Real-world Concepts

Kevin Murray

murmeister@hotmail.com

Overview

Integrating UI library

Adding content dynamically

Passing values

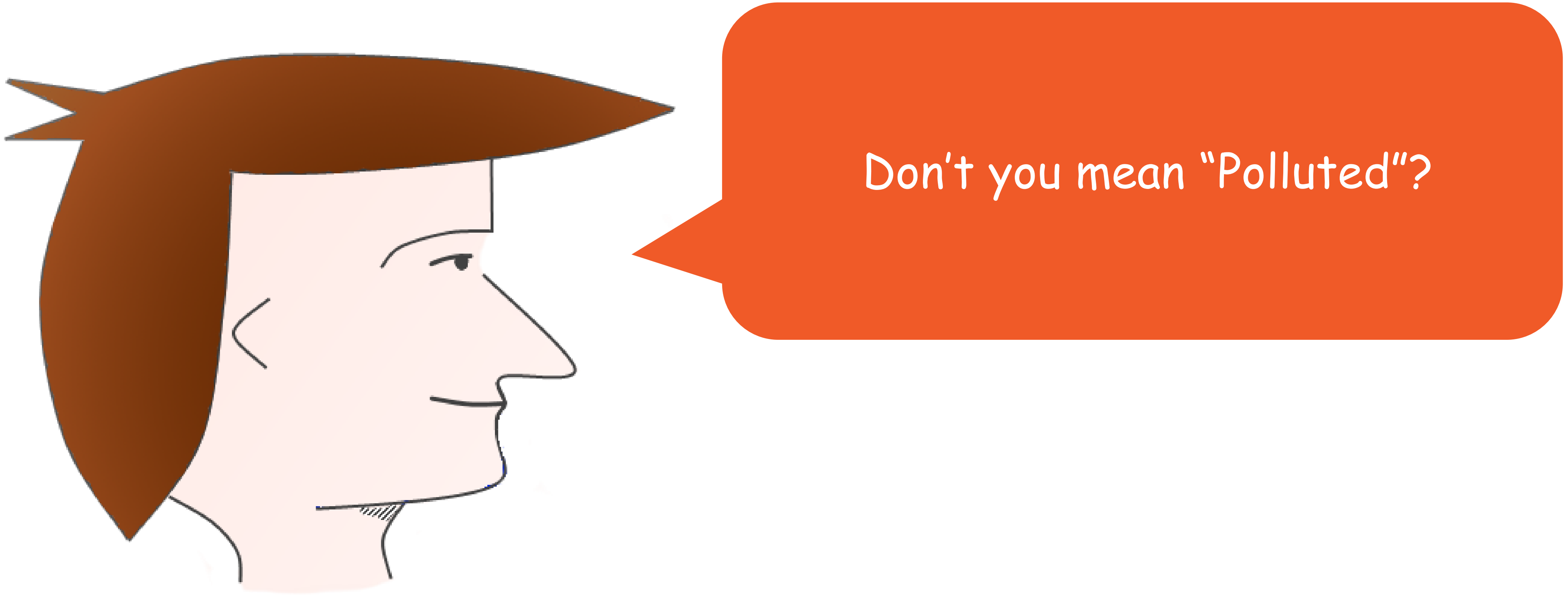
Bringing it all together

Complexity

Simple projects grow into complex projects

Change request may affect architecture

Cleaner UI


Don't you mean "Polluted"?

jQWidgets
better web, less time

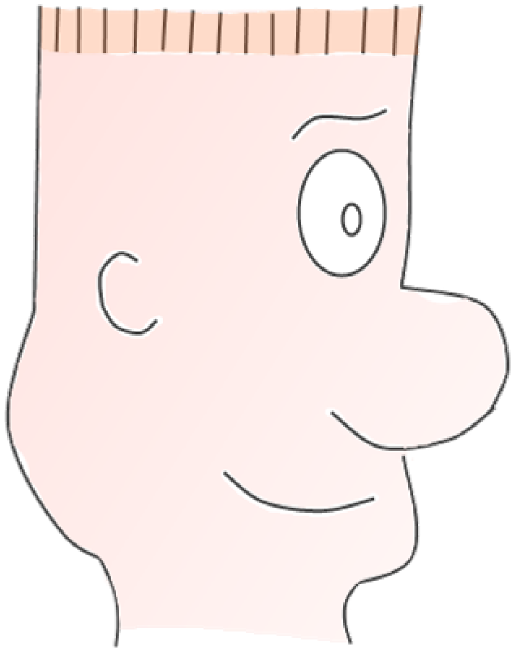
http://www.jqwidgets.com

- Creative Commons Attribution – Non Commercial 3.0 License

Indicative of any UI library

Using list menu

Better user interaction



Why didn’t you include jQWidgets in the dependency list for the callback function?

Coding for Efficiency

Responsive start page loading

jQWidgets is not the focus

How to hide HTML list?

- Hide with CSS
- Show with jQuery

Eliminate “decoration flicker”

Eliminate Decoration Flicker

Use jQuery to remove “hidden” class

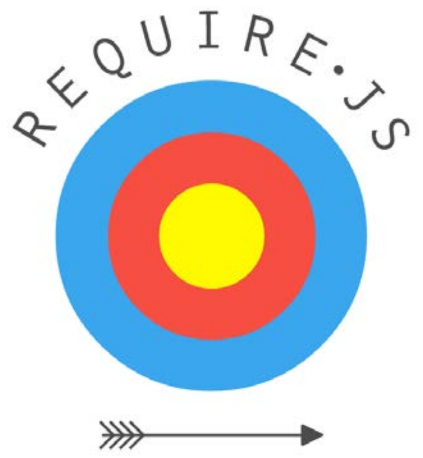
Wait until jQWidgets has decorated the list menu

