ThoughtWorks®

DDD Brisbane 2016

IF SIMPLE WERE EASY, WHY IS MAINTENANCE SO HARD?

Sarah Taraporewalla
www.sarahtaraporewalla.com

“Everything should be made as simple as possible,
but not simpler”

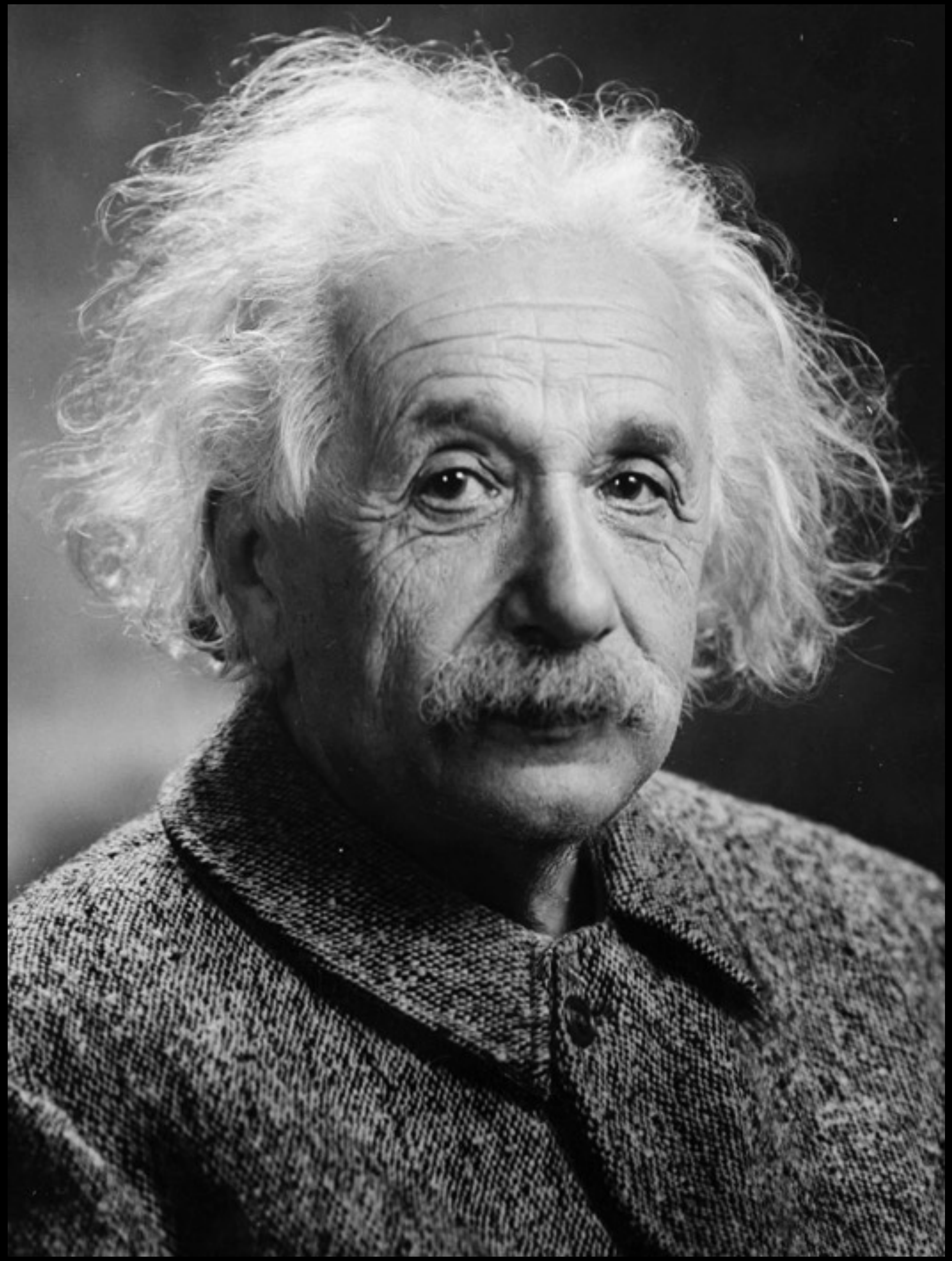


Photo courtesy of https://commons.wikimedia.org/wiki/File:Albert_Einstein_Head.jpg

2 6 8

2	6	8
1	3	4

2	6	8
1	3	4
2	5	7

2	6	8
1	3	4
2	5	7
5	8	11

$$x < y < z$$

2	6	8
1	3	4
2	5	7
5	8	11

TODAY WE WILL EXPLORE

Simple and Complex

Simple Code

Simple Explanations

SIMPLE AND COMPLEX

“Simplicity is a great virtue but it requires hard work to achieve it”

- Edsger W Dijkstra

Photo courtesy of https://upload.wikimedia.org/wikipedia/commons/d/d9/Edsger_Wybe_Dijkstra.jpg

