MacTech 2016 / Per Olofsson

Build your first Mac app with Xcode

Per Olofsson

IT Infrastructure Specialist (aka sysadmin)
~~Research engineer~~ at the University of Gothenburg

37000 students

6000 faculty + staff

I'm Per Olofsson, and I work as a sysadmin at the University of Gothenburg, Sweden, where my team manages a fleet of about 2000 macs and maybe twice as many PCs. We're Sweden's second largest university, we have about 37000 students, and about 6000 faculty and staff that we support.

I also do a bit of open source development, and when Ed talked me into presenting here at MacTech, we went back and forth a bit about what I should talk about, but eventually I decided that I wanted to talk to you about developing apps.

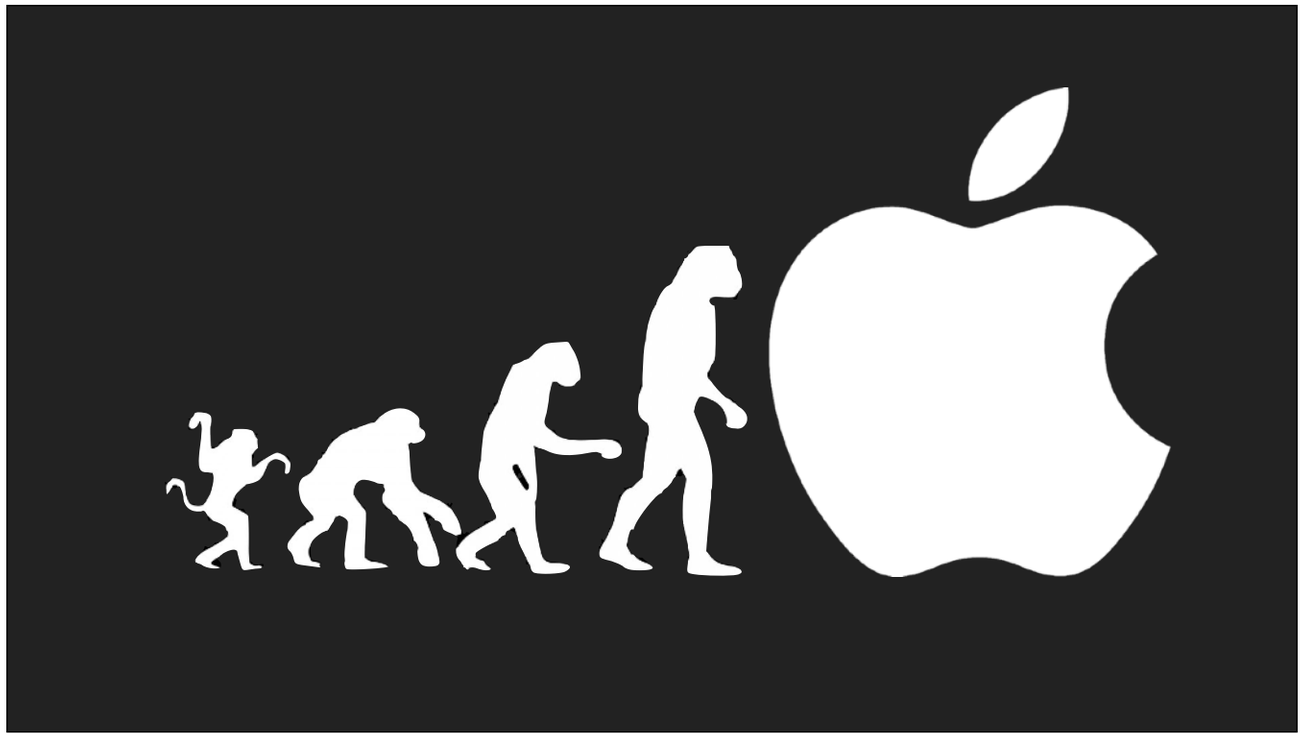
I've worked my entire career as a sysadmin, in various roles and in different places, and software development has never been part of my job description. I've always loved programming though, ever since I got my first computer almost 30 years ago.

From time to time though I've managed to sneak in a development project into my day job, some small, some large, some under the radar, some sanctioned. In most of my day to day though I write shell scripts, or small Python scripts.

Automation

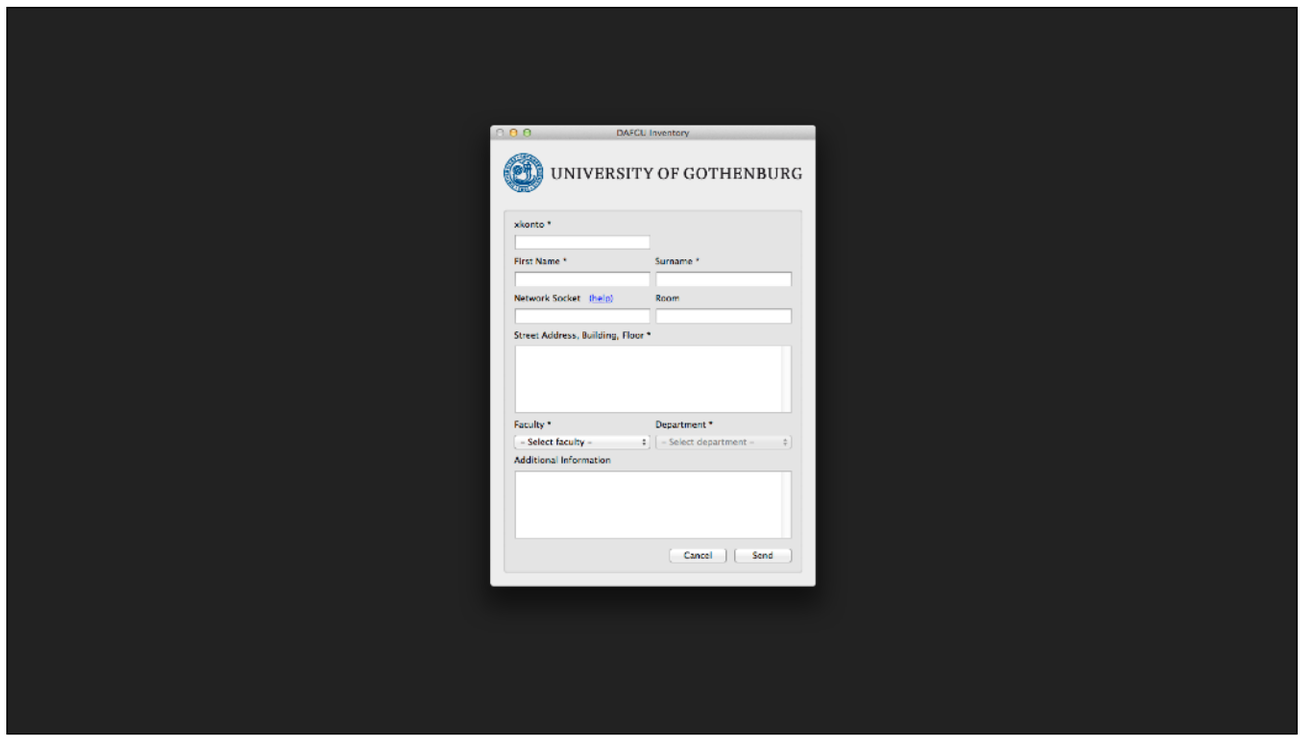
And automation is really key to what we do as sysadmins. It enables us to efficiently complete tasks that would take forever to do manually. With a single script you can solve a problem that affects the whole fleet, or even better you might able to pre-empt a problem and solve it before anyone even notices. And if you do your job right, no one should notice, right?

So we automate and fix things in our own organizations, but some problems don't lend them selves to silent automation. Maybe their machines aren't under your management, or maybe there needs to be more of a dialog with the user, they have to give their input for you to successfully solve the problem.

