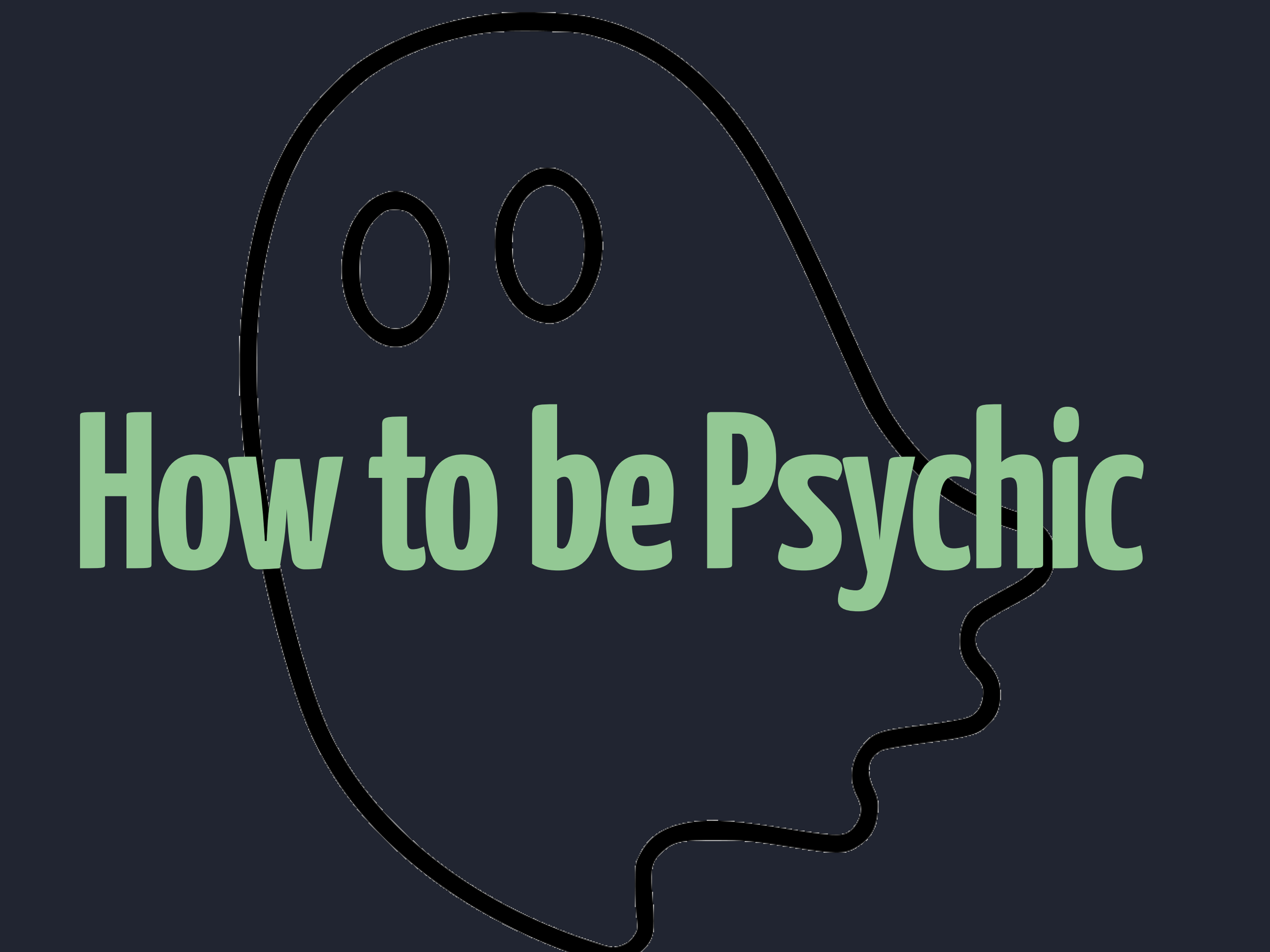


How to be Psychic

hey hi hello

derek graham @deejaygraham


Sage

https://ti.to/ne-bytes

Confession Time

Clickbait?

