Príloha A: Metodiky

Metodika tvorby dokumentácie

Slovník pojmov

Žiadne významné pojmy.

Účel metodiky

Táto metodika popisuje postup, akým sa vytvára dokumentácia. Dokumentovanie zahŕňa všetky typy dokumentov zahrnutých v dokumentácií k inžinierskemu dielu a v dokumentácií k riadeniu.

Dedikácia metodiky

Metodika je určená každému členovi tímu, ktorý sa zúčastňuje pri tvorbe dokumentácie.

Rola	Úloha
Správca dokumentácie	Spracúva dodané dokumenty do výsledných dokumentácií
Akýkoľvek člen tímu	Vytváranie nových textov

Informácie pre vytváranie dokumentácie

Hlavné kapitoly majú nadpis 1, podkapitoly majú nadpis 2. Ďalšie nadpisy sú na uvážení tvorcu dokumentácie. Číslovanie umiestňujeme na spodnú pravú stranu. Prílohy majú samostatné číslovanie strán.
Použité číslovanie je v tvare:

1. Prvý záznam
    a. Prvý podzáznam
2. Druhý záznam
    a. prvý podzáznam
    b. druhý podzáznam

Odrážky majú nasledovný tvar:

- prvý záznam
    - prvý podzáznam
- druhý záznam
    - prvý podzáznam
    - druhý podzáznam

Typ písma je Calibri, veľkosť 11.
Pri tvorbe nového dokumentu je nutné zvoliť správnu hlavičku ktorá obsahuje názov univerzity, fakulty, názov témy, názov tímu, vedúceho tímu a členov tímu.
Použité skratky je nutné vysvetliť v kapitole „Použité skratky“. Použité cudzie alebo technické výrazy je nutné vysvetliť v kapitole „Použité výrazy“.
Pre citácie a bibliografické údaje sa používa norma ISO 690 a ISO 690-2.

Používaný formát je docx. Tento formát sa pri odovzdaní konvertuje na pdf. Pred odovzdaním a tlačou je nutné skontrolovať výzor dokumentácie.

Metodika tvorby metodík

Účel metodiky

Táto metodika popisuje postup, akým sa vytvárajú nové metodiky do dokumentácie k riadeniu. Táto metodika je zároveň vzorovým príkladom pre tvorbu metodík.

Dedikácia metodiky

Metodika je určená každému členovi tímu, ktorý vytvára ďalšie metodiky.

Rola	Úloha
Správca dokumentácie	Spracúva metodiky do výslednej dokumentácie k riadeniu
Akýkoľvek člen tímu	Vytváranie nových metodík

Postup pre vytvorenie metodiky

1. Hlavný názov metodiky - nadpis 2
2. Sekundárne nadpisy - nadpis 3
3. Vytvorenie účelu metodiky
4. Dedikácia metodiky a vytvorení rolí
5. Popis potrebných zdrojov
6. Popis krokov pre dosiahnutie výsledku
7. Popis očakávaného výsledku
8. Pridanie použitých pojmov do *Slovníka pojmov* na začiatku dokumentu v prípade, že metodika obsahuje cudzie slová alebo technické a odborné výrazy
9. Pridanie použitých skratiek do *Použitých skratiek* v prípade, že sa v metodike nachádzajú skratky

Metodika tvorby testovacích prípadov

Účel metodiky

Táto metodika popisuje postup, akým sa vytvárajú testovacie prípady. Zápis testovacích prípadov môže mať 2 formy. Prvá je scenár pre automatické testovanie a druhá je tvorba manuálneho testu. Táto metodika popisu oba postupy.

Dedikácia metodiky

Metodika je určená každému členovi tímu, ktorý vytvára testovacie scenáre.

