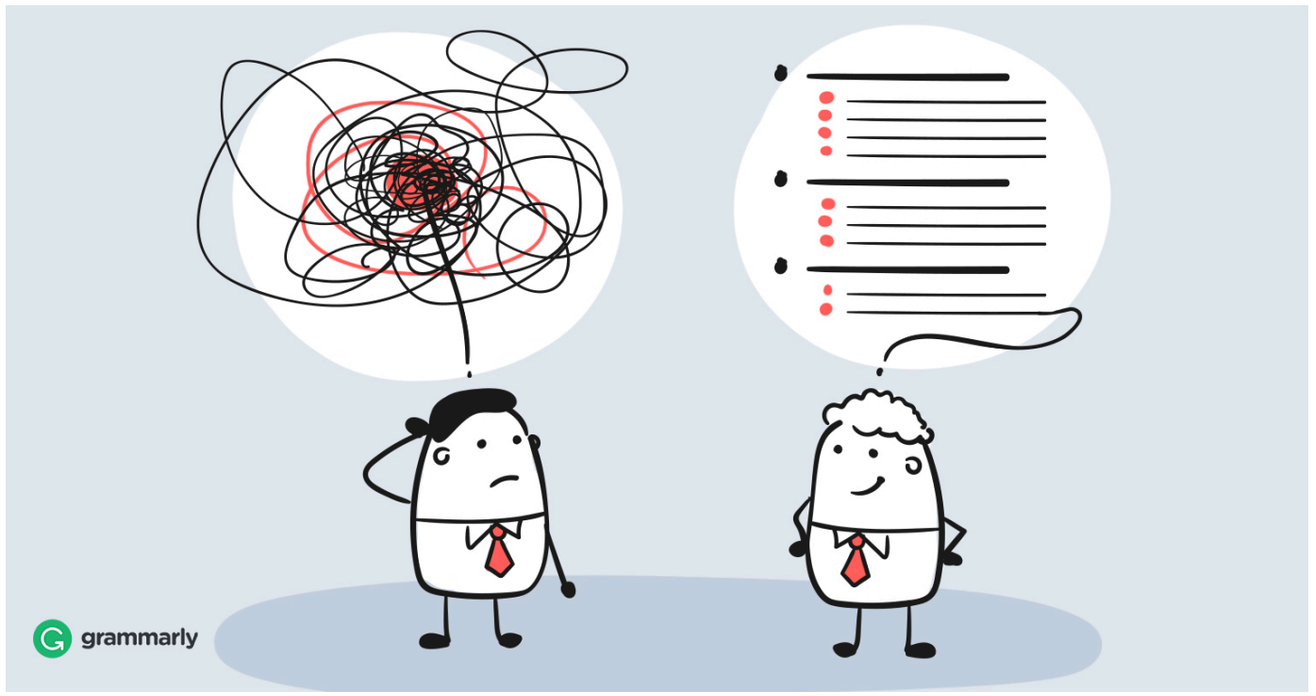
Lesson - 12

Methods to organize CSS
Breaking the Mock



SkillUp by Dobrea Vladislav

Organize CSS

It’s easy to write CSS code, but it is hard to scale and support it.

How can we organize?

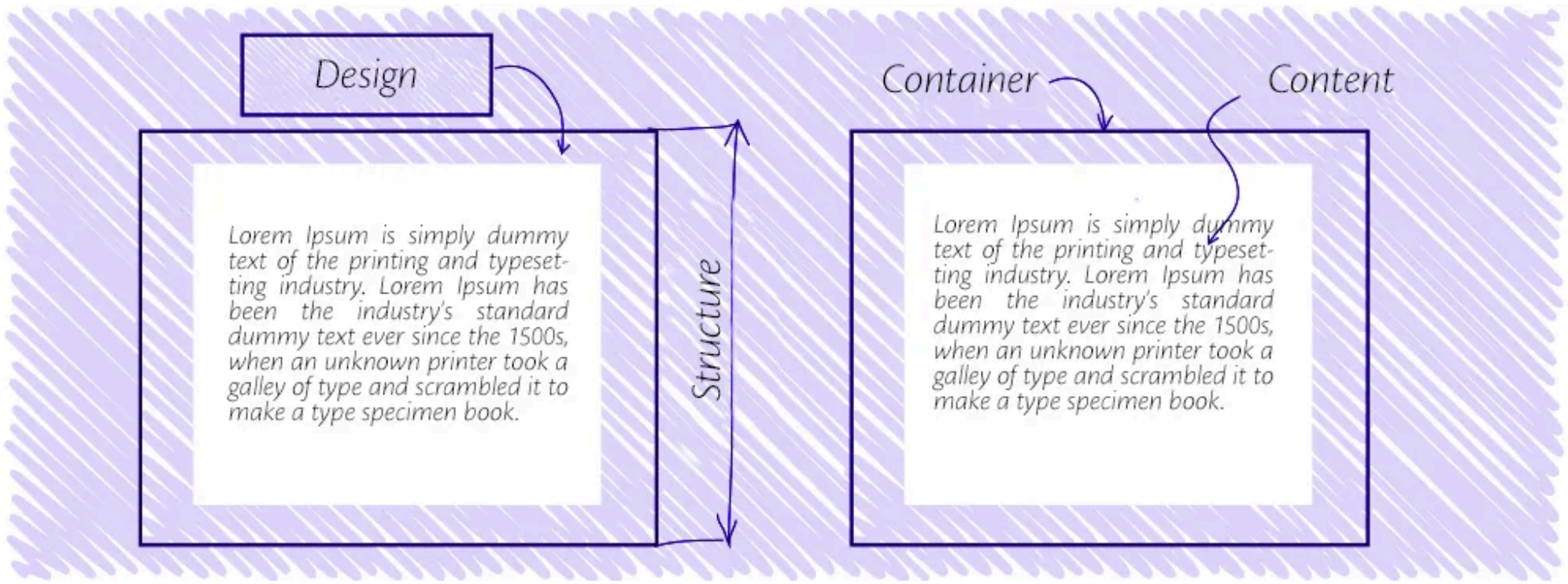
There are numbers of approaches that can help us organize and maintain our CSS code, no matter what approach you chose, the main rule is to **keep it consistent.**