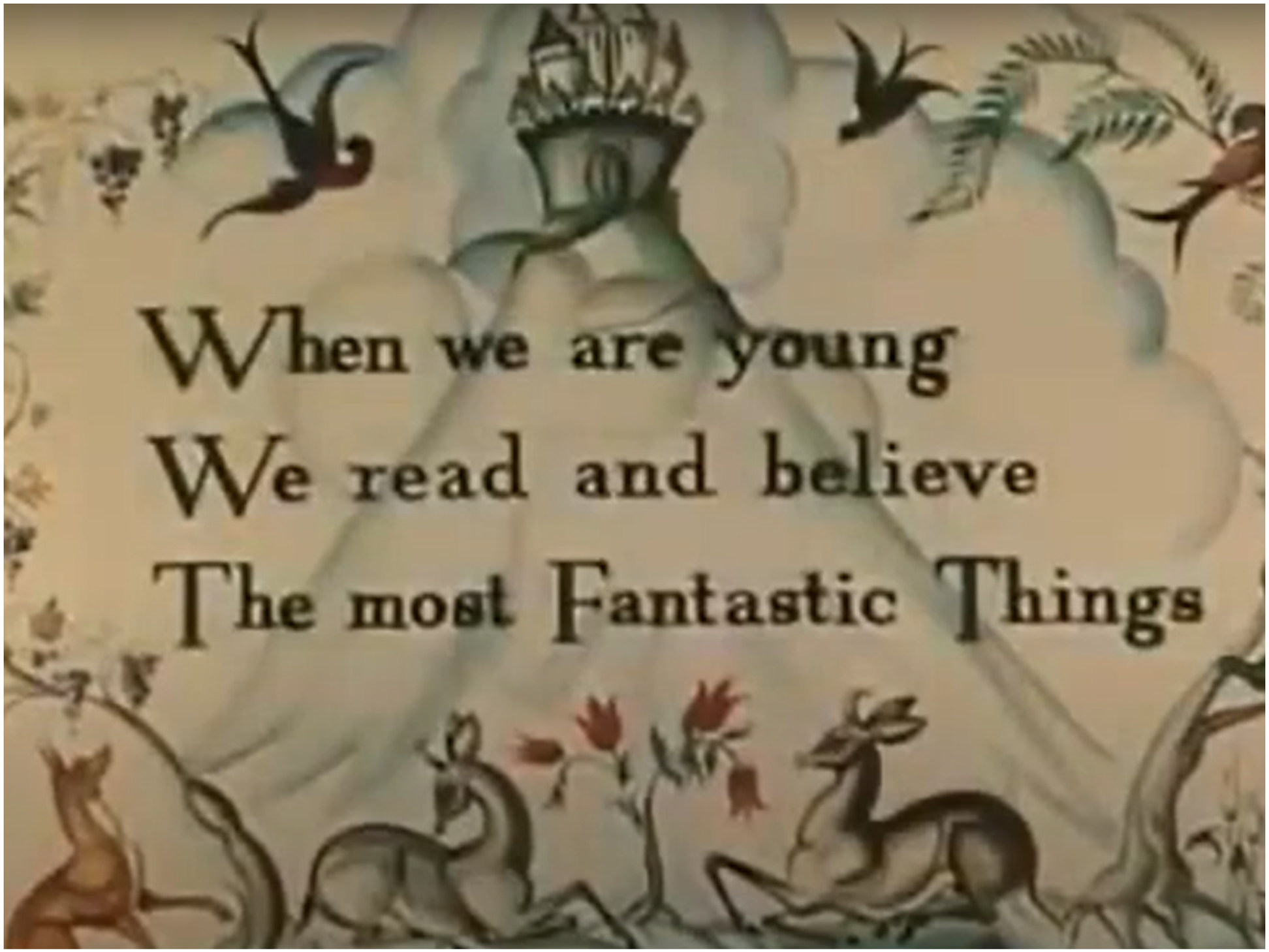
A TWO CITIES FILM

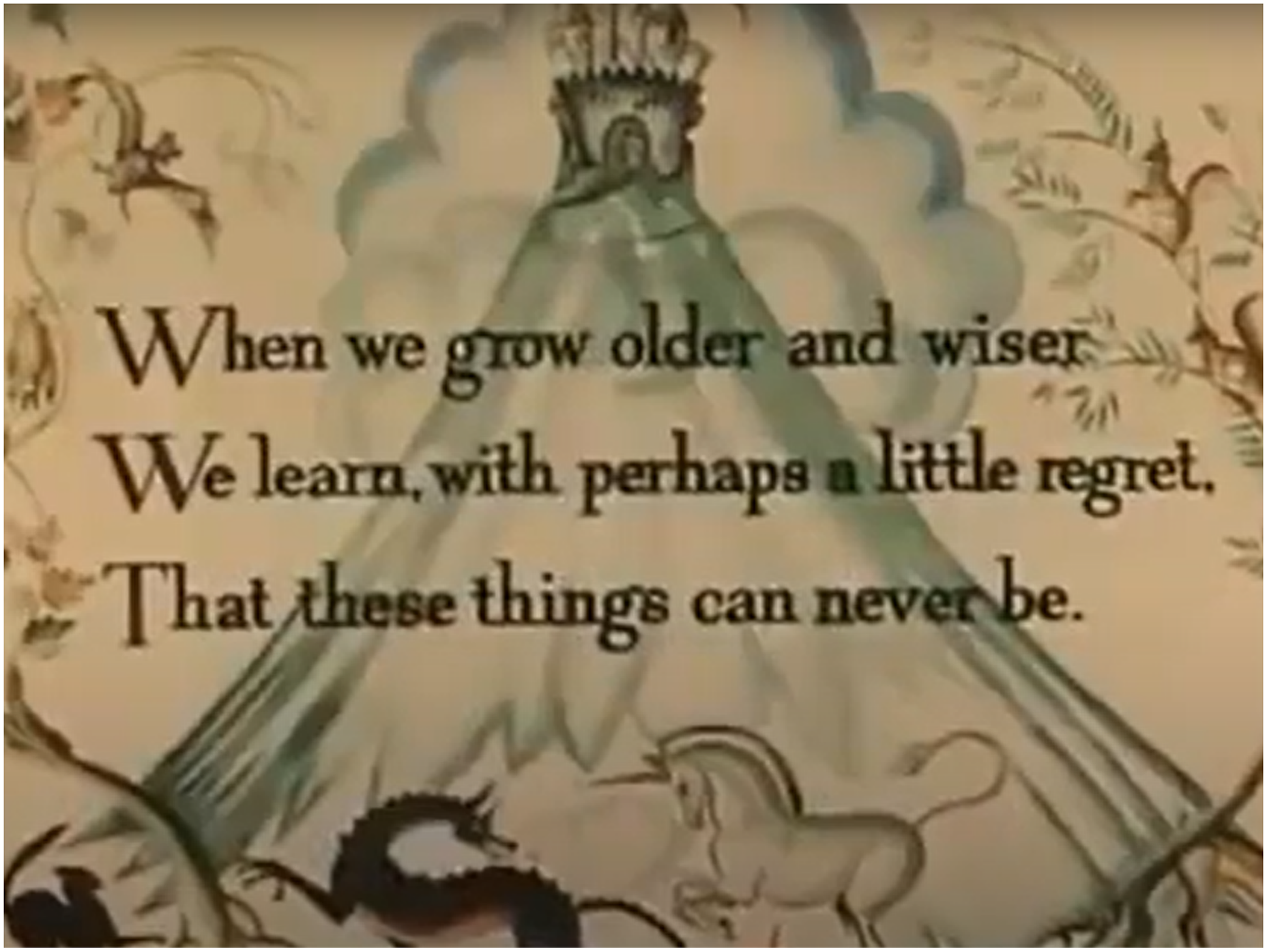
Noel Coward's

"BLITHE SPIRIT"

In Technicolor

A NOEL COWARD - CINEGUILD PRODUCTION


When we are young
We read and believe
The most Fantastic Things


When we grow older and wiser
We learn, with perhaps a little regret,
That these things can never be.

Maximum Value

Tradition!

new System();

if (exists)

“The need to be and feel in control is so strong that individuals will produce a pattern from noise to return the world to a predictable state.”

Whitson and Galinsky

B.D.U.F.


