Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Integrovaný webový informační systém organizace

Bc. Petr Čech

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Petr Čech

Informatika

Název práce

Integrovaný webový informační systém organizace

Název anglicky

Web-based integrated information system of an organization

Cíle práce

Cílem práce je vytvořit integrovaný webový informační systém pro řízení lidských zdrojů (personalistika) organizace typu menší IT firmy za využití Python 3 a frameworku Django. IS bude obsahovat všechny potřebné funkce pro správný chod organizace a zároveň bude vytvořen tříúrovňový přístup a k nim odpovídající agendy: administrátor, vedení organizace a zaměstnanec.

Metodika

Budou dodržovány standardy softwarového inženýrství, především agilní metodiky, WebML a UML. Na začátku bude porovnána vybraná vývojová platforma (Python/Django) s jinými alternativami (např. CMS s jazykem PHP nebo Java). Výsledný systém bude otestován v reálném prostředí.

Doporučený rozsah práce

80-120 stran

Klíčová slova

internet; intranet; CMS; web-technology; business and organization processes

---

Doporučené zdroje informací

BRUCKNER, T. *Tvorba informačních systémů : principy, metodiky, architektury.* Praha: Grada, 2012. ISBN 978-80-247-4153-6.

FORCIER, J. Python Web development with Django. NJ: Addison-Wesley, 2009. ISBN 9780132356138.

PILGRIM, M. *Ponořme se do Python(u) 3 = Dive into Python 3.* Praha: CZ.NIC, 2010. ISBN 978-80-904248-2-1.

SUMMERFIELD, M. *Python 3 : výukový kurz.* Brno: Computer Press, 2010. ISBN 978-80-251-2737-7.

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 25. 02. 2019

Čestné prohlášení

Prohlašuji, že svou diplomovou práci ”Integrovaný webový informační systém organizace” jsem vypracoval samostatně pod vedením vedoucího diplomové práce s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 29.3.2019 _____

Poděkování

V první řadě bych chtěl poděkovat doc. Ing. Vojtěch Meruňka, Ph.D. za příležitost vypracovat tak zajímavé téma diplomové práce. Děkuji za spolupráci a odbornou pomoc, která mi byla během tvorby diplomové práce poskytnuta. Poděkování patří i mé rodině za podporu během celého studia.

Integrovaný webový informační systém organizace

Shrnutí

Diplomová práce se zabývá webovým informačním systémem pro menší IT organizaci. V teoretické části jsou vysvětleny pojmy úzce spjaté s tématem včetně nástrojů potřebných pro tvorbu výsledného produktu. Dále je přiblížen pojem informační systém. Ujasněny jsou jak klíčová slova data, informace a znalost, tak i architektura IS, etapy životního cyklu IS nebo metodiky vývoje IS. Teoretická část dále nabízí porovnání výhod a nevýhod vybraných webových frameworků. Závěrem je představena organizace Neit Consulting s.r.o., pro níž je IS vyvíjen.

Praktická část se zaměřuje na tvorbu integrovaného webového informačního systému. Nejdříve je popsán sběr a analýza požadavků. Na tuto etapu navazuje tvorba návrhu. V této části jsou vytvořeny případy užití, stavové diagramy či uživatelské rozhraní pro lepší představivost, jak bude IS vypadat a fungovat. Poté už přichází na řadu samotný vývoj. Jednotlivé části informačního systému jsou zobrazeny a je nastíněno jejich řešení. Vývojový úsek se zabývá mimo jiné nastavením práv. Na závěr je nově vzniklý IS otestován a předán.

Klíčová slova: internet, intranet, CMS, web-technologie, podnikové a organizační procesy

Web-based integrated information systém of an organization

Summary

The diploma thesis deals with a web information system for a smaller IT organization. The theoretical part explains the concepts closely related to the topic, including the tools needed to create the final product. Furthermore, the concept, the information system, is explained. The data, key words, information, knowledge, as well as the IS architecture, the IS life cycle stages, and the IS development methodologies are explained. The theoretical part also offers a comparison of the advantages and disadvantages of selected web frameworks. Finally, Neit Consulting s.r.o., for which the IS is developed, is introduced.

The practical part focuses on creating an integrated web information system. First, the collection and analysis of requirements are described. This stage is followed by design creation. In this section, use cases, state diagrams, and user interface are created for better imagination, how the IS will look and work. Then the development itself is described. Individual parts of the information system are shown and their solution is outlined. The development section deals with, among other things, rights settings. Finally, the newly established IS is tested and handed over.

Keywords: internet, intranet, CMS, web-technology, business and organization processes

Obsah

1. Úvod

Před 30 lety měli mobil jen někteří, před 10 lety už měl mobil téměř každý a v dnešní době je nepředstavitelné, že by mobilní telefon někdo neměl. A podobně je to i s informačním systémem (IS).

Informační systémy zažily z historického hlediska velký rozmach v 90. letech minulého století. Hlavním důvodem byla snaha podniků získat výhodu oproti konkurenci. Objemy dat a informací se od této doby exponenciálně zvýšily a časem se z této výhody stala nutnost, aby podnik mohl vůbec fungovat. Pokud si však člověk myslí, že informační systémy jsou záležitostí pouze podniků, je na omylu. Doba pokročila a IS proniká i do kruhů jako je třeba školství. To vše svědčí o jejich praktičnosti a důležitosti.

Z tohoto důvod jsem se rozhodl vytvořit integrovaný webový informační systém pro společnost Neit Consulting s.r.o., ve které pracuji. Důvodem je současný nepříliš kvalitní IS pro řízení lidských zdrojů. Chtěl bych si tímto směrem zlepšit Python/Django znalosti, získat praxi s agilní metodikou, a hlavně vytvořit software, jenž usnadní společnosti každodenní práci.

2. Cíl a metodika

Cílem diplomové práce je za pomoci frameworku Django a Python 3 vytvořit integrovaný webový informační systém pro řízení lidských zdrojů, který bude šitý na míru organizaci odpovídající menší IT firmě. Informační systém bude obsahovat krom všech funkcí nezbytných pro chod organizace i tříúrovňový přístup.

Při tvorbě informačního systému budou dodržovány standardy softwarového inženýrství, především agilní metodiky a UML. Výsledek bude otestovaný a plně funkční.

3. Literární rešerše

3.1. Základní pojmy

3.1.1. Python

Python je vysokoúrovňový skriptovací programovací jazyk, jehož počátky sahají do roku 1981, kdy jej navrhnul Guido van Rossum. Zajímavé na tomto jazyku je podpora různých programovacích paradigmat. Python je multiplatformní, tudíž program v něm napsaný lze spustit jak na Windows, tak i na unixových systémech jako je Linux, BSD, Mac OS X, pouhým zkopírováním souboru, či souborů, které tvoří daný program, na cílový stroj, aniž by jej bylo nutné „sestavovat“ nebo kompilovat. [1]

V roce 2008 byl vydán Python 3.x, jež je nástupcem zastaralé verze 2.x. Tyto dvě verze jsou nekompatibilní. Python 3.x odstraňuje řadu nedostatků a chybných návrhů jazyka. Nicméně verze 2.x se pořád udržuje při životě, dokonce byla obohacena o některé vlastnosti 3.x. Oficiální prohlášení vývojářů z roku 2017 oznamuje ukončení podpory 2.x v roce 2020. Poté by už mohlo dojít k definitivnímu přechodu na 3.x. [1]

Jak už zaznělo, Python je hybridní jazyk, neboť podporuje různá programovací paradigmata. Co to znamená? Při psaní programu je možné využívat objektově orientované paradigma, procedurální paradigma a v omezené míře i funkcionální. Python má výborné vynikající schopnosti, je dobře čitelný a krátký. Navíc se jednoduše vkládá do jiných aplikací, kde funguje jako skriptovací jazyk. [1][2]

Python se řadí mezi pomalejší jazyky, ale u běžných aplikací jsou rozdíly ve výkonu jazyků nepatrné. Na výkonově kritické programy se bohužel nehodí. Naopak velkou výhodou Pythonu je, že se dodává s téměř kompletní standartní knihovou, díky čemuž existuje možnost třeba stahovat soubor z internetu nebo rozbalit zkomprimovaný archivní soubor. [1]

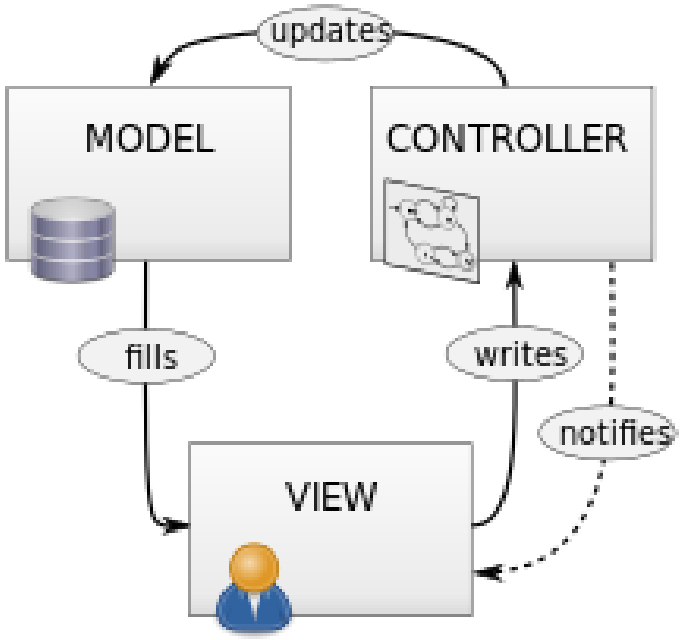
3.1.2. MVC

Model View Controller nebo zkráceně MVC je populární softwarová architektura pro vývoj webových aplikací. MVC odděluje aplikační vrstvu od uživatelského rozhraní. Rozdělení 3 nezávislých částí (model, view, controller) musí být tedy vytvořeno způsobem, aby při změně jedno z nich byl vliv na ostatní části minimální. Obrázek č. 1 znázorňuje procesy mezi částmi. [3]

Logika MVC architektury byla vysvětlena, teď je důležité vysvětlit jednotlivé části:

- Model (model) – Model představuje nejnižší úroveň MVC architektury. Poskytuje rozhraní s databází obsahující data aplikace.
- Pohled (view) – Pohled rozhoduje o tom, jaké informace se zobrazí uživateli a shromažďuje od nich informace.
- Ovladač (controller) – Ovladač zprostředkovává informaci mezi modelem a pohledem [3]

Obrázek 1: MVC architektura



zdroj: https://cs.wikipedia.org/wiki/Model-view-controller

MVC architektura tedy funguje následovně. Ovladač dostane požadavky od aplikace. Z modelu získá potřebná data a ty jsou pak odeslány do pohledu, kde se vytvoří grafické rozhraní určené pro uživatele. [3]

3.1.3. Frontend a Backend

Podstatné pro tuto diplomovou práci je vysvětlit slova frontend a backend. Začneme prvně jmenovaným. Pod pojmem frontend se skrývá vše, co uživatel spatří, když si rozklikne webovou stránku. Jedná se tudíž o grafické prvky stránky. Základní kamenem je HTML (HyperText Markup Language). Tento značkovací jazyk vytváří kostru celé webové stránky. Škála využití je od strukturování stránky až po přidávání obrázků, tabulek nebo úpravu textu. Další technologií je CSS (Cascade Styling Sheet), jež navazuje na HTML a stará se o vzhled

stránky. Konkrétními příklady může být změna fontů, barev textu a pozadí nebo velikost obrázků a pozicování. Poslední technologií je Javascript a ta se stará o dynamickou stránku. Pod tímto pojmem si lze představit akci při najetí myši na text tak, že text změní barvu nebo se dokonce sám změní a podobné interakce. [4]

Základní vysvětlení frontendu bychom měli. Zajímavější, a ne méně důležitý při tvorbě webových aplikací, je backend. Tuto část spravuje konkrétně v našem případě framework Django využívající Python. Ale samozřejmě to mohou být i jiné programovací jazyky typu PHP, Ruby nebo třeba Java. Backend slouží pro kompletní správu webového informačního systému a je pro uživatele neviditelný. Obvykle se skládá ze serveru, aplikace a databáze. Pokud se třeba uživatel přihlašuje do IS, pak si otevře webovou stránku, vyplní údaje a klikne většinou na tlačítko přihlásit. V tu chvíli se dostává na řadu backend, jež kontroluje, zdali je uživatel zapsán v databázi a jestli souhlasí přihlašovací údaje. Pokud vše proběhne správně, uživatel je odměněn přihlášením. [4]

3.1.4. Framework

Framework slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny API, podporu pro návrhové vzory nebo doporučené postupy při vývoji. [4]

Framework byl vynalezen, aby usnadnil práci návrhářům a vývojářům. Od ostatních knihoven se odlišuje inverzí kontrolou, rozšiřitelností a nemodifikovatelným kódem. Inverze kontroly je chápána tak, že framework, nikoliv volající, řídí průběh programu. Uživatelé mohou framework rozšiřovat a programátoři mohou přidávat specifické funkce, ale nelze modifikovat samotný kód frameworku. [4]

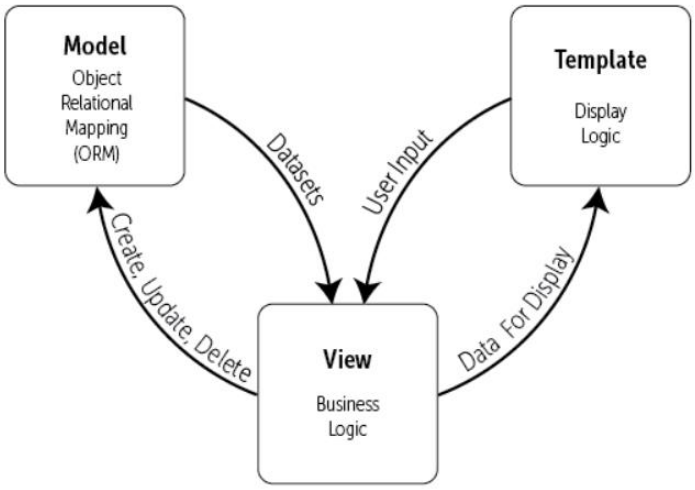
3.1.4.1. **Django**

Django je webový aplikační framework napsaný v Pythonu, který se volně drží MVC architektury, která byla vysvětlena výše. MVC architektura je zde lehce upravená (viz. obrázek č. 2). [4]

Model má stejnou funkcionalitu. Pohled spravuje většinu dat aplikace, aplikační logiku a zasílání zpráv. Poslední část šablona (template) poskytuje logiku zobrazení a je rozhraním mezi uživatelem a Django aplikací. [4]

Modely v Djangu poskytují objektově-relační mapování (ORM). ORM je výkonná programovací technika, která usnadňuje práci s daty a relačními databázemi. Django oficiálně podporuje databáze: PostgreSQL, MySQL, SQLite, Oracle.

Obrázek 2: MVC Architektura v Django frameworku



zdroj: Python Web development with Django, 2009

S Djangem jde ruku v ruce slovní spojení „Don't Repeat Yourself" ve zkratce DRY a je to vlastně jeden z hlavních principů Djanga. Tento princip je obzvláště patrný v dědičnosti šablon. Pokud se programátor podívá na stránky IS zjistí, že menu, hlavička a patička je ve většině šablon stejná a nijak se nemění. Proto jsou tyto části stránky označeny jako rodič. Potomek je jen ta část stránky, která se neustále mění, např. obsah stránky. Není potřeba zdůrazňovat, jak moc princip DRY usnadňuje programátorovi práci. [4]

3.1.5. UML

UML (Unified Modeling Language) je soubor grafických notací, který byl vytvořen pro snadnější vývoj SW. Příčinou vzniku UML bylo především rozšíření objektově orientovaného programování. Hlavní myšlenkou byla snaha dívat se na informační systémy z různých pohledů. Vznikalo velké množství specifikací a standardů. Počátky UML sahají do 90. let, kdy se firmě Rational Software povedlo sjednotit několik metod. Čas ukázal, že je tento standard praktický, protože usnadňuje komunikaci, všichni rozumí dokumentaci různých projektů a nemusí se učit neustále nové jazyky. [5]

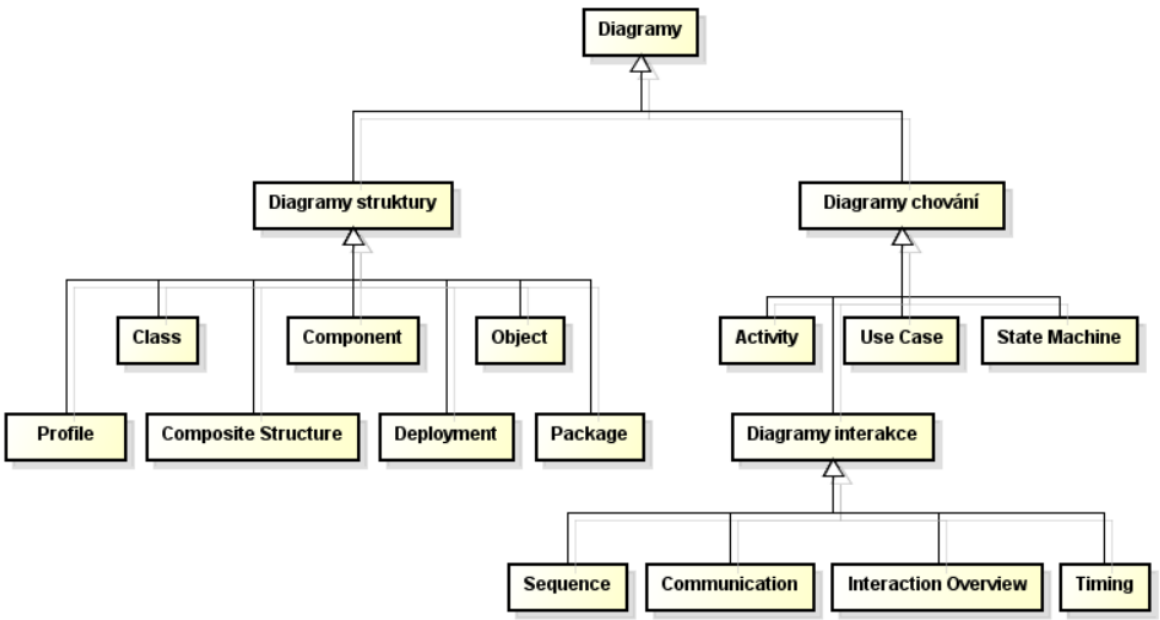
Význam UML je zřejmý. Roste komplexnost IS, na kterých se podílí mnoho lidí a díky UML lze systém navrhnout ještě před začátkem realizace. UML pomáhá třeba při

domluvě se zákazníkem. Grafické zobrazení činí IS přehlednější a zákazník si dokáže lépe představit, jak bude vše fungovat. [5]

UML se využívá třemi způsoby:

- Náčrt – Jak již bylo řečeno, grafické zobrazení činí IS přehlednější. Zákazník se v nakreslených diagramech dokáže orientovat, lépe chápe IS. Vyústěním je snadnější komunikace a snížení rizika, že dojde k nedorozumění mezi zákazníkem a dodavatelem a IS bude špatně navržen. Nejdůležitější vlastností diagramů je abstrakce. Diagramy představují pohled na určitou část IS. Zobrazují jen to podstatné a nepodstatné věci zahazují, což je další velké plus při komunikaci se zákazníkem.
- Plán – Jedná se o mnohem detailnější náčrt sloužící jako plán implementace pro programátory a následně jako součást dokumentace. Diagramy jsou vytvářeny v CAD nástrojích.
- Programovací jazyk – Samotný UML není programovací jazyk, ale z detailního diagramu je možné vygenerovat šablonu, kterou lze využít pro implementaci. Například v databázích se tyto modely používají pro vygenerování základních skriptů. [5]

Obrázek 3: UML diagramy



zdroj: https://www.itnetwork.cz/navrh/uml/uml-uvod-historie-vyznam-a-diagramy

Diagramy se rozdělují na:

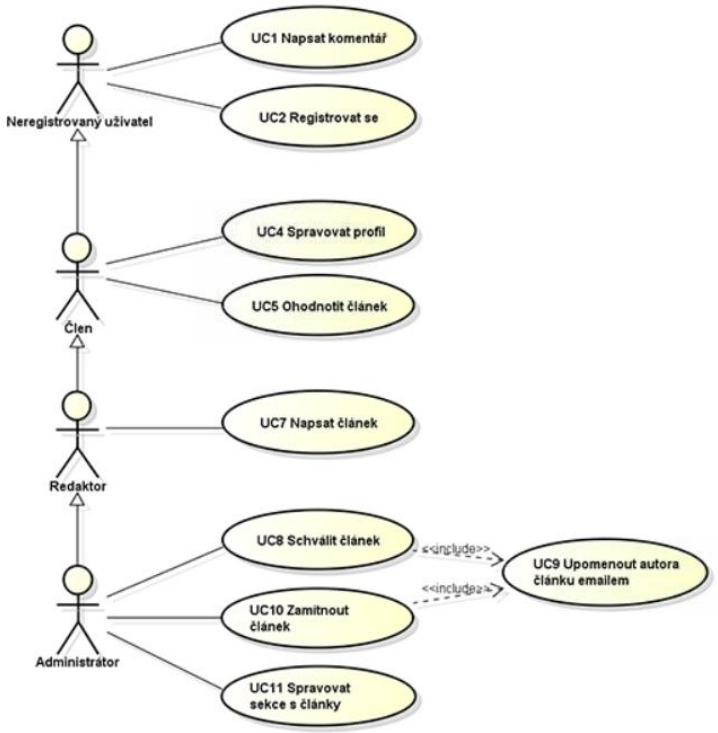
- Diagramy struktury – Popisují, z čeho je systém složený.
- Diagramy chování – Popisují, jak systém funguje.
- Diagramy interakce – Tyto diagramy spadají pod diagramy chování a popisují vzájemnou interakci mezi jednotlivými částmi systému. [5]

3.1.5.1. **Use Case**

Asi nejznámějším UML diagramem je Use Case diagram. Tento diagram zobrazuje chování systému z pohledu uživatele. Diagram má za úkol popsat funkcionalitu systému neboli co od něj zákazník očekává. V diagramu je zobrazeno, co má systém umět, nikoli jak to bude dělat. Je podstatné se shodnout na tom, co má systém dělat, a proto je Use Case první diagram, který se vytváří. Komponenty Use Case jsou:

- Případ užití – Definuje jednu funkcionalitu, jež by měl IS umět. V diagramu je zakreslen jako elipsa.
- Aktor – Role, která komunikuje s jednotlivými případy užití. Aktor se znázorňuje postavou z čar.
- Vztahy mezi aktorem a případem užití. [6]

Obrázek 4: Use Case diagram



zdroj: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram

Na obrázku č.4 vidíme 4 aktory: Neregistrovaný uživatel, Člen, Redaktor, Administrátor. Neregistrovaný uživatel může psát komentáře a registrovat se. Člen dědí případy užití z Neregistrovaného uživatele, navíc může spravovat profil a hodnotit články. Redaktor dědí z Člena a tím pádem i z Neregistrovaného uživatele, což ale není podstatné, neboť všechny případy užití Neregistrovaného uživatele jsou zděděny Členem. Redaktor rozšiřuje již zděděné případy užití o možnost napsat článek. Nejvýše postaveným aktorem je Administrátor, který spravuje sekce a články, které může schvalovat nebo zamítat. [6]

3.2. Informační systém

„Informační systém je soubor lidí, technických prostředků a metod (programů), zabezpečujících sběr, přenos, zpracování, uchování dat, za účelem prezentace informací pro potřeby uživatelů činných v systémech řízení.“ (Molnár, 2009) [7]

3.2.1. Data, informace, znalost

Tyto tři pojmy spolu úzce souvisí. Data jsou nejzákladnější fakta, v IS jsou to třeba čísla zaměstnanců. Pokud se pracuje s daty, je důležité dodržovat jejich stejnou hodnotu na všech místech, kde se objevují. Není možné tedy, aby měl zaměstnanec například rozdílná identifikační čísla ve dvou různých tabulkách. Pokud není dodrženo toto pravidlo, pak není zajištěna datová redundance a data ztrácejí svoji konzistenci. Právě datová redundance a konzistence jsou jedny ze základních pilířů pro práci s daty. Nezbytné je také zajistit integritu dat, což znamená, že když se změní fakt v reálném světě, pak tato změna musí nastat i v IS. [8]

Pro snadnější pochopení chápejme data jakožto části trati v železniční sadě. Nastavíme-li vztah mezi jednotlivými částmi, začnou nabývat na hodnotě, protože se začnou spojovat a tvořit železniční okruh. Takto fungují data a informace. Informace je tedy skupina organizovaných dat. Pro sales manažera nejsou podstatné jednotlivé tržby provedené každý den. Pokud shromáždíme jednotlivé tržby a dáme jim logiku, pak zjistíme například celkové měsíční tržby a tyto informace jsou už pro manažera velmi důležité. [8]

V předchozím odstavci bylo použito slovo logika pro přeměnu dat na informaci. Není to úplně správné pojmenování. Mnohem lepší pojmenování je znalost, což je velmi významný faktor pro správné nastavení pravidel a vztahů mezi daty. V naší modelové železniční síti je znalost například schopnost vymezit dostatečně velkou plochu pro rozvržení tratě nebo volba počtu vlaků. [8]

3.2.2. Software

Pro správnou funkci IS je důležité zajistit vhodný software (SW). SW obsahuje množinu modulů, komponent, objektů, služeb a vazeb mezi nimi. Při tvorbě IS je nezbytné zajistit soulad mezi IS a podnikem. Velkou chybou bývá nakoupení SW, jež vytváří nekompatibilitu mezi IS a strukturou podniku. Největší důraz z hlediska souladu je kladen na dynamiku a složitost. [9]