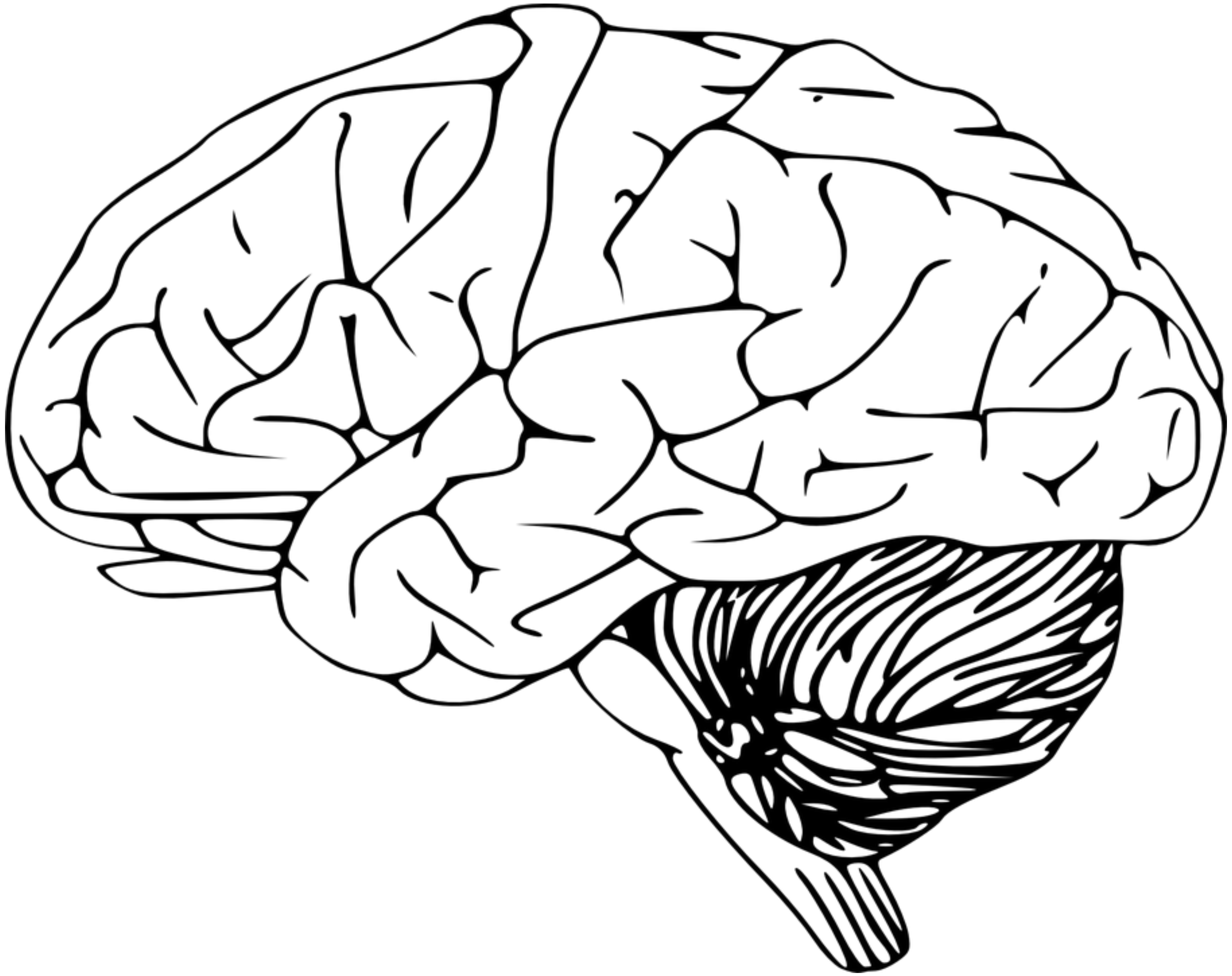
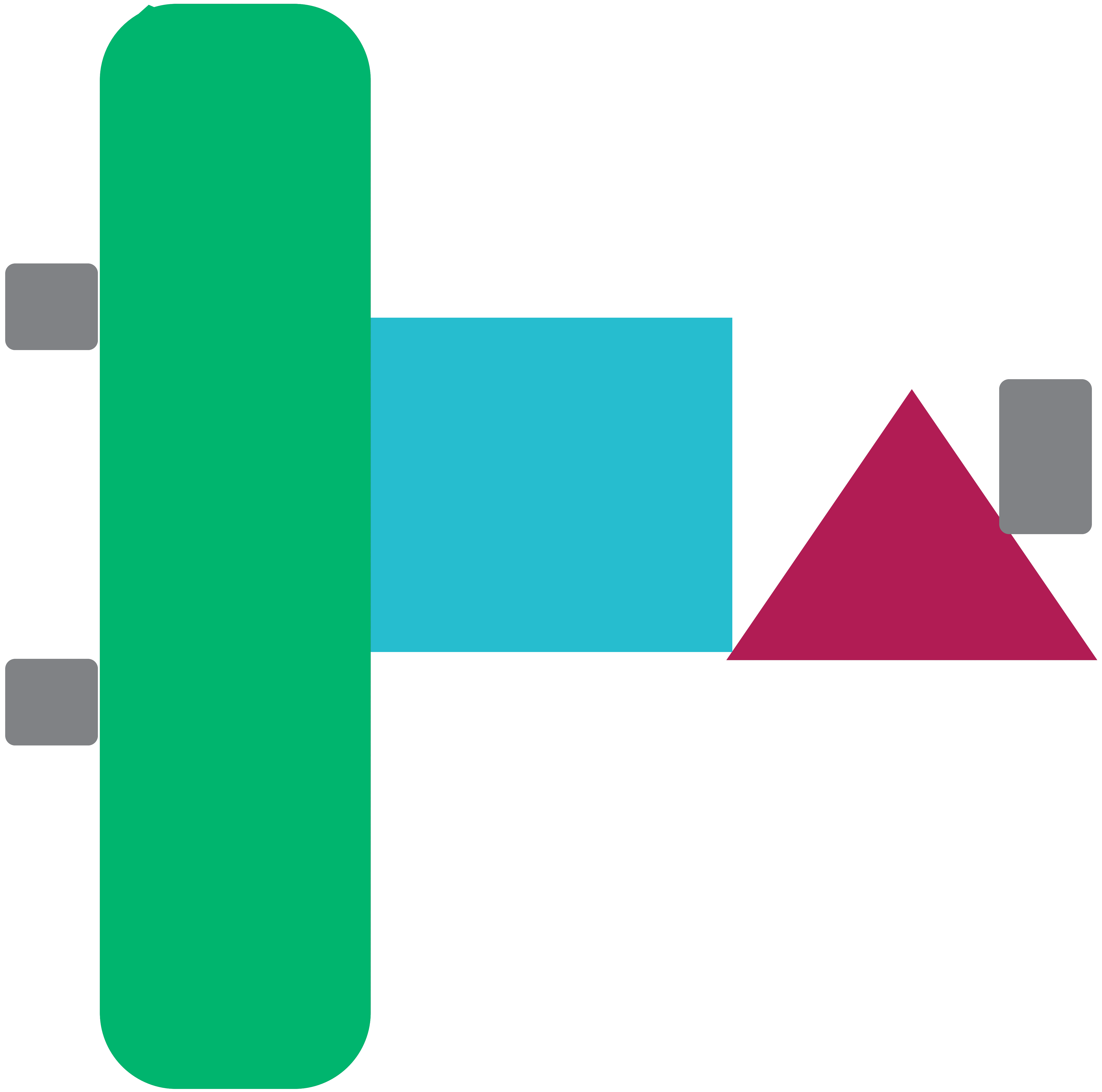
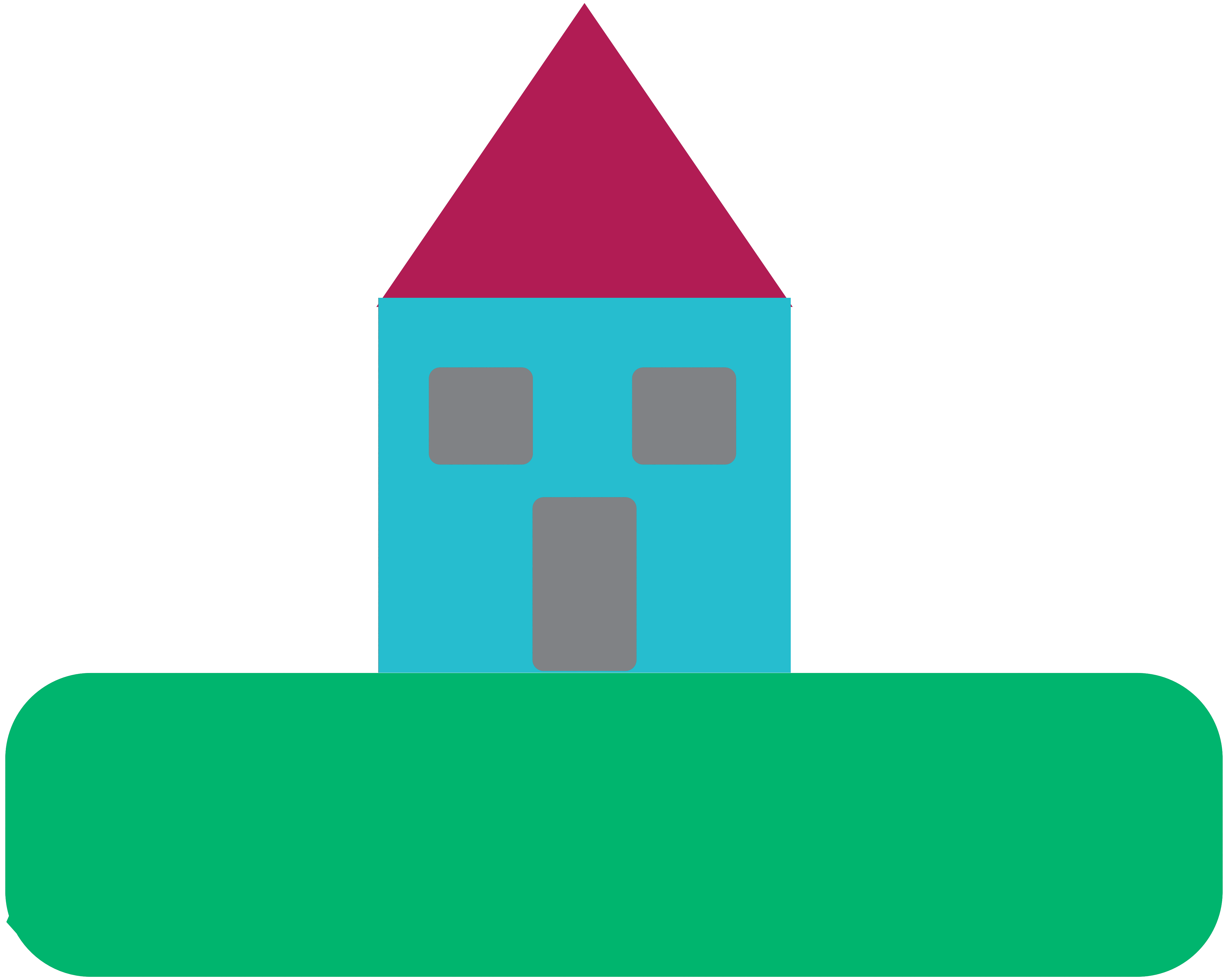


Photo courtesy of https://cdn.pixabay.com/photo/2014/03/24/17/16/brain-295291_960_720.png





