**Univerzita Hradec Králové**
**Fakulta informatiky a managementu**
**Katedra informačních technologií**

# Atomický web design

Diplomová práce

Autor: Bc. Natalja Ljubišová
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Hradec Králové duben 2021

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracovala samostatně a s použitím uvedené literatury.

*vlastnoruční podpis*

V Hradci Králové dne 30.4.2021 Natalja Ljubišová

Poděkování:

Děkuji vedoucí diplomové práce Mgr. Daniele Ponce, Ph.D. za metodické vedení práce, ochotu a cenné rady v průběhu práce.

## Anotace

Diplomová práce se zabývá využitím metodiky Atomického web designu pro práci s kaskádovými styly. Práce zpočátku zkoumá proces návrhu webu a rozdíl mezi rolí webového designera a webového kodéra. Následně zkoumá možnosti, které vedou ke zlepšení srozumitelnosti a čitelnosti CSS kódu v průběhu jeho vytváření. Zabývá se vlastnostmi preprocesorů CSS a doporučenými zásadami, které vedou kodéra k tomu, aby byl kód CSS srozumitelný i ostatním kodérům v týmu. Dále byly představeny metodiky CSS, mezi které patří i Atomický web design. Představené metodiky byly následně na základě získaných informací prakticky aplikovány na vzorovém stylopisu, který původně nevyužívá žádnou ze zkoumaných metodik. Výsledkem práce je praktické představení jednotlivých metodik, vyhodnocení jejich použití a navrhnutá doporučení.



## Annotation

### Title: Atomic web design

The diploma thesis deals with the use of the Atomic Web Design methodology for working with cascading styles. Initially the thesis examines the web design process and the difference between the role of a web designer and coder, it then explores the possibilities that lead to improved comprehensibility and readability of CSS code during its creation. The thesis deals with the characteristics of CSS preprocessors as well as the recommended rules/principals guiding the coder to create a CSS code understandable to other coders. Then CSS methodologies are introduced which include Atomic web design. The presented methodologies were subsequently applied practically to a CSS style sheet, which originally did not use any of the examined methodologies. The result of the thesis is a practical presentation of individual methodologies, evaluation of their use, and proposed recommendations.

# Obsah

# Seznam obrázků



# Seznam tabulek



# Seznam kódu

# Úvod

Od vzniku služby World Wide Web, kdy internet primárně sloužil jako uložiště stránek pro statistické a vědecké informace, patří dnes tato informační služba k nejrychleji se rozvíjejícím technologiím, která obsahuje mnoho dynamických webových aplikací. Webové aplikace dnes patří k základním prostředkům pro prezentaci služeb a produktů. Čím dál tím více jsou vytvářeny mnohem rozsáhlé weby, za jejichž návrhem a vývojem dnes nestojí pouze jeden člověk, ale mnohočetný tým lidí. Jelikož je proces tvorby webu komplexní, je nutné, aby se každý člen týmu specializoval a zaměřil na určitou část procesu vytváření webové aplikace.

Součástí týmu, který je zodpovědný za proces návrhu webu, jsou weboví designeři a weboví kodéři, kteří jsou často nesprávně řazeni do stejné skupiny, i když jejich role nejsou totožné. Od každé z těchto rolí se vyžadují odlišné dovednosti.

Doménou webového kodéra, při procesu návrhu webu, jsou jazyky HTML a kaskádové styly. Na velkých projektech, na kterých pracuje celý tým kodérů, se mohou v průběhu procesu vývoje objevit problémy s čitelnosti kódu CSS a s jeho porozuměním. Pro zlepšení srozumitelnosti a čitelnosti kaskádových stylů v průběhu jejich vývoje nebo při výměně vývojáře je důležité hledat možnosti, které pomohou k tomu, aby byl kód CSS lépe pochopitelný a čitelný. Vhodná metodika zajistí jednotný vzhled a správnou funkčnost aplikace.

Diplomová práce bude zkoumat možnosti, které vedou ke zlepšení srozumitelnosti a čitelnosti CSS kódu v průběhu jeho vytváření. Nejprve bude představen proces návrhu webu a rolí webového designera a webového kodéra, které se podílí na procesu návrhu. Dále se diplomová práce bude věnovat preprocesorům CSS, doporučeným zásadám při psaní CSS a metodikám CSS.

V praktické části bude představeno použití jednotlivých metodik na vzorovém stylopisu. Hlavním cílem bude zhodnocení přínosu jednotlivých metodik CSS pro zlepšení struktury a čitelnosti kódu CSS. Následně budou navržena doporučení při jejich použití.

# 1 Navrhování pro web

Dříve byly webové stránky jednoduché HTML stránky a webmaster byl jejich správce. Webmaster měl tendenci k vytváření spletitého HTML a přemýšlel, zda nově příchozí CSS je něco, čím by se měl zabývat. Většina z nich již odepsala Javascript jako pomíjivý výstřelek. Ale jako každé médium i web rostl. Javascript se zasekl a CSS bylo užitečné pouze pro nastavení fontů písma a barev stránky. Provoz na webu rostl, technologie webu dále expandovaly a webmasteři se ocitli na křižovatce. Objevilo se spoustu nových věcí, které nestíhali sledovat, a příliš mnoho práce. To donutilo webmastery specializovat se a na každé další křižovatce soustředit se na menší a menší část webového procesu. Někteří se zaměřili na poskytování souborů skrze server, jiní zdokonalovali své dovednosti v přístupu k databázi a další našli radost ve vytváření grafiky a obrázků. [1]

Dnes již za návrhem a vytvořením rozsáhlých webů stojí celý tým lidí, jejichž počet se pohybuje od hrstky až po stovky [1]. Každý člen týmu se zaměřuje na část procesu vytváření webu, má svoji roli a odpovědnost za tuto část. Součástí vytváření webu je webdesign. V průběhu let se webdesign stal pojmem, který zahrnuje řadu různých disciplín, mezi které patří [2]:

- vizuální grafický design,
- design uživatelského rozhraní (UI design),
- design uživatelské zkušenosti nebo prožitku (UX design),
- HTML a kaskádové styly,
- skriptování a programování,
- atd.

Nesčetné množství rolí a odpovědností, které jsou obvykle zastřešeny pod pojmem webdesign, jako jsou například webový kodér a webový designer, mnoho lidí kategorizuje do jedné skupiny. Avšak to nejsou stejné pracovní pozice a od každého se vyžadují jiné dovednosti. [3]

Cílem této kapitoly je popsat rozdíly mezi těmito rolemi a zároveň nastínit průběh navrhování webu.

## *1.1 Webový designer*

Podle Staníčka [4] je práce designera poměrné obsáhlá mezioborová práce zahrnující psychologii, sociologii, marketing, informační technologie, design, komunikaci i řízení projektu. Není však nutné, aby v každé z těchto oblastí byl jedinec odborníkem. Ve většině případů se webový designeři specializují na jednu z těchto oblastí. Dokonce je u rozsáhlých projektů žádoucí pro zajištění úspěchu a kvality výsledného webu, aby jednotlivé činnosti zajistil příslušný specialista [4].

Prvním krokem při budování webu je navrhnout, jak bude web fungovat. Podle Robbins [2] je důležité si stanovit cíle webu, jak bude web použit a jak se na něm budou návštěvníci pohybovat. Tyto úkoly spadají hned do několika oblastí jako je UX design, interakční design (IxD) a UI design. Avšak tyto tři oblasti UX, UI a IxD design se velmi často překrývají a není neobvyklé, že některou z jejich kombinací zvládne jeden tým nebo osoba [2].

### 1.1.1 UX designer, Interakční designer a UI designer

Návrhy webů často začínají průzkumem cílových uživatelů. Úkolem UX designera je zkoumat a analyzovat na základě pozorování a rozhovorů s uživateli, jak uživatel vnímá a používá nabízený produkt [2]. Dále tyto získané znalosti a zkušenosti aplikuje na vývoj produktu a zajistí, aby měl uživatel co možná nejlepší dojem a zkušenost s použitím tohoto produktu. Podle Staníčka [4] by se měl nejenom soustředit na cílovou skupinu, tedy budoucí uživatele, ale poznat i svého klienta a jeho cíle produktu, a stát se tak mostem mezi oběma břehy.

UX designeři v průběhu navrhování a vytváření webu provádí výzkum, analyzují své poznatky, následně o nich informují členy vývojového týmu a sledují vývojový projekt, aby zajistili, že jsou tyto poznatky správně implementovány. Podle Řezáče [5] je role UX designera být středem každého projektu a být ten, za kterým členové týmu přijdou, pokud je třeba ujasnit některou část webu. Čím je projekt větší, tím více je UX designer potřebný [6].

Interakční designer by měl zajistit, aby byl web co nejjednodušší, nejefektivnější a bylo ho snadné použít [2]. Jeho cílem je, aby web jako interaktivní médium poskytl vzájemnou, dvoustrannou komunikaci s uživateli, kteří svojí zpětnou vazbou, odezvou a rozhodováním ovlivňují a mění chování webu [4].

Interakční designer určuje, jak tato vzájemná komunikace bude vedena, určuje pravidla chování, navrhují, jak se zobrazí varovné hlášení, design vstupních polí nebo tlačítka, tak aby akce uživatele byla co nejsrozumitelnější a nejintuitivnější [4]. Interakční designer navrhuje co nejpřímočařejší cestu k cíli.

S rolí interakčního designera úzce souvisí i role UI designera. Avšak UI designer se více zaměřuje na funkční organizaci stránky a na konkrétní nástroje, které uživateli slouží k orientaci a pohybu v obsahu nebo k provádění určitých úkonů. Příkladem jsou různá tlačítka, menu, odkazy a další [2]. Zaměřuje se na jejich vzhled a použití. Ve vztahu k UX, UI rozšiřuje UX a UI je prostředkem pro UX [6].

Procesy a výstupy IxD, UI a UX designera pro návrh webu jsou představeny v následujících odstavcích.

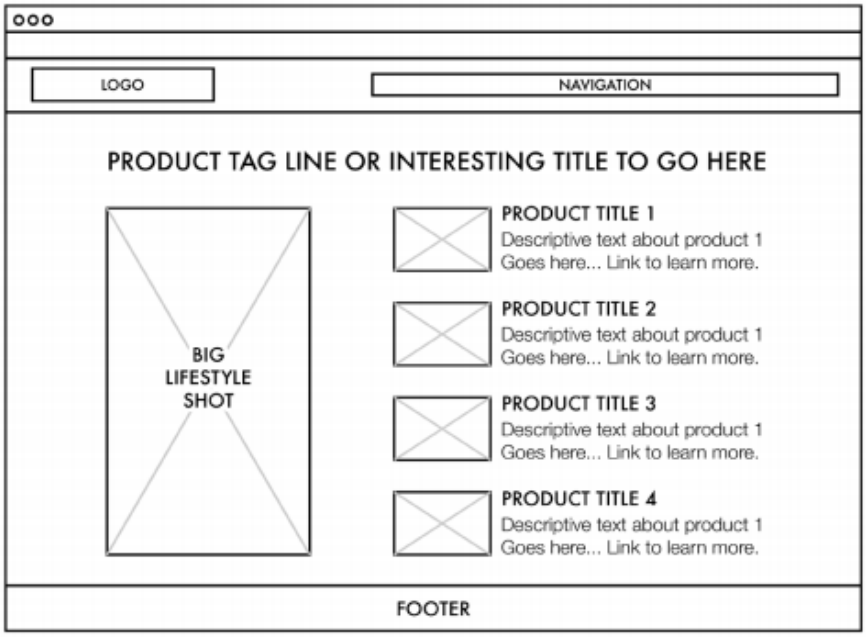
**Průzkum uživatelů a průběžné testování**

Úspěchem dobrého návrhu webu je zjistit a pochopit potřeby uživatelů a jaké bude jejich chování na webu. Jedná se o přístup zvaný User Centered Design (UCD), kdy se web navrhuje na základě potřeb uživatele. V současnosti se jedná o klíčový bod při návrhu webu. S použitím tohoto přístupu, návrhy webů začínají průzkumem uživatelů včetně rozhovorů a pozorování. Díky tomu, je možné lépe porozumět, jak bude web použit. Pro designéry je typické, že v každé fázi procesu návrhu provedou kolo uživatelských testů, aby otestovali použitelnost svých návrhů. V případě, že uživatelé obtížně zjišťují, kde najít obsah nebo jak přejít k dalšímu kroku procesu, pak je potřeba návrh upravit. [2]

**Drátěný model**

Drátěný model (*wireframe*) je abstraktní návrh, který ilustruje základní formu a funkce, které se nachází na jednotlivých stránkách webu [7]. Bez jakékoliv dekorace a grafického designu jsou v modelu černobílými obrysy a tvary naznačeny, kde se bude nacházet navigační menu, vyhledávací pole nebo prvky formuláře [2].

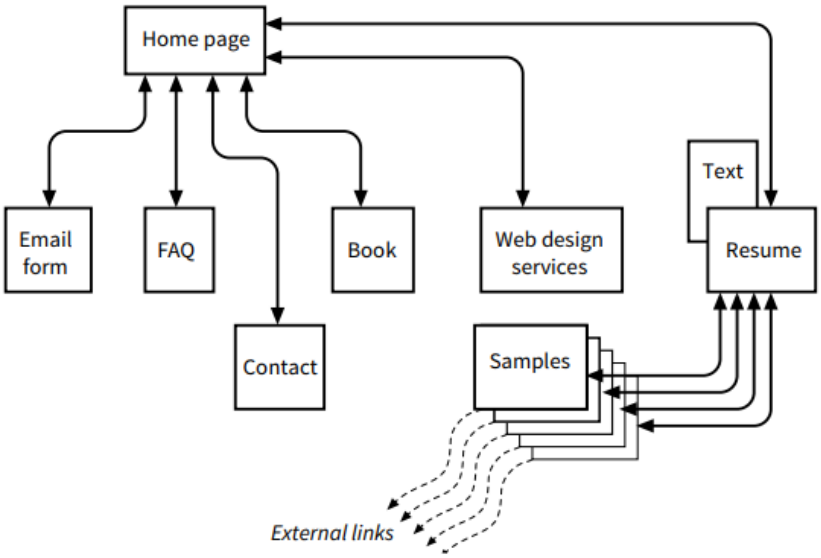
Kolekce drátěných modelů by měly poskytnout komplexní „kosterní“ pohled na celý produkt. K modelům jsou obvykle pro vývojový tým připojeny poznámky a pokyny, jak by určité části měly fungovat [2].




**Obrázek 1: Příklad drátěného modelu [7]**

### Schéma stránek

Schéma stránek (*site diagram*) označuje strukturu webu jako celek, na kterém je zobrazeno, jak spolu jednotlivé stránky souvisí [2].




