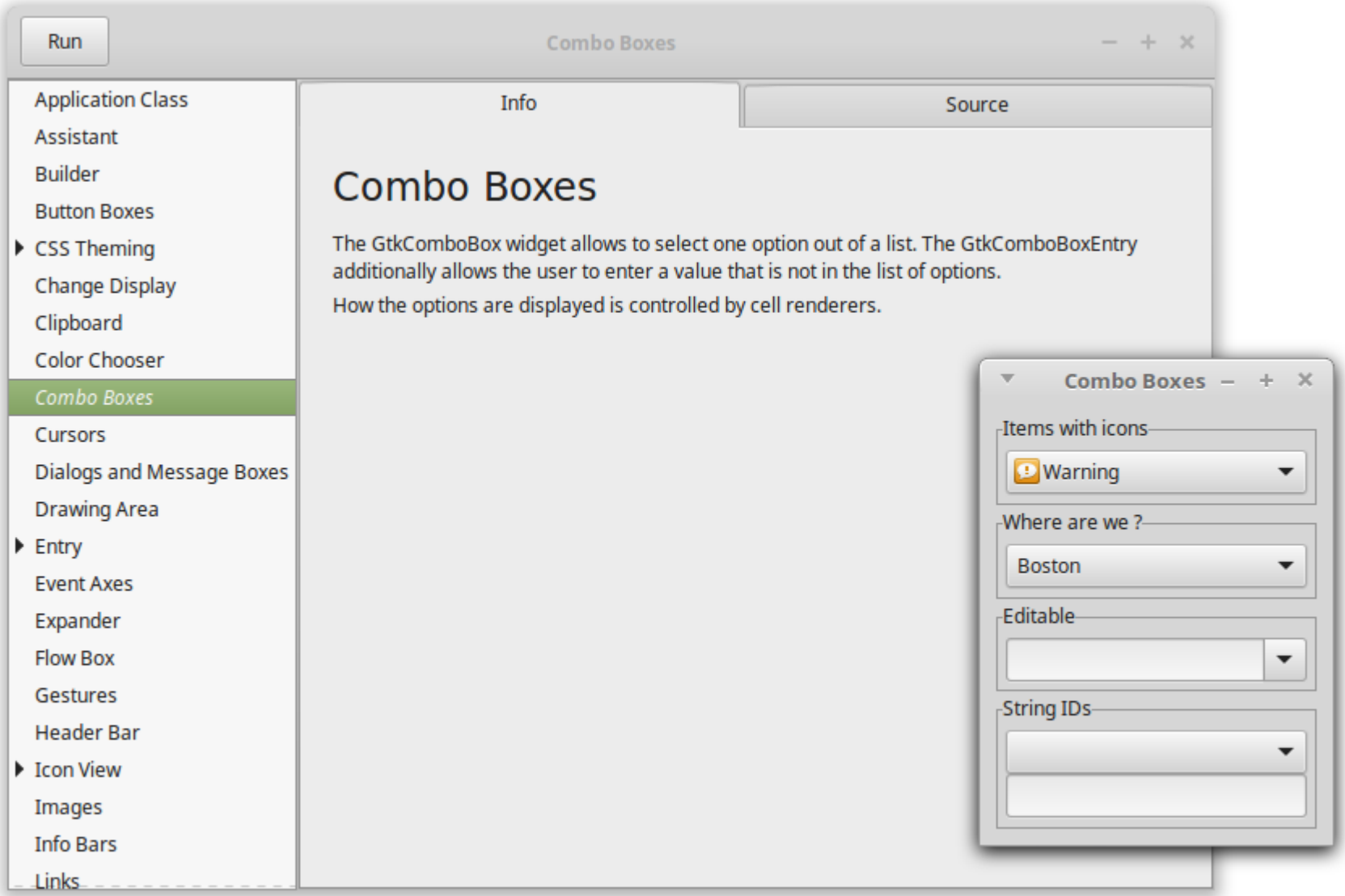
- WYSIWYG (“whizzy-wig”)
  - What you see (in your code) is what you get (on your screen)
- Direct correspondence between concepts, code, and output
- It can only be done well using some pretty neat language features ☺
- Lots of good (small) code examples
- It can be non-trivial to “get” the key concepts
  - So it’s worth teaching
  - If we don’t show how it’s done, you might think it was “magic”
- Graphics is fun!

Common GUI “Widgets”

- This also demos a conventional desktop layout
  (although one created in Java with Swing)



Using gtk3-demo



Note that the Source tab shows C (gtk+) code, NOT C++ (gtkmm) code!