GENERALISE

SPECIALISE

PROBLEM WITH COMPLEXITY

Slows down future work

Harder to onboard new people

Harder to come back to after an absence

Negatively affects response in an emergency

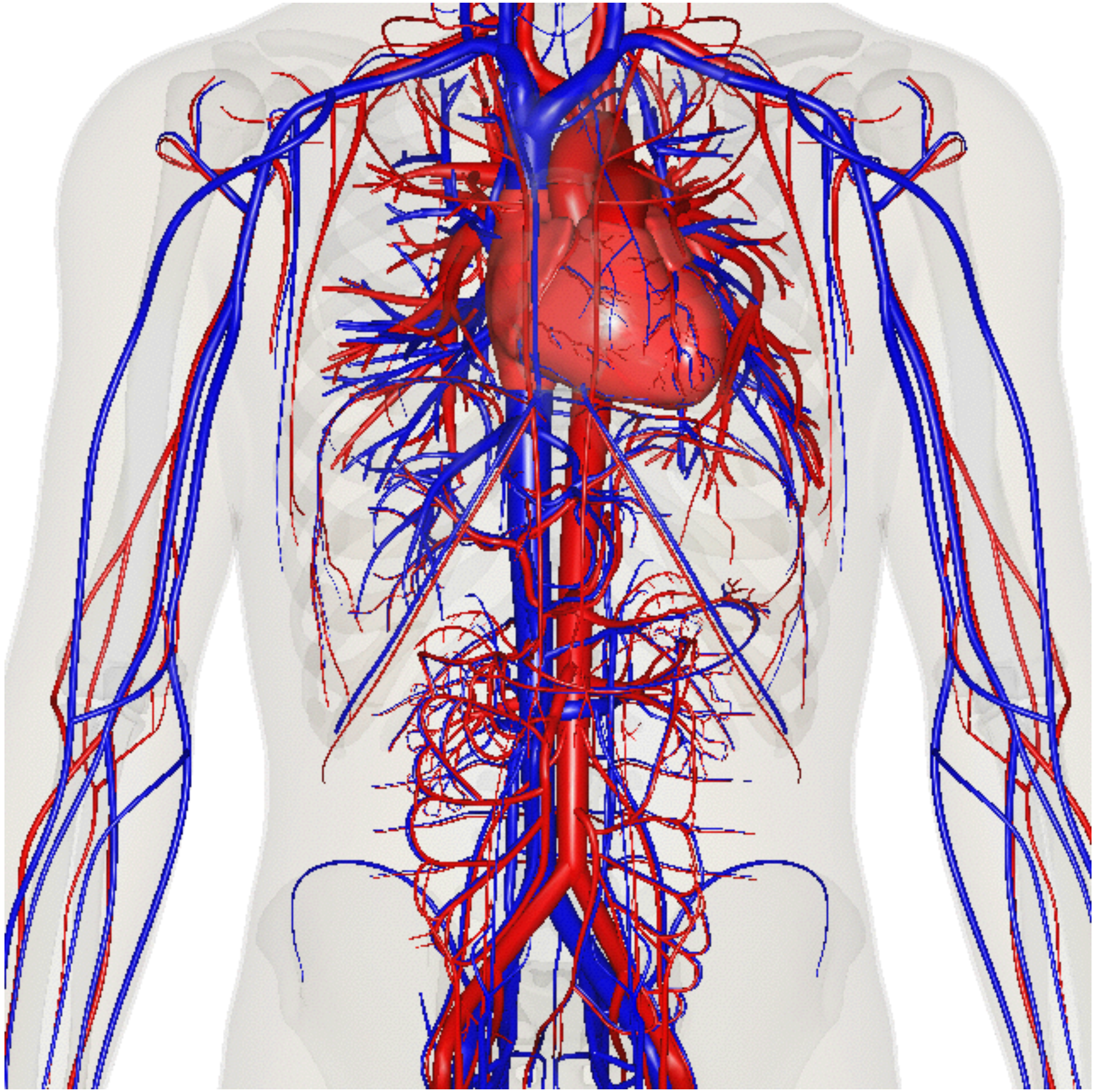


Photo courtesy of https://commons.wikimedia.org/wiki/File:Vein_art_near.png

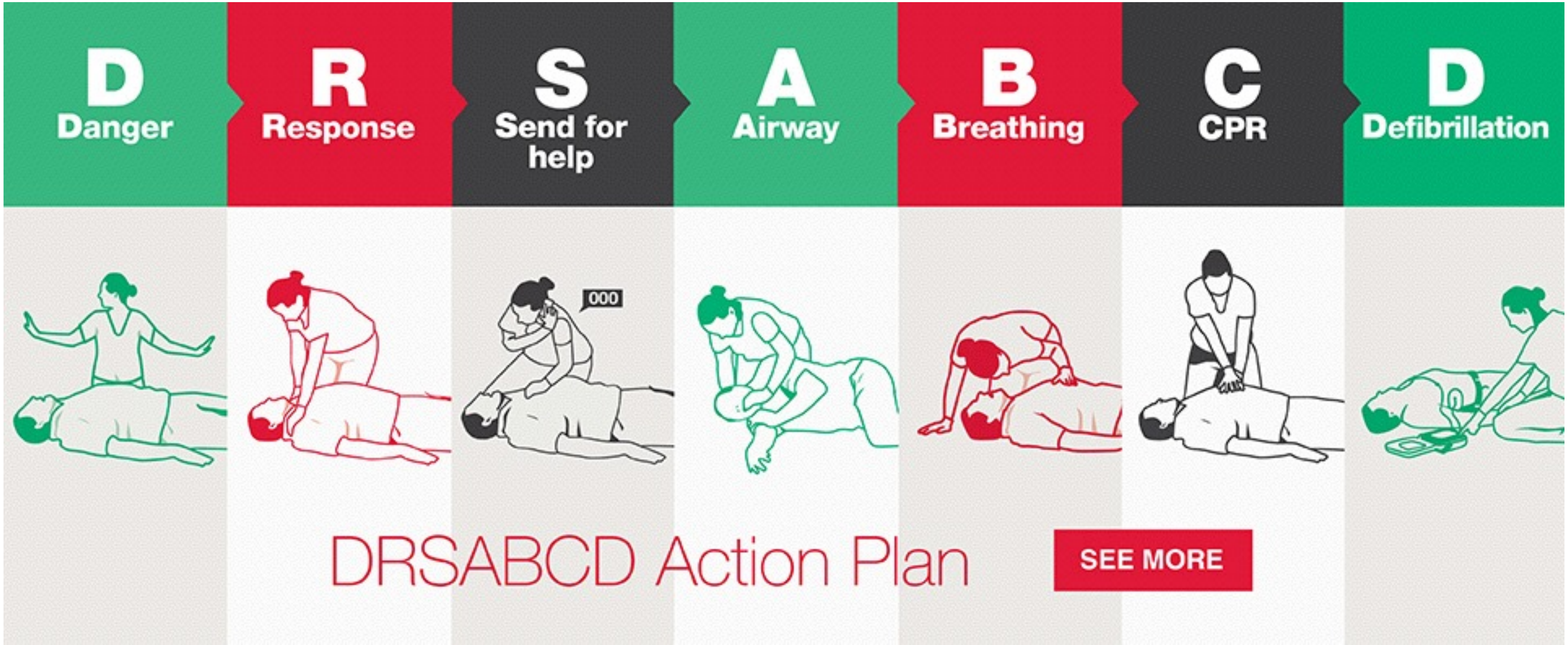


Photo courtesy of http://www.stjohnnsw.com.au/

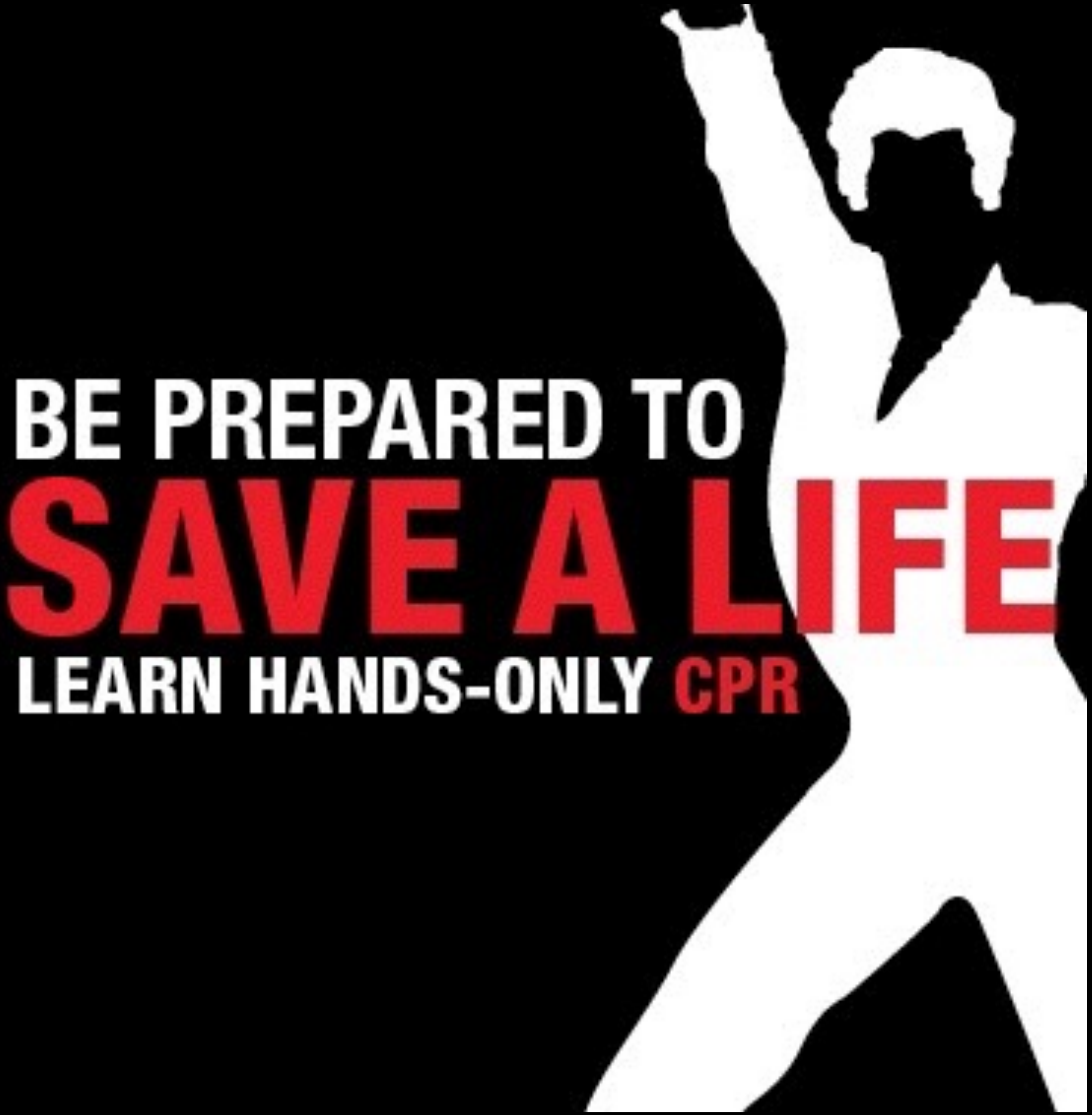


Photo courtesy of https://www.crowdrise.com/handsonlycpr

Photo courtesy of https://upload.wikimedia.org/wikipedia/commons/7/77/Philips_Automated_External_Defibrillator_in_Ta

SIMPLE CODE

“The function of good software is to make the complex appear to be simple.”

