Bringing UX to Your Code

Joe Mastey
@jmmastey
I help companies build awesome internal education programs
I’ve worked in software for a million years.
I am not a UX designer / developer
“Norman discussed door handles at length.” - Wikipedia
It’s not your fault
PUSH
## cyoa/weather/.gitignore

```bash
+ # See https://help.github.com/articles/ignoring-files for more about ignoring files.
+ # If you find yourself ignoring temporary files generated by your text editor
+ # or operating system, you probably want to add a global ignore instead:
+ # git config --global core.excludesfile ~/.gitignore_global
+ + # Ignore bundler config.
+ + ./bundle
+ + # Ignore the default SQLite database.
+ + # Ignore all logfiles and tempfiles.
+ + # Ignore Spring files.
```

## cyoa/weather/.rspec

```bash
+ ---color
+ ---require spec_helper
```
Things We Ignore:
Things We Ignore:
1. Anything invented a long time ago
Things We Ignore:
1. Anything invented a long time ago
2. Anything invented by non-engineers
His new black-rimmed glasses immediately made him 80% more quoteable.
decades of research < shit I made up
We’re missing out
Our code *is* an interface
Our users are stressed
Let’s try it!
Gulf of Execution
Is there an action that corresponds to my intentions?
An example: deleting an ActiveRecord object
macguffin.methods.sort
[:]


to_gid, :to_global_id, :to_json, :to_json_with_active_support_encoder, :to_json_without_active_support_encoder, :to_key, :to_model, :to_param, :to_partial_path, :to_query, :to_s, :to_sgid, :to

transition=, :status_events, :status_name, :status_paths, :status_transitions, :status_was, :status_will_change!, :store_full_sti_class, :store_full_sti_class?, :suppress, :suppress_warnings, :syck_


bute, :quietly, :quietly_with_deprecation_silenced, :quietly_without_deprecation_silenced, :quoted_id, :raise_in_transactional_callbacks, :raise_record_invalid, :read_attribute, :read_attribute_be


cast, :id_change, :id_changed?, :id_was, :id_will_change!, :in?, :include_root_in_json, :include_root_in_json=, :include_root_in_json?, :increment, :increment!, :init_with, :initialize_internals_
macguffin.methods.grep /...?/
The model *in my head* is what I use to search.
HTTP Verb: DELETE
SQL Statement: DELETE FROM *
Arrays: delete, delete_at, delete_if
Every OS Ever: DELETE
macguffin.methods.grep /delete/
=> [:delete]
macguffin.destroy
It’s completely unnecessary friction
Gulf of Evaluation
Did it work?
This is easy(ish) in code
macguffin.delete
=> #<MacGuffin id=75 ...>
macguffin.destroy

=> #<MacGuffin id=75 ...>
class MacGuffin
  private
  def self.retrieve
    ...
  end
end
Natural Mappings
What does that look like in code?
my_stuff.sort
my_stuff.first
my_stuff.delete :foo
brew update
brew upgrade
Design for Errors
Software is screwed up most of the time
undefined is not a function
“also, screw you”
Spooky Nulls at a Distance
Exploit the Power of Constraint
Make it hard to do it the wrong way
mysql --i-am-a-dummy
irb(main):042:0> User.last.destroy
=> <User name="Ron Rando" ...>
"You should think: 'If I was really smart I could write code that drunk children would understand'" - Sandi Metz #womenwhocode #railsconf
“So You Wanna Be a User Experience Designer”
4. Group Related Objects
Proximity implies relationship
```ruby
def say(message, options={})
  command = "cowsay"
  if options[:strings] && options[:strings][:eyes]
    command << " -e \'#{options[:strings][:eyes]}\'"
  end

  messages = case message
    when Array then message
    when nil then []
    else [message]
  end

  results = []
  messages.each do |message|
    @io_class.popen(command, "w") do |process|
      process.write(message)
      process.close_write
      results << process.read
    rescue Errno::EPIPE
      message
    end
  end
  output = results.join("\n")
  if options[:out]
    options[:out] << output
  end
  destination = case options[:out]
    when nil then "return value"
    when File then options[:out].path
    else options[:out].inspect
  end
  @logger.info "Wrote to #{destination}"
  if $? && ![0,1,72].include?( $? .exitstatus )
    raise ArgumentError, "Command exited with status #$?.exitstatus.to_s"
  end
  output
end
```
```ruby
def say(message, options={})
  return "" if message.nil?
  options[:cowfile] and assert(options[:cowfile].to_s =~ /\s*$/)  
  width = options.fetch(:width) { 40 }
  eyes = Maybe(options[:strings][:eyes]
  cowfile = options[:cowfile]
  destination = WithPath.new(options[:out]).path
  out = options.fetch(:out) { NullObject.new }
  messages = Array(message)
  command = "cowsay"
  command << " -W #{width}" unless eyes.nil?
  command << " -e "#{options[:strings][:eyes]}"" unless eyes.nil?
  command << " -f #{options[:cowfile]}" unless cowfile.nil?
  results = messages.map { |message|
    checked_popen(command, "w+", lambda(message) do |process|
      process.write(message)
      process.close_write
      process.read
    end
  }
  output = results.join("\n")
  out << output
  @logger.info "Wrote to #{destination}"
  output
end
```
what about methods in a class?
what about similarly named classes / methods?
macguffin.accounts_add
macguffin.accounts_delete
macguffin.accounts_list
18. Be Consistent
I’m not saying your code isn’t a beautiful and unique snowflake
rails dbconsole production
rails console production
rails server -e production
def update
  @macguffin = MacGuffin.find(params[:id])
  ...
end

