EKONOMICKÁ UNIVERZITA V BRATISLAVE

FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/I/2022/36122163606727940

RÔZNE PRÍSTUPY V MERANÍ KVALITY SOFTVÉRU V PODNIKOVEJ PRAXI

Diplomová práca

2022 **Miloš Mucha**

EKONOMICKÁ UNIVERZITA V BRATISLAVE

FAKULTA HOSPODÁRSKEJ INFORMATIKY

RÔZNE PRÍSTUPY V MERANÍ KVALITY SOFTVÉRU V PODNIKOVEJ PRAXI

Diplomová práca

Študijný program: Informačný manažment

Študijný odbor: Ekonómia a manažment

Školiace pracovisko: Katedra aplikovanej informatiky

Vedúci záverečnej práce: RNDr. Eva Rakovská PhD.

Bratislava 2022 **Miloš Mucha**

Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Miloš Mucha

Študijný program: informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)

Študijný odbor: ekonómia a manažment

Typ záverečnej práce: Inžinierska záverečná práca

Jazyk záverečnej práce: slovenský

Sekundárny jazyk: anglický

Názov: Rôzne prístupy v meraní kvality softvéru v podnikovej praxi

Anotácia: Meranie kvality softvéru v dnešnej dobe je významnou súčasťou informačného manažmentu. Existujú rôzne aspekty merania kvality softvéru a práca sa zameriava na zmapovanie rôznych prístupov v meraní kvality IS z viacerých aspektov a metrík. Detailne zmapuje meranie kvality z hľadiska vývoja softvéru.

Vedúci: RNDr. Eva Rakovská, PhD.

Oponent: Ing. Veronika Horniaková, PhD.

Katedra: KAI FHI - Katedra aplikovanej informatiky FHI

Vedúci katedry: Ing. Mgr. Peter Schmidt, PhD.

Dátum zadania: 29.10.2020

Dátum schválenia: 30.10.2020 Ing. Mgr. Peter Schmidt, PhD.

vedúci katedry

Čestné vyhlásenie

Čestne vyhlasujem, že som diplomovú prácu vypracoval samostatne pod vedením vedúceho diplomovej práce s použitím odbornej literatúry a ďalších zdrojov, ktoré sú uvedené v zozname použitej literatúry.

Dátum: 25.4.2022 .....

(podpis študenta)

Poďakovanie

Ďakujem svojej vedúcej diplomovej práce RNDr. Eve Rakovskej PhD. Za odborné vedenie, cenné rady, komentáre a pripomienky, a predovšetkým za ústretový postoj pri prejedávaní jednotlivých častí tejto práce.

Abstrakt

MUCHA, Miloš: *Rôzne prístupy v meraní kvality softvéru v podnikovej praxi* - Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky. Katedra aplikovanej informatiky. - Vedúca záverečnej práce: RNDr. Eva Rakovská PhD. - Bratislava: FHI EU, 2022, 80 s.

Cieľom diplomovej práce je zmapovanie rôznych prístupov merania kvality softvéru a informačných systémov v podnikovej praxi z rôznych hľadísk s aspektom na vývoj softvéru. Meranie kvality softvéru v dnešnej dobe je významnou súčasťou informačného manažmentu. Práca sa skladá z troch hlavných kapitol. Prvá kapitola je zameraná na porozumenie a analyzovanie modelov a certifikácií, ktoré pristupujú ku kvalite softvéru a následne techniky zaistenia a riadenia kvality, či metodiky práce, aké firmy využívajú v súčasnej dobe pre vývoj softvéru. V druhej kapitole je popísaný cieľ práce, jeho pod ciele a následne metódy skúmania a metodiku práce. V tretej kapitole na základe syntézy poznatkov je vytvorený návrh, akým by firmy mali pristupovať ku kvalite softvéru v metodike Scrum. Pre jednotlivé oblasti sú popísané a zmapované metriky merania kvality softvéru. V závere tejto kapitoly je vytvorený návrh metrík, ktoré najlepšie odzrkadľujú kvalitu vyvíjaného softvéru, na základe zmapovania a konzultácií s manažmentom v podnikovej praxi.

Kľúčové slová:

kvalita softvéru, metriky kvality softvéru, indikátory kvality softvéru, ISO normy, tradičné metodiky, agilné metodiky

Abstract

MUCHA, Miloš: *Different approaches in measuring software quality in business practice* - University of Economics in Bratislava. Faculty of Business Informatics; Department of Applied Informatics - Thesis supervisor: RNDr. Eva Rakovská PhD. - Bratislava: FHI EU, 2022, 80 s.

The aim of this master thesis was the mapping of various approaches of measuring the quality of software and information systems in the business practice from different viewpoints with the emphasis on software development. Today, measuring software quality is an important part of information management. The thesis is dived into three main chapters. The first one is focused on understanding and analysis of models and certificates, which deal with software quality and subsequently quality assurance and management techniques, or work methodologies that companies currently use for software development. The second chapter describes the main goal of the work, its sub-goals and then the research methods and methodology. In the third chapter, based on the acquired knowledge, a proposal is made for companies to approach the quality of software using the Scrum methodology. Software quality measurement metrics are described and mapped for each area. At the end of this chapter, a proposal of metrics is created, which best reflect the quality of the developed software, based on mapping and consultation with management in business practice.

Key words:

software quality, software quality metrics, software quality indicators, ISO standards, traditional methodologies, agile methodologies

OBSAH

Zoznam Ilustrácií

Úvod

V súčasnosti každá firma pôsobiaca v oblasti informačných technológií, ktorá vyvíja softvérový produkt sa snaží zvyšovať kvalitu svojho produktu, a tým získať výhodu oproti konkurencií. Kvalita je merateľná vlastnosť produktu, ktorá sa skladá z rôznych faktorov, o ktorej sme schopný rozhodovať na akej úrovni je oproti produktom rovnakého typu. Čím kvalitnejší produkt firma dodáva, tým viac zákazníkov má o produkt alebo samotnú firmu záujem a firma môže dosiahnuť vyšší zisk.

Diplomová práca je zameraná na metriky, pomocou ktorých sa meria kvalita softvéru. Aby sme mohli tieto metriky zmapovať, je v prvom rade potrebné porozumieť a analyzovať čo znamená kvalita softvéru. Na základe získaných poznatkov sa môžeme zamerať na samotné techniky zaistenia a riadenia kvality softvéru a normy, ktorými sa firma certifikuje, že dodržiava štandardy pre kvalitu softvéru. Taktiež je potrebné, aby sme porozumeli, akými metodikami vývoja softvéru sa v súčasnej dobe firmy môžu riadiť, a ktorá z nich je najvyužívanejšia a najvhodnejšia.