**Obrázek 2: Příklad schématu stránek [2]**

### Storyboard a uživatelské vývojové schéma

Storyboard trasuje cestu přes web z pohledu typického uživatele. Obsahuje scénář a k tomu scény, které se skládají z obrazovek nebo interakcí uživatele

s obrazovkou. Cílem storyboard je demonstrovat kroky k provedení úkolů a nastínit možné varianty. Zároveň představuje některé standardní typy stránek. [2]

Další metoda pro zobrazení propojení jednotlivých částí webu je použití uživatelského vývojového schématu. Oproti storyboardu se toto schéma zaměřuje spíše na technické detaily. Např. pokud uživatel vykoná nějakou činnost na webu, tak pomocí uživatelského vývojového schématu bude znázorněno, jaké spustí funkce na serveru. Schéma se vytváří většinou pro zobrazení kroků při registraci uživatelů nebo při online platbě. [2]

### 1.1.2 Webový grafik

Pod pojem webdesign spadá i role webového grafika. Webový grafik má za úkol vytvořit náhledy webových stránek. Podle Staníčka [4] by se grafický vzhled měl navrhnout až na závěr jako vyústění všech zadaných specifikací návrhu webu. Návrh grafického vzhledu je sice součástí návrhu webu, ale neměl by být ten nejpodstatnější. Kvalitní návrh webu spočívá hlavně v analýze a vymezení cílových uživatelů, navigační logice a jeho funkčnosti [4]. Dobrý grafik nemůže navrhnout funkční grafický návrh, pokud nezná základní vstupy, cíl webu a cílovou skupinu uživatelů. Jako příklad uvádí Staníček [4] původní podobu Facebooku nebo eBaye, kdy jejich grafický vzhled nebyl uchvacující, ale i přesto byly úspěšné, a to především díky tomu, že dobře fungovaly. Jejich vizuální vzhled byl vylepšen až později.

Jelikož má webový grafik na starosti tvorbu loga, grafiky, výběr barev, rozvržení a další části webu, které utváří jeho celkový vzhled, tak by měl mít grafický cit. Aby byl zajištěn dobrý první dojem webu, měly by části, které utváří vzhled webu, být v souladu se značkou a poselstvím, kterou organizace představuje [2]. Webový grafik by měl skvěle graficky interpretovat drátěné modely tak, aby zapadly do vizuálního stylu zákazníka [4]. Pro grafické ztvárnění návrhu by měl ovládat některý z grafických programů např. Photoshop. Zároveň je důležité, aby byl seznámen s různými UI frameworky a jinými knihovnami na straně prohlížeče, které určují výsledný vzhled webových stránek [4]. Při návrhu by neměli zapomínat i na to, jaká budou cílová média, a na kterých zařízeních bude web používán.

Často jsou weboví grafici zároveň i kodéři. Pokud to tak není, je důležité, aby kodér s grafikem mezi sebou dobře komunikovali. Může se stát, že kodér neumí s

Photoshopem a může vyžadovat například optimalizaci PNG a JPG obrázků pro jejich vložení do stránky [4]. V takovém případě je tu webový grafik, aby mu obrázky optimalizoval.

## 1.2 Kodér / frontend vývojář

Velká část procesu vytváření webových stránek zahrnuje vytváření a řešení problémů s dokumenty, kaskádovými styly, skripty a obrázky, které tvoří web. Ve firmách zabývajícími se webovým designem se obvykle tým, který se stará o vytváření webů nebo šablon pro webové stránky, označuje jako vývoj nebo vývojové oddělení. Robbins [2] tento vývojový tým rozděluje do dvou širokých kategorií: frontend a backend vývoj. Jako frontend se označuje jakýkoliv aspekt procesu tvorby designu, který se objeví v prohlížeči nebo se prohlížeče přímo týká. Tato kategorie zahrnuje znalost HTML, kaskádových stylů a Javascriptu. Podle těchto znalostí Řezáč [5] definuje dvě role: kodéra a frontend vývojáře. Roli kodéra popisuje jako roli, která by měla být schopná náhled webové stránky, který dostane od webového grafika, přepsat do HTML a CSS. Kodér má tedy blíž k CSS. Oproti tomu frontend vývojář má blíž ke skriptovacímu jazyku Javascript.

Dále by kodér měl být schopný napsat výsledný kód tak, aby se dobře zobrazoval v jednotlivých prohlížečích a zařízeních, a aby byl udržitelný, konzistentní a snadno rozšiřitelný. Podle Řezáče [5] by měl mít také typografický cit, aby byl schopný ladit s grafikem každý pixel.

Někteří mají mylnou představu, že kódování je programování. HTML a CSS však nejsou programovací jazyky. Většina kodérů a frontend vývojářů tak často sdílí kancelář s backend vývojáři a webový designeři sedí na jiném podlaží než vývojáři. Organizačně tak vzniká velká propast mezi designéry a kodéry/frontend vývojáři. [8]

Kodéři/frontend vývojáři pomocí HTML, CSS a Javascript vytváří uživatelská rozhraní, která designeři pečlivě vytváří ve svých nástrojích. Aby společně vytvářeli úspěšné návrhové systémy uživatelského rozhraní, je důležité brát kodéry a frontend vývojáře jako základní součást procesu návrhu. Spolupráce mezi webovými designery a webovými kodéry/frontend vývojáři je často velmi chaotická, a proto je důležitá neustálá komunikace, těsné vazby a skutečná

spolupráce, aby se zamezilo nerealistickým návrhům, a aby návrh vypadal stejně jako finální výtvor. Klíčové dále je, že rychlejší skok do prohlížeče zlepší vytváření vzorů, které vytvoří funkční designový systém. [8]

### Hypertext Markup Language (HTML)

Hypertext Markup Language (HTML) je značkovací jazyk, který slouží k vytváření webových stránek, kde obsah a funkce jsou označeny pomocí HTML tagů. Nejedná se tedy o programovací, ale o značkovací jazyk. Pomocí HTML tagů jsou označeny a charakterizovány různé komponenty HTML dokumentu, kterými jsou například nadpisy, odstavce nebo seznamy. [2]

Nyní je aktuální HTML verze 5, která přinesla podporu nejen pro webové prohlížeče, ale i smartphony, smart televize, tablety a další zařízení[9]. Dále přinesla nové prvky pro formulář a lepší rozvržení stránky díky HTML tagům: *main*, *header*, *footer*, *article*, *section*, *aside* a *navigation*. Také definuje API, která zlepšují programový přístup k dokumentu a prohlížeči, aby se zamezilo používání Adobe Flash z důvodu bezpečnosti a spolehlivosti webových aplikací. A další. [9], [10]

### Javascript

Javascript byl vytvořen Brendanem Eichem v Netscape v roce 1995 a původní název byl Livescript, ale jelikož byla v té době Java velmi populární, tak se pro lepší marketing název změnil na Javascript. Javascript však s Javou nic společného nemá. [2]

Javascript je jednoduchý, ale velmi výkonný skriptovací jazyk, který dodává webovým stránkám interaktivitu a chování. Jedná se o jazyk známý jako dynamický a volně psaný. Běží na straně klienta nikoliv na serveru, jako je tomu například u PHP nebo Ruby. To znamená, že Javascript a způsob jakým je použit závisí na možnostech a nastavení prohlížeče. Nemusí být k dispozici, buď z důvodu, že se uživatel rozhodl ho vypnout, nebo protože jej zařízení nepodporuje, na to však dobří frontend vývojáři myslí. Pomocí Javascriptu se například zajistí, že si prohlížeč zapamatuje informace o uživatelích nebo proběhne kontrola platnosti záznamů ve formuláři. [2]

**Kaskádové styly (CSS)**

Kaskádové styly neboli zkráceně CSS z anglického Cascading Style Sheets jsou standardem World Wide Web Consortium (W3C) od roku 1996 pro definování prezentace dokumentů psaných v HTML a XML [2].

Konsorcium W3C se stará o standardizaci webových technologií jako je CSS. Každá z jejich specifikací prochází řadou fází vývoje, než se stane specifikace jejich doporučením. Prvním doporučením W3C na konci roku 1996 se stala verze CSS 1, která obsahuje velmi základní vlastnosti jako jsou fonty, *margin* (vnější okraj), barvy textu, pozadí a jiných elementů. V roce 1998 se stala doporučením verze CSS 2. Tato verze rozšířila pokročilé možnosti stylování například o *float* a *position*, a také o nové selektory: selektor potomka, selektor sourozence a univerzální selektor. A další jiné pokročilé možnosti. Verze CSS 3 je velká samostatná specifikace definující různé funkce. CSS 3 je rozdělen do několika samostatných dokumentů zvané moduly. Každý z těchto modulů přidává nové funkce nebo rozšiřuje funkce definované v CSS 2 a zachovává zpětnou kompatibilitu. [11]

Často je CSS považováno za návrhový nástroj, který umožní návrháři určit vizuální vzhled webové stránky. Pomocí CSS jsou řešeny vlastnosti fontů, barvy, rozložení stránky, obrázky na pozadí atd. Umožňuje přidání speciálních efektů a elementárních animací na web [2]. Díky své kontrole nad konečným vzhledem webové stránky má CSS přímý dopad jak na použitelnost, tak na přístupnost. Kvůli těmto faktorům se vytváření stylů a psaní CSS obvykle považuje za konstrukční úkol, a je potřeba tyto styly udržovat. CSS má specifickou syntaxi, kterou je třeba dodržovat, a pravidla, která způsobí provedení akcí. [11]

# 2 Preprocesory

CSS preprocesor je jazyk, který je postavený nad CSS [2]. Rozšiřuje CSS o úsporné charakteristiky z programovacích jazyků. Poskytuje přístup k funkcím jako jsou funkce úpravy barev nebo hnízdící (*nesting*) pravidla, která nejsou v CSS k dispozici. Další výhodou je, že poskytují přístup k proměnným CSS. [13]

Preprocesory posilují CSS tím, že odstraňují neefektivitu a umožňují vytvářet jednodušší a více logické webové stránky. Kód CSS se snadno udržuje a tím šetří kodérovi čas. S CSS preprocesorem lze snadno aplikovat princip *Don't Repeat Yourself* (DRY), který pomáhá vyhýbat se repetitivnímu kódu. [12]

Mezi populární preprocesorové jazyky patří Less a Sass. Při psaní kódu v jazyku preprocesoru je potřebné použít program nebo skript, aby kód v tomto jazyce zkompiloval do CSS, kterému prohlížeč rozumí [13]. Na obrázku Obrázek 3 je znázorněn proces kompilace z SCSS formátu jazyku Sass do CSS.


**Obrázek 3: Kompilace z SCSS formátu do CSS [13]**

Čím více je projekt komplexní a čím více lidí upravuje CSS kód, tím více je doporučené CSS preprocesor použít [14]. Zároveň CSS metodiky pro organizaci kódu mají bez použití preprocesorů menší dopad [15], jelikož pomocí preprocesoru je možné organizovat kód přes jednotlivé části kódu vztažené ke konkrétní komponentě uživatelského rozhraní.

## 2.1 Vlastnosti preprocesorů

Kapitola 2.1 byla zpracována zejména z těchto zdrojů: [12], [13].

### 2.1.1 Proměnné

Pomocí proměnné lze uložit do šablon stylů informace, které lze znovu použít. Týká se to jakýkoliv vlastností jako jsou barvy, fonty atd. Použitím proměnných, lze následně tuto proměnnou upravit pouze na jednom místě. Kodér tak nemusí ztrácet čas hledáním, kde všude byla v celém CSS kódu barva použita a nahrazovat ji.

Pro deklaraci proměnné se v Sass používá symbol *„$“.*

```
$color-red: dd3b2a

button {
  color: $color-red;
}
```

**Kód 1: Příklad použití proměnné s použitím preprocesoru Sass**

### 2.1.2 Hnízdění (Nesting)

Preprocesory umožňují vnořovat selektory CSS jako je tomu podobně u HTML.

```
nav {
  margin: 1em 2em;

  ul {
    list-style: none;
    padding: 0;
    margin: 0;

   li {
     display: block;
     width: 6em;
     height: 2em;
    }
  }
}
```

**Kód 2: Příklad hnízdění v Sass [2]**

```
nav {
  margin: 1em 2em;
}

nav ul {
  list-style: none;
  padding: 0;
  margin: 0;
}

nav ul li {
  display: block;
  width: 6em;
  height: 2em;
}
```

**Kód 3: Příklad zkompilovaného *.scss* souboru do *.css* [2]**

### 2.1.3 Import

Import umožňuje vytvářet soubory, do kterých lze umístit proměnné, různé kombinace a opakovaně použitelný kód. Importy v Sass fungují podobně jako importy v CSS, protože obsahují šablony stylů, které jsou importovány.

### 2.1.4 Mixiny

Mixiny slouží k vytváření částí CSS, které se mohou znovu použít v šablonách stylů. Mixiny lze dále konfigurovat předáním parametrů a nastavením výchozích hodnot pro dané parametry. Mixiny jsou velmi užitečné, jelikož snižují dlouhé přepisování kódu.

```
@mixin rounded-corners($radius:5px) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
      -ms-border-radius: $radius;
          border-radius: $radius;
}

.container { @include rounded-corners(10px); }
```

**Kód 4: Příklad mixinu s parametrem a výchozími hodnotami v Sass [13]**

### 2.1.5 Extend

Extend umožnuje preprocesorům sdílet kolekci vlastností CSS do jednoho nebo několika selektorů. Lze tak psát preprocesorový kód podle principu DRY.

```
.messagebox {
  font-size:1em;
  line-height:40px;
}
.successBox {
  @extend .messagebox;
  color:#0F0;
}
```

**Kód 5: Příklad použití extend v Sass [13]**

### 2.1.6 Matematické operátory

Preprocesory poskytují standardní matematické operátory. Lze tak použít matematické operace s čísly nebo barvami.

```
.mainContent {
  float: left;
  width: 750px / 960px * 100%;
}
```

**Kód 6: Příklad použití matematických operátorů [13]**

## *2.2 Sass*

Syntactically Awesome Stylesheets (Sass) je nejpopulárnější preprocesor. Jedná se o jazyk preprocesoru, který navrhly a vyvinuly Hampton Catlin a Natalie Weizenbaum [13]. Jejich nová syntaxe umožnila kodérům CSS používat typ zkratek typických pro skriptování.

Sass je open-source preprocesor a jeho původní verze byla napsána v Ruby. Pokud je SassScript kompilován, generuje CSS kód pro různé selektory, které jsou v soboru Sass. Sass monitoruje soubory *.sass* nebo *.scss* a generuje výstupní soubor *.css*, kdykoliv se do *.sass* nebo *.scss* souboru něco uloží. [13]

## *2.3 Less*

Leaner Style Sheets (Less) je další populární preprocesorový jazyk. Stejně jako Sass, jde o open-source. Původně byl napsán v Ruby, avšak novější verze je napsána v jazyku Javascript. Less je jednoduchý na naučení a podobný jako Sass, ale chybí mu

pokročilé logické funkce programování a má menší rozdíly v syntaxi na rozdíl preprocesoru Sass. [2]

Hlavní rozdíl Lessu oproti jiným preprocesorům je, že umožňuje kompilaci v reálném čase skrze *less.js* v prohlížeči [2].

# 3 Doporučené zásady při návrhu CSS kódu

CSS je snadné se naučit, avšak brzy se může stát problematickým v jakémkoli měřítku. Není toho mnoho, co by kodér mohl udělat, aby změnil fungování CSS, ale může změnit způsob, jakým jej vytváří a strukturuje.

Při práci na velkých dlouhodobých projektech s desítkami vývojářů různých specializací a schopností je důležité, aby všichni pracovali jednotně a dále [16]:

- udržovali šablony stylů,
- udržovali kód transparentní, rozumný a čitelný,
- udržovali šablony stylů škálovatelné.

Aby byly tyto cíle splnitelné, musí se použít řada technik při psaní CSS. Doporučení a přístupy, které tyto cíle pomohou splnit budou podrobněji rozvedeny v další kapitole.

Následující doporučení byla zpracována zejména z těchto zdrojů: [16], [17].

