INFINUM

WHY BOTHER WITH CODE CHECKING?

DENIS ŽOLJOM
DENIS.ZOLJOM@INFINUM.CO

We're an independent
design & development
agency.

$$\infty \longrightarrow \boxed{8}$$

WHAT WILL YOU GET?

What are coding standards,

and how can we use them to make our code even better

Better code === happy developer :)

01

WHAT ARE CODING STANDARDS?

Coding standards, or conventions, are set of programming styles, practices and rules for each aspect of a program written in a certain programming language.

WIKIPEDIA*

***ALWAYS CITE YOUR SOURCES :D**

They come in all shapes and sizes

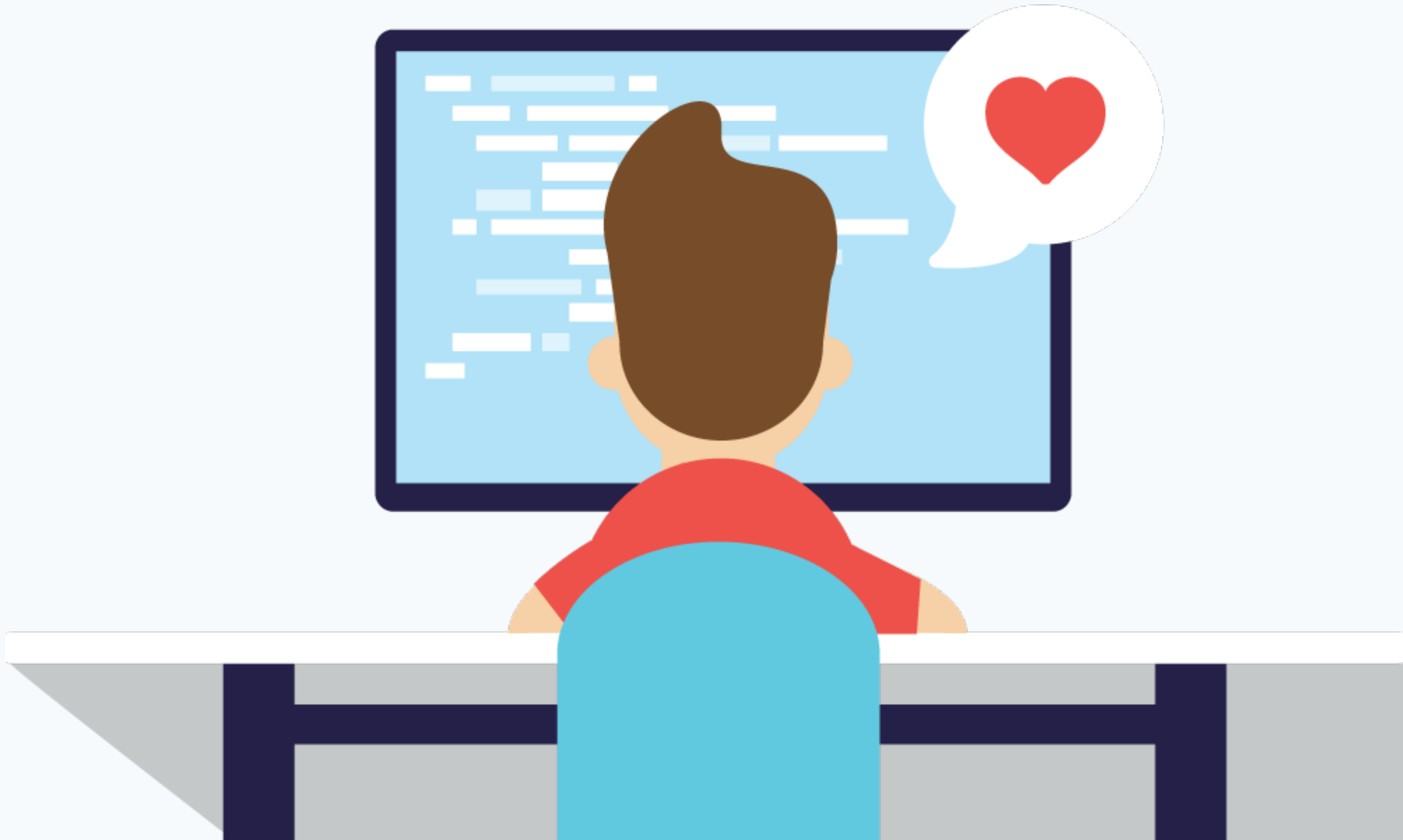
Programming language – PSR