Philosophy: CLI vs GUI

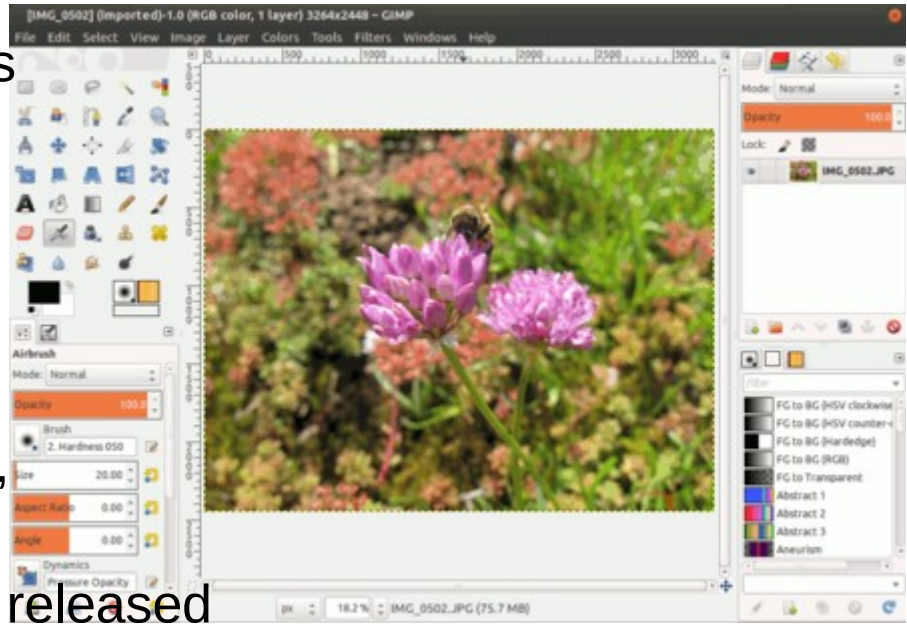
- CLI: The **program** is in control
  - Offers menus and accepts selections
  - Asks questions and accepts answers
  - Limited, keyboard-centric choices
- GUI: The **user** is in control
  - Menus, buttons, and components all available
  - Extensive, mouse or touch-centric choices
  - Click, drag, multi-touch or video gesture, shortcut keystroke, and voice control simultaneously available

Some Popular C++ UI Libraries

- For CLI
  - Built-in: iostream
  - 3rd Party: Gnu readline, CLI toolkit, CLAP / TCLAP
- For GUI
  - Built-in: None
  - 3rd Party native: Windows Foundation Classes (WFC), Windows Template Library (WTL)
  - 3rd Party cross-platform: Qt, GTK+ / **gtkmm**, wxWidgets, FLTK (in textbook)
  - 3rd Party web: CppCMS, Wt

GIMP Tool Kit + (Gtk+)

- Gtk+ is a popular cross-platform GUI toolkit written in C
  - 1995: The GIMP, a free high-end graphics program, was released
  - 1997: Its key graphics primitives were factored out, named the GNU Tool Kit (Gtk), and used to implement GIMP 0.60
  - 1998: The Gtk library was rebuilt using an object-oriented paradigm (in native C), renamed Gtk+, and used for GIMP 0.99
  - 2000: A C++ wrapper named gtkmm was released



- Gtk+ is widely used for desktops (GNOME, Unity, Cinnamon, MATE, and Xfce) as well as all GNOME applications
  - gtkmm is used for applications such as Inkscape and VMware Workstation
- Gtk+ is licensed under the GNU Library General Public License

https://gtkmm.org/en/

Why gtkmm?

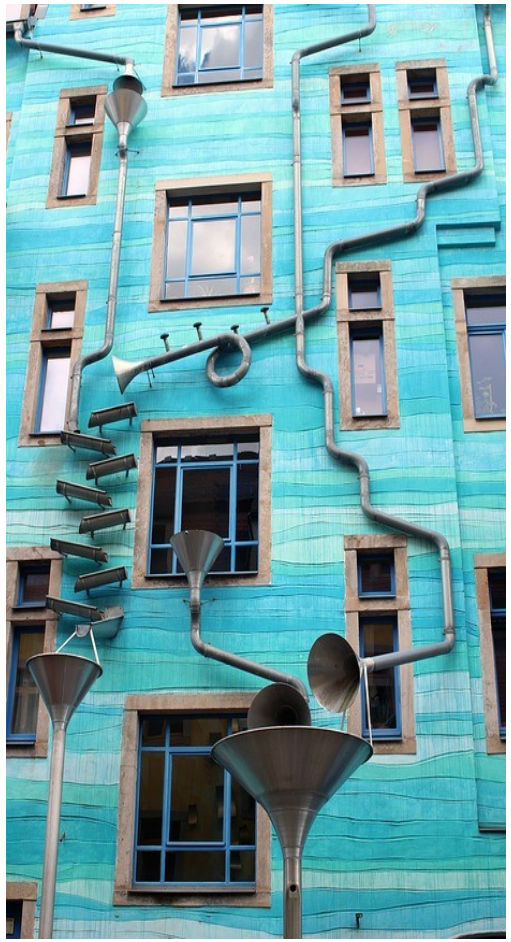
- Must use native C++ object-oriented capabilities
  - “Real” classes with “real” inheritance among widgets
  - “Real” constructors
- Must be widely used in industry for “real” applications
  - Wide use implies (but doesn’t prove) fewer bugs
  - Looks much better on a resume! :-)
- Must have excellent on- line resources
  - Searchable documentation
  - Answer base in Stack Overflow et. al.
- Must be portable to major OS – Linux, Windows, Mac
- Must be simple enough to learn to use in practical applications in only 3½ weeks!

Why Not Something Else?

- Fast Light Tool Kit (FLTK)
  - Used in the textbook, though with a rather extensive facade
  - Rarely used in the “real world”, thus few on-line resources
  - Most common student feedback: “Don’t use FLTK again!”
- Qt
  - Widely used and native C++
  - Implemented early, and thus custom implementation of some features, thus redundant with a lot of subsequently standardized C++
  - Relatively complex even for simple programs
- Anything Else (that’s reasonably popular)
  - wxWidgets is popular, native C++, and a reasonable choice – but rather complex for simple programs
  - Tk is native C and quite dated at this point
  - <Your Favorite Goes Here>

The Ins and (Mostly) Outs of the Façades

- I've mentioned that Dr. Stroustrop's textbook uses a façade named std_lib_facilities.h
  - This attempts to “simplify” learning C++
  - We've carefully avoided using this in class
  - You're experienced enough for “raw C++”
- The textbook also uses a façade for GUI
  - We also carefully avoided using this in class
  - You're also experienced enough for “raw FLTK”, although FLTK is significantly tougher to chew
  - Due to popular demand, however, we won’t cover raw FLTK *this* semester



But what is a façade, *exactly*?