- OOCSS
- SMACSS
- Atomic CSS
- MCSS
- AMCSS
- FUN

OOCSS

OOCSS stands for object-oriented CSS. This approach has two main ideas:

- Separation of structure and design
- Separation of container and content



OOCSS

Using this structure, the developer obtains general classes that can be used in different places.

At this step, there are two pieces of news (as usual, good and bad):

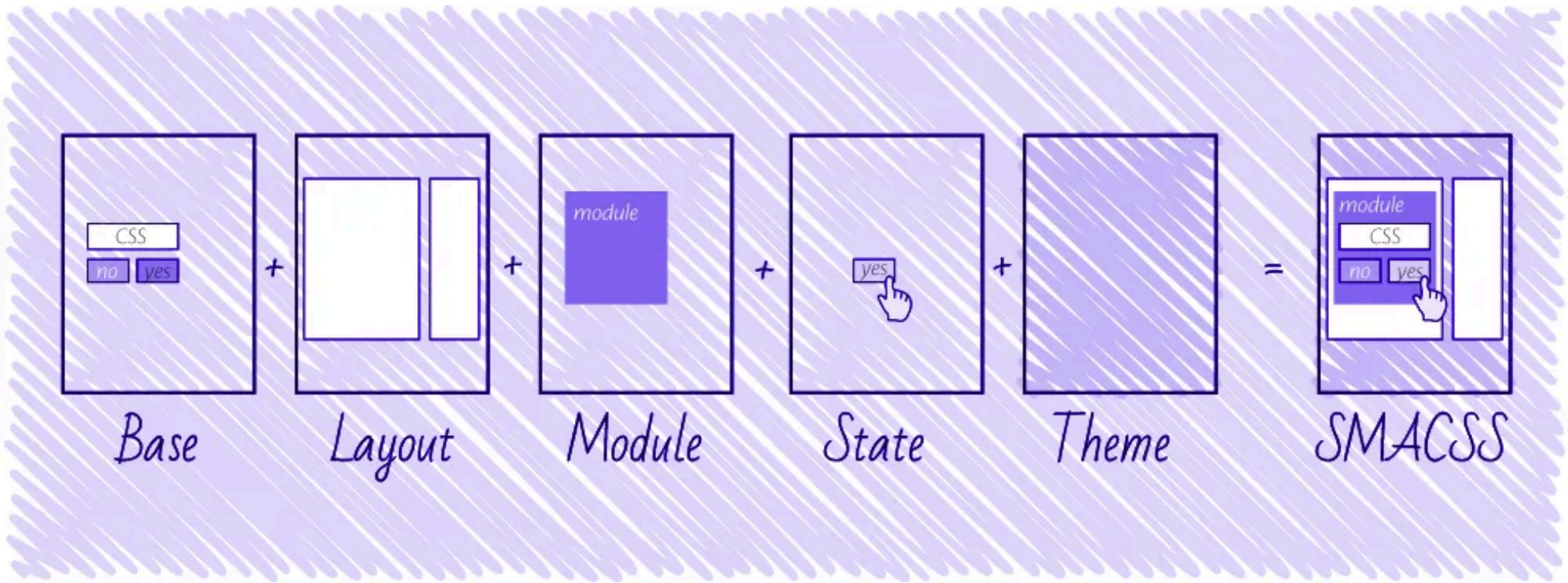
- Good: reducing the amount of code by reusing it (DRY principle).
- Bad: complex support. When you change the style of a particular element, you will most likely have to change not only CSS (because most classes are common), but also add classes to the markup.