Libraries – jQuery

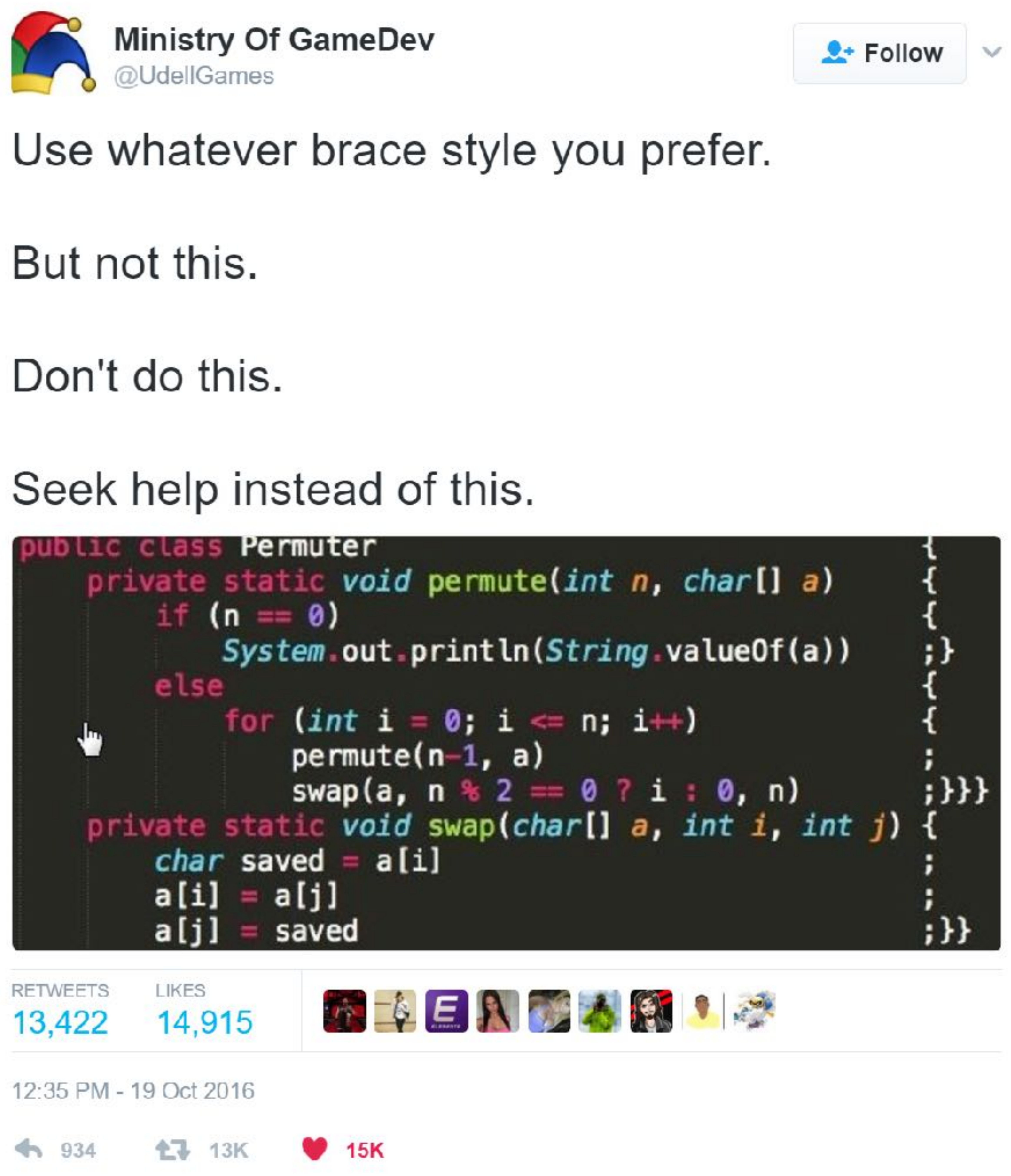
Frameworks – Zend, Symfony

Company – Airbnb, Infinum

Software – WordPress



Basically they tell you how NOT to code



Ministry Of GameDev
@UdellGames

Follow

Use whatever brace style you prefer.

But not this.

Don't do this.

Seek help instead of this.

```
public class Permuter                                           {
    private static void permute(int n, char[] a)                {
        if (n == 0)                                             {
            System.out.println(String.valueOf(a))               ;}
        else                                                    {
