Growth of a small GUI

Antelope Users' Group Meeting

June 11, 2006

IRIS, Tucson, AZ

Dr. Kent Lindquist

Lindquist Consulting

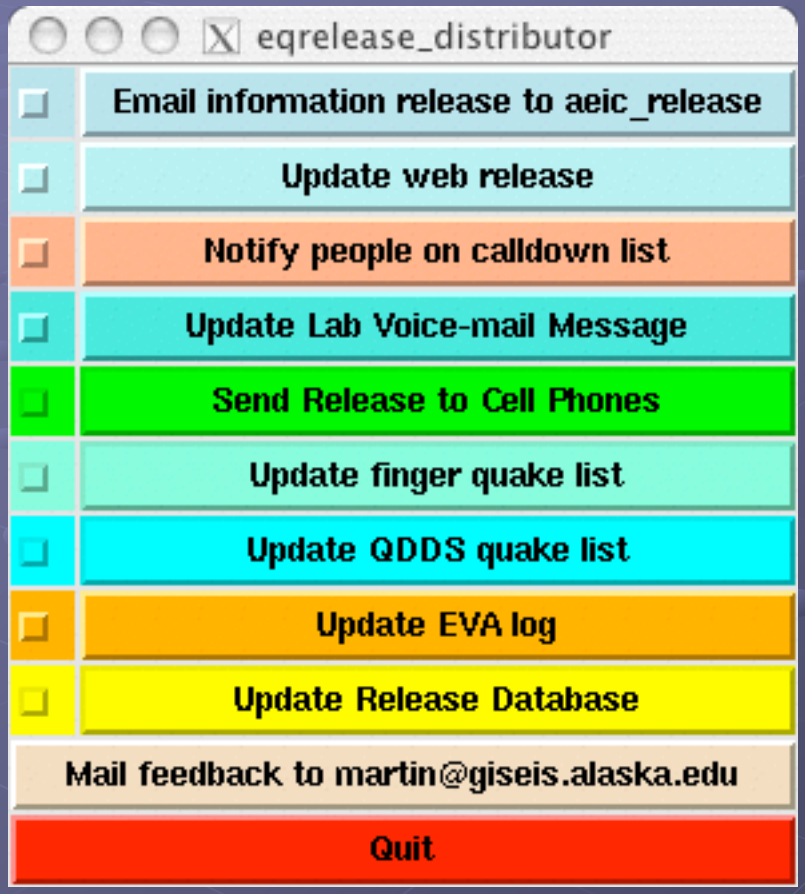
Storyboard

- User has a procedural to-do list
- User launches application to track progress through the list
- Each step gets a button, which you can click to record step completion or step skipping
- Each step might execute something
- Step can be marked intentionally skipped
- Checklist warns on exit about missed steps

Background

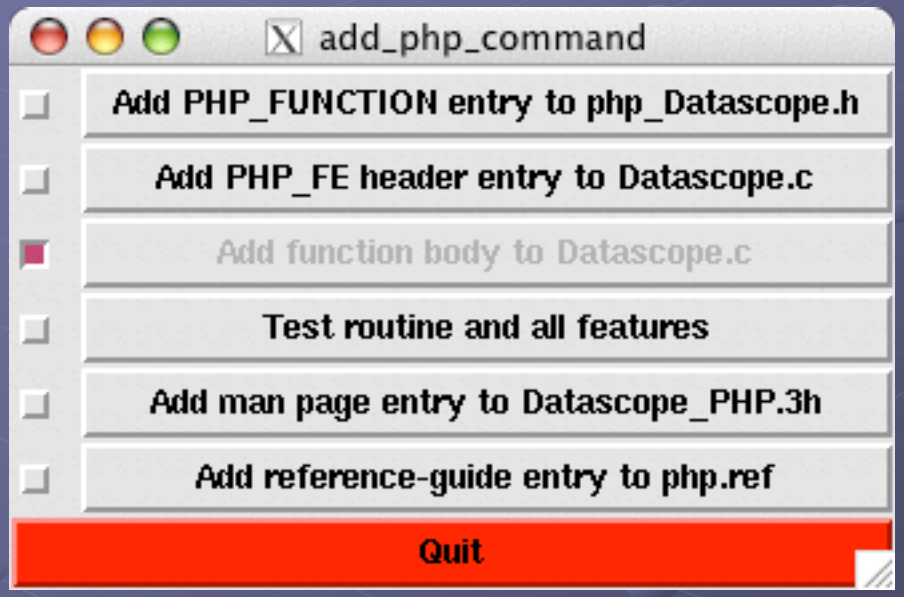
- I've written several of these
  - aeic_release_distributor
  - Eqcellphone_release
  - Add_matlab_command
  - Add_php_command
- All are repetitive but custom one-off scripts
- Have one GUI element written (CommandCheckoff.tcl (3t))

eqrelease_distributor



eqrelease_distributor

Add_matlab_command, add_php_command



New program: checklist

- Generic version
  - checklist.pf names the steps
- Coding time:
  - Initial: 45 minutes
  - Enhancements: 3 hrs and counting…
  - Current: 174 lines of code