## *3.1 Syntaxe a formátování*

Sada pravidel týkající se syntaxe a formátování je jednou z nejjednodušších forem šablon stylů. Způsob psaní CSS by měl vypadat tak, aby byl výstup jasný celému týmu. Kód, který vypadá „čistě“, vytváří příjemnější prostředí pro práci a vybízí tým k tomu, aby ho nadále udržovali „čistý“ a přehledný.

Doporučené zásady pro „čistý“ a přehledný CSS kód:

- šířka řádku by měla obsahovat 80 znaků,
- víceřádkový kód CSS,
- odsazení pomocí mezerníku (2 mezery) nikoli tabulátoru,
- smysluplné použití prázdných řádků.

### 3.1.1 Soubory

S příchodem preprocesorů kodér častěji rozděluje CSS kód do více souborů. V adresáři by měl být jeden hlavní soubor. V případě, že se preprocesor nepoužívá, je i přesto doporučeno rozdělovat CSS kód na jednotlivé části a do jednotlivých

souborů. Tyto jednotlivé sobory by měly být následně zřetězeny do jednoho hlavního.

Podle Michálka [17], by neměla délka kódu v soboru být větší než 200 řádků. Zároveň by se mělo zamezit mícháním více komponent a funkcí v souborech.

### 3.1.2 Tabulka obsahu

Tabulka obsahu obsahuje název sekce a krátké shrnutí, co to je za sekci a co dělá. Její udržování je poměrně podstatné. Týmu poskytuje přehled toho, co CSS kód obsahuje, co dělá a v jakém pořadí jsou jednotlivé sekce. Z tohoto důvodu je důležité, aby byl kodér pečlivý a udržoval pravidelně tabulky obsahu.

```
/**
 * CONTENTS
 *
 * SETTINGS
 * Global...............Globally-available variables and config.
 *
 * TOOLS
 * Mixins...............Useful mixins.
 *
 * GENERIC
 * Normalize.css........A level playing field.
 * Box-sizing...........Better default `box-sizing`.
 *
 * BASE
 * Headings.............H1–H6 styles.
 *
 * OBJECTS
 * Wrappers.............Wrapping and constraining elements.
 *
 * COMPONENTS
 * Page-head............The main page header.
 * Page-foot............The main page footer.
 * Buttons..............Button elements.
 *
 * TRUMPS
 * Text.................Text helpers.
 */
```

**Kód 7: Příklad tabulky obsahu v CSS kódu [16]**

### 3.1.3 Šířka řádku

Pokud je to možné, doporučuje se omezit šířku řádku na 80 znaků, aby bylo pohodlnější:

- otevřít několik souborů vedle sebe,
- zobrazit CSS kód například v GitHubu nebo jiných terminálních oknech,
- pohodlnější zapisovat komentáře.

### 3.1.4 Titulek

Každá část CSS souboru by měla začínat titulkem. Nadpis sekce začíná symbolem „*#*“, který zúží kodérovi výsledky vyhledávání a pomůže cíleněji vyhledávat. Nadpis by se měl označit zpětnými lomítky „*/““*“, aby byl označen jeho začátek a konec.

V případě velkého projektu, kde jsou části CSS rozděleny do několika souborů, by se měl titulek psát na začátku každého souboru. U malých projektů, kde jsou všechny sekce v jednom CSS souboru, by mělo být před každým titulkem pět volných řádků. To usnadní kodérovi vyhledávání mezi sekcemi.

```
/*------------------------*\
  #PRVNÍ-SEKCE
\*------------------------*/
selektor {deklarace}




/*------------------------*\
  #DRUHÁ-SEKCE
\*------------------------*/
selektor {deklarace}
```

**Kód 8: Příklad titulků u jednotlivých sekcí**

### 3.1.5 Anatomie sady pravidel

Doporučená pravidla při zápisu sady pravidel jsou následující:

- Při zápisu sady pravidel by související selektory měly být zapsány na stejném řádku. Nesouvisející selektor by měl být na dalším řádku.

- Mezi názvem selektoru a otevírací složenou závorkou by měla být mezera.
- Vlastnosti a jejich hodnoty by měly být v jednom řádku. Zároveň by za dvojtečkou měla být mezera.
- Každá další deklarace by měla být na novém řádku. Středník ukončující deklaraci je na stejném řádku jako samotná deklarace.
- První deklarace za otevírací složenou závorkou by měla začínat na novém řádku.
- Uzavírací složená závorka je na novém řádku. Není v jednom řádku s deklarací.
- Každá deklarace je odsazena dvěma mezerami.

```
selektor {
  vlastnost1: hodnota1;
  vlastnost2: hodnota2;
  deklarace;
}
```

**Kód 9: Anatomie sady pravidel**

### 3.1.6 Víceřádkové CSS

Kód CSS by měl být rozepsán na více řádcích. Kodér tím sníží sloučení konfliktů, jelikož každá část funkce bude existovat na svém vlastním řádku. Každý řádek bude tak nést vždy jednu změnu.

Roberts [16] uvádí jako výjimku podobnou sadu pravidel, která nesou pouze jednu deklaraci (viz. třída *.icon1* na ukázce kódu Kód 10). Taková sada může být v jednom řádku, pokud splňuje pravidlo jedné změny a sdílí podobnosti, které nejsou potřeba číst tak důkladně jako u jiných sad pravidel.

```
.icon {
  background-repeat: no-repeat;
  background-image: url('icon1.png');
}

.icon1 { background-position: 30% 50%; }
```

**Kód 10: Příklad víceřádkového a jednořádkového CSS**

### 3.1.7 Odsazení

Odsazené by neměly být pouze jednotlivé deklarace, ale i celé související sady pravidel. Tím se vyjádří jejich vzájemný vztah. Kodér tak na první pohled vidí, že *selektor1* je uvnitř *selektor2*.

```
selektor1 { deklarace }

  selektor1_selektor2 { deklarace }
```

**Kód 11: Příklad odsazení**

Preprocesor Sass umožňuje funkci hnízdění (*nesting*). Při odsazování v Sass je doporučeno odsadit také dvěma mezerami a zanechávat prázdný řádek na začátku a konci vnořené sady pravidel.

```
selektor1 {
  deklarace;

  selektor2 {
    deklarace;
  }

}
```

**Kód 12: Příklad funkce hnízdění preprocesoru Sass**

### 3.1.8 Zarovnání

Kodér by se měl snažit zarovnat společné, související nebo identické řetězce v deklaracích.

```
selektor {
  position: absolute;
  top:    0;
  right:  0;
  bottom: 0;
  left:   0;
  margin-right: -10px;
  margin-left:  -10px;
  padding-right: 10px;
  padding-left:  10px;
}
```

**Kód 13: Příklad zarovnání deklarací [16]**

### 3.1.9 Smysluplné použití prázdných řádků

Využití prázdných řádků by mělo být promyšlené a smysluplné. Roberts [16] doporučuje následující pravidla použití prázdných řádků v CSS kódu:

- Jeden prázdný řádek by měl být mezi úzce související sadou pravidel.
- Dva prázdný řádky by měly být mezi související sadou pravidel.
- Pět prázdných řádků mezi jednotlivými sekcemi.

```
/*------------------------*\
  #PRVNÍ-SEKCE
\*------------------------*/
selektor1 {deklarace}

  selektor1_selektor2 { deklarace }


selektor3 { deklarace }




/*------------------------*\
  #DRUHÁ-SEKCE
\*------------------------*/
selektor {deklarace}
```

**Kód 14: Příklad smysluplného využití prázdných řádků**

## 3.2 Komentáře

Dalším doporučení je, že je CSS kód potřeba komentovat. Obzvláště v případě velkého projektu, na kterém pracuje několik kodérů a zároveň přichází noví, kteří se musí v kódu zorientovat. Okomentování jednotlivých částí usnadňuje novému členu týmu rychle se v kódu zorientovat. Zpravidla by se měly komentovat jakékoliv části kódu, u kterých není na první pohled zřejmé, co je jejich účelem, souvisí s jinou částí kódu nebo jeho změna bude mít za následek změnu jiného kódu. Obzvlášť by se měly komentovat části, které dědí některé styly, jako je tomu u rozdělení kódu do více souborů.

Je doporučeno například i na začátku mít rozsáhlejší komentář v podobě dokumentace, kde je popsáno, jaké jsou rozměry, barvy nebo jiné informace o vzhledu základních komponent webu.

U preprocesorů je možnost vytvořit komentáře, které se nebudou kompilovat do výsledného CSS souboru. Zpravidla by měly být takto okomentovány ty části kódu, které se ani do výsledného CSS kódu kompilací nedostanou.

V případě použití „*// TODO*“ komentáře by měl být vysvětlen důvod [17].

```
// Tento komentář se do výsledného CSS kódu nedostane
$colorWhite: white;

/**
 * Tento komentář se kompilací dostane do výsledného CSS kódu
 */
h2{
  color: $colorWhite;
  font-size: 22px;
  font-style: italic;
}
```

**Kód 15: Příklad použití komentářů při použití preprocesoru**

## 3.3 Pojmenování

Při vytváření CSS kódu by se mělo dbát na konvenci pojmenování, pomocí které bude zřejmé, co třída dělá, kde jí lze použít a s čím dalším může souviset.

Například autor, od kterého vychází tyto doporučení, Roberts [16], využívá podobnou konvenci pojmenování metodiky BEM u větších a více vzájemně propojených částí uživatelského rozhraní, které vyžadují několik tříd.

Metodika BEM je podrobněji vysvětlena v kapitole 4.2.

## 3.4 CSS selektory

Podle Robertse [16] jsou CSS selektory jedním z nejzásadnějších a nejdůležitějších aspektů pro psaní udržovatelného a škálovatelného CSS. Jejich specifičnost, přenositelnost a znuvupoužitelnost má dopad na množství použitého CSS kódu, ale zároveň mohou kodérovi zamotat hlavu. Proto je důležité si dát pozor, aby selektory nebyly psány příliš globálně a dalekosáhle, aby se zamezilo většímu množství CSS kódu. Selektor by měl být explicitní a jednoznačný.

V poslední době se více a více používá komponentní přístup UI pro zajištění znuvupoužitelnosti. Podle Robertse [16] by se pro možnost duplikace komponent napříč projekty měly využívat třídy. Použití identifikátoru je příliš konkrétní a na dané stránce lze použít pouze jednou. V tomto případě jsou třídy opakem, které lze použít více než jednou. Pro zajištění znuvupoužitelnosti by se mělo zajistit, aby styl komponent nebyl závislý na jejich umístění a zároveň, aby název nebyl příliš jednoznačný a určitý. Názvy komponent by neměly popisovat konkrétní popis obsahu nebo případ užití.

## 3.5 Principy architektury

CSS potřebuje pro svoji jednoduchost, uvolněnost a divokou povahu, řídit a spravovat pomocí přísné a specifické architektury. Dobrá architektura pomůže s kontrolou specifičnosti, bude vynucovat konvence pojmenování, spravovat a vytvářet čisté prostředí a obecně spravovat CSS konzistentnější a pohodlnější.

Podle Robertse [16] žádný nástroj nebo preprocesor nepomůže vylepšit CSS. Tvrdí, že nejlepším nástrojem pro práci s tak volnou syntaxí je sebekázeň, svědomitost, píle a architektura, která pomůže prosadit a usnadnit tyto vlastnosti.

Pomocí architektury lze zajistit konzistentní a čisté prostředí, pomůže přizpůsobit se lépe změnám, rozšiřovat zdrojový kód, zajistit podporu pro opětovné použití a efektivnost, a zvýší výkonnost. Ve většině případů, to znamená použít komponentovou architekturu, rozdělenou do spravovatelných, většinou pomocí preprocesoru, modulů.

Principy architektury jsou následující.

- Metodika CSS.
- Princip jedné odpovědnosti – CSS kód by měl být složen z menších částí tzn. tříd, které jsou zaměřeny na konkrétní a limitovanou funkci. To znamená, že UI by mělo být rozloženo na nejmenší části, z nichž každá má pouze jednu odpovědnost. Každá je zaměřená jen na jednu věc a lze je tak snadno kombinovat a skládat, aby vytvořily mnohem univerzálnější a komplexnější konstrukce. Všechny části CSS kódu by se měly zaměřit a dělat pouze jednu věc.

- Princip otevření/uzavření – Entity jako jsou třídy, moduly, funkce atd. by měly být otevřené pro rozšíření, ale uzavřené pro úpravy.
- DRY – Neopakovat se neboli zkráceně DRY z anglického *Don't Repeat Yourself* je princip, jehož cílem je omezit opakování klíčových informací na minimum. Důležité je normalizovat a používat abstraktní smysluplné opakování.
- Složení přes dědičnost – Princip skládání menších částí CSS kódu do komplexnějších celků.

# 4 Metodiky CSS

Při vývoji webových aplikací, zejména v týmu, má mnoho webových vývojářů řadu problémů spojených se zdánlivě triviálním úkolem a to „porozumět kódu“. Důvody mohou být různé, ale primárně se týkají struktury samotné aplikace a přístupu k vývoji kódu. Vývojář, který přijde k projektu, který už nějakým způsobem funguje, tráví mnoho času procházením struktury, kód aplikace a zjišťováním v jakém aktuálním stavu se aplikace nachází.

Pro zajištění funkčnosti aplikací, zlepšení srozumitelnosti a čitelnosti kódu v průběhu jejich vývoje nebo při výměně vývojáře je důležité hledat řešení problémů a snažit se zjednodušit projekt tak, aby byl lépe pochopitelný. Vhodná metodika zajistí jednotný vzhled a správnou funkčnost aplikace.

Objevilo se několik metodik, které návrhářům pomáhají psát škálovatelné a modulární HTML a CSS kódy. Díky těmto metodikám se stávají weby lépe udržovatelné.

V následujících kapitolách jsou popsané následující vybrané metodiky:

- Block, Element, Modifier (BEM),
- Object Oriented CSS (OOCSS),
- Scalable and Modular Architecture for CSS (SMACSS),
- Invert Triangle CSS (ITCSS),
- Atomický web design.

Tyto metodiky mají za cíl vytvářet smysluplný, udržitelný a rozšiřitelný kód s pomocí různých přístupů.

## *4.1 Object Oriented CSS (OOCSS)*

Metodiku objektově orientovaného CSS poprvé v roce 2009 představila Nicole Sullivan na konferenci Web Directions North v Denveru [18]. OOCSS je určitý přístup k psaní CSS, který je rychlý, udržovatelný a založený na standardech.

Metodika je zaměřena na psaní stylů pro jednotlivé CSS objekty [19]. CSS objekt je opakující se vizuální vzor, který může být abstrahován kouskem HTML, CSS a případně Javascriptem a lze ho poté znovu použít v projektu.

Hlavní dva principy OOCSS podle Nicole Sullivan [18]:

**Oddělení struktury a vzhledu**

- Při oddělování struktury a vzhledu by se měly pro pojmenování objektů a jejich komponent použít třídy. Například v případě použití obrázku by se měla pojmenovat třída jako *class =“img“.*
- U vzhledu je důležité si definovat opakující se vizuální prvky jako je pozadí a ohraničení, které se mohou kombinovat s různými objekty. Kombinováním se docílí rozmanitosti vzhledu bez nutnosti použití velkého množností kódu.