            for (int i = 0; i <= n; i++)                        {
                permute(n-1, a)                                 ;
                swap(a, n % 2 == 0 ? i : 0, n)                  ;}}}
    private static void swap(char[] a, int i, int j)            {
        char saved = a[i]                                       ;
        a[i] = a[j]                                             ;
        a[j] = saved                                            ;}}
```

RETWEETS 13,422 LIKES 14,915

12:35 PM - 19 Oct 2016

934 13K 15K

02

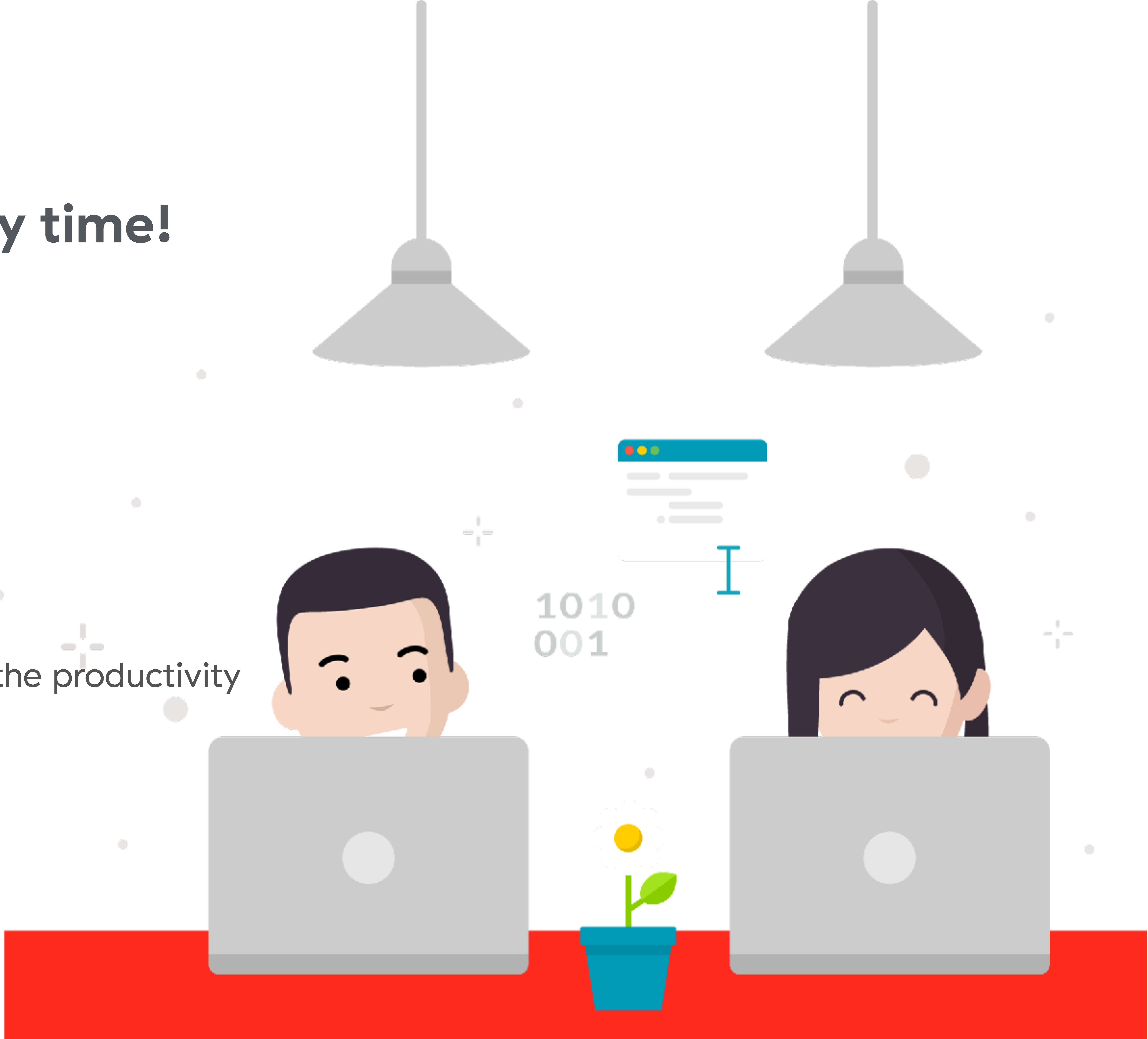
WHY DO WE FOLLOW CODING STANDARDS?