Dynamika je v informačních technologiích častý pojem. Tím pádem se očekává hodně změn v čase. V podnicích dochází k mnoha změnám. Zaměstnanci přichází, odchází, mění se postupy, struktury podniku. Postupem času se v SW hromadí spousta funkcí, jež už nejsou aktuální a funkce, jež by zaměstnanci potřebovali, nejsou naprogramovány. Důsledkem je zpomalení zaměstnance. Proto by měla existovat úzká vazba mezi strukturou podniku a ICT. [9]

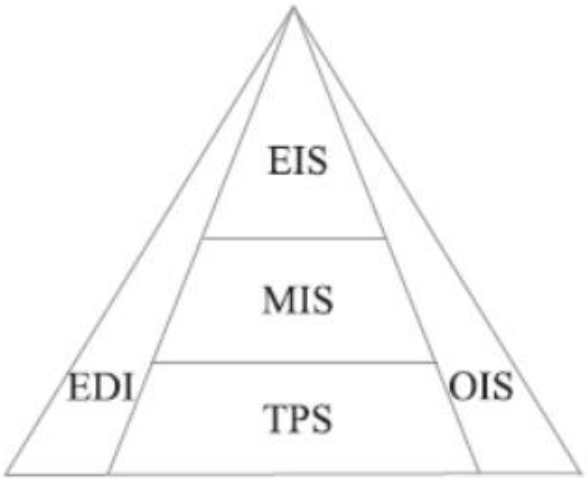
Podnik je většinou velice složitý systém tvořící velké množství zaměstnanců rozdělených do více oblastí. Každý zaměstnanec, respektive okruh zaměstnanců, potřebuje mít v IS potřebné věci pro svojí práci. Tudíž by při tvorbě SW měli programátoři úzce spolupracovat s manažery, aby se na žádný okruh zaměstnanců nezapomnělo a předešlo se problémům, jež mohou vyeskalovat až v nepoužitelnost SW. [9]

3.2.3. Globální architektura

Globální architektura je základní schéma zobrazující hrubou podobu IS. Využívá abstrakce, aby byla co nejjednodušší a nejsrozumitelnější. Globální architektura se dělí do tří vertikálních úrovní odpovídající členění managementu a obsahuje také i horizontální dimenzi, kde jednotlivé části odpovídají podnikovým útvarům a funkcím. [10]

Globální architektura funguje jako prostředek komunikace mezi zákazníkem a dodavatelem, pomáhá definovat priority nasazování IS do provozu. Kvalitní návrh architektury IS pomáhá minimalizovat dodatečné náklady spojené s chybným zadáním nebo řízením podniku. Jednotlivé komponenty architektury musí být v souladu s podnikovou strategií, funkčními a řídicími postupy v podniku a musí zajišťovat flexibilní přizpůsobení měnícím se podnikovým procesům. Globální architektura zahrnuje také odolnost proti technickým závadám, negativním zásahům obsluhy apod. [10]

Obrázek 5: Základní bloky globální architektury



zdroj: https://akela.mendelu.cz/~rybicka/prez/infsyst.pdf

3.2.3.1. TPS (Transaction Processing System)

TPS reprezentuje druhy činností podniku na úrovni operativního řízení v krátkodobém horizontu. Je vyžadována vysoká rychlost odezev a spolehlivost. Hlavní úkol TPS spočívá v pořizování a aktualizaci dat, jejich evidenci a základních přehledech. Výstupy z TPS se využívají ve vyšších vrstvách IS. [10]

Na úrovni operativního řízení se nejčastěji používají aplikace podporující konstrukční a technologické procesy jako jsou CAD (Computer Aided Design) pro konstrukce a návrhy, CAM (Computer Aided Manufacture) pro automatizovanou podporu řízení výrobních provozů, MRP (Material Resource Planning) pro plánování zdrojů materiálu, ERP (Enterprise Resource Planning) pro plánování podnikových zdrojů. Často se můžeme setkat i s aplikacemi typu typu CRM (Customer Relationship Management) pro řízení vztahů se zákazníkem, RIS (Reservation IS), což jsou rezervační systémy nebo GIS (Geographic IS), jež digitalizuje mapy, vytváří územní modely atd. [10]

3.2.3.2. MIS (Management Information System)

MIS podporuje taktické řízení podniku ve střednědobém horizontu. Opírá se o vrstvu operativního řízení a na základě ucelených a sumarizovaných dat z TPS umožňuje kontrolovat a řídit základní podnikové aktivity. Jsou zde kladeny vysoké nároky na bezpečnost, dostupnost, zálohování a archivaci dat a průkazností operací. Podstatná je volba aplikačního software a kvalita použitých metod a nástrojů. [10]

Taktická úroveň řeší oblasti týkající se analýzy, plánování a modelování. Například řízení jakosti, lidské zdroje, správa zdrojů, marketing apod. Mezi MIS a EIS existují systémy DSS (Decision Support System), jež podporují rozhodování a tvoří mezi nimi přechod. [10]

3.2.3.3. EIS (Executive Information System)

EIS obsahuje aplikace určené pro podporu vrcholového vedení a ke strategickému řízení podniku v dlouhodobém horizontu. Aplikace sbírá data nejen z TPS a MIS, ale i z externích zdrojů. Sledují se zde historická data a vytvářejí se komplexní analýzy současného stavu, budoucího vývoje, prognózy a trendy. [10]

Na úrovni EIS se spolupracuje s OLAP (OnLine Analytical Processing) za využití specializovaných SW nástrojů. Základní myšlenkou OLAP je uložení dat ve vícedimenzionální databázi. Vícedimenzionální znamená, že na uložená data lze nahlížet různými pohledy. Ekonomický pohled zkoumá data, aby zjistil obrat a zisk, ale existuje mnoho dalších pohledů, třeba časový. [10]

Výstupy EIS vrstvy jsou určeny pro vrcholový management, důsledkem jsou relativně nízké nároky na zajištění provozu. Důraz je kladen na podporu uživatelů, jednoduché ovládání, přehlednost prezentace dat a na možnost průběžně měnit aplikace. [10]

3.2.3.4. OIS (Office Information System)

Pod OIS spadá sada aplikací se zaměřením na podporu kancelářských prací. Podstatou je snížení nároků na administrativní operace, zjednodušení zpracování dat, zrychlení a zefektivnění komunikace, zvýšení formální úrovně výstupů, snížení nároků na administrativní operace. [10]

OIS poskytuje textový a tabulkový procesor, SW pro tvorbu prezentací, klienta elektronické pošty, plánovací kalendář a aplikace pro podporu týmové práce a pro správu dokumentů.

3.2.3.5. EDI (Electronic Data Interchange)

EDI obsahuje aplikace zprostředkovávající elektronickou komunikaci s okolím podniku – dodavateli, odběrateli, zákazníky atd. Využívá se zde možností poskytovaných sítí Internet. [10]

3.2.4. Typy IS

Nejjednodušším možným způsobem by se daly informační systémy rozdělit na veřejné informační systémy a podnikové informační systémy. Rozdíl v nich je celkem jasný. Veřejné IS uchovávají a nabízejí takové informace, ke kterým má přístup veřejnost nebo alespoň její určitá část. Jsou financovány z dobročinných příspěvků, z veřejných rozpočtů apod. Na druhou stranu podnikové IS jsou uzavřené systémy, do nichž se dostanou jednotlivci s uděleným přístupem, tudíž třeba zaměstnanci daného podniku. [10]

3.2.5. Etapy životního cyklu

Tvorba informačního systému je velmi složitá a časově náročná činnost. Celý proces od vzniku projektu až po ukončení provozu IS se nazývá životní cyklus. Ten se dále rozděluje na etapy, jejichž úspěšné dokončení přibližuje zákazníka k dokončení IS. [10]

3.2.5.1. Úvodní studie

Životní cyklus začíná úvodní studií. Za účelem nalezení co nejlepšího řešení, dochází k přesné specifikaci potřeb zákazníka. Očekávána je od něj informační strategie, která obsahuje analýzu IS a definuje zákazníkovi cíle. Na základě informační strategie se rozhoduje, zdali je za potřebí celková rekonstrukce IS nebo jen částečná. Občas problémy s IS vyřeší dodatečné proškolení zaměstnanců nebo drobné personální změny. Jedná se asi o nejpříjemnější řešení, neboť problémy s IS nejsou nijak velké a dají se snadno vyřešit. Pokud problémy s IS přesahují určitou mez, je navržena buď kompletní rekonstrukce nebo dokonce tvorba nového IS. Všechny nedostatky a jejich případné řešení jsou sepsány do dokumentu Úvodní studie. Ta navíc obsahuje i plánovaný odhad doby a nákladů potřebných na projekt a definici požadavků pro řešení problémů. [10]

3.2.5.2. Analýza

Po úvodní studii následuje neméně důležitá analýza. V této fázi se rozděluje problém na menší části, které jsou lépe srozumitelné. Typické pro analýzu je tvorba modelů, které zjednodušují zobrazení systému. Co je to vlastně model? Vraťme se opět k železniční sadě. Každá část této sady je svým způsobem modelem, například vlak je modelem reálného vlaku. Jelikož IS obsahuje nepřeberná množství informací, jsou modely ztvárněny pomocí diagramů. Vznikají datové modely a funkční modely za cílem popsat celý systém. Analýza

se snaží pomocí modelů dostatečně poznat informační systém a vytvořit tak základ pro další tvorbu IS. [10]

Požadavky na analýzu:

- Definice informačního systému včetně cílů a hranic
- Definice oblastí, které budou a nebudou spadat do informačního systému
- Definice uživatelů IS
- Definice vztahů zaměstnanců k procesům a činnostem [10]

Analýza zároveň odhaluje nedostatky IS, ať už se jedná o problémová místa nebo části informačního systému s prvky nejednoznačnosti. Tyto nedostatky musí být vyřešeny ze strany zákazníka ihned na začátku a dokud nejsou vyřešeny, je zbytečné pokračovat další fází. [9]

3.2.5.3. Návrh

Po úspěšném překlenutí analýzy nastává čas na globální a detailní návrh. Globální návrh má na starost návrh na logické úrovni, detailní pak řeší úroveň fyzickou. [10]

Globální návrh využívá informace z dokumentace vzniklé v úvodní studii a převádí je do větších podrobností. Tato fáze probíhá na dané hardware a software platformě. Vznikají HW požadavky na počty a rozmístění terminálů, servery a rychlost odezvy a přenosovou kapacitu datových strojů. Na druhou stranu SW zas obsahuje návrh architektury, protokoly, konzistentní chování všech modulů a interaktivní a dávkové zpracování úloh. Je vytvořen datový model. Funkční model je rozebrán až na úroveň transakcí, rolí uživatelů a nastavení oprávnění. [10]

Detailní návrh spoléhá na to, že globální návrh vyřešil hardware a software komponenty a soustředí se tak na databázový systém. Řeší vše od datových typů až po vstupy a výstupy. [10]

3.2.5.4. Implementace

Implementace realizuje myšlenky předchozích fází. Konkrétně se specializuje na:

- Zisk hardware, software a služeb
- Vývoj software a jeho modifikaci
- Tvorbu databáze

- Testování programů, postupů a hardware
- Dokumentaci – Dokumentace je alfa omega. Je potřeba jak při řešení chyb, úpravě IS, tak i pro účely školení.
- Volbu metody zavádění informačního systému
- Školení uživatelů [10]

3.2.5.5. Zavedení systému do provozu

V této fázi dochází k instalaci, konfiguraci a testování dostupného hardware a software. Zaměstnanci jsou školeni a přicházejí prvně do kontaktu s novým informačním systémem. Ten obsahuje zpočátku testovací data, s nimiž se provádí zátěžové testy. Pokud jsou výsledky pozitivní, dochází k překlopení testovacího prostředí na produkční. Když jsou vytěžena všechna data ze starého informačního systému, je starý IS vyřazen. Přechod z testovacího na produkční prostředí by měl probíhat vždy co nejrychleji, neboť data přicházejí každý den, každý okamžik a je potřeba je ihned zpracovávat. [10]

Strategie pro zavedení nového IS:

- Paralelní strategie – Jak předpona napovídá, jedná se o strategii, kdy něco probíhá současně (paralelně). Není těžké uhádnout, že se jedná o informační systémy. Starý a nový informační systém běží tedy současně, dokud nejde ke kompletnímu přesunutí dat. Jakmile k tomu dojde, starý IS je vypnut a běží už jen nový. Tato strategie je velmi efektivní a využívá se nejčastěji ve větších podnicích, kde jsou i vetší IS. Navíc nabízí slušnou pravděpodobnost, že se nebude nacházet v novém informačním systému mnoho chyb. Bohužel je paralelní strategie náročná na finanční prostředky, vyžaduje hodně času a častou údržbu.
- Strategie pilot – Nový informační systém je před nasazením do produkce otestován na testovacím serveru. Pokud funguje, je nasazen, pokud nikoliv, nic se neděje, data nejsou ztracena, protože pořád běží starý informační systém. Strategie pilot se doporučuje pro velké podniky s velkým počtem uživatelů, kteří mají možnost si IS vyzkoušet ještě před nasazením.
- Postupná strategie – Důležité pro tuto strategie je rozdělit nový informační systém do implementovatelných částí. Poté je postupně nahrazována část nového IS za část starého IS. Postupná strategie nabízí dobrou ovladatelnost při implementaci IS, avšak dokáže být na druhou stranu i velmi náročná na čas.

- Přímá strategie – Nejrychlejší a zároveň nejrizikovější strategií ze všech zde zmíněných je přímá strategie. Principem této strategie je rychlý přechod ze starého informačního na nový informační systém, a to takovým způsobem, že je starý IS ihned zahozen. Přímou strategii můžou zkomplikovat chyby, které se objeví při takto drastickém přechodu. Proto je doporučeno využívat strategii na malých IS. [11]

3.2.5.6. Provoz a správa

Pro tuto fázi existují dva možné scénáře. Který z těchto dvou bude vybrán, záleží na domluvě mezi dodavatelem informačního systému a zákazníkem. [10]

První scénář počítá s dodáním zákazníkovi dokumentace s veškerými informacemi o IS. Dokumentace je natolik plnohodnotná, že interní zaměstnanci po zaškolení dokážou IS spravovat. Zákazník kontaktuje dodavatele IS pouze v krajních případech. Druhý scénář předpokládá spolupráci s dodavatelem informačního systému, který bude IS nadále provozovat a spravovat. [10]

V této fázi často dochází k:

- Opravě hardware nebo nákup nových komponent
- Nákupu hardware kvůli posílení výkonu
- Tvorbě záložních serverů
- Modifikaci software
- Aktualizaci dokumentace a případnému přeškolení uživatelů [10]

3.2.5.7. Ukončení provozu

Tato fáze nastává při výměně informačních systémů. Starý IS je zálohován, data převedena do nového IS. [10]

3.2.6. Modely životního cyklu

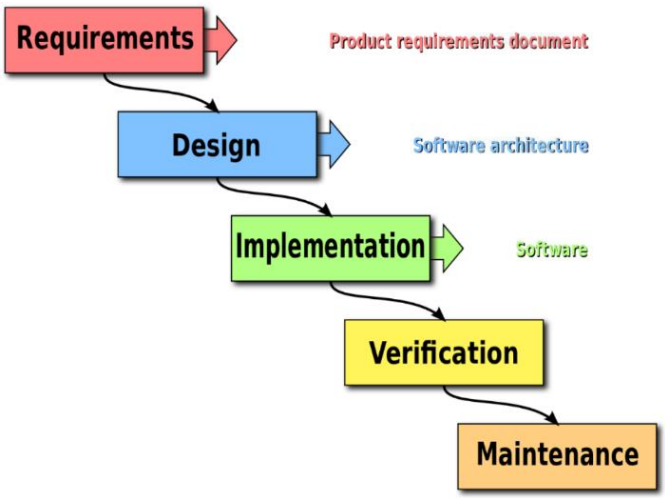
Fáze životního cyklu sepsány výše dávají v takovémto pořadí sice smysl, ale ve skutečnosti se obvykle překrývají, opakují anebo dokonce chybí. Jakým způsobem jsou tyto fáze využívány, je popsáno v modelech životního cyklu. [12]

3.2.6.1. Vodopádový model

Vodopádový model je jeden z nejstarších modelů softwarového inženýrství. Doposud se využívá na mnoha velkých projektech. Primárním znakem je plánování již na počátku před vývojem systému, díky čemuž se lépe objevují konstrukční nedostatky. Model je vhodný pro projekty, kde je značným problémem kontrola kvality. Vodopádový model se skládá z několika fází, tyto fáze na sebe navazují a započínají při ukončení předchozí fáze. Začíná shromážděním veškerých požadavků, po globální a detailní návrh, implementaci, testování a končí údržbou. Hlavní výhodou modelu je jednoznačně jednoduchost a srozumitelnost. Fáze jsou jasně vymezené a postupy s výsledky zdokumentované. [12]

Vodopádový model ale také obsahuje některé nedostatky. Pokud je software špatně navržen, pak je objevena chyba až na konci vývoje. Navíc nedokáže reagovat na měnící se požadavky, a tak se nehodí pro dlouhé projekty. [13]

Obrázek 6: Vodopádový model



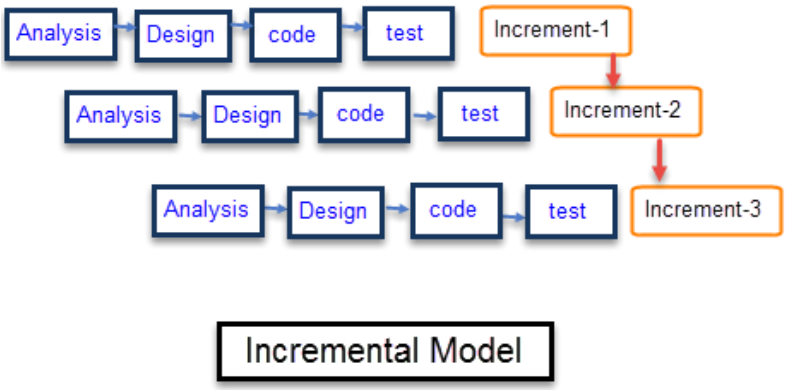
zdroj: https://en.wikipedia.org/wiki/Waterfall_model

3.2.6.2. Inkrementální model

Inkrementální model je založen na postupném navržení, implementování a testování, dokud není informační systém dokončen. Dokončením chápeme splnění všech požadavků. Tento model iterativně kombinuje prvky Vodopádového modelu. [12]

Celý model se skládá z několika inkrementů, přičemž každý inkrement přidává do IS něco nového. Při prvním inkrementu, jež má nejvyšší prioritu, je zákazníkovi dodán nedokončený IS, který již lze použít. Na základě zpětné vazby od zákazníka je vytvořen plán úprav a celý proces pokračuje, dokud není dodána finální verze informačního systému. [12]

Obrázek 7: Inkrementální model



zdroj: https://www.guru99.com/what-is-incremental-model-in-sdlc-advantages-disadvantages.html

Velkou výhodou je zpětná vazba od zákazníka po každém inkrementu a jednoduché otestování funkcionalit IS. Když se IS vytváří po částech, je snadnější se přizpůsobit případné změně požadavků. Na druhou stranu se inkrementální model kvůli složitosti nehodí pro menší projekty. Navíc se mohou objevit problémy s architekturou systému, neboť se neshromažďují všechny požadavky na začátku celého životního cyklu. [13]

3.2.6.3. Spirálový model

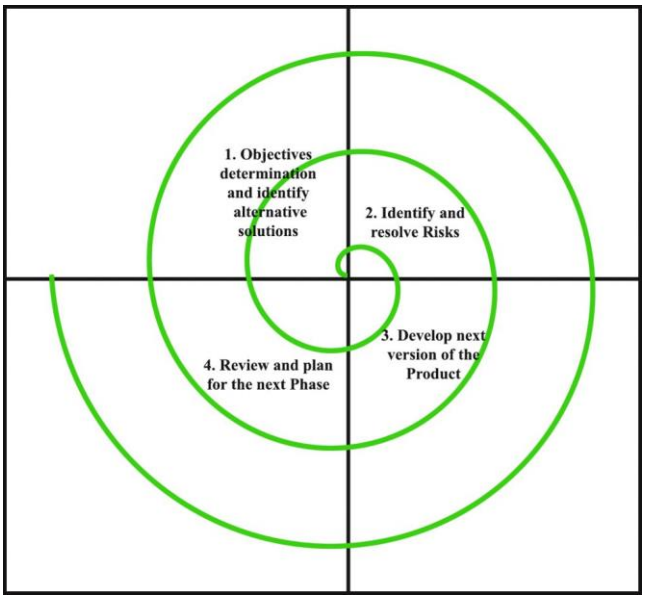
Spirálový model je jeden z nejdůležitějších modelů životního cyklu. Založen je na iterativním vývoji spolu s prvky vodopádového modelu. Velkou výhodou tohoto modelu je podpora řízení rizik. Model se skládá ze 4 částí (kvadrantů) a několika spirál. Spirály představují jednotlivé iterace (fáze) a jejich počet se mění závisle na složitosti projektu. [12]

Části spirálového modelu

- Stanovení cílů a určení alternativních řešení – Od zákazníka jsou shromažďovány požadavky, identifikují se cíle. Probíhá analýza a navrhují se alternativní řešení pro tuto fázi.
- Identifikace a řešení rizik – Vyhodnotí se všechna řešení za účelem zvolení nejlepšího možného řešení. Identifikují se rizika a vytvoří se prototyp vyladěný od těchto rizik, které byly strategicky odladěny.
- Vývoj další verze produktu – Třetí fáze se zabývá vývojem a jeho následným testováním. Výsledkem této fáze je nová verze software.
- Revize a plán pro další fázi – Zákazník dostane do rukou nejnovější verzi, ohodnotí ji. Zákazníkovo hodnocení je výchozím bodem pro další fázi. [12]

Riziko je každá nepříznivá situace, jež by mohla negativně ovlivnit úspěšné dokončení projektu. Pro odstranění rizik využívá spirálový model prototypy. V prototypu jsou uvedeny základní vlastnosti systému, které se každou fázi (iteraci) upřesňují. [12]

Obrázek 8: Spirálový model



zdroj: https://www.geeksforgeeks.org/software-engineering-spiral-model/

Výhodami tohoto modelu je právě řešení rizik, flexibilita a spokojenost zákazníka, jelikož zasahuje od začátku do dění, vidí jednotlivé fáze a zvyká si na IS během vývoje. Navíc je spirálový model vhodný pro velké a kritické projekty. Na druhou stranu je z těchto informací zřejmé, že se model nehodí na malé projekty, neboť je velmi složitý a finančně náročný. Úspěch modelu velmi závisí na analýze rizik, na níž je potřeba velmi zkušených odborných znalostí. [13]

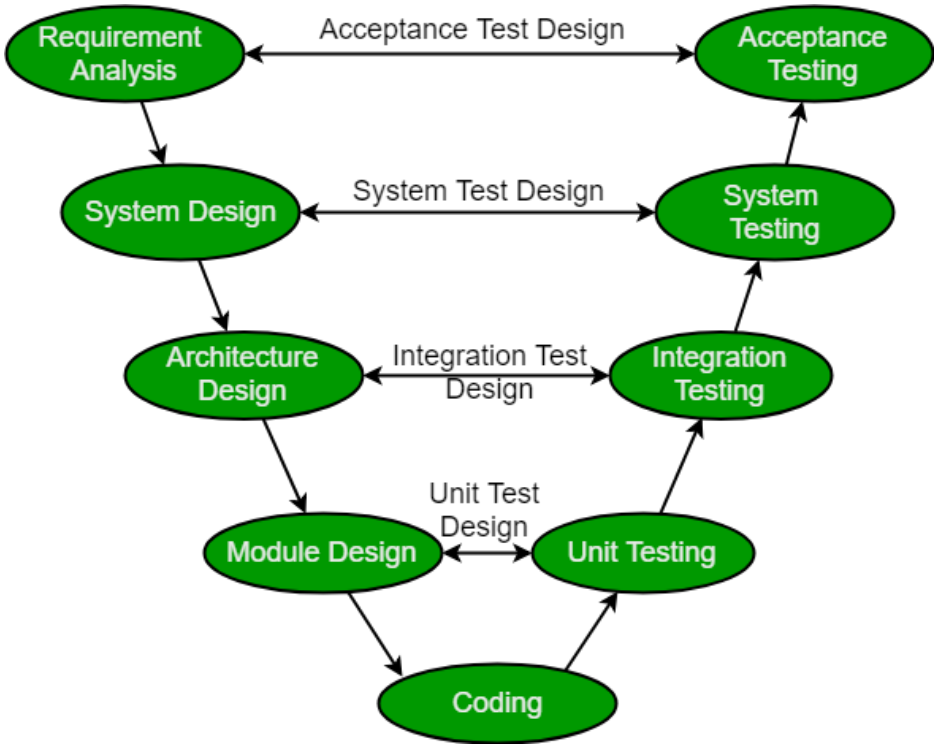
3.2.6.4. V model

Ve V modelu procesy probíhají sekvenčně, tudíž jeden proces začne až po dokončení předchozího, ve tvaru V. Základem pro tento model je testování, které se provádí každou fázi. Související termíny s V modelem jsou verifikace a validace. Verifikace konkrétně zahrnuje statickou analýzu provedenou bez spuštění kódu. Zjišťuje, zdali vývoj splňuje stanovené požadavky. Validace zas zahrnuje dynamickou analýzu za použití kódu. Provádí se na konci, aby bylo zjištěno, jestli bylo splněno očekávaní a požadavky zákazníků. [12]

Prvním krokem je analýza požadavků a následně se vytváří testovací plán, jež má za úkol zjistit, zda nahromaděné informace platí. Následně se stanoví design systému opět

s příslušným testovacím plánem. Podobně se navrhne architektura IS neboli High Level Design. Testovací plán se zaměřuje na jednotlivé díly a jejich integraci. Poté přichází čas na návrh modulů (Low Level Design), testují se jednotky, aby se snížilo maximum chyb. Když je hotova verifikace, vybere se patřičný programovací jazyk a začíná kódovací část následována validací, která v sobě zahrnuje testování kódu, integraci systému a testování IS v prostředí zákazníka. [12]

Obrázek 9: V model



zdroj: https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/

Výhodami tohoto modelu je jednoduchost. Hodí se pro menší projekty. Naopak pro dlouho trvající a komplexně orientované projekty se naprosto nehodí. Nedokáže reagovat na změnu funkčnosti zadanou zákazníkem. [13]

3.2.6.5. Agilní model

Technologie v současné době postupuje rychleji než kdy jindy a vynucuje globální softwarové společnosti, aby pracovaly v rychle se měnícím prostředí. Agilní model si tak bere ponaučení z vodopádových modelů a snaží se vymýtit nedostatek flexibility. Zároveň k jednotlivému projektu přistupuje tak, aby se co nejlépe přizpůsobil jeho požadavkům. [12]

Principy agilního modelu:

- Nejvyšší prioritou je uspokojit zákazníka
- Flexibilita
- Časté doručování SW
- Zaměstnávat motivované jedince, dodávat potřebnou podporu a důvěru
- Fungující SW je primárním měřítkem pokroku
- Jednoduchost je nutnost maximalizovat množství nevykonané práce.
- Nejlepším způsobem přenosu informací v rámci vývojového týmu je osobní rozhovor. [12]

Agilní model se zabývá v každé iteraci plánováním, analyzováním, návrhem, kódováním a testováním. Na konci každé iterace je doručen fungující SW zákazníkovi. Výhodami tohoto modelu je úzká spolupráce se zákazníkem, kterému informační systém doslova roste pod rukama. Lépe si tak dokáže představit funkcionality IS a lépe přijde na možné nedostatky. V případě nalezení nedostatku nebo pokud zákazník navrhne požadavek na změnu, pro agilní model to není problém. Nevýhodou je nevhodnost modelu pro komplexně závislé projekty. Navíc je i dost složitý na interakci se zákazníkem. Většinou nejsou dostatečně zdokumentovány.