- Grady Booch

Photo courtesy of https://upload.wikimedia.org/wikipedia/commons/d/d9/Edsger_Wybe_Dijkstra.jpg

WONDERFUL GAME

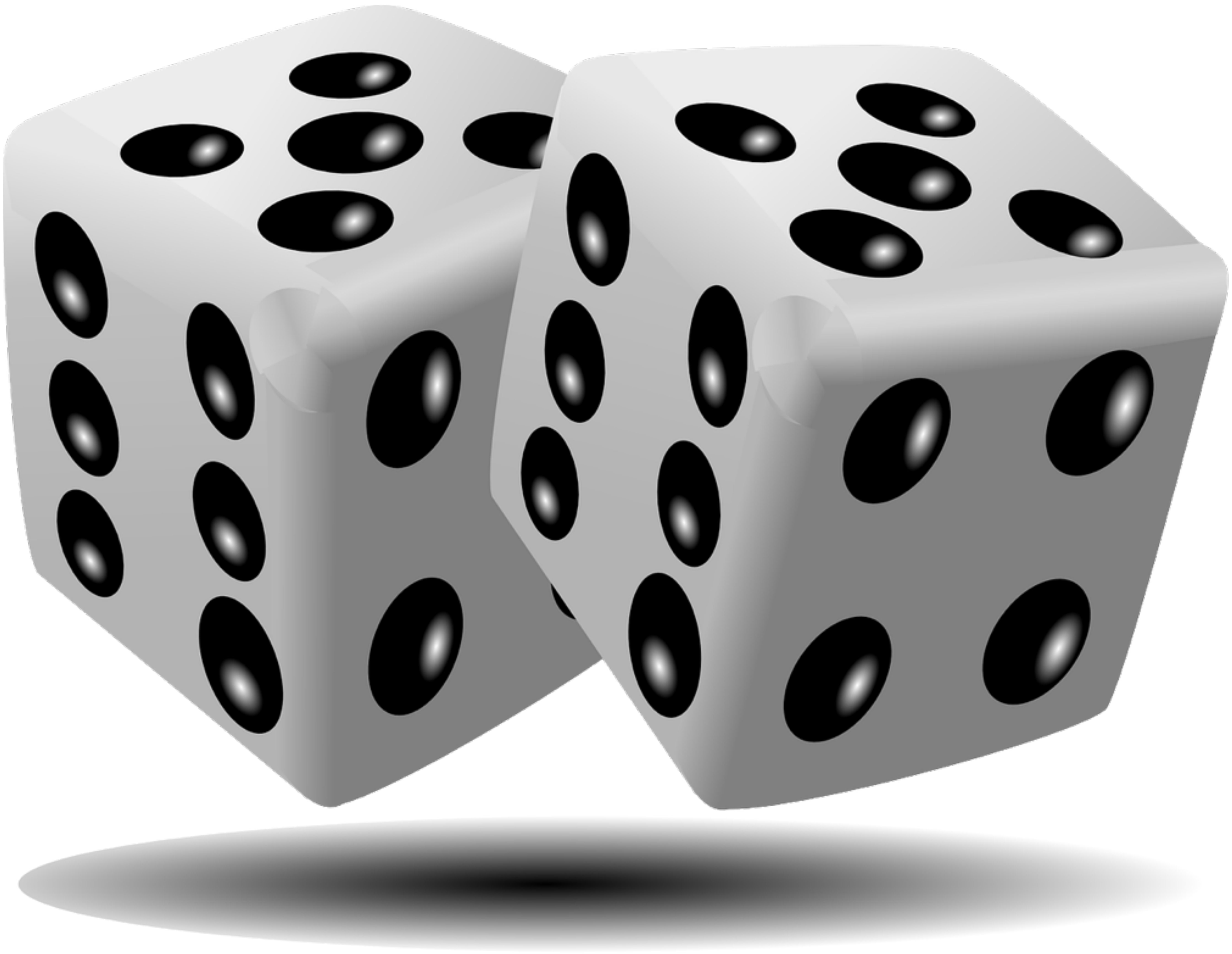
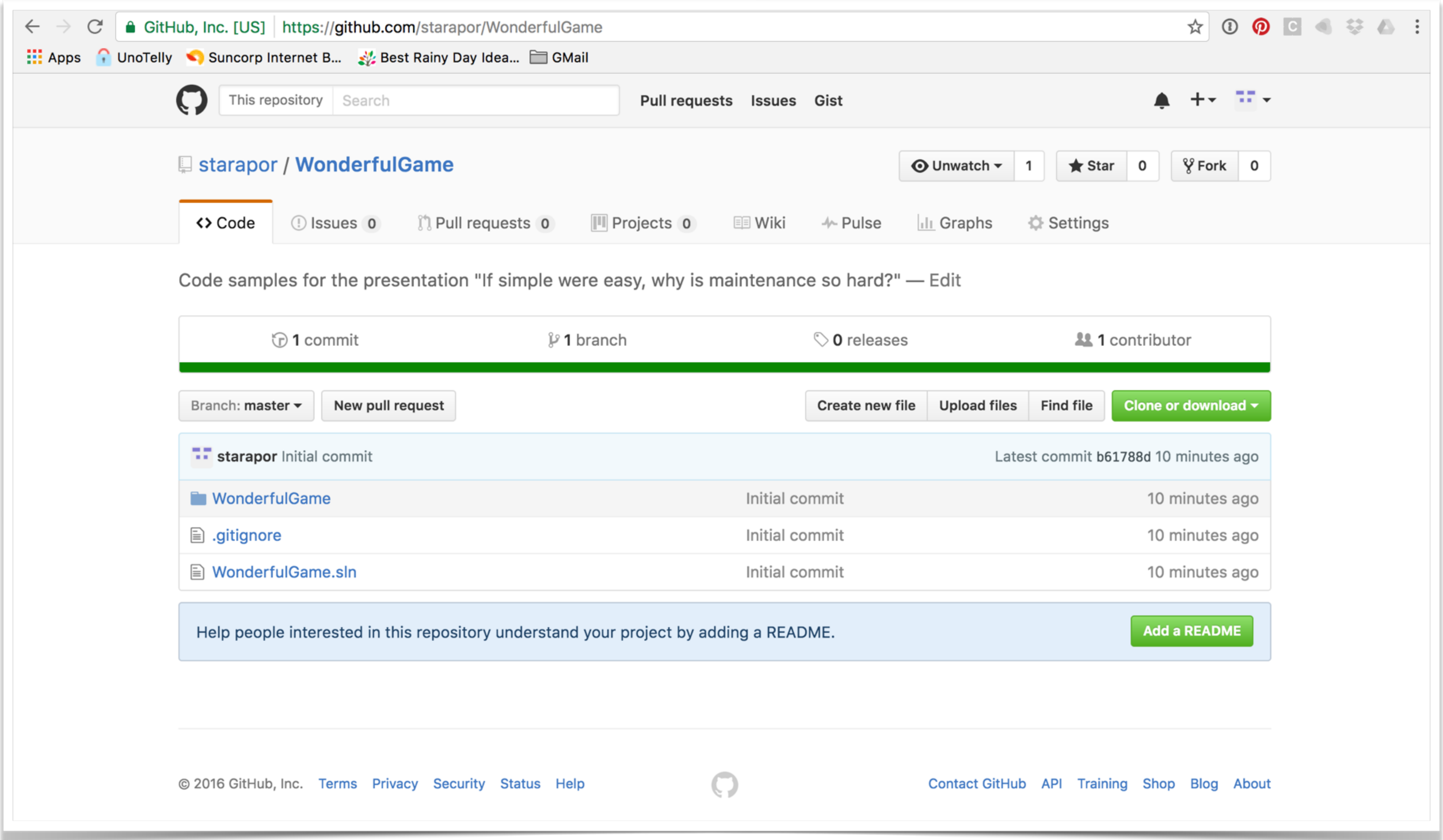


Photo courtesy of https://pixabay.com/en/dices-game-gambling-cubes-numbers-1600005/



starapor / WonderfulGame

Code samples for the presentation "If simple were easy, why is maintenance so hard?" — Edit

1 commit 1 branch 0 releases 1 contributor

starapor Initial commit Latest commit b61788d 10 minutes ago

WonderfulGame	Initial commit	10 minutes ago
.gitignore	Initial commit	10 minutes ago
WonderfulGame.sln	Initial commit	10 minutes ago

Help people interested in this repository understand your project by adding a README.