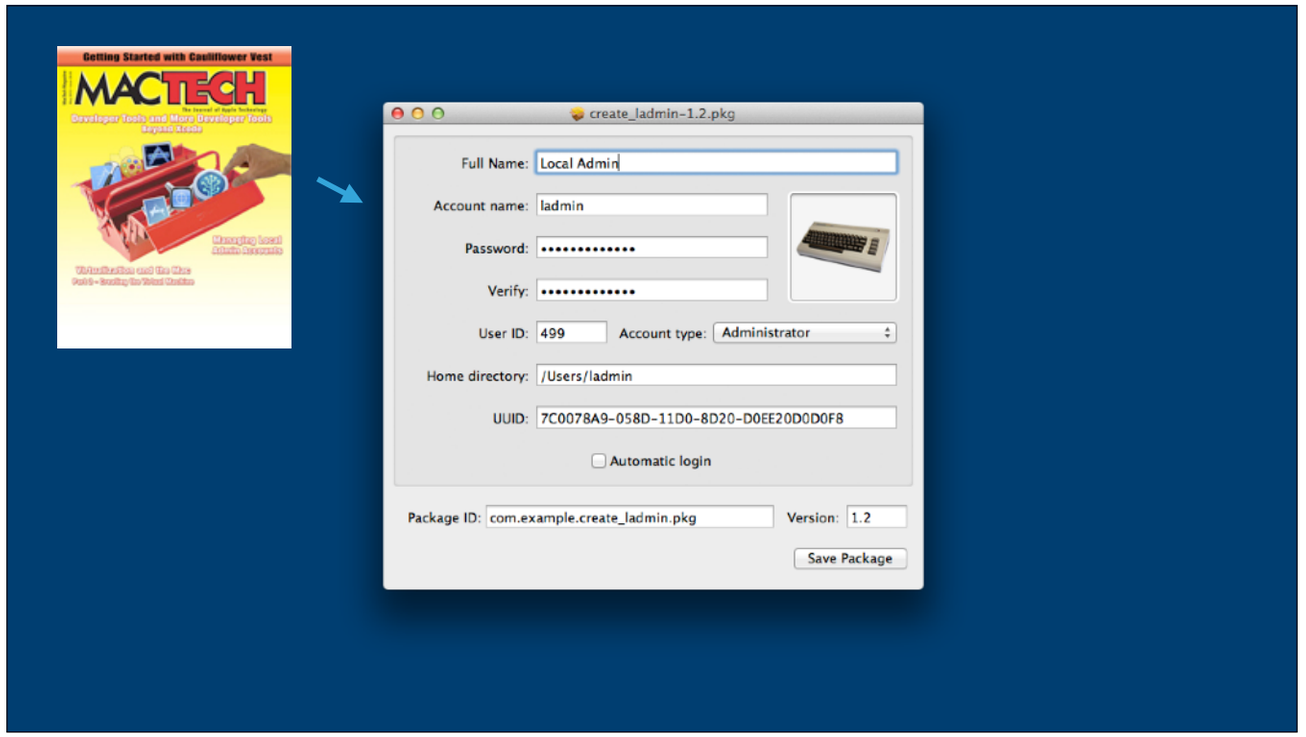


This is how my path into developing apps on the mac started. We were just about to launch our centrally managed Mac platform, and we needed some way of reaching all the mac users in our organization.

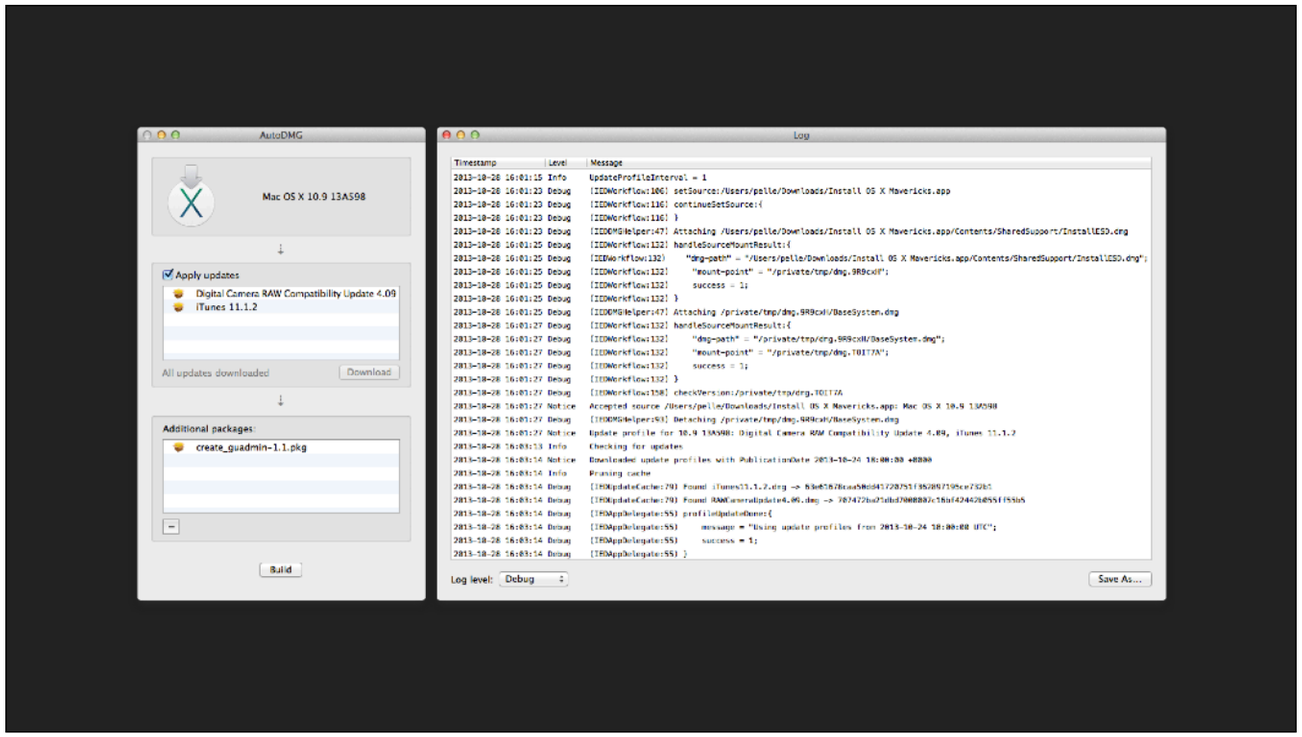
It should be noted that the University of Gothenburg is really spread out, there's no central campus, and we're in hundreds of buildings both in the city, and even outside of it, so just finding people can be a challenge sometimes. If they're even in their office.



So my first Cocoa app was a simple form that asked the user a few key questions, like their name and location, but really the core of the app was a bunch of Python scripts that gathered some inventory data and uploaded it to our servers, and then installed the Munki agent. And had we had enough techs, and knew where to send them, we could probably just have walked around with a USB stick and done the install, but this way we could just post the app, send an email everyone, and ask them to run it.

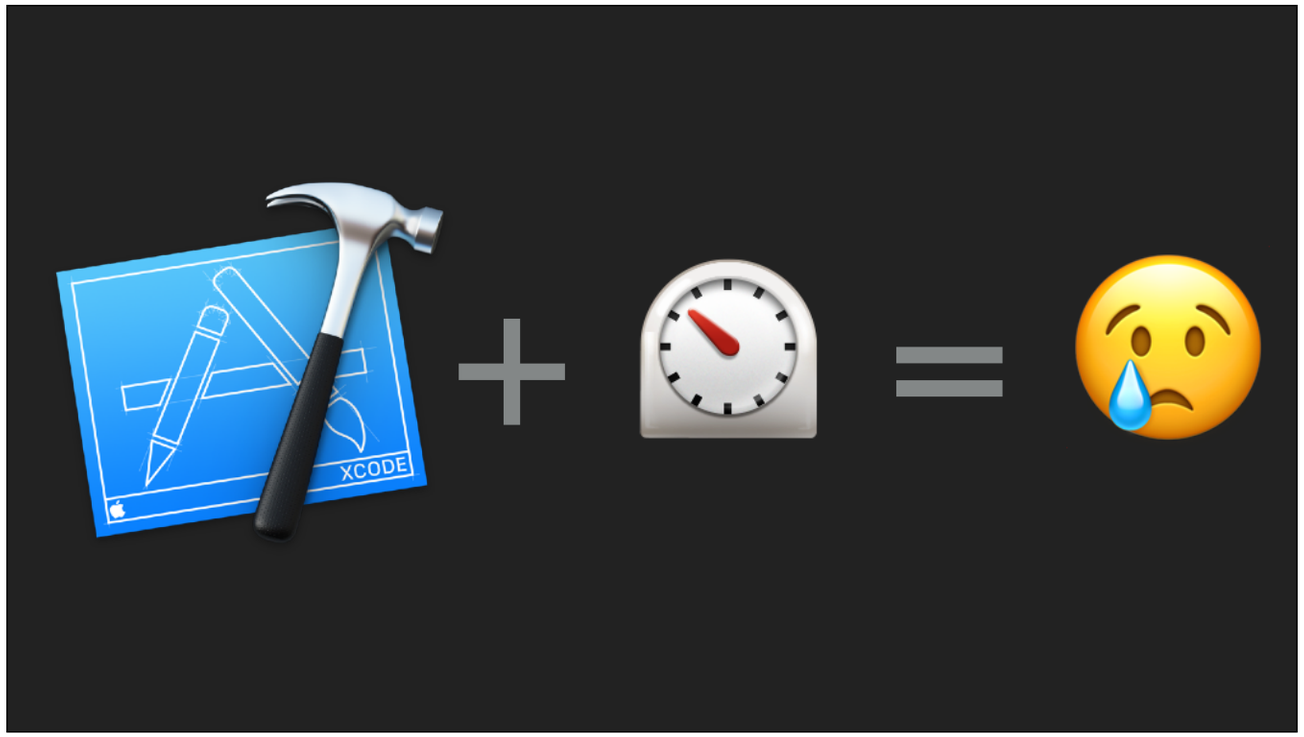


And of course, once you've acquired a new skill, you suddenly find lots of opportunities to apply it. Soon after our migration project Greg Neagle wrote an article in MacTech magazine on how to manage local admin accounts, and I took his scripts, put a GUI on it, and released it in the App Store as CreateUserPkg.