def show_macguffin
  m = MacGuffin.find(params[:macguffin_id])
  ...
end
"gulp".upcase  =>  "GULP"
"gulp".upcase => "GULP"
"GULP".upcase => "GULP"
"gulp".upcase => "GULP"
"GULP".upcase => "GULP"
"gulp".upcase! => "GULP"
"gulp".upcase          => "GULP"
"GULP".upcase          => "GULP"
"gulp".upcase!         => "GULP"
"GULP".upcase!         => "GULP"
“gulp”.upcase               #=> “GULP”
“GULP”.upcase               #=> “GULP”
“gulp”.upcase!              #=> “GULP”
“GULP”.upcase!              #=> nil?!
params[:product_code].upcase!
params = { product_code: "fo0" }

# do work

expect(macguffin.upc).to eq("F00")
Consistent with what?
16. Use Emotion
We have an emotional relationship with the software we use
A demonstration
See?
Pending:
   Some class should be pending
      # Not yet implemented
      # ./spec/pride_spec.rb:19

Failures:

   1) Some class should fail
      Failure/Error: expect(1).to eq 0

         expected: 0
         got: 1

         (compared using ==)
      # ./spec/pride_spec.rb:27:in `block (2 levels) in <top (required)>'

Fabulous tests in 0.00468 seconds
62 examples, 1 failures, 1 pending
This isn’t about kitsch
Warmth and kindness make software a pleasure to use
9. Avoid Jargon
stop_csrf
versus
protect_from_forgery
AJAX
def __private_method
7. Provide Signposts and Cues
Where you came from, where you’re going
RTFM? STFU.
User.find_by_email_and_first_name
User.find_by(email: '', first_name: '')
One thing I like about rspec is that it prints exactly how to run the failing test at the end. Wish I could do it with minitest.
Failures:

1) MyObject works at all
   Failure/Error: expect(1).to eq(0)
     expected: 0
     got: 1

Finished in 0.00375 seconds (files took 0.29196 seconds to load)
1 example, 1 failure

Failed examples:

rspec ./bomb_spec.rb:4 # MyObject works at all
> rspec bomb_spec.rb

Failures:

1) MyObject works at all
   Failure/Error: expect(1).to eq(0)

   expected: 0
   got: 1

... stacktrace ...

Finished in 0.00375 seconds (files took 0.29196 seconds to load)
1 example, 1 failure

Failed examples:

  rspec ./bomb_spec.rb:4 # MyObject works at all
Failures:

1) MyObject works at all
   Failure/Error: expect(1).to eq(0)
       expected: 0
       got: 1

Failed examples:

rspec ./bomb_spec.rb:4 # MyObject works at all
rspec ./bomb_spec.rb:4 # MyObject works at all
@macguffin = MacGuffin.find(1)
So what’s the point?
“The world is filled with plenty of anguish — make your life goal not to add to it.”
This is the mechanism for self documenting code
The part where I ask you for things
“So you wanna be a user experience designer”
- Whitney Hess (http://bit.ly/1Hn1nQS)

“The Design of Everyday Things” - Don Norman
(I dunno, Amazon?)
Thanks!

Frustration Photo by jseliger2 (https://www.flickr.com/photos/91262622@N02/)
Happy Kids by deepblue66 (https://www.flickr.com/photos/deepblue66/14503838952)
Annotated code by Avdi Grimm (https://avdi.org)
All icons from the NounProject (https://nounproject.com)