Also, the OOCSS approach itself does not offer specific rules, but abstract recommendations, so how this method ends up in production varies.

As it happens, the ideas in OOCSS inspired others to create their own, more concrete, ways of code structuring.

SMACSS

SMACSS stands for Scalable and Modular Architecture for CSS. The main goal of the approach is to reduce the amount of code and simplify code support.



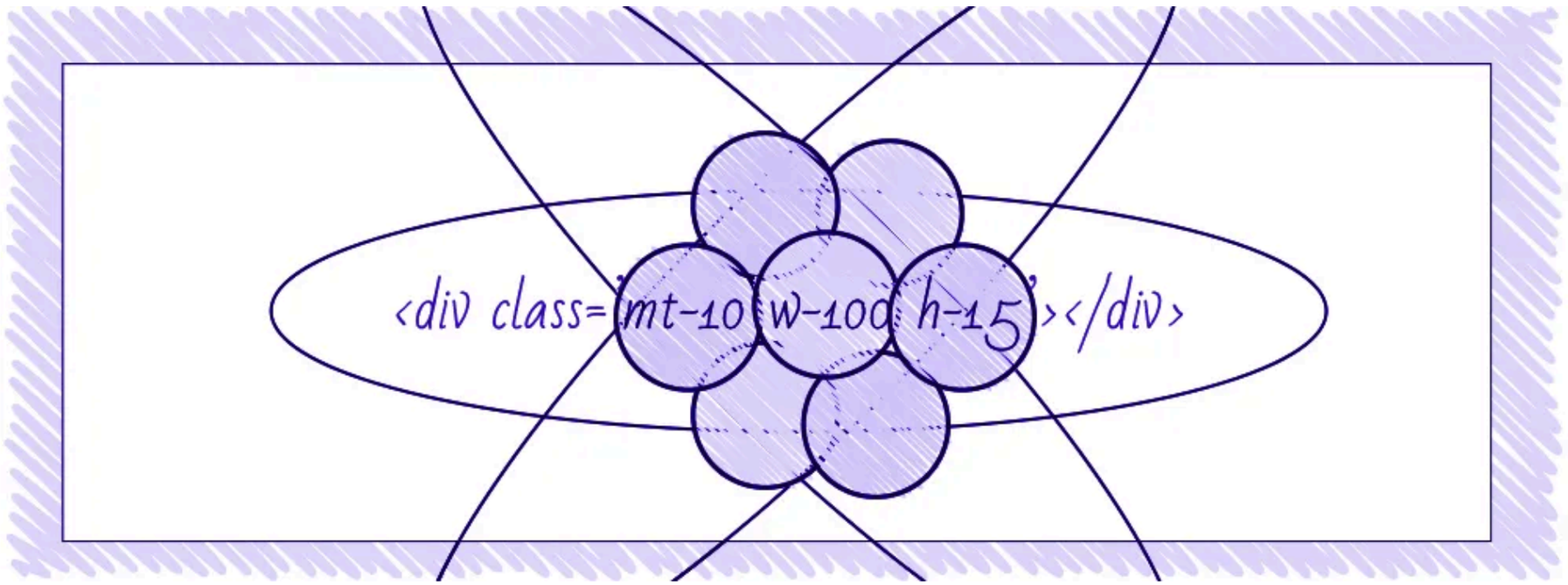
SMACSS

Jonathan Snook (author of book) divides styles into 5 parts:

- **Base rules.** These are styles of the main website elements – body, input, button, ul, ol, etc. In this section, we use mainly HTML tags and attribute selectors, in exceptional cases – classes (for example, if you have JavaScript-style selections);
- **Layout rules.** Here are the styles of global elements, the size of the cap, footer, sidebar, etc. Jonathan suggests using id here in selectors since these elements will not occur more than 1 time on the page. However, the author of the article considers this a bad practice (whenever an id appears in the styles, somewhere in the world the kitten is sad).
- **Modules rules.** Blocks that can be used multiple times on a single page. For module classes, it is not recommended to use id and tag selectors (for reuse and context independence, respectively).
- **State rules.** In this section, the various statuses of the modules and the basis of the site are prescribed. This is the only section in which the use of the keyword “! Important” is acceptable. (example with display: none!important;)
- **Theme rules.** Design styles that you might need to replace.

Atomic CSS

With Atomic CSS, a separate class is created for each reusable property. For example, `margin-top: 1px;` assumes creation of a class `mt-1` or `width: 200px;` the class `w-200`.



Atomic CSS

This style allows you to minimize the amount of CSS-code by reusing declarations, and it's also relatively easy to enter changes into modules, for example, when changing a technical task.