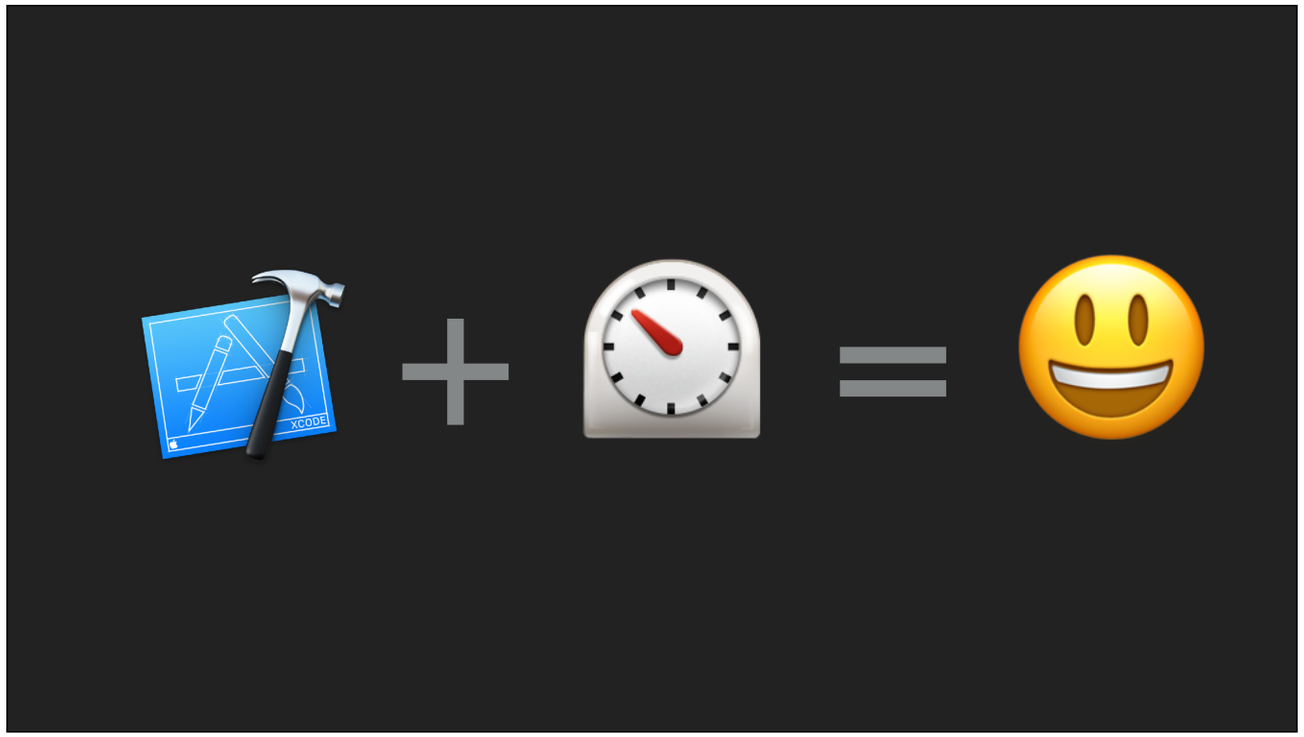


A year later InstaDMG, after long and distinguished service, went into retirement. Duncan McCracken took the essence of InstaDMG and distilled that into 10 lines of bash, which was the spark that ignited AutoDMG. As you can see, there's a pattern here: we have a problem, here's a script that solves it, but when you put an approachable user interface on it, it becomes useful to a lot more people.

So this is why I want to talk to you today about app development. Apps allow you to take a problem, encapsulate your solution to that problem, which then in turn empowers your users to solve that problem. I'm hoping to show you today, that if you're comfortable scripting and automating things, how you can take that to the next level and make user friendly applications out of those scripts.



So I set out, in preparation for this talk, and started building a demo project that demonstrated a few key concepts, I did some intro slides, and then I did a test run with my partner at home, and at this point I realized that I was completely crazy and how on earth did I think that you could teach someone app development in less an hour. It's a really complex subject.



So out of necessity today I'm going to have to gloss over a few things, and we won't have time to create an app that does something particularly useful, but we can do something fun while still demonstrating those key concepts that you do need to learn.

I'm also not going to have time to teach you programming, for obvious reasons, but I suspect that most of us here are already comfortable in at least one language.

Show of hands, bash & applescript? Python, Ruby, Perl, PHP? Swift or Objective-C? C#, Java, C, Go?

Let's build an app

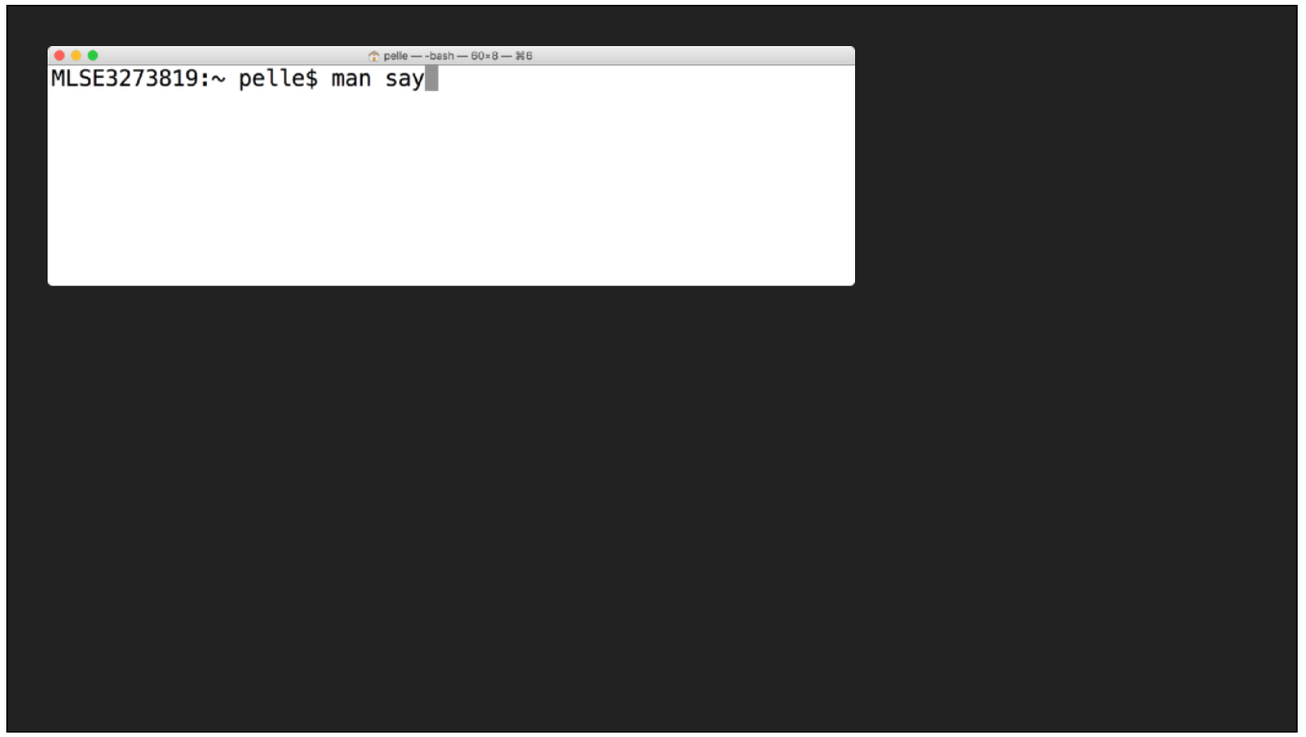
OK, having said that, let's build an app.