Po analýze a syntéze získaných poznatkov môžeme zmapovať metriky merania kvality softvéru z rôznych uhľov pohľadu, respektíve, z rôznych oblastí v rámci vývoja softvéru. Zmapované metriky následne prezentujeme manažmentu z podnikovej praxe, vďaka ktorým na základe konzultácií a získaných dát urobíme porovnanie a vytvoríme návrh metrík, ktoré by mala zaužívať každá vývojová firma, aby dosiahla kvalitu vyvíjaného softvéru.

1 Súčasný stav riešenej problematiky doma a v zahraničí

1.1 Kvalita a akosť

Kvalita a akosť je vo všeobecnosti pozitívna vlastnosť výrobku alebo služby. Táto vlastnosť je výrobku alebo službe priradená, pokiaľ sú splnené predom definované štandardy alebo požiadavky klienta. Môžeme tvrdiť, že o produktoch a službách hovoríme, že sú kvalitné, pokiaľ sme s nimi spokojní a spĺňajú naše očakávania.

Kvalita a akosť majú rovnakú výpovednú hodnotu, ale líšia sa v oblasti použitia. Akosť sa využíva v oblasti výroby, predovšetkým v spojení s konkrétnym výrobkom. Naopak pojem kvalita sa využíva v oblasti služieb. Keďže sa práca zameriava na oblasť služieb, budeme používať tento pojem [2].

Neexistuje správna definícia kvality, každý si pod kvalitou predstavuje niečo iné a zameriava sa na subjektívne aspekty. Akademický slovník cudzích slov definuje kvalitu ako „súhrn úžitkových vlastností výrobku alebo služby“. Armand Vallin Feigenbaum hovorí, že: „kvalita výrobku je súhrn všetkých jeho konštrukčných a výrobne technických charakteristík, ktoré určujú úroveň, akou produkt napĺňa očakávania zákazníka“. Podľa normy ISO 9001 je kvalita „stupeň splnenia požiadaviek súborom obsiahnutých znakov“ [2]. Z vyššie spomenutých definícií vyplýva, že kvalita je merateľná vlastnosť, ktorá tvorí rôzne aspekty, a ktorou sme schopný porovnať produkty rovnakého typu.

1.2 Kvalita softvéru

Kvalita softvéru je zhoda s explicitne stanovenými požiadavkami na funkcionalitu a správanie, explicitne dokumentovanými vývojovými štandardmi a implicitnými charakteristikami, ktoré sa očakávajú od každého profesionálneho vyvíjaného softvéru.

Jednoducho povedané kvalitný softvér je taký softvér, ktorý funguje v súlade s požiadavkami klienta, ktoré zodpovedajú štandardom a je vyvinutý s ohľadom na obecne platné zásady tvorby konkrétneho softvéru [3].

Vývoj kvality softvéru je históriou rastu. Kvalita softvéru v počiatkoch (okolo 50. rokov) bola často záležitosťou jednej osoby, ktorá bola dizajnérom, vývojárom a testerom naraz. Táto osoba pracovala, kým sa program „nerozbehol“. Kvalita znamenala vyladenie všetkých chýb, ktoré sa vyskytli počas vývoja.

Netrvalo dlho (do 70. rokov), kým sa cieľ stal ambicióznejším: kvalita bude dosiahnutá, keď budú splnené všetky požiadavky. To znamená byť schopný pokryť všetky prípady opísané koncovým užívateľom, prípadne technickú, funkčnú či dizajnovú analýzu.

Nevyhnutne sa vytvorilo silné prepojenie medzi kvalitou a testovaním: kvalita by sa merala v porovnaní s úspechmi testov a testy by preskúmali všetky funkcie na dosiahnutie cieľovej kvality.

Ako sa tímy začali rozširovať, pribúdali aj profesie. Vývojári dostali koncepty od dizajnérov a dodávali testerom funkcionality na overenie. V tomto vertikálnom vzťahu by každý účastník odovzdal výsledok svojej práci ďalšiemu. Fungovalo to dobre, ale nie veľmi efektívne: dlhé oneskorenia, neskoré testovanie, nákladné opravy a nechuť k zmenám.

Od 80. rokov sa presadilo niekoľko myšlienok:

- Vnútorná kvalita podľa analýzy kódu - Začať používať nástroje statickej a dynamickej analýzy zdrojového kódu na posúdenie jeho vnútornej kvality,
- Zdravšia kvalita vďaka skorému testovaniu - Zastavme testovanie príliš neskoro, opravy zistených chýb sú oveľa lacnejšie,
- Lepšia kvalita vďaka automatizovanému testovaniu - Keď sú testy na nízkej úrovni správne opísané, môžu byť automatizované. Testy by tiež nemali len overiť očakávané, ale mali by tiež automaticky preskúmať zákutia pre kombináciu, o ktorej si nikto nemyslel, že by sa mohla stať,
- Širšia kvalita vďaka viacnásobným testom - Ako sa hardvér stával vyspelejším, výpočtový výkon umožnil spustiť testovacie kampane na niekoľkých verziách programu,
- Zdieľaná kvalita akceptovaním zmien - Proti nepružnosti starých vývojových procesov spoločnosť Agility navrhla zrušiť „efekt tunela“, spojiť všetky zainteresované strany a udržať ich v zapojení,
- Nepretržitá kvalita vďaka integrovanému prístupu - S DevOps sa kvalita môže stať súčasťou procesu. A keďže sú dostupné všetky komponenty životného cyklu vývoja, kvalita sa môže rozšíriť z jeho kódu a definície zameranej na test, aby zahŕňala viac prvkov z projektu.

Dnes sme sa dostali do bodu, keď kvalita softvéru presahuje rámec konformity, ale je tiež prostriedkom zrelosti [26].

Pri kvalite softvéru existuje mnoho noriem, ktoré sa tým zaoberajú ako napr. ISO 9000, IEEE Std.730-1984, IEEE Std.983-1986, DOD-STD-2167A a mnoho ďalších.

Napríklad rad noriem ISO 9000 konkrétne definuje systém manažérstva kvality. Tieto normy vydáva Medzinárodná organizácia pre normalizáciu. Normy umožňujú preukázať daným organizáciám schopnosť výroby či distribúciu produktov v súlade so všetkými potrebnými predpismi a potrebami zákazníka. ISO 9000 zlučuje tri štandardy ISO 9001, ISO 9002 a ISO 9003.

Štandard ISO 9001 slúži ako referenčný model pre nastavenie základných riadiacich procesov v organizácii, ktoré pomáhajú neustále zlepšovať kvalitu poskytovaných výrobkov a služieb, spokojnosť zákazníka, strategické riadenie a prácu s rizikami. Je to norma procesne orientovaná. Rovnako ako ostatné normy ISO vyžaduje následnú certifikáciu zavedeného systému riadenie organizácii. Výsledkom je certifikát, ktorý je medzinárodne uznávaný a je predpokladom určitej zrelosti a vyspelosti organizácie [16].