Checklist

eagle% checklist

Usage: checklist template

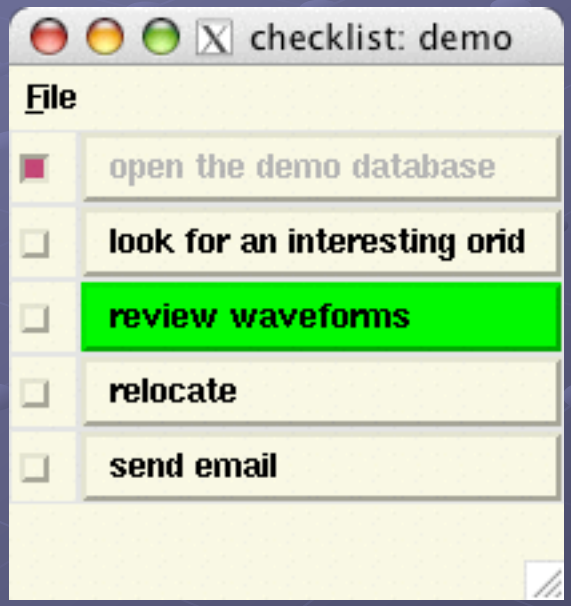
Configured checklists are:

add_pfdbquery_command
demo
add_php_command
add_matlab_command

eagle%

Checklist

% checklist demo

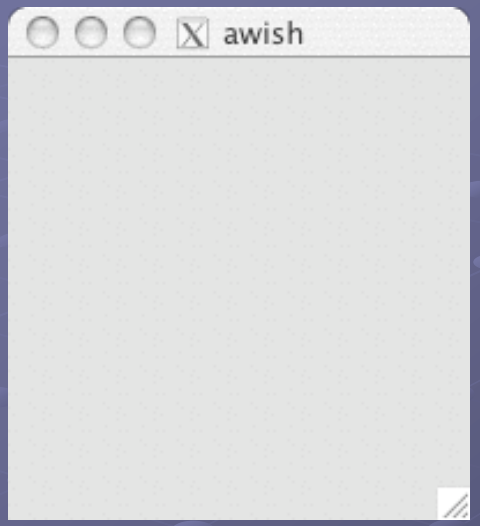


checklist.pf (first version)

```
demo  &Tbl{
    &Arr{
        step    open the demo database
    }
    &Arr{
        step    look for an interesting orid
    }
    &Arr{
        step    review waveforms
    }
    &Arr{
        step    relocate
    }
    &Arr{
        step    send email
    }
}
```

1) “Wheels underneath…”

eagle% awish



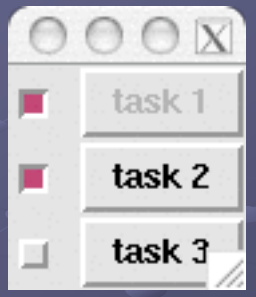
% button .b -text "Hi Kent"
.b
% pack .b

2) Walk before you fly

% man CommandCheckoff

% template xwish > mytest

% cat >> mytest




```
package require Tclx
set auto_path [linsert $auto_path 0 $env(ANTELOPE)/data/tcl/library_contrib]
CommandCheckoff .c1 -label "task1" -variable v1 -command {}
CommandCheckoff .c2 -label "task2" -variable v2 -command {}
CommandCheckoff .c3 -label "task3" -variable v3 -command {}
pack .c1 .c2 .c3
```

% ./mytest

Invest: proc add_step { w step command}

```
proc add_step { w step command } {

        global numsteps

        set stepw $w.s$numsteps

        CommandCheckoff $stepw -label "$step" \
                        -command "$command" \
                        -variable var$numsteps

        pack $stepw -side top -fill x

        incr numsteps

        return
}
```

Restructure

```
proc add_step {
….
}
frame .commands
pack .commands

add_step .commands  “task 1” {}
add_step .commands “task 2” {}
add_step .commands  “task 3” {}
```

Step into line

- mkmk - > Makefile
  - edit; result has 3 or 4 lines
- manpage - > checklist.1
  - Edit--just a few lines, 45 seconds first pass
  - Keep expanding throughout
- mv mytest checklist.xwish
  - remove the awish headers
  - make will put them in

Invest: pf driven

- Write checklist.pf as shown above
- Add a ‘command’ array entry since CommandCheckoff.tcl requires it: e.g.

```
demo &Tbl{
    &Arr{
        command exec dbe /opt/antelope/data/db/demo/demo &
        step    open the demo database
    }
```

Restructure

```
set Pf checklist
pfgetarr templates_arr $Pf
set templates [array names templates_arr]

set itemlist [pfgetlist @$Pf#$template]

 foreach itemarr $itemlist {

            set itemstep [pfget $itemarr step]
            set itemcommand [pfget $itemarr command]

            add_step $w $itemstep $itemcommand
 }
```

Invest & Restructure

- Encapsulate the above code into a create_commands procedure

```
proc create_commands {} {
……..
}
wm title . "checklist: $template"
wm minsize . 200 200

create_menubar
create_commands
```

- Why? Your GUI code will get complicated fast.