**Oddělení kontejneru a obsahu**

- Objekt by měl vypadat stejně bez ohledu na to kde je umístěn. Podle Sullivan [18], by se místo stylování například konkrétní úrovně nadpisu *<h2>* pomocí *.mujObjekt h2 {..}* měla vytvořit a použít třída, která popisuje tuto úroveň nadpisu *<h2>* například jako *<h2 class=“kategorie“>*. Všechny *<h2>* bez třídy budou vypadat stejně a stejně tak ty, které budou mít definovanou třídu. Tím se zajistí, že se nebudou muset přepisovat styly.

## *4.2 Block, Element and Modifier (BEM)*

Metodika Blok, element a modifikátor se zrodila ve společnosti Yandex. Než začala tato společnost pracovat na dalších velkých projektech, byl projekt Yandex set statických HTML stránek, které sloužily jako základ pro vytvoření XSL šablon. V případě, kdy se v HTML provedly změny, tak všechny tyto úpravy musely být ručně zkopírovány do XSL. A to platilo i v opačném případě. Pokud nastaly změny v šablonách XSL, tak se tyto změny musely ručně upravit i v HTML. [20]

V roce 2006 společnost Yandex začala vyvíjet své první velké projekty Ya.Ru a Yandex.Music během nichž byly odhaleny hlavní nevýhody v přístupu při vývoji. První nevýhoda byla, že se těžko vybíral název třídy. Další nevýhodou bylo, že jakékoliv změny v kódu jedné stránky ovlivnilo kód v dalších stránkách. [20]

Aby se těmto problémům vývojáři vyhnuli, potřebovali si nastavit pravidla pro práci s třídami, značkami, vizuálními komponentami atd. Po několika letech práce na projektu se tak postupně zformovala metodika BEM.

Zkratka BEM je sestavená z počátečních písmen entit *Block*, *Element* a *Modifier*, na kterých metodika stojí. Jedná se o komponentní přístup k vývoji webu. Myšlenkou BEM metodiky je rozdělit uživatelské rozhraní na nezávislé bloky [20]. Díky BEM se stává vývoj rozhraní snadným a rychlým i při složitém uživatelském rozhraní. Tímto přístupem se zároveň umožňuje opětovné použití existujícího kódu bez kopírování a vkládání.

**Blok**

V HTML jsou bloky reprezentovány atributem *class*. Jedná se o funkčně nezávislou součást stránky, kterou lze znovu použít. Tím se usnadňuje proces vývoje a podpory na projektu. Bloku by se neměl nastavovat vnější okraj (*margin*) a jeho pozice na stránce, aby se zajistila nezbytná nezávislost pro opětovné použití bloků a změny jejich pozice na stránce bez úpravy CSS kódu. Aby se zajistila jejich nezávislost, tak by se podle metodiky neměly použít v blocích CSS tagy a identifikátory [20]. Důležité je také zmínit, že se bloky mohou do sebe libovolně zanořovat [21].

Podle konvencí pojmenování metodiky BEM by měl být zvolen název bloku takový, aby názvem bylo vyjádřeno o co se jedná. Pokud se jedná o menu, název by měl být *menu*. Pokud se jedná o tlačítko, název bloku by měl být *tlačítko*. Název bloku má popisovat jeho účel nikoliv jak blok vypadá. [20]

**Element**

Element je součást bloku, kterou nelze samostatně použít. Její existence má smysl pouze v rámci bloku [22]. Je sémanticky vázaná na svůj blok.

Elementy se mohou do sebe libovolně zanořovat. Měl by být součástí bloku nikoli jiného elementu. To znamená, že názvy elementů nemohou definovat hierarchií např. *blok__element1__element2*. [20]

Název elementu by měl být zvolen tak, aby popisoval svůj účel. Stejně jak je tomu u názvu bloku. Zvolený název by měl odpovídat na otázku: „Co to je?“. Příklad názvu může být např. *položka* nebo *text*. Podle metodiky BEM by neměl název

popisovat stav, co je to za typ nebo jak to vypadá. Název elementu by zároveň měl být separován od názvu bloku pomocí dvou podtržítek. Struktura názvu elementu by měla tedy vypadat: *blok-název__element-název*. Tím se zaručí, že element je závislý na bloku. Jak již bylo zmíněno dříve, element je součástí bloku a neměl by se používat odděleně od bloku. Blok však element nemusí obsahovat. [20]

**Modifikátor**

Modifikátor je entita, která definuje vzhled, stav nebo chování bloku či elementu. Měl by změnit vzhled, chování nebo stav entity, nikoli jej nahradit. Z pohledu BEM metodiky nelze modifikátor použít samostatně (izolovaně) od upraveného bloku nebo elementu. [20]

Název modifikátoru by měl zvolen tak, aby popisoval jednu z těchto tři skupin:

- vzhled,
- stav,
- chování.

Měl by být oddělen od názvu bloku nebo elementu jedním podtržítkem: *blok-název_modifikátor-název*.

## 4.3 Scalable and Modular Architecture for CSS (SMACSS)

Škálovatelná a modulární architektura pro CSS zkráceně SMACSS je CSS metodika od autora Jonathana Snooka a je založená na kategorizaci [12].

Snook [23] definoval ve své metodice 5 kategorií:

- Base
- Layout
- Module
- State
- Theme

Podle Snooka [23], se často míchají styly z každé z výše uvedených kategorií. To vede ke zbytečné složitosti, která vyplývá z propletení jednotlivých pravidel.

Kategorizace by měla umožnit vývojáři uvědomit si, co se snaží stylovat a donutit si během procesu vývoje pokládat otázky typu jak a proč bude takto kódovat. Účelem kategorizace je zredukovat opakující se vzory v designu. Výsledkem by měl být zredukovaný kód, se snadnou údržbou a větší důslednost v rámci uživatelských zkušeností [23]. Metodika SMACSS by měla pomoci lépe kategorizovat styly podle toho o co se jedná.

Následně budou představeny a popsány jednotlivé kategorie SMACSS podle Snooka [23].

**Base**

Base, neboli základní kategorie slouží k nastavení výchozích hodnot. Základní pravidlo se aplikuje na selektory elementů, atributů, pseudotříd, selektor potomků nebo selektor sourozenců. Výjimkou jsou selektory tříd a ID.

V Base jsou zahrnuta nastavení velikosti záhlaví, výchozí styl odkazů, font styly a pozadí. Týká se to *paddingu*, *marginu*, rámečků, fontů a dalších vlastností a CSS Reset. Příkladem těchto elementů jsou *form*, *body*, *a:hover* nebo *a* [23]. Kategorie definuje defaultní stylování elementu, jak by mělo vypadat kdekoliv se použije na stránce.

**Layout**

Layout, neboli kategorie rozvržení definuje vzhled a rozvržení elementů na stránce. Jedná se o rozdělení stránky na sekce s prvky. Spadá sem záhlaví, zápatí nebo postranní panel. Obecně styl rozložení má pouze jeden sektor (ID nebo název třídy). Snook [23] označuje elementy rozvržení předponou třídy *l-*, aby byly jasně rozpoznatelné v kódu.

**Modules**

Moduly jsou opakovaně použitelné části designu. Jsou umístěny v komponentách rozvržení a mohou být umístěny i do jiného modulu. Podle Snooka [23], by měl být každý modul navržen tak, aby existoval jako samostatná část. To zajišťuje flexibilnost stránky a při přesunu modulů do jiných částí komponent rozvržení nedojde k jejich rozbití. U modulů by se měly používat pouze názvy tříd,

nikoliv ID či selektory elementů. Příkladem modulu může být například navigační lišta, kolotoče (*carousely*), či tabulky atd.

**State**

Kategorie popisuje vizualizaci modulů nebo rozvržení pomocí stavů. Příkladem může být sbalené/rozbalené menu, validní/nevalidní formulář nebo viditelný/skrytý modul atd. Popisují i jak budou vypadat moduly například na domovské stránce a jak na jiné stránce. Stavy se označují prefixem *is-*.

**Theme**

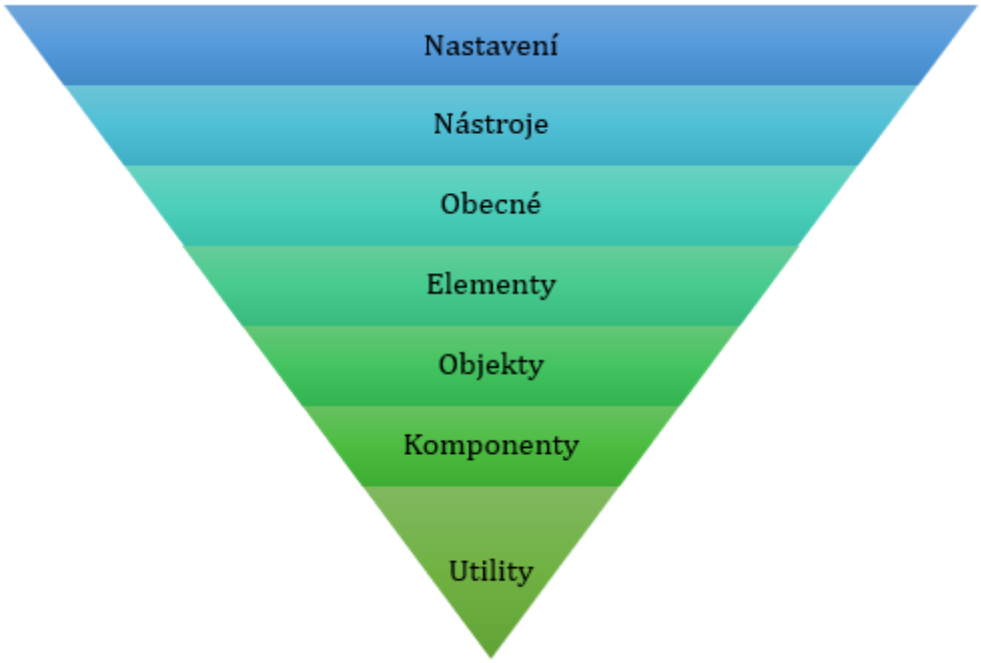
Téma je volitelná kategorie, kterou není nutné v projektech využít [12]. Témata deklarují vzhled modulů nebo rozložení podobně jako je to u stavů.

## *4.4 Inverted Triangle CSS (ITCSS)*

Autorem metodiky Inverted Triangle CSS je Harry Roberts. Metodika ITCSS se pokouší řešit specifičnost selektoru pomocí přirozené priority CSS. Jedná se o metodiku pro organizaci CSS kódu, aby byl co nejlépe udržitelný a škálovatelný. Šablony stylů jsou zařazeny do vrstev na základě jejich účelu velmi odlišně na rozdíl od metodik OOCSS a SMACSS. [12]

Základna trojúhelníku (Obrázek 4) představuje nejširší a nejméně konkrétní pravidla, přičemž nejjasnější pravidla jsou na vrcholu trojúhelníku.

Princip metodiky ITCSS spočívá v rozdělené CSS do sedmi skupin, které mají přesně definované pořadí:




**Obrázek 4: Vrstvy metodiky ITCSS [12]**

**Nastavení**

První vrstvou metodiky ITCSS je Nastavení. Tato vrstva jinými slovy skupina je určená pro preprocesory. Vrstva obsahuje definice proměnných a nastavení celého webu. Jedná se například o nastavení fontů a barvy či velikostí. [12]

**Nástroje**

Do této vrstvy spadají obecné mixiny a funkce, které mohou být znovu použity. Stejně jako první vrstva Nastavení, tak i tato skupina je určená pro preprocesory. Tyto první dvě vrstvy nejsou zkompilované do CSS výstupu [24].

**Obecný**

Teprve od této vrstvy se začíná generovat CSS kód. Spadá sem normalizace a resetování stylů, CSS proměnné a nastavení *box-sizing* pro vytvoření základu pro styly. [12]

**Elementy**

Tato vrstva definuje základní styly pro standardní elementy HTML. Příkladem může být *p* nebo *h1*. Společně s předchozí vrstvou Obecný, tato skupina stylizuje výchozí vzhled HTML elementů [25].

**Objekty**

Tato vrstva obsahuje třídní selektory, které definují znovupoužitelné *layouty* a nedekorované návrhové vzory [24].

**Komponenty**

Zde se nachází kategorie konkrétních UI komponent, které se většinou skládají z objektů a komponent [24].

**Utility**

Poslední skupinou jsou pomocné třídy, u kterých je potřeba mít možnost přepsat cokoli co jim ve vrstvách trojúhelníku předchází. Tyto třídy ovlivňují CSS vlastnosti [25]. Jedná se o vrchol trojúhelníku, který zahrnuje nejexplicitnější typy pravidel s nejužším zaměřením [26]. Tato vrstva má v podstatě hlavní slovo v tom, jaká bude barva pozadí nebo zarovnání textu.

## 4.5 Atomický web design

Autor Atomického designu Brad Frost [8] se při hledání nové metodiky pro vytváření návrhových designových systému nechal inspirovat chemií, přesněji složením hmoty. Popis této metodiky vychází z toho, že atom je základem veškeré hmoty. Spojením několika atomů do skupin se vytvoří molekuly a spojením několika molekul ve fungující celek vzniká organismus. Takové relativně složité struktury se mohou pohybovat od jednobuněčných organismů až po složité organismy, jako je člověk [8].

Atomický design je složený z pěti odlišných fází viz. Obrázek 5Obrázek 4. Tyto fáze společně vytvářejí návrh rozhraní systému promyšlený a hierarchický. Atomový design pomáhá pohlížet na uživatelské rozhraní jako na soudržný celek a zároveň na souhrn částí. Každá z pěti fází na obrázku Obrázek 5 hraje klíčovou roli v hierarchii návrhových systémů rozhraní. [8]


**Obrázek 5: Fáze atomického designu [8]**

**Atomy**

Atomy jsou základní prvky HTML. Slouží jako základní stavební kameny uživatelského rozhraní, které nelze dále členit. Každý atom neboli prvek UI má své vlastní jedinečné vlastnosti. Příkladem atomů mohou být popisky (*label*) formuláře, inputy nebo tlačítka. [8]

**Molekuly**

Molekuly jsou soubory atomů, které společně fungují jako celek. Jsou to jednoduché, funkční, opakovaně použitelné komponenty, které lze zasadit do širších souvislostí. Tvoří relativně jednoduché komponenty uživatelského rozhraní. UI designeři a vývojáři tak dodržují princip jedné odpovědnosti, kdy každý objekt je zodpovědný za jednu logickou funkcionalitu a měl by jí provést správně. Vytváření jednoduchých UI molekul usnadňuje testování, podporuje opětovné použití a podporuje konzistenci v celém rozhraní. [8]

Spojením atomů do molekuly získávají jednotlivé atomy svůj účel. Frost [8] účel atomů vysvětluje na příkladu, kdy pomocí tlačítka, které představuje atom, se při jeho stisknutí odešle formulář. Právě vyhledávací formulář je molekula, která je vytvářena skupinou atomů jako je popisek, vstupní pole a tlačítko. Výsledkem je jednoduchá, přenosná, opakovaně použitelná součást, kterou lze umístit kdekoli, kde je potřeba vyhledávací funkce.

**Organismy**

Organismy jsou poměrně složité UI komponenty, které tvoří jednotlivé části rozhraní. Tyto komponenty jsou složené ze skupin molekul a/nebo atomů a/nebo jiných organismů [8]. Příkladem je například vyhledávací formulář, který se vloží do

organismu, kterým je záhlaví. Tento organismus by dále mohl být sestaven z různých elementů jako je navigační list nebo logo. Postupné budování od molekul ke komplikovanějších organismů poskytuje UI designerům a vývojářům důležitý smysl pro kontext. Organismy demonstrují menší, jednodušší komponenty v akci a poskytují odlišné vzory, které lze znovu a znovu použít. Příkladem, který uvádí Frost [8], může být organismus produktového gridu, který lze použít při zobrazení produktů a pak také při výpisu z hledání souvisejících produktů.