Rola	Úloha
Akýkoľvek člen tímu	Vytváranie nových testovacích prípadov

Postup pre vytvorenie testovacieho prípadu

- Naštudovanie testovanej funkcionality
- Vytvorenie popisu testu
    - Základné kroky, potrebné pre vyhľadanie testovanej oblasti
    - Popis funkcionality, ktorá sa testuje a jej správanie
    - Prípadné doplňujúce údaje – napr. je potrebná tlačiareň, je potrebný mobilný telefón, test sa vykonáva na produkčnom prostredí, prihlásenie možné len v určenom čase,...
- Kroky sa opisujú ako akcia, ktorú je nutné vykonať, testovacie dáta nad, ktorými sa pracuje (ak sa vykonáva s nejakými danými dátami) a očakávaný výsledok kroku
- Je vhodné pridávať obrazovky do testovacích prípadov ak ide o akceptačný test

Metodika ohlasovania chýb v systéme Jira

Slovník pojmov

Pojem	Vysvetlenie
Jira	Webová aplikácia, v ktorá nám pomáha sledovať vývoj produktu, úlohy a chyby.

Účel metodiky

Táto metodika slúži ako návod na ohlásenie chýb. V tejto metodike je popísaný spôsob a potrebné informácie, ktoré by mal vývojár dostať.

Dedikácia metodiky

Metodika je určená každému členovi tímu, ktorý vytvára ďalšie chyby. Túto funkciu väčšinou zastáva tester.

Rola	Úloha
Hlavný vývojár	Spracúva chyby a priradzuje ich na riešenie ďalším členov tímu. Po uvážení môže chybu riešiť aj on sám.
Akýkoľvek iný vývojár	Oprava chyby, ktorú mu priradí hlavný vývojár.
Tester	Osoba, ktorá vykonáva testy a analyzuje údaje z automatických testov.

Postup pre ohlásenie chyby v systéme Jira

- **Summary** - názov v skratke popisuje chybu. Názov by mal byť čo najviac objasniť typ chyby - napr. pri zadní kódu cvičenia sa zobrazí neočakávaná stránka.
- **Priority** – priorita vyjadruje ako veľmi je riešenie chyby naliehavé.
    - Blocker - túto prioritu je vhodné použiť len ak funkcionalita blokuje akékoľvek ďalšie testovanie (napr. nefungujúce prihlásenie)

    - Critical - používa sa ak je chyba závažná, ale umožňuje čiastočné testovanie ostatných funkcionalít (napr. nefungujúce odoslanie testu)
    - Major - chyba je stále závažná, ale ovplyvňuje len jeden test (napr. test sa po odoslaní nesprávne vyhodnotil)
    - Minor - chyba v zobrazení (napr. pretekajúci text)
    - Trivial - kozmetická chyba (napr. preklep v slove)
- **Assignee** - volí sa hlavný vývojár a ten preradí úlohu podľa potreby
- **Environment** - typ testu, podľa toho kedy sa udial
    - Unit- prvotné testovanie pri nasadzovaní
    - Modulové - testovanie funkčnosti a výzoru testermi
    - Akceptačné- testovanie funkčnosti reálnymi používateľmi
- **Description** - opis chyby sa skladá z minimálne 3 častí a môžu byť doplnené ešte ďalšie informácie okrem troch vyžadovaných častí, čas kedy sa chyba prejavila a ďalšie informácie
    - Popis chyby - čo najpodrobnejšie popísať ako sa chyba prejavuje
    - Scenár pri akom sa chyba udiala
    - Očakávaný výsledok na konci scenára a reálny výsledok
- **Attachment** - je vhodné vložiť obrázok alebo video prejavu chyby, prípadne dôležitá informácie z vývojárskej konzoly prehliadača.
- **Labels** - použitie značiek je voliteľné, je vhodné napríklad keď chceme oddeliť chyby pochádzajúce z migrácie od ostatných chýb.

Metodika vytvárania úloh

Slovník pojmov

Pojem	Vysvetlenie
Jira	Proprietárny softvér na sledovanie úloh a chýb
Šprint	Šprint je základná jednotka vývoja v Scrume
Epic	Epic je združenie viacerých príbehov, ktoré spolu súvisia.
Story Points	Základná jednotka ohodnotenia náročnosti úlohy.
Scrum	Agilná metodika vývoja softvéru.