Okrem ISO noriem je dôležité spomenúť aj CMMI (Capability Maturity Model Integration), ktorý opisuje model kvality organizácie práce, ktorý je určený pre vývojové tímy a do istej miery ovplyvňuje kvalitu softvéru. Ide o súhrn cieľov a odporúčaných pracovných postupov pre vývojové tímy, ktoré vedú ku kvalitnému plánovaniu a riadeniu prác, a mali by zabezpečiť aj zodpovedajúcu kvalitu výstupu. CMMI definuje procesné oblasti, ktoré musí tím realizovať, a ciele, ktoré musí v každej oblasti dosahovať.

Model rozdeľuje procesné oblasti do niekoľkých skupín podľa typu činností:

- **riadenie procesov** – zameranie sa na procesy organizácie, definície procesov organizácie, školenia organizácie,
- **riadenie projektov** – plánovanie, monitorovanie a riadenie projektov, riadenie vzťahov so subdodávateľmi, riadenie rizík, integrované riadenie projektov,
- **návrh a realizácia –** riadenie požiadaviek, vývoj požiadaviek, technické riešenie, integrácia produktu, verifikácia a validácia,
- **podporné procesy** – riadenie konfigurácií, zaistenie kvality produktov a procesov, meranie a analýza, rozhodovanie na základe analýzy variantov.

Pre porovnanie môžeme tvrdiť, že norma ISO 9000 je určením blízka k modelu CMMI, ale je medzi nimi niekoľko zásadných rozdielov. Norma ISO 9000 nie je určená pre žiadnu konkrétnu oblasť a je aplikovaná na firmy z najrôznejších odborov. Naproti tomu CMMI je určená pre vývojové tímy. ISO 9001 je stručný štandard, ktorý definuje iba ciele,

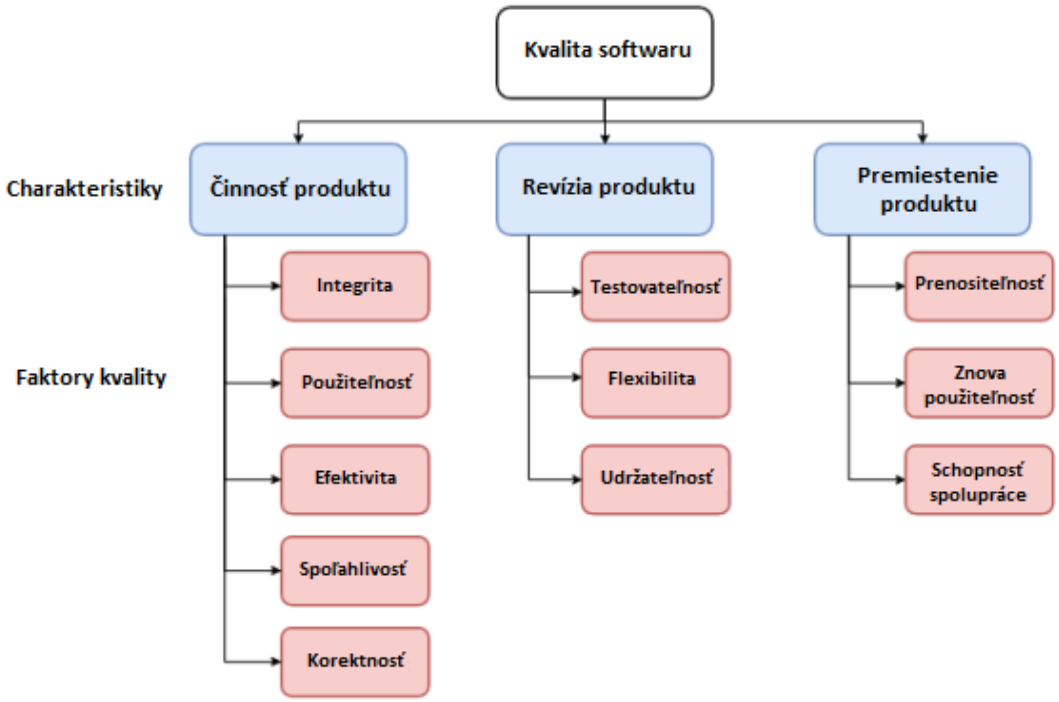
zatiaľ čo CMMI je podrobný model, ktorý ide do podrobností, keď definuje očakávané činnosti a ich pracovné výstupy. Vďaka tomu má CMMI návodný charakter, takže na jeho základe je možné procesy priamo definovať [14].

Kvalita softvér je taktiež ovplyvnená mnohými faktormi. Existuje niekoľko modelov pre delenie týchto faktorov. Môžeme sem zaradiť McCallov model, Deutch-Willisov model, Evans-Marciniakov model, FURPS, jeho rozšírenie FURPS+ a niektoré ISO štandardy [4]. Detailnejšie si rozoberme McCallov model a štandard ISO/IEC 25010.

1.2.1 *McCallov model*

McCallov model delí faktory do troch kategórií [4]:

- činnosť produktu: integrita, použiteľnosť, efektivita, spoľahlivosť , korektnosť,
- revízia produktu: testovateľnosť, flexibilita, udržateľnosť,
- premiestenie produktu: prenositeľnosť, znova použiteľnosť, schopnosť spolupráce.



Obrázok 1 McCallov model [5]

1.2.2 *ISO/IEC 25010*

ISO/IEC 25010 je súčasť rady štandardov s názvom Software product Quality Requirements and Evaluation (SQuaRE). V roku 2011 nahradil štandard ISO/IEC 9126-1

Softvérové inžinierstvo – Kvalita produktu. ISO/IEC 25010 je medzinárodná norma, ktorá definuje model kvality pre používanie (zložený z 5 charakteristík) a model kvality produktu (zložený z ôsmich charakteristík). Model kvality produktu je použiteľný ako pre počítačové systémy, tak aj pre softvérové produkty, a tým sa budeme ďalej zaoberať. Hierarchia modelu ISO/IEC 25010 reprezentuje obrázok č. 2. Každá charakteristika je ďalej členená na sady indikátorov. Výber indikátorov je založený na sade pokynov pre kvalitu webu, štandardov W3C a analýze stavajúcich webových stránok.