However, this approach has significant drawbacks:

- Class names are descriptive property names, not describing the semantic nature of the element, which can sometimes complicate development.
- Display settings are directly in the HTML.

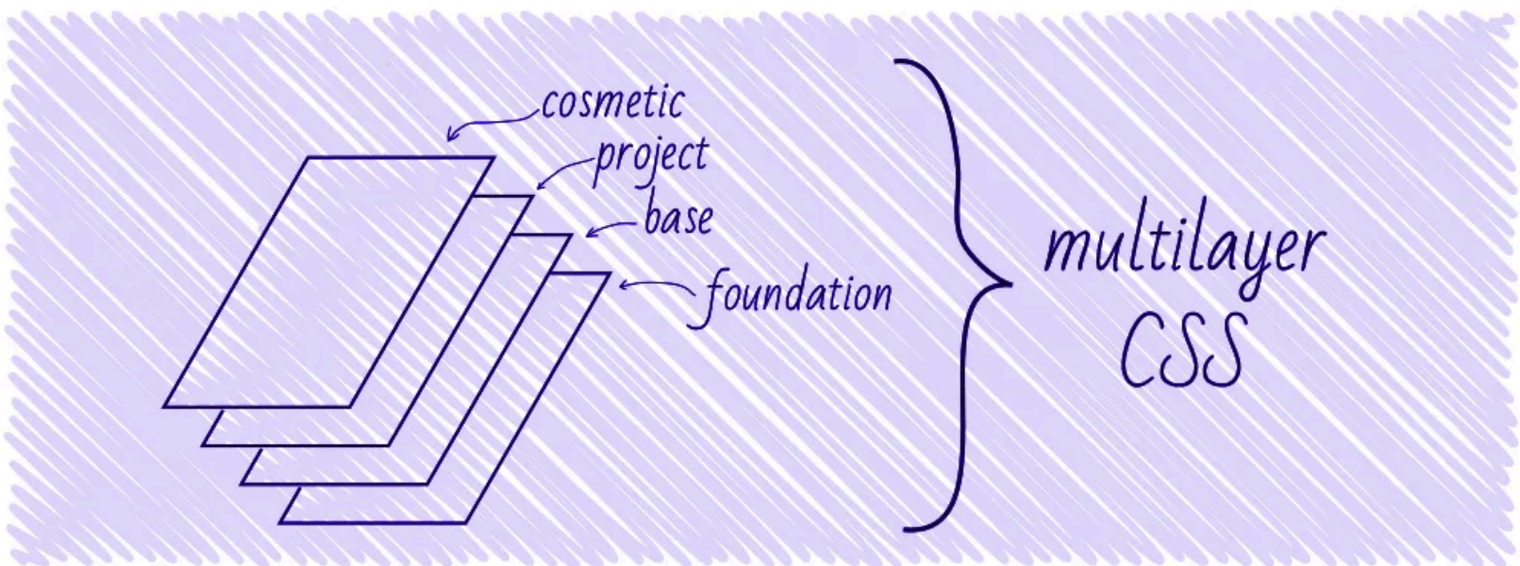
Because of these shortcomings, the approach has been met with a significant amount of criticism. Nevertheless, the approach can be effective for large projects.

Also, Atomic CSS is used in various frameworks to specify corrective element styles and in some layers of other methodologies.

MCSS

MCSS is Multilayer CSS. This style of code writing suggests splitting styles into several parts, called layers.

- Zero layer or foundation. The code responsible for resetting browser styles (e.g. reset.css or normalize.css);
- Base layer includes styles of reusable elements on the site: buttons, input fields for text, hints, etc.
- Project layer includes separate modules and a “context” – modifications of the elements depending on the client browser, the device on which the site/application is viewed, user roles, and so on.
- Cosmetic layer is written the OOCSS style, making minor changes in the appearance of elements. It is recommended to leave only styles that affect the appearance and are not capable of breaking the layout of the site (e.g colors and non-critical indents).