Účel metodiky

Táto metodika popisuje postup, ako sa vytvárajú a prideľujú nové úlohy v tímovom projekte členom tímu pomocou nástroja Jira.

Dedikácia metodiky

Rola	Úloha
Správca Jiry	Štartuje, kontroluje a ukončuje šprint. Kontroluje stavy a popisy úloh.
Ktorýkoľvek člen tímu	Vytváranie a riešenie úloh
Zákazník	Vytváranie a uzatváranie úloh

Úlohy

Úloha je naplánovaná činnosť, ktorú je nutné v rámci tímového projektu vykonať. V tímovom projekte rozlišujeme viaceré typy úloh:

- Príbeh (angl. Story) – Príbeh sa využíva v plánovaní šprintov, ako úloha, ktorá má byť po ukončení šprintu vykonateľná vo vyvíjanom systéme.
- Chyba (angl. Bug) – Chyba je odhalená chyba systému, ktorá vyžaduje zmenu vo vyvíjanom systéme.
- Úloha (angl. Task) – Ak vytváraná úloha nie je príbeh ani chyba, napríklad úprava dokumentácie či kontrola logov systému alebo nasadenie na server, jedná sa o všeobecnú úlohu.

Kedy sa vytvárajú?

Väčšina príbehov sa vytvára pred začatím šprintu, pričom počas šprintu sa môže ukázať, že do šprintu budú zahrnuté aj iné príbehy. Správca Jiry má za úlohy vytvoriť a držať v aktuálnosti *BackLog* (zoznam príbehov, ktoré zákazník vyžaduje v dokončenom systéme). Z *BackLogu* sa vyberajú príbehy na riešenie. Zvyšné úlohy sa vytvárajú počas celej doby behu projektu.

Ako ich tvoriť?

Úlohy v tomto tímovom projekte sa vytvárajú v nástroji Jira, dostupnom na adrese jira.fiit.stuba.sk . Pri vytváraní úlohy je najprv nutné zvoliť zodpovedajúci typ úlohy a následne zadať názov úlohy a popis úlohy. V prípade tvorby príbehu je nutné zvoliť zodpovedajúci epic, odhadovaný čas, prípadne Story Points a zvoliť riešiteľa. Doplnkovo je možné zadať prostredie, ktoré má súvis s úlohou, napríklad tímový server pri chybe dátového generátora. V prípade, ak je priorita dôležitá z hľadiska riešenia tvorenej úlohy, je možné ju zvoliť.

Ohodnocovanie príbehov

Príbehy sú ohodnocované pomocou Story Points podľa metodiky Scrum. Na začiatku projektu ale vzhľadom na prednášku o Scrume a Story Points, ktorá bola až v neskoršej fáze semestra, boli príbehy ohodnocované predpokladaným počtom hodín strávených riešením úlohy. Ne-príbehové úlohy sú stále ohodnocované predpokladaným počtom hodín.

Riešenie úloh

Pri riešení úloh je nutné vždy zaznamenať a popísať vykonanú prácu. V prípade, ak sa úloha riešila vo viacerých súvislých časových intervaloch, Je vhodné počas riešenie písať aj komentáre hodnotiace aktuálny stav riešenie a fázu riešenia.

Pokiaľ úloha vyžaduje spoluprácu viacerých členov tímu, hlavný riešiteľ úlohy úlohu neprehadzuje na člena, od ktorého vyžaduje pomoc, ale vytvorí pod-úlohu s popisom, ktorú priradí na člena tímu. Tento spolupracujúci člen tímu v pod-úlohe tiež zaznamenáva strávený čas.

Priebeh šprintu

V tímovom projekte postupujeme metodikou Scrum, pričom pokiaľ sa nestanoví inak, tak pracujeme v dvoj-týždňových šprintoch. Štart a ukončenie šprintu prebieha na stretnutí celého tímu. Na tomto stretnutí sa tiež volia úlohy do šprintu a kontroluje sa ich stav riešenia. Úlohy si vyberajú členovia tímu na základe oblastí pôsobenia členov tímu, každý člen tímu má viaceré zamerania, pričom sa prikladá dôraz aj na zaťaženie jednotlivých členov, aby bolo približne rovnomerné a samozrejme zákazníkových požiadaviek. Počas šprintu sa vyberá len hlavný riešiteľ, často-krát počas riešenia sa zistí, že je nutná spolupráca viacerých členov, vtedy sa tvoria pod-úlohy. Zákazník po skončení šprintu si úlohy preberie a uzatvorí, ak je spokojný s riešením, inak ich znovuotvorí.