Structural

Façade Pattern

- The **Façade** pattern implements a simplified interface to a complex class or package
  - The Façade class may simplify the interface by making good default assumptions or eliminating rarely used options, reduce dependencies on external libraries, or reorganize the interface for clarity or familiarity

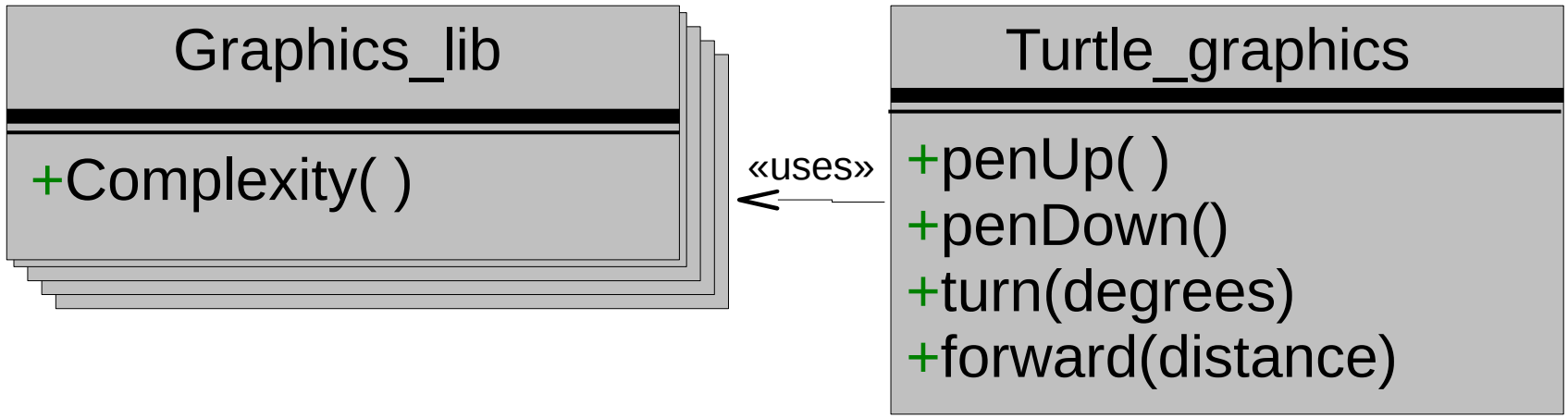
Structural

The Façade Pattern

(Slightly Simplified)

Complex class library

Very simple Façade



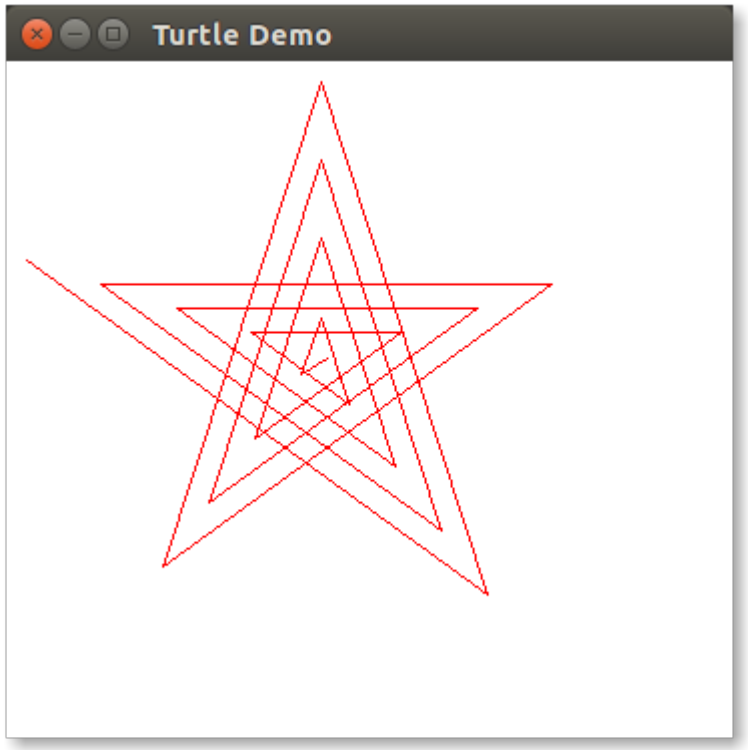
A Turtle Graphics Façade

```
class Line {    // Define a line from (x1, y1) to (x2, y2)
     double _x1, _y1, _x2, _y2;
   public:
     Line(double p_x1, double p_y1,double p_x2, double p_y2)
       : _x1{p_x1}, _y1{p_y1}, _x2{p_x2}, _y2{p_y2} { }
     double x1() {return _x1;}
     double y1() {return _y1;}
     double x2() {return _x2;}
     double y2() {return _y2;}
};
class Turtle_graphics {
       const double d2r = M_PI/180.0; // Degrees to radians
       double x, y, angle; // Current turtle position
       bool pen_is_down;
       vector<Line> lines;
    public:
        void penUp() {pen_is_down = false;}
        void penDown() {pen_is_down = true;}
        void turn(double degrees) {angle += degrees * d2r;}
        void forward(double distance) {
          double x2 = x + distance*cos(angle);
          double y2 = y + distance*sin(angle);
          if (pen_is_down) lines.push_back(Line(x, y, x2, y2));
          x = x2;
          y = y2;
        }
        vector<Line> get_lines() {return lines;}
};
```

gtkmm's line drawing capability
is needed for an actual implementation
(soon...)