What's the point? Story time!

One project and multiple developers

Real life will happen

Lack of consistency ultimately hurts the productivity



```php
<?
if(has_post_thumbnail() == true)
{
  echo '<h1>'.get_the_title().'</h1>';
  echo get_the_post_thumbnail();
}
else {
  echo '<h1>'.get_the_title().'</h1>';
}
```

```php
<?php if( has_post_thumbnail() === true) : ?>
  <h1><?php the_title(); ?></h1>
  <?php the_post_thumbnail();
else: ?>
  <h1><?php the_title(); ?></h1>
<?php endif; ?>
```

Virtually identical code, but drastically different coding style

And what are the **benefits**?

Better readability

Code consistency and uniformity – important in huge projects such as WordPress

Easier maintainability – saves time and money

Easier code review

Software **maintenance**

Maintenance typically consumes 40 to 80 percent of software costs.

Richard Glass, Facts and Fallacies of Software Engineering

WordPress Coding Standards (WPCS)

Huge open source system – consistency is the key

HTML/CSS

Write valid HTML code

PHP

WordPress-Core
WordPress-Docs
WordPress-Extra
WordPress-VIP
WordPress-Theme*

JAVASCRIPT

Modified jQuery standards

03

AUTOMATION

Automation, automation, automation

Manual reviews are necessary but take time

Better code means less time spent on manual reviews

The benefit of being a programmer – or how I made my computer do all the dreary work for me

04

JAVASCRIPT

Linters

Static analysers

Check suspicious code without execution

JSLint, JSHint, ESLint (the best one)

Many configurations already exist:

```
npm install eslint-config-airbnb
```

```
{
  "globals": {
    "__VERSION__": true
  },
  "env": {
    /**
     * You can change environment depending on the project
you are working on
     */

    "browser": true,
    "node": true,
    "commonjs": true,
    "es6": true,
    "jquery": true
  },
    "rules": {
        /**
         * Possible errors
         */

        "comma-dangle": [1, "never"],
        "no-cond-assign": [2, "always"],
        "no-console": 1,
        "no-constant-condition": 1,
        "no-control-regex": 1,
        "no-debugger": 1,
       ...

        /**
        * Best practices
        */

        "accessor-pairs": 1,
        "array-callback-return": 2,
        "block-scoped-var": 2,
       ...
```

```
      "curly": [2, "all"],
      "default-case": 2,

      /**
      * Stylistic Issues
      */

      "array-bracket-spacing": [2, "never"],
      "block-spacing": 2,
      "brace-style": [2,
        "1tbs", {
        "allowSingleLine": false
      }],
      "camelcase": [2, {"properties": "always"}],
      ...
    }
  }
```