© 2016 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

IF ELSE...

```
public void TakeTurn(Player player)
{
    if (player.InPenaltyBox())
        ReducePenaltyTime();
    else
        PlayARound();
}
```

… GROWS TO …

```
public void TakeTurn(Player player)
{
    if (player.InPenaltyBox())
        ReducePenaltyTime();
    else if (player.WearingAHat())
        RemoveTheHat();
    else
        PlayARound();
}
```

... BUT MATCHERS ARE BETTER

```
{  p => p.InPenaltyBox(), ReducePenaltyTime  },
{  p => p.WearingAHat(),  RemoveTheHat  },
{  p => true,             PlayARound }
```

… BUT MATCHERS ARE BETTER

```
private readonly Matcher<Player> _matcher;

public Game()
{
    _matcher = new Matcher<Player>
    {
        {  p => p.InPenaltyBox(), ReducePenaltyTime  },
        {  p => p.WearingAHat(),  RemoveTheHat  },
        {  p => true,             PlayARound }
    };
}

public void TakeTurn(Player player)
{
    _matcher.DoActionThatMatches(player);
}
```

… BUT MATCHERS ARE BETTER

```
public class Matcher<T> : IEnumerable<KeyValuePair<Predicate<T>, Action>>
{
    private readonly IDictionary<Predicate<T>, Action> _matcher
                = new Dictionary<Predicate<T>, Action>();

    public void DoActionThatMatches(T t)
    {
        _matcher.First(m => m.Key(t)).Value();
    }

    public void Add(Predicate<T> predicate, Action action)
    {
        _matcher.Add(predicate, action);
    }

}
```

PRIMITIVES ...

```
public Player(string name)
{
    _name = name;
}
```

… GROWS TO …

```
public Player(string name, string nickname, string email)
{
    _name = name;
    _nickname = nickname;
    _email = email;
}
```

… BUT TINY TYPES ARE BETTER

```
public Player(Name name, Nickname nickname, Email email)
{
    _name = name;
    _nickname = nickname;
    _email = email;
}
```

TINY TYPES

```
public class Name : TinyType<string>
{ public Name(string value) : base(value) { } }

public class Nickname : TinyType<string>
{ public Nickname(string value) : base(value) { } }

public class Email : TinyType<string>
{ public Email(string value) : base(value) { } }
```

TINY TYPES

```
public class TinyType<T>
{
    private readonly T _value;

    public TinyType(T value)
    {
        _value = value;
    }

    public static implicit operator T(TinyType<T> tt)
    {
        return tt._value;
    }
    public static implicit operator TinyType<T>(T value)
    {
        return new TinyType<T>(value);
    }
}
```

FAT OBJECTS ...

```
public interface IRuleHandler
{
    void NotWearingAHatHandler();
}
```

… GROW TO …

```
public interface IRuleHandler
{
    void NotWearingAHatHandler();
    void SpinAgainHandler();
    void JumpUpAndDownHandler();
    void DoMoreCoolStuffHandler();
}
```

… BUT SRP IS BETTER

```
public interface NotWearingAHatHandler  { void Handle(); }
public interface SpinAgainHandler       { void Handle(); }
public interface JumpUpAndDownHandler   { void Handle(); }
public interface DoMoreCoolStuffHandler { void Handle(); }
```

MIXED CONCERNS...

```
class Dice : IDice
{
    private readonly Random _random;
    public Dice(Random random) {_random = random;}

    public int Roll()
    {
        var randomNumber =  _random.Next(1,6);
        return randomNumber;
    }
}
```

… GROWS TO …

```
public Dice(Random random, ILogger logger, bool skipEverySecondRoll)
{
    _random = random;
    _logger = logger;
    _skipEverySecondRoll = skipEverySecondRoll;
}

public int Roll()
{
    var randomNumber =  _random.Next(1,6);
    _logger.Log("We just rolled {0}", randomNumber);
    if (_skipEverySecondRoll)
    {
     randomNumber = _random.Next(1, 6);
    }
    return randomNumber;
}
```

… BUT DECORATORS ARE BETTER

```
public interface IDice
{
    int Roll();
}
```

... BUT DECORATORS ARE BETTER

```
class Dice : IDice
{
    private readonly Random _random;
    public Dice(Random random)
    {
        _random = random;
    }

    public int Roll()
    {
        return _random.Next(1,6);
    }
}
```

… BUT DECORATORS ARE BETTER

```
class SkipEverySecondRoll : IDice
{
    private readonly IDice _dice;
    public SkipEverySecondRoll(IDice dice)
    {
        _dice = dice;
    }
    public int Roll()
    {
        var ignoreThisValue = _dice.Roll();
        return _dice.Roll();
    }
}
```

... BUT DECORATORS ARE BETTER

```
public DiceLogger(IDice dice, ILogger logger)
{
    _dice = dice;
    _logger = logger;
}

public int Roll()
{
    var rolledNumber = _dice.Roll();
    _logger.Log("We just rolled an awesome random number {0}",
        rolledNumber);
    return rolledNumber;
}
```

… BUT DECORATORS ARE BETTER

```
builder.RegisterType<Dice>().Named<IDice>("real");

builder.RegisterDecorator<IDice>((c, inner)
    => new SkipEverySecondRoll(inner),
        fromKey: "implementor", toKey: "decorator1");

builder.RegisterDecorator<IDice>((c, inner)
    => new DiceLogger(inner, new Logger()),
        fromKey: "decorator1", toKey: null);
```

BOY SCOUT RULE

Photo courtesy of http://disney.wikia.com/wiki/Russell

GOOD CITIZEN

Trailing commas in lists

```
    _matcher = new Matcher<Player>
    {
        {  p => p.InPenaltyBox(), ReducePenaltyTime  },
        {  p => p.WearingAHat(),  RemoveTheHat  },
        {  p => true,             PlayARound },
    };
```

Brackets { } around conditionals

```
      public void TakeTurn(Player player)
      {
          if (player.InPenaltyBox())
          {
              ReducePenaltyTime();
          }
      }
```

SQUINT TEST

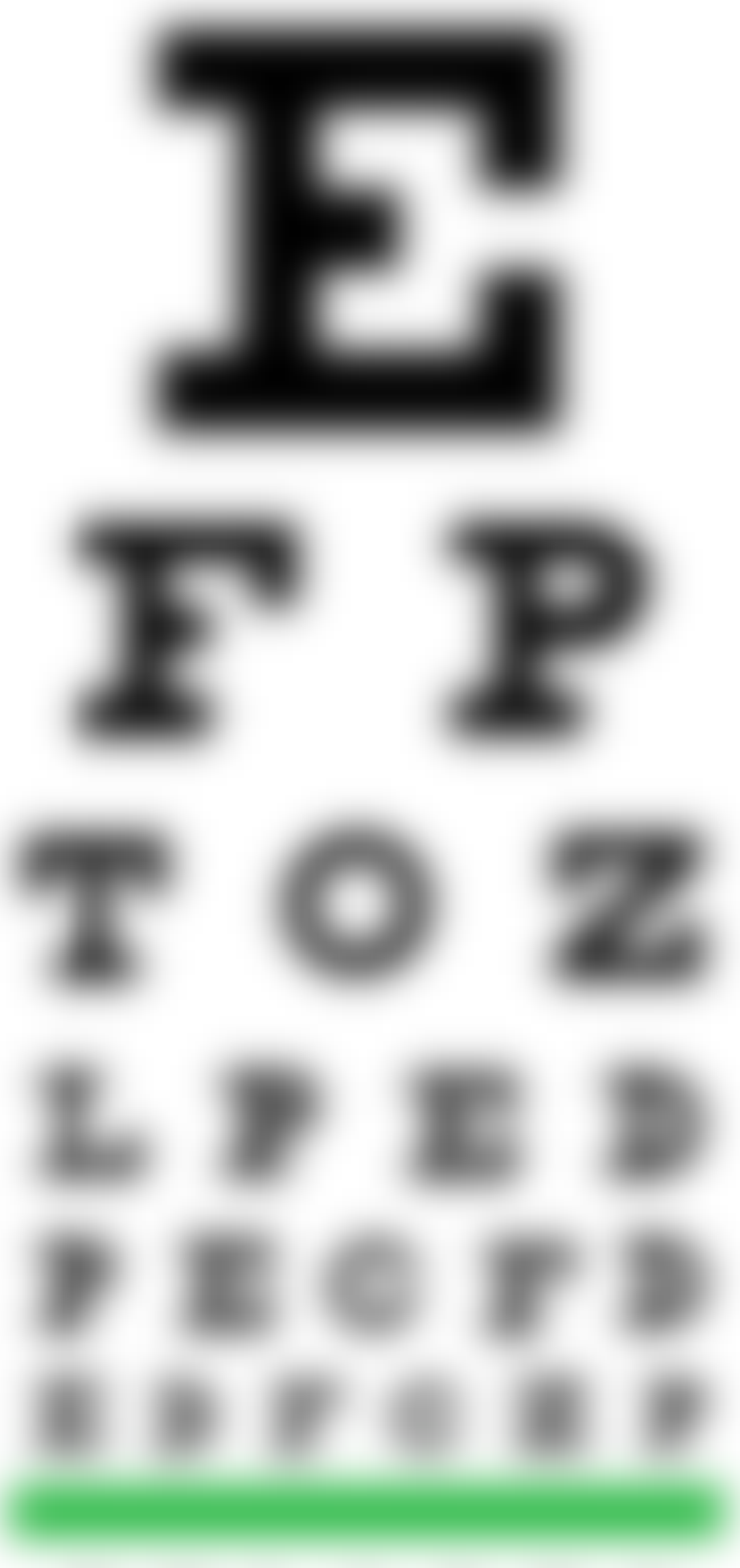
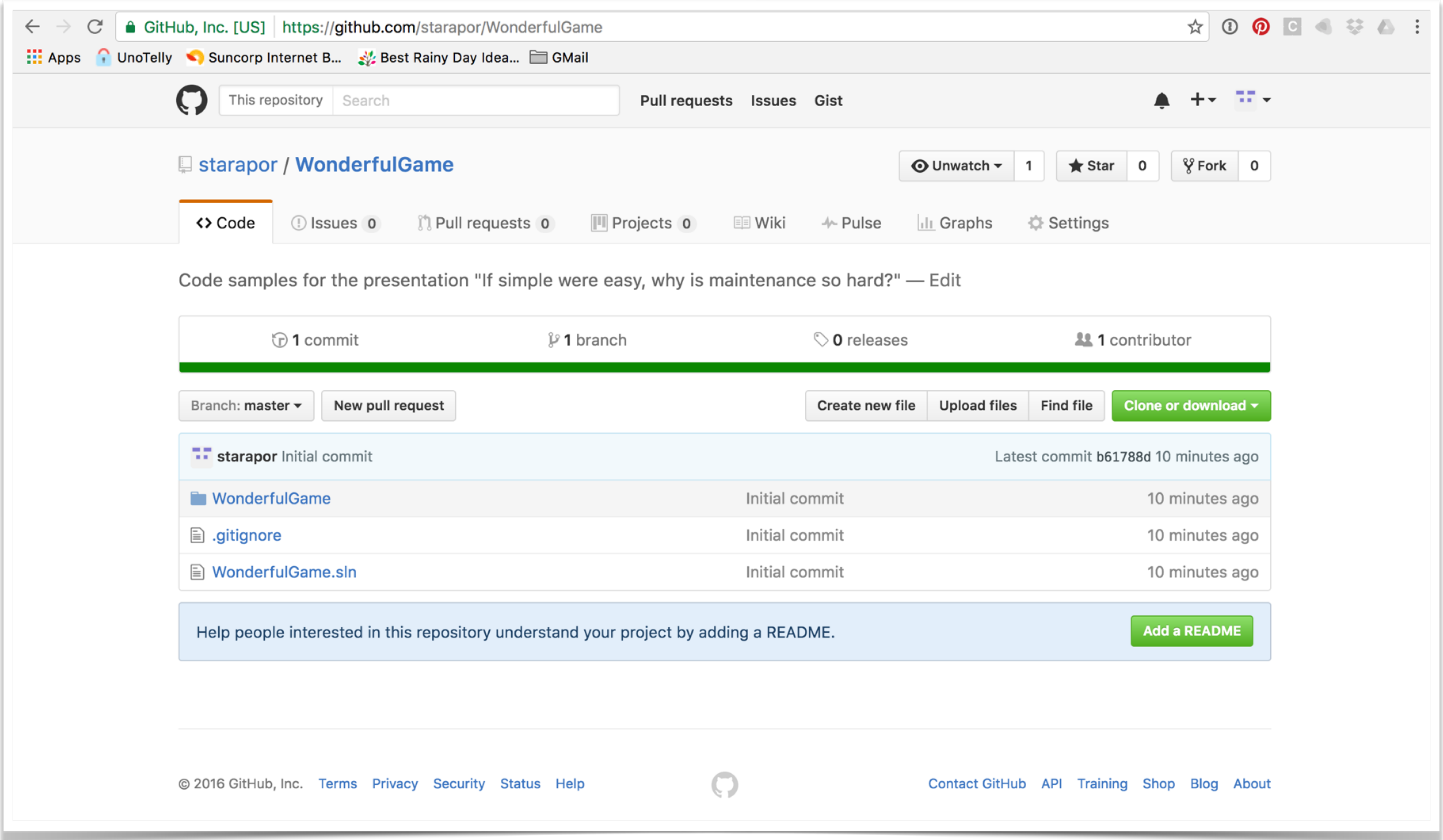