Integrated non-modular library

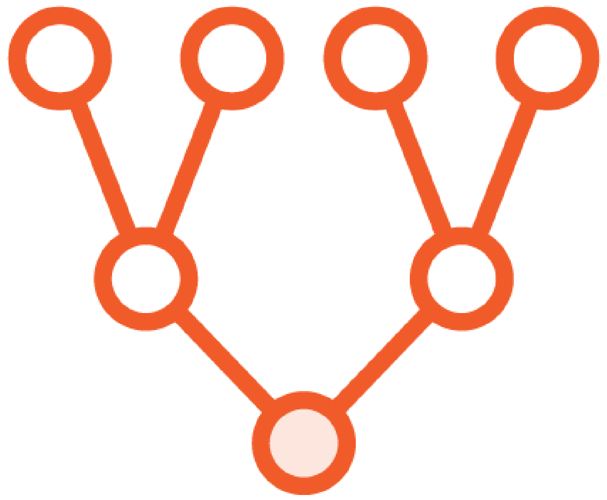
Decorated HTML content

Coded for asynchronous file loading

Loading Legacy Scripts



Loading order not guaranteed



Dependencies must be considered

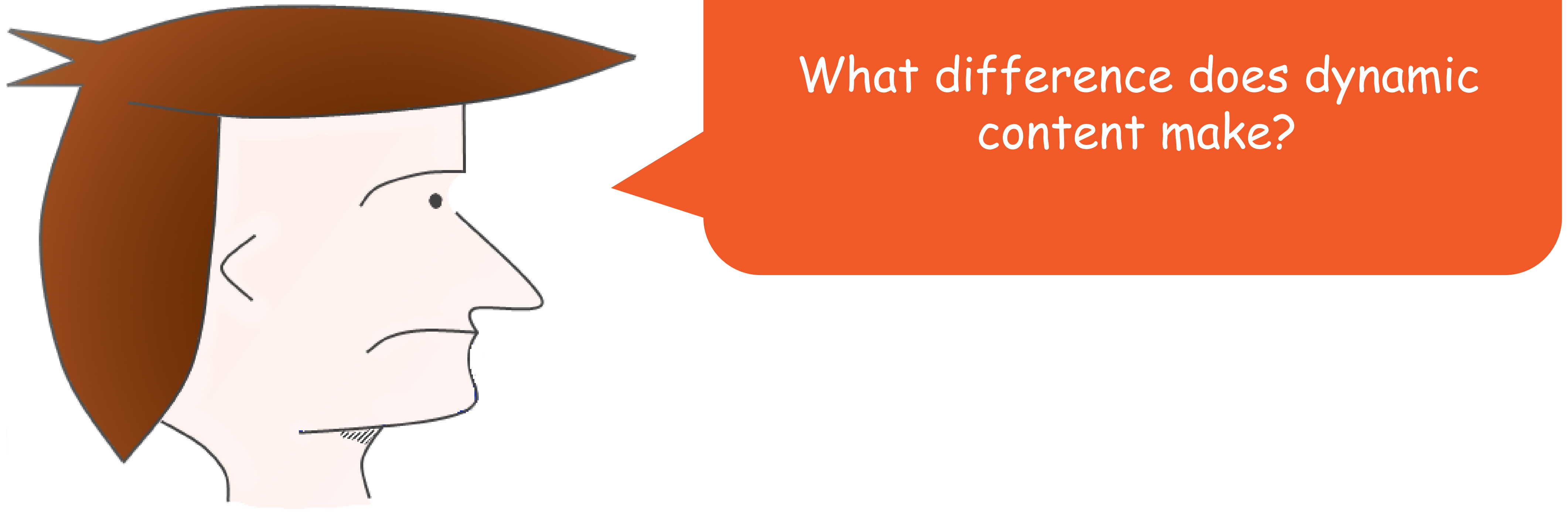
Dependent Code in Callback Function

```
require(['jqx'], function()
{  $('ul').attr('data-role', 'listmenu');
   $('ul:first') .jqxListMenu(
   {  theme: 'energyblue',
      enableScrolling: false
   })
   .removeClass('hidden');
});
```

Dynamic Content

More than static HTML

Decorating asynchronous content

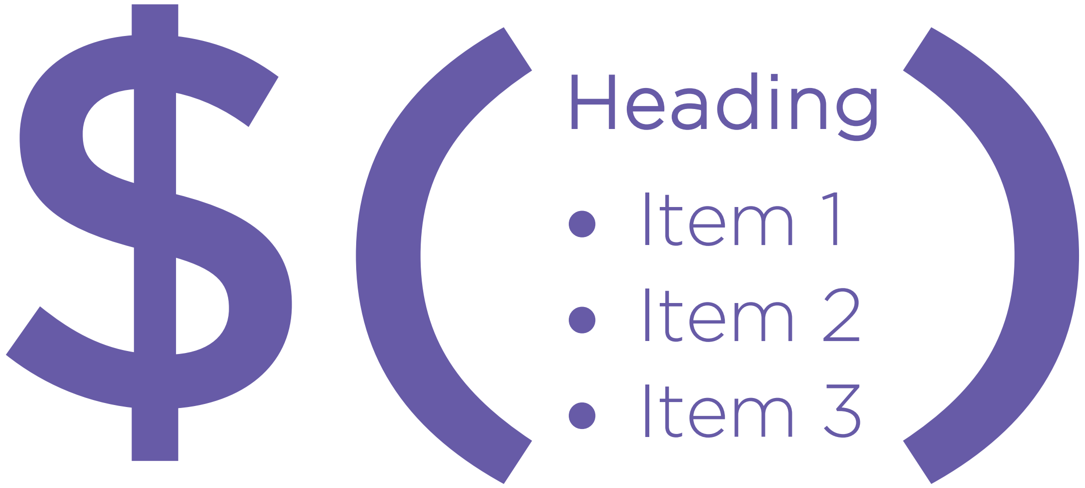

What difference does dynamic content make?