Drawing is Simple with Turtle Graphics

```
#include “Turtle_graphics.h”
int main() {
    Turtle_graphics g;
    for (double d=0; d<20; ++d) {
        g.forward(d * 10);
        g.right(144);
    }
}
```



More on GUIs next week!

CSE 1325: Object-Oriented Programming

Agile Processes and Scrum

Mr. George F. Rice

george.rice@uta.edu

Based on material by Bjarne Stroustrup
www.stroustrup.com/Programming

ERB 402
Office Hours:
Tuesday Thursday 11 - 12
Or by appointment

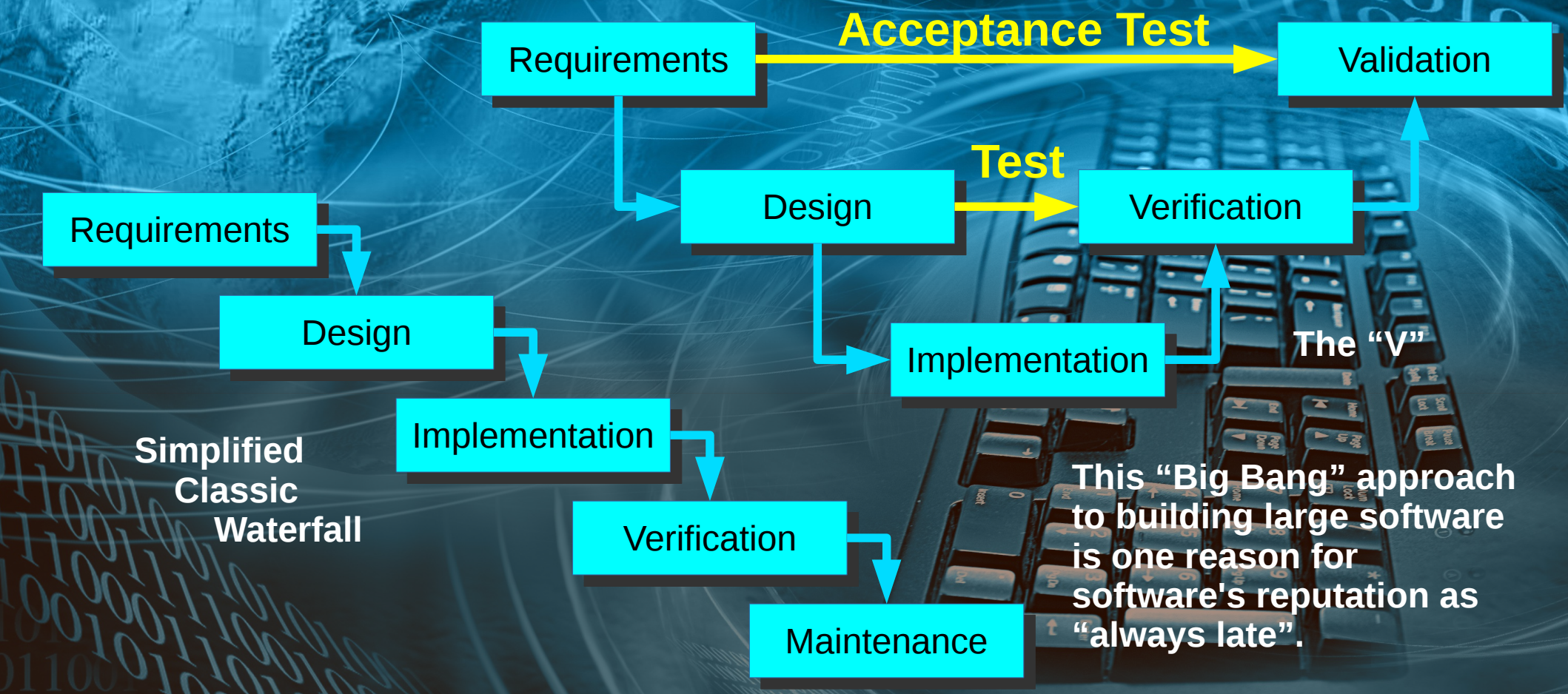
Software Process
How to Manage Large Software Development

- Small programs are easy to build
  - A few classes, some test code, and voilà
- Large programs are hard to build
  - Or are hard to build correctly
  - Or are hard to build on time and within budget
- Large teams are hard to manage
  - Communication paths grow quadratically n(n-1)/2
  - The project vision blurs with distance
- The Second System Effect identifies Version 2.0 as the most dangerous version ever attempted
  - Architects tend to include *every feature* omitted from Version 1.x
  - Duke Nukem Forever, Windows Longhorn, Animusic 3...

99 little bugs in the code.
99 little bugs.
Take one down,
Patch it around.
127 little bugs in the code...

The Waterfall Process

- The Waterfall Process was defined by Winston W. Royce in 1970



Waterfall Issues

- Getting requirements correct is **very** difficult
  - Communication issues / ambiguity
  - The Customer is always right... they just have no idea what they're talking about
- Requirements change throughout the development
  - Freezing requirements at the top of the waterfall **ensures** delivery of a suboptimal system at the bottom
- All requirements are typically completed before delivery
  - But 20% of requirements may deliver 80% of value

The Agile Manifesto

- **Individuals and interactions**
  over processes and tools
- **Working software**
  over comprehensive documentation
- **Customer collaboration**
  over contract negotiation
- **Responding to change**
  over following a plan

Agile can help, but there are no silver bullets.