https://eslint.org/docs/user-guide/configuring

05

PHP

Linters, Sniffers, Fixers

Variety of tools to choose from

PHP Linter – ok, but harder to customise

PHP Code fixer – recommended if you want to follow PSR standards

PHP_CodeSniffer – ideal tool for WordPress code

PHP_CodeSniffer (PHPCS)

Static code checker

Better than Theme Check – text parser cannot tell valid php code

Has built-in automatic code fixer – phpcbf

Developed by SquizLabs, forked by WordPress

2016 – golden year for WPCS – Juliette Reinders Folmer

Takes your files one at the time

Tokenizes your PHP code into tokens

Easier to check specific tokens

Cannot help you with certain policy guidelines – rtl check for instance

Easy integration with IDE – PHPStorm, Sublime Text 3, Atom and Visual Studio

You can add your own rulesets \O/

Let's see an example

```php
<?php
/**
 * This is a test file
 *
 * @package  Test
 */

$name = 'Mittens';

/**
 * A function that takes a name and returns a happy message
 *
 * @param  str $var Input string.
 * @return str      Modified string.
 */
function happy_message( $var ) {
  return $var . 'is awesome!';
}

happy_message( $name ); // Should return 'Mittens is awesome!'.
```

```
Array
(
    [0] => Array
        (
            [type] => T_OPEN_TAG
            [code] => 374
            [content] => <?php

            [line] => 1
            [column] => 1
            [length] => 5
            [level] => 0
            [conditions] => Array
                (
                )
        )
```

```
[1] => Array
    (
        [content] => /**
        [code] => PHPCS_T_DOC_COMMENT_OPEN_TAG
        [type] => T_DOC_COMMENT_OPEN_TAG
        [comment_tags] => Array
            (
                [0] => 14
            )

        [comment_closer] => 19
        [line] => 2
        [column] => 1
        [length] => 3
        [level] => 0
        [conditions] => Array
            (
            )
    )
    [35] => Array
        (
            [content] => A function that takes a name
and returns a happy message
            [code] => PHPCS_T_DOC_COMMENT_STRING
            [type] => T_DOC_COMMENT_STRING
            [line] => 11
            [column] => 4
            [length] => 67
            [level] => 0
            [conditions] => Array
                (
                )
        )
)
```

```
[57] => Array
        (
            [code] => 335
            [type] => T_FUNCTION
            [content] => function
            [line] => 16
            [column] => 1
            [length] => 8
            [parenthesis_opener] => 60
            [parenthesis_closer] => 64
            [parenthesis_owner] => 57
            [scope_condition] => 57
            [scope_opener] => 66
            [scope_closer] => 78
            [level] => 0
            [conditions] => Array
                (
                )

        )
```

Let's write our own sniff!

An example from Infinum WordPress Coding Standards

Don't use do_shortcode() function, unless you really need to

```
\[(\[?)(wp_caption|caption|gallery|playlist|audio|video|embed)(?![\w-])([^\]
\/]*(?:\/(?!\])[^\]\/]*)*?)(?:(\/)\]|\](?:([^\[]*+(?:\[(?!\/\2\])[^\[]*+)*+)
\[\/\2\])?)(\]?)
```