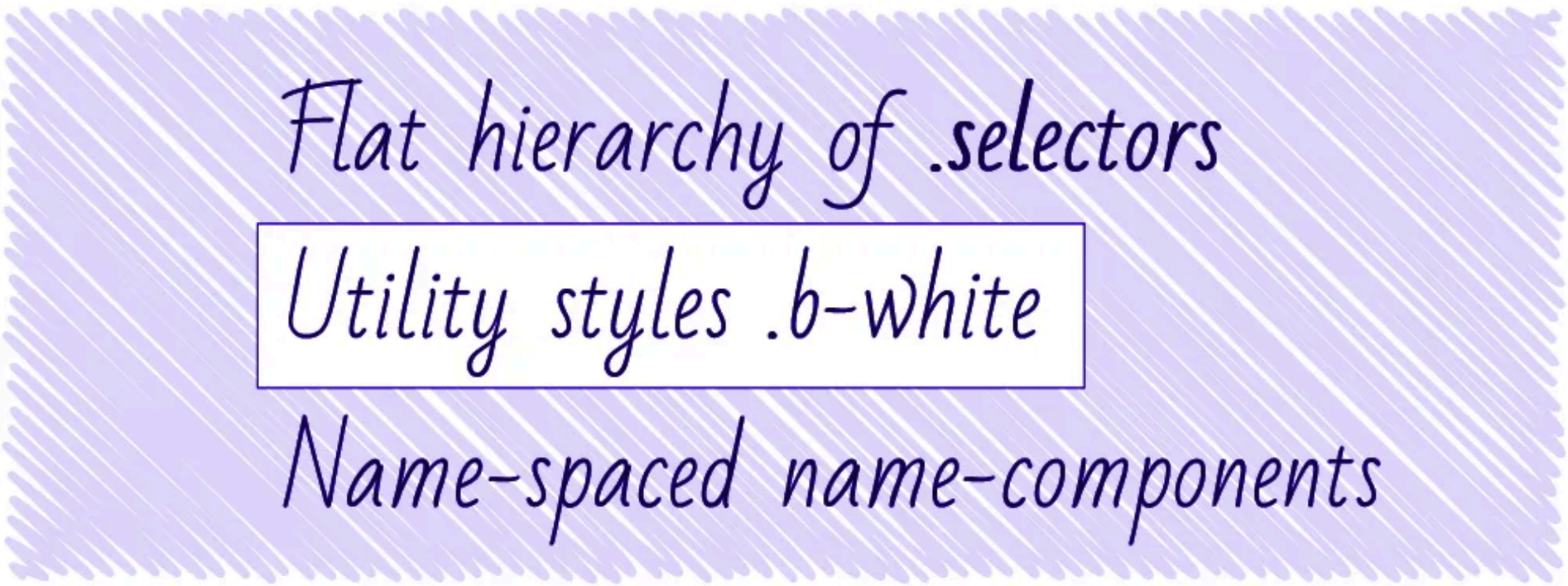
MCSS

The hierarchy of interaction between layers is very important:

- The base layer defines neutral styles and does not affect other layers.
- Elements of the base layer can only affect the classes of its layer.
- Elements of the project layer can affect the base and project layers.
- The cosmetic layer is designed in the form of descriptive OOCSS-classes (“atomic” classes) and does not affect other CSS-code, being selectively applied in the markup.

FUN

FUN stands for Flat hierarchy of selectors, Utility styles, Name-spaced components.



FUN

There is a certain principle behind each letter of the name:

- F, flat hierarchy of selectors: it is recommended to use the classes to select items, avoid a cascade without the need, and do not use ids.
- U, utility styles: it is encouraged to create the service atomic styles for solving typical makeup tasks, for example, `w100` for `width: 100%;` or `fr` for `float: right;`
- N, name-spaced components: Ben recommends to add namespaces for specifying the styles of specific modules elements. This approach will avoid overlapping in class names.

Some developers note that the code written using these principles is quite convenient to write and maintain; in some way, the author took the best from SMACSS and laid out this technique in a simple and concise manner.

This approach imposes quite a few requirements on the project and the code structure, it only establishes the preferred form of recording selectors and the way they are used in the markup. But in small projects, these rules can be quite enough to create high-quality code.

Which one to chose?

As you probably noticed, there is no absolute winner or ideal approach, each one of them has it’s pros and cons, and whatever to chose, is mostly a matter of taste and experience of each developer, the important is to know different approaches and pick whatever you like or maybe a few. Whatever you chose just use it consistent in whole project.

OK, and what is BEM?

Blocks, Elements and Modifiers

You will not be surprised to hear that BEM is an abbreviation of the key elements of the methodology — Block, Element and Modifier. BEM's strict naming rules can be found here.

Block

Standalone entity that is meaningful on its own.

Examples

`header` , `container` , `menu` , `checkbox` , `input`

Element

A part of a block that has no standalone meaning and is semantically tied to its block.

Examples

`menu item` , `list item` , `checkbox caption` , `header title`

Modifier

A flag on a block or element. Use them to change appearance or behavior.

Examples

`disabled` , `highlighted` , `checked` , `fixed` , `size big` , `color yellow`

BEM Examples?

Let's look how one particular element on a page can be implemented in BEM. We will take **BUTTON** from GitHub

HTML

```
<button class="button">
        Normal button
</button>
<button class="button button--state-success">
        Success button
</button>
<button class="button button--state-danger">
        Danger button
</button>
```

CSS

```
.button {
        display: inline-block;
        border-radius: 3px;
        padding: 7px 12px;
        border: 1px solid #D5D5D5;
        background-image: linear-gradient(#EEE, #DDD);
        font: 700 13px/18px Helvetica, arial;
}
.button--state-success {
        color: #FFF;
        background: #569E3D linear-gradient(#79D858, #569E3D) repeat-x;
        border-color: #4A993E;
}
.button--state-danger {
        color: #900;
}
```

BEM Benefits

Modularity

Block styles are never dependent on other elements on a page, so you will never experience problems from cascading.