Photo courtesy of https://upload.wikimedia.org/wikipedia/en/7/7b/Snellen-myopia.png

USE CONVENTIONS UNTIL YOU DON’T

```
IFooHandler     => IHandleLotsOfFoos
IGameRepository => ICanFindTheGame
```



https://github.com/starapor/WonderfulGame

starapor / WonderfulGame

Code samples for the presentation "If simple were easy, why is maintenance so hard?" — Edit

1 commit | 1 branch | 0 releases | 1 contributor

starapor Initial commit — Latest commit b61788d 10 minutes ago

WonderfulGame	Initial commit	10 minutes ago
.gitignore	Initial commit	10 minutes ago
WonderfulGame.sln	Initial commit	10 minutes ago

Help people interested in this repository understand your project by adding a README.

© 2016 GitHub, Inc.

ThoughtWorks®

SIMPLE EXPLANATIONS

“I have made this longer than usual because I have not had time to make it shorter.”

Photo courtesy of https://upload.wikimedia.org/wikipedia/commons/6/66/Blaise_Pascal_works_p6.png

EXPLAIN YOUR SYSTEM

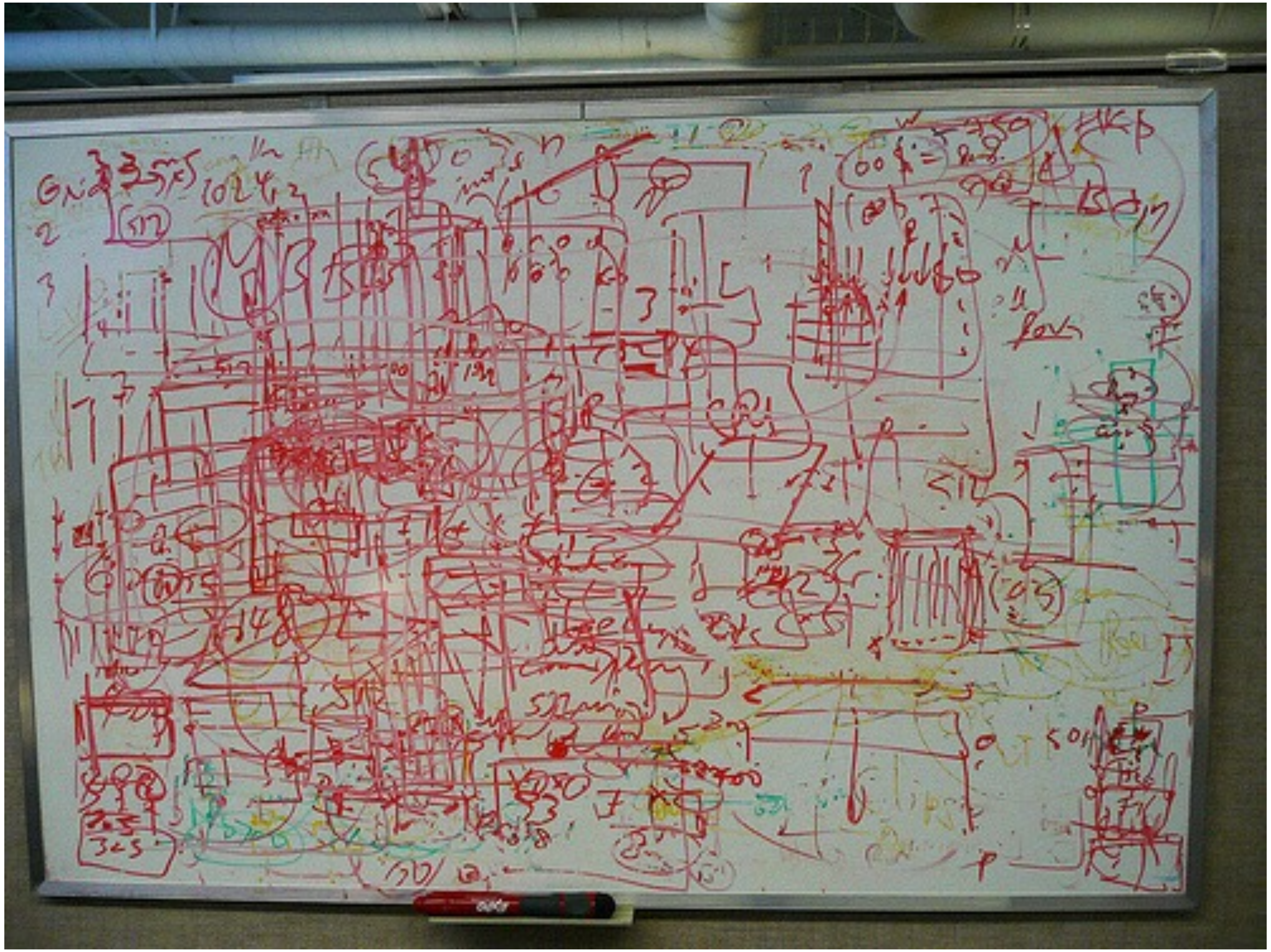


Photo courtesy of http://evan.bottch.com/wp-content/uploads/2010/06/whiteboard.jpg

HOW?