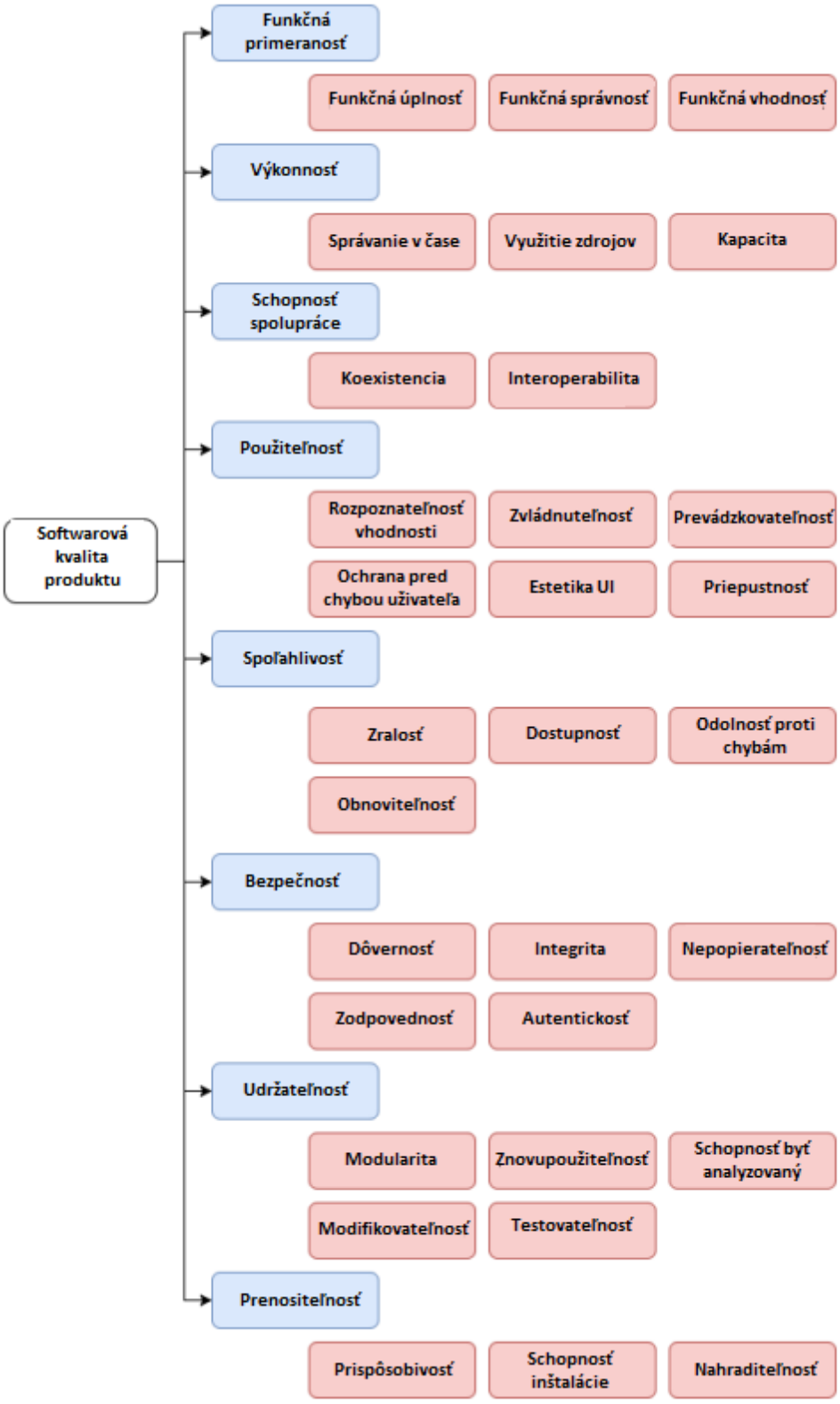
Charakteristiky sú [6]:

- funkčná primeranosť - systém poskytuje funkcie, ktoré splňujú stanovené a implicitné potreby pri používaní systému zo stanovených podmienok
    - funkčná úplnosť - funkčnosť systému spĺňajú všetky zadané úlohy a ciele užívateľa,
    - funkčná správnosť - systém vracia správne výsledky s určitou mierou presnosti,
    - funkčná vhodnosť - funkcie uľahčujú splnenie stanovených úloh a cieľov.
- výkonnosť – predstavuje výkon vzhľadom k množstvu použitých prostriedkov za stanovených podmienok,
    - chovanie v čase - splnenie požiadaviek na dobu odozvy a spracovania,
    - využitie zdrojov - množstvo a typy prostriedkov potrebné pre systém, aby splnil všetky požiadavky,
    - kapacita - najvyšší výkon, ktorý je možné splniť.
- schopnosť spolupráce - schopnosť systému alebo jeho časti vymieňať informácie s inými systémami alebo komponentami alebo vykonávať svoje požadované funkcie, pričom zdieľa rovnaké hardvérové alebo softvérové prostredie,
    - koexistencia - systém môže správne splniť požadované funkcie a zároveň zdieľať spoločné prostredie a zdroje s inými produktami bez negatívneho dopadu na akýkoľvek iný produkt,
    - interoperabilita - dva alebo viaceré systémy alebo komponenty môžu vymieňať informácie a využívať vymieňane informácie.
- použiteľnosť - výrobok môže byť používaný určenými používateľmi k dosiahnutiu stanovených cieľov účinne, efektívne a uspokojivo v stanovenom kontexte použitia,
    - rozpoznateľnosť vhodnosti - používatelia môžu rozpoznať, či je produkt alebo systém vhodný pre ich potreby,

    - zvládnuteľnosť - možnosť naučiť sa ako používať systém efektívne, bez rizika a so spokojnosťou v špecifikovanom kontexte použitia,
    - prevádzkovateľnosť - systém je ľahko ovládateľný,
    - ochrana pred chybou používateľa - systém má vytvorené opatrenia, ktoré sa snažia ochrániť používateľa pred jeho chybami,
    - estetika používateľského rozhrania - používateľské rozhranie umožňuje prijemnú a uspokojujúcu interakciu pre používateľa,
    - prístupnosť - systém môže byť použitý osobami s najširším spektrom vlastností a schopností k dosiahnutiu stanoveného cieľa v konkrétnom kontexte použitia.
- spoľahlivosť - systém alebo jeho súčasť plní stanovené úlohy v rámci stanovených podmienok na určitú dobu,
    - zrelosť - systém alebo jeho súčasť splňuje požiadavky na spoľahlivosť pri normálnej prevádzke,
    - dostupnosť - systém alebo jeho súčasť funguje a je prístupný, ak je vyžadovaný pre použitie,
    - odolnosť proti chybám - systém alebo jeho súčasť funguje tak, ako je plánované a to aj napriek tomu, že dochádza k chybám hardvéru alebo softvéru,
    - obnoviteľnosť - systém môže v prípade prerušenia alebo zlyhania obnoviť priamo ovplyvnené údaje a obnoviť požadovaný stav systému.
- bezpečnosť - informácie a dáta sú chránené tak, aby nedošlo k neoprávnenej modifikácie alebo prístupu neautorizovanou osobou alebo systémom a naopak oprávneným osobám alebo systémom nie je odmietnutý prístup k nim,
    - dôvernosť - systém zaisťuje, že údaje sú prístupné osobám oprávneným k prístupu,
    - integrita - systém alebo jeho súčasť zabraňuje neoprávnenému prístupu alebo úpravám počítačových programov alebo dát,
    - nepopierateľnosť - preukázateľnosť prebehnutia akcie alebo udalosti, aby nemohli byť popreté neskôr,
    - zodpovednosť - činnosti subjektu môžu byť jednoznačné sledované,
    - autentickosť - preukázateľnosť totožnosti subjektu alebo zdroja.
- udržateľnosť - možnosť opravy chyby alebo modifikácie systému,