3.2.6.6. Prototypování

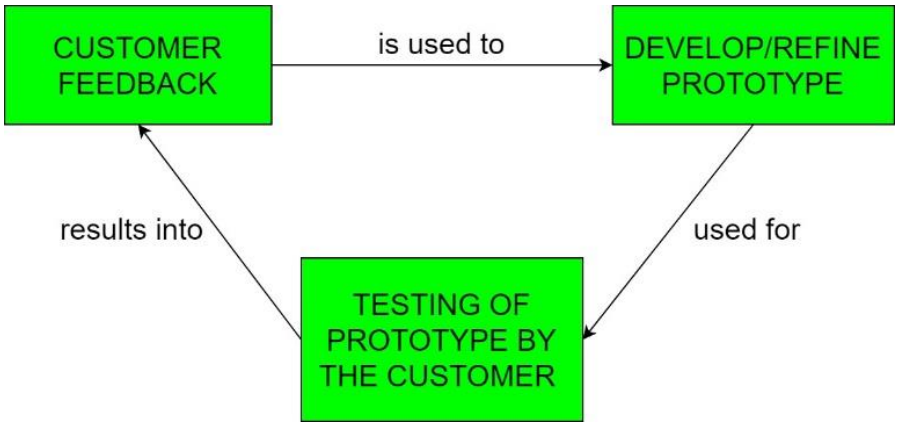
Prototypování je proces vývoje fungující repliky informačního systému, který musí být navržen. Poskytuje malou kopii konečného produktu, jež poukazuje na funkčnost produktu. Slouží tím k získání zpětné vazby od zákazníků, proto se stalo prototypování tak populární. Navíc je tento model vhodný pro zákazníky, jež předem neznají přesné požadavky na IS.

Průběh prototypování nejlépe vystihuje obrázek č. 10. Vše začíná rozhovorem se zákazníkem a sepsáním dokumentu obsahujícím informace ohledně uživatelského rozhraní, podle kterého je vytvořen první prototyp. Ten je předán zákazníkovi, jenž jej otestuje. Poté zákazník připraví feedback, jež je základem pro tvorbu dalšího prototypu. Čím detailnější feedback, tím bude další prototyp vhodněji vylepšený. Vše se opakuje, dokud se nedosáhne přijatelného prototypu vhodného pro základ vývoje konečného produktu.

Jsou nasnadě 2 přístupy prototypování. Zřejmě bude lepší tyto přístupy nepočešťovat. Prvním je Rapid Throwaway Prototyping. Vyvinutý prototyp v této metodě

nemusí být součástí konečně přijatého prototypu. Feedback zákazníků pomáhá při předcházení zbytečným konstrukčním poruchám, a proto je výsledný prototyp vylepšený. Druhým přístupem je Evolutionary Prototyping. Tato metoda je vesměs stejná s již popsanou výše. Ve srovnání s Rapid Throwaway Prototyping nabízí lepší přístup, který šetří čas i úsilí hlavně vývojářů, pro něž může být vývoj prototypu od začátku pro každou iteraci občas velmi frustrující.

Obrázek 10: Prototypování



zdroj: https://www.geeksforgeeks.org/software-engineering-prototyping-model/

Výhodami tohoto modelu jsou rozhodně flexibilita, možnost časně detekovat vady a zpětná vazba vedoucí k vylepšení řešení. Na druhou stranu se nabízí riziko nedostatečné analýzy a obava z nikdy nekončící tvorby prototypů.

3.2.6.7. Cleanroom model

Cleanroom model klade důraz na spolehlivost. Je pro něj důležité defektům předcházet než je odstraňovat. Principy tohoto modelu by se daly shrnout do 3 bodů. [14]

Snaží se o vývoj SW nástrojů založených na nějakém matematickém formalismu typu kontrola modelu, procesní algebra nebo třeba Petriho síť. Na specifikaci a návrhu IS se často podílí tzv. Box Structure Method. Tato metoda definuje tři funkční pohledy na systém, vytvářející abstrakční hierarchii, která umožňuje postupné zpřesňování a ověřování. První pohled je černá skříň, což je bezstavový popis vnějšího pohledu na systém. Druhý je stavová skříň, jež navazuje na černou skříň a přidává vnitřní stav. Třetím pohledem je čistá skříň, ta implementuje stavovou skříň. Každý pohled musí být úplný, konzistentní a vysledovatelný. Ověření, že návrh správně implementuje specifikaci, se provádí prostřednictvím kontroly týmů. [14]

Cleanroom model využívá iterativní přístup. Tudíž je IS vytvářen v krocích, jež postupně zvyšují implementované funkce. Existují předem stanovené standardy, které jsou měřítkem kvality každého přírůstku. Lze tak ověřit, že proces probíhá správně. Pakliže dojde k nedodržení standardů, ukončí se testování aktuálního inkrementu a opět se zavádí fáze návrhu. Testování se provádí čistě jako experiment. Otestuje se podmnožina vstupních a výstupních trajektorií, která je pak statisticky analyzována, což slouží k odhadu spolehlivosti, při čemž poskytuje i úroveň důvěry v tento odhad. [14]

Výhodou modelu Cleanroom je kvalita výsledného IS a produktivita při jeho tvorbě. Kvůli své myšlence předcházet defektům, snižuje náklady životního cyklu. Avšak Cleanroom model je příliš teoretický, matematický a vyžaduje velmi intenzivní trénink. [14]

3.2.7. Metodika vývoje IS

„Doporučený souhrn etap, přístupů, zásad, postupů, pravidel, dokumentů, řízení, metod, technik a nástrojů pro tvůrce informačních systému, který pokrývá celý životní cyklus informačních systémů. Metodika by se měla vztahovat na všechny prvky informačního systému, na pracovníky, data, software, hardware, organizační procedury, ekonomické otázky spojené s vývojem a provozem systému, doporučené dokumenty, způsoby řízení v jednotlivých fázích životního cyklu systému.“ (Řepa, 1999) [15]

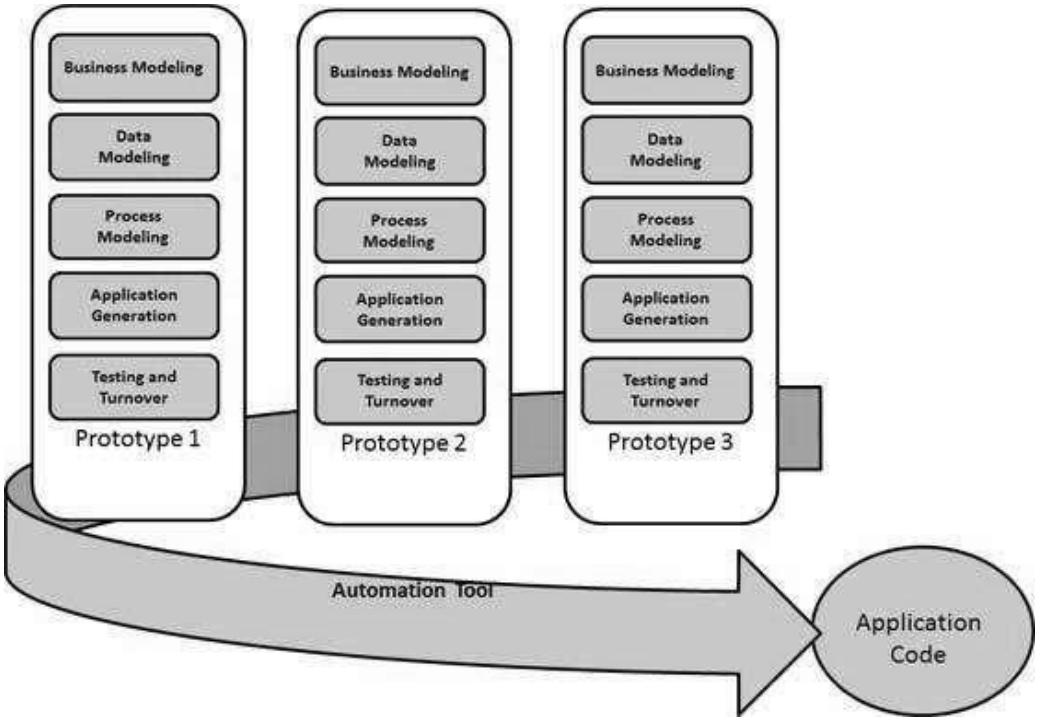
3.2.7.1. Rapid Application Development (RAD)

Hlavní myšlenkou RAD metodiky je prototypování a iterativní vývoj na úkor plánování. Prototypy jsou vytvářeny paralelně a jsou integrovány tak, aby se dal celý produkt doručit zákazníkovi, co nejrychleji. Jelikož neexistuje žádné podrobné plánování, je snadnější zakomponovat změny do vývoje. RAD využívá iterační a inkrementační model. Rapid Application Development dbá na znovupoužitelnost prototypů. [16]

Každý prototyp se skládá z 5 fází, které zahrnují analýzu, návrh, kódování a testování. Jako první je Byznys modelování (Business Modeling). Zde se pomocí dostupných informací vytváří byznysový model pro vývojový produkt. Vytváří se tu také kompletní byznys analýza. Po této fázi přichází na řadu Modelování dat (Data Modeling). Všechny informace z fáze Byznys modelování jsou přezkoumány, analyzovány a přetvořeny do datových objektů. Jejich atributy jsou identifikovány a doplněny. Stanoví se vztah mezi objekty a vztah k obchodnímu modelu. [16]

Třetí fází je Modelování procesů (Process Modeling), kde se všechny datové objekty z předchozího kroku převedou tak, aby vytvářely tok obchodních informací. Modelování procesů též definuje procesní model, popisuje procesy pro přidání, odstranění, načtení nebo úpravu datových objektů. Předposlední fází je Generování aplikací (Application Generation). Informační systém je vytvořen, převádí se procesní a datové modely na skutečné prototypy. Poslední fází je Testování a obrat (Testing and Turnover). Protože jsou jednotlivé prototypy nezávisle testovány již během iterace, snižuje se celková doba testování, ale také riziko závažných problémů. Je však nezbytné důkladně otestovat datový tok a rozhraní mezi všemi součástmi. [16]

Obrázek 11: Rapid Application Development



zdroj: https://www.tutorialspoint.com/sdlc/sdlc_rad_model.htm

Velkou výhodou RAD metodiky je tedy schopnost reagovat na změnu požadavků, podporuje zpětnou vazbu od zákazníka, využívá znovupoužitelnosti a dosti zkracuje vývojový čas. RAD se doporučuje pro škálovatelné systémy založené na komponentech. Naopak je metodika závislá na technicky silných členech týmech, vyžaduje totiž vysoce kvalifikované vývojáře a návrháře. Je nepoužitelná pro levnější projekty, protože náklady na modelování a automatizované generování kódu jsou dost vysoké. [16]

3.2.7.2. Adaptive Software Development (ASD)

Adaptive Software Development se vyvinul z praktik RAD. ASD poskytuje schopnost přizpůsobovat se změnám v turbulentním prostředí s IS vyvíjejícím se při nedostatečném plánování. Adaptive software development je cyklický a skládá se z fází, jež odrážejí nepředvídatelnost v komplexních systémech. Speculate, Collaborate a Learn. [15]

Speculate, nebo také spekulace, nahrazuje v ASD termín plán, který je příliš deterministický a naznačuje relativně vysoký stupeň jistoty ohledně požadovaného výsledku. Spekulace na rozdíl od plánu uznává skutečnost nejistoty, podporuje výzkum a experimentování. [15]

Jelikož komplexní informační systémy se neustále vyvíjejí a aby bylo možné v těchto IS vyřešit jakýkoliv problém, je zapotřebí shromáždit velké množství informací, zanalyzovat je a poté aplikovat na problém, což si žádá týmovou spolupráci (collaborate). Ta vyžaduje schopnost společně pracovat na výsledcích, sdílet vědomosti nebo se rozhodovat. [15]

Nejdůležitější fází ASD je ale učení se (learn). Tým musí neustále zdokonalovat své znalosti pomocí různých praktik (technické hodnocení, projektová retrospektiva apod.) a učit se z chyb a neúspěchů. [15]

Výhodou Adaptive Software Development je rozhodně adaptivní přístup, který je schopen reagovat na měnící se požadavky klientů. Navíc přímá komunikace eliminuje prostor pro případné dohady v systému. Na druhou stranu se tato metodika zaměřuje více na vývoj IS než na dokumentaci, což vzývá problém do budoucna. [15]

3.2.7.3. Rational Unified Process (RUP)

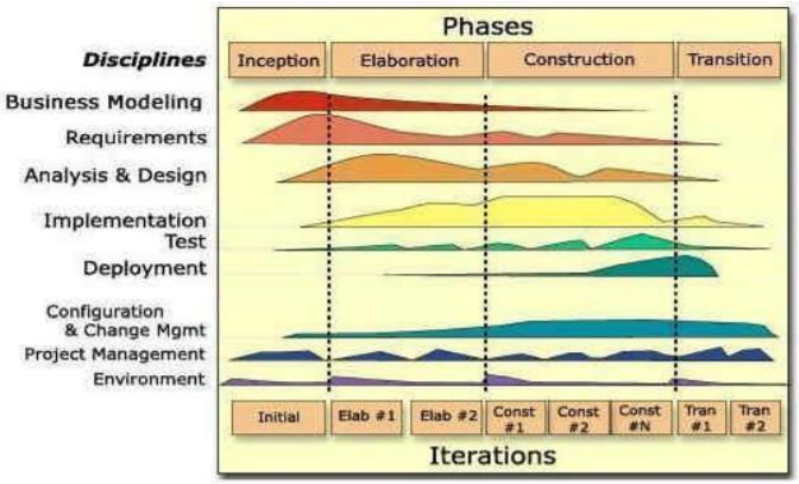
Metodika Rational Unified Process byla vytvořena firmou IBM. RUP je normativní, dobře definovaný proces vývoje systému, často používaný pro vývoj systémů založených na technologiích využívající objekty. [17]

Obrázek níže znázorňuje životní cyklus RUP. Vidíme, že se RUP skládá ze 4 fází a 9 disciplín. Horizontální výkyvy představují hrubý odhad relativního úsilí v průběhu všech čtyř fází. Je patrné tedy, že se očekává velké úsilí v první fázi Byznys modelovaní nebo třeba Nasazení (Deployment) IS nezačne dříve než po dokončení fáze Příprava (Elaboration) a hlavní práce na Nasazení nastává až v polovině fáze Konstrukce (Construction). [17]

Fáze:

- Zahájení (Inception) - Hlavními cíli této fáze je dosažení souhlasu zúčastněných stran ohledně cílů projektu a získání finančních prostředků. Vytváří se zde důkladně zpracovaný model požadavků, který vymezí rozsah projektů. Začíná se instalovat pracovní prostředí. Důkladně se připravuje i plán pro pokračování projektu. Na konci této fáze musí být všechny informace o rozsahu projektu, počátečních požadavcích, realistickém pohledu na IS, možných rizicích apod. sesumírovány.
- Příprava (Elaboration) – Během této fáze se upřesňují požadavky, ověřuje se architektura systému. Požadavky jsou podrobně rozepsány za účelem pochopení architektonických rizik a pro zajištění pochopení rozsahu každého požadavku pro následné plánování. Na konci Přípravy by měla existovat už realistická vize projektu, stabilní architektura, seznam detailních požadavků, podrobné iterační plány pro následující iterace. Rizika jsou nadále řízena.
- Konstrukce (Construction) – Konstrukce se zaměřuje na vyvinutí systému do bodu, kdy už bude připraven na nasazení. Důraz se nyní klade na upřednostnění požadavků a doplnění jejich specifikace, jejich analýzu, návrh řešení, kódování a testování IS. Nemělo by se pokračovat v další fázi, pokud není IS přijatelný pro nasazení a pokud nejsou zainteresované strany připravené pro nasazení. Rizika jsou pořád řízena, jsou připraveny další iterační plány pro navazující iterace.
- Předání (Transition) – V poslední fázi se předává IS spolu s dokumentací. Provádí se testování systémovými testery a koncovými uživateli. Proškolují se koncoví uživatelé, pracovníci podpory a obsluhy. [17]

Obrázek 12: Rational Unified Process



zdroj: Manager's Introduction to The Rational United Process, 2005

Disciplíny:

- Byznys modelování (Business Modeling) se snaží posoudit aktuální stav organizace, zkoumá obchodní procesy, role a odpovědnosti, identifikuje a vyhodnocuje potenciální strategie podnikových procesů.
- Požadavky (Requirements) mají za cíl získat, zdokumentovat a dohodnout se na rozsahu toho, co je a co nemá být postaveno. Tyto informace používají analytici, konstruktéři a programátoři k vybudování systému, testeři k ověření systému a manažer projektu k plánování a řízení projektu.
- Analýza a návrh (Analysis and Design) zanalyzuje požadavky na systém a navrhne řešení (komponenty, služby, uživatelské rozhraní, databázi atd.), které je v souladu s požadavky a omezeními.
- Implementace (Implementation) se zabývá transformací návrhu na spustitelný kód a provedení základní úrovně testování, zejména jednotkové testování.
- Testování se využívá za účelem získání objektivního hodnocení. Patří sem také zjištění závad, ověření správného fungování systému a splnění požadavků.
- Nasazení (Deployment) plánuje strategii nasazení systému a provádí plán pro dodání IS koncovým uživatelům. Školení těchto uživatelů se uskutečňuje v této disciplíně.
- Správa konfigurace a změn (Configuration and Change Management) sleduje verze v čase, kontroluje je a spravuje změny.
- Projektový management (Project Management) se zabývá řízením rizik a koordinací zaměstnanců tak, aby bylo zajištěno, že je IS doručen včas a projekt při tom nepřekročí rozpočet.
- Prostředí (Enviroment) se snaží zajistit pro tým správné procesy, pokyny a nástroje (hardware, software atd.) [17]

Výhodou této metodiky je důraz na přesnou dokumentaci, proaktivně řeší rizika spojená s požadavky zákazníka. Integrace není náročná z časového hlediska, protože probíhá po celou dobu životnosti IS. Díky opětovnému použití součástí je i potřebná doba vývoje nižší. Nevýhody jsou takové, že pro vyvinutí IS podle této metodiky je potřeba mít v týmech odborníky ve svém oboru. Vývojový proces je příliš složitý. U projektů využívající nové technologie není možné opětovné použití součástí a u některých obzvláště velkých projektů je velmi těžké organizovat všechny vývojové proudy, což může způsobit problémy při testování. [17]

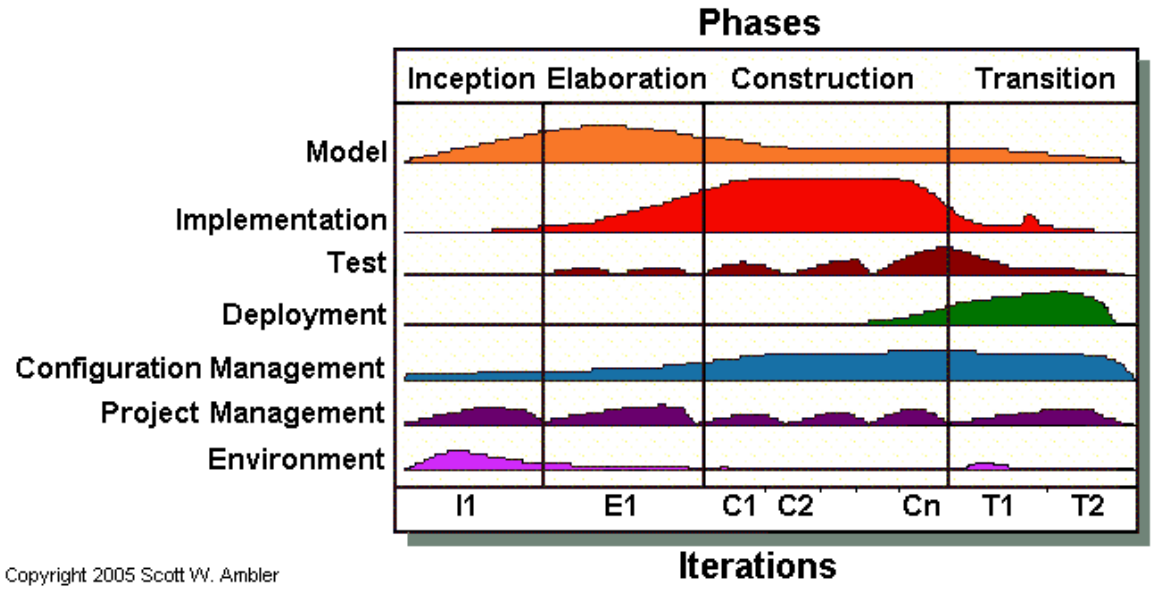
3.2.7.4. Agile Unified Process (AUP)

Agile Unified Process je vlastně zjednodušená verze RUP. Popisuje jednoduchý a snadno srozumitelný přístup vývoje IS s použitím agilních technik a konceptů, které čerpají právě z RUP. [18]

Na obrázku níže lze vidět v porovnání s RUP, že zmizely části Byznys modelování (Business modeling), Požadavky (Requirements), Analýza a návrh (Analysis a Design). Tyto všechny položky jsou teď skryté pod disciplínou model. Také zmizela disciplína Správa konfigurace a změn (Configuration and Change Management), jež byla nahrazena Správou konfigurace (Configuration Management). [18]

Stejně jako RUP má AUP 4 fáze: Zahájení (Inception), Příprava (Elaboration), Konstrukce (Construction) a Předání (Transition). Počet disciplín se sice zvýšil, ale stále se provádějí iterativně a definují činnosti, které členové vývojového týmu provádějí při vytváření, ověřování a dodávání IS. [18]

Obrázek 13: Agile Unified Process



zdroj: http://www.ambysoft.com/unifiedprocess/agileUP.html

Nemá důvod popisovat opět všechny disciplíny, neboť jejich funkce byly popsány výše v kapitole Rational Unified Process, proto se zaměříme pouze na nové disciplíny.