**Šablony**

Šablony jsou objekty na úrovni stránky. Umisťují komponenty do layoutu a představují základní strukturu obsahu návrhu. Příkladem šablony může být předloha domovské stránky, kde je použit organismus záhlaví. Šablona zobrazuje všechny potřebné části stránky, které společně fungující, což dává význam pro relativně abstraktní molekuly a organismy. [8]

Při tvorbě efektivního návrhového systému je důležité demonstrovat, jak komponenty vypadají a fungují společně v kontextu grafického rozvržení, aby se dokázalo, že jednotlivé části tvoří dobře fungující celek. V šablonách se zaměřuje spíš na základní strukturu obsahu stránky než na její konečný obsah. Návrhové systémy musí zohledňovat dynamickou povahu obsahu a je tedy velmi užitečné formulovat důležité vlastnosti komponent jako jsou velikosti obrázků a počet znaků pro nadpisy a textový obsah. [8]

**Stránky**

Stránky jsou konkrétní instance šablon. Jedná se o nejkonkrétnější fází atomického designu, která je velmi důležitá.

Právě v této fázi je zobrazeno propojení komponent, které vytvářejí funkční uživatelské rozhraní, a jak toto finální rozhraní uvidí samotní uživatelé. Stránky prezentují, jak vypadá uživatelské rozhraní se skutečným obsahem, který se „nalije“ do šablony. Příkladem může být šablona pro domovskou stránku, do které se vloží text, obrázky a média, a zobrazí se tak stránka se skutečným obsahem v akci. [8]

Kromě výše zmíněného je tato fáze nezbytná pro testování, kdy si lze zobrazit, jak všechny tyto vzory obstojí v případě, kdy je v návrhovém systému použit skutečný obsah. V této vrstvě se testuje, zda vše vypadá a funguje tak jak má,

a případně se vrátit zpět a upravit molekuly, organismy a šablony, tak aby vše sedělo k potřebám obsahu. [8]

# 5 Praktické použití metodik a jejich vyhodnocení

Cílem praktické části diplomové práce je představit jednotlivé použití metodik a zhodnotit jejich využití. Tato kapitola demonstruje použití jednotlivých metodik CSS na vybrané ukázce externího stylopisu tedy CSS souboru.

## *5.1 Vybraný CSS soubor*

Vybraný CSS soubor, na kterém je demonstrováno praktické použití metodik, je jedním ze stylopisů dostupných na webu CSS Zen Garden[1]. Jedná se o web Davida Shea, na kterém demonstroval, čeho všeho lze dosáhnout pomocí CSS. Na svém webu Shae zveřejnil dokument HTML a vyzval ostatní kodéry, aby přispěli svými vlastními externími stylopisy a dodali tak dokumentu vizuální design. Všechny tyto návrhy používají jeden stejný zdrojový dokument HTML. Různé vzhledy jednotlivých stránek jsou docíleny pomocí jednotlivých externích souborů CSS. Ukázky jednotlivých stránek demonstrují, že CSS může být pro tvorbu designu robustný a mocný návrhový nástroj. [2]

Pro demonstraci praktického použití jednotlivých metodik byl vybrán jeden z těchto externích stylopisů s názvem „A Robot Named Jimmy“ [27][2] (viz. Obrázek 6) od společnosti Meltmedia. Design stránky byl navrhnut Guillermo Montem a kód CSS napsán Jimmem Kingem.

[1] Odkaz: www.csszengarden.com

[2] Zdrojový kód byl použit v souladu s podmínkami licence „CC BY-NC-SA 1.0“. Odkaz: https://creativecommons.org/licenses/by-nc-sa/1.0/


**Obrázek 6: Webová stránka „A Robot Named Jimmy“ [27]**

Vybraný stylopis byl původně napsán v jazyce preprocesoru LESS. Autor Jim King [28] strukturu stylopisu rozdělil do několika následujících složek[28]:

- *Common* – V této složce se nachází jednotlivé soubory LESS pro animace, mixiny, normalizace, základní vlastnosti HTML elementů a soubor pro proměnné barev, fontů atd.
- *Elements* – Tato složka obsahuje jednotlivé soubory rozložení stránky pro záhlaví, obsah a zápatí.
- *Mobile* – Zde jsou soubory, ve kterých se definují vlastnosti pro zobrazení na různých zařízeních.
- *all.less* – Jedná se o primární soubor.

Autor soubor *all.less* zkompiloval do samostatného CSS souboru, který vložil na web CSS Zen Garden. Podle struktury souborů v LESS i zkompilovaného CSS souboru stylopis působí dojmem, že žádnou z předcházejících metodik autor Jim King nepoužil.

Následující podkapitoly jsou rozdělené podle jednotlivých metodik. V každé části je demonstrováno praktické použití jednotlivých metodik na vybraném zkompilovaném stylopisu „A Robot Named Jimmy“.

Podle Michálka [15] mají CSS metodiky pro organizaci kódu bez použití preprocesorů menší dopad. Proto byl pro plné využití metodik CSS zvolen preprocesor Sass se syntaxí SCSS, který je podle Prabhu [13] nejpoužívanější preprocesor a napomáhá kodérovi kód lépe udržovat. Ukázkové kódy byly psány v editoru Visual Studio Code s použitím pluginu Live Sass Compiler sloužící pro kompilaci z SCSS do CSS souboru.

## 5.2 OOCSS

Metodika OOCSS má za cíl vytvářet komponenty, které jsou flexibilní, modulární a zaměnitelné [29]. Metodika je postavena na dodržovaní dvou principů. Prvním principem je oddělit strukturu od vzhledu a druhým principem je oddělit obsah od kontejneru.

```scss
h3 {
  color: $colorArticleHeading;
  font-family: $fontFamilyMain;
  font-size: 40px;
  font-weight: 500;
  line-height: 1.25em;
  margin: 0.5em 0;
}

p {
  font-family: $fontFamilyGeneral;
  font-size: 16px;
  color: $colorGeneral;
}
// struktura a vzhled
.participation {
  background: $colorLightBlue;
  border-top: 5px solid $colorGrayishBlue;
  padding-top: 15px;
  width: 100%;
}
```

```scss
.heading {
  color: $colorArticleHeading;
  font-family: $fontFamilyMain;
  font-size: 40px;
  font-weight: 500;
  line-height: 1.25em;
  margin: 0.5em 0;
}

.text {
  font-family: $fontFamilyGeneral;
  font-size: 16px;
  color: $colorGeneral;
}

.article {
    padding-top: 15px;
    width: 100%;
}

.theme-blue {
  background: $colorLightBlue;
  border-top: 5px solid $colorGrayishBlue;
}
```

**Kód 16: Ukázka kódu zapsaná bez použití metodiky (nalevo) a s metodikou (napravo) při dodržením principu oddělení struktury a vzhledu**

Na ukázce kódu Kód 16, po levé straně, je část kódu z původního CSS souboru upravená v preprocesoru Sass se syntaxí SCSS bez použití metodik. Na pravé straně se nachází CSS kód, který je také zapsán v preprocesoru, avšak byl pozměněn tak, aby splnil první princip metodiky OOCSS. Třída *.participation*, po levé straně ukázky, reprezentuje komponentu článku a je zapsána obvyklým způsobem, který definuje vlastnosti struktury i vzhledu článku. S použitím metodiky OOCSS, byla tato třída abstraktněji přejmenována na *.article* a rozdělena na třídu *.article* a *.theme-blue*. Tímto rozdělením došlo k oddělení struktury od vzhledu komponenty článku. Třída *.article*, původně *.participation*, působí více abstraktněji a popisuje strukturu prvku. Třída *.theme-blue* je upravena tak, aby definovala pouze vzhled prvku. Tím je docíleno splnění doporučení metodiky, kdy lze tuto třídu použít na širokou škálu prvků, jelikož není pouze součástí jednoho celku. Lze tak udržovat vlastnosti třídy *.theme-blue* pouze na jednom místě.

V následující ukázce kódu Kód 17 (viz. níže) se po levé straně nachází původní kód upravený do SCSS. Třída *.participation* představuje kontejner, do kterého jsou vnořeny prvky nadpisu *h3* a text *p*. Podle metodiky OOCSS je třeba zajistit možnost zobrazení jednotlivých prvků bez ohledu na kontejner [29]. Je tedy třeba, aby se nadpis *h3* a text *p* zobrazily bez ohledu na kontejner tedy třídu *.participation*. Zároveň by se v selektorech neměly objevovat HTML elementy [30]. Za pomoci metodiky OCSS byla vytvořena samostatná třída pro nadpis článku *.article-heading*, která se nachází po pravé straně ukázky, bez použití HTML elementu a pro text článku byla vytvořena třída *.article-text*. Lze tak tyto prvky znovu použít a zamezí se tak jejich přepisování.

```
// kontejner a obsah
.participation h3,
.participation p {
  clear: left;
  float: left;
  left: 50%;
  margin-left: -520px;
  max-width: 495px;
  position: relative;
}
```

```
.article-heading {
  clear: left;
  float: left;
  left: 50%;
  margin-left: -520px;
  max-width: 495px;
  position: relative;
}

.article-text {
  clear: left;
  float: left;
  left: 50%;
  margin-left: -520px;
  max-width: 495px;
  position: relative;
}
```

**Kód 17: Ukázka kódu zapsaná bez použití metodiky (nalevo) a s metodikou (napravo) s dodržením principu oddělení kontejneru a obsahu**

Ukázka Kód 18 demonstruje zápis v HTML bez použití metodiky OOCSS. Ze zápisu podle názvu třídy *participation* nelze vydedukovat, co je jejím účelem.

```
<div class="participation">
    <h3>Participation</h3>
    <p>Strong visual design has always been our focus. You are modifyi
    <p>You may modify the style sheet in any way you wish, but not the
    <p>Download the sample <a href="/examples/index" title="This page'
</div>
```

**Kód 18: Zápis HTML bez použití metodiky OOCSS**

Zatímco v zápisu HTML (viz. Kód 19) s použitím metodiky OOCSS a abstraktnějšího pojmenování, lze vydedukovat, že se jedná o komponentu článku a ze zápisu je zřejmé, z jakých částí je tato komponenta složena.

```
<div class="article theme-blue">
    <h3 class="heading article-heading">Participation</div>
    <p class="text article-text">Strong visual design has always been our fc
    <p class="text article-text">You may modify the style sheet in any way y
    <p class="text article-text">Download the sample <a href="/examples/inde
</div>
```

**Kód 19: Zápis HTML s použitím metodiky OOCSS**

Výhodou OOCSS je možnost kombinace tříd a jejich znovu použití na různé prvky. Tím se potvrzuje tvrzení, kdy při použití OOCSS metodiky na projektu, může nově příchozí kodér znovu použít to, co bylo abstrahováno [29]. Avšak, podle zápisu HTML (Kód 19), lze usoudit, že zvýšením počtu tříd a přidáním všech tříd pro nastylováni prvku, lze způsobit nepřehlednost a dlouhé nepřehledné zápisy v HTML kódu. Zároveň se některé oddělení struktury a vizuálního stylu zdá být zbytečné, pokud je použit pouze pro jednu komponentu.

## 5.3 BEM

Jak již bylo zmíněno v kapitole 4.2, BEM je jednoduchá konvence pojmenování pro třídy v CSS. Často se používá i v kombinaci s jinou metodikou [22].

```
.article {
    padding-top: 15px;
    width: 100%;
}

.theme-blue {
  background: $colorLightBlue;
  border-top: 5px solid $colorGrayishBlue;
}

.article-heading {
  clear: left;
  float: left;
  left: 50%;
  margin-left: -520px;
  max-width: 495px;
  position: relative;
}

.article-text {
  clear: left;
  float: left;
  left: 50%;
  margin-left: -520px;
  max-width: 495px;
  position: relative;
}
```

```
.article {
    padding-top: 15px;
    width: 100%;
}

.article_theme-blue {
  background: $colorLightBlue;
  border-top: 5px solid $colorGrayishBlue;
}

.article__heading {
  clear: left;
  float: left;
  left: 50%;
  margin-left: -520px;
  max-width: 495px;
  position: relative;
}

.article__text {
  clear: left;
  float: left;
  left: 50%;
  margin-left: -520px;
  max-width: 495px;
  position: relative;
}
```

**Kód 20: Ukázka kódu bez použití metodiky BEM (nalevo) a s použitím metodiky BEM (napravo)**

Na ukázce kódu Kód 20 je na levé straně použitý kód podle metodiky OOCSS z předešlé ukázky Kód 16 a na pravé straně je kód, na který byla aplikována konvence pojmenování BEM. Na levé straně je třída *.article*, která je funkčně

nezávislá součást stránky. Třídu lze znovu použít a nemá nastavenou žádnou pozici. Podle metodiky BEM se jedná o blok. Podle konvence pojmenování BEM by tento blok měl být pojmenován tak, aby popisoval svůj účel, tedy účel např. článku. Na ukázce kódu Kód 16 z kapitoly 5.2 po levé straně by byl název tohoto prvku *.participation* podle BEM nevhodný, jelikož název nepopisuje účel tohoto prvku. Vhodným pojmenováním je proto *.article*, u kterého je zřejmý jeho účel, tedy že se jedná o článek.

Na ukázce Kód 20 (viz. výše), po levé straně, jsou třídy pod názvem *.article-heading* a *.article-text* sémanticky vázané k bloku *.article*. Jedná se o součásti tohoto bloku. Zároveň třídy nedefinují vzhled, stav ani chování bloku, proto se jedná o entity elementu podle metodiky BEM. V tomto případě, podle konvence BEM, je třeba upravit strukturu názvu (viz. pravá strana ukázky kódu Kód 20). Je zde zobrazeno, jak vypadá kód SCSS po úpravě názvů na základě konvence pojmenování BEM. Pomlčka byla nahrazena dvěma podtržítky, jež oddělily název elementu od názvu bloku.

Třída *.theme-blue* popisuje a mění vzhled bloku *.article*. Dle metodiky se jedná o modifikátor. Na základě konvence pojmenování BEM by měla být struktura názvu zapsána takto *blok_modifikátor*, tedy ve výsledku *.article_theme-blue*. Pomocí jednoho podtržítka je oddělen název bloku a název modifikátoru.

```
<div class="article article_theme-blue">
    <h3 class="heading article__heading">Participation</h3>
    <p class="text article__text">Strong visual design has a
    <p class="text article__text">You may modify the style sl
    <p class="text article__text">Download the sample <a hre
</div>
```

**Kód 21: Ukázka zápisu HTML při použití metodiky BEM**

V ukázce Kód 21 stejně jako u OOCSS (viz. Kód 19) dochází k zápisu několika tříd. V případě použití konvence BEM je zápis i delší.