- modularita - systém alebo počítačový program je zložený z komponentov tak, aby zmena jedného komponentu mala minimálny dopad na iné komponenty,
- znovu použiteľnosť - systém, časť systému alebo počítačový program môže byť použitý vo viac než jednom systéme,
- schopnosť byť analyzovaný - možnosť posúdenia dopadu zmeny na systém alebo jeho časti, pripadne diagnostikovanie systému z dôvodu jeho nedostatku alebo príčni zlyhania.
- modifikovateľnosť - možnosť upravovať systém účinne a efektívne bez toho, aby došlo k porušeniu kvality výrobku,
- testovateľnosť - možnosť stanoviť skúšobné kritéria pre systém alebo jeho časť a možnosť vykonať testy, aby bolo možné určiť, či boli kritéria splnené

- prenositeľnosť- systém alebo jeho súčasť môže byť účinne a efektívne použitá na rôznom hardvér a softvér.
  - prispôsobivosť - možnosť systému účinne sa prispôsobovať rôznym hardvérovým, softvérovým zariadeniam a používateľským prostrediam,
  - schopnosť inštalácie - možnosť systém úspešne nainštalovať alebo odinštalovať v špecifickom prostredí,
  - nahraditeľnosť - možnosť systém nahradiť iným špecifickým softvérovým produktom pre ten istý účel v rovnakom prostredí.



Obrázok 2 Štandard ISO/IEC 25010 [6]

1.3 Zaistenie a riadenie kvality softvéru

Zaistenie kvality (Software Quality Assurance - SQA) je súhrnný pojem, ktorý zahrňuje už na začiatku vývoja stanovisko procesu a metód, ako správne definovať požiadavky na systém, správne vyvíjať produkt a ďalšie etapy vývoja. Úlohou SQA je pochopiteľne nie len stanovenie procesu, ale aj kontrola procesu a všetkých jeho výstupov, ktoré sa týkajú riadenia kvality (Software Quality Control - SQC) [4].

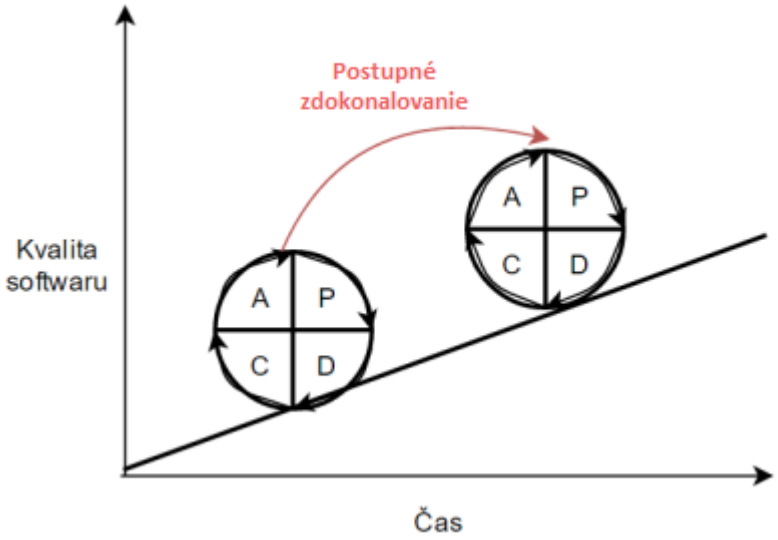
Kvalita a jej zabezpečenie a riadenie sú jedným z rozhodujúcich faktorov stabilného ekonomického rastu firiem. Kľúčovým pozitívnym prejavom dobre zavedeného systému riadenia kvality je rastúca spokojnosť, lojalita zákazníkov, zefektívnenie procesov kontroly, zníženie nákladov a zvýšenie produktivity. Keďže rastie podiel na prvýkrát dobre vykonanej práce, znižuje sa počet prerábaní a opráv všetkého druhu alebo dodatočné hľadanie najrôznejších dát a informácií. Firmy s fungujúcim systémom riadenia kvality dosahujú dlhodobo podstatne lepšie výsledky ako ostatné firmy.

V rámci riadenia kvality a jeho neustáleho zlepšovanie vo všetkých fázach vývoja, si popíšeme cykly zlepšovania PDCA.

1.3.1 *PDCA*

PDCA je metóda postupného zlepšovania napr. kvality výrobkov, služieb, procesov, aplikácií atď., prebiehajúca iteratívne v štyroch krokoch [7].

- naplánuj (Plan) – stanovovanie cieľov podľa existujúcich podmienok, čo zahŕňa určenie úloh, opatrení a procesov na dosiahnutie cieľov,
- rob (Do) – realizácia cieľa podľa navrhnutého riešenia,
- kontroluj (Check) – kontrola výsledku realizácie oproti očakávaným výsledkom,
- uskutočni (Act) – vyhodnotenie, na ktorého konci sa rozhodne, či mala úprava pozitívny efekt, následne sa nový proces pridá do funkcionalít systému. Ak nie, tak sa nebude nič meniť. Ak pri vyhodnotení vyplynuli možnosti na zlepšenie, tak sa metóda vracia do prvého kroku a proces sa opakuje.



Obrázok 3 PDCA [8]

1.4 Techniky zaistenia a riadenia kvality

Jadrom zaistenia a riadenia kvality je testovanie, ktoré si preberieme v ďalšej kapitole. Medzi techniky zistenia a riadenia kvality patria [3]:

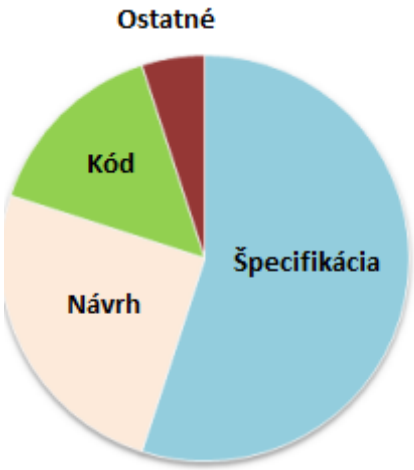
- preskúmanie (revízia) – je činnosť vykonávaná na zabezpečenie vhodnosti, primeranosti, efektívnosti a účinnosti predmetu s cieľom dosiahnuť stanovené ciele,
- inšpekcia – je formálna metóda preskúmania podľa prísnych pravidiel. Jej cieľom je predovšetkým odkryť chyby vo funkcii, logike či implementácii, overiť či softvér vyhovuje požiadavkám, uistiť sa, že softvér je implementovaný podľa definovaných štandardov atď.,
- audit – je preverenie dodržiavania a stavu plnenia úloh nezávislou skupinou. Niektorú špecifickú oblasť, napr. účtovný audit, audit kvality podľa normy ISO 9000, môže preverovať len akreditovaná organizácia,
- simulácia – z pohľadu techník zaistenia kvality sa jedná o ručné alebo poloautomatické prechádzanie častí programu so symbolickým vykonávaním výpočtu.