First we need to set up our ruleset.xml inside our custom sniff folder

Then we create Sniffs folder and inside Shortcodes folder create DisallowDoShortcodeSniff.php

ruleset.xml

```xml
<?xml version="1.0"?>
<ruleset name="Infinum" namespace="Infinum">
  <description>Infinum internal coding standards for WordPress</
description>

  <!-- <config name="installed_paths" value="vendor/wp-coding-
standards/wpcs"/> -->

  <!-- Exclude node and composer folders -->
  <exclude-pattern>vendor/*</exclude-pattern>
  <exclude-pattern>node_modules/*</exclude-pattern>
  <exclude-pattern>*/data/*</exclude-pattern>

  <!-- Only check php code, linters are checking js and css -->
  <arg name="extensions" value="php"/>

  <!-- Define default report type -->
  <arg name="report" value="full"/>
```

```
<!-- We work with modified VIP standard -->
  <rule ref="WordPress-VIP">
    <exclude name="WordPress.VIP.FileSystemWritesDisallow" />
    <exclude name="WordPress.VIP.RestrictedFunctions" />
    <exclude name="WordPress.VIP.RestrictedVariables" />
    <exclude name="WordPress.VIP.SuperGlobalInputUsage" />
    <exclude name="WordPress.VIP.DirectDatabaseQuery" />
    <exclude name="WordPress.PHP.YodaConditions" />
  </rule>
```

```xml
<!-- Set the default indent value and force spaces instead of tabs -->
  <rule ref="WordPress">
      <exclude name="Generic.WhiteSpace.DisallowSpaceIndent" />
  </rule>
  <rule ref="Generic.WhiteSpace.ScopeIndent">
      <properties>
          <property name="indent" value="2"/>
          <property name="tabIndent" value="false"/>
      </properties>
  </rule>
  <rule ref="Generic.WhiteSpace.DisallowTabIndent" />
```

DisallowDoShortcodeSniff.php

```php
<?php

namespace Infinum\Sniffs\Shortcodes;

use PHP_CodeSniffer_File;
use PHP_CodeSniffer_Sniff;
use PHP_CodeSniffer_Tokens;

/**
 * Ensures do_shortcode() function is not being used.
 */
class DisallowDoShortcodeSniff implements
PHP_CodeSniffer_Sniff {
```

```php
/**
 * Returns an array of tokens this test wants to listen for.
 *
 * @return array
 */
public function register() {

  $tokens = PHP_CodeSniffer_Tokens::$functionNameTokens;

  return $tokens;
}
```

```
/**
 * Processes this sniff, when one of its tokens is encountered.
 *
 * @param PHP_CodeSniffer_File $phpcsFile The file being scanned.
 * @param int                  $stackPtr  The position of the current
token in
 *                                        the token stack.
 *
 * @return void
 */
public function process( PHP_CodeSniffer_File $phpcsFile, $stackPtr ) {

  $tokens = $phpcsFile->getTokens();
  $token  = $tokens[ $stackPtr ];

  if ( preg_match( '`^do_shortcode`i', $token['content'] ) > 0 ) {
    $phpcsFile->addWarning( 'Do not include do_shortcode() function in
theme files. Use shortcode callback function instead.' , $stackPtr,
'do_shortcodeDetected' );
  }
}
```

Testing!

Create a sample php file and run the PHPCS on it

```
phpcs --standard=Infinum test.php
```