Add menus

```
proc create_menubar {} {
        global bg

        set w .menubar

        frame $w -background $bg

        menubutton $w.file -bg $bg -text File \
                -underline 0 -menu $w.file.m -highlightthickness 0

        menu $w.file.m -tearoff 0 -bg $bg

       $w.file.m add command -label Quit -underline 0 -command "destroy ."

        pack $w.file -side left

        pack $w -side top -fill x
}
```

Don’t forget command-line parsing…

```
if { $argc < 1 } {

        puts stderr "\n   Usage: checklist template\n”

} else {

        set template [lindex $argv 0]
}
```

Add Feature: Cntl-C

```
bind . <Control-KeyPress-c> "destroy ."
bind . <Control-KeyPress-C> "destroy ."
```

*** FIX *** glitches

- Mac OSX requires replacement of

  package require Tclx

  With something like

  catch {package require Tclx}

  Or in this case

```
set auto_path [linsert $auto_path 0 $env(ANTELOPE)/data/tcl/library]
```

- Source: community knowledge

Add Feature: list Reset

- Add a reset_steps {} proc (7 lines of code)
- Add an item to the ‘File’ menu for it
- Add a Cntl-R binding for it (upper and lowercase)

Add feature: list available templates

-
- (Now we're having fun!)
- Usage-line check section gets 12 extra lines of code to give the list of configured checklists

** FIX *** feature: text alignment

- Need text left-justified in the buttons instead of centered (looks sloppy…)
- Correct place: add a general capability to CommandCheckoff.tcl tool to take arguments for label justification and label anchoring
- New features are designed to “feel like” the rest of the TCL interface

Add feature: colored mouse-overs

- Re-color button on mouse-over: green if command will launch, yellow otherwise
- Requires two new args to CommandCheckoff.tcl (faithful to TCL interface)
- Add Config section to checklist.pf (better than the initial hard-wires)
- Keep above from breaking usage-line template listing capability

Add feature: balloon instructions

- Add optional ‘description’ parameter for each command
- Hunt through my old code for a way to do balloon help
- Vaguely recall DanQ had something
- Find DanQ’s balloon_set(3T) (he got it from Stewart Allen, apparently…)
- Add balloon for each button, with description
- Test that exercising the option of omitting ‘description’ still produces working code

Add feature: balloon instructions

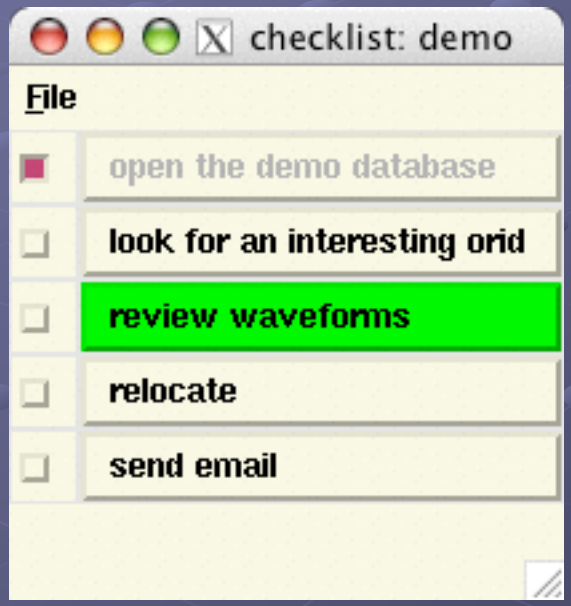
```
&Arr{
        command
        description     &Literal{
                Find an origin of choice in the origin table
                from the dbe instance launched in the previous step
        }
        step    look for an interesting orid
    }
```

- Make the irritating little balloons optional…

(another Config entry in checklist.pf)

Checklist

% checklist demo



Conclusions

- Evolutionary delivery
  - Start simple
  - Always one step away from a working piece of code
- You have to know the language
- You have to know and/or find the tools/toolkit(s)
- You have to be willing to invest.
  - Notice the multiple rewrites
  - Fix every glitch you find
  - Small learning stages at each iteration
  - Code cleaning / invest / restucture; repeat…
- Add (needs-driven, irritation-reducing) features!