## 5.4 SMACSS

Metodika SMACSS je oproti OOCSS detailněji popsána. S metodikou SMACSS byly jednotlivé částí původního CSS souboru, s použitím preprocesoru Sass se syntaxí SCSS, rozděleny do následujících samostatných souborů:

- _settings.scss
- _mixins.scss
- _base.scss
- layout
    - _content.scss
    - _intro.scss
    - _sidebar.scss
- modules
    - _article.scss
    - _footer.scss
    - _header.scss
    - _list.scss
    - _summary.scss
    - _wrapper.scss
- _states.scss
- main.scss

Snook [23] doporučuje použití samostatných souborů pro proměnné (*settings*) a mixiny (*mixins*) v případě využití některého z preprocesoru. Jelikož je použit preprocesor Sass, byly tyto soubory vytvořeny. Jak bylo už zmíněno v kapitole 2, výhodou použití preprocesorů je možnost použití proměnných, které v případě změny stačí upravit pouze na jednom místě.

První vrstvou metodiky SMACSS je základní (*base*) vrstva. V této vrstvě se nachází všechna základní pravidla, která jsou uložena do jednoho samostatného souboru. V tomto souboru se nachází všechna stylování HTML elementů tedy: *body*, *html*, *img*, *li* atd. A neobjevují se tu žádné selektory tříd (viz. Kód 22).

```
html {
  font-family:sans-serif;
  -ms-text-size-adjust:100%;
  -webkit-text-size-adjust:100%;
}
body{
  margin:0;
  color: $colorGeneral;
  font-family: $fontFamilyGeneral;
  font-size: $fontSize6;
  line-height: 1.25em;
}
```

**Kód 22: Ukázka části kódu ze souboru *_base.scss***

Po vrstvě základní (*base*) následuje vrstva rozložení (*layout*). Podle typu rozložení a doporučení metodiky byly jednotlivé části rozložení stránky rozděleny do samostatných souborů pod složkou *layout*. V souboru *_intro.scss* se nachází identifikátor pro záhlaví. V souboru *_content.scss* je identifikátor pro formátování obsahu na webové stránce. V souboru *_sidebar.scss* se nachází stylování zápatí. V této vrstvě Snook [23] jednotlivé styly dělí na hlavní a vedlejší. Styly jako je záhlaví, zápatí a v tomto případě i obsah jsou hlavní styly stylizované pomocí identifikátorů ID (viz. ukázka Kód 23). Třídy a prefix *l-* se používají u vedlejších stylů. Příkladem vedlejšího stylu v ukázce Kód 23 (viz. níže) je třída *.l-leftintro,* která definuje, že záhlaví bude posunuto k levému okraji. Tuto třídu lze znovu použít, kdežto element *#intro* definovaný identifikátorem lze použít pouze jednou. Důvodem je dle metodiky SMACSS, že není třeba, aby se rozložení na webové stránce opakovalo, a proto se formátuje identifikátorem. Metodika SMACSS doporučuje pro lepší orientaci v HTML zápisu a CSS kódu použít prefixy. Prefixem tříd, které se nachází v této vrstvě je prefix *l-*.

```
#intro {
  @include gradient ($colorWhite, $colorLightBlue);
  height: 590px;
  width: 100%;
}

.l-leftintro {
  float: left;
}
```

**Kód 23: Ukázka stylování záhlaví ve vrstvě layout**

V další vrstvě jsou moduly (*modules*) reprezentovány složkou *modules*, kde má každý modul svůj vlastní soubor. V souborech jednotlivých modulů se nachází pouze třídy, aby bylo možné jednotlivé moduly znovu použít. Například modul patičky s odkazy (viz. Kód 24) nebo modul článku, nadpisu atd.

```
.footer {
  font-size: $fontSize5;
  background: $colorFooterItem;
  height: 65px;
  text-align: center;
  width: 100%;

  .footer-link {
    margin-right: 25px;
    vertical-align: -1em;

    &:last-child {
      margin-right: 0;
    }

  }

}
```

**Kód 24: Ukázka modulu patičky s odkazy v SCSS**

Na první pohled je stylování modulu oproti objektům u OOCSS méně abstraktní. Stejně jako v OOCSS, tak i v SMACSS je doporučeno používat názvy tříd namísto prvků HTML. Namísto použití HTML prvku „*a*“ je vytvořena třída s názvem *.footer-link*. Podle Snooka [23], se tímto krokem odstraní nejednoznačnost stylingu prvků a sníží se závislost na konkrétním HTML elementu. To znamená, že restrukturalizace

HTML bude mít minimální dopad na CSS [23]. Je zde i vidět výhoda SCSS, která umožňuje zanořování jednotlivých komponent.

Pro vrstvu stavů (*state*) byl vytvořen samostatný soubor *_states.scss* na základě doporučení metodiky SMACSS. V této vrstvě se nachází stylovací pravidla, která přepisují všechny ostatní předešlé styly. Do této vrstvy byla vložena „*@media*“ pravidla, která slouží k přepisu pravidel podle šířky okna zařízení, na kterém se má webová stránka zobrazit (viz. Kód 25).

```
/**
 * Zobrazení na různých zařízeních.
 */

@media (max-width: 1040px) {···
 }

 @media (max-width: 750px) {···
 }

 @media (max-width: 650px) {···
 }

 @media (max-width: 500px) {···
 }
```

**Kód 25: *@media* pravidla**

Vrstva téma (*theme*) je vrstva, která se nemusí vždy použít. Většinou se používá u větších projektů a definuje vzhled stránky nebo přepisuje stylovací pravidla podobně jako vrstva stavů [23].

Ukázka kódu Kód 26 demonstruje použití vrstvy téma (*theme*) na modul patičky z ukázky kódu Kód 24. Při importu souboru *_theme1.scss* do primárního souboru SCSS bude přepsán styl pozadí patičky na červenou barvu a při importu a použití souboru *_theme2.scss* na zelenou.

```scss
// soubor _theme1.scss
.footer {
    background: $colorBrightRed;
}

// soubor _theme2.scss

.footer {
    background: $colorLimeGreen;
}
```

**Kód 26: Příklad téma (*theme*) vrstvy pro přepis pozadí patičky**

Nakonec byl vytvořen primární soubor SCSS, který obsahuje všechny soubory použité pro formátování webové stránky, a zahrnuje je do jednoho stylu. Jako jediný má primární soubor SCSS název bez podtržítka, aby kompilátor poznal, že má tento soubor zkompilovat. Jednotlivé soubory s podtržítkem nejsou zkompilovány do jednotlivých CSS souboru.

```scss
@import "settings";
@import "mixins";

@import "base";

@import "layout/intro";
@import "layout/content";
@import "layout/sidebar";

@import "modules/header";
@import "modules/summary";
@import "modules/article";
@import "modules/footer";
@import "modules/wrapper";
@import "modules/list";

@import "states";
```

**Kód 27: Primární soubor SCSS**

Na ukázce části výstupu HTML Kód 28, díky identifikátoru a použití prefixu *l-*, je zřejmé, že se jedná o část, která styluje část rozvržení stránky v tomto případě záhlaví. Další třídy bez prefixů jsou jednotlivé moduly např. modul hlavičky (*header*), který je jednou z částí záhlaví a obsahuje hlavní nadpis (*main-heading*) a podnadpis (*subtitle-heading*).

```
<div id="intro" class="l-leftintro">
    <div class="header">
        <h1 class="main-heading">CSS Zen Garden</h1>
        <h2 class="subtitle-heading">The Beauty of <abbr title="Cascading Styl
    </div>
    <summary class="summary">
        <p class="summary-text">A demonstration of what can be accomplished th
        <p class="summary-text">Download the example <a href="/examples/index"
    </summary>
</div>
<div id="supporting" class="l-leftsupporting">
```

**Kód 28: Zápis HTML při použití metodiky SMACSS**

Výhodou metodiky SMACSS je logické členění souborů a zavedení prefixů. Oproti metodice OOCSS je mnohem snadnější upravovat CSS kód, který je díky kategorizaci mnohem přehlednější a lze snadno vydedukovat, kde se část kódu, kterou je potřeba upravit, nachází. Kód SCSS je s použitím metodiky SMACSS srozumitelný a škálovatelný.

## 5.5 ITCSS

Podobně jako metodika SMACSS se i ITCSS zaměřuje na dělení a organizaci CSS kódu.

První dvě vrstvy trojúhelníku ITCSS nastavení (*settings*) a nástroje (*tools*) jsou věnovány preprocesorům. Tyto dvě vrstvy jsou stejné jako v metodice SMACSS. První vrstva je samostatný soubor *_settings.scss*, který obsahuje proměnné barev, velikosti fontů atd. Druhá vrstva, prezentována složkou nástroje (*tools)*, obsahuje mixiny. Obě tyto vrstvy nejsou vygenerovány do výsledného CSS kódu.

Následující složka *generic* reprezentuje třetí vrstvu a obsahuje soubor *_normalize.scss*, ve kterém jsou nastavené výchozí vlastnosti HTML elementů (viz. Kód 29). Tento soubor je první, který generuje CSS kód ve výsledném souboru. Jak popisuje Roberts [26], tato vrstva bývá stejná pro všechny projekty.

```
aside,footer,header {
  display:block;
}

html {
  font-family:sans-serif;
  -ms-text-size-adjust:100%;
  -webkit-text-size-adjust:100%;
}
body{
  margin:0;
}

a:focus{
  outline:thin dotted;
}

a:active,a:hover{
  outline:0;
}
```

**Kód 29: Příklad nastavení výchozích hodnot HTML elementů**

Ve vrstvě elementů (*elements*) se nachází několik souborů. Každý soubor styluje jednotlivé HTML elementy, které je třeba předefinovat. Na ukázce Kód 30 je zobrazena část kódu ze tří souborů tedy tří elementů: *body*, *footer*, *h1*. V této vrstvě se většinou HTML elementům nedefinují vlastnosti jako barva. Definuje se velikost nebo pozice elementů.

```scss
// soubor _body.scss
body {
  color: $colorGeneral;
  font-family: $fontFamilyGeneral;
  font-size: $fontSize6;
  line-height: 1.25em;
}

// soubor _footer.scss
footer {
  font-size: $fontSize5;
  height: 65px;
  text-align: center;
  width: 100%;
}

// soubor _headings.scss
h1 {
  font-size: $fontSize1;
  font-weight: $fontWeight1;
}
```

**Kód 30: Příklad jednotlivých elementů v ITCSS**

U následujících vrstev objekty (*objects*) a komponenty (*components*) je zpočátku obtížné oddělit a kategorizovat části SCSS kódu. Během kategorizace SCSS kódu je důležité přemýšlet o objektech jako o elementech, které řídí rozložení, a o komponentách jako o samostatných dílech, které se mohou použít kdekoliv a jejich rozložení určuje objekt [31].

```scss
// soubor _intro.scss
.o-intro {
  @include gradient ($colorWite, $colorLightBlue);
  float: left;
  height: 590px;
  width: 100%;
}
```

**Kód 31: Příklad objetu *.o-intro* ITCSS**

Na ukázce kódu Kód 31 třída *.o-intro* představuje objekt záhlaví webové stránky. V názvu třídy se používá podle metodiky prefix *o-* pro označení objektu.

Třída *.c-summary* (viz. Kód 32) představuje komponentu pro stručný popis. Ve svém názvu pro označení komponenty podle metodiky obsahuje prefix *c-*.

```scss
// soubor _summary.scss
.c-summary {

  .c-summary__text {
    clear: right;
    color: $colorBrightRed;
    float: right;
    font-style: italic;
    margin: 0;
    width: 300px;

    .c-summary__link {
      color: $colorBrightBlue;
    }

  }

}
```

**Kód 32: Ukázka kódu SCSS komponenty *.c-summary***

Zápis HTML Kód 33 je názorný příklad použití objektu *.o-intro* a komponenty *c-summary*, kde objekt určuje rozložení komponent *c-header* a *c-summary*. Díky prefixům je zápis lépe organizovaný. Kodér snadno rozpozná, o jakou vrstvu SCSS souboru se jedná a ušetří tak čas hledáním v kódu. Pokud se tento zápis porovná se zápisem Kód 21, tak lze konstatovat, že bez použití prefixu je pojmenování podobné jako u metodiky BEM. Je to z toho důvodu, že se metodika ITCSS často používá v kombinaci s konvencí pojmenování BEM.

```html
<body>
    <div class="o-intro">
        <div class="c-header">
            <h1 class="c-heading__main">CSS Zen Garden</h1>
            <h2 class="c-heading__subtitle">The Beauty of <abbr title="Cascading Style Sheets">CSS</abbr> Design</h2>
        </div>
        <summary class="c-summary">
            <p class="c-summary__text">A demonstration of what can be accomplished through <abbr title="Cascading Style
            <p class="c-summary__text">Download the example <a href="/examples/index" class="c-summary__link" title="Thi
        </summary>
    </div>
```

**Kód 33: Zápis HTML při použití metodiky ITCSS**

V poslední vrstvě utility (*utilities*) se nacházejí soubory *_media.scss* a *_links.scss*, které přepisují vlastnosti definované v předchozích vrstvách. V souboru *_links.scss* je třída *.u-link__designer-name*, která přepisuje HTML element odkazu *a*. Tato vrstva je podobná vrstvě stavů z předchozí metodiky SMACSS, která také přepisuje předchozí vlastnosti komponent.

## 5.6 Atomický web design

Metodika Atomického web designu (dále AWD) je užitečná pro návrh a vývoj webových aplikací. Avšak v zásadě jde o model konstrukce uživatelského rozhraní. Atomový design umožňuje vidět uživatelská rozhraní rozdělená na atomové prvky a současně procházet, jak se tyto prvky spojují a tvoří konečné uživatelské rozhraní. Metodika AWD poskytuje možnost pro diskusi o modularitě mezi kolegy a potřebný smysl pro hierarchii v systému designu. Metodika se zabývá tvorbou designových systémů UI bez ohledu na použitou technologii k vytvoření UI. Metodika je zaměřená na webové designery tak, aby tvořili promyšlené *„style guidy"* pro rozhraní systému designu. [8]

Na základě navrženého *„style guidu"* webovým designerem, za použití metodiky AWD, musí následně kodér napsat CSS kód pro jednotlivé komponenty. Lze tak usoudit, že myšlenku metodiky AWD pro tvorbu *„style guidu*" může aplikovat kodér i jako metodiku pro organizaci CSS kódu.

Při aplikování metodiky na vybraný CSS soubor byly jednotlivé části kódu CSS rozděleny do složek podle kategorií atomy (*atoms*), molekuly (*molecules*), organismy (*organisms*) a šablona (*template*).

Jak již bylo zmíněno v kapitole 4.5, první vrstvu tvoří atomy, které slouží jako základní stavební kameny rozhraní. Nelze je dále rozdělit. Patří sem HTML elementy, a i abstraktnější prvky jako barevné palety, písmo nebo i animace. [8]

Příklad obsahu adresáře *atoms*:

- _variables.scss
- _animation.scss
- _headings.scss
- _links.scss
- a další.

Soubor *_variables.scss* je identický se souborem *_settings.scss* , který byl použit v předešlých metodikách (SMACSS, ITCSS). V souboru _variables.scss jsou definovány proměnné barev, fontů atd. V souboru *_headings.scss* jsou předefinovány vlastnosti jednotlivých nadpisů, které jsou zobrazeny v ukázce Kód 34 (viz. níže).