- Model se snaží pochopit podnikání organizace, problémové oblasti projektu a identifikovat vhodné řešení problémové oblasti.
- Implementace (Implementation)

- Test
- Nasazení (Deployment)
- Správa konfigurace (Configuration Management) spravuje přístup (kontrolu a správu změn) k důležitým položkám projektu.
- Projektový management (Project Management)
- Prostředí (Enviroment) [18]

AUP se snaží využít to nejlepší z RUP a za pomoci větší agility být ještě lepší metodikou. Důležité je i zjednodušení složitého předchůdce. [18]

3.2.7.5. Dynamic Systems Development Method (DSDM)

„DSDM popisuje řízení projektů, odhadování, prototypování, časový rozmezí, správu konfigurace, testování, zajištění kvality, role a odpovědnost uživatelů a pracovníků IT, týmové struktury, prostředí nástrojů, řízení rizik, prostředí pro údržbu, opětovné použití a vztahy mezi prodejcem a nákupcem – vše v prostředí RAD“ (Stapleton, 1997) [19]

Jak již bylo řečeno v citaci, DSDM vychází z metodiky RAD. Jedná se o iterativní a přírůstkový přístup, který zahrnuje principy agilního vývoje včetně trvalého zapojení uživatelů a zákazníků. [15]

Skládá se z 5 částí. První z nich je Studie proveditelnosti. V této fázi se uvažuje nad možností aplikovat navrhovanou metodu, navíc se provádí důkladný výzkum s cílem nalezení existujících problémů. Následuje Byznys studie, kde se uceluje jasná představa byznys flow a jak jsou procesy vzájemně propojeny. Poté se pracuje na vylepšení požadavků na high level byznysové informace a na funkcích systému, které byly identifikovány při byznys studiu metodiky, proto aby mohl být vytvořen prototyp. Tato fáze se nazývá Iterace funkčního modelu. Dále se v ní identifikují rizika a vytváří se plán, jak je řešit. V předposlední fázi Návrh a tvorba systému už vzniká skutečný IS. Implementace, jak se jmenuje poslední fáze, v sobě skrývá přesun již vytvořeného informačního systému na produkci. [15]

DSDM dbá na dodržování časů a rozpočtu, je flexibilní při přijímání požadavků. Lpí na testování a řadí zákazníka na nejvyšší místo. Nevýhodou DSDM je to, že vyžaduje významné a trvalé zapojení uživatelů. Náročně se vede dokumentace a pro úspěšné dokončení projektu je zapotřebí mít zkušený vývojový tým. [15]

3.2.7.6. Extreme Programming (XP)

Jedním z nejpodstatnějších agilních metodik pro vývoj IS je právě extrémní programování. Využívá se ke zlepšení kvality IS a zároveň ke splnění požadavků zákazníka. Tato metodika doporučuje, aby se osvědčené postupy z minulosti promítly na extrémní úroveň. Je vhodná pro malé projekty s málo početnými týmy, protože je zde snadnější setkání tváří v tvář nebo pro projekty s měnícím se nebo ne zcela jasným zadáním. [15]

Pro Extreme Programming jsou typické některé praktiky. Jednou z nich je tzv. párové programování. Vývojáři pracují ve dvojici, která se čas od času mění. Ti se u kódu střídají, debatují nad ním a hledají si navzájem chyby. Aby se předešlo v XP chybám, vytváří se testovací kódy ještě před psaním kódu. Interakce se zákazníkem je prováděna během každé iterace, což je otázka několika dní. Extreme Programming si zakládá na jednoduchosti a jasném kódu. Pro docílení lepší přehlednosti kódu se provádí refaktorizace, což je přepis kódu bez změny vnějšího chování na čistší a čitelnější kód. Programuje se pouze to, co je právě potřeba. [15]

Metodika XP šetří náklady a čas potřebné pro realizaci projektu. Zajišťuje včasné dodání. Díky jednoduchému kódu se i lépe upravuje, což opět šetří čas. Velkým plusem je i zpětná vazba. Extreme Programming má i nevýhody. Většinou se programátoři soustřeďují na kód a design jde stranou. Dokumentace je často nedostatečná, to komplikuje budoucí manipulaci s kódem. Navíc XP není vhodný pro geograficky oddělené programátory. [15]

3.2.7.7. Feature-Driven Development (FDD)

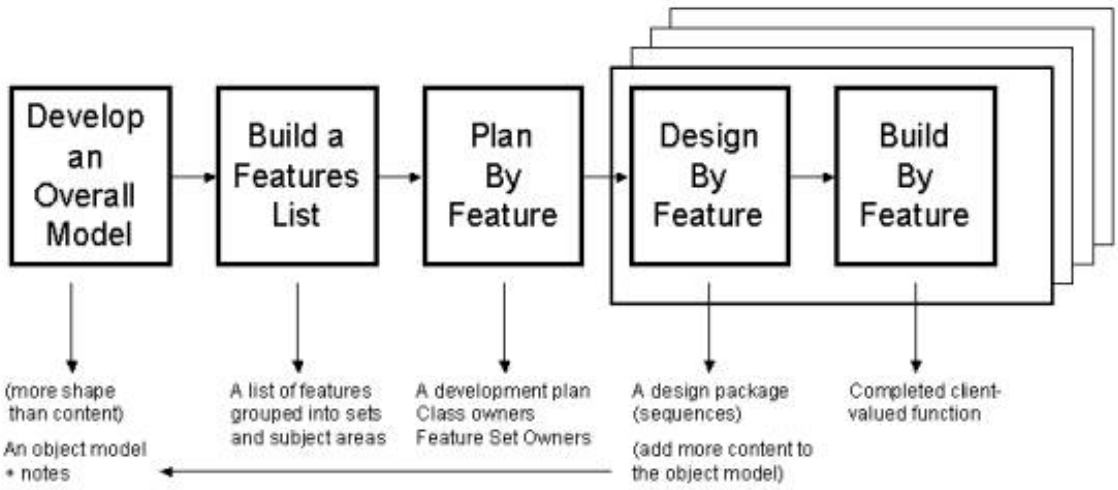
Metodika Feature-Driven Development si zakládá na iterativním vývoji s krátkými iteracemi založenými na modelovaní. Zdůrazňuje kvalitu v každém kroku. [15][20]

FDD je založen na několika principech. Prvním z nich je Doménové objektové modelování (Domain Object Modeling). To je tvořeno diagramem tříd UML, jež obsahuje nejdůležitější typy objektů v problémové doméně a vztahy mezi nimi. Vytvoření doménového objektového modelu má předejít vzniku nekonzistencí a sporům mezi členy týmu. Vývoj podle Užitných vlastností (Developing by Feature) doporučuje FDD přijímat pouze funkční požadavky, které mají hodnotu pro zákazníka. Hlavním důvodem pro tento pohled je časté zaplnění funkční specifikace velkým množstvím nedůležitých funkcí, což hodně ztěžuje práci vývojáře. Princip Vlastnictví tříd (Individual Class Ownership) jasně určujte osoby nebo role zodpovědné za obsah dané třídy. Vlastník této třídy lépe zná tuto část kódu a je schopen implementovat změny rychleji. Vznikají Týmy pro užitné vlastnosti

(Feature Teams). Vlastníkům tříd je přidělena užitná funkcionalita. Tuto funkcionalitu musí implementovat a jelikož implementace vyžaduje spolupráci několika vlastníků tříd, vznikají dočasné týmy. Inspekce (Inspections) zajišťuje pomocí různých forem kontrol a inspekcí vysokou kvalitu návrhu a kódu. Neposledním principem jsou Pravidelné buildy (Regular Builds). Pravidelně se integruje kód všech hotových užitných vlastností do jednoho celku, což dovoluje včas najít integrační problémy a předvést část IS zákazníkovi. Řízení konfigurací (Configuration Management) má na starost verzování jak dokumentace, tak i třeba testovacích případů apod. Posledním důležitým principem je Reporting/Viditelnost výsledků (Reporting/Visibility of Results). FDD nabízí jednoduchou metodu sběru přesných informací o stavu projektu a nabízí několik formátů zpráv o stavu projektu. [15][20]

FDD vyvíjí software ve formě procesů viz obrázek níže. Cílem je však vytvoření fungujícího SW pro zákazníka a nikoliv splnění všech procesů. Během procesu Vypracování celkového modelu (Develop an Overall Model) je vytvořen hrubý model celé domény. V čele je architekt, pod nímž pracují malé specializované týmy vývojářů a ostatních pracovníků. Prezentují se představy o systému a jeho kontext, vytváří se kostra modelu. Výsledky jsou integrovány, výstupem tohoto procesu jsou diagramy tříd a alternativy řešení. Poté přichází na řadu Sestavení seznamu užitných vlastností (Build a Features List), kde se

Obrázek 14: FDD Procesy



zdroj: https://project-management.com/xp-fdd-dsdm-and-crystal-methods-of-agile-development/

rozkládá problémová doména se zaměřením na hodnotu pro zákazníka. Výsledkem je formální, kategorizovaný seznam užitných vlastností. Když jsou užitné vlastnosti sepsány, je na řadě Plánování užitné vlastnosti (Plan By Feature). Výchozím dokumentem je právě seznam užitných vlastností, podle kterého probíhá plánování i realizace. Vedoucí projektu,

vedoucí vývoje a hlavní programátoři vytvoří tým, jež plánuje pořadí vývoje užitných vlastností. Ke každé užitné vlastnosti je přiřazen hlavní programátor, ten má za úkol určit termín dokončení. [15][20]

Čtvrtý proces je Návrh užitné vlastnosti (Design By Feature) a řídí ho hlavní programátor. Pro danou užitnou vlastnost vypracuje návrhový balíček obsahující detailní sekvenční diagram, rozšíření tříd a metod v diagramu tříd a opět alternativy návrhu. Posledním procesem je Realizace užitné vlastnosti (Build By Feature), kde každý vlastník třídy realizuje metody, pro danou třídu vytváří testovací případy a provádí jednotkové testy. Po otestování je třída vložena do systému pro správu verzí. [15][20]

Jelikož FDD využívá 5 jednoduchých procesů, snižuje dobu celkové práce. Navíc je založen na standardech pro vývoj SW, což pomáhá snadnějšímu vývoji. FDD transparentní, a tak lze dobře sledovat pokrok. Výhodou je i podpora paralelní práce týmů. Feature-Driven Development se nedoporučuje pro menší projekty. Nevýhodou je vysoká závislost na hlavním vývojáři, nedostatečná nebo úplně chybějící písemná dokumentace, nesdílnost kódů a iterace není definována tak dobře jako u jiných agilních metodologií. [15][20]

3.2.7.8. Lean Development

Lean Development je založen na konceptu dynamické stability. Typická je pro tuto metodiku snaha rychle se přizpůsobit a efektivně reagovat na požadavky. Zároveň dbá na vnitřní procesy, aby byly co nejstabilnější a neustále se zlepšovaly. Cílem Lean Development je vytvoření SW, jež ušetří třetinu lidské práce, třetinu času a třetinu investic. Lean Development se zaměřuje spíše na strategickou než taktickou úroveň. Stejně jako Scrum klade důraz na řízení vývoje SW před softwarově inženýrskou oblastí. [15]

Více než o metodiku se ale jedná o souhrn pravidel:

- Odstranění zbytečného – Pravidlo se snaží odstranit vše, co nemá vliv na konečný produkt. Lze si tak představit vše od dokumentů až po diagramy, které vznikají při vývoji SW, ale nejsou nutné pro finalizaci produktu.
- Minimalizace zásob – Zásobou chápejme dokumentaci, jež není součástí pro finalizaci produktu. Minimalizace zásob je snaha snížit 100 stran specifikace na 10 stran pravidel.
- Maximalizace toku – Pokud je nutné snížit dobu vývoje, pak je nutné snížit práci na procesu během iterace.

- Vývoj tažený poptávkou – Toto pravidlo klade důraz na možnosti rozhodovat, co nejpozději. Konkurenční výhodou v měnícím se prostředí je jednoznačně schopnost přizpůsobit dodávku SW požadavkům uživatelů. Ti ale nedokážou definovat soudobé, natož budoucí potřeby. Pokud je tedy návrh zahrnut již na začátku životního cyklu, pak existuje velká šance konfliktu s požadavky.
- Pracovníci s rozhodovací pravomocí – Vývojáři musí vědět, co a do jaké doby musejí udělat a musí mít možnost rozhodovat.
- Uspokojení požadavků zákazníků – Dříve byla nedostatečná specifikace požadavků zákazníka hlavním důvodem neúspěchu. Dnes je již vedena detailní specifikace. Zákazníci, ale nedokážou předem určit všechny potřeby, a tudíž je nutné specifikovat požadavky i v budoucnu.
- Zavedení zpětné vazby – Pokud nelze identifikovat všechny požadavky hned na začátku projektu, pak je zapotřebí je postupně doplnit za pomoci zpětné vazby.
- Odstranění lokálních optimalizací – Jelikož se vše neustále mění, nemá smysl optimalizovat stávající řešení.
- Partnerství s dodavateli – Pro zrychlení a vylepšení kvality je důležité partnerství s dodavateli.
- Zavedení kultury pro neustále zlepšování – Toto pravidlo vytváří motivaci zlepšovat procesy při vývoji SW. [15]

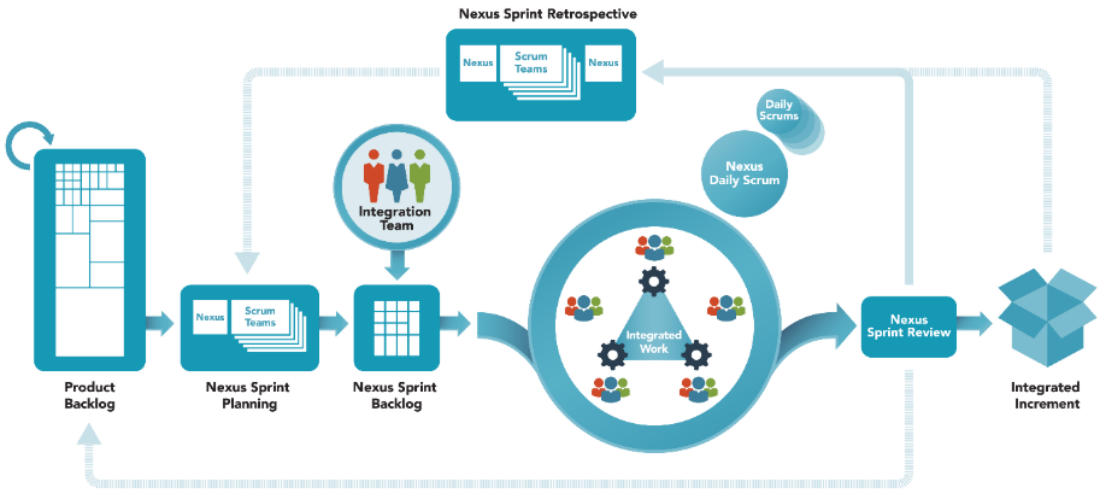
Odstraněním zbytečného se zrychluje proces vývoje SW, jež má za důsledek snížení času, nákladů a umožňuje předčasné dodání produktu. Lean Development je vysoce závislý na soudržnosti týmu, což může být občas problém. Úspěch závisí na disciplinovanosti členů a jejich technický dovednostech. Lean Development občas nereaguje včas na nové požadavky zákazníka. [15]

3.2.7.9. Scrum

Scrum je navržen tak, aby dokázal reagovat na měnící se požadavky a vylepšit produktivitu procesu vývoje SW. Každý den přezkoumává realizované požadavky a určuje další kroky. Využívá maximální týmovou spolupráci a častou komunikaci se zákazníkem. Scrum je ideální pro menší týmy o několika členech, ale zvládá zapojit do projektu i více týmů. [15][21]

Scrum se skládá z Vlastníka produktu (Product Owner), Scrum mastera a Vývojového týmu (Team). Vlastník produktu by měl mít jistou vizi, autoritu a být neustále dostupný. Zároveň je zodpovědný za zadání projektu, za rozhodnutí o rozsahu, rozpočtu a termínu projektu. Průběžně komunikuje s Vývojovým týmem ohledně vize a prioritách. Scrum master je jakýsi prostředník mezi Vlastníkem produktu a Vývojovým týmem, pomáhá jim ve vzájemné komunikaci a dbá na dodržování pravidel Scrumu. Scrum Master pracuje na odstranění překážek, které braní Vývojovému týmu v dosažení cílů během sprintu, motivuje je a snaží se je nasměrovat k samostatnému řízení. Vývojový tým je zodpovědný za organizaci sebe sama k dokončení práce. Měl by obsahovat tři až devět členů od softwarových inženýrů, přes architekty, programátory, analytiky až po UI návrháře a testery. Ideálně by měli všichni členi sedět v jedné místnosti. Tým má zodpovědnost za splnění cílů během sprintů. [15][21]

Obrázek 15: Scrum metodika



zdroj: https://www.scrum.org/resources/online-nexus-guide

Scrum začíná vytvořením dokumentu s názvem Product Backlog (dále jen PB dokument). Ten obsahuje seznám úkolů k provedení, zákazník jednotlivé úkoly ohodnotí prioritou. Poté přichází řada na plánování Sprintu (Sprint Planning). Vývojový tým na základě PB dokumentu vybírá úkoly s nejvyšší prioritou a plánuje, jak budou postupovat, aby úkol vyřešili. Sprint je krátká doba určená pro dokončení úkolu. Většinou trvá dva až čtyři týdny a skládá se z mnoha schůzek. Plánovací schůzka (Sprint Planning) za přítomnosti všech členů Scrumu řeší, jaký přírůstek vznikne a kolik bude stát práce. Denní schůzka (Daily Scrum) zahrnuje pouze Vývojový tým, který odhaduje plán pro následující den. Vyhodnocení sprintu (Sprint Review) se účastní všechny zúčastněné strany. Cílem je

přezkoumat přírůstek a případně upravit dokument PB. Posledním typem schůzky je Retrospektiva Sprintu (Sprint Retrospective). Během této schůzky se hodnotí, jak Scrum tým fungoval a kde se může pro další iteraci zlepšit. Výsledkem Sprintu je demo vzniklých úprav, které je předvedeno zákazníkovi. Ten poskytne zpětnou vazbu, což umožňuje rychle reagovat na změny v požadavcích. Po několika Sprintech, kdy jsou domluvené cíle úspěšně splněny, je produkt dokončen. [15][21]

Scrum zajišťuje efektivní využití času a peněz. Velké projekty jsou rozděleny do snadno ovladatelných Sprintů. Scrum dobře funguje i pro rychle rozvíjející se projekty. Krátké Sprinty umožňují změny na základě zpětné vazby mnohem snadněji a během každodenních setkání je vidět individuální práce každého člena Vývojového týmu. Nevýhodou Scrumu může být nedostatečně zvolený konec projektu. Všichni zakomponovaní jednotlivci musí být odhodlaní a schopni pracovat v týmech. Scrum se nedoporučuje pro velké týmy. Denní schůzky se mohou časem stát frustrující. [15][21]

3.2.7.10. Crystal

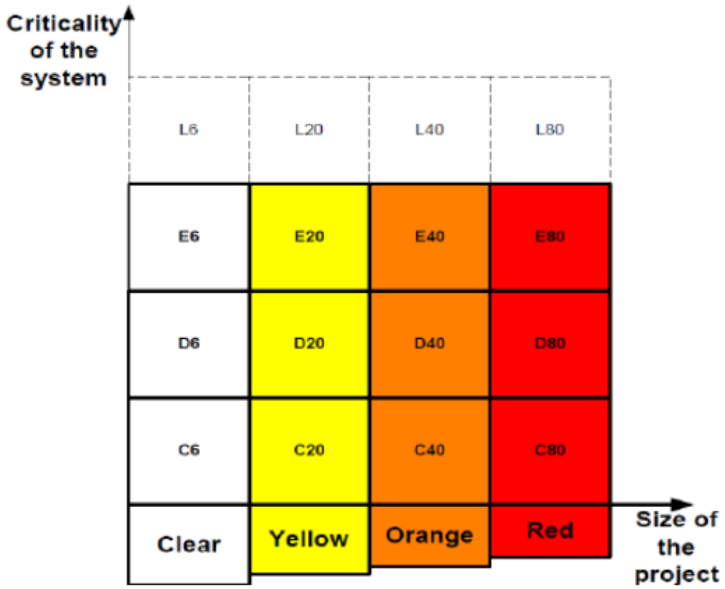
Crystal metodika se zaměřuje spíše na spolupráci a komunikaci lidí než na proces. IS je vyvíjen inkrementálně a jedna iterace by neměla překročit 4 měsíce. Existují celkem 4 typy Crystal metodiky, od čisté barvy až po červenou. Crystal Clear se využívá pro malé projekty, Crystal Yellow pro střední, Crystal Orange pro velké a Crystal Red pro velmi velké projekty. Podle velikosti projektu, složitosti, kritičnosti, úrovně dovedností, dostupných technologií až po velikost týmu se vybírá nejvhodnější metodika. Každá z nich má své vlastní praktiky, role a techniky, avšak běžná pravidla, stejně tak snaha úspěšně doručit produkt, jsou u všech metodik stejná. Avšak pouze Crystal Clear a Crystal Orange jsou definovány a používány. [15]

Na obrázku níže písmenka C, D, E a L zobrazují potenciální ztráty v důsledku selhání systému. C reprezentuje komfort, D diskrétní příjmy, E nezbytné příjmy a L představuje život. Kritičnost systému znázorňuje osa Y, velikost projektu osa X. E80 tedy udává velmi kritický projekt, pro nějž bude potřeba 80 lidí. [15]

Crystal Clear metodika je navržena pro malé projekty s šestičlenným týmem (D6). Projektový tým musí být kvůli komunikaci situován na stejném místě. Členové týmu jsou sponzor, obchodní expert, senior designer, senior programátor, tester a člověk píšící dokumenty. Každodenně diskutují o požadavcích projektu, návrhu, nástrojích a prioritách. SW je inkrementálně rozvíjen v iteracích o délce dva až tři měsíce. Jsou stanoveny standardy pro kódy, regresní testování a uživatelské rozhraní. Průběh projektu se měří pomocí SW

nebo dokončením důležitých milníků. V dokumentaci se nachází plán vypouštění verzí SW, use case, objektové modely, uživatelské manuály, testovací scénáře a návrhové výkresy. [15]

Obrázek 16: Crystal metodika



zdroj: http://www.wiki.amachu.in/index.php?title=Crystal_family

Crystal Orange se používá pro středně velké projekty s velikostí dvacet až čtyřicet členů (D40). Z pravidla na těchto projektech pracuje více než jeden tým. Součástí týmů je projektant uživatelského rozhraní, technický zprostředkovatel, návrhář databáze, obchodní analytik, analytik využití, tester a zapisovatel. Projekt trvá jeden až dva roky. Produkt je opět dodáván inkrementálně, přičemž jedna iterace trvá od tří do čtyř měsíců. Na konci každé iterace se provádí schůzka pro vyladění chyb. Na rozdíl od Crystal Clear používá Crystal Orange dokument požadavků a ještě navíc podrobný plán návrhu. [15]

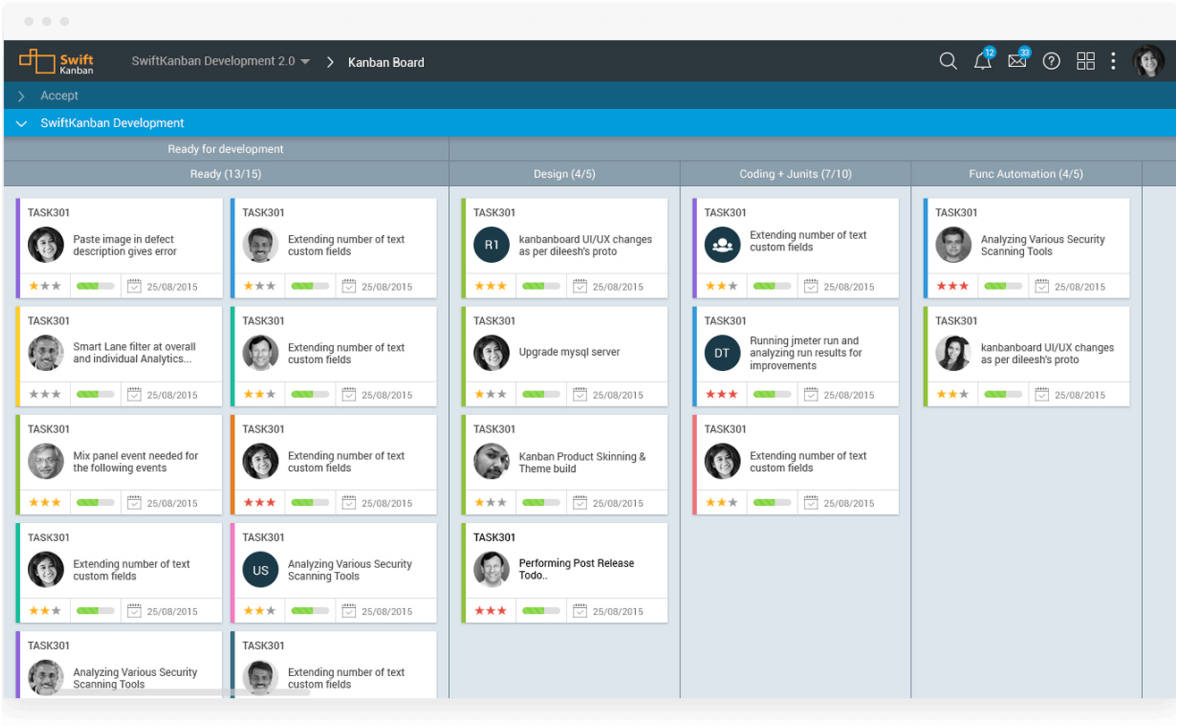
Výhodou Crystal metodik je efektní týmová spolupráce. Na různé projekty lze použít různé metody. Poskytuje technické postupy a dobrou kontrolu rizik. Nevýhodou je určitě nadefinování pouze dvou metod. Ty navíc nedostatečně ověřují návrh a kódování. Jelikož chybí validace systému, stává se nevhodným pro vývoj životně důležitých systémů. [15]

3.2.7.11. Kanban

Kanban je metodika správy týmů tak, aby týmy vizualizovaly svojí práci, identifikovaly, odstranily překážky a dramaticky se zlepšily v ohledu výkon a kvalita. Charakteristikou Kanbanu je několik prvků. Prvním z nich je Vizualizace workflow (Visualize Workflow). Tento proces představuje rozdělení na definované segmenty, které

vypadají jako pojmenované sloupce. Ty jsou umístěny na tzv. Kanban Board. Na kartičky se napíše každá položka SW a zařadí se do sloupce, vzniká workflow. Jednotlivé procesy jsou ohodnoceny To Do (tzn. Potřeba udělat), In Progress (tzn. Probíhající), Done (tzn. Hotové). [22]

Obrázek 17: Kanban board



zdroj: https://www.digite.com/kanban/kanban-board/

Kritériem řízení Kanbanu jsou WIP, neboli Work In Progress, což lze do češtiny přeložit jakožto „probíhající práce“. Správou a sledováním WIP lze optimalizovat tok pracovních položek. [22]

Snižuje se tím čekací doba, závislosti dokončení úkolů a identifikují se úzká místa, jež jsou následně řešena. Kanban si také velmi zakládá na odstraňování odpadu (zastaralé dokumentace, nepoužitelná částečně dokonaná práce, extra procesy navíc). Užitečné pro týmy, které se potýkají s vysokou mírou nejistoty a variability. Za nevýhodu lze brát samotný Kanban Board. Nesprávné používání desky nebo dokonce její zastaralá verze může vést k problémům při vývoji. Nevýhodou je i nedostatečné načasování, protože neexistuje žádný časový rámec pro jednotlivé fáze. [22]

3.3. Porovnání webových frameworků

Pro porovnání byly vybrány tyto frameworky: Django, Ruby on Rails a Express.js. Než dojde k porovnání jednotlivých nástrojů je nasnadě porovnat rychlost programovacích jazyků, jež jsou frameworky využívány. Node.js vlastně ani není programovací jazyk jako spíš prostředí pro vytváření aplikací na straně serveru pomocí JavaScriptu. Pro porovnání jej ale použijeme. Z obrázku níže je jasné, že jasně vede Node.js a Python s Ruby jsou na tom více méně stejně, nicméně na Node.js koukají z uctivé vzdálenosti.