There is value in the lower items
but greater value in the upper items

Scrum

- Scrum is an agile **process**, not a software development **method**
  - Excels at rapid, flexible response to change in requirements and other unexpected (or expected) challenges
  - Can manage any activity that delivers value
  - For software, often paired with Extreme Programming (which we will NOT cover)
- In use since 1986
  - Named Scrum in 1995
  - First used to design cars, printers, and photocopiers

Scrum Roles

- Product Owner
  - That's your TA and me :-)
- Scrum Master
  - The team's shepherd, champion, facilitator, and (typically) expert in Scrum
  - We're flexible here
- Team Member
  - That's you!

4 Primary Scrum Artifacts
(for modestly sized projects)

- Product Backlog
  - Anything of value to the Product Owner
  - Prioritized by the Product Owner
- Sprint Backlog
  - The team's To Do List for a single sprint
  - Consists of tasks that must be done to clear a backlog item
  - Each task belongs to exactly one person
- Burn Chart
  - Simple line graph of work remaining over time
  - We hope the slope is downward!
- Task Board
  - List of tasks for the sprint arranged in columns – To Do, Doing, Done
  - Provides visibility of activity across the team!

Sprints

- All work is organized into **sprints**
- A sprint is a time-boxed period of work
  - Implements 1 or more Product Backlog items
  - Consists of a number of tasks required to implement those items
- Each sprint ends when scheduled
  - Any incomplete Product Backlog items go back on the backlog for a later sprint
  - The product **must** be in deliverable shape – compiles, passes all tests, needed docs, ready to deliver
  - A demo to the Product Owner follows every sprint

Simple Product Backlog with Burn Chart

Product Name:
Team ID:

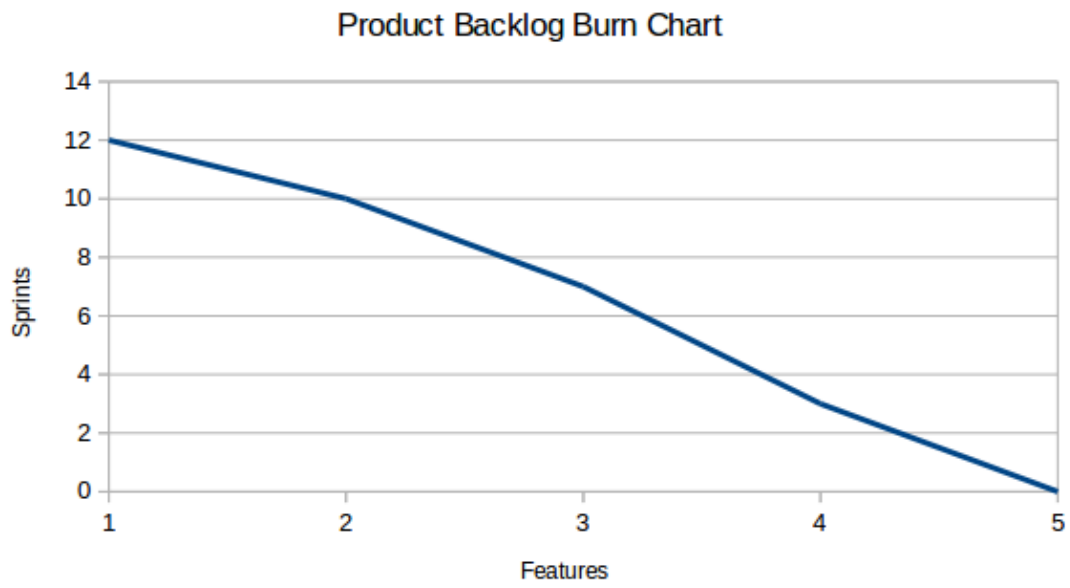
Name:
Initials:
Student ID:

Name:
Initials:
Student ID:

Name:
Initials:
Student ID:

Name:
Initials:
Student ID:

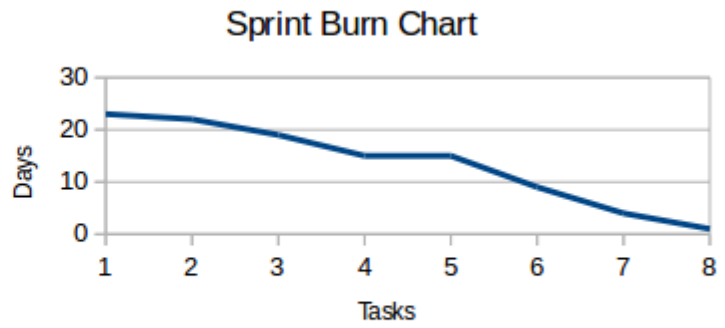
Total Features	**12**
Sprint 1 Left	**10**
Sprint 2 Left	**7**
Sprint 3 Left	**3**
Sprint 4 Left	**0**



