From waaa? to oooh.

Writing Maintainable Code

Who is this dude?

Josh Noe

Freelance software developer

Mostly enterprise (business) software

Currently under contract with Mixed In Key. Not enterprise!

I just finished a gig as a code reviewer, so maintainability is on my mind.

Sometimes I work in a van, sometimes down by a river

The Psychopathic Maintainer Rule

What is Maintainable Code?

“Maintainability isn't a binary property, either being maintainable or not. It's a continuum. Roughly speaking, maintainability is inversely proportional to the amount of time it takes a developer to make a change and the risk that change will break something.”

-Karl Bielefeldt, some guy on Stack Overflow

“Maintainable code is code that doesn’t piss you off when you try to work on it”

-Josh

But the program works. So who cares?

- You do! As a program grows, it generally takes more time to fix bugs and add features. Make it easier on yourself.
- Your colleagues do! They may not “know” your code like you do. Make it easier on them.
    - All meaningful software is a team effort.
- Your boss does! ...well they actually don’t, but if they ask you to add a feature to your program, and it takes you a week instead of a few hours, they do.
- The Psychopath does.

Our example app: First Round

Basic idea:

You’re in a new city. You want to go out for a beer or three, but you don’t know anyone, and the prospect of drinking alone AGAIN has got you down :(

Open up the First Round app!

Search for local people who are offering to hang with out of towners. The locals give you the local experience, you buy the first round.

Let’s get started...

Self-documenting code

Code that describes what it does. It’s meaning is included in the code itself.

“This line isn’t self-documenting, it took me 10 seconds to understand.”
-David Mohundro, technical lead, Clear Function

What the hell does this do?

```
var l = G(string c, Date d)
```

Let’s make it self-documenting!

Meaningful Names: Don’t abbreviate

If we rename our variable and arguments, it’s a little more clear...

```
var local = Get(string city, Date date)
```

Meaningful Names: Functions too

...but how would we know what does the Get function does if we didn't see a call to it?

```
User Get(string city, Date date)
{
    var dbReturn = dbConnection.Query($@“select * from Users users
                                            join UserAvailability uAvail on users.Id = uAvail.UserId
                                           where users.city = {city} and Date.ConvertTo(4, users.Date) = {dayOfWeek}
                                           and users.IsActive and users.cthulhu > 8 and uAvail.proclaimation = ‘the end is upon us’..........
    return ConverToUser(dbReturn)
}
```

Our function gets a local who's available to hang today. What would be a better name for it?

Meaningful Names: Be explicit

Our function gets a local who’s available to hang today...

```
User GetLocalAvailableToHangToday(string city, Date date)
{
   //blah blah
}
```

However, First Round is booming! Surely there are hoards of locals eager to hang in this city today. How does this function determine which local to choose? Do we care? Sometimes.

Meaningful Names: Optionally describe implementation details.

More specifically, our function gets a local who’s available to hang today, weighted by how long they’ve lived in the city.

What should we name it?

Meaningful Names: Optionally describe implementation details.

Our function gets a local who’s available to hang today, weighted by how long they’ve lived in the city...

```
User GetLocalWithLongestCityResidencyAvailableToHangToday(string city, Date
date)
{
  //blah blah
}
```

Is this too verbose? Do we have other functions that GetLocalsAvailableToHangToday in different ways? Should we move this into a “LocalAvailability” module????!!!

Naming is hard. Don’t be afraid to come back to it.

Functions should do 1 thing

Functions should do 1 thing

Size does matter.

Smaller functions are generally better.

Even 1-line functions are great if they add meaning! Tiny little adorable functions

General rule of thumb, if you’ve gotta scroll, your function is too big

More important than size: Functions should do 1 thing

Our example. GetTheCityIAmIn

What does GetTheCityIAmIn do? Is the below code expected given the name of the function?

```
City GetTheCityIAmIn()
{
    var location = device.GetGPSLocation()
    If (location.IsNotFound)
    {
        location = device.GetLocationOfWifi()
    }
    return location.City
}
```

Unexpected side-effects

Would you expect GetTheCityIAmIn to post to your Tinder profile?

```
City GetTheCityIAmIn()
{
    var location = device.GetGPSLocation()
    If (location.IsNotFound)
    {
        location = device.GetLocationOfWifi()
    }
    tinderService.PostToMyProfile(location.City)
    return location.City
}
```

I’d swipe that code left

If a functions changes state, explicitly say so

```
void PostMyLocationOnTinder()
{
    var city = GetTheCityIAmIn()
    tinderService.PostToMyProfile(city)
}
```

Anything wrong with this function’s name?

Should functions that update things return anything? What would they return? Why have we been using camelCase this whole time? Should we put brackets on newlines?!??!

Follow conventions.

What are code conventions?