Programmed With
Ampion
AMC
F2
F3
Esc
Alt

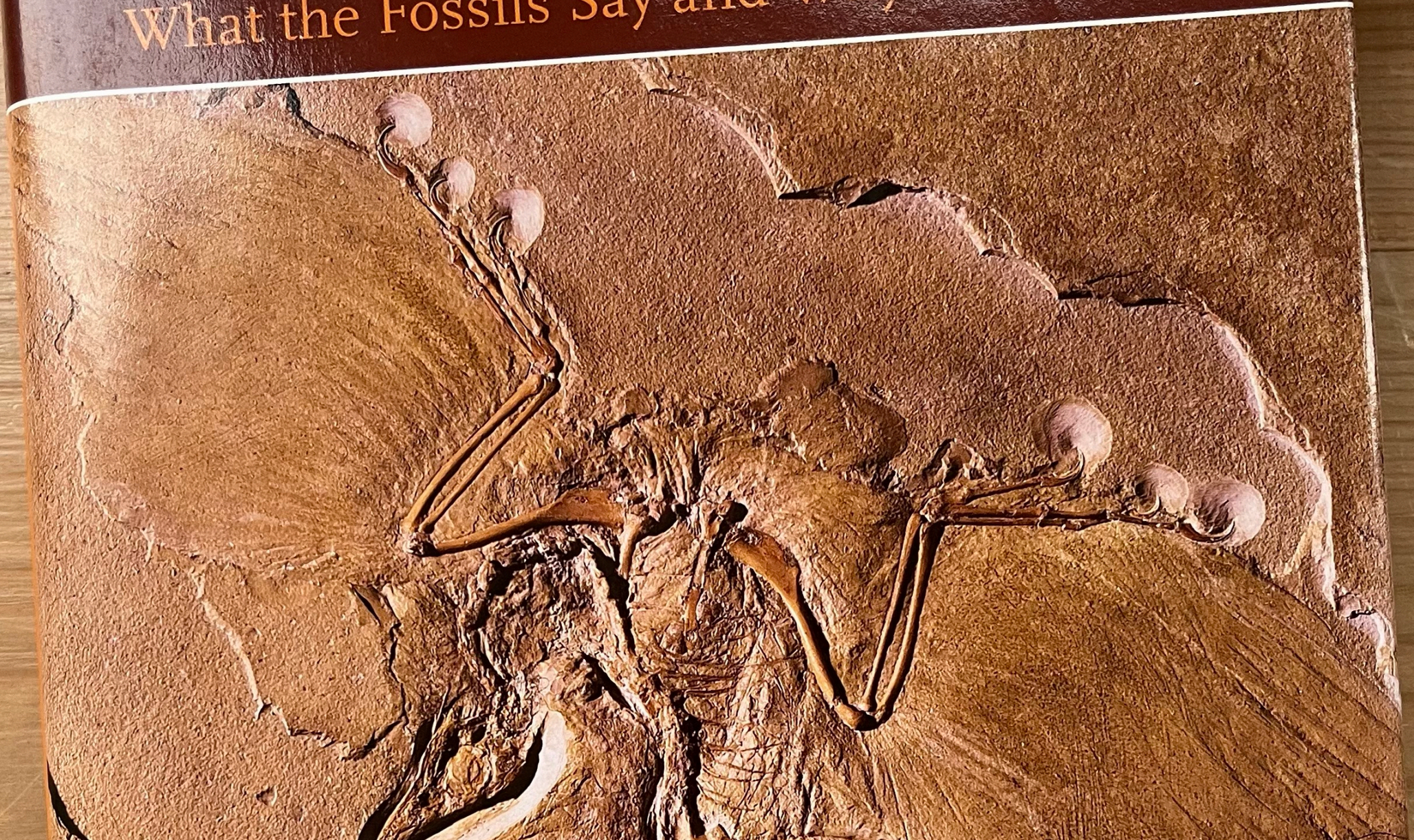

Programmed With
Ampion
AMC
F2
F3
Esc
Alt

Incremental, Iterative

Ad-hoc

EVOLUTION

What the Fossils Say and Why It Matters



"It is not necessary to change.
Survival is not mandatory."

W. Edwards Deming

"All Complex systems that work evolved from simpler systems that worked. If you want to build a complex system that works, build a simpler system first, then improve it over time.”