Metodika AWD nedefinuje žádnou konvenci pojmenování, ale pro lepší čitelnost v kódu SCSS byl pro pojmenování jednotlivých prvků použit prefix z názvu kategorie, ve které se prvek nachází.

```scss
//Atomy
.atom-heading-main {
  font-family: $atomFontFamilyMain;
  font-size: $atomFontSize1;
  font-weight: $atomFontWeight1;
  color: $atomColorMainHeading;
}

.atom-heading-subtitle {
  color: $atomColorBrightRed;
  font-family: $atomFontFamilyGeneral;
  font-size: $atomFontSize5;
  font-style: $atomFontStyle1;
  font-weight: $atomFontWeight2;
}
```

**Kód 34: Ukázka zápisu SCSS atomů nadpisu**

Další kategorií jsou molekuly (*molecules*). Z hlediska uživatelského rozhraní a webového designu je molekula skupinou prvků nebo atomů uživatelského rozhraní, které pracují společně. Tím získává skupina atomů svůj účel a funkčnost. [32]

Ukázka Kód 35 demonstruje spojení atomů nadpisů *.atom-heading-main* a *.atom-heading-subtitle*, kdy vznikne molekula s názvem třídy *.molecule-header* (hlavička) a to pomocí hnízdění.

```scss
//molekula hlavičky
.molecule-header {
  margin: 0 auto;
  max-width: 1040px;
  position: relative;
  text-align: center;
  top: 0;
  width: 100%;
  z-index: 1;

  .atom-heading-main {
    border-top: 4px solid $atomColorWite;
    display: inline-block;
    min-height: 70px;
    margin: 55px auto 0;
    padding: 0 10px;
    z-index: 2;

    &:after {
      background-color: $atomColorBrightRed;
      content: '';
      height: 2px;
      left: 0;
      margin-top: -3px;
      position: absolute;
      width: 100%;
      z-index: -1;
    }

  }

  .atom-heading-subtitle {
     text-align: right;
     max-width: 695px;
     margin: -10px auto 0;
     width: 100%;
   }
}
```

**Kód 35: Ukázka kódu molekuly hlavičky**

Následují obrázek Obrázek 7 je vizuální zobrazení jednotlivých atomů, ze kterých je složená molekula *.molecule-header*.


**Obrázek 7: Vizuální představení vytvořené molekuly**

Následně jsou vypsány některé soubory jednotlivých molekul, které vznikly při použití metodiky na původní CSS soubor:

- _article.scss – molekula článku tvořená složením atomu nadpisu a textu,
- _footer.scss – molekula patičky tvořená atomy odkazů,
- _header.scss – molekula hlavičky tvořená atomy nadpisů,
- _summary.scss – molekula stručného popisu tvořená atomy textu a odkazů,
- a další.

Tyto soubory se nachází v adresáři *molecules*. Lze si povšimnout, že soubory atomů a molekul připomínají stromovou strukturu dokumentu a kaskády, avšak v opačném definování vlastností. Ve stromové struktuře se postupuje od větších bloků po elementy. V této metodice AWD se postupuje od malých komponent po velké celky.

Třetí vrstvou atomového designu jsou organismy (*organisms*). Organismy lze vytvořit kombinací dvou nebo více molekul či molekulou s atomy. Jedná se o relativně složité komponenty uživatelského rozhraní, které tvoří jednotlivé sekce rozhraní. Několik příkladů organismů zahrnuje běžné součásti uživatelského rozhraní, jako je záhlaví, zápatí nebo část pro obsah. Na ukázce Kód 36 je příklad organismu záhlaví s názvem třídy *.organism-intro*.

```scss
//organism záhlaví
.organism-intro {
  @include gradient ($atomColorWite, $atomColorLightBlue);
  height: 590px;
  width: 100%;
}
```

**Kód 36: Ukázka organismu záhlaví**

Další vrstvou jsou šablony (*template*). Zde je definováno rozložení jednotlivých organismů na stránce (viz. Kód 37).

```scss
.template-main-intro {
  float: left;
}

.template-main-content {
  float: left;
  position: relative;
  z-index: 3;
}

.template-main-aside {
  clear: both;
  display: block;
  float: left;
  margin-bottom: -5px;
}
```

**Kód 37: Rozložení organismu**

Z následujícího zápisu HTML Kód 38 je patrné, že záhlaví pojmenované třídou *.organism-intro* je tvořené molekulou hlavičky (*.molecule-header*) a molekulou stručného popisku (*.molecule-summary*). Lze si také všimnout použití předposlední kategorie šablony (*template*), ve které je definováno rozložení stránky. Spojení vrstev organismů a šablon AWD odpovídá vrstva rozložení (*layout*) v metodice SMACSS.

```
<section class="template-main-intro organism-intro">
    <header class="molecule-header">
        <h1 class="atom-heading-main">CSS Zen Garden</h1>
        <h2 class="atom-heading-subtitle">The Beauty of <abbr title="Cascading Style Sheets">C
    </header>
    <div class="molecule-summary">
        <p class="atom-text">A demonstration of what can be accomplished through <abbr title="
        <p class="atom-text">Download the example <a href="/examples/index" class="atom-link"
    </div>
</section>
<div class="template-main-content organism-content">
```

**Kód 38: Příklad HTML zápisu v metodice Atomického webu**

Poslední vrstva stránka (*page*) je reprezentována samostatným souborem *main-page.scss*, kde jsou naimportovány jednotlivé části vrstev AWD podobně jako u primárního souboru Sass v metodice SMACSS (viz. Kód 27).

## 5.7 Výhody a nevýhody

V této podkapitole jsou popsané výhody a nevýhody metodik OOCSS, BEM, SMACSS, ITCSS a AWD.

Následující tabulka Tabulka 1 popisuje výhody a nevýhody metodiky OOCSS, které vyplynuly z praktického použití metodik. Jednou z hlavních výhod metodiky OOCSS je rozdělení kódu SCSS na objekty, které jsou znovupoužitelné. Díky rozdělení, lze v SCSS kódu lépe hledat jednotlivé objekty což zvyšuje čitelnost kódu. Avšak při vytváření samostatných tříd pro většinu vlastností, které definují vzhled, může dojít k znehodnocení čitelného SCSS kódu, který se tak stane nečitelným. Následné množství zapsaných vlastností do atributu *class* způsobuje nepřehlednost i v HTML zápisu. Proto je důležité se zamýšlet nad vytvářením jednotlivých objektů.

**Tabulka 1: Výhody a nevýhody metodiky OOCSS**

| Výhody | Nevýhody |
|---|---|
| Rozdělení SCSS na objekty | Zbytečné oddělení vzhledu od struktury v některých komponentách |
| Čitelnost v SCSS souborech | Čitelnost v HTML zápisu |
| Znovupoužitelnost objektů | Velké množství tříd |
| Oddělení obsahu od kontejneru | Nedefinuje pojmenování |

Hlavní výhodou metodiky BEM je definice užitečné konvence pojmenování (viz Tabulka 2). Vhodné pojmenování přispívá k lepší čitelnosti v SCSS kódu a udržuje

tak čisté prostředí. Může se však stát, že kvůli delším názvům některých prvků, dojde k zhoršení čitelnosti v HTML zápisu.

**Tabulka 2: Výhody a nevýhody metodiky BEM**

| Výhody | Nevýhody |
|---|---|
| Rozdělení SCSS kódu na bloky, elementy a modifikátory | Dlouhé názvy prvků |
| Čitelnost v SCSS souborech | Čitelnost v HTML zápisu |
| Užitečná konvence pojmenování | |
| Znovupoužitelnost bloků | |

Následující tabulka Tabulka 3 shrnuje výhody a nevýhody SMACSS a ITCSS. Zde převažují výhody nad nevýhodami použití metodik. Metodiky SMACSS a ITCSS rozdělují SCSS kód do více vrstev. Díky prefixům, a nepřehlcenému množství zapsaných vlastností do atributu *class*, je zajištěna dobrá čitelnost v HTML zápisu i v SCSS kódu. Obě metodiky předpokládají možné použití preprocesorů. Nevýhodou těchto metodik je jejich částečné zaměření na pojmenování prvků, a to pouze pomocí prefixů.

**Tabulka 3: Výhody a nevýhody metodiky SMACSS a ITCSS**

| Výhody | Nevýhody |
|---|---|
| Rozdělení SCSS kódu do více vrstev | Zaměřuje se částečně na pojmenování |
| Čitelnost v SCSS souborech | Hůře zapamatovatelné rozdělení vrstev |
| Čitelnost v HTML zápisu | |
| Snadné vyhledání části kódu | |
| Znovupoužitelnost modulů | |
| Použití prefixů | |
| Počítá s použitím preprocesorů | |

Výhodou metodiky Atomického web designu (AWD) (viz. Tabulka 4) je snadno zapamatovatelné rozdělní na atomy, molekuly a organismy, kdy lze definice jednotlivých vrstev snadno odvodit. Nevýhodou metodiky je, že nedefinuje pojmenování prvků, která by podpořila lepší čitelnost v kódu SCSS a HTML. Kodér si tak musí vymyslet svojí konvenci pojmenování nebo použít konvenci pojmenování BEM.

**Tabulka 4: Výhody a nevýhody metodiky AWD**

| Výhody | Nevýhody |
|---|---|
| Rozdělení SCSS kódu na atomy, molekuly a organismy | Nedefinuje prefixy ani pojmenování |
| Snadno zapamatovatelné rozdělení | Soubory pro preprocesory součástí atomů |
| Znovupoužitelnost prvků | |
| Pojmenování vrstev/kategorií | |
| Škálovatelnost | |
| Udržitelnost | |
| Čitelnost v SCSS souborech | |

## 5.8 Vyhodnocení

Při vývoji webových aplikací v týmu je podstatné, aby byl CSS kód srozumitelný a snadno udržovatelný. Podle doporučení z kapitoly 3 je třeba k zajištění těchto kritérií, aby byl CSS kód rozdělen na jednotlivé části do samostatných souborů. Délka kódu CSS v jednotlivých souborech by neměla být vyšší než 200 řádků. Části kódu CSS by měly být zřetězeny do jednoho primárního souboru.

K udržitelnosti a přehlednosti velkého množství CSS kódu je třeba udržovat jeho logickou strukturu souborů CSS kódu. Struktura rozděluje části kódu do jednotlivých vrstev. Pomocí správné struktury ví kodér, kde jednotlivé části kódu CSS hledat. Struktura by zároveň měla být snadno pochopitelná. Vhodná struktura přispívá k snadnější údržbě CSS kódu a zvyšuje efektivnost psaní kódu.

Dále k lepší čitelnosti kódu CSS přispívá vhodně zvolené pojmenování jednotlivých komponent. Pojmenování by mělo být takové, aby bylo zřejmé, co je to za komponentu a jaká je její funkčnost. Kodér, který vidí kód poprvé, je díky vhodně zvolenému pojmenování schopný rychle porozumět CSS kódu.

Znovupoužitelnosti komponent se docílí pomocí použití tříd namísto identifikátorů a HTML elementů. Komponenta by neměla být závislá na umístění. Měly být otevřené pro rozšíření, ale uzavřená pro úpravy. Název komponenty by měl být obecný, ale jednoznačný.

Následující tabulka Tabulka 5 hodnotí metodiky OOCSS, BEM, SMACSS, ITCSS a AWD na základě výše zmíněných kritérii. Tedy, zda metodika podporuje logickou strukturu (vrstvy) pro rozdělených jednotlivých souborů CSS. Zda definuje jmennou konvenci pro komponenty. Dále, jestli klade důraz na znovupoužitelnost komponent a jestli se zaměřuje na škálovatelnost prvků.

**Tabulka 5: Vyhodnocení metodik**

| Metodika | Logická struktura (vrstvy) | Jmenná konvence | Znovupoužitelnost | Škálovatelnost |
|---|---|---|---|---|
| **OOCSS** | nepodporuje | nedefinuje | dává velký důraz | zaměřená |
| **BEM** | nepodporuje | definuje | dává menší důraz | zaměřená |
| **SMACSS** | podporuje | částečně definuje | dává menší důraz | zaměřená |
| **ITCSS** | podporuje | částečně definuje | dává menší důraz | zaměřená |
| **AWD** | podporuje | nedefinuje | dává menší důraz | zaměřená |

Metodika OOCSS nepopisuje žádnou logickou strukturu (vrstvy). Jejím cílem je na základě dvou principů vytvořit CSS objekty, které jsou znovupoužitelné. Doporučuje používat názvy tříd místo HTML prvků, ale dále doporučené zásady pro pojmenování prvků nerozvádí.

Metodika BEM rozděluje entity na bloky, elementy a modifikátory. Lze pomocí ní rozdělit SCSS kód do souborů pro jednotlivé bloky, ve kterých se nachází elementy a modifikátory, které s blokem souvisí. Metodika BEM tak dále nerozvádí logickou strukturu pro rozdělení adresářů do více vrstev. Popisuje však užitečnou konvenci pojmenování prvků. Pomocí této konvence lze z názvu určit, co je to za prvek, jaký má účel a zda je součástí nějakého dalšího prvku.

Metodika SMACSS definuje více podrobnou logickou strukturu, která se skládá z 5 vrstev: *base*, *layout*, *module*, *state* a *theme*. Částečně podporuje pojmenování, a to za pomocí prefixu *l-* a identifikátoru. Ty slouží k tomu, aby bylo zřejmé, že se jedná o prvky pro rozložení stránky. Dále se pro označení prvků ve vrstvě *state* doporučuje použít prefix *is-*. Co se týče znovupoužitelnosti komponent, metodika SMACSS

definuje vrstvu *modules*, ve které se nachází moduly, které lze opakovaně použít. Avšak metodika SMACSS dává menší důraz na znovupoužitelnost než metodika OOCSS. Pro metodiku OOCSS je znovupoužitelnost komponent primárním cílem, kterým se zabývá.

Podobně jako metodika SMACSS tak i metodika ITCSS definuje velmi podrobnou logickou strukturu. ITCSS řadí prvky do jednotlivých vrstev na základě jejich účelu. Částečně používá pojmenování prvků pomocí prefixů. Pomocí prefixu *o-* se označuje objekt a prefixem *c-* komponenta. ITCSS definuje komponenty, které lze znovupoužít.

Metodika AWD používá vrstvy rozdělené na atomy, molekuly, organismy, šablony a stránky. Prvky atomů, molekul a organismů jsou znovupoužitelné. Metodika nedefinuje žádné speciální pojmenování prvků, ale pro lepší orientaci v kódu lze přidat k názvům třídy prvků prefix vrstvy, které náleží.

# 6 Shrnutí výsledků

Při neorganizovaném psaní kaskádových stylů se po čase stává kód CSS nepřehledným, což vede k častému přepisování vlastností a hromadění dalšího nadbytečného CSS kódu. Z tohoto důvodu vznikají metodiky, které mají kodérovi pomoci lépe vytvářet a organizovat CSS kód. Představené metodiky byly následně prakticky aplikovány na vzorovém stylopisu „A Robot Named Jimmy".

První byla použita metodika OOCSS, pomocí které byl CSS kód rozdělen do nejmenších objektů. Jednotlivé objekty jsou stylovány za použití tříd. Samostatná třída je vytvořená pro definování barvy pozadí nebo ohraničení a jednotlivých komponent například odstavce nebo nadpisu. Vzhled je definován v zápisu HTML v atributu *class*, ve kterém je několik tříd, které definují všechny vlastnosti elementu. Konečné množství zapsaných vlastností do atributu je dlouhé a může způsobit nepřehlednost v HTML zápisu.

Vhodné použití metodiky OOCSS je například pro vytvoření knihovny základních abstraktních elementů jako je tlačítko, formulář nebo například tabulka, které lze tak znovu použít i následně na jiných projektech. Avšak vytváření samostatných tříd pouze pro barvu pozadí lze označit za nadbytečné. Metodika OOCSS nabízí mnoho užitečných doporučení, avšak je třeba vážit jejich aplikaci pro některé situace při psaní a organizaci kódu.