Load contents from data

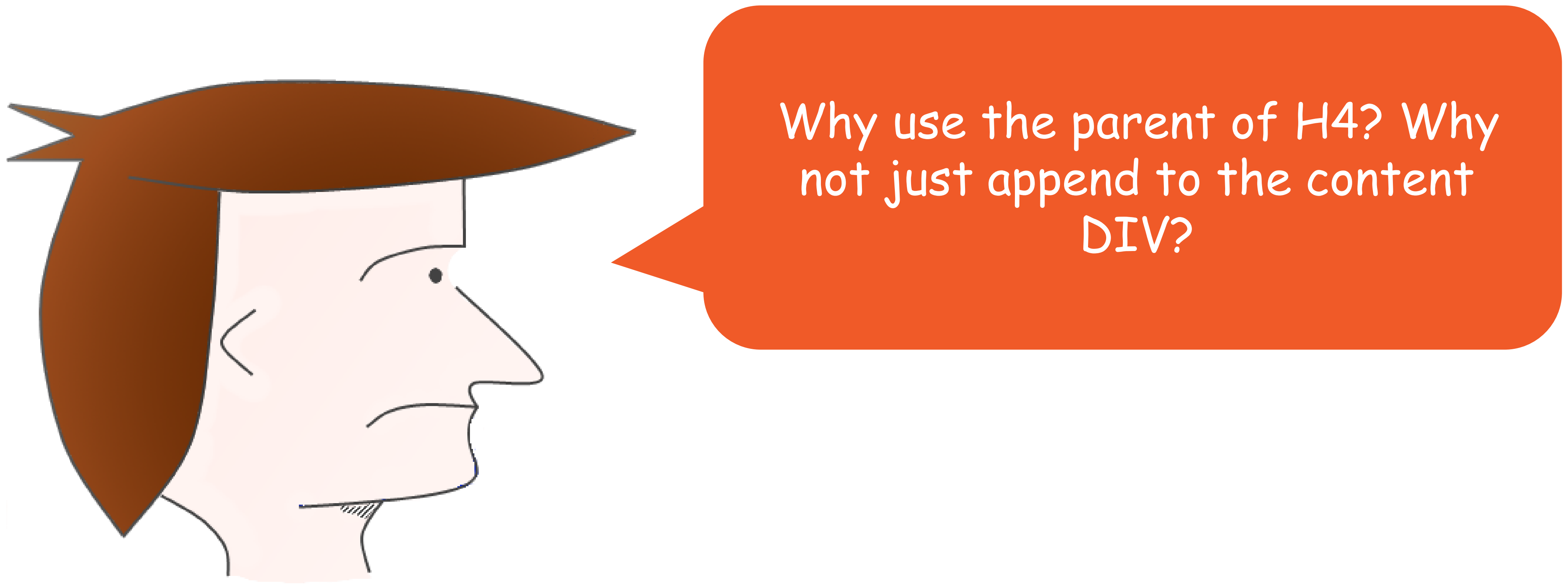
New data proxy module

- Approximates web service
- Not discussed deeply
- Comments in exercise files
- Provides interface for data access

"loadAsObject" Method



Returns unordered list as jQuery object


Why use the parent of H4? Why not just append to the content DIV?

Toolbar Interaction

Integrate a click event

Item Not Decorated

New item added by external process

List menu should decorate when new data arrives

Add code to click event handler

Mixing UI and Data Coding

Focused on working code

Mixing Code

```
$('.publish').one('click', function()
{  $('ul:first')
      .append('<li>User added item via click<li>')
      .jqxListMenu('refresh');
});
```

One of the primary purposes of modular programming is reusability

Modular Reusability

Other pages could benefit

Separating code into chunks

Using Configuration Module

Event listener context can be changed easily

Disclaimer

Simplistic example

Validation required

Events

Overcome asynchronous timing issues

Transfer data between modules

Decouple modules

Code Location

All page-related code in one file

Complex pages contain more code

Web Page

Wraps numerous capabilities in one place

Page functions can be unrelated to each other

Separating Code

Chunks of related logic

Refactor existing project

Another Look

Module Parameters

Can parameters be passed during module creation?

Currently, no support for parameters during load

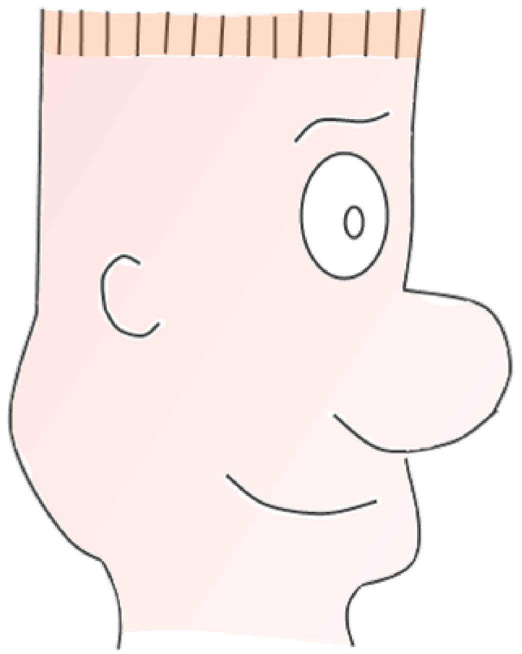
There are other options

Footer Module as Example

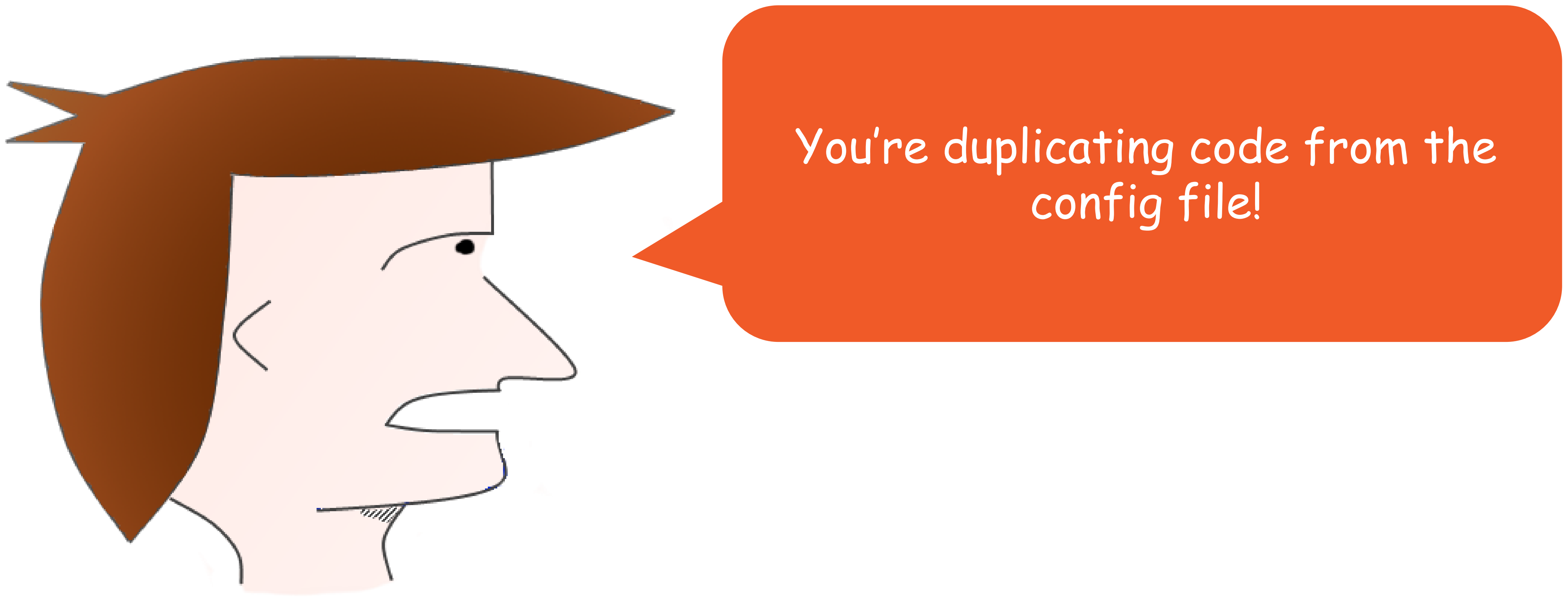
Refactor to allow options

Literal Values

Where should literal values be maintained?



The config module can hold literal values


You're duplicating code from the config file!