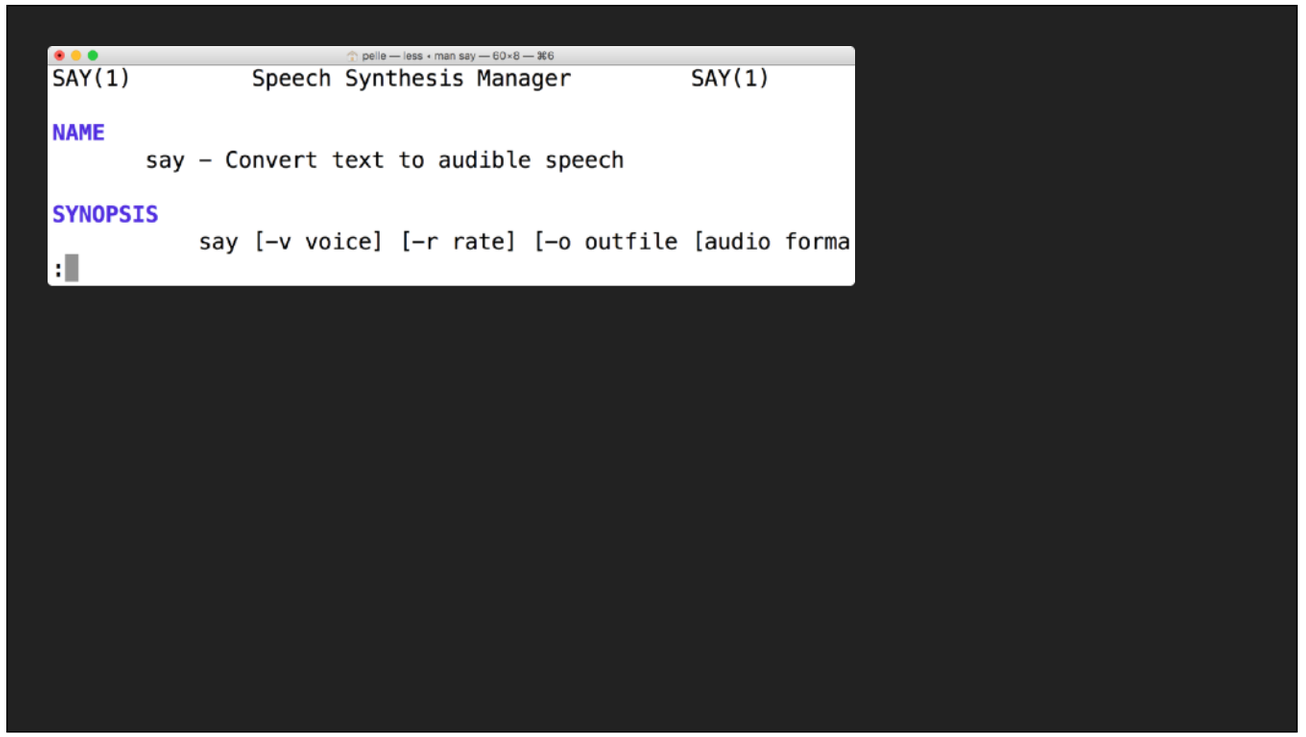
Proof of concept

The first thing I’d want to start with is a prototype, a proof of concept.

Are you familiar with the say command in macOS? Let's open the Terminal.

Check the man page for the say command.


pelle — -bash — 60×8 — ⌘6
MLSE3273819:~ pelle$ man say



So you feed it a line of text and it speaks it for you. Hours of fun. Let's try that out.

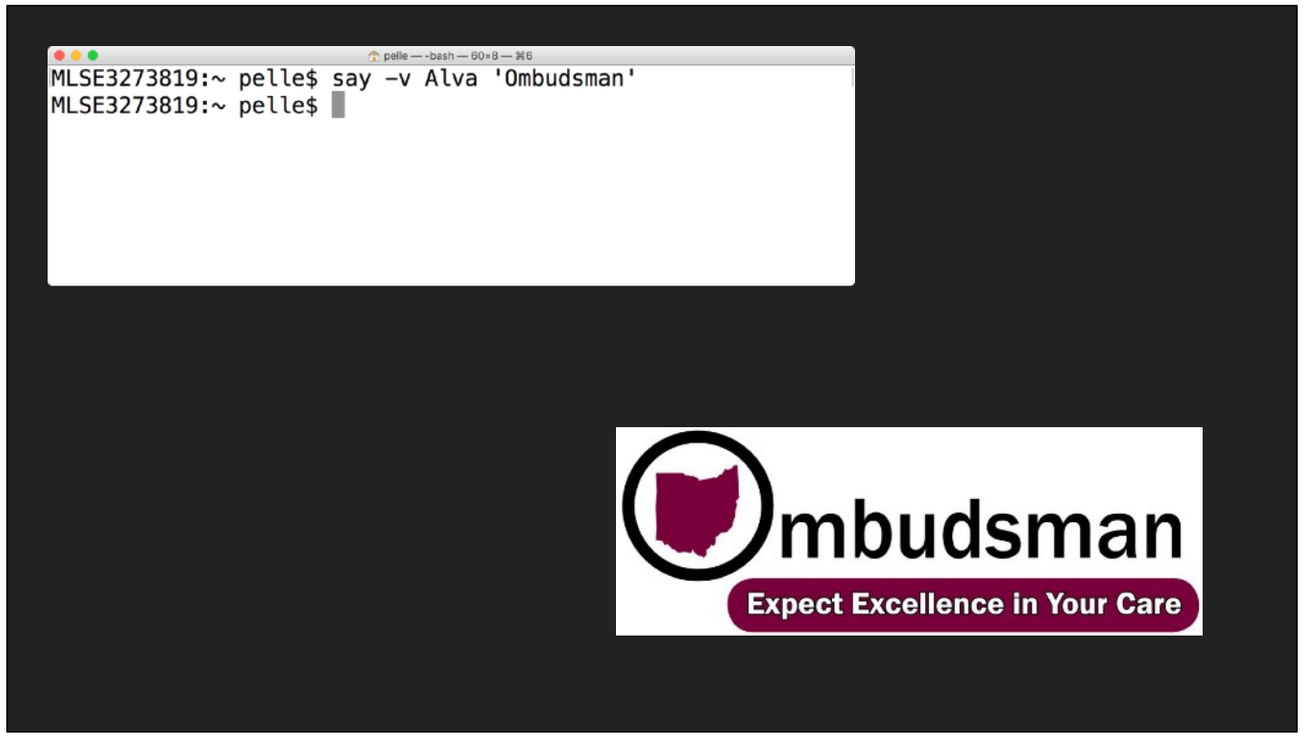

pelle — -bash — 60×8 — ⌘6
MLSE3273819:~ pelle$

OK! But have you ever wondered about those pesky Swedish words in your vocabulary, how they're really supposed to be pronounced? Let's choose a Swedish voice and find out.

```
pelle — -bash — 60×8 — ⌘6
MLSE3273819:~ pelle$ say -v Alva 'Smörgåsbord'
MLSE3273819:~ pelle$
```

```
pelle — -bash — 60×8 — ⌘6
MLSE3273819:~ pelle$ say -v Alva 'Gravlax'
MLSE3273819:~ pelle$
```

```
pelle — -bash — 60×8 — ⌘6
MLSE3273819:~ pelle$ say -v Alva 'Husqvarna'
MLSE3273819:~ pelle$
```