Odhadovanie a vykazovanie času

Ako bolo spomenuté, členovia tímu sú povinný vykazovať strávený čas počas riešenia úloh tímového projektu. Počas časových odhadov sa odhadovaný čas odhadoval tak, že sa doňho započítal všetok strávený čas na úlohe, t.j. všetok strávený čas na úlohe a jeho pod-úlohách. Hlavný riešiteľ úlohy si čas vykazuje k úlohe a zvyšní členovia k jednotlivým pod-úlohám úlohy. Nástroj Jira následne spočíta logovaný čas strávený na úlohe a pod-úlohách a ten ukáže ako výsledný strávený čas.

Metodika dokumentovania úloh

Slovník pojmov

Pojem	Vysvetlenie
Úloha (task)	Úloha, ktorú člen tímu vykonával. Je popísaná niekoľkými vetami a je zaznamenaná v systéme Jira.
Šprint (sprint)	Je krátka časová perióda, ktorá má priniesť vždy nový míľnik vývoja.
Používateľský príbeh (user story)	Je to krátky príbeh, ktorý opisuje novú používateľskú aktivitu.
Jira	Systém slúžiaci na zaznamenávanie práce na projekte.

Účel metodiky

Táto metodika popisuje spôsob ako dokumentovať úlohy a šprinty. Metodika je návodom k tvorbe kapitoly „Moduly systému“.

Dedikácia metodiky

Metodika je určená každému členovi tímu.

Rola	Úloha
Správca dokumentácie	Spracúva dokumentované úlohy a šprinty do výslednej dokumentácie inžinierskeho diela
Riešiteľ úlohy	Dokumentovanie úloh, na ktorých pracoval

Časti dokumentácie úlohy

- Názov používateľského príbehu
- Analýza
    - Opis príbehu
    - Počiatočný bod používateľského príbehu a ako sa k nemu dostaneme
    - Očakávaný priebeh príbehu
    - Očakávaný koniec používateľského príbehu
    - Alternatívne scenáre (voliteľné)
- Návrh (Nie vždy je dostupné)
    - Možné dostupné riešenia v dobe práce na príbehu
- Riešenie
- Testovanie

Postup pre dokumentovanie úlohy

1. Zapísanie názvu používateľského príbehu
2. Zapísanie krátkeho opisu
3. Opis všetkých častí
4. Zaslanie zdokumentovanej úlohy správcovi dokumentácie

Dodatočné informácie

- Testovanie požiadaviek je v našom prípade sprostredkované prototypom.
- Opis návrhu by nemal byť príliš technický
- Nie vždy existuje návrh – napr. používateľ sa prihlási (návrh už vymysleného nemá zmysel)
- Návrh stačí aj stručný, ale musí byť vecný
- Opísať akým bol používateľský príbeh technicky realizovaný
- Testovanie opisuje spôsob akým bolo riešenie overené

Metodika verziovania a udržiavania zdrojových kódov

Úvod

Táto metodika patrí pod manažment podpory a vývoja. Je určená pre všetkých členov Osičiek a pre prípadných iných prispievateľov do nášho projektu TESA. Cieľom tejto metodiky je určenie pravidiel pre verziovanie zdrojových kódov a udržanie konzistentnosti. Na účel verziovania používame v našom projekte VCS s úložiskom v službe github.

Pri písaní tejto metodiky sa inšpirujeme odporúčaniami z metotológie písanej pre službu github.