Feature ID	Sprint #	Category	As a...	I want to...	So that...	Status	Notes

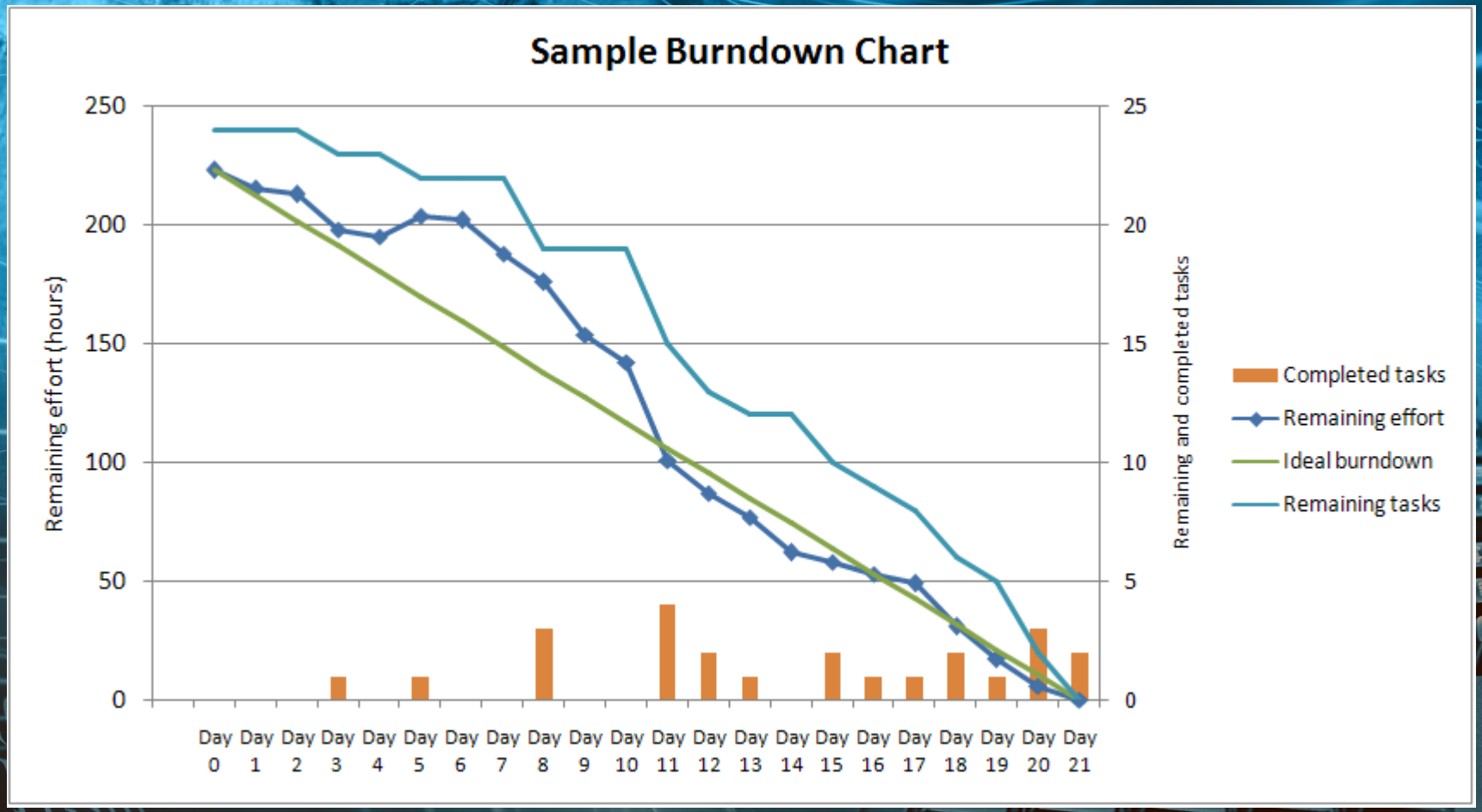
Simple Sprint Backlog with Burn Chart

Sprint #	
Start on	
End on	
Demo on	
Total Tasks	**23**
Day 1 Left	**22**
Day 2 Left	**19**
Day 3 Left	**15**
Day 4 Left	**15**
Day 5 Left	**9**
Day 6 Left	**4**
Day 7 Left	**1**



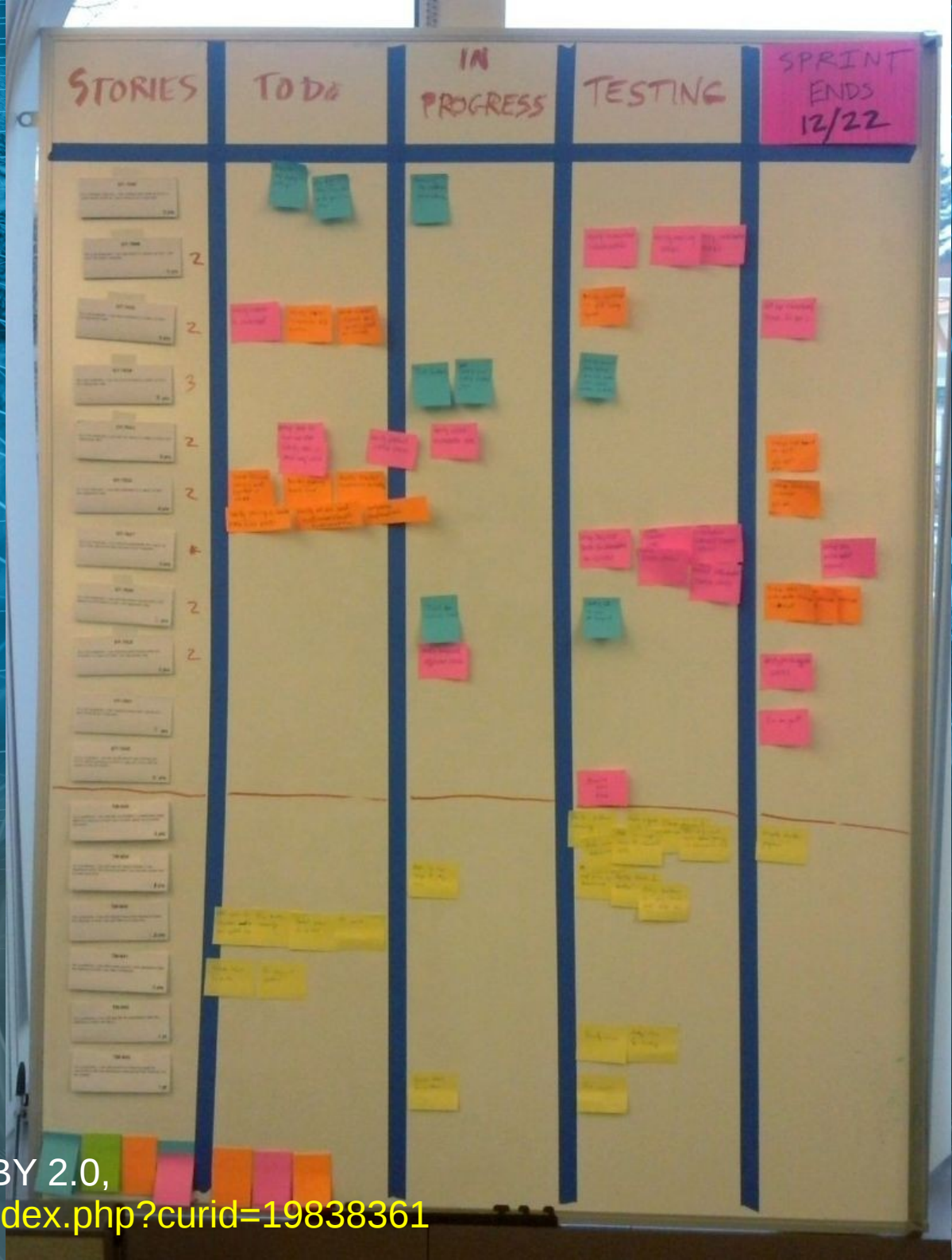
Task ID	Feature ID	Category	Type	Assigned To	Description	Status	Notes