Use a whiteboard where possible

Find ways that a developer can relate to the concepts

Not too much, too early

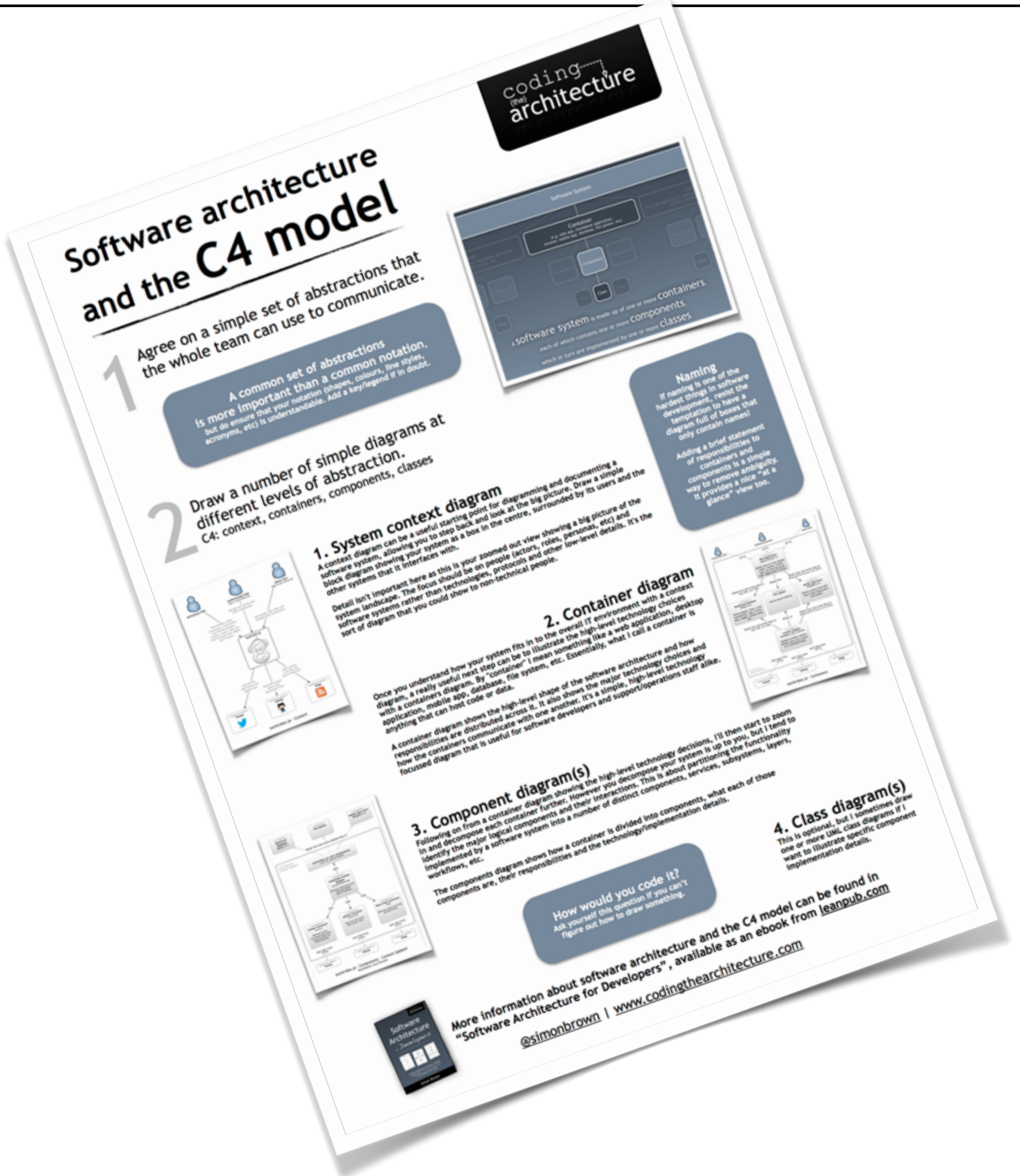
Iterate (revisit this after having had some code)

See Simon Brown’s C4 model

Use diagramming/modelling syntax consistently

No need for strict UML - as long as everyone shares the same understanding

SIMON BROWN'S C4 MODEL


coding (the) architecture
Software System
Container
Component
Class
Software Architecture for Developers

Software architecture and the C4 model

1 Agree on a simple set of abstractions that the whole team can use to communicate.

A common set of abstractions is more important than a common notation, but do ensure that your notation (shapes, colours, line styles, acronyms, etc) is understandable. Add a key/legend if in doubt.

A software system is made up of one or more containers, each of which contains one or more components, which in turn are implemented by one or more classes.

Naming
If naming is one of the hardest things in software development, resist the temptation to have a diagram full of boxes that only contain names!

Adding a brief statement of responsibilities to containers and components is a simple way to remove ambiguity. It provides a nice "at a glance" view too.

2 Draw a number of simple diagrams at different levels of abstraction.
C4: context, containers, components, classes

1. System context diagram

A context diagram can be a useful starting point for diagramming and documenting a software system, allowing you to step back and look at the big picture. Draw a simple block diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interfaces with.

Detail isn't important here as this is your zoomed out view showing a big picture of the system landscape. The focus should be on people (actors, roles, personas, etc) and software systems rather than technologies, protocols and other low-level details. It's the sort of diagram that you could show to non-technical people.

2. Container diagram

Once you understand how your system fits in to the overall IT environment with a context diagram, a really useful next step can be to illustrate the high-level technology choices with a containers diagram. By "container" I mean something like a web application, desktop application, mobile app, database, file system, etc. Essentially, what I call a container is anything that can host code or data.

A container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another. It's a simple, high-level technology focussed diagram that is useful for software developers and support/operations staff alike.

3. Component diagram(s)

Following on from a container diagram showing the high-level technology decisions, I'll then start to zoom in and decompose each container further. However you decompose your system is up to you, but I tend to identify the major logical components and their interactions. This is about partitioning the functionality implemented by a software system into a number of distinct components, services, subsystems, layers, workflows, etc.

The components diagram shows how a container is divided into components, what each of those components are, their responsibilities and the technology/implementation details.

4. Class diagram(s)

This is optional, but I sometimes draw one or more UML class diagrams if I want to illustrate specific component implementation details.

How would you code it?
Ask yourself this question if you can't figure out how to draw something.

More information about software architecture and the C4 model can be found in "Software Architecture for Developers", available as an ebook from leanpub.com