Použité skratky a použité pojmy

Skratka/pojem	Vysvetlenie
VCS	(version control system) – systém pre udržiavanie verzií
PMS	(project management system) – systém na manažment projektu (v našom prípade Atlassian Jira)
github	služba pre webovo orientovaný git repozitár
branch	vetva kódu
commit	odovzdanie zmien v kóde

Predpoklady

Pre zvládnutie a pochopenie tejto metodiky sa očakáva základná znalosť git-u a jeho príkazov.

Vetvenie kódu

Základné typy vetiev kódu

Sú rôzne štýly vytvárania a rozdeľovania vetiev verzií kódu. Jednou z nich je pre každý nový task - úlohu v PMS urobiť aj novú vetvu s príslušným označením a až následné spojenie do vetvy developu – vývojovej vetvy. Táto metóda má svoje výhody keďže je to prehľadnejšie v rámci úloh a ich spracovaní, ale je aj náročnejšia na manažment. Hodí sa viac do väčších tímov, kde je veľa developerov kódu a vzniká tam väčšie riziko kolízií (keď viacerí developeri upravujú tú istú časť kódu a verziovací systém potom nevie, ktorá z nich je tá správna).
Ďalšou možnou alternatívou je použiť na celý šprint vývojovú vetvu develop. Konkrétne tasky – úlohy sa označujú k commitoch aj spolu s označeniami z PMS. Na konci každého šprintu sa táto otestovaná verzia spojí s doterajšou verziou v mastrovi – hlavnej vetve. Tá sa následne nasadí na produkčný server. V prípade dokončenia verzie projektu, ktorú chceme ponechať nemennú sa používa tagovanie. To slúži na uloženie si aktuálnej verzie projektu bez možnosti jej zmeny. Táto druhá metóda je ľahšia na údržbu a je

veľmi výhodná presne pre menšie tímy s nižšou pravdepodobnosťou kolízií. Preto túto metódu používame aj my v našom projekte.

Pri vývoji projektu si udržujeme tieto základné vetvy:

- osicky_master (hlavná vetva)
    - táto vetva obsahuje spustiteľnú otestovanú verziu nasadenú na produkcii
    - táto vetva musí byť stále plne funkčná
- osicky_develop (vývojová vetva)
    - odzrkadľuje aktuálny stav projektu
    - je nasadená na testovacom prostredí
    - po otestovaní sa spája do hlavnej vetvy

Pravidlá pre odovzdávanie (commits)

Odovzdávanie (angl. commits) sa má týkať jednej zmeny. Čím menšia bude tá zmena, tým bude história zmien kódu prehľadnejšia. Každá zmena môže ovplyvňovať viacero súborov, ale všetky tieto úpravy spolu majú súvisieť, to znamená týkať sa jednej úlohy z PMS. Pri každom odovzdávaní sa píšu doprovodné správy týkajúce sa zmeny. Musia sa riadiť týmito pravidlami:

- každé odovzdanie musí obsahovať správu o vykonanej zmene
- správy musia byť písané v anglickom jazyku
- správa musí stručne, ale vecne obsahovať popis vykonanej zmeny
- správa musí byť vo formáte id z PMS : popis zmeny, napr. ,,TESA12: added login“

Postup pridania zmeny

Predpokladáme, že developer, ktorý chce pridať zmenený kód do nášho projektu má skloňovaný náš projekt z github-u spolu s jeho repozitárom.

1. Pred každým vykonaním zmien v projekte je potrebné si stiahnuť najnovšiu verziu v danej vetve príkazom *git pull*. Je to potrebné z dôvodu možných nových vykonaných zmien od iného developera, ktorý pracuje na projekte. Je potrebné, aby každý člen pracoval s najaktuálnejšou verziou (pre zamedzenie kolízií).

2. Keď programátor vykoná zmeny, s ktorými je spokojný a chce ich pridať do spoločného repozitára, potrebuje si pozrieť čo vlastne menil – *git status.* Tam sa mu zobrazia zmenené súbory projektu jeho verzie s tou v globálnom repozitári. Ak je s nimi spokojný a všetko si skontroloval zadá *git add –A* (pridá všetky zmenené súbory do commit-u, ak chce pridať len niektoré špecifikuje to na konkrétne súbory napr. *git add --/file/file.java*).