John Gall

- Simple Design
- Small Steps
- Incremental
- Continuous
- Refactoring & Tests

Walking Skeleton

The Nature of Software Development

Keep It Simple,
Make It Valuable,
Build It Piece by Piece

Ron Jeffries

$@?&%! First Draft

“Almost all good writing begins with terrible first efforts”

Annie Lamott

Wait

Kaizen

Review

Process

- Review
- Reverse Engineer
- Imagine
- Small Steps
- Refactor
- Add Tests

Addison-Wesley Professional Ruby Series

REFACTORING

RUBY EDITION

JAY FIELDS ■ SHANE HARVIE ■ MARTIN FOWLER
with KENT BECK

Addison-Wesley Professional Ruby Series

REFACTORING

RUBY EDITION

JAY FIELDS ■ SHANE HARVIE ■ MARTIN FOWLER
with KENT BECK

- Encapsulate Var
- Extract Method
- Introduce Variable
- Rename Method
- Move Method

extra hard to understand.

Mechanics

1. Choose the parameters that you want to name. If you are not naming all of the parameters, move the parameters that you want to name to the end of the parameter list.

   ⟹ *That way your calling code does not need to wrap the named parameters in curly braces.*

2. Test

3. Replace the parameters in the calling code with name/value pairs

4. Replace the parameters with a `Hash` object in the receiving method. Modify the receiving method to use the new `Hash`.

5. Test.

Example 1: Naming All of the Parameters

- Inspect
- Prepare
- Research
- Refactor
- Clean up

All The Tests

TDD

- Micro tests
- Approval tests
- Contract tests

Just Test!!!