Obrázek 18: Rychlostní test

n-body

source	secs	mem	gz	cpu	cpu load
Node js	**26.31**	33,476	1297	26.30	0% 0% 1% 100%
Python 3	850.24	7,916	1242	850.06	0% 0% 0% 100%
Ruby	387.73	16,196	1137	433.22	48% 4% 10% 51%

binary-trees

source	secs	mem	gz	cpu	cpu load
Node js	**46.28**	653,316	434	93.74	46% 60% 50% 49%
Python 3	83.95	451,732	589	290.57	88% 87% 87% 97%
Ruby	64.07	488,156	1107	218.34	98% 78% 90% 83%

mandelbrot

source	secs	mem	gz	cpu	cpu load
Node js	**18.21**	605,668	748	65.22	84% 82% 96% 97%
Python 3	263.04	51,780	688	1,050.80	100% 100% 100% 100%
Ruby	452.81	59,660	954	1,805.81	100% 100% 100% 100%

regex-redux

source	secs	mem	gz	cpu	cpu load
Node js	**12.05**	854,584	408	12.87	29% 47% 21% 12%
Python 3	16.98	445,760	512	31.44	28% 74% 47% 38%
Ruby	23.58	269,664	751	25.12	79% 68% 74% 85%

spectral-norm

source	secs	mem	gz	cpu	cpu load
Node js	**15.81**	31,780	381	15.81	2% 100% 1% 2%
Python 3	182.12	52,752	443	705.87	96% 96% 96% 100%
Ruby	237.96	18,312	326	280.55	97% 4% 6% 12%

k-nucleotide

source	secs	mem	gz	cpu	cpu load
Node js	**64.56**	1,819,728	935	137.90	69% 74% 92% 76%
Python 3	72.80	189,720	1967	276.09	95% 96% 98% 94%
Ruby	189.81	137,920	667	714.29	94% 97% 96% 91%

fasta

source	secs	mem	gz	cpu	cpu load
Node js	**9.15**	37,052	1785	9.20	27% 0% 1% 74%
Python 3	63.19	680,688	1947	135.52	62% 70% 63% 22%
Ruby	63.32	98,996	1069	106.23	39% 28% 61% 42%

reverse-complement

source	secs	mem	gz	cpu	cpu load
Node js	16.61	708,876	1103	18.16	11% 54% 14% 31%
Python 3	16.03	1,007,016	814	19.29	18% 59% 44% 23%
Ruby	23.09	1,604,952	295	40.79	59% 45% 35% 40%

fannkuch-redux

source	secs	mem	gz	cpu	cpu load
Node js	**80.97**	30,924	473	80.95	0% 100% 1% 0%
Python 3	507.56	50,988	950	1,998.88	99% 99% 97% 99%
Ruby	545.63	14,968	1454	2,111.38	94% 97% 97% 100%

pidigits

source	secs	mem	gz	cpu	cpu load
Node js	14.26	62,876	530	14.33	63% 0% 1% 37%
Python 3	1.22	9,824	386	1.22	10% 4% 1% 100%
Ruby	1.71	172,712	485	2.35	24% 100% 72% 31%

zdroj: https://www.techempower.com/

3.3.1. Django vs. Ruby on Rails

Oba webové frameworky jsou velmi oblíbené, zároveň jsou oba open source (otevřený software s otevřeným zdrojovým kódem) a poskytují všechny potřebné funkce pro vývoj webových aplikací založených na modelu MVC. Určit lepší framework z těchto dvou je prakticky nemožné, lepší je najít vhodnější z nich pro určité zadání. [23]

Django bylo představeno na začátku diplomové práce, proto si teď řekneme něco ohledně jeho konkurentu. Jak již bylo řečeno, Ruby on Rails (RoR) je webový framework. Je založený na interpretovaném skriptovacím programovacím jazyku Ruby. Asi není potřeba na začátek více popisovat tyto technologie, lepší bude je porovnat s Djangem. [23]

Django a RoR jsou si podobný v architektuře. Oba sdílejí stejný architektonický vzor MVC. Lehký rozdíl nabízí pojmenování tří částí. RoR zachovává název MVC, zatímco Django pracuje s názvem Model Template View. Podobný si jsou i ve výkonu. Předpokládá se větší výkonnost frameworku Django, avšak tento rozdíl je zanedbatelný. Na výkonu závisí ale mnoho dalších věcí. Ruby je více orientovaný na konvenční část a přeskakuje většinu konfigurace. Na druhou stranu Python je imperativní, vývojáři tak dodává více svobody, ale může vzít i více času. Dalším společným znakem je škálovatelnost. Ta, v případě synchronně běžících webových frameworků, se zvyšuje přidáním aplikačních serverů, a nikoliv zrychlením jednoho z nich. Proto není prioritní řešit srovnání škálovatelnosti obou frameworků. Jak Django i RoR jsou dobře zdokumentovány s rozsáhlou komunitou. [23]

Po společných znacích je na čase představit rozdílné. Django využívá Python, což je, jak spousta vývojářů tvrdí, velmi srozumitelný a jednoduchý jazyk na rozdíl od ostatních programovacích jazyků. Ruby je složitější. Ve srovnání s Django má více skrytých funkcí a poskytuje více elegance, kterou by mohli ocenit zkušenější vývojáři. Elegancí chápejme potřebu psát méně kódu se stejnou funkcionalitou. Neposledním rozdílem je produktivita. Django vyžaduje kódování většinu věcí, jež jsou v RoR prováděny standardně. Směrování se provádí explicitně pomocí regulérních výrazů. Explicitně by měly být definovány i proměnné a třídy. V RoR, jak bylo řečeno, je většina věcí prováděna automaticky bez potřeby kódování. Například pro řadič nemusí být definována žádná proměnná ani třída importu. Směrování je prováděno též standardně. Je známo, že Python je pomalejší než Ruby. V podobném duchu se nese srovnání frameworků, kdy Ruby je 0,7x rychlejší. Odborné znalosti v oblasti rozvoje a dobrý hardware dokážou řešit většinu problémů týkající se rychlosti, a tak rychlost není kritickým měřítkem pro zvýšení výkonu. Oba frameworky mají velké množství knihoven, Django má navíc rozsáhlou sbírku knihoven pro výzkumy, analytiku a strojové učení (Machine Learning). [23]

Vybrat celkově lepší framework je nemožné. Lze pouze vybrat lepší framework pro řešení daného projektu, a to na základě analýzy cílů a odborných znalostí. Dalším aspektem při výběru jsou sympatie k jednomu z programovacích jazyků. [23]

3.3.2. Django vs. Express.js

Na začátku kapitoly bylo lehce naznačeno, co vlastně je Node.js. Pro lepší představivost si jej přiblížíme více. Primárním účelem Node.js je tvorba serverové části webových aplikací. Dá se tedy používat jak na straně klienta, tak i na serverové straně. Node.js klade důraz na škálovatelnost a vysokou výkonnost. [23]

Nejdůležitějším prvkem architektury Node.js je smyčka událostí (Event Loop), jež sbírá všechny uživatelské požadavky. Ty jsou následně přiděleny nezávislým vláknům. Manipulace se zdroji OS jsou řešeny pomocí událostí (na obrázku tzv. Neblokující I/O) a je řízena smyčkou událostí. Dalo by se říct, že smyčka událostí řídí pomocí různých událostí téměř všechno. Jsme svědky velice jednoduchého a efektivního řešení notabene pro webové aplikace s velkým množstvím uživatelských požadavků. [23]

Express.js je nejmladším nástrojem ze všech již uvedených frameworků. Je to výkonná technologie vhodná k budování REST API serveru pro jednoduché aplikace, stejně jako k vytváření vysoce škálovatelných produktů. Zároveň je Express.js nejjednodušším frameworkem pro webové aplikace využívajícím Node.js. Jedním z důvodů, proč se využívá tento framework je výše uvedená rychlost Node.js. Express.js obsahuje funkce webové aplikace včetně základního směrování, middlewaru, engine šablon, porce statických souborů apod. Početné pluginy třetích stran řeší npm, správce knihoven ve světě JavaScriptu, který je součástí standardní instalace Node.js. [23]

Výhodou oproti Djangu je určitě rychlost, avšak na úkor škálovatelnosti. Výhodou je také asynchronní tok dat. To umožňuje klientu odesílat a přijímat data bez přerušení. Pro příjem dat není nutná aktualizace stránky. Za zmínění stojí i výkonné směrované rozhraní API a určitě npm nabízející tisíce užitečných nástrojů pro tvorbu webové aplikace. [23]

3.3.3. Rekapitulace frameworků

Pokud se na frameworky Django vs. Ruby on Rail vs. Express.js podíváme zjednodušeně, tak by se vše dalo shrnout následovně. Pokud je pro vývoj webové aplikace vyžadována vysoká škálovatelnost, je vhodné využít Django. Pokud není vyžadována přílišná škálovatelnost a vývojáři si chtějí ušetřit práci s kódem, pak je vhodný Ruby on Rails a pro malé projekty vyžadující rychlost a očekávající velké množství uživatelských požadavků, poté je tím pravým nástrojem Express.js. Celkový souhrn frameworků je uveden v obrázku č. 19.

Obrázek 19: Porovnání frameworků

Tool	Django	Ruby on Rails	ExpressJS
Type	Web framework	Web framework	Web framework
Release date	2005	2008	2009
Programming language	Python	Ruby	JavaScript/Node.js runtime
Usage	Complex database-driven web apps	Database-backed modern web apps	Real-time apps with intense I/O, single page apps, server-side apps
Development principles	Explicit is better than implicit DRY	Convention over Configuration DRY	Event-driven programming
Architectural Pattern	Model-View-Template	Model-View-Controller	Single-Thread Executor Pattern
Learning curve	Low	Middle	Steep
Top apps	BillGuard, Instagram, Pinterest, Disqus, Sentry	Basecamp, Airbnb, GitHub, Zendesk, Hulu	TotalCast, Dial Once, impraise, HYPH
Pros	- Rapid development - Large set of features and functionalities - Low learning curve - Scalability	- High code quality - MVC principle - Convention over configuration - Great Gems - REST architecture - Flexibility - Robust community	- Non-blocking I/O- Powerful routed API- Full stack JS back-end development- Middleware- Ecosystem- Corporate support
Cons	- Monolithic architecture - Regular expressions to specify URL - Small projects are not a fit - Components are deployed together	- Runtime speed - Hosting - Documentation	- Callback hell- No object-relational mapping support- Dependent cloud-based architecture- Insufficiency with heavy computations
Websites built	Almost 205K	Over 2200K	Over 500K

zdroj: https://railsware.com/blog/2018/06/13/python-vs-ruby-vs-node-js-which-platform-is-a-fit-for-your-project/

3.4. Neit Consulting s.r.o.

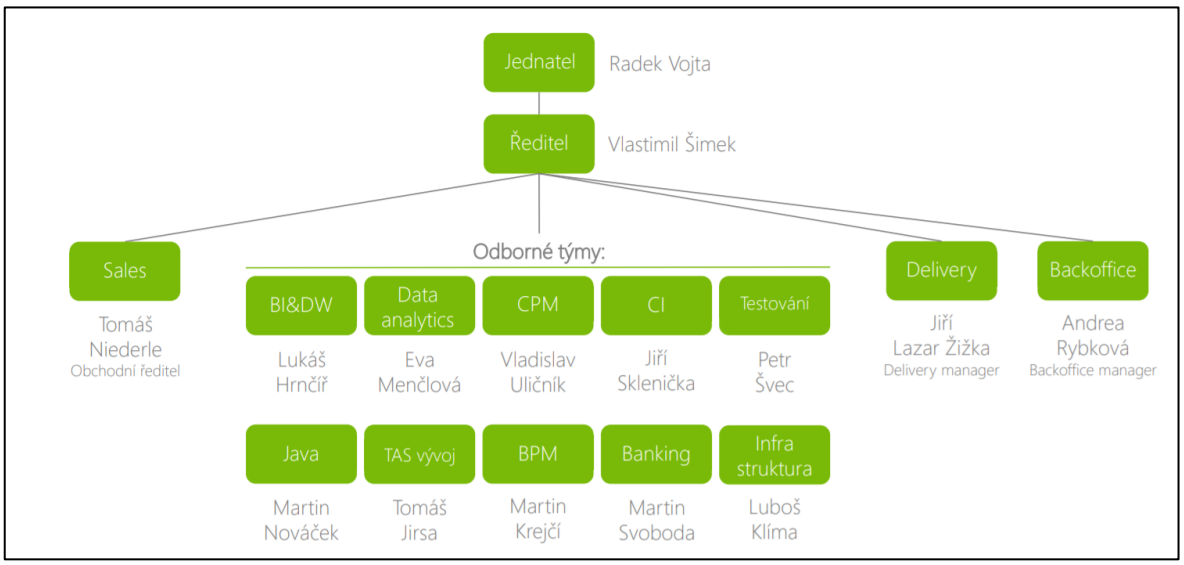
Společnost Neit Consulting s.r.o. je významná a dynamicky rostoucí společnost pohybující se v IT odvětví od roku 2005. Její působnost se za dobu působnosti na trhu rozrostla. Momentálně poskytuje následující služby:

- BI & Analitycs – Business Intelligence. Reporting, analýza. Samoobslužné BI. BI mobile. Data mining, statistiky, strojové učení.
- Customer Intelligence – Customer Relationship Management. Příprava, realizace a hodnocení kampaní. Detekce podvodů.
- Big Data Solutions – Agilní metodologie. Architektura DWH a datový model. Data Vault 2.0. Real-time nebo dávkový ETL. Strukturovaná a nestrukturovaná data ze sociálních médií, senzorů, strojů, webových protokolů a dalších zdrojů.
- CPM – Corporate Performance Manager. Plánování, rozpočtování a prognózování. Konsolidace a podávání zpráv.

- Regulační výkaznictví – Místní předpisy a předpisy EU.
- Automatizaci – Modelování a optimalizace procesů. Automatizace pracovních postupů. Datová a aplikační rozhraní. Formuláře uživatelů, přehledy a zprávy.
- Vývoj aplikací – Java, PHP, NodeJS, Oracle APEX, Python. Architektura orientovaná na služby. Rychlý vývoj aplikací. Agilní metodologie. Windows, Linux, iOS, Android.
- Testování – Analýza současného aplikačního prostředí, testování scénářů a testovacích plánů, testování administrace, testování rozlišení. Funkční, akceptační, výkonové a zátěžové zkoušky.
- Trénování a vzdělání – Oracle produkty, technologie a aplikace. Autorizované školení Oracle ve spolupráci s Oracle University. Individuální školení.

Ve struktuře společnosti je nejvýše jednatel s ředitel. Pod nimi se nachází obchodní ředitel, delivery manager, backoffice manager a manažeři jednotlivých divizí viz obrázek níže.

Obrázek 20: Struktura společnosti Neit Consulting s.r.o.



zdroj: Neit Consulting s.r.o.

Neit Consulting s.r.o. je zlatým partnerem Oracle a stříbrným partnerem SAS. Zároveň dodává řešením takovým společnostem jako je Equa bank, Česká národní banka, ČSOB, KB, MND nebo třeba Dr. Max.

4. Vlastní práce

Teoretickou část bychom měli za sebou. Ucelili jsme si v ní představu o tom, co jsou informační systémy, z čeho jsou složené a jak se postupuje při jejich tvorbě. Vysvětleny byly i pojmy nezbytné pro tvorbu informačního systému, byly porovnány vybrané frameworky a nakonec byla přiblížena i samotná společnost, pro kterou je informační systém vyvíjen. Na řadě je tedy vytvořit informační systém.

4.1. Sběr a analýza požadavků

Před začátkem projektu byla uskutečněna schůze mezi mnou a pověřenými zaměstnanci společnosti Neit Consulting s.r.o. Tato schůze se stala výchozím bodem pro tvorbu informačního systému.

Neit Consulting s.r.o. navrhl seznam požadavků pro informační systém. Primárním požadavkem se stala funkce pro výkaz práce. Tato funkce musí zaměstnanci nabídnout možnost zadat počátek a konec práce pro každý pracovní den v měsíci. Management pak bude schopný si jednotlivé výkazy prohlížet. Další neméně důležitou funkcí je žádost o nepřítomnost. Zaměstnanec, pokud bude chtít absentovat, zadá pomocí formuláře žádost, kterou mu management buď schválí nebo zamítne. Výsledné rozhodnutí se dozví sám zaměstnanec na webových stránkách informačního systému. Dále byl zadán požadavek na vytvoření sekce s články a schopnost nahrávat a stahovat soubory. Informační systém by měl rozeznávat 3 skupiny uživatelů a to: zaměstnanec, manažer a admin. U všech skupin budou jasně rozdělená práva. Uživatelé, kteří nebudou přihlášení do IS, nebudou schopní si zobrazit obsah. Jelikož Neit Consulting s.r.o. má Python nově vznikající oddělení, byl pro vytvoření informačního systému vybrán framework Django podporující právě jazyk Python. Zároveň bylo domluveno, že společnost zavede informační systém sama do provozu, neboť pro zavedení mají vše potřebné, postačí jim pouze balíček s nakonfigurovaným obsahem.

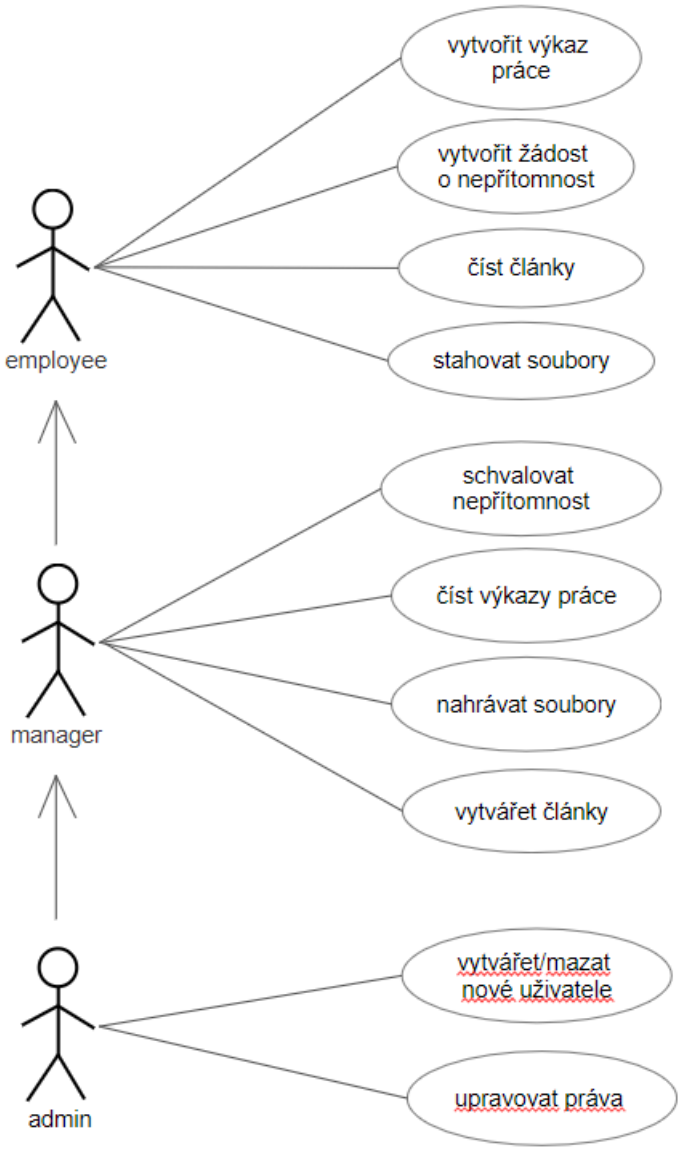
Očekává se, že informační systém nebude zpracovávat velké množství dat, nicméně kvůli eventuálním změnám požadavků byla pro projekt zvolena agilní metodika zahrnující agilní modely životního cyklu. Prioritou je tedy dodávání IS v malých částech. Nejdříve bude vytvořen celkový návrh informačního systému, po schválení ze strany společnosti Neit Consulting s.r.o. se přejde na implementaci, která bude rozdělena do několika iterací a průběžně představována zákazníkovi kvůli zpětné vazbě.

4.2. Návrh

4.2.1. Případy užití

Případy užití nebo Use Cases byly vysvětleny už v teoretické části, proto není zapotřebí tyto pojmy více rozebírat. Na obrázku č. 21 se nacházejí 3 aktoři podle zadání. Employee může vytvořit výkaz práce, žádost o nepřítomnost, číst články a stahovat soubory. Manager může provádět stejné úkony jako employee a navíc schvaluje nepřítomnost, pročítá výkazy práce, nahrává soubory a vytváří články. Posledním aktorem je admin, také sdílí možnosti employee a dokonce i managera. Navíc může spravovat uživatele a upravovat práva.

Obrázek 21: Use Case diagram

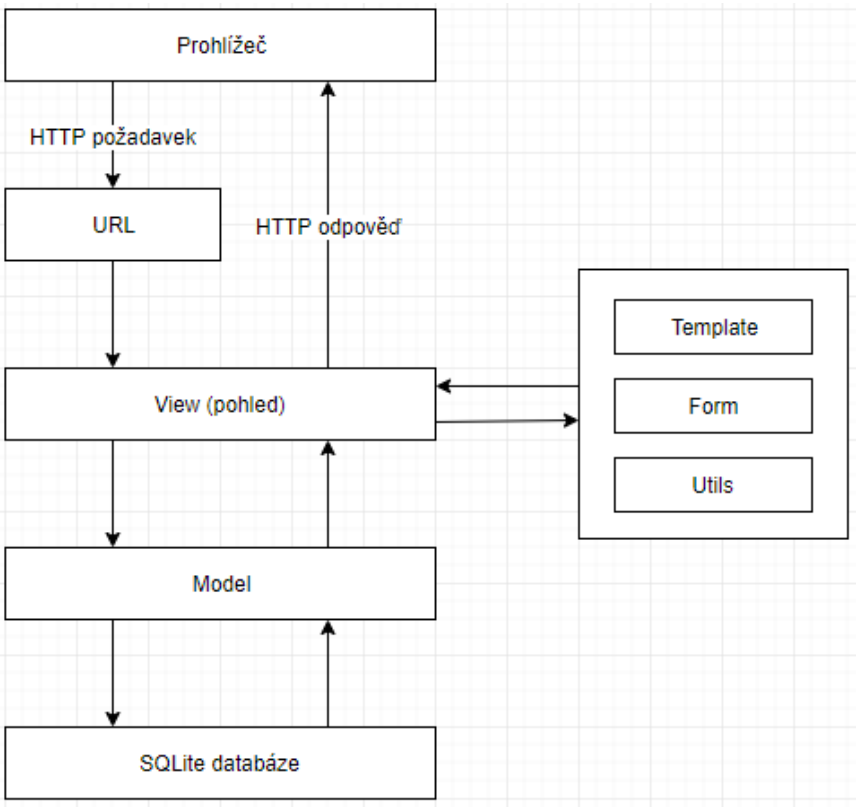


zdroj: vlastní tvorba

4.2.2. Architektura systému

Architektura systému je úzce spjata s architekturou Django. Jak již bylo řečeno, tento framework využívá architekturu MVC. Tudíž prohlížeč vyšle HTTP požadavek, ten je za směrován podle adresy URL (v Djangu soubor urls.py) a napojuje se na View. Odtud probíhá komunikace s ostatními komponenty frameworku.

Obrázek 22: Architektura systému



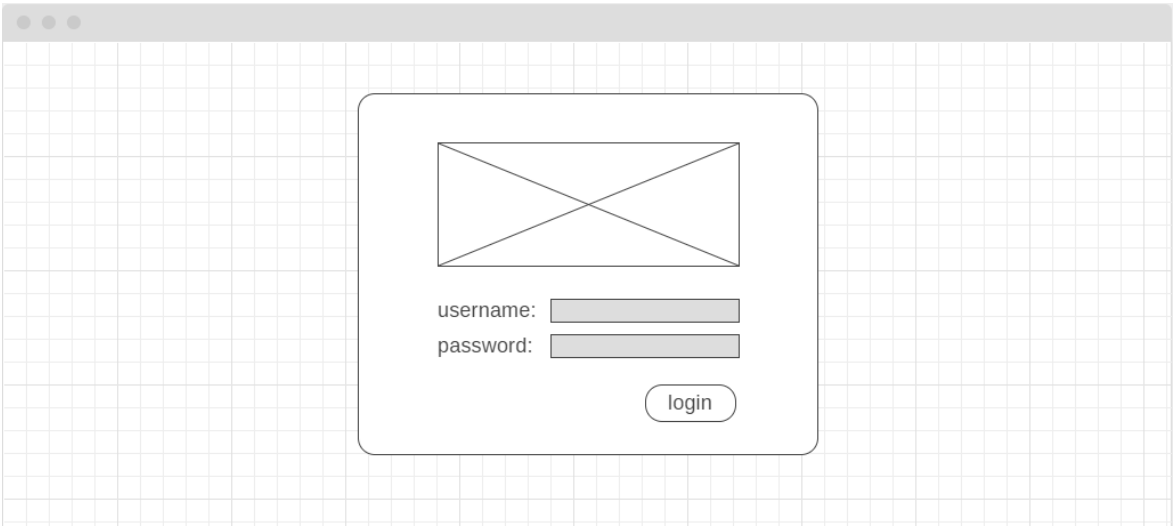
zdroj: vlastní tvorba

4.2.3. Uživatelské rozhraní

Při tvorbě uživatelského rozhraní byl pro lepší názornost využit wireframe. První wireframe (obrázek č.23) zobrazuje přihlašovací stránku. Na té bude logo Neit Consulting s.r.o., přihlašovací údaje a tlačítko pro přihlášení.