You also get the ability to transfer blocks from your finished projects to new ones.

Reusability

Composing independent blocks in different ways, and reusing them intelligently, reduces the amount of CSS code that you will have to maintain.

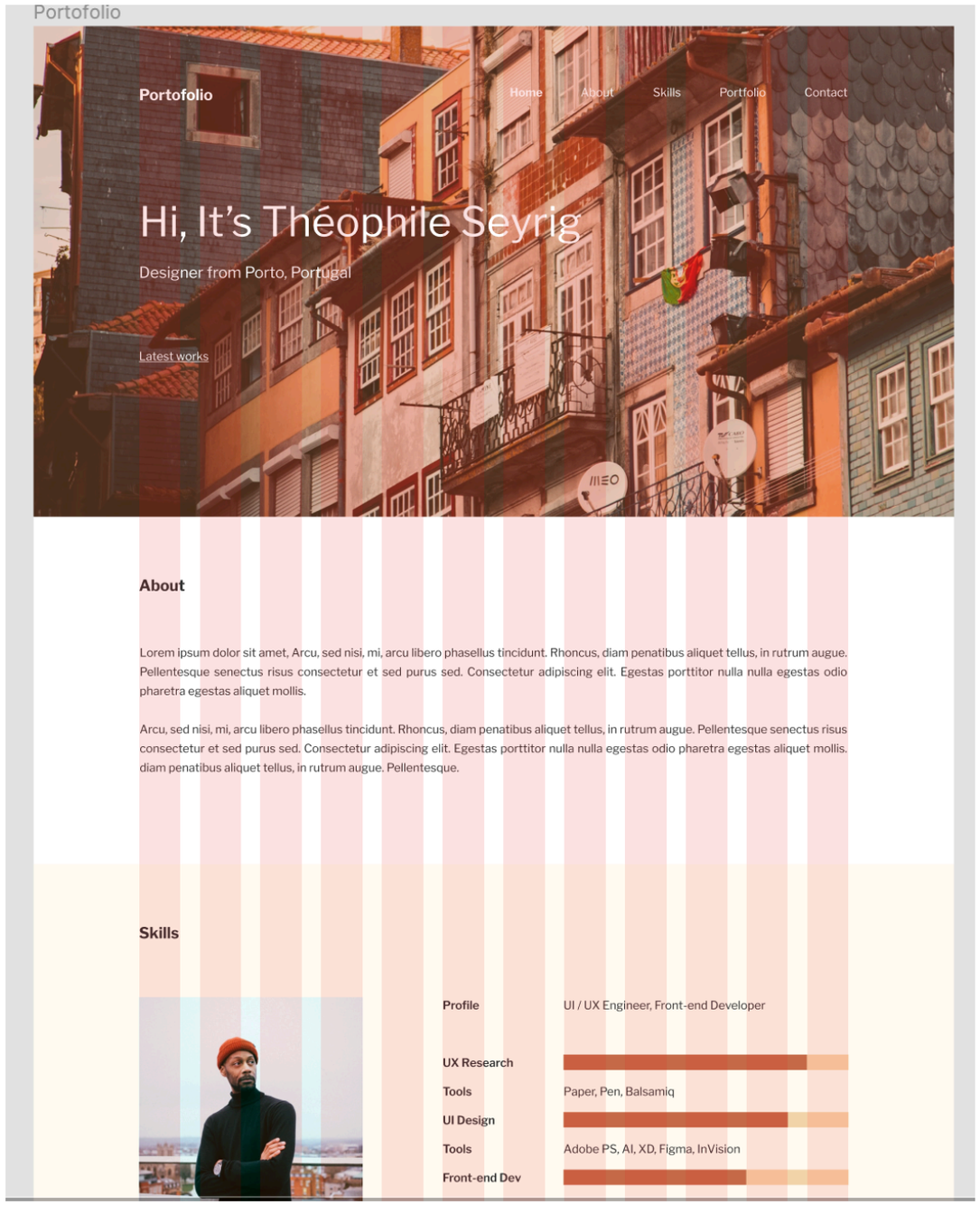
With a set of style guidelines in place, you can build a library of blocks, making your CSS super effective.

Structure

BEM methodology gives your CSS code a solid structure that remains simple and easy to understand.