3. Nasleduje odovzdanie tejto zmeny – commit. To sa vykoná príkazom *git commit –m “správa o zmene“.*

4. Posledným krokom je ,,zatlačenie“ tejto zmeny do globálneho repozitára na danú vetvu príkazom *git push*. Pokiaľ nenastali kolízie, tak je všetko dokonané a všetky zmeny sa nachádzajú

v globálnom repozitári. Ak nastali kolízie musia sa riešiť ich ručným opravením a určením si, ktorá verzia je tá správna. Táto oprava sa robí samotnými developermi, ktorí vedia čo a prečo to menili. Ak by nastali komplikovanejšie problémy, tak ich rieši samotný git-master tímu.

Metodika tvorby zápisníc

Slovník pojmov

Pojem	Vysvetlenie
Jira	Proprietárny softvér na sledovanie úloh a chýb
Šprint	Šprint je základná jednotka vývoja v Scrume
Epic	Epic je združenie viacerých príbehov, ktoré spolu súvisia.

Účel metodiky

Táto metodika popisuje postup, akým sa vytvárajú zápisnice zo stretnutí projektového tímu.

Dedikácia metodiky

Rola	Úloha
hlavný zapisovateľ zástupca hlavného zapisovateľa v jeho neprítomnosti	Vytváranie zápisníc zo stretnutia

Informácie pre vytváranie zápisníc

Zápisnica je ucelený dokument zaznamenávajúci celý priebeh tímového stretnutia, resp. pre projekt relevantných častí. O jednotlivých bodoch stretnutia je potrebné zaznamenávať vecne dôležité informácie, pričom je potrebné prihliadať na fakt, že zápisnica môže byť pre členov tímu dôležitá aj s väčším časovým odstupom od stretnutia. Preto je potrebné jednotlivé body opísať v dostatočnej miere, ktorá rešpektuje tento fakt. Pri písaní zápisnice treba rešpektovať aj fakt, že zápisnica je formálny dokument. Obsahom by nemali byť niektoré výrazy a oslovenia spadajúce do hovorového štýlu komunikácie.

Formát zápisnice

Dokument zápisnice je textový dokument vo formáte docx alebo pdf, ktorý sa skladá z hlavičky a tela dokumentu.

Hlavička dokumentu

V hlavičke každej strany zápisnice sa nachádza základný identifikátor tímu. Obsahuje logo tímu, názov tímu a jeho číslo v rámci predmetu Tímový projekt, názov školy a fakulty a akademický rok.

Telo dokumentu

Telo dokumentu sa skladá z niekoľkých častí. Na začiatku dokumentu sa nachádza názov dokumentu (*Zápisnica zo stretnutia tímu [nový riadok] zo dňa ...*), *miesto a čas stretnutia a zoznam účastníkov stretnutia*. Veľkosti písma pre názov dokumentu je potrebné zvoliť podľa uváženia prehľadnosti, dátum v druhom je vhodné zvoliť menší a bez voľby tučným (odporúčaná veľkosť 14, pre názov veľkosť 26).
Nasleduje program stretnutia. Na začiatok sa uvedie zoznam bodov stretnutia. Body stretnutia sa číslujú nasledovne:

Program stretnutia:

1. *Bod 1*
2. *Bod 2*

Za prehľadom programu stretnutia nasleduje podrobnejší opis stretnutia. Nie je potrebné sa vyjadrovať k všetkým bodom stretnutia pokiaľ je z ich názvu jasné čo bolo jeho obsahom. K jednotlivým bodom sa môže vyjadriť rôzne dlhým textom, ktorý by mal byť však čo najstručnejší s prihliadnutím na potrebu zrozumiteľnosti aj po časovom odstupe. Formát je nasledovný:

Podrobnejšie k niektorým bodom:

1) *Opis bodu stretnutia, ktorý je vedený pod číslom 1 v programe stretnutia*
2) *Opis bodu stretnutia, ktorý je vedený pod číslom 2 v programe stretnutia*
3) *...*