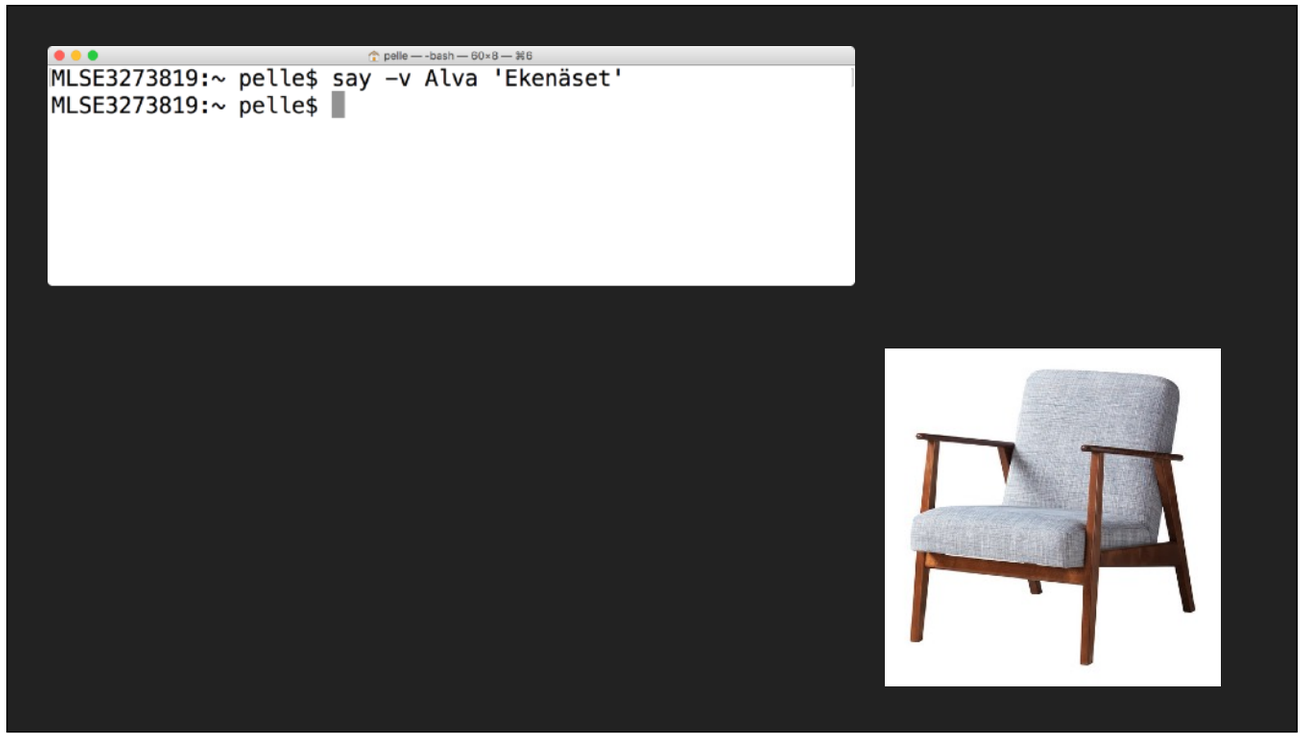
OK, so now we know!

And have you ever gone to IKEA and wondered how the heck you're supposed to pronounce all these names?

Well, I can't get the dang thing pronounce IKEA properly. But let's try some product names.

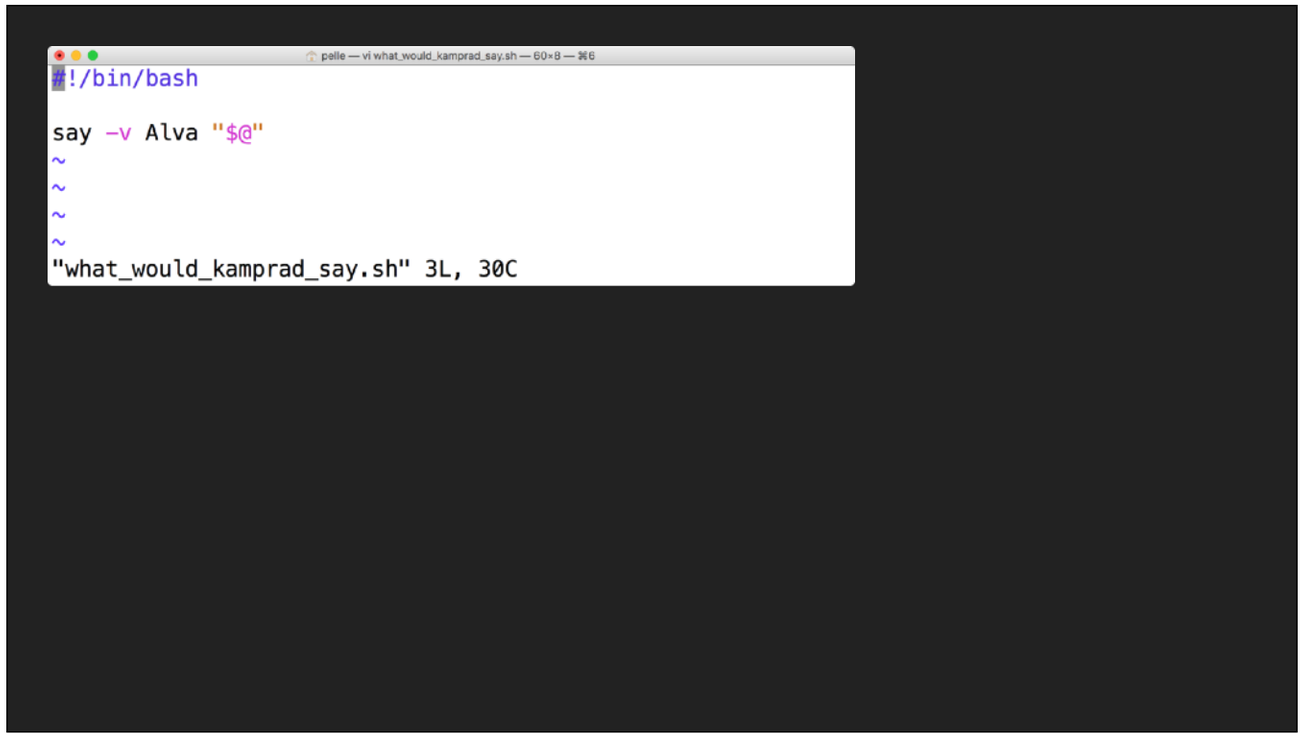
OK, you probably could have figured that one out. Let's try a trickier one.


pelle — -bash — 60×8 — ⌘6
MLSE3273819:~ pelle$ say -v Alva 'Poäng'
MLSE3273819:~ pelle$


pelle — -bash — 60×8 — ⌘6
MLSE3273819:~ pelle$ say -v Alva 'Ekenäset'
MLSE3273819:~ pelle$

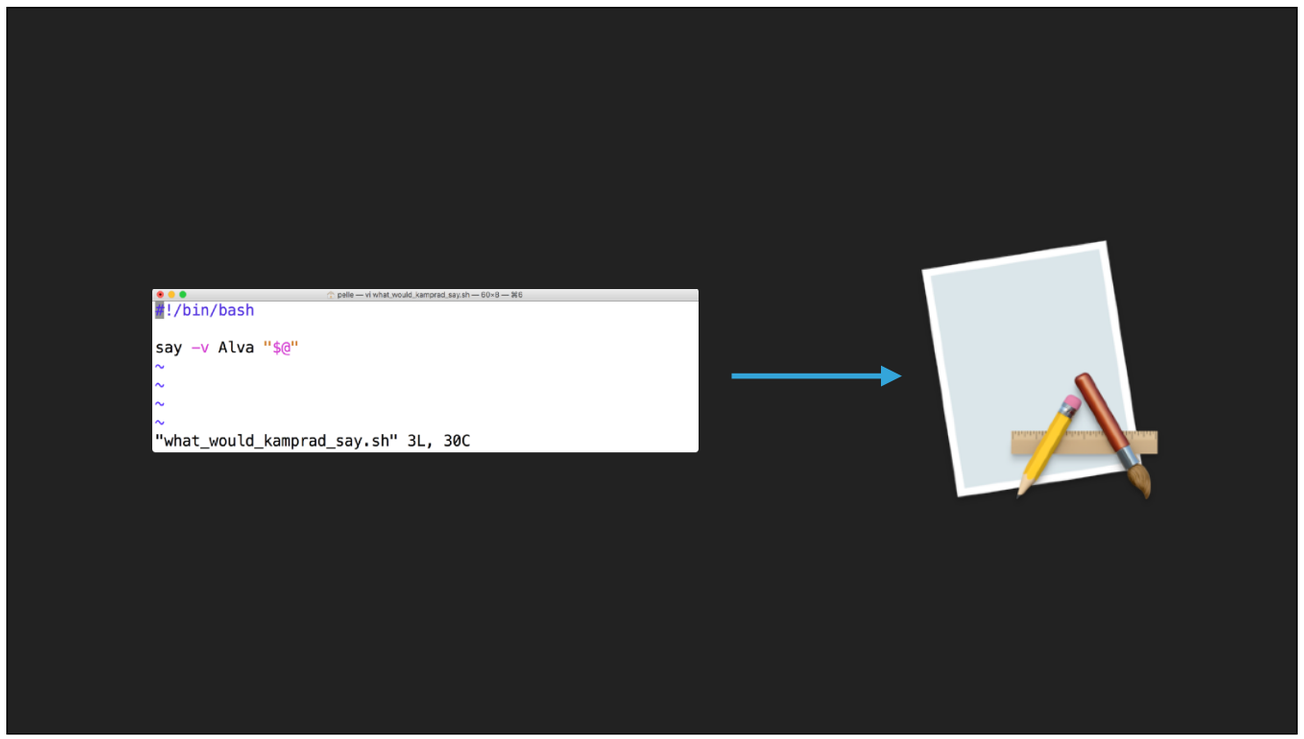
OK, now you no longer have to be embarrassed and confused while for shopping furniture.