1.5 Testovanie

Testovanie je preverovanie funkčnosti softvéru. Proces testovania začína stanovením vízie a cieľov testovania. Ďalej sa určí sada testovania, teda čo všetko treba testovať, vyberajú sa testy, zbierajú sa dáta a pripravujú nástroje, ktoré tím na testovanie potrebuje.

Navyše sa kontroluje, či všetky požiadavky na softvér sú v takej forme, aby bolo možné jednoznačne skontrolovať ich splnenie. Samotné testovanie prebieha skúmaním softvéru na niekoľkých úrovniach a reportovaním nájdených skutočností. Proces testovania sa často vykonáva vo viacerých iteráciách, kedy každá iterácia začína odovzdaním novej verzie softvéru testom.

Môžeme tvrdiť, že súčasťou procesu testovania softvéru je zisťovanie informácií o jeho kvalite. Často mylná predstava je, že kvalita softvéru sa znižuje v dôsledku softvérových chýb, ktoré vznikajú na strane programátora, ktorý niečo prehliadol alebo sa mu v kóde objavila chyba. Nie je tomu tak. Podľa Rona Pattona je najčastejším zdrojom chýb zlá špecifikácia. Je to spôsobené tým, že často chýba úplná dokumentácia, čo znamená, že je nedostačujúca alebo sa v priebehu vývoja mení. Na obrázku č. 4 môžeme vidieť, že ďalšie softvérové chyby vznikajú v návrhu a zdrojovom kóde [9].



Obrázok 4 Vznik softvérových chýb [9]

Aby sme predišli k takýmto chybám je dôležité, aby testeri boli zapojení už na samotnom začiatku spisovania požiadaviek s cieľom zlepšovať užívateľské scenáre, a tak znížiť riziko nesprávneho alebo netestovateľného návrhu aplikácie.

Aby boli testy kvalitné a čo najviac pokrývali funkcionalitu, je dôležitá aj spolupráca testerov s programátormi. Porozumenie kódu zo strany testerov znižuje riziko defektov už v samotnom kóde a následne v testoch.

Taktiež je dôležité, aby sa kládol dôraz na analýzu koreňových príčin, kde zistenie a odstránenie príčin defektov zabezpečuje prevenciu ďalších chýb, ktoré sa môžu vyskytnúť [23].

1.5.1 *Základné členenie testovania softvéru*

Testovanie softvéru môžeme deliť z viacerých hľadísk. Základné členenie testovania [9]:

- statické a dynamické,
- black box a white box,
- alfa a beta testovanie,
- manuálne a automatizované,
- testovanie funkčných a nefunkčných požiadaviek.

Statické testovanie nevyžaduje beh softvéru, teda je možné s ním začať ešte pred vytvorením prvého prototypu. Môže pomôcť so spresnením odhadov náročnosti na čas a zdroje pre samotný nadchádzajúci vývoj. Dynamické testovanie vyžaduje existenciu spustiteľnej verzie softvéru a prebieha hlavne na základe poskytovania rôznych vstupov a posudzovanie výstupov testovaného programu [9]. Oba typy testov môžu mať rovnaké ciele ako je posúdenie kvality pracovných produktov a včasná identifikácia defektov. Môžeme povedať, že sa navzájom dopĺňajú, tým že nachádzajú rôzne typy defektov.

Statické testovanie možno použiť na zlepšenie konzistencie a vnútornej kvality pracovných produktov, zatiaľ čo dynamické testovanie sa zvyčajne zameriava na externe pozorovateľné správanie sa systému [23].

Čierna skrinka (Black box) testovania pracuje iba so vstupmi a výstupmi bez znalostí implementácie. Tester vidí iba, ako sa produkt chová navonok. Zmyslom je analyzovať správanie softvéru vzhľadom na očakávané vlastnosti tak, ako ho vidí používateľ. Pri testovaní bielej skrinky (white box) má tester prístup k zdrojovému kódu a testuje produkt na základe neho. Vidí nielen čo sa deje na povrchu skrinky, ale aj vnútorné reakcie systému. Tým trochu stráca pohľad používateľa, ale môže lepšie odhadnúť, kde hľadať chyby a kde nie [9].

Alfa a beta testovanie slúži na získanie spätnej väzby od potenciálnych alebo existujúcich užívateľov, zákazníkov alebo operátorov, skôr než uvedú softvérový produkt na trh. Alfa testovanie sa vykonáva potenciálnymi alebo existujúcimi zákazníkmi, alebo operátormi, či tímom nezávislých testerov. Beta testovanie je vykonávané potenciálnymi alebo existujúcimi zákazníkmi, alebo operátormi priamo na ich pracoviskách. Beta testovanie môže byť vykonané až po alfa testovaní, ale môže byť vykonané aj bez predchádzajúceho alfa testovania. Jedným z cieľov alfa a beta testovania je budovanie

dôvery medzi potenciálnymi alebo existujúcimi zákazníkmi alebo prevádzkovateľmi, aby bezproblémovo dosiahli svoje ciele s minimálnymi ťažkosťami, nákladmi a rizikami, a tým zvýšili kvalitu softvéru [23].