Cases for Default Values

Improperly built dynamic config file

Incompatible future config file version

Config file refactored or split into segments

Defensive Programming

Protect from future changes

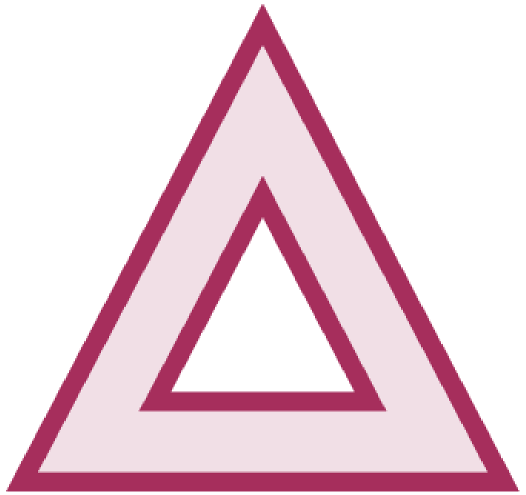
Potential Downside

Behavior changes may not be immediately noticed

Few people actually read footers

Could take months for someone to notice

Same Name – Different Function



“init” function in module is different than “init” function in configuration

Sample Footer Use

```
require(['KSM_FooterAMD'], function(footer)
{   footer.init(
    {   footerText: Footer from code',
        footerTag: '<div>',
        insertAfter: '.bottomContent'
    });
});
```

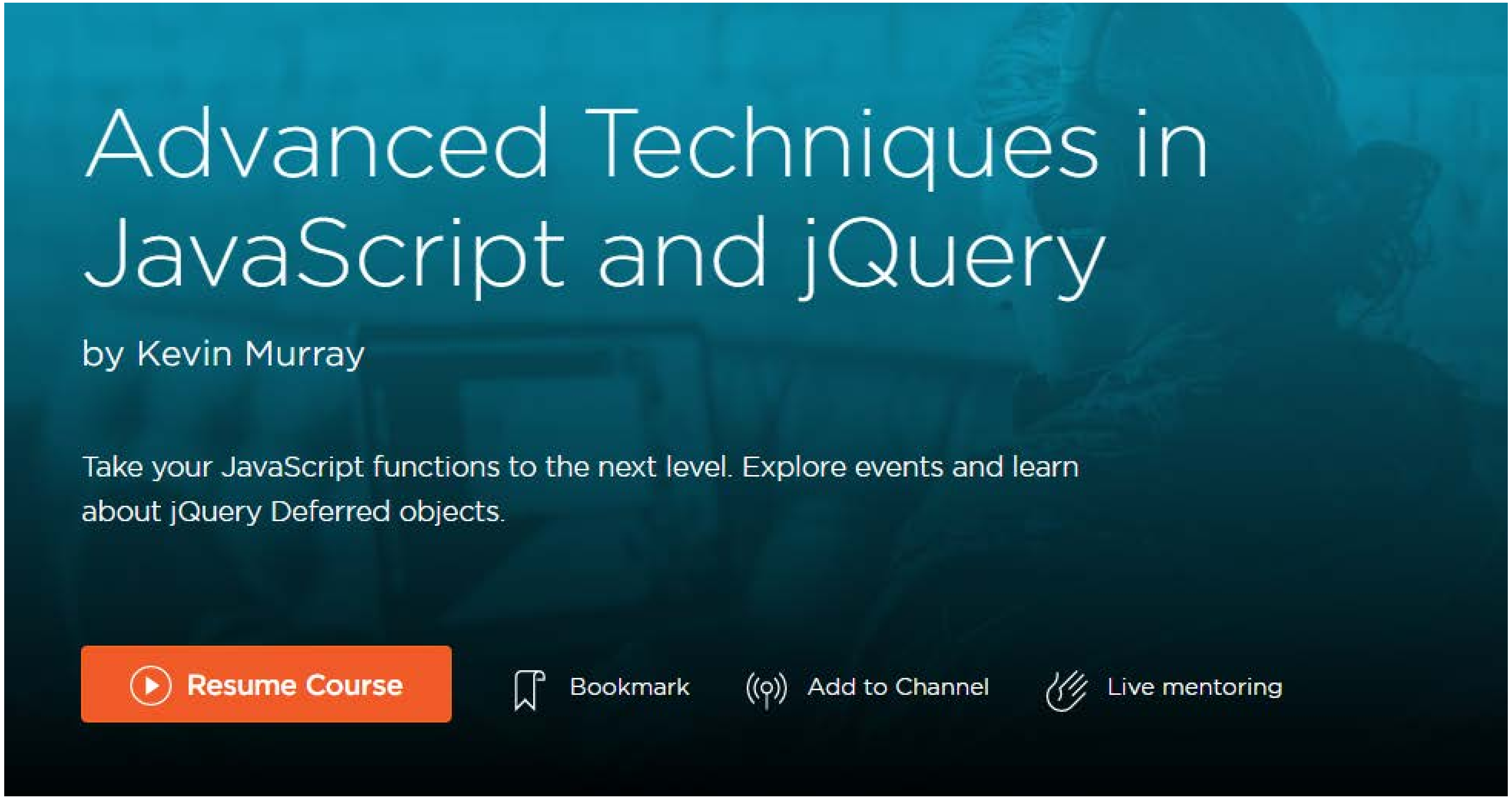
One object parameter

Positional parameters

- Difficult to omit or skip
- Order and meaning must be known

Object parameters can change without breaking legacy code

Additional Learning Opportunity



Advanced Techniques in JavaScript and jQuery

by Kevin Murray

Take your JavaScript functions to the next level. Explore events and learn about jQuery Deferred objects.

Storing Values in Object

Not really required by the footer module

Useful to have initialization parameters stored

Getter/Setter methods could be added if necessary

Footer Load

RequireJS only loaded footer

No need for a callback function

Now, reference to footer is desired

We could have just changed the original footer module to get that new text!

Footer Question

What happens to legacy code?

Legacy Code

Older projects don’t use the “init” function

Can the new module work with legacy and new code?

Modification required for legacy support

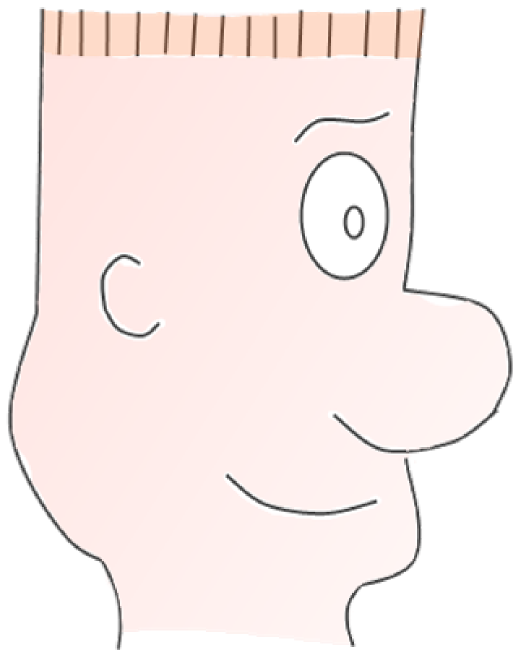
Simple Solution

Call the “init” function before returning the footer object

Legacy code shouldn’t be affected

Another Footer Question

What happens to new code?