More Complex Sprint Burn Chart



By Pablo Straub - Own work, Public Domain
https://commons.wikimedia.org/w/index.php?curid=7132232

Sample Physical Taskboard



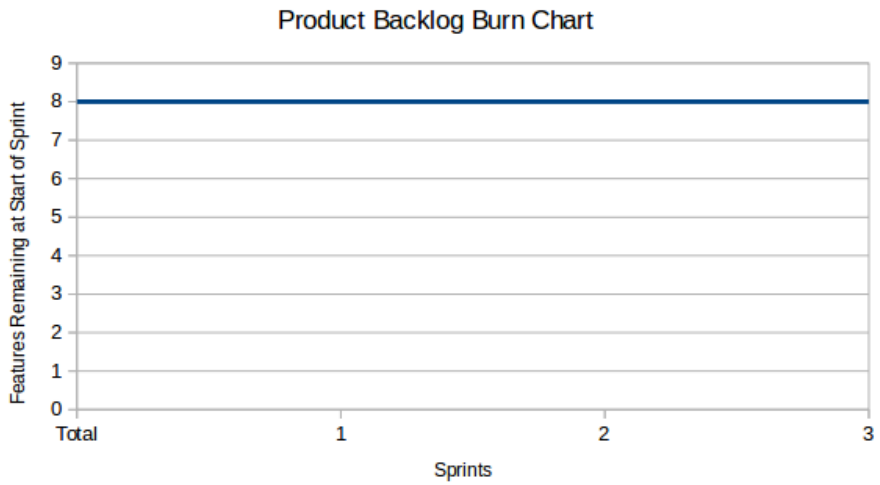
Homework Scrum Spreadsheet
Product Backlog

Product Name: **C1325 Library Management System**
Team ID:

Complete Fields in Green!!!

Name:
Initials:
Student ID:

Teaming is not available for this assignment!
Optional teaming may be available in future assignments



	Remaining	Completed (this sprint)
Total Features	8	
Sprint 1 Left	8	0
Sprint 2 Left	8	0
Sprint 3 Left	8	0

Note: Priority of unfinished Features is subject to change at the end of each sprint at the whim of the Product Owner
Additional features may be proposed by the student but must be approved by the Product Owner in writing

Feature ID	Prio	Status	Planned	Sprints Status	As a...	I want to...	So that...	Notes
AP	1	NS	1		LIB	Add a publication	I can keep track of what is in the library	
LP	2	NS	1		LIB	List all publications	I can find out what is in the library	This will also be needed for the next two features
CO	3	NS	1		LIB	Check out a publication	I can track who has borrowed each publication	Each publication will need to know if it is checked out or not
CI	4	NS	1		LIB	Check in a publication	I can tell when a publication is returned	
HE	5	NS	1		LIB	Get help	I can learn how to use the system	
PA	6	NS	1		LIB	Add a patron	I can conveniently keep track of all patrons	This is the Bonus level
SP	7	NS	1		LIB	Select a patron when checking out a pub	I don't have to rekey recurring patron info	This is the Bonus level
CU	8	NS	1		LIB	Add custom info for each publication type	I can keep better track of library assets	This is the Extreme Bonus level

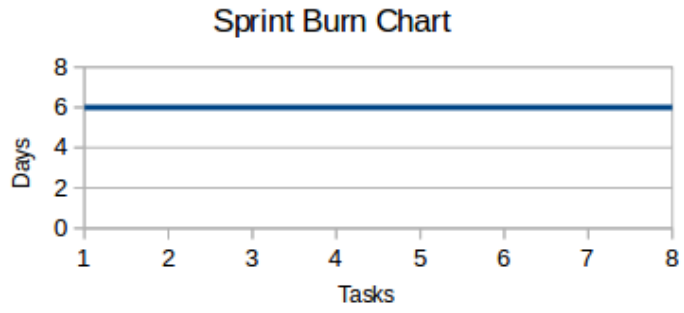
Homework Scrum Spreadsheet
Sprint Backlog

Sprint #	1
Start on	Sep 21
End on	Sep 28
Demo on	TBD

Complete Fields in Green!!!