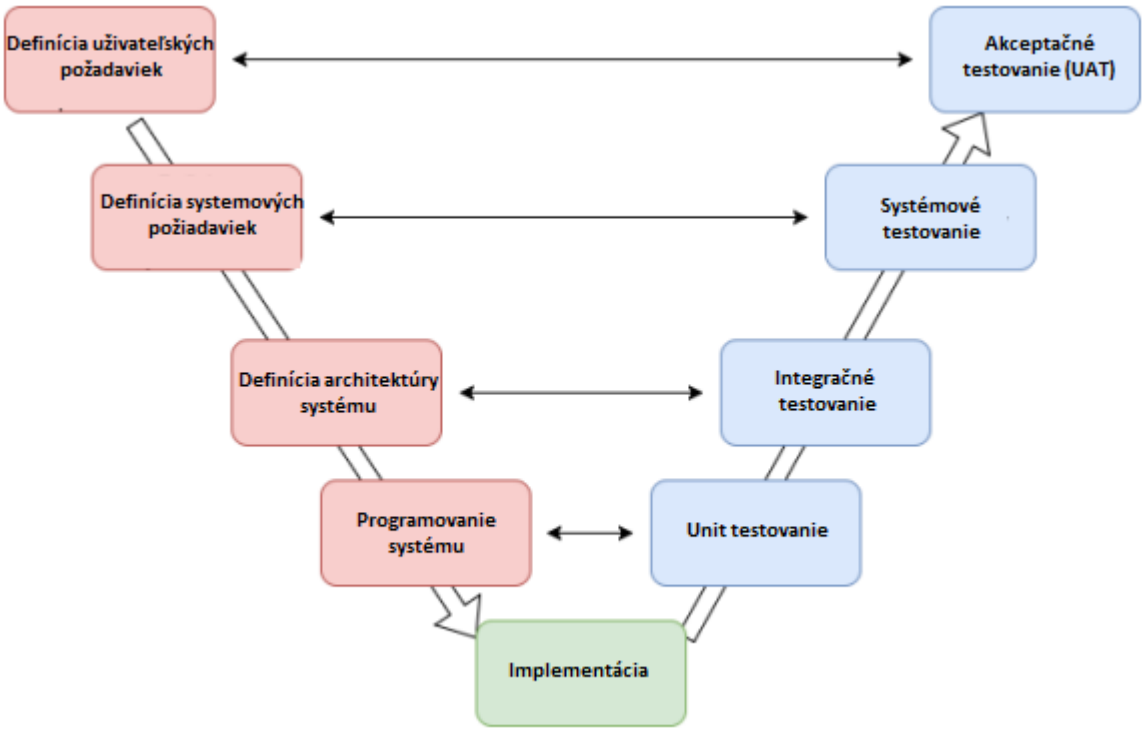
Manuálne a automatické testovanie sa líši tým, či ho vykonáva človek alebo softvér. Ak test vyžaduje ľudské ohodnotenie a úsudok alebo rozličné prístupy, ktoré nie je treba zaznamenať a pravidelne opakovať, je vhodnejšie manuálne testovanie. Pre opakované spúšťanie veľkého množstva testov alebo testu s veľkým množstvom generovaných dát, rovnako ako na záťažové testovanie je dobré použiť automatizované testy.

Delenie testovania podľa funkčných a nefunkčných požiadaviek si viac rozoberieme v nasledujúcich dvoch podkapitolách [9].

1.5.1.1 Testovanie funkčných požiadaviek

Testovanie funkcionality, alebo aj funkčné testy, zodpovedajú vzťahom medzi fázami vývoja softvéru, najlepšie ich popisuje V-model zobrazený na obrázku č. 5.

V-model je grafické znázornenie vývojového systému životných cyklov. Zhŕňa hlavné kroky, ktoré je potrebné vykonať na dosiahnutie výsledkov pri validácii softvéru. V ľavej vetve sú uvedené fázy vývoja aplikácie, v pravej vetve jednotlivé druhy testovania. Každý druh testov slúži na overenie inej fázy vývoja. Základom tohto prístupu je postup testovania od malých častí aplikácie cez väčšie funkčné celky a integráciu komplexných častí aplikácie, prípadne integráciu viacerých aplikácií až po kompletné otestovanie celej aplikácie. Prechod na ďalší druh testovania predpokladá úspešné dokončenie predchádzajúcej fázy [10].



Obrázok 5 V-model [10]

Prvou fázou testovania sú jednotkové (unit) testy, ktoré sa zameriavajú na najmenšie testovateľné časti aplikácie. Ide obvykle o testy jednotlivých komponentov aplikácie na úrovni modulov, objektov a tried. Tento druh testovania vykonávajú vývojári a nebýva zahrnutý do plánov testov aplikácie. Vývojári si týmito testami overujú, že nová alebo upravená časť kódu funguje a že jej funkčnosť zodpovedá zadaniu.

Pri integračných testoch je nutné rozlišovať medzi vnútornou integráciou, ktorá spočíva vo vzájomnej komunikácii jednotlivých častí aplikácie (modulov) a vonkajšej, kedy ide o prepájanie jednotlivých aplikácií do väčších funkčných celkov. Vonkajšia integrácia väčšinou spočíva v prepojení aplikácií od rôznych výrobcov, je teda náročná na kooperáciu vývojových tímov. Pri oboch týchto integráciách je nutné vykonávať integračné testy.

Systémové testy overujú, že aplikácia ako celok funguje správne. Testuje sa či správne plní úlohu, pre ktorú bola vyvinutá, vracia správne výstupy, a či boli ošetrené všetky neštandardné situácie a v neposlednom rade, že boli pokryté všetky požiadavky zo strany zákazníka. Systémové testy zvyčajne prebiehajú v niekoľkých iteráciách. Sú hlásené nájdené chyby, tie sú opravené a v nasledujúcich iteráciách pretestované.

Z pohľadu dodania aplikácie zákazníkovi sú najpodstatnejšie akceptačné testy (UAT). Tie majú overiť, že aplikácia spĺňa všetky zákazníkove požiadavky. Tieto testy môže

vykonávať rovnaký tím, ktorý vykonal systémové testy, ale rovnako časté je aj to, že zákazník poverí týmito testami iný tím, ktorý si často na tieto účely vytvorí. Požiadavky, ktoré zákazník na funkciu aplikácie má, je vhodné ešte pred zahájením vývoja (a určite pred začatím testovania) zhrnúť do tzv. akceptačných kritérií. Splnením týchto požiadaviek vývoj aplikácie v podstate končí a aplikáciu preberá zákazník [11].

1.5.1.2 Testovanie nefunkčných požiadaviek

Testovanie nefunkčných požiadaviek spočívajú v testovaní všetkých vlastností aplikácie, ktoré priamo nesúvisia s jej funkciami, ale zároveň sú podstatné pre jej správne fungovanie. Pokiaľ si načrtneme tie najobvyklejšie - aplikácia by mala byť spoľahlivá, stabilná a bezpečná. Od toho sa tiež odvíja, aké testy sú medzi nefunkčné zahrnuté.

Test výkonnosti má za úlohu overiť, že aplikácia je schopná spoľahlivo pracovať aj pri náraste záťaže, napríklad v zmysle zvýšeného počtu súčasne pracujúcich používateľov alebo súčasne spracovávaných úloh atď. Existujú rôzne nástroje, ktoré dokážu simulovať práve túto záťaž, a to aj s ohľadom na jej reálne rozloženie v rámci celého dňa (teda napríklad nízka prevádzka v noci, opakované vrcholy v priebehu dňa atď.). Pri takto narastajúcej záťaži by sa nemali výraznejšie predlžovať odozvy aplikácie na vstupné otázky. Výkonnostné (performance) testy potom skúmajú správanie aplikácie predovšetkým v medzných hodnotách a po ich prekročení.