Why does it matter?

Supporting Legacy and New

Legacy code doesn’t call “init” function – no default footer present

New code calls “init” function after default footer has been added – removal necessary

Footer Module Evolution

Maintains local properties

Contains an initialization method

Changes behavior based on configuration data

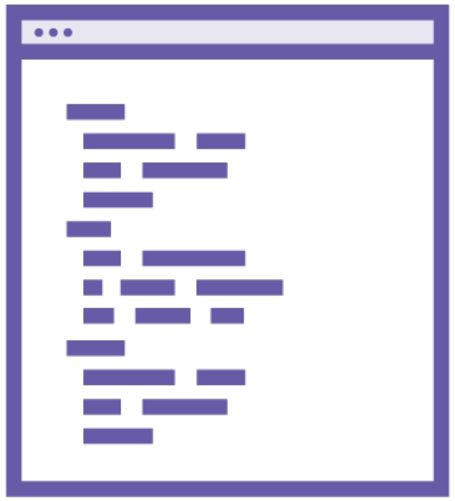
Footer Module Evolution

ia

rts legacy
code

The footer module has really grown up!

Review if Necessary



Rewind course or examine code files

Final Project

Building a modular project template

Presentation will appear the same

Code Evolution

eat
ve

d all code
in startup
file?

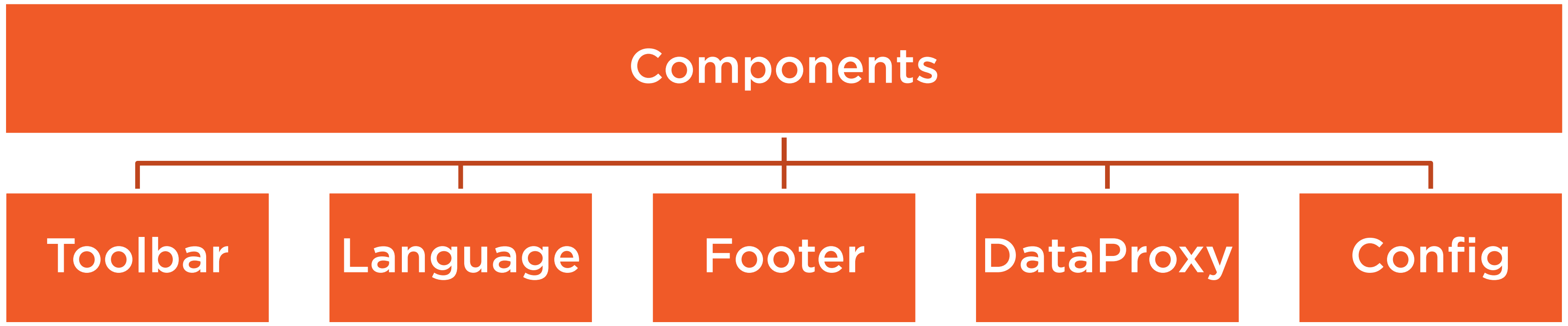
Few capabilities

- Toolbar
- List menu
- Simple footer

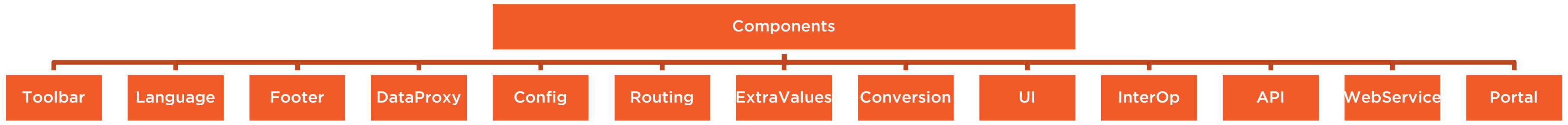
Started small and grew