Součástí metodiky BEM je použití užitečné konvence pro vytváření názvů CSS prvků. Z názvu je kodérovi hned jasné, co je to za prvek, jaký má účel a zda je součástí určitého prvku. Například pokud vidí kodér v zápisu HTML název třídy *article,* tak ví, že se jedná o komponentu článku nebo v případě *article-text*, je mu zřejmé, že se jedná o text, který je součástí článku. Oproti OOCSS, kde se doporučuje použít modifikátor samostatně, metodika BEM neumožňuje jeho samostatné použití. Modifikátor je dle konvence BEM vždy připojen k elementu nebo bloku. To je výhodné řešení, jelikož kodér nemusí vytvářet mnoho zbytečných modifikátorů a vytvoří pouze ty, které jsou užitečné a nevytváří tak další CSS kód navíc.

Další metodiky SMACSS, ITCSS a Atomický web design už více zabývají strukturou rozdělení a celkovou organizaci CSS kódu do složek a samostatných souborů. Tyto metodiky jsou oproti předešlým dvěma propracovanější a rozdělují

prvky do více vrstev. Zároveň počítají s možností použití CSS preprocesorů, které mají sloužit kodérovi k snadnější údržbě CSS kódu. V Atomickém designu jsou preprocesory již součástí vrstvy atomů. Autor SMACSS navrhuje použití pěti vrstev, ovšem ty nepředpokládají použití CSS preprocesorů. V případě jejich použití doporučuje vytvořit samostatné soubory pro proměnné a pro mixiny a vytvořit tak další vrstvu, která předchází ostatním vrstvám. ITCSS metodika oproti ostatním metodikám klade důraz na použití CSS preprocesorů, kterým jsou věnovány první dvě vrstvy trojúhelníku ITCSS. Jelikož není povinné využít preprocesory při psaní CSS kódu, je výhodné nezahrnovat preprocesory do základních vrstev dané architektury, ale udělit jim vlastní prostor, tak jak to navrhují metodiky ITCSS a SMACSS.

# 7 Závěry a doporučení

Webové stránky dnes slouží jako základní prostředek pro prezentaci služeb a produktů. Za jejich návrhem a vytvořením dnes stojí již celý tým lidí, z nichž každý má svojí roli. Diplomová práce je zaměřená na roli kodéra, který vytváří designovou kostru webové aplikace. Dále je mapován přístup k psaní CSS kódu, který slouží jako základní nástroj pro vytvoření vzhledu stránek.

V první části byl nastíněn průběh navrhování webu a popsán rozdíl mezi rolemi, které spadají pod pojem webdesign. Tj. webového designera, jenž má za úkol navrhnout, jak web bude fungovat, jak se budou návštěvníci na webu pohybovat a navrhnout náhled webu. A dále webového kodéra, který na základě vytvořeného náhledu za pomocí HTML a CSS vytvoří uživatelské rozhraní. V průběhu procesu vytváření návrhu UI systému je klíčová vzájemná komunikace mezi těmito rolemi, jedině tak lze dosáhnout funkčního a přehledného rozhraní.

Následně diplomová práce zkoumala možnosti, které vedou ke zlepšení srozumitelnosti a čitelnosti CSS kódu v průběhu jeho vytváření. Zkoumala úsporné vlastnosti preprocesorů CSS, které rozšiřují kaskádové styly o prvky známe z programovacích jazyků jako je přístup k proměnným. Dále zkoumala doporučené zásady, které mění způsob, jakým se CSS kód vytváří a strukturuje. Avšak bylo zjištěno, že bez použití metodik CSS samotné preprocesory a doporučené zásady nemusí vést ke zlepšení čitelnosti kódu CSS a práci s ním obecně.

V diplomové práci byly dále detailně popsány vybrané metodiky CSS: OOCSS, BEM, SMACSS, ITCSS a Atomický web design. Metodika OOCSS je zaměřena na vytváření abstraktních CSS objektů, které jsou stylovány dalšími třídami. Metodika BEM představuje komponentní přístup k vývoji webu a zároveň popisuje konvenci pojmenování CSS komponent. SMACSS metodika je mnohem propracovanější a její použití je dobře zdokumentováno v odborné literatuře. Cílem metodiky je umožnit lépe kategorizovat styly, tak aby se kodér musel zamýšlet nad tím co, jak a proč bude tvořit pomocí CSS kódu. Zároveň metodika slouží k redukci opakujících se kaskádových stylů. Metodika ITCSS využívá přirozených priorit CSS a trojúhelníku ITCSS, kdy nejširší část trojúhelníku představuje nejméně konkrétní pravidla, zatímco nejkonkrétnější pravidla jsou na špičce trojúhelníku. Atomický web design

je založený na rozdělení prvků dle chemické terminologie. Vychází z toho, že atom je základním stavebním kamenem, jejich spojením vznikají molekuly a spojením molekul vzniká složitější organismus. Tato metodika široce popisuje její aplikaci pro webového designera, avšak pro potřeby kodéra je popsána velmi stručně.

Cílem diplomové práce bylo představit jednotlivé použití těchto metodik s použitím preprocesoru Sass se syntaxí SCSS na vzorovém stylopisu a zhodnotit jejich využití. Použití každé z vybraných metodik vede ke zlepšení srozumitelnosti a čitelnosti CSS kódu. Metodika BEM má velmi užitečnou konvenci pojmenování, která je velmi praktická a jednoduchá. Metodiky SMACSS, ITCSS a Atomický web jsou velmi přínosné pro použití u velkých webů, kde pomáhají udržovat logickou strukturu velkého množství CSS kódu. Správná struktura zajistí znovupoužitelnost, zvýší efektivnost psaní kódu. Díky logické organizaci lze jednoduše implementovat změny v kódu. Nevýhodou metodik SMACSS a ITCSS je nutnost správné identifikace vrstvy, do které má být komponenta zahrnuta, jelikož je nutné zapamatovat si mnoho kritérií. To se však netýká metodiky Atomického webu, kde je rozdělení na atomy, molekuly a organismy snadno zapamatovatelné a definice jednotlivých vrstev se dá snadno odvodit.

Každá metodika má své plusy a mínusy, v dalším zkoumání by bylo vhodné se zaměřit na vytvoření metodiky nové, která využívá všech kladných vlastností jednotlivých metodik. Tato metodika by mohla být založena na Atomickém webu z důvodu jeho jasného dělení, ovšem využívala by jmennou konvenci pro vytváření komponent, tak jak jí popisuje metodika BEM. Dále by bylo vhodné nezahrnovat soubory preprocesorů do kategorie atomů, ale udělit jim vlastní prostor jako tomu je u metodiky SMACSS a ITCSS.

Závěrem této práce je, že by se při vytváření CSS kódu měla využít alespoň některá metodika či vhodná kombinace více metodik, jelikož výhody použití metodik převažují nad nevýhodami.

# 8 Seznam použité literatury


[1] GODBOLT, Micah. *Frontend architecture for design systems: a modern blueprint for scalable and sustainable websites*. " O'Reilly Media, Inc.", 2016.

[2] ROBBINS, Jennifer Niederst. *Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics*. " O'Reilly Media, Inc.", 2018.

[3] PESLAK, Alan. Web Designer or Web Developer? A Detailed Comparison of Job Requirements. In: *Proceedings of the Conference on Information Systems Applied Research ISSN*. 2019. p. 1508

[4] STANÍČEK, Petr; DOUŠA, Marek. *Dobrý designér to všechno ví!*. Vydáno vlastním nákladem autora, 2016.

[5] ŘEZÁČ, Jan. *Web ostrý jako břitva: návrh fungujícího webu pro webdesignery a zadavatele projektů*. Baroque Partners, 2014.

[6] CANZIBA, Elvis. *Hands-On UX Design for Developers: Design, prototype, and implement compelling user experiences from scratch*. Packt Publishing Ltd, 2018.

[7] HAMM, Matthew J. *Wireframing essentials*. Packt Publishing Ltd, 2014.

[8] FROST, Brad. *Atomic design*. Pittsburgh: Brad Frost, 2016. ISBN 978-0-9982966-0-9.

[9] JACKSON, Wallace. *HTML5 quick markup reference*. Apress, 2016.

[10] GIBBINS, Nicholas. HTML5: COMP3220 Web Infrastructure. *EdShare Southampton* [online]. [cit. 2021-4-29]. Dostupné z: http://edshare.soton.ac.uk/20588/5/03b%20-%20HTML5.pdf

[11] BUDD, Andy; BJORKLUND, Emil. *CSS mastery*. Berkeley: Apress, 2016.

[12] DOWDEN, Martine a Michael DOWDEN. Architecting CSS: The Programmer's Guide to Effective Style Sheets. Berkeley, CA: Apress Media, 2020. ISBN 978-1-4842-5749-4.

[13] PRABHU, Anirudh. *Beginning CSS Preprocessors*. Apress, Berkeley, CA, 2015.

[14] MICHÁLEK, Martin. Průvodce CSS preprocesory: co a jak? *Vzhůru dolů* [online]. 10. 3. 2014 [cit. 2021-03-25]. Dostupné z: https://www.vzhurudolu.cz/blog/12-css-preprocesory-1

[15] MICHÁLEK, Martin. Průvodce CSS preprocesory: jak vám vylepší pracovní postupy. *Vzhůru dolů* [online]. 24. 3. 2014 [cit. 2021-03-25]. Dostupné z: https://www.vzhurudolu.cz/blog/14-css-preprocesory-3

[16] ROBERTS, Harry. High-level advice and guidelines for writing sane, manageable, scalable CSS. *Css { guide: lines; }* [online]. 28.12.2020 [cit. 2021-03-25]. Dostupné z: https://cssguidelin.es/#the-importance-of-a-styleguide

[17] MICHÁLEK, Martin. Zásady psaní respektujícího CSS. *Vzhůru dolů* [online]. 3. 6. 2018 [cit. 2021-03-25]. Dostupné z: https://www.vzhurudolu.cz/prirucka/rcss-zasady#2

[18] SULLIVAN, Nicole. Object Oriented CSS [online]. In: . 2010 [cit. 2021-01-25]. Dostupné z: https://github.com/stubbornella/oocss/wiki

[19] MICHÁLEK, Martin. OOCSS: objektové psaní CSS [online]. In: . 11. 5. 2017 [cit. 2021-01-25]. Dostupné z: https://www.vzhurudolu.cz/prirucka/oocss

[20] Methodology [online]. [cit. 2020-12-01]. Dostupné z: https://en.bem.info/methodology/

[21] ПОПКОВ, И. В.; КУРЗАЕВА, Л. В. БЭМ. КОМПОНЕНТНЫЙ ПОДХОД К ВЕБ-РАЗРАБОТКЕ. Международный студенческий научный вестник, 2018, 5: 165-165.

[22] MICHÁLEK, Martin. BEM: Pojmenovávací konvence pro třídy v CSS [online]. In: . 5. 6. 2017 [cit. 2021-01-12]. Dostupné z: https://www.vzhurudolu.cz/prirucka/bem#blok

[23] SNOOK, Jonathan. Scalable and Modular Architecture for CSS: A flexible guide to developing sites small and large. Ottawa, Ontario, Canada: Snook.ca Web Development, 2012. ISBN 978-0-9856321-0-6.

[24] KMETKO, Lubos. ITCSS: Scalable and Maintainable CSS Architecture [online]. 10.02.2016 [cit. 2021-01-18]. Dostupné z: https://www.xfive.co/blog/itcss-scalable-maintainable-css-architecture/

[25] KUNDRA, Adam. Proč je ITCSS nejpokročilejší metodika organizace CSS [online]. 16.06.2020 [cit. 2021-01-18]. Dostupné z: https://frontend.garden/proc-je-itcss-nejpokrocilejsi-metodika-organizace-css/

[26] ROBERTS, Harry. Manage large CSS projects with ITCSS [online]. 12.10.2016 [cit. 2021-01-10]. Dostupné z: https://www.creativebloq.com/web-design/manage-large-css-projects-itcss-101517528

[27] MONTE, Guillermo a Jim KING. A Robot Named Jimmy: CSS kód (stylopis). In: *GitHub* [online]. [cit. 2021-04-22]. Dostupné z: https://github.com/mezzoblue/csszengarden.com/tree/master/215

[28] MONTE, Guillermo a Jim KING. A Robot Named Jimmy: CSS kód (stylopis). In: *GitHub* [online]. [cit. 2021-04-22]. Dostupné z: https://github.com/jking90/zen-garden

[29] ARSENAULT, Cody. OOCSS - The Future of Writing CSS. *KeyCDN* [online]. 2019, 19.04.2019 [cit. 2021-04-20]. Dostupné z: https://www.keycdn.com/blog/oocss

[30] KONEČNÝ, Ondřej. Vývoj efektivity zápisu CSS: OOCSS. *MEDIUM* [online]. 2020, 29.08.2020 [cit. 2021-04-20]. Dostupné z: https://medium.com/@ondrej.konecny/efektivn%C3%AD-stylov%C3%A1n%C3%AD-od-html-element%C5%AF-po-styled-components-be9198308904

[31] CSS methodologies. *TGAM Pattern Library Documentation* [online]. [cit. 2021-04-22]. Dostupné z: https://globeandmail.github.io/ux-pattern-library/css-methodologies/

[32] DEVERO, Alex. Atomic Design – Your Ultimate Guide to Scalable & Modular CSS (Sass). *Alex Devero Blog: Learn about development, and programming, especially in JavaScript, TypeScript and React, and design* [online]. [cit. 2021-04-21].

# Příloha 1: Obsah elektronické přílohy

Adresáře *OCSS*, *BEM*, *SMACSS*, *ITCSS* a *AWD*, na kterých bylo demonstrováno použití jednotlivých metodik CSS, jsou přiloženy k elektronické podobě diplomové práce v archivu *zrojove_kody.zip*. Archív *zdrojove_kody.zip* zároveň obsahuje adresář *original* s původním stylopisem „A Robot Named Jimmy“.

# Příloha 2: Zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu

## Zadání diplomové práce

| | |
|---|---|
| **Autor:** | **Bc. Natalja Ljubišová** |
| Studium: | I1800115 |
| Studijní program: | N1802 Aplikovaná informatika |
| Studijní obor: | Aplikovaná informatika |
| **Název diplomové práce:** | **Atomický web dizajn** |
| Název diplomové práce AJ: | Atomic web design |

**Cíl, metody, literatura, předpoklady:**

1. Úvod
2. Navrhování pro web
3. Doporučené zásady při návrhu CSS kódu
4. Metodiky CSS
5. Vyhodnocení
6. Závěr
7. Literatura

Cílem práce: Představit a zhodnotit využití metodik CSS.


- FROST, Brad. Atomic design. Pittsburgh: Brad Frost, 2016.
- WATTS, Luke. Mastering Sass. Packt Publishing Ltd, 2016.
- Quick start. Methodology / BEM [online]. [cit. 2020-10-09]. Dostupné z: https://en.bem.info/methodology/quick-start/
- http://bradfrost.com/blog/post/atomic-web-design/


| | |
|---|---|
| Garantující pracoviště: | Katedra informačních technologií, Fakulta informatiky a managementu |
| Vedoucí práce: | Mgr. Daniela Ponce, Ph.D. |
| Datum zadání závěrečné práce: | 21.10.2020 |