Druhý wireframe (obrázek č.24) zobrazuje, jak bude informační systém vypadat. V horní části bude umístěna hlavička s logem společnosti a možností se odhlásit. Bylo vybráno boční menu, které se bude nacházet v levé části. Veškerý obsah se bude zobrazovat ve vedlejším bloku. Jak menu, tak i blok s obsahem jsou v horní části napojeni na hlavičku a ve spodní zakončeny zaoblenými rohy. Stránka je ukončena patičkou.

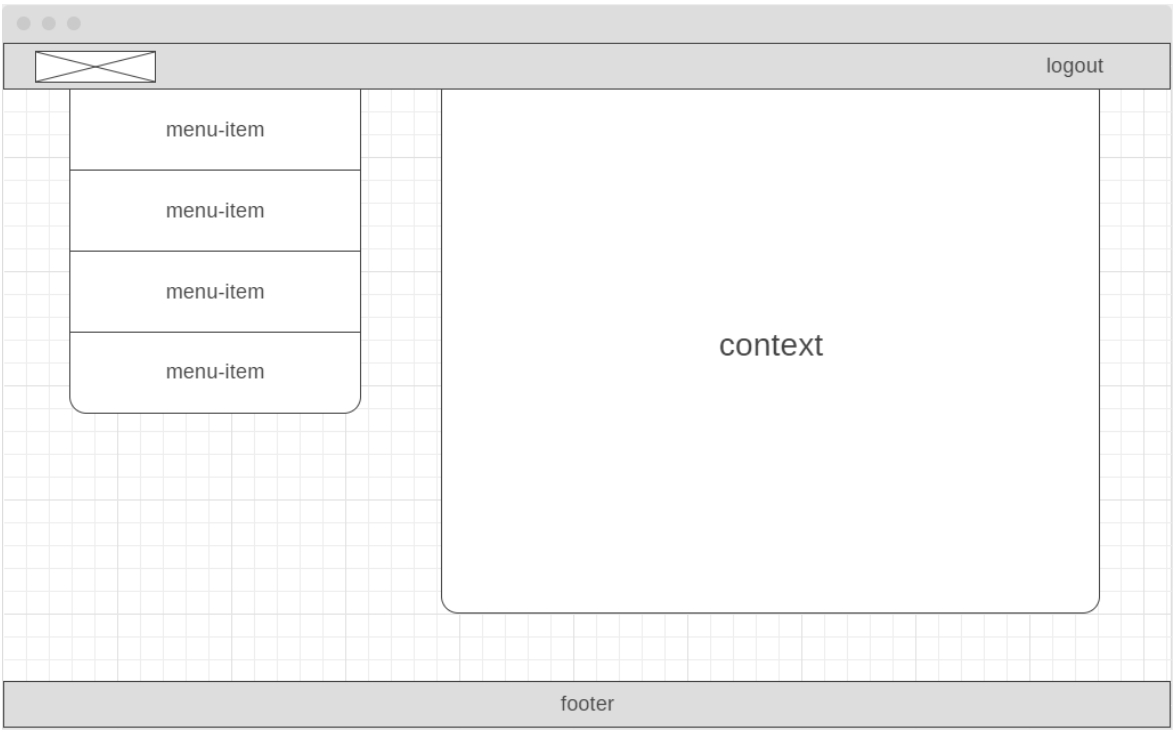
Obrázek 23: Wireframe přihlašovací stránky



zdroj: vlastní tvorba

Obě šablony byly navrženy tak, aby uživatelské rozhraní informačního systému bylo pro uživatele přehledné, jednoduché pro práci, ale zároveň moderní a hezky vypadající.

Obrázek 24: Wireframe hlavní stránky

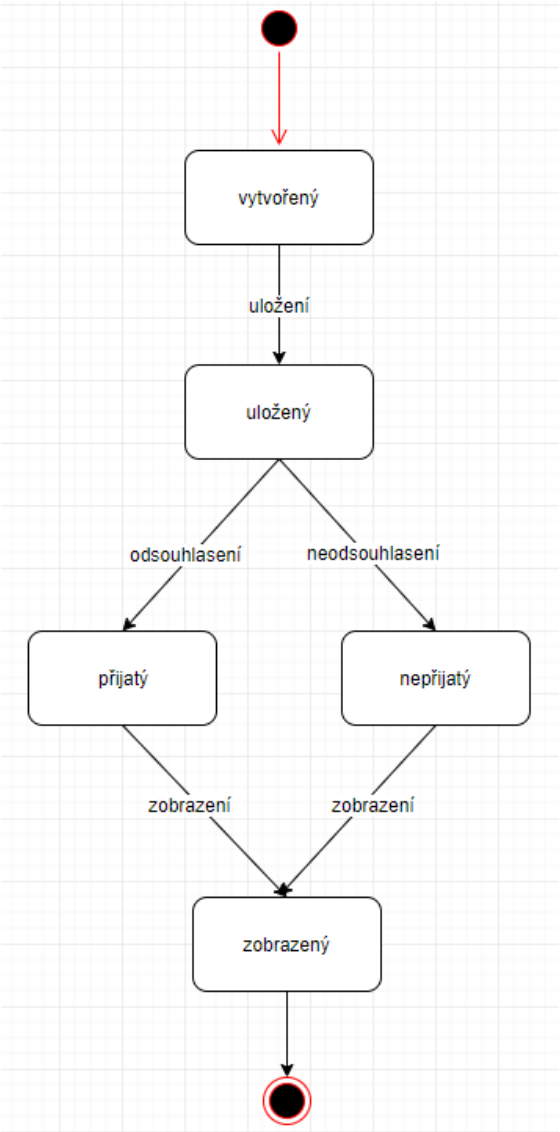


zdroj: vlastní tvorba

4.2.4. Stavové diagramy

Stavový diagram patří pod diagram chování, popisuje stavy a přechody mezi nimi. Stavový diagram se zpravidla tvoří pro důležité entity, vytváří pohled na jejich životní cyklus. Můžeme tak vidět, jak systém funguje. První diagram (obrázek č.25) se týká výkazu práce. Žádost se vytvoří, uloží. Poté dojde buďto ke schválení nebo zamítnutí a výsledné rozhodnutí bude zobrazeno uživateli.

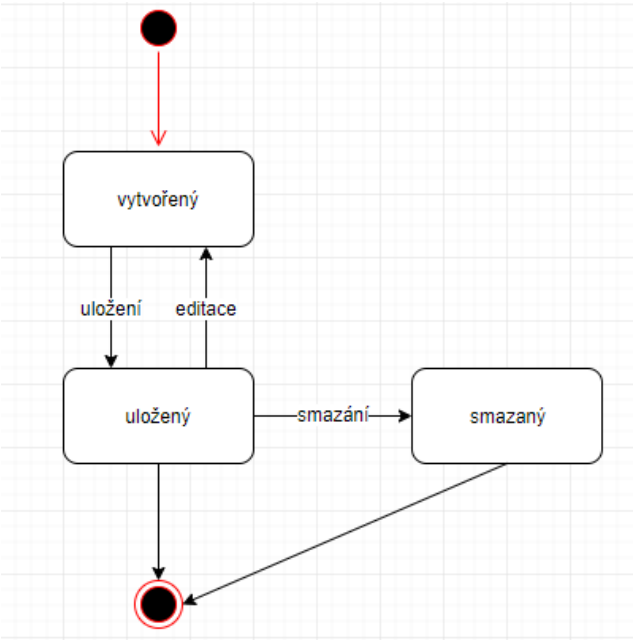
Obrázek 25: Stavový diagram pro žádost o nepřítomnost



zdroj: vlastní tvorba

Druhý stavový diagram (obrázek č.26) zobrazuje žádost o nepřítomnost. Uživatel vytvoří žádost, která se uloží. Následně může tuto žádost ještě upravit nebo smazat.

Obrázek 26: Stavový diagram pro výkaz práce



zdroj: vlastní tvorba

4.3. Vývoj

Po odsouhlasení návrhu přichází na řadu vývoj. Ten byl naplánován do 4 iterací. V první iteraci bylo vytvořeno přihlášení s následným přesměrováním na hlavní stránku informačního systému, kde se zobrazovali články. Vše bylo otestováno a předvedeno společnosti Neit, která byla s prozatímním výsledkem spokojená.

Druhá iterace se věnovala výkazu práce, což se dá považovat za stěžejní bod projektu. Tato funkce byl opět otestována a předvedena. Zde již pár výtek bylo, a tak se museli předchozí činnosti opakovat, dokud nebyli zástupci společnosti Neit spokojení.

Třetí iterace se zabývala zadáním nepřítomnosti a možnosti zobrazení této nepřítomnosti zadávajícím uživatelem IS. Znovu se opakovali již zmíněné kroky. Po schválení i třetí iterace následovala poslední čtvrtá iterace specializovaná na ukládání a stahování souborů.

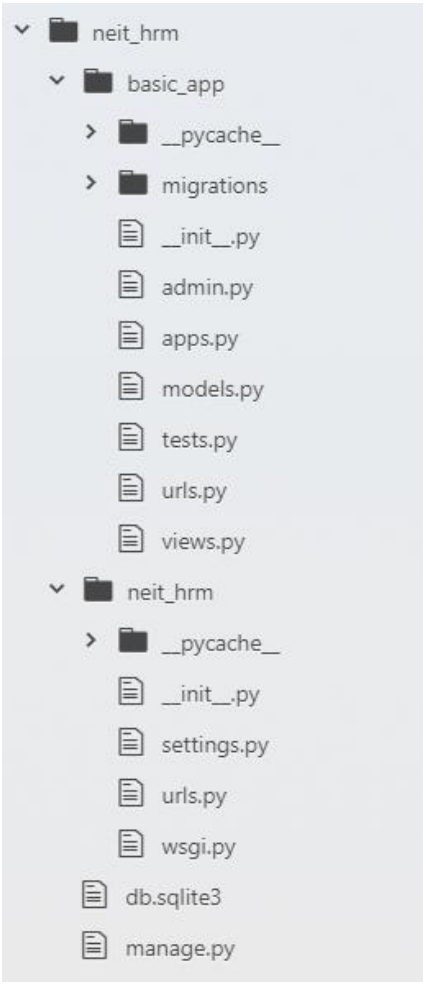
4.3.1. Vývojové prostředí

Vývojovým prostředím se pro tento projekt stal open source textový editor, dostupný na všech platformách, Atom, jenž nabízí velké množství pluginů nebo například otevření příkazové řádky přímo v editoru. Na konfiguraci prostředí včetně instalace Python, byl použit balíček Anaconda, obsahující přes 1,500 Python/R data science balíčků.

4.3.2. Vytvoření aplikace a správy systému Django

Zadáním příkazu *'django-admin startproject neit_hrm'* do příkazové řádky byla vytvořena struktura projektu. Tímto momentem je možné si zobrazit základní stránku informačního systému. Nicméně je nutné ještě přidat do projektu aplikaci, která bude obsahovat v sobě vše potřebné viz obrázek níže, a zajistit přístup do správy systému.

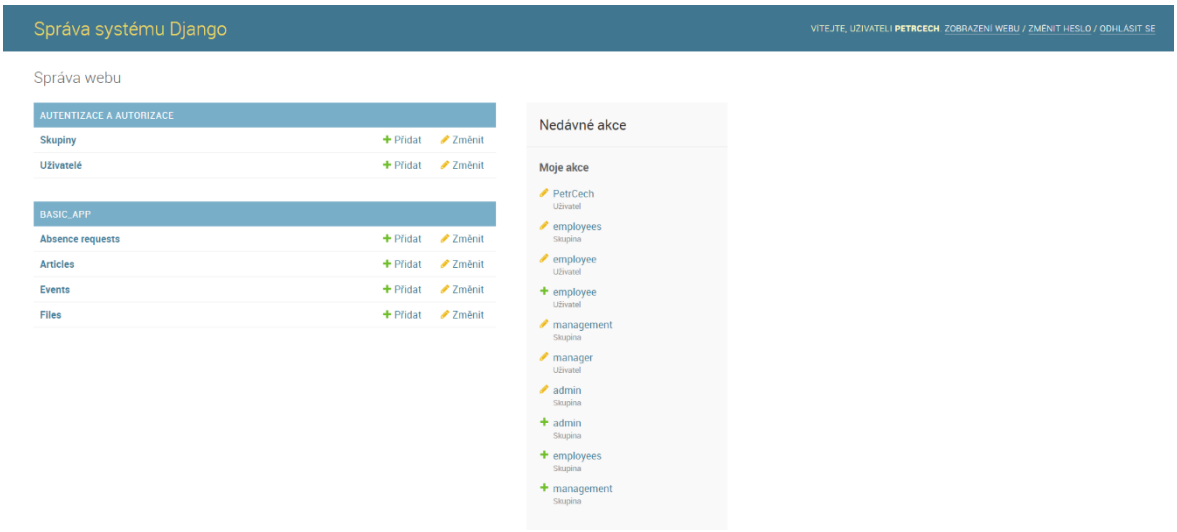
Obrázek 27: Struktura IS



zdroj: vlastní tvorba

Vznik aplikace byl uskutečněn příkazem *'python manege.py startapp basic_app'*. Nutné je zapsat aplikaci basic_app do settings.py, aby s ní framework Django počítal. Přihlášení do Správy systému Django bylo vyřešeno příkazem *'python manage.py createsuperuser'*, jenž vytvoří super uživatele, který se následně může do Správy systému Django přihlásit. Jak vypadá prostředí Správy systémů se lze podívat na obrázku č. 28. Skládá se z horního navigačního baru a obsahu, který nabízí manipulaci se skupinami a uživateli, vytvořenou aplikací a zobrazuje nedávné akce.

Obrázek 28: Rozhraní Správy systému Django



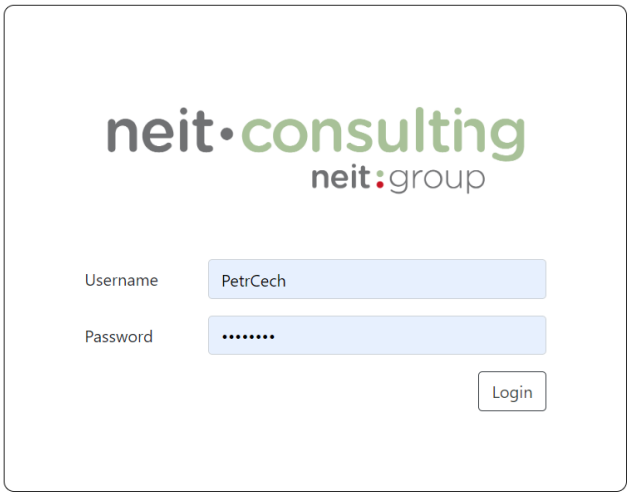
zdroj: vlastní tvorba

Samozřejmě se obsah Správy systému liší podle jednotlivých uživatelů, lépe řečeno, liší se podle skupiny, do které uživatel spadá. O tom, ale více v další části.

4.3.3. Přihlášení

První stránkou informačního systému je přihlášení viz obrázek níže. Přihlašovací stránka nese logo společnosti, pole pro zadání uživatelského jména, hesla a tlačítko, které odešle údaje do systému a ověří, zdali přihlašovací údaje odpovídají těm uloženým v databázi.

Obrázek 29: Přihlášení do informačního systému



zdroj: vlastní tvorba

Pokud se podíváme na strukturu stránky, lze vidět kromě HTML kódu a stylizace pomocí Bootstrapu 4 i kód nespadající pod již uvedené možnosti. Je uzavřen ve složených závorkách a slouží k propojení s Djangem. Nachází se na druhém řádku a načítá statické soubory. Statické soubory se využívají například pro načtení obrázku, čehož jsme svědky na 33. řádku. Výsledkem je zobrazení loga na přihlašovací stránce. Hlavním důvodem rozboru HTML kód je ale 34.řádka. Action ve formuláři se odkazuje na user_login. Co to je a jak to funguje?

Obrázek 30: HTML přihlašovací stránky

```
1  <!DOCTYPE html>
2  {% load staticfiles %}
3  <html lang="en" dir="ltr">
4    <head>
5      <meta charset="utf-8">
6      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
7      <title></title>
8    </head>
9  >  <style media="screen">…
31   <body>
32     <div class="container">
33       <img src={% static "images/logo.png" %} alt="neit logo">
34       <form method="POST" action="{% url 'user_login' %}">
35         {% csrf_token %}
36         <div class="form-group row">
37           <label class="col-sm-3 col-form-label" for="username">Username</label>
38           <div class="col-sm-9">
39             <input class="form-control" type="text" name="username" placeholder="Username">
40           </div>
41         </div>
42         <div class="form-group row">
43           <label class="col-sm-3 col-form-label" for="password">Password</label>
44           <div class="col-sm-9">
45             <input class="form-control" type="password" name="password">
46           </div>
47         </div>
48         <div class="col-sm-13 text-right">
49           <input class="btn btn-outline-dark" type="submit" name="" value="Login">
50         </div>
51       </form>
52     </div>
53   </body>
54 </html>
```

zdroj: vlastní tvorba

Pro odpověď je podstatné se podívat do views.py v aplikaci base_app. Jde totiž o pohled, jehož funkcí je uložit do proměnných hodnoty z vyplněného formuláře a ověřit je s existujícími daty. Pokud uživatel existuje a jeho účet je aktivován, pak dochází k přesměrování na hlavní stránku.

Obrázek 31: Views user_login

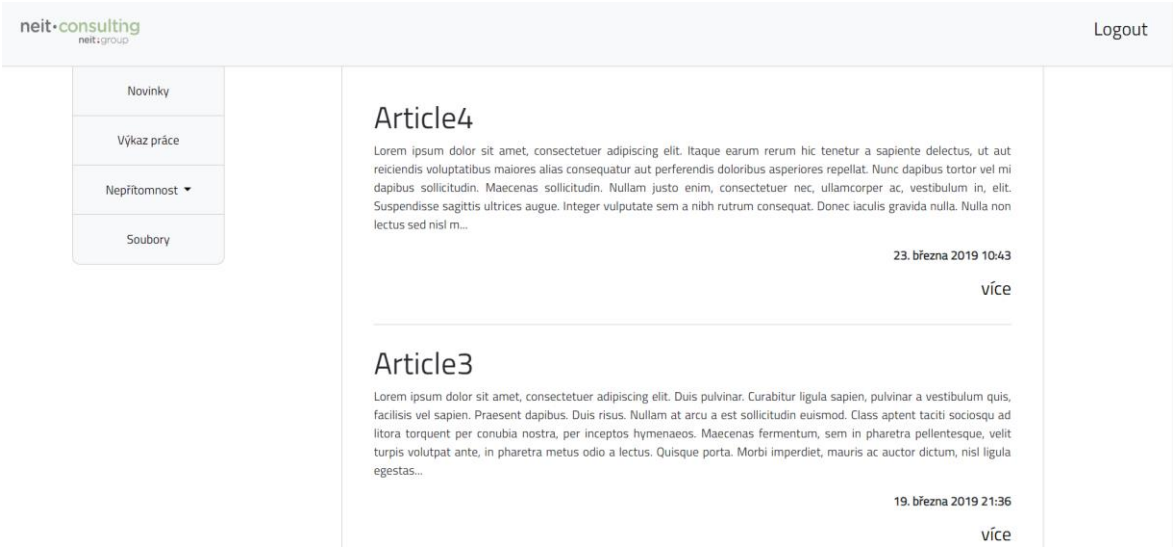
```
122  #prihlaseni
123  def user_login(request):
124      if request.method == 'POST':
125          username = request.POST.get('username')
126          password = request.POST.get('password')
127          user = authenticate(username=username, password=password)
128          if user:
129              if user.is_active:
130                  login(request,user)
131                  return HttpResponseRedirect(reverse('index'))
132              else:
133                  return HttpResponse("Your account is not active.")
134          else:
135              print("Username: {} tried to login and failed.".format(username,password))
136              return HttpResponse("Invalid login details supplied.")
137
138      else:
139          return render(request, 'login.html', {})
```

zdroj: vlastní tvorba

4.3.4. Články

Hlavní stránkou je stránka s články viz obrázek níže. Jednotlivé články se skládají z nadpisu, náhledu, datumu a možnosti více pro zobrazení celého článku. Jsou řazeny od nejnovějšího článku po nejstarší.

Obrázek 32: Hlavní stránka IS



zdroj: vlastní tvorba

HTML kód pro přihlášení byl jednorázový, teď už je však nutné mít pro všechny zbylé stránky stejnou základní strukturu. Django nabízí možnost napojovat na sebe jednotlivé stránky pomocí vnoření. Proto byl vytvořen soubor base.html (obrázek č.33).

Obrázek 33: base.html

```
<!DOCTYPE html>
{% load staticfiles %}
<html>
  <head>
    <meta charset="utf-8">
    <title>Neit HRM</title>
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.bundle.min.js ">
    <link href="https://fonts.googleapis.com/css?family=Titillium+Web" rel="stylesheet">
    <style media="screen">=
  </head>

  <body>
    {% if user.is_authenticated %}
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <a class="navbar-brand" href="{% url 'index' %}">
          <img src={% static "images/logo.png" %} width="170" height="50" alt="neit logo">
      </a>
      <div class="collapse navbar-collapse" id="navbarText">
        <ul class="navbar-nav mr-auto">
        </ul>
        <span class="navbar-text">
          <h4><a class="nav-link" href="{% url 'logout' %}">Logout</a></h4>
        </span>
      </div>
    </nav>

    <div class="container-fluid">
      <div class="row" id="main-row">
        <div class="col-sm-2" id="main-col-sm-2">
          <nav class="nav flex-column">
            <a class="nav-link" href="{% url 'index' %}">Novinky</a>
            <a class="nav-link" href="{% url 'calendar' %}">Výkaz práce</a>
            <a class="nav-link dropdown-toggle" href="{% url 'request_form_view' %}" id="navbarDropdown" role="button"
            data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                Nepřítomnost
            </a>
            <div class="dropdown-menu" aria-labelledby="navbarDropdown">
              <a class="dropdown-item" href="{% url 'request_form_view' %}">Formulář</a>
              <a class="dropdown-item" href="{% url 'request_list' %}">Výpis nepřítomnosti</a>
            </div>
            <a class="nav-link" href="{% url 'file_list' %}">Soubory</a>
          </nav>
        </div>

        <div class="col-sm-8" id="main-col-sm-8">
          {% block body_block %}
          {% endblock %}
        </div>
      </div>
    </div>

    {% else %}
    <blockquote class="blockquote text-center">
      <h2>Nejste přihlášen.</h2>
      <p>Pro přihlášení klikněte prosím <a href="{% url 'user_login' %}">zde!</a></p>
    </blockquote>
    {% endif %}
    <footer>2018 &copy Petr Čech</footer>
    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
    integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.11.0/umd/popper.min.js"
    integrity="sha384-b/U6ypiBEHpOf/4+1nzFpr53nxSS+GLCkfwBdFNTxtclqqenISfwAzpKaMNFNmj4" crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/js/bootstrap.min.js"
    integrity="sha384-h0AbiXch4ZDo7tp9hKZ4TsHbi047NrKGLO3SEJAg45jXxnGIfYzk4Si90RDIqNm1" crossorigin="anonymous"></script>

  </body>
</html>
```

zdroj: vlastní tvorba

Base.html v sobě obsahuje hlavní strukturu a nabízí schopnost napojit se do ní ostatním HTML souborům, avšak pouze na určené místo. Tím místem je oblast context zobrazená výše ve wireframu. Pokud se podíváme, jak vypadá base.html, vidíme HTML kostru včetně head a body. V body se nachází hlavička s logem a možností odhlášení. Krom toho se zde nachází i postranní menu, oblast pro napojení dalšího HTML souboru a patička. Nejdůležitější části base.html jsou dvě, první se nachází na řádku 221. Tento příkaz zaručuje, že obsah bude skrytý pro nepřihlášené uživatele. Pakliže si stránku bude chtít zobrazit nepřihlášený uživatel, provede se část HTML kódu začínající na řádku 262 a oznámí uživateli, že není přihlášen a nabídne mu přesměrování na přihlašovací stránku. Druhou neméně důležitou částí jsou řádky 254 a 255, jež oznamují, že právě zde je možné napojit další HTML soubor.

Napojení pak vypadá třeba jako na obrázku níže Toto je výtah z HTML souboru určeného pro hlavní stránku. Stejně jako v předchozím případě se propojuje s Djangem, tentokráte pomocí for cyklu.

Obrázek 34: index.html

```
1  {% extends "base.html" %}
2  {% block body_block %}
3    {% for art in articles %}
4      <div class=article_list>
5        <h1> {{ art.title }}</h1>
6        <div class=article_detail>
7          <p> {{ art.snippet }}</p>
8        </div>
9        <div class="article_rest">
10         <p class="article_date"><b>{{ art.date }}</b></p>
11         <h4><a href="{% url 'article_detail' slug=art.slug %}">více</a></h4>
12       </div>
13     </div>
14   {% endfor %}
15 {% endblock %}
```

zdroj: vlastní tvorba

Abychom správně pochopili, jak funguje tento pohled, musíme se poprvé podívat i do models.py (obrázek č. 35), kde se nalézá třída Article. Tato třída obsahuje atributy title, slug, body a date metodu __str__ a funkci snippet, jenž vytváří náhled článku dlouhý 500 znaků. Metoda str je využívána hlavně kvůli lepšímu zobrazení, v tomto případě Article, ve Správě systému.

Obrázek 35: Model class Article

```
19  class Article(models.Model):
20      title = models.CharField(max_length = 100)
21      slug = models.SlugField(unique=True)
22      body = models.TextField()
23      date = models.DateTimeField(auto_now_add=True)
24
25      def __str__(self):
26          return self.title
27
28      def snippet(self):
29          return self.body[:500] + '...'
```

zdroj: vlastní tvorba

Výše zmíněná třída Article se používá k uložení jednotlivých článků. Ve views.py je tato třída využívána ve dvou funkcích. První funkce s názvem index vrací slovník obsahující všechny články seřazená podle od nejnovějšího článku po nejstarší a ta druhá využívá atribut slug. Podle slugu se vyhledá daný článek v databázi a po kliknutí na článek se uživatel právě pomocí slugu dostane na jeho plnou verzi.