When not to evolve

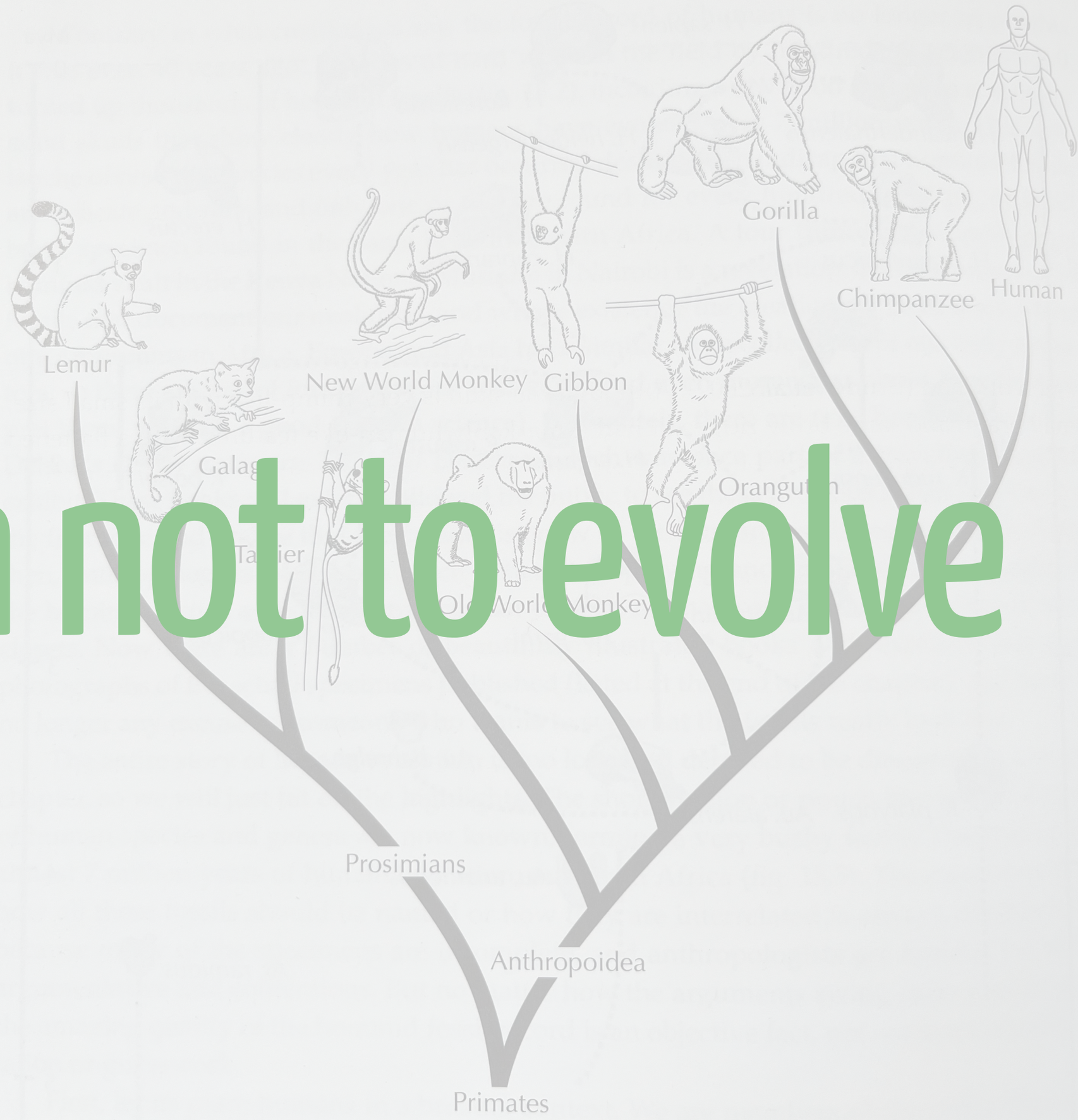


Figure 15.4 Family tree of the major groups of living primates, showing the relationships between

Feeling More Psychic?

Harry Houdini's 10 Spooky tips for Evolutionary Design

#1 SOLID

#2 Simple Design

- Tests
- Duplication
- Clarity
- Smallest

Notice and Improve the Names of Things

#3 Three C's of Design

- Cohesion
- Coupling
- Connascence

Connascence

... a change in one would **require** the other to be modified in order to maintain the overall **correctness** of the system

Dynamic:

Static:

#4 Keyboard

ReSharper

part of dotUltimate

The Visual Studio Extension for .NET Developers

DOWNLOAD

Free 30-day trial

WHY RESHARPER

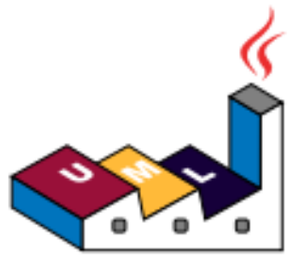
#5 Draw

'The mind a great place to have ideas but a terrible place for keeping them'