Modular programming should help

Current Modules



Future Modules



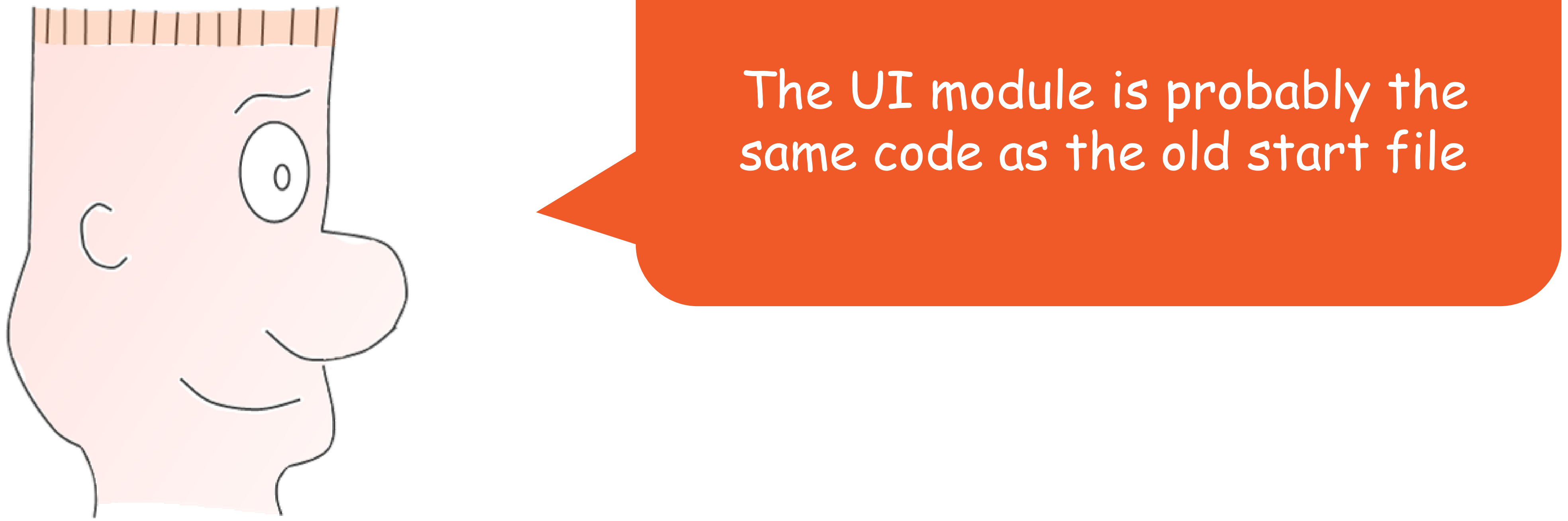
Other Projects

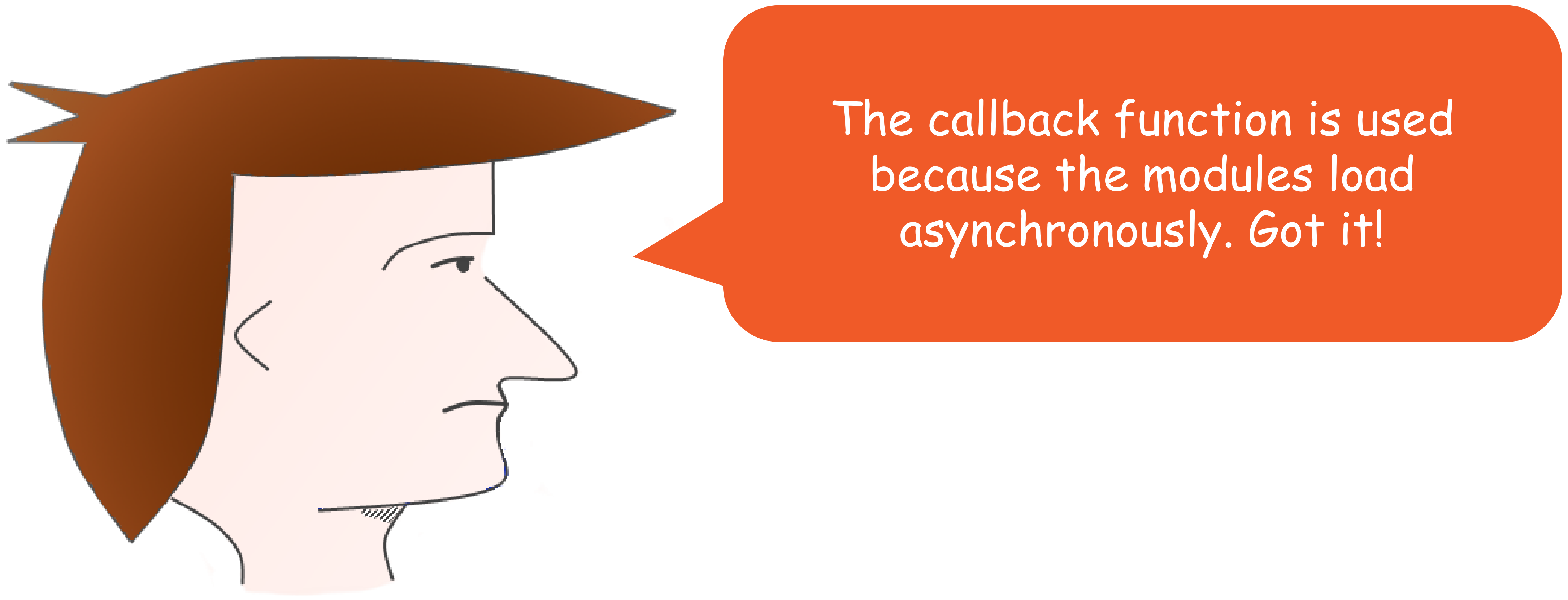
```
require([],function()
{   require(['KSM_UI']);
    require(['KSM_Security']);
    require(['KSM_Database']);
    require(['KSM_LocalStorage']);
    require([‘KSM_Miscellaneous']);
});
```

Initializing Additional Modules

Single line of code for each module

KSM_UI Module Contents


The UI module is probably the same code as the old start file


The callback function is used because the modules load asynchronously. Got it!

Encapsulate dependent logic in a callback function

Loading Data

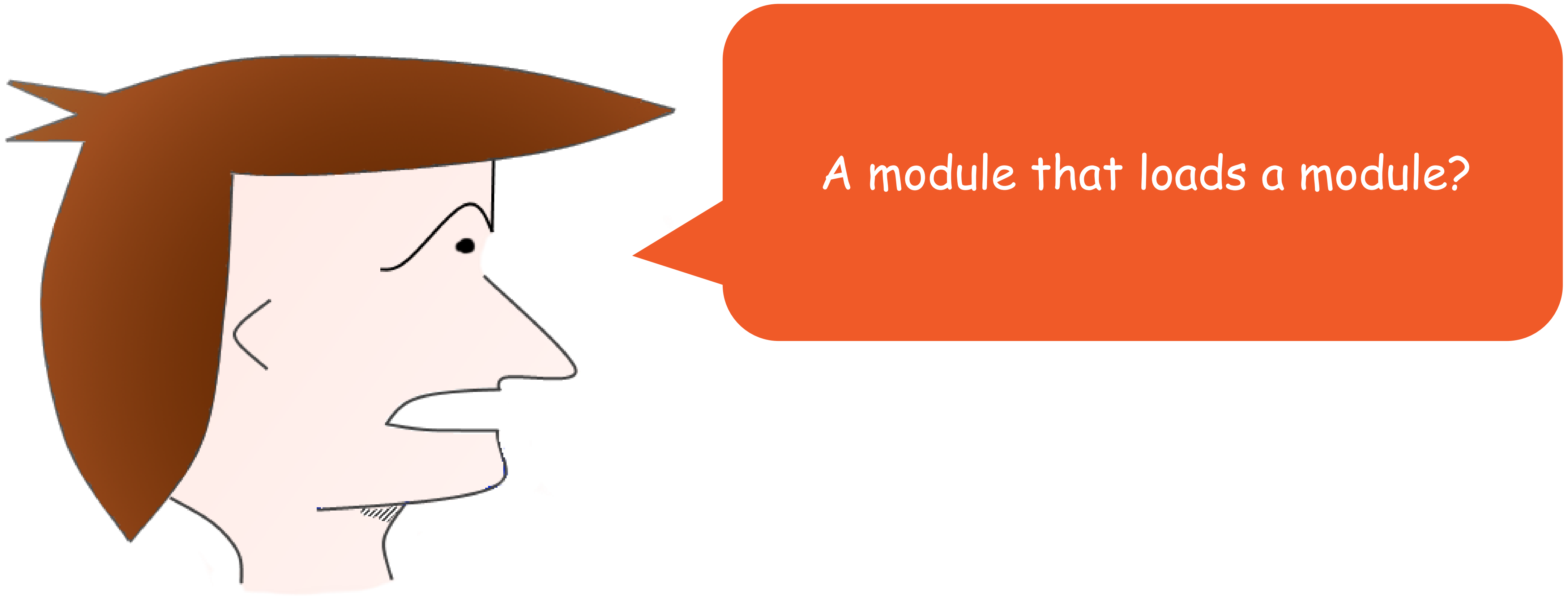
ɔc
ıe

ıtes a web
ıice call

Data Loader Module

New module for final project

Provides interface to “DataProxy” module


A module that loads a module?