Obrázek 36: Views index a article_detail

```
15  # clanky
16  def index(request):
17      articles = Article.objects.all().order_by('-date')
18      return render(request,'index.html',{'articles':articles})
19
20  #clanky detail
21  def article_detail(request,slug):
22      article = Article.objects.get(slug=slug)
23      return render(request,'articles/article_detail.html',{'article':article})
```

zdroj: vlastní tvorba

Jedinou možností, jak spravovat články je přes Správu systému. Jak již bylo uvedeno, pod basic_app se nachází položka Articles. Má-li přihlášený uživatel patřičná práva, po rozkliknutí této položky se mu zobrazí všechny již vytvořené články. Články mohou být přidávány, upravovány a samozřejmě mazány. Pohled do Správy systému konkrétně do části Articles je zobrazen na obrázku č.37.

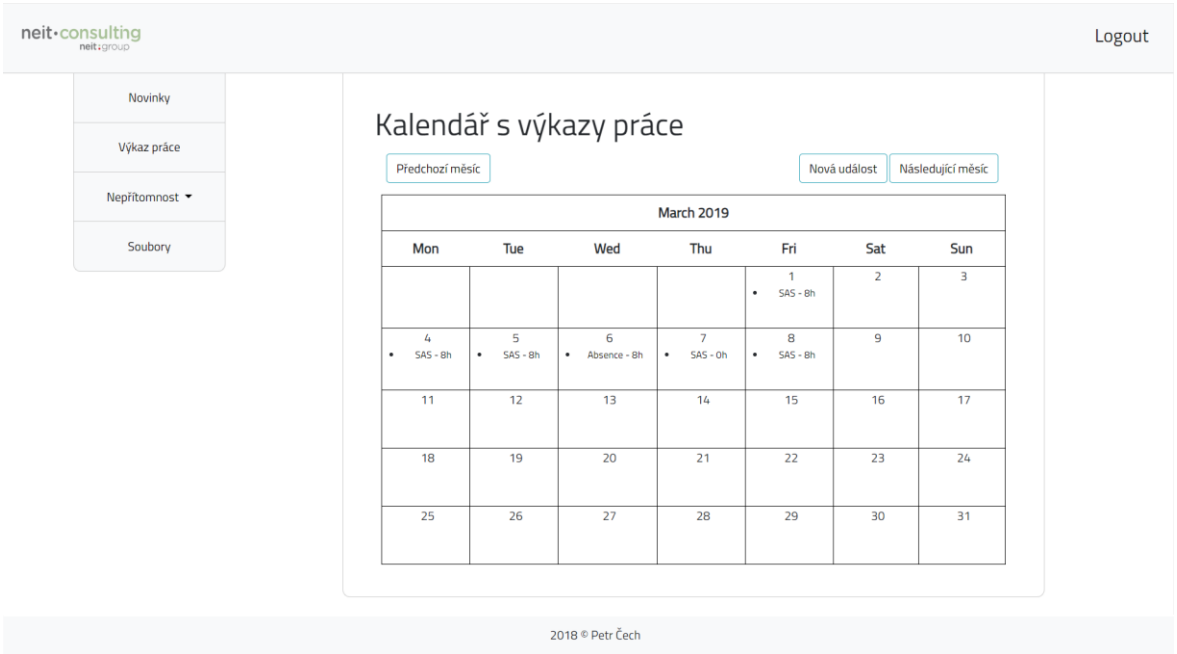
Obrázek 37: Správa systému - Articles

zdroj: vlastní tvorba

4.3.5. Výkaz práce

Po představení sekce článků se pozvolně dostáváme na další sekci, konkrétně na výkaz práce. Stránka obsahuje kalendář s daným měsícem. Nad kalendářem nalezneme možnosti přepínaní mezi měsíci a vytvoření nové události (viz. obrázek níže). Tato stránka slouží uživatelům pro výkaz práce.

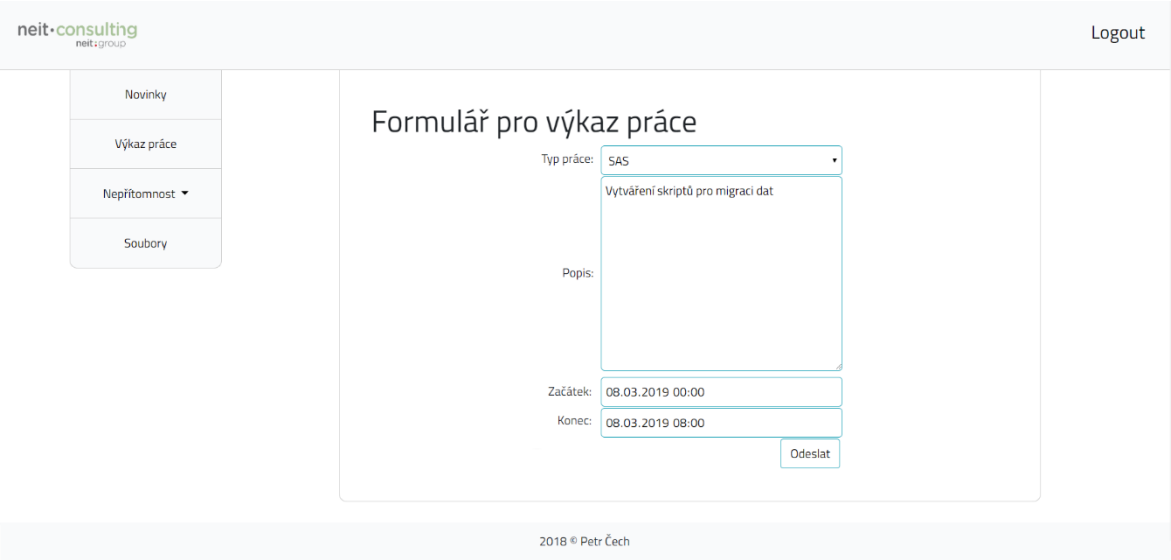
Obrázek 38: Výkaz práce



zdroj: vlastní tvorba

Chtějí-li vykázat odpracované hodiny, kliknou do kalendáře na určitý den. Po kliknutí je uživatel přenesen na další stránku s formulářem (obrázek č. 39). Ve formuláři se vyplňují pole pro typ práce, její popis a časové rozmezí, kdy byla práce vykonávána. Kliknutím na tlačítko odeslat se odešle výkaz do systému a zobrazí se v kalendáři, jak je možné vidět na předchozím obrázku.

Obrázek 39: Formulář pro výkaz práce



zdroj: vlastní tvorba

Pro to, aby vše fungovalo tak, jak má, bylo nutné zajistit několik věcí. Nejdříve byla vytvořena třída Event s potřebnými atributy a funkcemi.

Obrázek 40: Model class Event

```
6   class Event(models.Model):
7       title = models.CharField(max_length=100,default='')
8       author = models.CharField(max_length=100,default='')
9       description = models.TextField()
10      start_time = models.DateTimeField(default=datetime.date.today)
11      end_time = models.DateTimeField(default=datetime.date.today)
12      hour_number = models.IntegerField(null=True)
13
14      def __str__(self):
15          return str(self.start_time) + ' ' + self.author + ' - ' + str(self.hour_number)
16
17      @property
18      def get_html_url(self):
19          url = reverse('event_edit', args=(self.id,))
20          return f'<a href="{url}"> {self.title} - {self.hour_number}h</a>'
```

zdroj: vlastní tvorba

Lze vidět, že zde existují dva atributy, jež se ve formuláři pro výkaz práce výše nevyskytují. Jedním z nich je author a druhý hour_number. Author slouží pro uložení uživatelského jména uživatele, který vyplňuje formulář a hour_number automaticky počítá, kolik uživatel odpracoval odečtením end_time od start_time.

Tato funkce byla následně přivolána ve forms.py, kde se vytváří formuláře. V tomto případě byl vytvořen formulář EventForm.

Obrázek 41: Forms EventForm

```
project_type = (
    ('SAS','SAS'),
    ('Oracle','Oracle'),
    ('BI','BI'),
    ('Java','Java'),
    ('Absence','Absence')
)


class EventForm(ModelForm):
  class Meta:
    model = Event
    widgets = {
      'start_time': DateInput(attrs={'type': 'datetime-local'},
                              format='%Y-%m-%dT%H:%M'),
      'end_time': DateInput(attrs={'type': 'datetime-local'},
                              format='%Y-%m-%dT%H:%M'),
      'title': forms.Select(choices=project_type)
    }
    fields = ['title','description','start_time','end_time',]

  def __init__(self, *args, **kwargs):
    super(EventForm, self).__init__(*args, **kwargs)
    self.fields['start_time'].input_formats = ('%Y-%m-%dT%H:%M',)
    self.fields['end_time'].input_formats = ('%Y-%m-%dT%H:%M',)
```

zdroj: vlastní tvorba

Speciálně pro tento případ je vytvořen utils.py. V tomto souboru se vytváří kalendář, pro jehož tvorbu byl využit v Pythonu zabudovaný modul Calendar, který nejenže vytvoří kalendář, ale také nabízí mnoho užitečných funkcí. Jako je například nastavení HTML struktury kalendáře.

Obrázek 42: Utils Calendar

```
7   class Calendar(HTMLCalendar):
8       def __init__(self, year=None, month=None):
9           self.year = year
10          self.month = month
11          super(Calendar, self).__init__()
12
13
14      def formatday(self, day, events):
15          events_per_day = events.filter(start_time__day=day)
16          d = ''
17          for event in events_per_day:
18              d += f'<li> {event.get_html_url} </li>'
19
20          if day != 0:
21              return f"<td><span class='date'>{day}</span><ul> {d} </ul></td>"
22          return '<td></td>'
23
24
25      def formatweek(self, theweek, events):
26          week = ''
27          for d, weekday in theweek:
28              week += self.formatday(d, events)
29          return f'<tr> {week} </tr>'
30
31
32      def formatmonth(self, withyear=True):
33          events = Event.objects.filter(start_time__year=self.year,
34                                        start_time__month=self.month)
35
36          cal = f'<table border="0" cellpadding="0" cellspacing="0" class="calendar">\n'
37          cal += f'{self.formatmonthname(self.year, self.month, withyear=withyear)}\n'
38          cal += f'{self.formatweekheader()}\n'
39          for week in self.monthdays2calendar(self.year, self.month):
40              cal += f'{self.formatweek(week, events)}\n'
41          return cal
```

zdroj: vlastní tvorba

Když je vše potřebné připraveno, je podstatné vše pospojovat ve views.py. Ve funkci třídy CalendarView s názvem get_context_data je napojen kalendář. Vytvořeny jsou i funkce pro posouvání měsíců v kalendáři (def prev_month a def next_month) a podobně. Na řádku 74 a 75 můžeme vidět naplnění atributů hour_number a author, jak již bylo zmíněno výše.

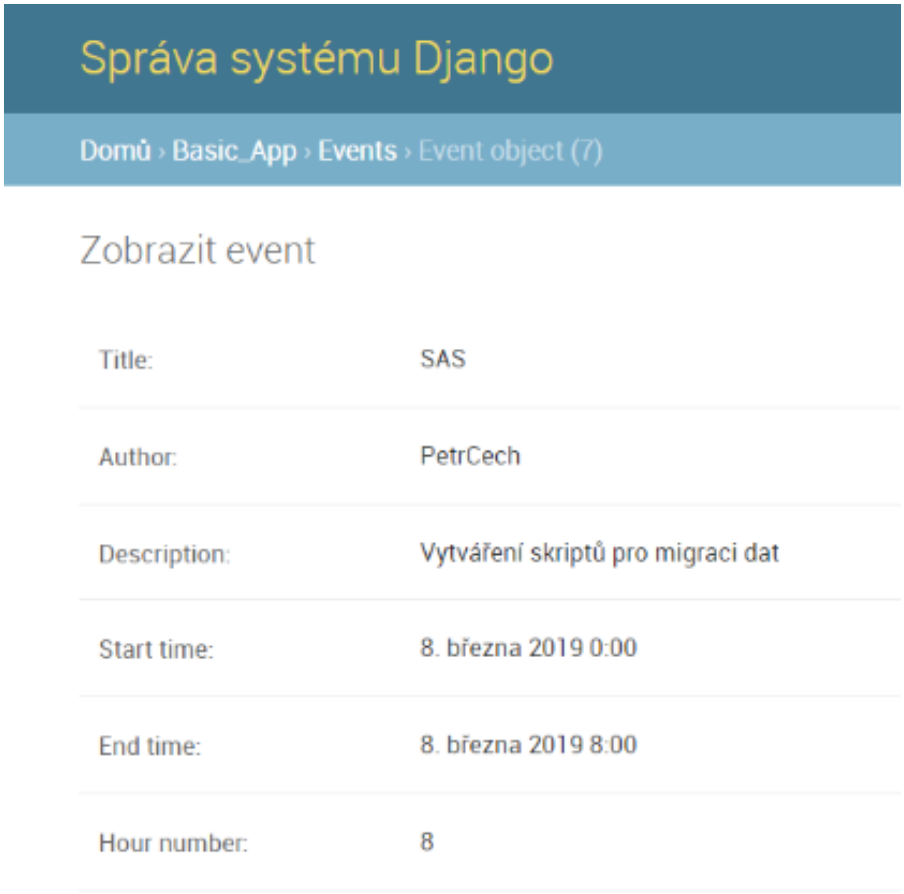
Obrázek 43: Views CalendarView

```
26  class CalendarView(generic.ListView):
27      model = Event
28      template_name = 'calendar.html'
29
30      def get_context_data(self, **kwargs):
31          context = super().get_context_data(**kwargs)
32          d = get_date(self.request.GET.get('day', None))
33          cal = Calendar(d.year, d.month)
34          html_cal = cal.formatmonth(withyear=True)
35          context['calendar'] = mark_safe(html_cal)
36          d = get_date(self.request.GET.get('month', None))
37          context['prev_month'] = prev_month(d)
38          context['next_month'] = next_month(d)
39          return context
40
41
42  def prev_month(d):
43      first = d.replace(day=1)
44      prev_month = first - timedelta(days=1)
45      month = 'month=' + str(prev_month.year) + '-' + str(prev_month.month)
46      return month
47
48
49  def next_month(d):
50      days_in_month = calendar.monthrange(d.year, d.month)[1]
51      last = d.replace(day=days_in_month)
52      next_month = last + timedelta(days=1)
53      month = 'month=' + str(next_month.year) + '-' + str(next_month.month)
54      return month
55
56
57  def get_date(req_day):
58      if req_day:
59          year, month = (int(x) for x in req_day.split('-'))
60          return date(year, month, day=1)
61      return datetime.today()
62
63
64  def event(request, event_id=None):
65      instance = Event()
66      if event_id:
67          instance = get_object_or_404(Event, pk=event_id)
68      else:
69          instance = Event()
70
71      form = EventForm(request.POST or None, instance=instance)
72      if request.POST and form.is_valid():
73          instance=form.save(commit=False)
74          instance.hour_number=int((instance.end_time - instance.start_time).seconds)/3600
75          instance.author=request.user.get_username()
76          form.save()
77          return HttpResponseRedirect(reverse('calendar'))
78      return render(request, 'event.html', {'form': form})
```

zdroj: vlastní tvorba

Jednotlivé výkazy práce je opět možné si prohlédnout ve Správě systému. Nicméně tuto část již není možné nijak upravovat. Krom uživatele SuperUser je nastavena pouze pro čtení. Výstup si lze prohlédnout na obrázku č. 44. Atributy Author a Hour Number jsou už samozřejmě vyplněny.

Obrázek 44: Správa systému - Events



zdroj: vlastní tvorba

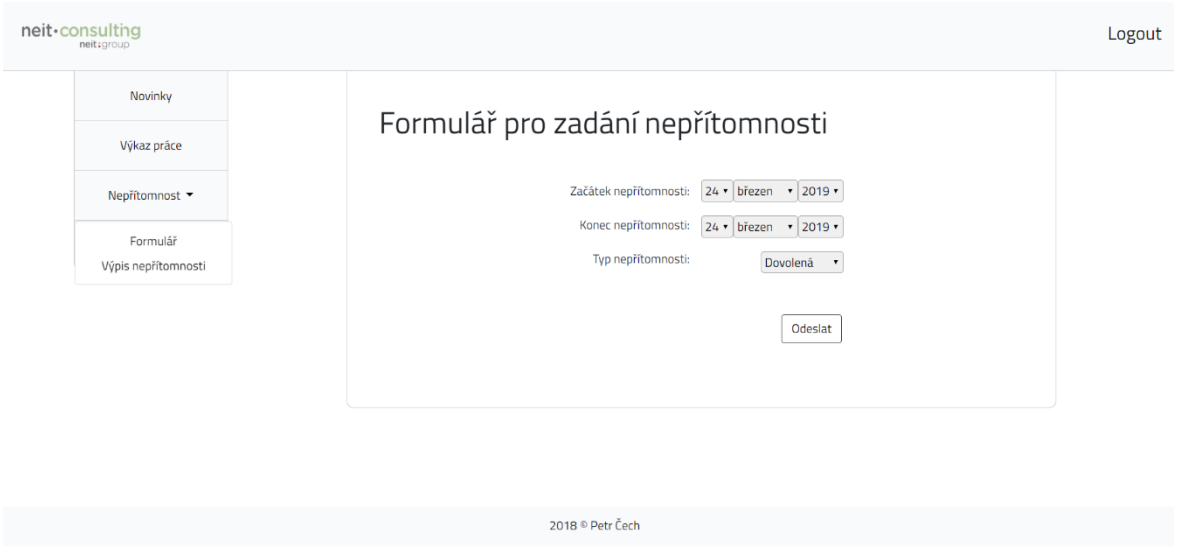
4.3.6. Zadání nepřítomnosti

Pro tento informační systém bylo důležité zajistit i možnost žádat o nepřítomnost. Pod pojmem nepřítomnost si lze představit dovolenou nebo třeba home office. Tato sekce se skládá ze dvou částí. Z formuláře pro zadání nepřítomnosti (obrázek č.45) a z výpisu nepřítomnosti (obrázek č.46).

Nejdříve si rozebereme formulář. Uživatel zde zadává začátek a konec nepřítomnosti, volí typ nepřítomnosti a vše odesílá do systému kliknutím na tlačítko Odeslat. Jestli odeslání

vyplněného formuláře proběhlo v pořádku, si může zkontrolovat na stránce s výpisem nepřítomnosti.

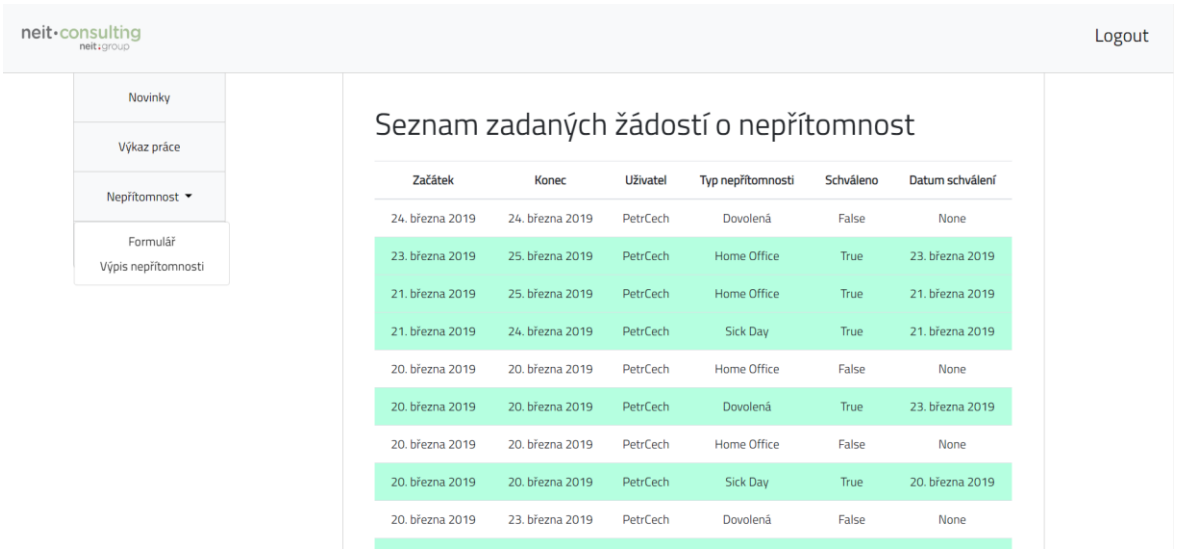
Obrázek 45: Formulář pro zadání nepřítomnosti



zdroj: vlastní tvorba

Zde jsou v tabulce zaznamenány všechny žádosti seřazené podle začátku absence. Důvodem, proč jsou některé řádky vybarvené zeleně je takový, že nepřítomnost schvaluje zaměstnanec s právy minimálně na úrovni managementu. Pokud požadavek schválí, pak se uživateli zobrazí se zeleným pozadím.

Obrázek 46: Výpis žádostí o nepřítomnost



zdroj: vlastní tvorba

Aby informační systém fungoval tak, jak má, byla vytvořena třída AbsenceRequest, opět s několika neviditelnými atributy viz. obrázek níže.

Obrázek 47: Models class AbsenceRequest

```
45  class AbsenceRequest(models.Model):
46      startDate = models.DateField(default=datetime.date.today)
47      endDate = models.DateField(default=datetime.date.today)
48      nickname = models.CharField(max_length=100)
49      requestType = models.CharField(max_length = 100)
50      approval = models.BooleanField(default=False)
51      approvalDate = models.DateField(null=True)
52
53      def __str__(self):
54          tmp1 = str(self.startDate)
55          tmp2 = str(self.approval)
56          return tmp1 + ' - ' + self.nickname + ' '+ self.requestType + ' ' + tmp2
```

zdroj: vlastní tvorba

Nově vzniklá třída AbsenceRequest je ihned využita při tvorbě formuláře NewRequestForm.

Obrázek 48: Forms NewRequestForm

```
32  absence_type = (
33      ('Dovolená','Dovolená'),
34      ('Home Office','Home Office'),
35      ('Sick Day','Sick Day'),
36  )
37
38
39  class NewRequestForm(forms.ModelForm):
40      class Meta:
41          model = AbsenceRequest
42          fields = ['startDate','endDate','requestType']
43          widgets = {
44              'startDate': forms.SelectDateWidget(),
45              'endDate': forms.SelectDateWidget(),
46              'requestType': forms.Select(choices=absence_type)
47          }
```

zdroj: vlastní tvorba

Ve views.py se nachází dvě funkce. Request_form_view má na starost přijmout data z formuláře a pokud jsou validní, pak se uloží do databáze. Request_list vypisuje všechny žádosti o nepřítomnost s filtrem, tudíž každý přihlášený uživatel vidí jen svoje žádosti.

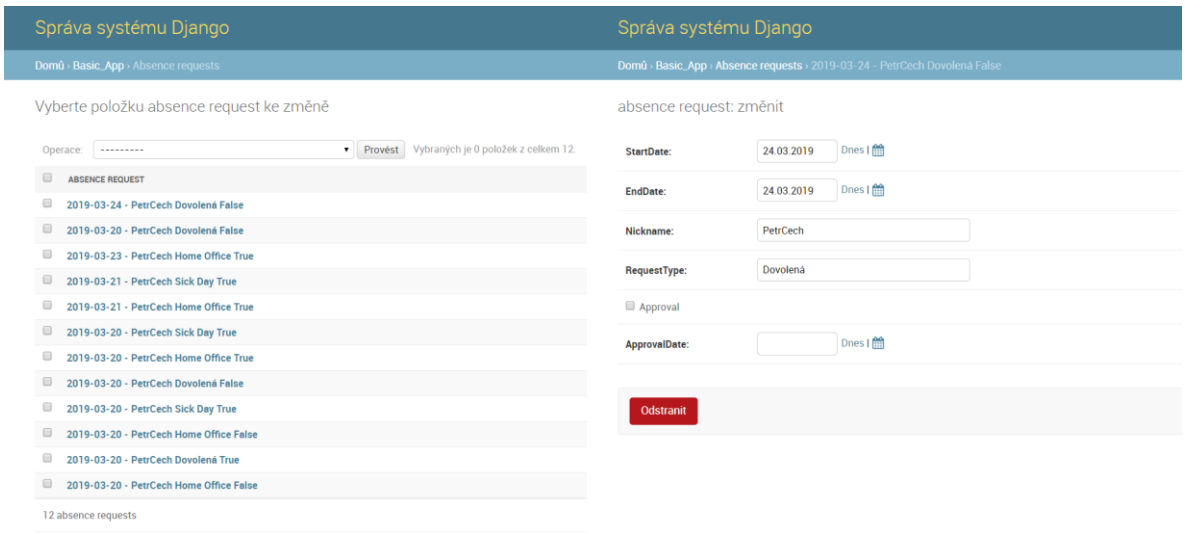
Obrázek 49: Views request_form_view a request_list

```
81  def request_form_view(request):
82      form = NewRequestForm()
83      if request.method == 'POST':
84          form = NewRequestForm(request.POST)
85
86          if form.is_valid():
87              instance=form.save(commit=False)
88              instance.nickname=request.user.get_username()
89              instance.save()
90
91      return render(request,'requests.html',{'form':form})
92
93  #dochazka seznam
94  def request_list(request):
95      logged_user=request.user.get_username()
96      absenceRequests = AbsenceRequest.objects.filter(nickname=logged_user).order_by('-startDate')
97      return render(request,'absence_list.html',{'absenceRequests':absenceRequests})
```

zdroj: vlastní tvorba

Na obrázku č. 50 se v levé části nachází seznam všech zadaných žádostí o nepřítomnost ve Správě systému. V pravé části vidíme detailní pohled na jednotlivou žádost. Právě políčko Approval určuje, zdali bude nepřítomnost zaměstnance přijata nebo ne, přičemž ApprovalDate je nutné vždy vyplnit. Když si uživatel IS rozklikne výpis, pak buď vidí zeleně vybarvený řádek anebo šedý řádek s false hodnotou v sloupci Schváleno a vyplněným Datumem schválení.