@simonbrown | www.codingthearchitecture.com

METAPHORS

ELEVATOR PITCH

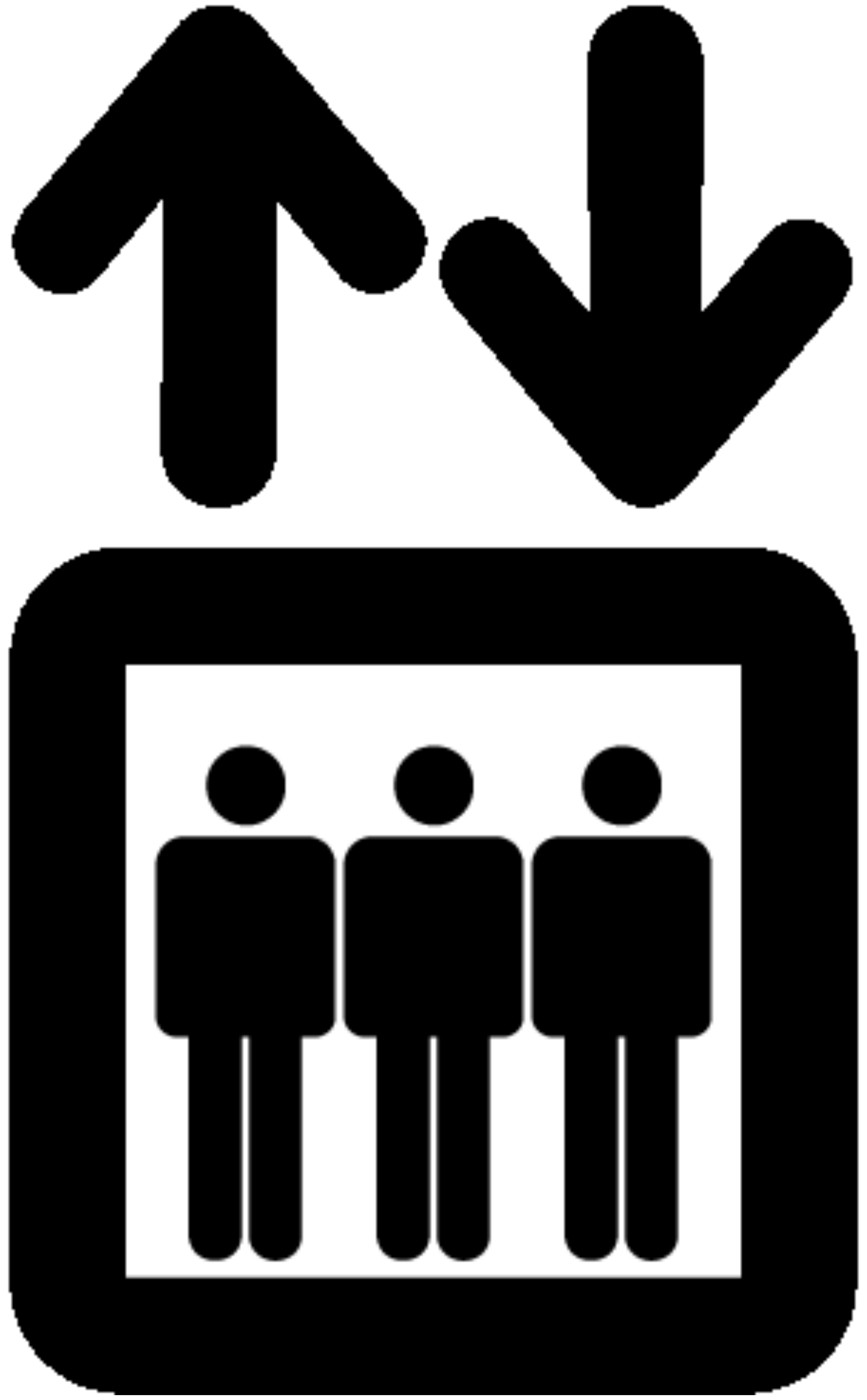


Photo courtesy of https://commons.wikimedia.org/wiki/File:Aiga_elevator.gif

WALKING SKELETON

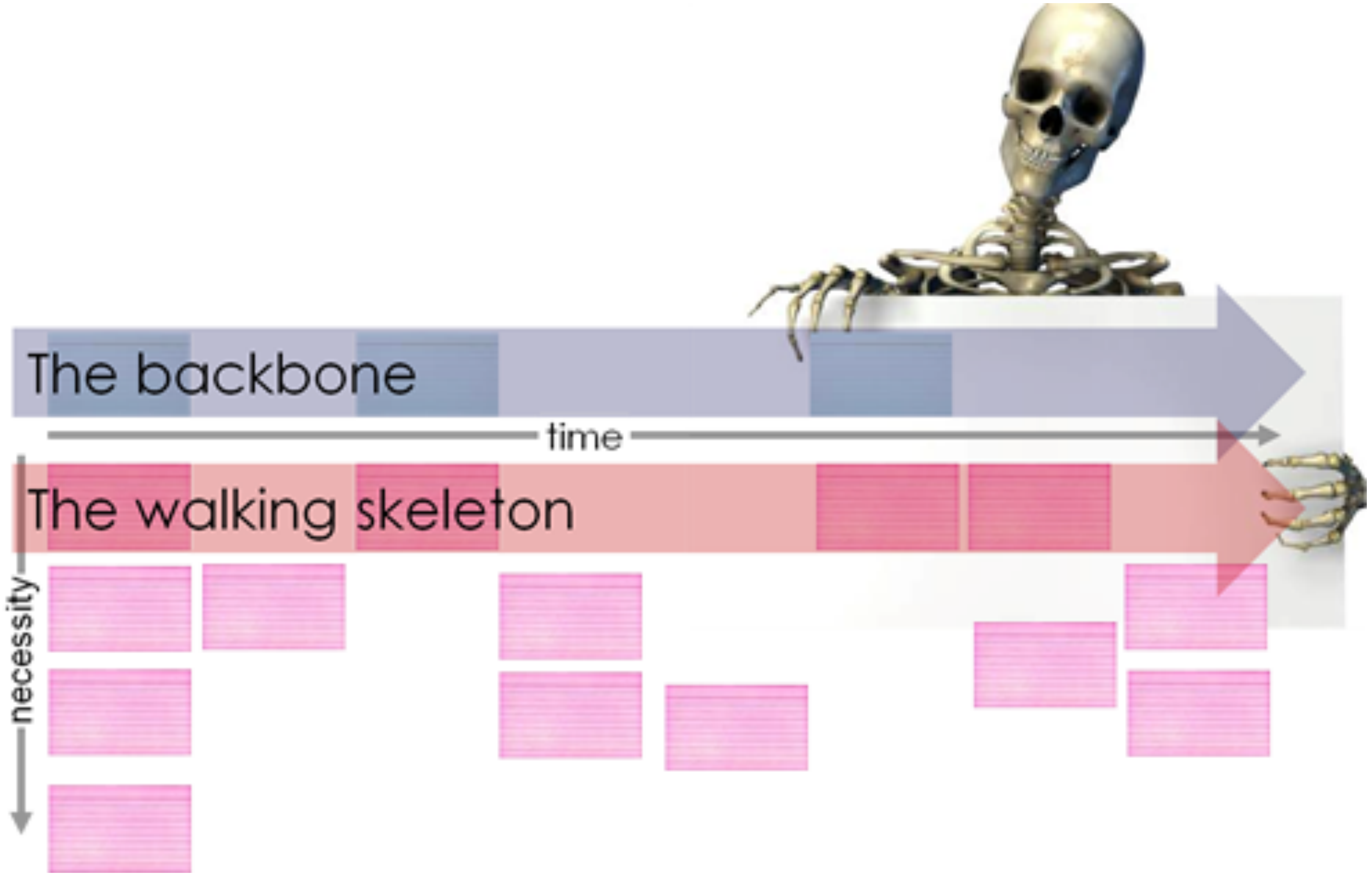
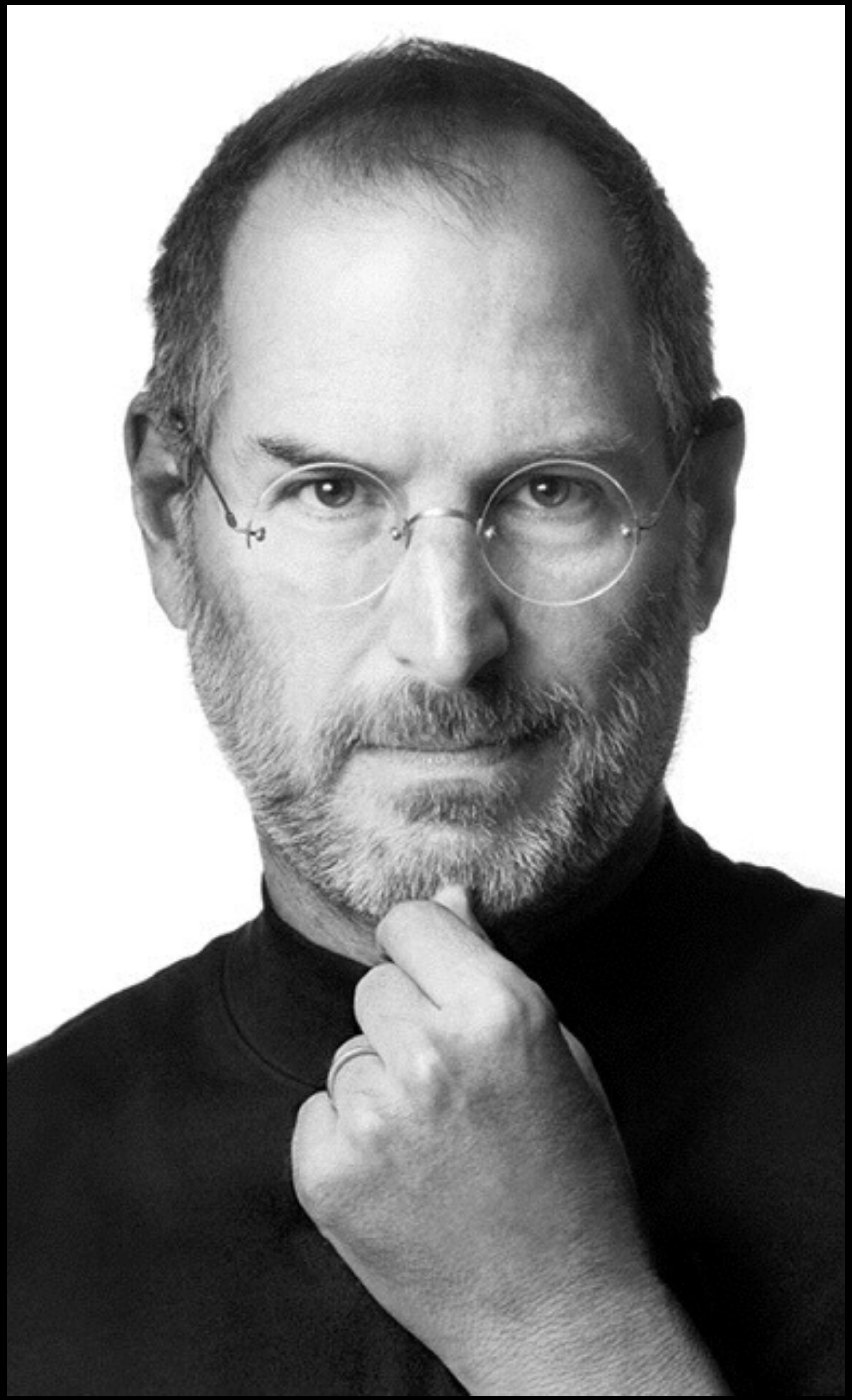


Photo courtesy of http://jpattonassociates.com/the-new-backlog/

EXPLAIN TO A 3 YEAR OLD

ThoughtWorks®

SUMMARY



"That's been one of my mantras - focus and simplicity.

Simple can be harder than complex: You have to work hard to get your thinking clean to make it simple. But it's worth it in the end because once you get there, you can move mountains."

Photo courtesy of https://upload.wikimedia.org/wikipedia/commons/6/66/Blaise_Pascal_works_p6.png

TODAY WE EXPLORED

Simple and Complex

Simple Code

Simple Explanations

PLEASE SUPPORT OUR SPONSORS

TattsGroup

ThoughtWorks®

gooroo
next level

readify

NDC { Sydney }
14-18 August 2017

Octopus Deploy

Wardy it
solutions

DevExpress™

Microsoft

ASPOSE
File Format APIs

avanade®
Results Realized

GitHub

ROBERT WALTERS
SPECIALIST PROFESSIONAL RECRUITMENT

Seq

Cloud Ctrl
CREATED BY SixPivot

FILL OUT YOUR FEEDBACK

To go into the draw for prizes, please remember to complete your feedback at:

http://www.dddbrisbane.com/feedback

No feedback = No Prizes!

THANK YOU

Sarah Taraporewalla
www.sarahtaraporewalla.com

ThoughtWorks®