So let's put that in a script.

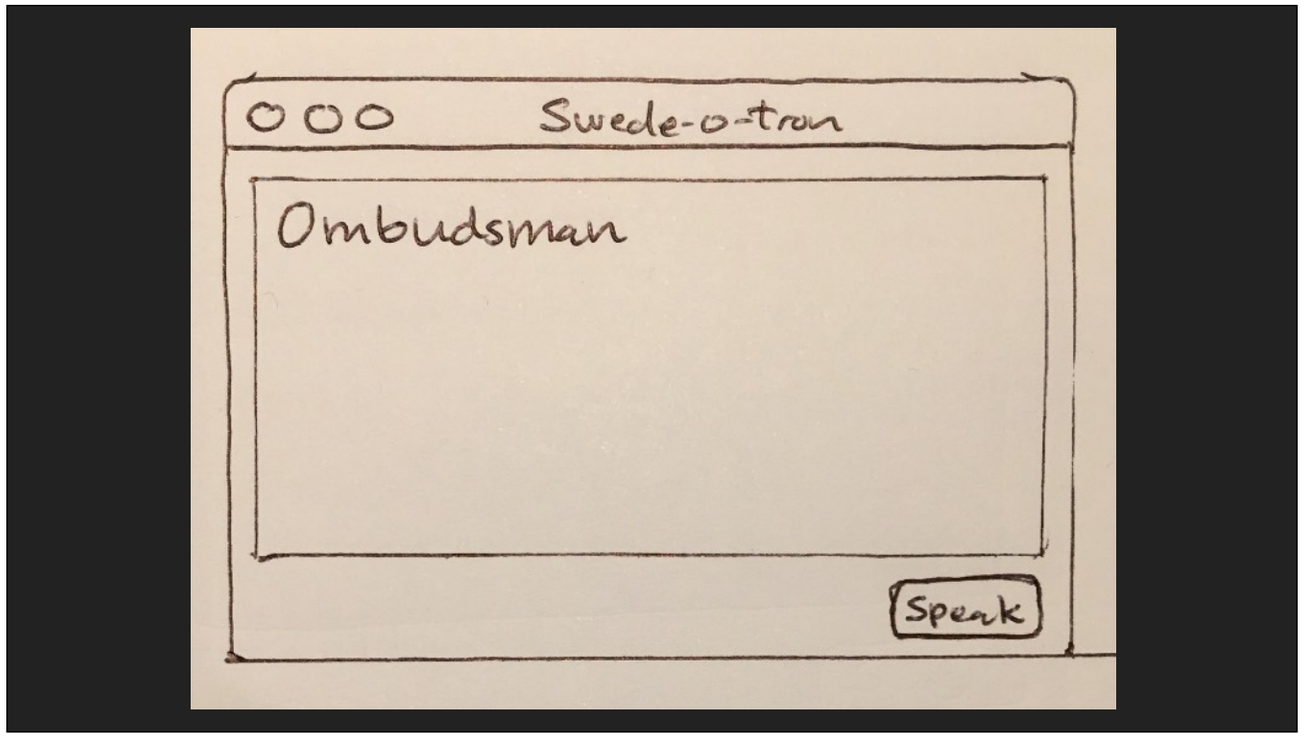


bin bash
say dash v Alva quote dollar at
save that as what would Kamprad say

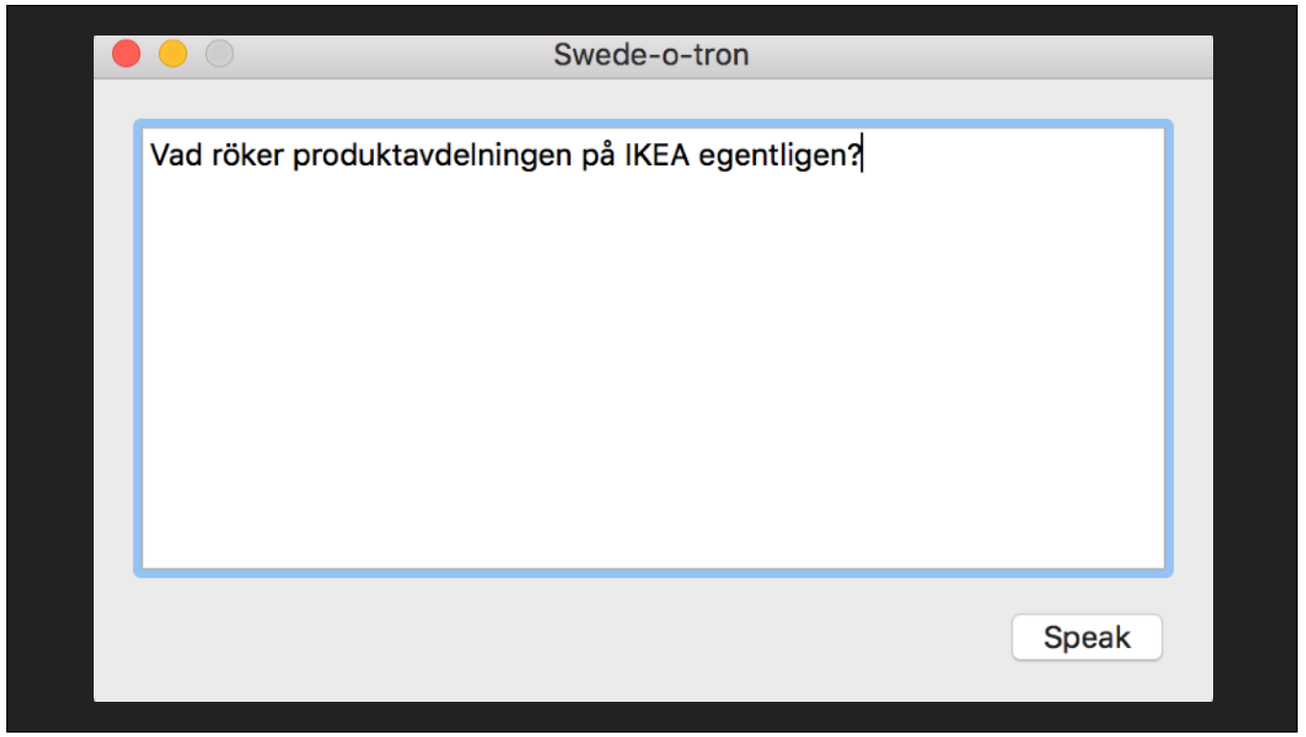
And now we have a super useful script that we want to use as the foundation of our app.



So what's the next step in turning this into a GUI application? Well, we need to think about what the user interface for this should look like. The goal here is to make something that's as familiar for the user as possible, with few surprises. And if I told you that I downloaded this great app from the App Store, where you type in a phrase and the computer speaks it back to you in Swedish, what would that look like?