- David Allen


create_

Drawing UML with PlantUML



PlantUML Language Reference Guide

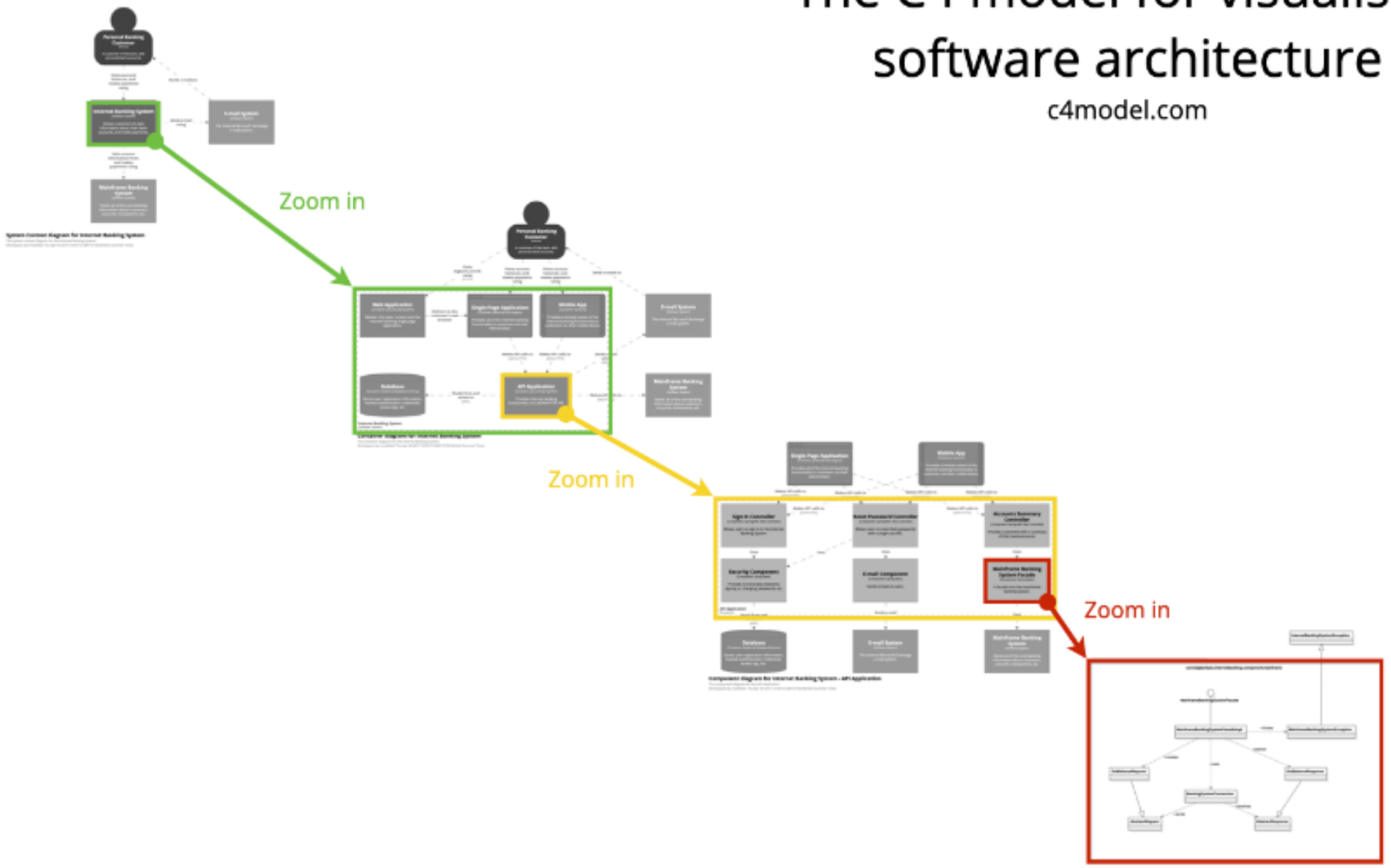
(Version 1.2021.2)

PlantUML is a component that allows to quickly write :

- Sequence diagram
- Usecase diagram
- Class diagram
- Object diagram
- Activity diagram
- Component diagram
- Deployment diagram
- State diagram

The C4 model for visualising software architecture

c4model.com



Level 1
Context

Level 2
Containers

Level 3
Components

Level 4
Code

#6 Smell

Chapter 3

Bad Smells in Code

If it stinks, change it.

Grandma Beck, discussing child-rearing philosophy

By now you have a good idea of how refactoring works. But just because you know how doesn't mean you know when. Deciding when to start refactoring, and when to stop, is just as important to refactoring as knowing how to operate the mechanics of a refactoring.

Now comes the dilemma. It is easy to explain how to delete an instance variable or create a hierarchy. These are simple matters. Trying to explain when you should do these things is not so cut-and-dried. Rather than appealing to some vague notion of programming aesthetics (which frankly is what we consultants usually do), I wanted something a bit more solid.

I was mulling over this tricky issue when I visited Kent Beck in Zurich. Perhaps he was under the influence of the odors of his newborn daughter at the time, but he had come up with the notion describing the "when" of refactoring in terms of smells. "Smells," you say, "and that is supposed to be better than vague aesthetics?" Well, yes. We look at lots of code, written for projects that span the gamut from wildly successful to nearly dead. In doing so, we have learned to look for certain structures in the code that suggest (sometimes they scream for) the possibility of refactoring. (We are switching over to "we" in this chapter to reflect the fact that Kent and I wrote this chapter jointly. You can tell the difference because the funny jokes are mine and the others are his.)