Previously used “DataProxy” within startup code

Performed data and UI processing

Established a tight coupling

“DataLoader” provides a loose coupling

Benefits of Event Processing

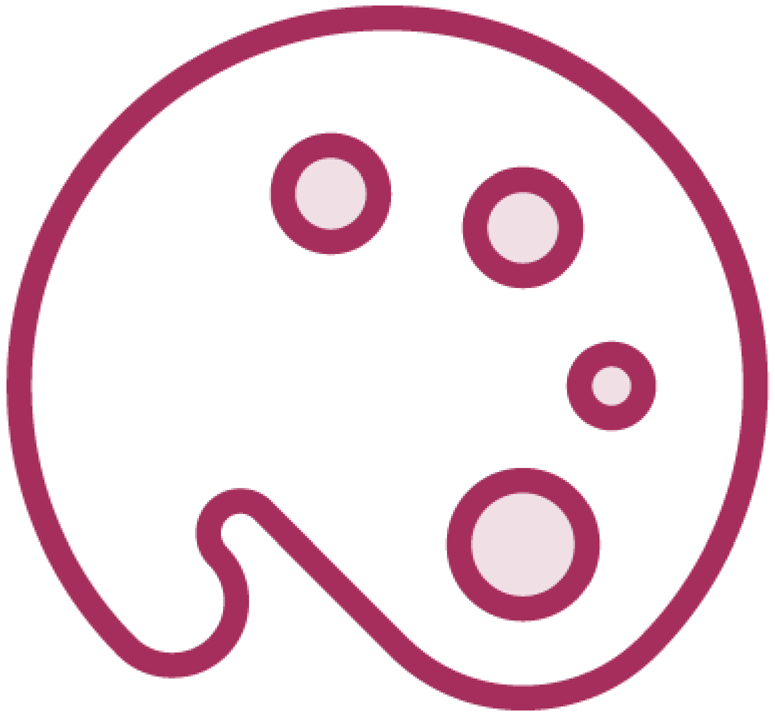
Isolates data retrieval from user interface logic

User interface requests and receives data via events

No tight coupling between user interface and data modules

UI Decorations Module

s" refers to the
n of plain HTML
ng a UI library



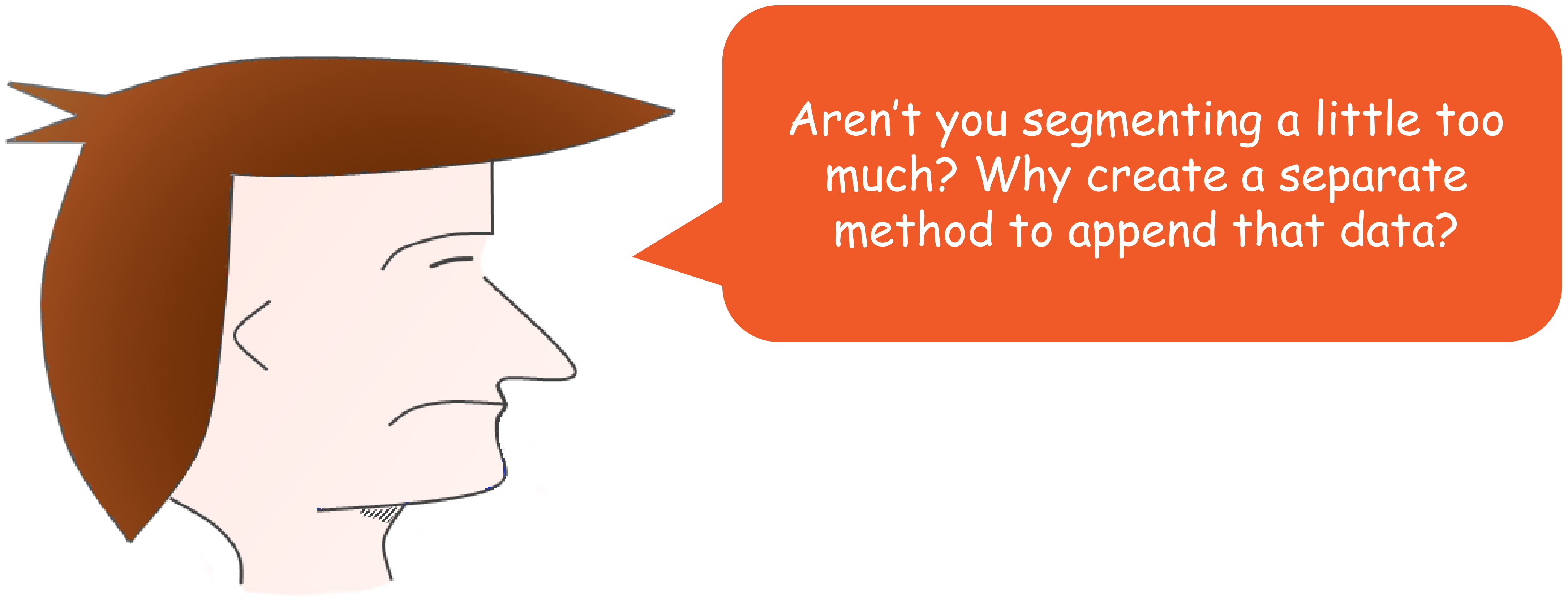
Decorations module

- Allows for alternate UI library
- Groups decoration changes in one place
- Provides alternative for mobile version

Easily excluded during mobile presentation

Avoids tight coupling to specific UI library

Proper modular programming allows for decoupled connections


Aren't you segmenting a little too much? Why create a separate method to append that data?

“appendRefresh” Function

Externally available method for adding a list menu item

Can add a list menu item that is not supplied by the data module

Click listener could be in UI module

System events

- SignalR

File, memory, and storage events

All listeners reside in one module

Personal Choice

Listeners are grouped based on previous experiences

Organize code to be easily: maintained, trained, and explained

Who cares about future maintenance as long as the code works today?