Obrázek 50: Správa systému - Absence Requests

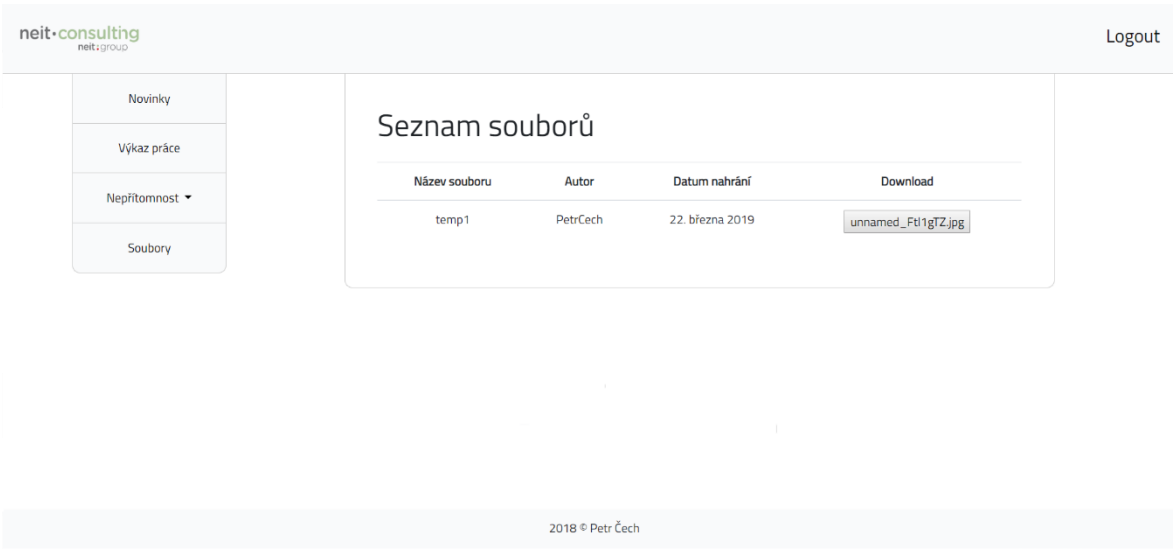


zdroj: vlastní tvorba

4.3.7. Soubory

Toto je poslední stránka informačního systému. Obsahuje seznam souborů, které se dají nahrát pouze ve Správě systému. Seznam obsahuje informace s názvem souboru, autorem, datu nahrání a samozřejmě s tlačítkem pro stažení.

Obrázek 51: Soubory



zdroj: vlastní tvorba

Opět byla vytvořena třída tentokráte s názvem File (obrázek č. 52). Poprvé se ve třídě objevuje FileField(), jenž uživateli umožní nahrávat soubory a u atributu author funkce get_user_model(), která uživateli nabídne při nahrávání souborů vybrat svoje uživatelské jméno jakožto autora souboru.

Obrázek 52: Models class File

```
31  class File(models.Model):
32      filename = models.CharField(max_length=100)
33      author = models.ForeignKey(get_user_model(),
34        on_delete=models.CASCADE)
35      uploadDate = models.DateField(null=True)
36      file = models.FileField()
37
38      def __str__(self):
39          return self.filename + self.author
```

zdroj: vlastní tvorba

Pro výpis a stažení souboru bylo zapotřebí vytvořit dvě funkce. První file_list vypisuje seznam souborů, druhá download_file zajišťuje stažení souboru.

Obrázek 53: Views file_list a download_files

```
100  def file_list(request):
101      file_list = File.objects.all().order_by('-uploadDate')
102      return render(request,'files.html',{'file_list':file_list})
103
104
105  def download_file(request, file_id):
106      cust_file = get_object_or_404(File, pk=file_id)
107      file_name = cust_file.filename
108      file_path = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),"media")
109      wrapper = FileWrapper(open(cust_file.file_path))
110      response = HttpResponse(wrapper, content_type='text/plain')
111      response['Content-Disposition'] = 'attachment; filename=%s' % file_name
112      response['Content-Length'] = os.path.getsize(cust_file.file_path)
113      return response
```

zdroj: vlastní tvorba

Nahrávání souborů je umožněné pouze přes Správu systému. V levé části obrázku níže opět vidíme list nahraných souborů, v pravé části jsou detailněji zobrazeny atributy včetně možnosti nahrát soubor.

Obrázek 54: Správa systému - Files

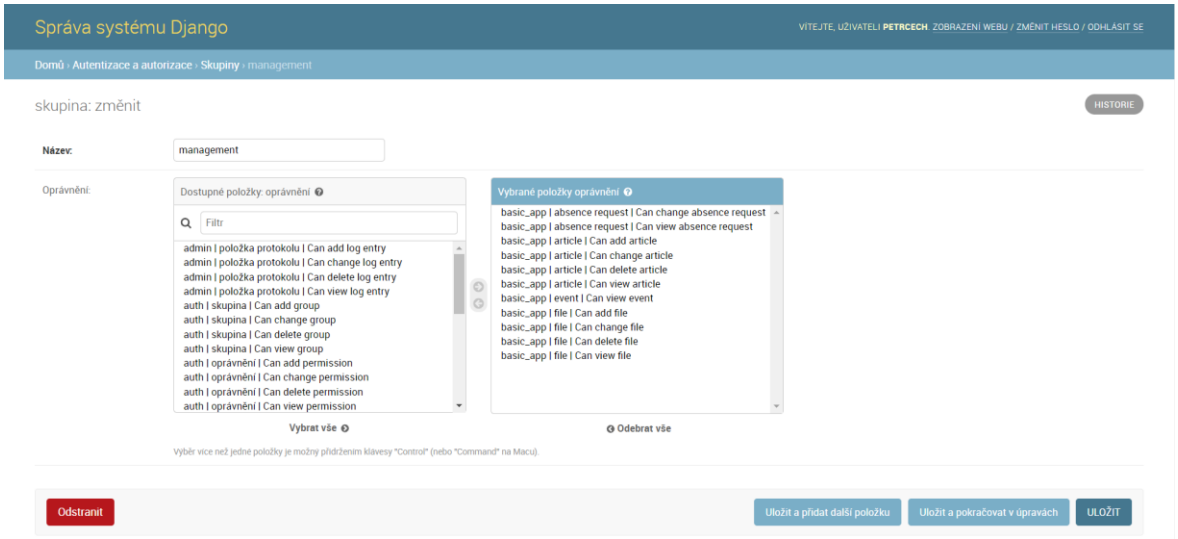
zdroj: vlastní tvorba

4.3.8. Uživatelé a skupiny práv

Jak bylo zadáno, uživatelé informačního systému byli rozděleny do 3 skupin. První skupinou jsou řadoví zaměstnanci. Tato skupina nemá téměř žádná práva. Samozřejmě mohou plně využívat jednotlivé funkce informačního systému, ale jen ty, co jsou dostupné mimo Správu systému.

Druhá skupina je pojmenována jako management a zahrnuje výše postavené zaměstnance. Tento okruh zaměstnanců již má možnost přihlásit se do Správy systémů. V nastavení práv má tato skupina možnost spravovat soubory a články, prohlížet si výkazy práce a žádosti o nepřítomnost, jež mohou i schvalovat (obrázek č. 55.).

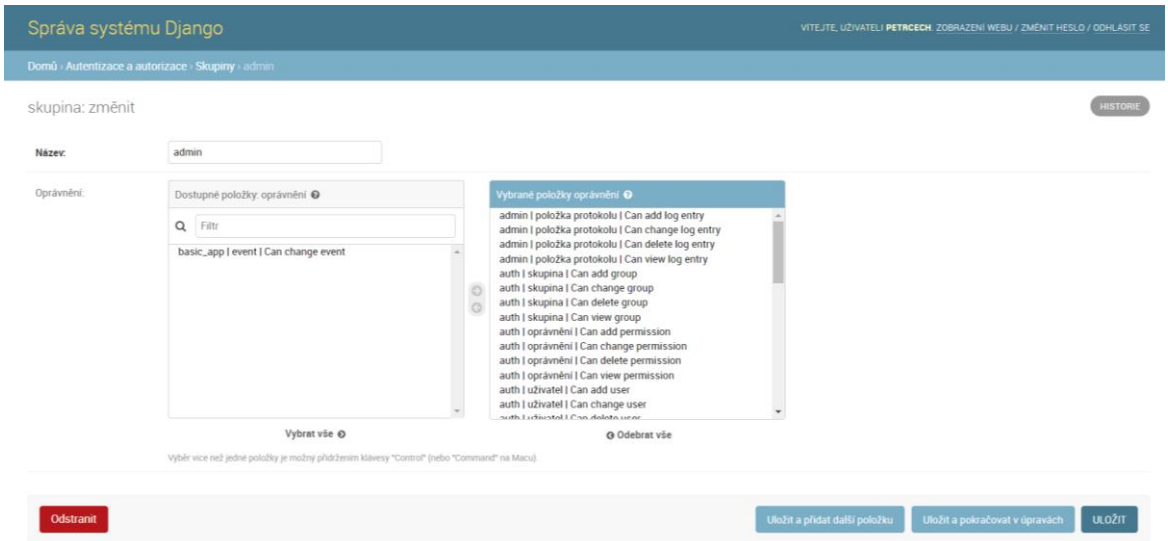
Obrázek 55: Skupina management



zdroj: vlastní tvorba

Poslední skupina svým názvem napovídá, jaká je její funkce. Skupina admin má právo na téměř všechno, kromě úpravy výkazu práce. Hlavní funkcí je správa uživatelů a jejich práv.

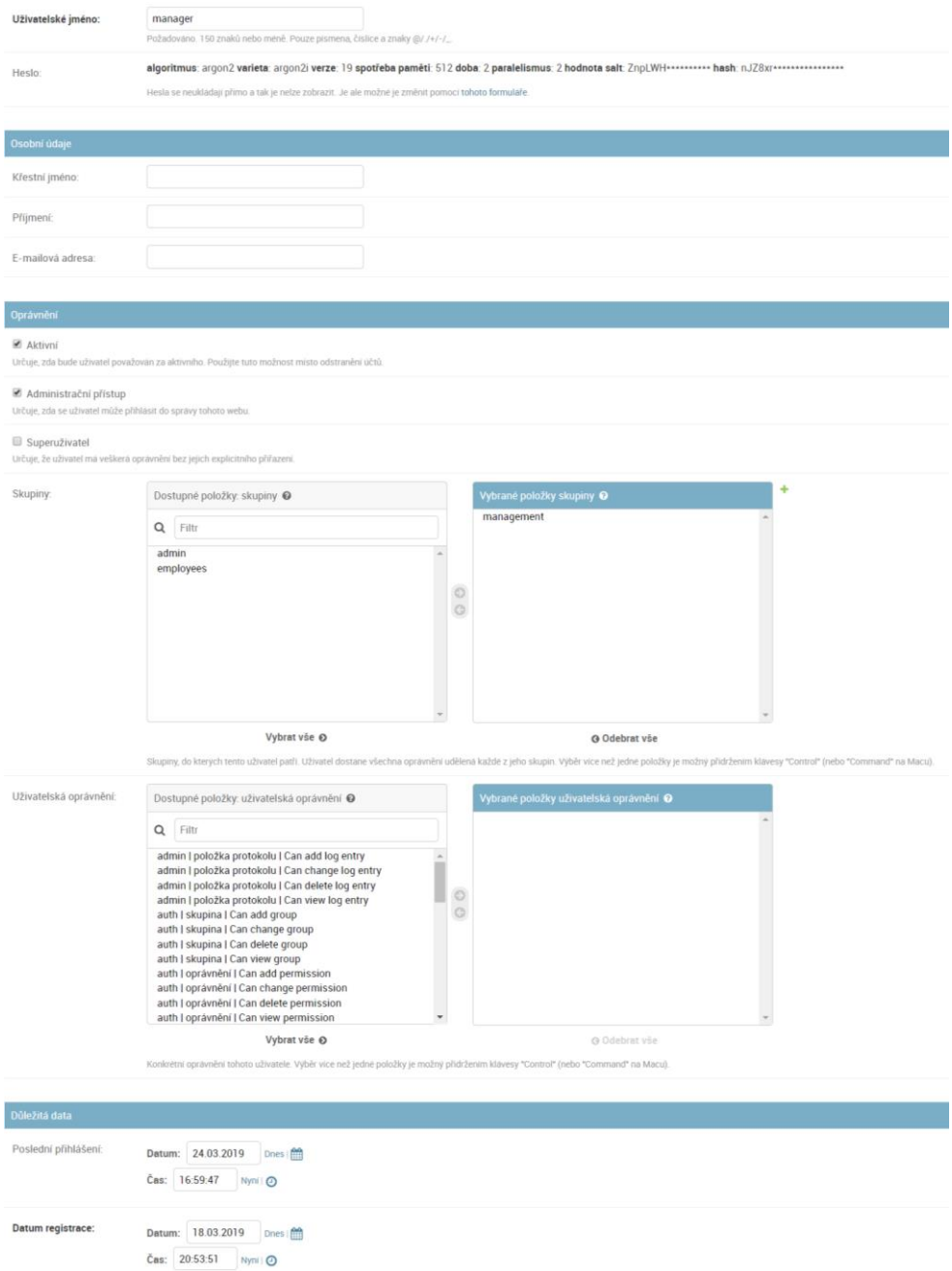
Obrázek 56: Skupina admin



zdroj: vlastní tvorba

Uživatelé jsou zaznamenáni ve Správě systému na stejném místě jako skupiny práv. Zároveň je Správa systému výchozí a jediná možnost, jak upravovat uživatele. Rozkliknutím daného uživatele se zobrazí jeho modifikovatelný přehled. Hlavními prvky jsou uživatelské jméno a heslo. Poté je možné vyplnit křestní jméno, příjmení a emailová adresa. V další části je prostor pro nastavení oprávnění a vše končí podrobnostmi ohledně posledního přihlášení a datu registrace.

Obrázek 57: Správa systému - Uživatel



zdroj: vlastní tvorba

Pokud se detailněji podíváme na heslo, lze spatřit, že je využit bezpečný algoritmus argon2 pro zahashování hesla. Vyžaduje knihovnu argon2-cffi, která závisí na nativním kódu C. Jakým algoritmem se budou hesla hashovat se nastavuje v settings.py (obrázek č. 58)

Obrázek 58: Settings nastavení hashů

```
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.Argon2PasswordHasher',
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.BCryptPasswordHasher',
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
]


AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

zdroj: vlastní tvorba

4.3.9. Testování

Pro otestování informačního systému bylo zvoleno několik praktik. Vynecháno bylo testování jednotek, protože systém neobsahuje velké množství tříd a metod.

4.3.9.1. Testovací scénáře

ID:	**#1**
Název:	Ověření vzniku nového výkazu práce
Kroky:	1. V menu kliknout na Výkaz práce 2. Kliknout na tlačítko Nová událost 3. Vyplnit všechna políčka 4. Kliknout na odeslat
Očekávaný výsledek:	1. V kalendáři se zobrazí nově vzniklý výkaz práce
Provedení testu:	OK

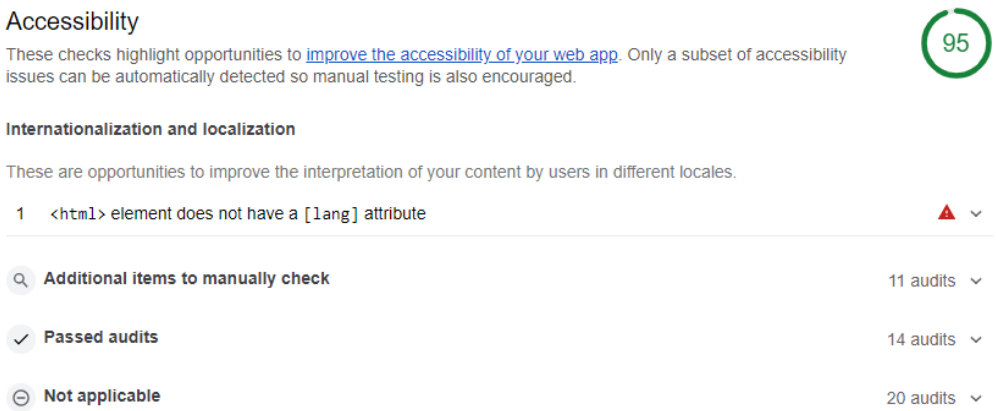
ID:	#2
Název:	Ověření vzniku nové žádosti o nepřítomnost
Kroky:	1. V menu kliknout na Nepřítomnost/Formulář 2. Vyplnit všechna políčka 3. Kliknout na odeslat
Očekávaný výsledek:	1. Na stránce Nepřítomnost/Výpis nepřítomnosti se zobrazí nově vzniklý výkaz práce
Provedení testu:	OK

ID	#3
Název:	Ověření odsouhlasení žádosti o nepřítomnost
Kroky:	1. Přihlásit se do Správy systému informačního systému 2. Kliknout na Absence Requests 3. Vybrat první záznam s hodnotou False a rozkliknout 4. Označit políčko u Approval fajfkou 5. Kliknout na tlačítko Dnes u ApprovalDate 6. Uložit
Očekávaný výsledek:	1. Na stránce Nepřítomnost/Výpis nepřítomnosti se zobrazí odsouhlasená žádost o nepřítomnost zeleně.
Provedení testu:	OK

4.3.9.2. Použitelnost

Pro otestování použitelnosti byl využit speciální plugin dostupný od společnosti Google se jménem Lighthouse.

Obrázek 59: Test použitelnosti



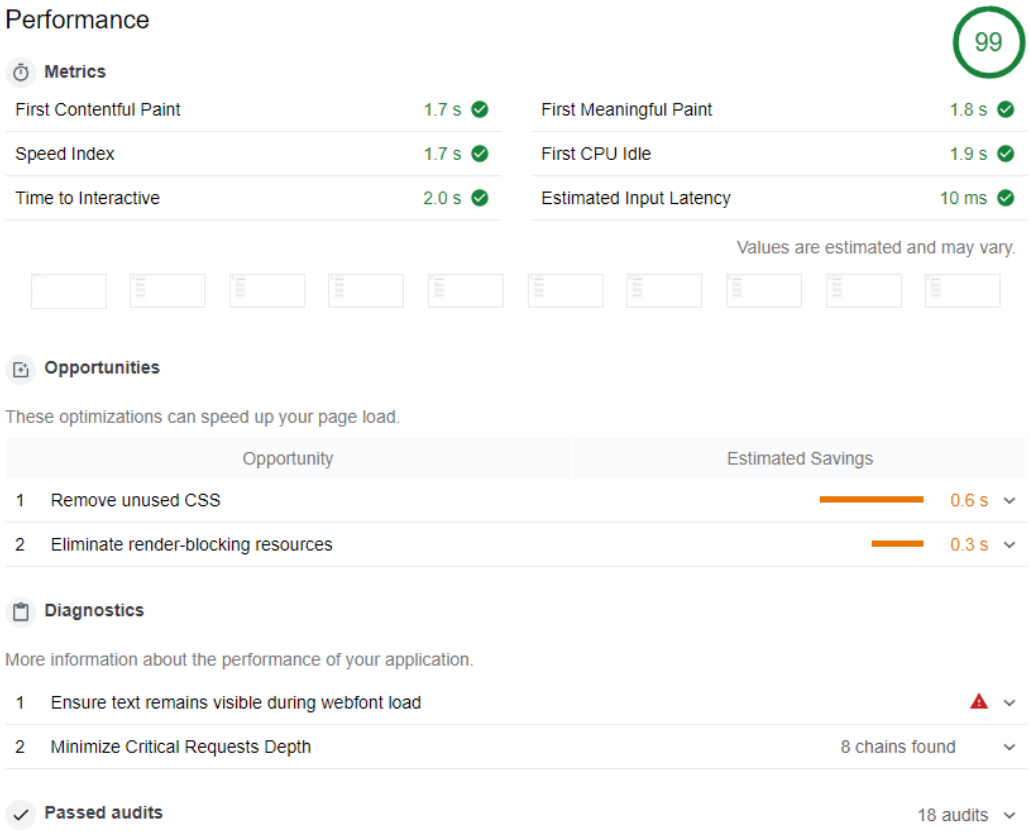
zdroj: Google plugin Lighthouse

Výsledkem testu bylo krásných 95 %. Jedinou výtkou bylo neumístění atributu lang do HTML souboru. Lang je universální atribut a určuje jazyk, ve kterém je příslušný obsah napsán. Tento nedostatek byl opraven.

4.3.9.3. Výkon

Pluginem Lighthouse byl otestován i výkon. Výsledek byl ještě o něco příznivější než test použitelnosti a to konkrétně 99 %. Avšak v této fázi není výsledek moc směrodatný, neboť informační systém neobsahuje provozní data.

Obrázek 60: Test výkonu



zdroj: Google plugin Lighthouse

4.4. Nasazení a provoz

Společnosti Neit Consulting s.r.o. byl předán materiál potřebný pro nasazení a provoz informačního systému. Uživatelé byli proškoleni. Nadále bude poskytována podpora IS pro kritické momenty.

5. Závěr

V této diplomové práce byl vytvořen ucelený pohled na informační systém a na vše spjaté s tímto tématem. Představena byla architektura IS, ale i etapy životního cyklu nebo metodiky vývoje IS. Kromě toho byly porovnány vybrané frameworky a zhotoven webový informační systém pro organizaci Neit Consulting s.r.o.

Pro přístup na webový informační systém je nutné přihlášení, jinak uživatel nebude moci si obsah IS zobrazit. Po přihlášení je uživatel přesměrován na hlavní stránku. Tato stránka je složená z interních článků. Informační systém dále nabízí možnost výkazu práce, žádosti o nepřítomnost a práci se soubory. Byly vytvořeny 3 úrovně přístupu: zaměstnanci, management a admin. Zaměstnanci jsou odkázáni pouze na funkce, které se nacházejí na webových stránkách. Management a admin mají navíc přístup do Správy systému, odkud mohou přidávat články, schvalovat žádosti o nepřítomnost a podobně. Vytvářet nové uživatele a upravovat práva smí pouze admin.

Výsledkem této diplomové práce je plně funkční a otestovaný webový informační systém předaný do rukou společnosti. Uživatelé byli proškoleni. Nadále bude poskytována podpora IS pro kritické momenty.

6. Seznam použitých zdrojů

[1] SUMMERFIELD, Mark. Python 3: výukový kurz. Brno: Computer Press, 2010. ISBN 978-80-251-2737-7.

[2] PILGRIM, Mark. Ponořme se do Pythonu 3. 2017. Praha: CZ.nic, 2017. ISBN 978-80-904248-2-1.

[3] ALTHAMMER, Egbert & PREE, W. Design and Implementation of a MVC-Based Architecture for E-Commerce Applications. International Journal of Computers and Applications (2001).

[4] FORCIER, J. Python Web development with Django. NJ: Addison-Wesley, 2009. ISBN 9780132356138.

[5] IT Network. IT Network [online]. Česká republika: IT Network, 2013 [cit. 2017-01- 03]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-uvod-historie-vyznam-a-diagramy

[6] IT Network. IT Network [online]. Česká republika: IT Network, 2013 [cit. 2017-01- 03]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram

[7] MOLNÁR, Zdeněk. Podnikové informační systémy. Praha: ČVUT, 2009. 195 s. S. 13.

[8] STAIR, Ralph M. a George Walter REYNOLDS. Principles of information systems: a managerial approach. 9th ed. United States: Course Technology Cengage Learning, c2010. ISBN 0324665288.

[9] BRUCKNER, Tomáš. Tvorba informačních systémů: principy, metodiky, architektury. Praha: Grada, 2012. Management v informační společnosti. ISBN 978-80-247-4153-6.

[10] Informační systém. Informační systémy [online]. Brno, Česká republika: Mendelu, 2015 [cit. 2017-03-14]. Dostupné z: https://akela.mendelu.cz/~rybicka/prez/infsyst.pdf

[11] International journal of enterprise information systems Volume 5 Issue 1. Hershey, Pa.: Idea Group Pub., 2009. ISSN 1548-1115

[12] ISAIAS, Pedro, a TOMAYESS Issa. High Level Models and Methodologies for Information Systems. New York, Springer New York, 2015. ISBN 978-1-4614-9254- 2

[13] IJCSI. IJCSI [online]. India: IJCSI, 2010 [cit. 2017-02-11] Dostupné z: http://www.ijcsi.org/papers/7-5-94-101.pdf

[14] HOPCROFT Phillipa J. & BROADFOOT Guy H. Combining the Box Structure Development Method and CSP for Software Development (2005).

[15] BUCHALCEVOVÁ, Alena. Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky. Praha: Grada, 2005. Management v informační společnosti. ISBN 80-247-1075-7.

[16] BEYNON-DAVIES, Paul & MACKAY. Rapid application development (RAD): An empirical review. European Journal of Information Systems (1999).
[17] AMBLER, Scott W. A Manager’s Introduction to The Rational United Process (2005).
[18] AMBLER, Scott. W. The Agile Unified Process (2019).
[19] STAPLETON Jennifer, DSDM: The Method in Practice. Boston, MA, USA 1997. ISBN:0201178893
[20] JONES, Capers. Software methodologies: A quantitative guide. Boca Raton: CRC press, 2017. ISBN 9781138033085
[21] SUTHERLAND, Jeff. Jeff Sutherland's Scrum Handbook (2010).
[22] AHMAD, Muhammad O. & MARKKULA, Jouni & OIVO, Markku. Kanban in Software Development: A Systematic Literature Review (2013).
[23] YUNG, Zakhar. Python vs. Ruby vs. Node.js. Railsware [online]. 2018 [cit. 2019-03-28]. Dostupné z: https://railsware.com/blog/2018/06/13/python-vs-ruby-vs-node-js-which-platform-is-a-fit-for-your-project/

7. Seznam obrázků