Create unit tests

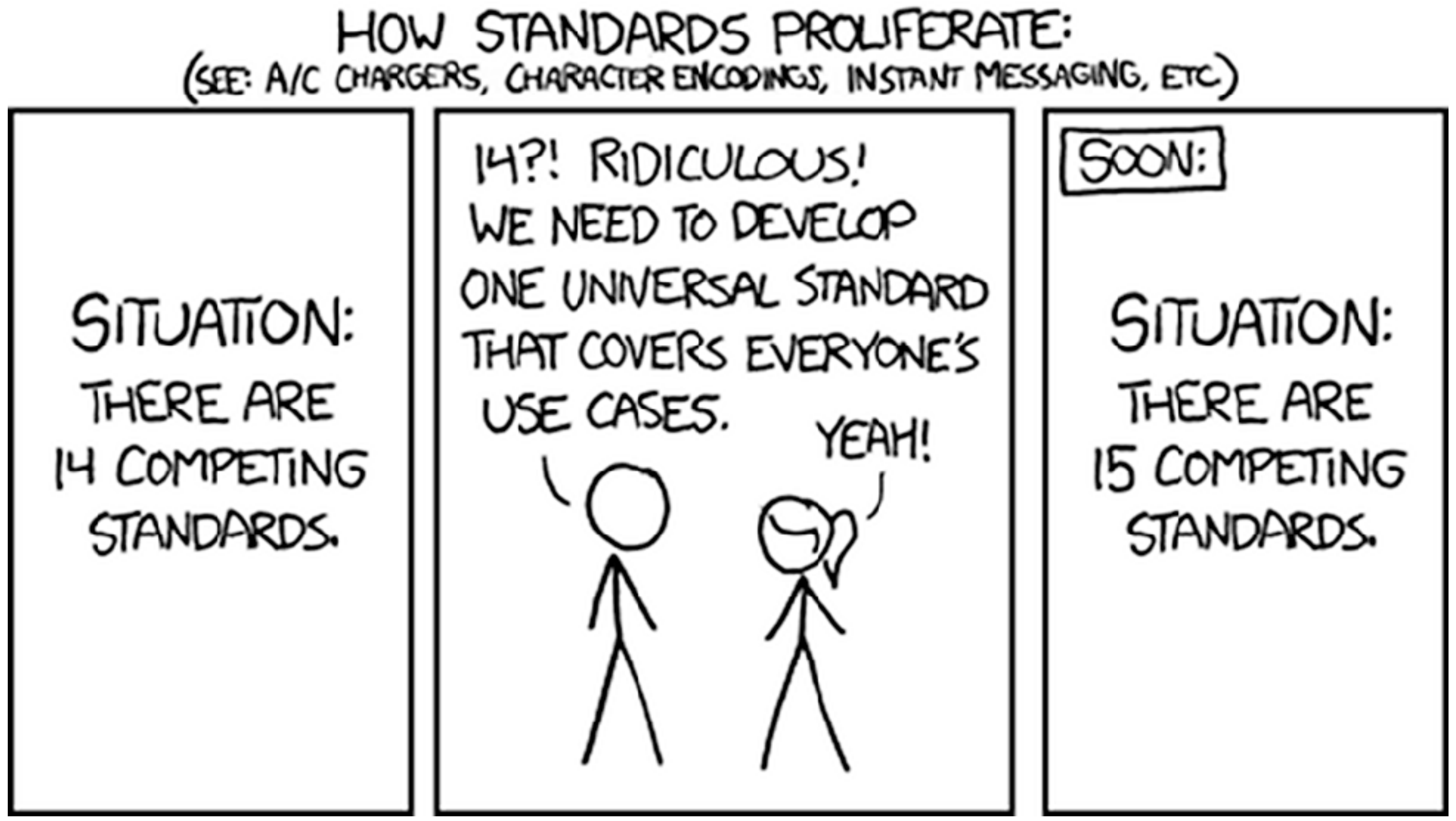
06

CONCLUSION

Automation makes our lives easier

Try to write your own standards

Contribute to cool open source projects such as Theme Sniffer plugin or WordPress Coding Standards


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

https://xkcd.com/927/

https://github.com/WPTRT/theme-sniffer

https://github.com/WordPress-Coding-Standards/WordPress-Coding-Standards/

HELPFUL RESOURCES

https://make.wordpress.org/core/handbook/best-practices/coding-standards/
https://github.com/WordPress-Coding-Standards/WordPress-Coding-Standards/
https://github.com/squizlabs/PHP_CodeSniffer/
https://packagist.org/packages/infinum/coding-standards-wp

INFINUM

Any questions?

DENIS.ZOLJOM@INFINUM.CO
@MADE_BY_DENIS

Visit infinum.co or find us on social networks:

infinum.co infinumco infinumco infinum