***** Create a new Sprint Backlog for EVERY SPRINT *****

	Remaining	Completed (this day)
Total Tasks	6	
Day 1 Left	6	0
Day 2 Left	6	0
Day 3 Left	6	0
Day 4 Left	6	0
Day 5 Left	6	0
Day 6 Left	6	0
Day 7 Left	6	0



Feature ID	Description	Status	Notes
AP	Create a Publication class with data and a constructor		Included a WRITTEN test with each constructor and method!
AP	Add the to_string method to Publication		
AP	Add the is_checked_out method to Publication		
AP	Add the check_out method to Publication		
AP	Add the check_in method to Publication		
LP	--> Add additional tasks to complete the homework		
CO			
CI			
HE			
PA			
SP			
CU			

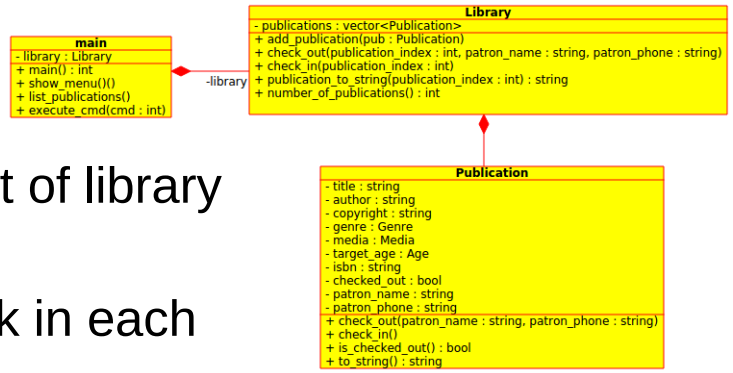
Multi-Week Projects

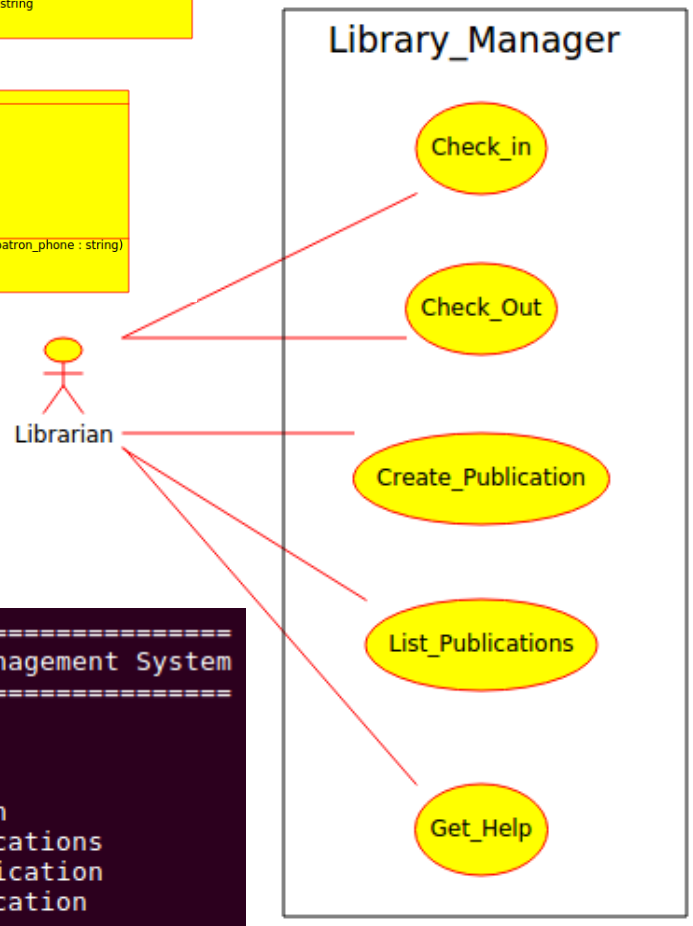
- Work individually*
- Work will be managed by super-simplified Scrum
  - Use git version management as always
  - A Product Backlog and Sprint Backlog will be maintained with automatically generated burn charts (delivered after each sprint)
  - Your sprint backlog will be pre-defined but is negotiable (especially on later sprints)
  - A *possible* demo to the class after the last sprint
    - To be scheduled as time permits

* Team development *may* become available later, **NOT for Homework #4**

Homework #4
Sprint 1

- Build a simple Library Management System
  - Add publications to the list of library assets
  - Check out and check back in each publication
  - Provide basic help
- As always, use git for version management
- Manage the sprint with the Scrum spreadsheet
- Details are on Blackboard
- **Due September 28 at 8 am**





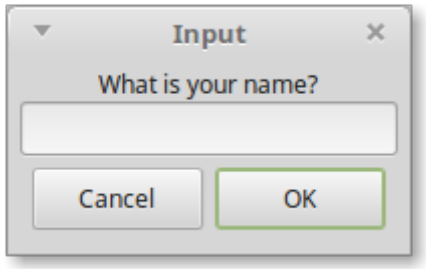

```
==================================
CSE1325 Library Management System
==================================

Publications
------------
(1) Add publication
(2) List all publications
(3) Check out publication
(4) Check in publication

Utility
-------
(9) Help
(0) Exit

Command?
```

Sprints

Feature ID	Priority	Planned	Status	As a...	I want to...
AP	1	1		Librarian	Add a publication
LP	2	1		Librarian	List all publications
CO	3	1		Librarian	Check out a publication
CI	4	1		Librarian	Check in a publication
HE	5	1		Librarian	Get help

Previews of Sprints #2 and #3
(Also known as Homework #5 and #6)

- Sprint #2
  - Add *dialogs*



- Sprint #3
  - Add a *main window*

For Next Class

- Review Exam #1
- We begin GUIs in gtkmm!
  - You'll need the gtkmm libraries in your environment (they are pre-installed in the VM!)