Referencing configuration file

- Event types
- Footer options
- Various literal values

Change behavior by changing an option

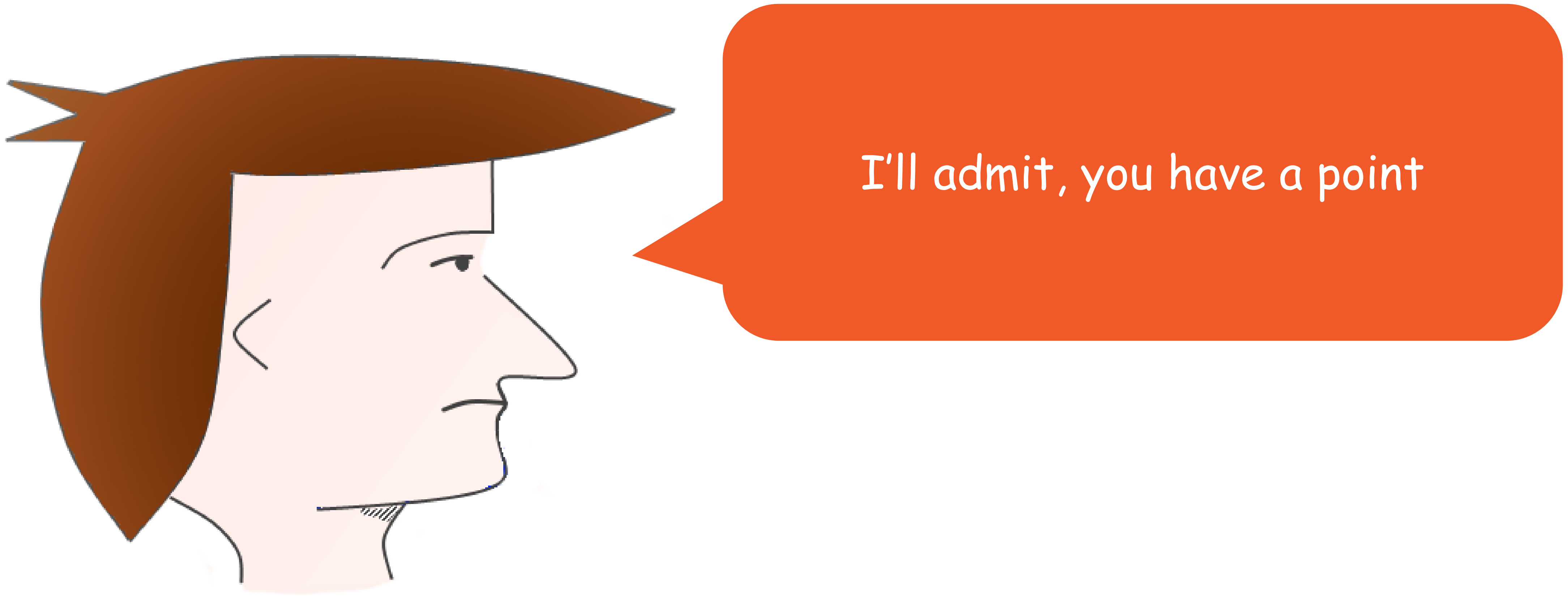
Better than changing code

Validation of Changes

Unit testing sufficient for configuration changes

Regression testing is best when actual code changes

Any code change can introduce unexpected behaviors


I'll admit, you have a point

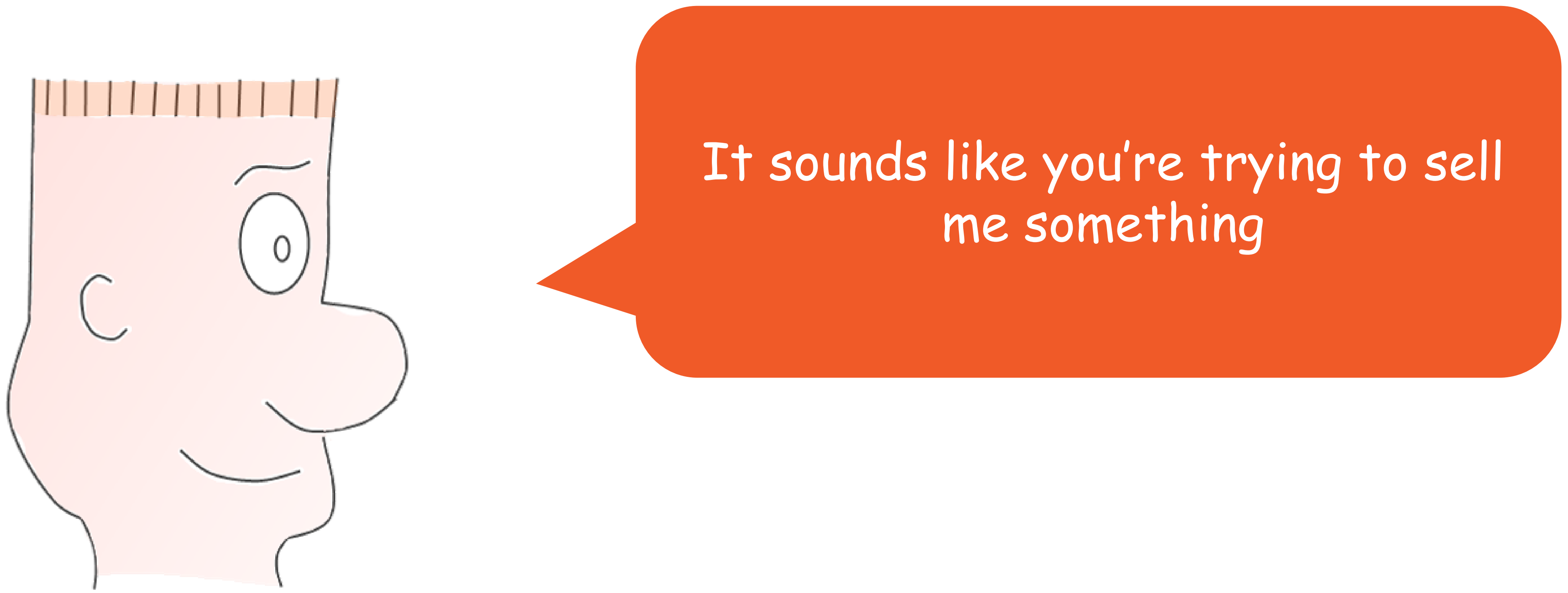
Benefits of Current Design

Modules are decoupled through use of events

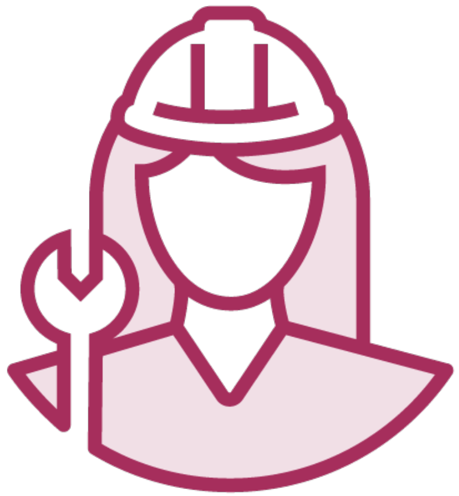
Architecture ready for growth and porting to mobile

Established event rules allow for numerous team members

Minimal effort for maximum benefit


It sounds like you're trying to sell me something

Revisiting “appendAndRefresh”



I thought all events are defined in the listeners module!

Module Dependencies

Modules can easily reference each other

Just because you can – doesn’t mean you should

Consider impact of coupled modules

Decoupled Data Access

Access to “DataLoader” through “Decorations”

No coupling to “DataLoader” or “DataProxy” modules

Button click starts event processing to add new item

Final Review

Final Review

Toolbar module integrated with language module

Footer module initialization function

List items loaded from a decoupled data module

Final Review

User interface decorated with a dedicated module

Third party user interface library isolated from code

User interface handles asynchronous data access

Summary

Use sample code for additional understanding

Integrated user interface library

Provided dynamic content

New “init” modular pattern

Decoupled modules

Write code that is easily maintained, trained, and explained