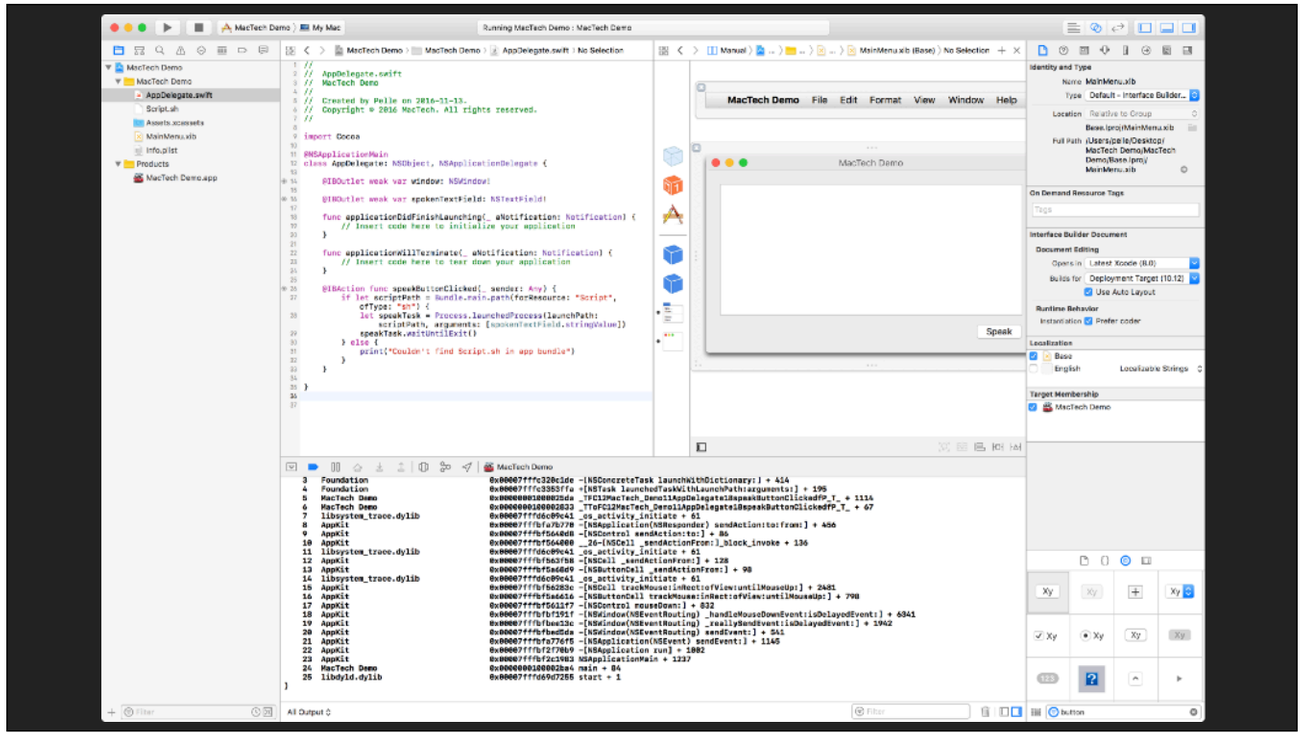
In my mind, there's a window and it has a text field where you enter the text, and there's a button you click on to make it speak.



It's going to look something like this.

So now we have our idea, we have a proof of concept, we know that it works, and a rough idea of what the app should look like.

At this point I'd be all gung ho...



...and I'd fire up Xcode, and within minutes I would be really, really confused. There is a lot going on here, and at least for me, until I did my homework I couldn't get past the complexity.

Before we start

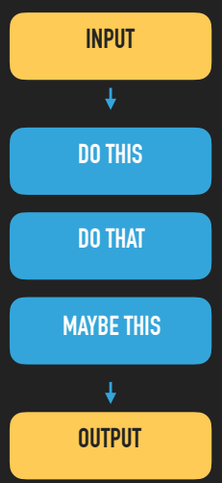
Important Cocoa concepts

So there are a few key concepts that you need to learn before you start.

EVENT DRIVEN

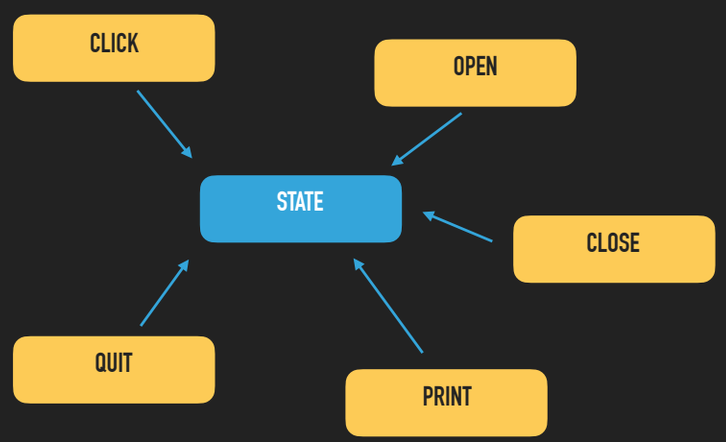
The first is that of event driven application architecture.

Command-line app



When we write shell scripts or command line tools, things are fairly straightforward. There's some input, you do some things to it, change a few things, and then it's done and maybe there's some output.

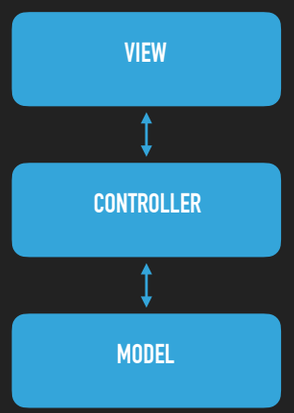
GUI APP



But a GUI app doesn't work that way, when it's running it spends most of it time waiting for the user to do something. The application is in a certain state, and as the user clicks on buttons or maybe it loads a file from disk, the state of the application changes. All it does is react to events.

Model-View-Controller

- The model contains the core app logic
  - The problem you're trying to solve
- View objects compose the app's UI
  - Buttons, text fields, ...
- Controllers manage application flow
  - React to events, prepare data for display, ...

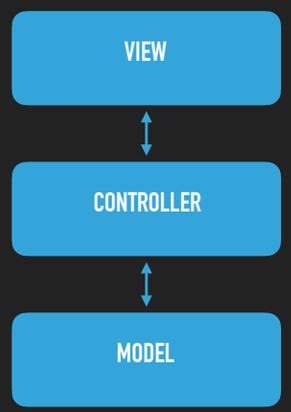


So what's a good way to organize an event driven application? The most common design pattern is called Model-View-Controller. It divides our code into three parts.

* The model is the core logic, the problem you're trying to solve
* The view contains the user interface, the buttons and text fields, and so on
* The controller sits between the model and the view and handles the program flow
It reacts to events, prepares data for display, and so on

Model-View-Controller

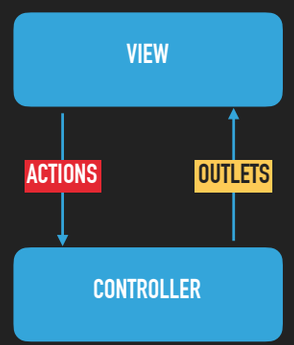
- The model contains the core app logic
  - The problem you're trying to solve
- View objects compose the app's UI
  - Buttons, text fields, …
- Controllers manage application flow
  - React to events, prepare data for display, …



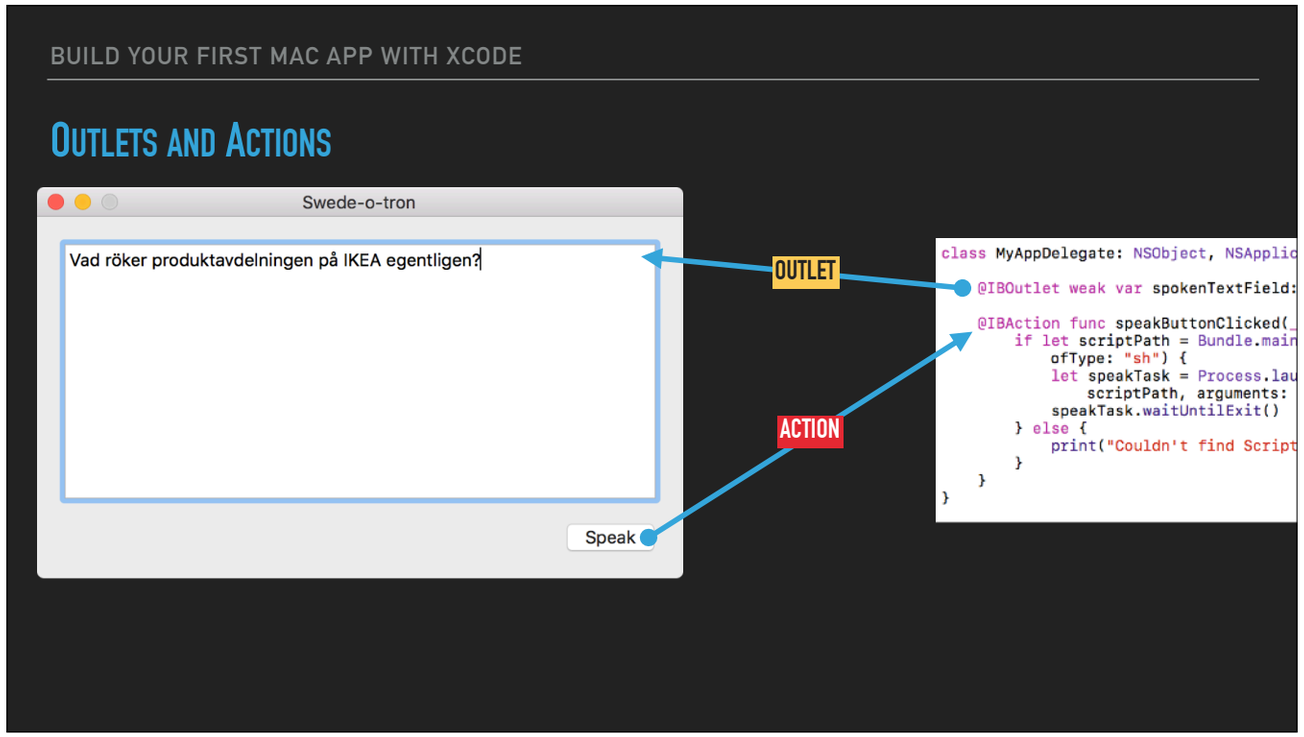
If you've written web apps, this is all probably very familiar.