Coding conventions are a set of guidelines for a specific programming language that recommend programming style, practices and methods for each aspect of a piece program written in this language

-Wikipedia, the largest online encyclopedia and originator of edit wars

Coding conventions are rules you follow to keep the program’s code consistent. They can be for a language, but they can also be for a framework or methodology, or even for coding in general

-Josh

Examples

Some aspects of code conventions:

- Casing (camelCase, PascalCase, underscore_case)
- Syntax (same line brackets, new line brackets)
- File / Folder Structure
- Naming
- Language constructs that are there, but shouldn’t be used (goto, switch)
    - For the record, I think switch is just fine, but don’t listen to me
- Flow of code

Why follow conventions?

iMAGINE*iF*sOMEONE*wROTE*lIKE*tHIS*aND*sEPARATED*cHARACTERS*wITH*sTARS*aND*tERMINATED*sENTENCES*wITH*hASSELHOFFS*iNSTEAD*oF*pERIODS

It would be hard to understand.

An example: Back to First Round

In First Round, we want to create a function that “books” the hangout between an out of towner and a local, and returns the id of the hangout so our app can put it on our calendar:

```
int BookHangout(User outOfTowner, User local, Date dayToHangOut)
{
   If (local.IsBookedForDay(dayToHangOut))
   {
      return -1;
   }
  else
  {
    int hangoutId = local.CreateNewHangOut(dayToHangOut, outOfTowner)
    return hangoutId
  }
}
```

...but is someone who calls this function expecting to check for a -1? Could this be a convention? What about functions that don't have possible failures?

BookHangout solutions?

We could to this:

```
void BookHangout(User outOfTowner, User local, Date dayToHangOut)
{
    If (local.IsBookedForDay(dayToHangOut)
    {
        throw Exception("The local is already booked for {dayToHangOut}")
    }
   else
   {
       local.BookForDay(dayToHangOut, outOfTowner)
   }
}
```

...but as a general convention, using exception handling for flow control is bad. User unavailability is not an "exceptional" condition.

What to do?

We see that this is used elsewhere in the code:

```
bool TryToCancelAvailabilityOfLocal(User local, Date day, out int cancellationId)
{
    If (local.IsBookedForDay(day))
    {
        return false
    }
   else
   {
        cancellationId = local.CancelAvailability(day)
        return true
   }
}
```

...how about using this TryTo naming convention?

TryToBookHangOut

Using the same “TryTo” convention...

```
int TryToBookHangout(User outOfTowner, User local, Date dayToHangOut, out int hangoutId)
{
    If (local.IsBookedForDay(dayToHangOut))
    {
        return false
    }
   else
   {
      hangoutId= local.CreateNewHangOut(dayToHangOut, outOfTowner)
      return true;
   }
}
```

But everyone writes code differently, what’s the ONE TRUE way?

Pick a convention by this order of precedence:

1. Does the code base have a common convention? Use it. Even if you aren’t crazy about it.
2. Starting a new project? Does the language / framework have a standard convention? Use it.
3. Still No??! Ask Jeeves “<language> conventions” and find someone with more experience than you who’s come up with a good one.
4. Still No???!!! I don’t believe you, but create a convention, stick to it, and publish it after you’re finished.

Comments?

Code comments.

More comments >! fewer comments

Generally speaking, self-documenting code is better than code comments.

Code changes. If you have comments everywhere, you have to change them every time your code changes.

In the unfortunate situation where you change code, but forget to change its comments, the psychopath will be very displeased.

As a side-note, commenting out code instead of deleting it is generally considered smelly. Use version control!

So no comments? No, comments.

Use comments in these situations:

- You’re writing an api
  - Possible exceptions
- You want to annotate places in code that you’ll want to come back to
  - Common annotations:
    - //TODO:
    - //TESTING:
    - //HACK:
- Return values for languages where the return isn’t named
- You’re describing “why” the code is doing something instead of “what” it’s doing. Sometimes helpful when dealing with APIs you don’t control. Example on next page....

Commenting for “why”

For First Round, we want to remove all hangouts for a certain user. Let’s do it:

```
void RemoveAllHangoutsForUser(Hangout[] allHangOuts, User user)
{
    for(int i = 0; i < allHangouts.length; i++)
    {
        bool userIsAPartOfThisHangout = allHangouts[i].HasUser(user)
        if (userIsAPartOfThisHangout)
        {
            allHangouts.RemoveAtIndex(i)
            //since we removed a hangout above, we must decrement i by 1 so the next hangOut isn’t skipped
            i = i - 1
        }
    }
}
```

A cooooler way!

Check this out! Way cooler and fewer lines of code! FEWER. LINES. OF. CODE.

```
void RemoveAllHangoutsForUser(Hangout[] allHangOuts, User user)
{
    for(int i = 0; i < allHangouts.length; i--)
    {
        int j = allHangouts[i].HasUser(user) ? i - 1: i
        if (i != j) allHangouts.RemoveAtIndex(i);
        i = j;
    }
}
```

Stop showing off.

Write uncomplicated code

- Avoid “one-liners” if they do more than one simple thing
    - Solution: break it up into a sweet multi-liner
- Avoid function calls as function arguments or in if statements
    - Solution: use intermediate variables instead
- Avoid nested loops
    - Solution: call functions from inside your loops. These functions have the inner loops
- Avoid “magic” values
    - Use constants or enums
- Premature optimization is the root of all evil
    - Solution: when you encounter a performance issue, profile it and fix it

Who has code we can review?

Questions?

- Dependency Injection
- Unit testing
- Working remotely
- Working with larger groups
- Coding jokes

Thanks!

josh.noe@gmail.com
609-602-8801
Skype: josh_noe