Kvôli prehľadnosti úloh sa za touto časťou zhrnú niektoré priradené úlohy do tabuľky nasledovne:

Pridelenie úloh zo stretnutia:

Názov úlohy	***Priradený riešiteľ***
Stručný a jasný názov úlohy	*Riešiteľ alebo skupina riešiteľov danej úlohy*

Počas priebehu šprintu je možné na záver dokumentu uviesť tabuľku obsahujúcu jednotlivé úlohy začlenené do šprintu a jeho stav. Využíva sa pritom formát vychádzajúci z prehľadu šprintu v Jire:

Označenie	*Popis*	*Epic*	*Riešiteľ*	*Stav*
TESA-číslo	*Názov úlohy*	*prototyp/alef*	*Meno riešiteľa*	*Stav úlohy*

Na konci dokumentu sa uvádza *meno zapisovateľa*.

Formálna stránka dokumentu

Celý dokument je písaný vo formáte Times New Roman, veľkosť textu 12 (ak nie je uvedené inak), zarovnanie do bloku.

Metodika a štandardy pre tvorbu kódu

Účel metodiky

Táto metodika popisuje postup a štandardy ako písať kód tak aby bol jednoducho čitateľný pre všetkých členov projektu

Dedikácia metodiky

Rola	Úloha
Programátor	Píše nový kód a musí rozumieť pôvodnému kódu
Ktorýkoľvek iný člen tímu	Ak potrebuje pracovať s kódom niekto iný ako programátor, napríklad kvôli testovanie či code review
Zákazník	Uzatvára úlohy, musí byť schopný porozumieť kódu a posúdiť kvalitu riešenia

Pravidlá pre formátovanie a písanie kódu

- kód musí byť konzistentne formátovaný
- používať odsadzovanie vo forme dvoch medzier namiesto tabulátoru(niektoré IDE to robia automaticky)
- nepoužívať ';' na konci príkazov, radšej píšte jeden príkaz na riadok
- písať medzery medzi operátormi, napr. : '1 + 2'. Okrem operátorov '**' a '!'
- žiadne medzery pred ']',')' a po '[','('
- odsadzovať 'when' rovnako ako 'case'
- prázdne riadky v kóde použiť na rozdelenie kódu do logických celkov
- Snažte sa písať kód tak aby v jednom riadku bolo maximálne 80 znakov
- Ukončiť každý súbor novým riadkom
- v prípade, že metóda nemá argumenty, nepíšte v deklarácií zátvorky, naopak ak argumenty má tak tieto v budú zátvorkách.
- Namiesto cyklu for, používajte cyklus each
- Namiesto NOT, AND, OR používajte '!', '&&' a '||'
- Ak to nie je nutné kvôli kontrole vykonávanie, snažte sa nepoužívať výraz return
- Používajte operátory '||=' a '&&=' k inicializácií a preprocessingu premenných, ktoré nemusia existovať

Pravidlá pre komentovanie v kóde

- V prvom rade sa snažte tvoriť samovysvetľujúci sa kód, používajte komentáre iba keď to je naozaj nevyhnutné
- Komentáre zásadne v anglickom jazyku
- Komentáre v ruby sa začínajú znakom '#', vkladajte medzi tento znak a váš komentár prázdny riadok

- Udržujte komentáre aktuálne, nesmie sa stať, že komentár vzťahujúci sa k staršej verzií kódu stále existuje v pôvodom stave
- nepoužívať blokové komentáre
- pri komentároch, ktoré slúžia ako anotácie používajte slová TODO, FIXME, OPTIMIZE, HACK a REVIEW

Pravidlá pre tvorbu názvov v kóde

- Vždy po anglicky
- Používať samovysvetľujúce názvy premenných, tried a funkcií
- Vždy pracovať v súlade s konvenciami nazývania jazyka Ruby a frameworku Ruby on Rails
- Používať snake_case pre metódy a premenné
- Používať CamelCase pre moduly a triedy(okrem skratiek)
- Používať SCREAMING_SNAKE_CASE pre konštanty
- Správne používať '!' a '?' v názvoch funkcií, ktoré automaticky prepisujú dáta a **funkcií**, ktoré vracajú boolean hodnoty. Napríklad string.gsub!('-',':') prepise string, user.nil? Slúži na spýtanie sa či objekt existuje