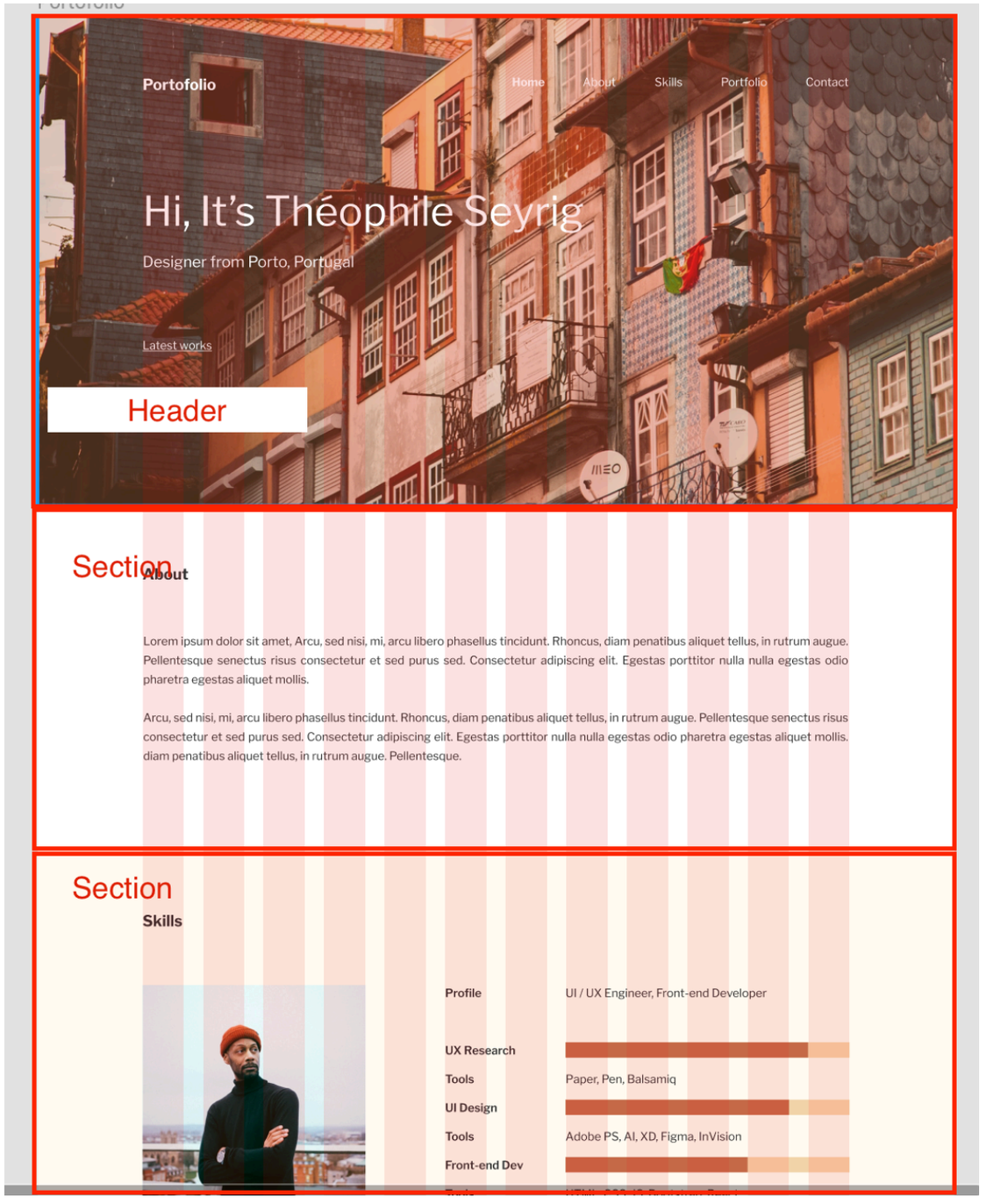
Breaking down Design

- Try to split UI into blocks (semantic blocks, header, content, section article)
- Try to find layout patterns or repeated elements
- Try to find common spaces usages (paddings and margins)
- Look at common and repeating UI elements (title, paragraphs, images)



Breaking into blocks

- Try to split UI into blocks (semantic blocks, header, content, section article)