Cocoa apps follow this pattern, and the Cocoa framework also comes with a rich set of user interface objects that we can use to construct the view, as well as classes that help us build models and controllers.

Outlets and Actions



Cocoa also has a standard way of communication between the view and the controller. When for example a user clicks on a button, that connects to an action in the controller code. When the controller code needs to access a view object, it does so through an outlet.



So if we look at our user interface, we're going to need two connections. We need an action for the button to handle clicks, and we need an outlet to the text field so we can read the text that the user entered.

That's the first important concept, Model-View-Controller, with outlets and actions.

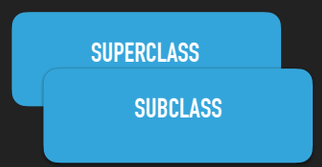
Delegation

The second concept I want to explain before we start is that of delegation.

Cocoa often provides an object for you with a lot of default functionality, for example you have NSApplication that provides the necessary code to start up and tear down an application, or you have NSTableView that can display tabular data that you can scroll around, and you can sort and resize the columns, and you know.. act the way we're used to. Often though you want to customize that behavior, for example in your application you might want to warn the user not to quit if the app is busy doing something.

Inheritance

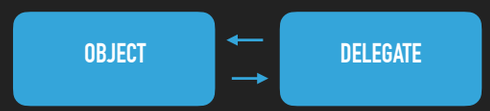
- Superclass provides default functionality
- Subclass inherits and overrides
- Strong, compile time coupling



A common way of solving that in object oriented environments is through inheritance. The framework provides a superclass which you inherit from, and you override methods in a subclass. This is used in a few places in Cocoa, particularly for view objects, but in your code it's more common to use delegation.

Delegation

- A delegate object is an object that gets notified when the object to which it is connected reaches certain events or states
- Loose, runtime coupling



So instead of creating a subclass, you create a delegate object, a buddy, and you let the other object know that you're there, and you're willing to listen, share some of the responsibility. When the companion object reaches certain points, it notifies the delegate, or asks it what it should do.

So for example NSApplication, when it starts up it tells the delegate that hey, we're good to go, if you have anything you need to initialize, now's a good time. Or when the user presses Command-Q, it asks the delegate, the user wants to quit, should we? And the delegate can control that with a yes or a no.

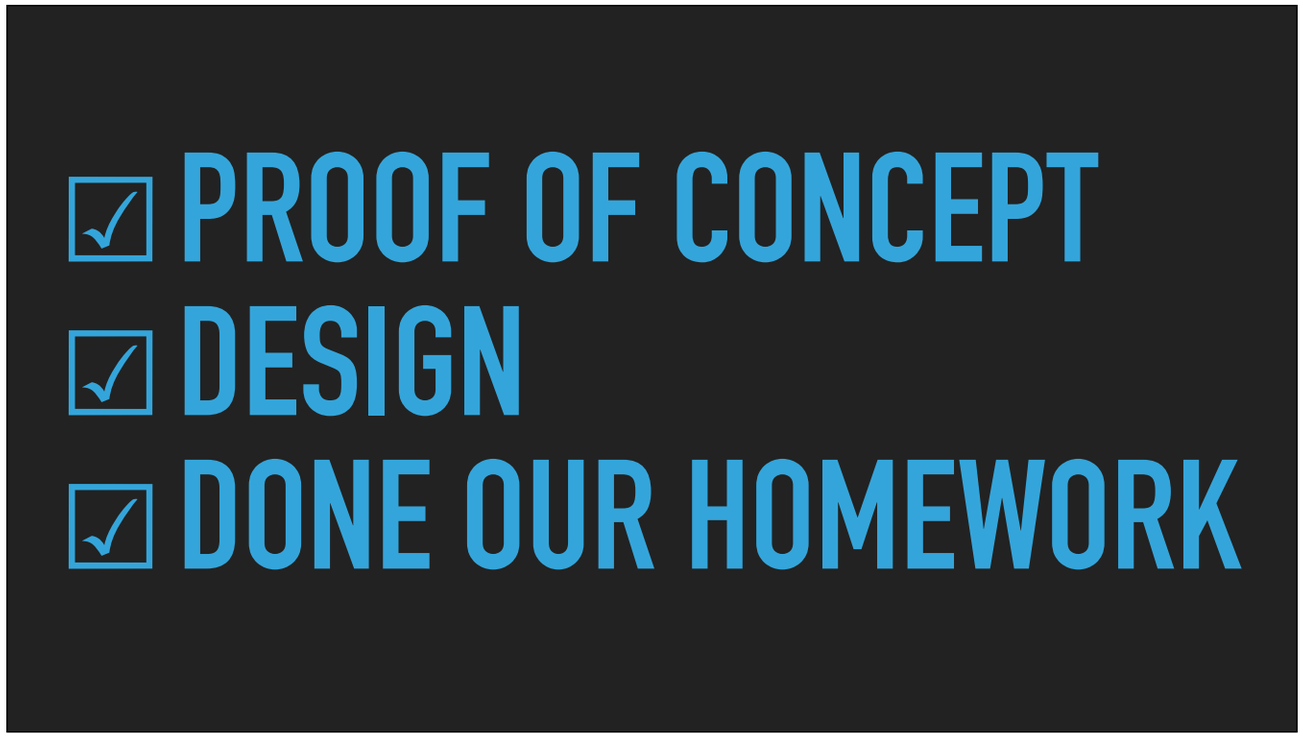
Protocols

- A formalized way of saying:
  - If it walks like a duck
  - and quacks like a duck
  - it is a duck

```
protocol Duck {
    var quack: String { get }
    func waddle() -> String

}


class Mallard: Duck {

    let quack = "Quack!"

    func waddle() -> String {
        return "Like a duck"
    }

}
```

The way delegates are implemented is through protocols. Protocols are basically a formalized way of saying that if something quacks like a duck and walks like a duck, it is a duck. So when we declare our class to conform to the NSApplicationDelegate protocol, we make some promises about what we respond to and how.

So that's the second important concept, delegation implemented through protocols.



We have a proof of concept and a design, we learned that we should structure our app into models, views, and controllers, and to customize behavior we implement a delegate.

With that I think we're ready to jump into Xcode.

I will warn you that I will be coding live, and there is at least one Xcode 8 bug that makes it not compile any more, so I really hope we don't run into that one.

Any questions before we start? Everything clear so far?

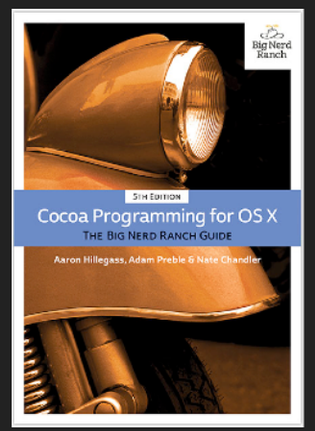

You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.
Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de réinitialisation.
Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.
Musisz zrestartować swój komputer. Wciśnij i przytrzymaj przez kilka sekund przycisk Power lub wciśnij Restart.
TIME FOR A...
LIVE DEMO!

WHERE TO GO FROM HERE

- Apple's starting point for learning how to create Mac apps:
  - https://developer.apple.com/library/content/documentation/General/Conceptual/MOSXAppProgrammingGuide/Introduction/Introduction.html
- Xcode docs
  - http://help.apple.com/xcode/mac/8.0/

WHERE TO GO FROM HERE

- Cocoa Programming: For Mac OS X 5th Edition
  - https://www.bignerdranch.com/we-write/cocoa-programming/



WHERE TO GO FROM HERE

- https://github.com/MagerValp/MacTech-2016
  - MacTech Demo in Swift, Objective-C, and Python
- per.olofsson@gu.se / @magervalp