Konvencie v Ruby on Rails

- Ak model sa volá User tak databáza sa volá users, rozlišovanie na základe jednotného a množného čísla je bežné. Preto nie je vhodné používať slovenské názvy
- Aby vstavaný objektovo-relačný mapovač mohol správne fungovať cudzie kľúče v databáze musia byť v tvare singularized_table_name_id (napríklad item_id, order_id)
- DRY(Dont repeat yourself) prístup

Metodika code review

Slovník pojmov

Pojem	Vysvetlenie
Code review	Proces vzájomného kontrolovania a hodnotenia kvality kódu medzi vývojármi
Commit	Odovzdanie a uloženie zmien do zdieľaného repozitára
Jira	Proprietárny softvér na sledovanie úloh a chýb
GitHub	Distribuované úložisko s manažmentom verzií fungujúce na princípe repozitárov

Účel metodiky

Cieľom kontroly kódu je hlavne zvýšenie kvality zdrojového kódu, ako aj celkového produktu. Pomocou kontroly kódu sa rozširuje obzor programátorov v aplikácii, čo prispieva k efektivite pri vývoji a budúcich zmenách v zdrojovom kóde.
Táto metodika popisuje postup, akým je vykonávaná kontrola a hodnotenie kvality zdrojového kódu. Určuje všeobecné pravidlá a zásady, ktoré je potrebné dodržať pri vykonávaní code review.

Dedikácia metodiky

Metodika je určená každému členovi tímu, ktorý sa zúčastňuje na tvorbe alebo kontrole kvality zdrojového kódu.

Rola	Úloha
Programátor – autor zdrojového kódu	Člen tímu podieľajúci sa na vývoji funkcionality aplikácie
Programátor – hodnotiteľ zdrojového kódu	Iný člen tímu, ktorý pozná funkcionalitu kódu, ale nezúčastnil sa na vývoji

Kedy je potrebné vykonať code review ?

Kontrolu kvality kódu je potrebné vykonať po ukončení vývoja uceleného bloku funkcionality. Eliminuje sa tak možnosť nasadenia nekvalitného kódu a prípadných chýb spôsobených pri vývoji. Kontrola sa bude vykonávať hlavne po dokončení úloh patriacich do používateľského príbehu. O nutnosti kontroly kvality kódu ostatných úloh sa poradí autor s ostatnými členmi tímu.

Kedy nie je treba vykonať code review ?

Ak sa jedná o triviálne úlohy, ktorých implementácia nezaberie viac ako niekoľko minút. Sú nimi napríklad zmeny v konfigurácii, alebo úprava formátovania zdrojového. Taktiež ak sa jedná o úlohy, ktoré sa neevidujú v zdieľanom repozitári, akými sú napríklad úpravy v databáze, alebo úprava importovacieho nástroja.

Popis prípravy úlohy na code review :

1. Programátor po ukončení a skontrolovaní svojej úlohy vykoná commit do vetvy development v zdieľanom repozitári.
2. V nástroji Jira, kde má pridelenú túto úlohu, ju označí za hotovú.
3. Pridá úlohu na koniec zoznamu úloh pripravených na code review.
4. Informuje ostatných členov tímu o aktualizácii zoznamu.

Popis vykonania code review úlohy:

1. Programátor si otvorí zoznam úloh čakajúcich na code review.
2. Vyberie si prvú úlohu zo začiatku zoznamu, tá sa nachádza v zozname najdlhšie, čiže jej kontrola má najvyššiu prioritu.
3. V nástroji na code review integrovanom priamo službou GitHub si nájde commit súvisiaci s vybranou úlohou.
4. Na základe svojich znalostí a pravidiel dohodnutých pre code review zreviduje všetky zmeny.
5. Po skontrolovaní a zrevidovaní všetkých zmien úlohu presunie zo zozname čakajúcich na code review do zoznamu skontrolovaných úloh.