One thing we won't try to do here is give you precise criteria for when a refactoring is overdue. In our experience no set of metrics rivals informed human intuition. What we will do is give you indications that there is trouble

Smells to Refactorings
Quick Reference Guide

Smell	Refactoring
Alternative Classes with Different Interfaces: occurs when the interfaces of two classes are different and yet the classes are quite similar. If you can find the similarities between the two classes, you can often refactor the classes to make them share a common interface [F 85, K 43]	Unify Interfaces with Adapter [K 247]
	Rename Method [F 273]
	Move Method [F 142]
Combinatorial Explosion: A subtle form of duplication, this smell exists when numerous pieces of code do the same thing using different combinations of data or behavior. [K 45]	Replace Implicit Language with Interpreter [K 269]
Comments (a.k.a. Deodorant): When you feel like writing a comment, first try "to refactor so that the comment becomes superfluous" [F 87]	Rename Method [F 273]
	Extract Method [F 110]
	Introduce Assertion [F 267]
Conditional Complexity: Conditional logic is innocent in its infancy, when it's simple to understand and contained within a few lines of code. Unfortunately, it rarely ages well. You implement several new features and suddenly your conditional logic becomes complicated and expansive. [K 41]	Introduce Null Object [F 260, K 301]
	Move Embellishment to Decorator [K 144]
	Replace Conditional Logic with Strategy [K 129]
	Replace State-Altering Conditionals with State [K 166]
Data Class: Classes that have fields, getting and setting methods for the fields, and nothing else. Such classes are dumb data holders and are almost certainly being manipulated in far too much detail by other classes. [F 86]	Move Method [F 142]
	Encapsulate Field [F 206]
	Encapsulate Collection [F 208]
Data Clumps: Bunches of data that that hang around together really ought to be made into their own object. A good test is to consider deleting one of the data values: if you did this, would the others make any sense? If they don't, it's a sure sign that you have an object that's dying to be born. [F 81]	Extract Class [F 149]
	Preserve Whole Object [F 288]
	Introduce Parameter Object [F 295]
Divergent Change: Occurs when one class is commonly changed in different ways for different reasons. Separating these divergent responsibilities decreases the chance that one change could affect another and lower maintenance costs. [F 79]	Extract Class [F 149]
Duplicated Code: Duplicated code is the most pervasive and pungent smell in software. It tends to be either explicit or subtle. Explicit duplication exists in identical code, while subtle duplication exists in structures or processing steps that are outwardly different, yet essentially the same. [F76, K 39]	Chain Constructors [K 340]
	Extract Composite [K 214]
	Extract Method [F 110]
	Extract Class [F 149]
	Form Template Method [F 345, K 205]
	Introduce Null Object [F 260, K 301]
	Introduce Polymorphic Creation with Factory Method [K 88]
	Pull Up Method [F 322]
	Pull Up Field [F 320]
	Replace One/Many Distinctions with Composite [K 224]
	Substitue Algorithm [F 139]
	Unify Interfaces with Adapter [K 247]
Feature Envy: Data and behavior that acts on that data belong together. When a method makes too many calls to other classes to obtain data or functionality, Feature Envy is in the air. [F 80]	Extract Method [F 110]
	Move Method [F 142]
	Move Field [F 146]

#7 Read

#8 Deliberate Practice

cyber-dojo

a place to practice programming

create a new practice

enter an existing practice

Is cyber-dojo free?

It depends...

- Using `https://cyber-dojo.org` in a commercial organization requires a licence
- Non-commercial use is free, but please donate
- 100% of the licence fees and donations help children learn about software
- The cyber-dojo Foundation is a Scottish Charitable Incorporated Organisation

Donate £ Donate $ Donate €

How much is a licence?

It's up to you!

- Large companies/organizations typically pay ~ £1000
- Smaller teams/individuals, less

How long does a license last?

That's also up to you!

- You can pay a larger amount once
- You can pay a smaller amount every year

How do I buy a licence?

There are two ways:

- For larger corporations, email license@cyber-dojo.org with much you would like to pay

#9 Steal

#10 Get Going

- Learning Hour
- Join an Ensemble
- Individual Practice
- Find a Mentor

Bad Judgement ->
Terrible Mistakes ->
Good Judgement

Finally...

- Fix **most** important thing
- Reduce cost of change
- Ctrl + Z

If in doubt...

- Add a test
- Remove duplication
- Reduce scope
- Find a better name


@deejaygraham


Questions
Answers