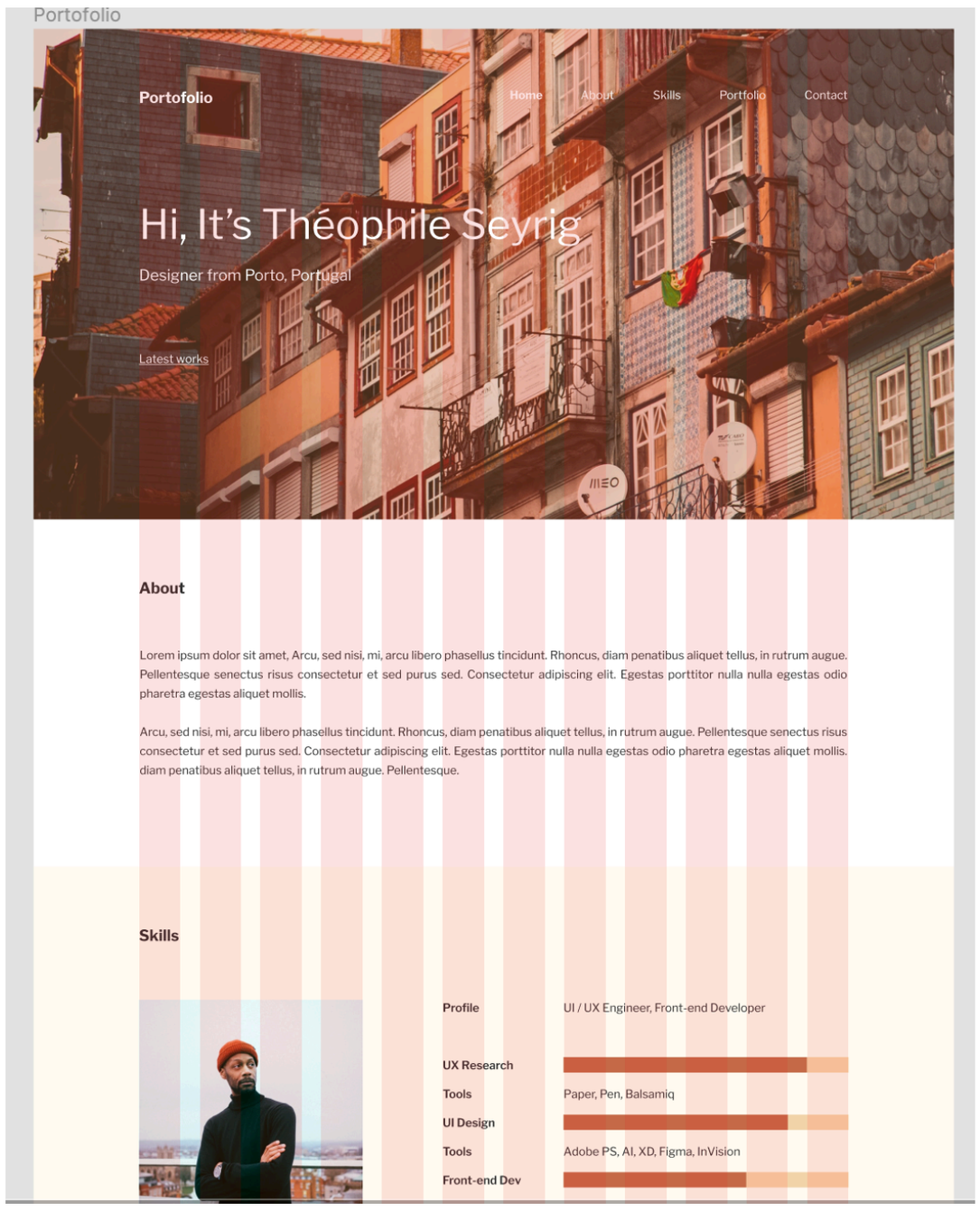
Common spacings and margins

- Finding common spacings and margins can help us separate them in some utility classes

Column design in complex UI’s

- Usually designer arrange their work using grid system, most general rule is 12 columns grid.
- In this example we can see how columns are defined (red stripes from top to bottom)
- We will implement our custom columns system (using SCSS)

```scss
.col-md-1 {
  width: 7.41667%;
  border: 1px solid #000; }
.col-md-2 {
  width: 15.83333%;
  border: 1px solid #000; }
.col-md-3 {
  width: 24.25%;
  border: 1px solid #000; }
.col-md-4 {
  width: 32.66667%;
  border: 1px solid #000; }
.col-md-5 {
  width: 41.08333%;
  border: 1px solid #000; }
.col-md-6 {
  width: 49.5%;
  border: 1px solid #000; }
```



Learning Resources

1. **Organize CSS**
    1. http://vanseodesign.com/css/smacss-introduction/
    2. **SMACSS Book** http://smacss.com/book/
    3. https://css-tricks.com/methods-organize-css/
    4. **Atomic CSS** https://css-tricks.com/lets-define-exactly-atomic-css/
2. **BEM**
    1. http://getbem.com/introduction/
    2. https://blog.decaf.de/2015/06/24/why-bem-in-a-nutshell/
    3. **[RU]** https://ru.bem.info/
3. **CSS Related**
    1. https://designshack.net/articles/css/5-steps-to-drastically-improve-your-css-knowledge-in-24-hours/
    2. https://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048

Home Work

1. Create a new project in GitHub lesson-12-hw
2. Clone project and **npm init**
3. Install sass
4. Create a build and watch command
5. Implement HTML/CSS for design provided in figma: https://www.figma.com/file/wQ0XVf5AbE2tMPFFFvZMF3/Porto-(Copy)?node-id=0%3A1
6. Upload to GitHub
7. Create site in netlify