Ruka v ruke s výkonnostnými (performance) testami idú záťažové testy. Reálne sa tieto testy robia súčasne a veľmi často sa medzi nimi ani nerozlišuje. Úlohou záťažových testov je overiť stabilitu aplikácie pri narastaní objemu dát, ktoré sú v jej rámci ukladané a spracovávané. Okrem nárastu súčasne pracujúcich používateľov môže práve existujúca dátová záťaž spomaliť prácu aplikácie napríklad pri vyhľadaní, sťahovaní a pod..

Dá sa povedať, že oba vyššie spomenuté typy testov majú za úlohu predovšetkým optimalizáciu aplikácie pre taký výkon a takú záťaž, ktorá je pre danú aplikáciu očakávaná.

Iným typom týchto testov sú bezpečnostné testy aplikácie. V dnešnej dobe tieto testy získavajú na dôležitosti. Najčastejším typom bezpečnostných testov sú testy penetračné. Tie spočívajú vo vykonávaní simulovaných útokov na aplikáciu s cieľom odhaliť možné slabé miesta. Tieto útoky sú vykonávané buď ručne alebo za pomoci rôznych nástrojov,

Existujú ale aj testy obnovy, použiteľnosti, inštalácie, dokumentácie, spoľahlivosti, podpory a mnoho ďalších [12].

1.5.2 *Údržba v rámci testovania softvéru*

Softvér a systémy musia byť po nasadení do produkčného prostredia naďalej udržiavané. Zmeny môžu predstavovať opravy defektov odhalených v produkčnom prostredí, odstránenie či zmena už dodanej funkčnosti alebo pridanie novej funkčnosti. Údržba je tiež potrebná na zlepšenie alebo zachovanie nefunkcionálnych charakteristík kvality komponentov alebo systému počas jeho životnosti, obzvlášť výkonnostná efektivita, bezpečnosť, spoľahlivosť a prenositeľnosť. Ak sa v rámci údržby vykonajú akékoľvek zmeny, malo by sa vykonať údržbové testovanie a to na vyhodnotenie úspešnosti, s akou boli zmeny vykonané, a aby sa skontrolovali možné vedľajšie účinky v častiach systému, ktoré zostávajú nezmenené. Údržba môže zahŕňať plánované vydanie (release) alebo neplánované vydanie (hotfix). Údržbové vydanie (maintenance release) môže vyžadovať údržbové testovanie na viacerých testovacích úrovniach s použitím rôznych typov testov a to v závislosti od rozsahu zmeny [23].

1.6 Výber vhodného životného cyklu vzhľadom na zabezpečenie kvality

V rámci riadenia kvality je dôležitý aj výber správneho životného cyklu pre konkrétny projekt. Výber správneho životného cyklu nie je jednoduchý a odvíja sa od väčšieho počtu faktorov, napr. dimenzie, prepracovanosť zadania, rozsah projektu či typ klienta. Pre zaistenie kvality softvéru je správny výber životného cyklu kľúčový.

Existujú tri úrovne dimenzií, podľa ktorých môžeme zvoliť vhodný životný cyklus. Tieto dimenzie rozdeľujeme podľa istoty, ktorú daný softvérový projekt má:

- istota produktu – istota je určená dvoma faktormi a to, či sú užívateľské požiadavky jasne špecifikované (funkcionalita a kvalita) a aká je zraniteľnosť týchto požiadaviek,
- istota procesu – istota je určená faktormi možností presmerovať vývojový proces, stupňom merateľnosti procesu, znalosťou efektu riadiacich akcií, stupňom použitia nových, neznámymi nástrojmi,
- istota zdrojov – hlavným faktorom je dostupnosť správnych kvalifikovaných pracovníkov [18].

Pokiaľ máme istý produkt, proces a najlepšie aj zdroje, máme teda všetky podklady a dobré zadanie práce, môžeme sa pustiť do vývoja pomocou vodopádového modelu. Zaručiť kvalitu v tejto dimenzii je najjednoduchšie, pretože môžeme podľa zadania jasne

definovať procesy a metódy, a tak správne vyvinúť dielo, ktoré bude týmto postupom jednoducho kontrolovateľné a testovateľné.

Ak nie je proces istý, ale máme istý aspoň produkt, môžeme ho vyvíjať inkrementálne alebo iteratívne. Tento prístup je tiež dobré zvoliť pri rozsiahlejších projektoch s menej podrobným zadaním. Rozdelením diela do menších celkov (ako z hľadiska rozsahu, tak funkčnosti) je uľahčená analýza daného celku, ako aj jeho následný vývoj a testovanie, ktoré tak môžu byť špecifickejšie a presnejšie pre daný celok. V tejto dimenzii je tiež možné kvalitu dodržať.

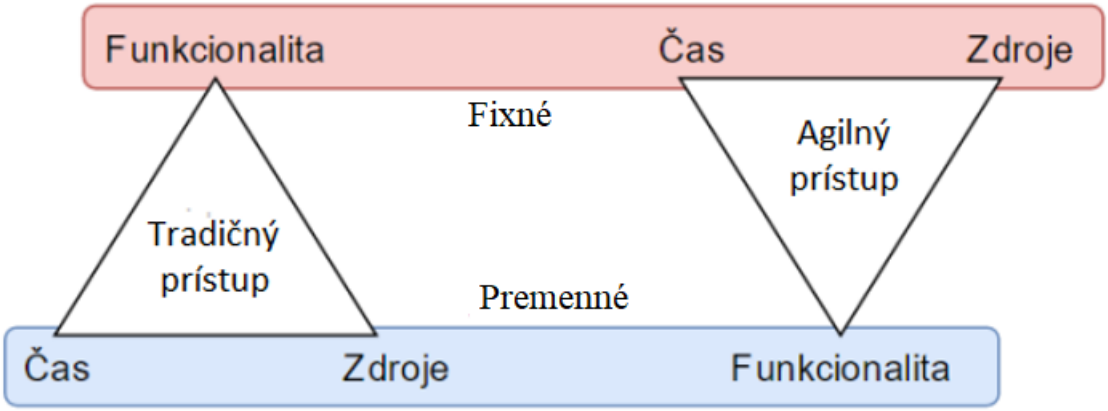
Najmenej výhodný projekt je pri neistote produktu, procesu aj zdrojov. Do takýchto projektov sa väčšinou púšťajú iba výskumné organizácie, kde najlepším vývojovým postupom je výskumník. V tejto dimenzii je kvalita premenná a záleží na schopnostiach tímu, či bude dodržaná alebo nie [17].

1.7 Metodiky vývoja softvéru

V tejto časti sa zameriame na vysvetlenie metodiky vývoja softvéru. Riadenie kvality je závislé aj na vybranej metodike vývoja. Jedná sa o súhrn postupov, pravidiel a nástrojov používaný pre návrh, plánovanie a riadenie vývoja softvéru. Metodika je len šablóna, musíme ju prispôsobiť firme, tímu, projektu a účelu (vývoj, prevádzka, prispôsobenie). Metodiky môžeme rozdeliť do dvoch veľkých skupín, a to tradičné (rigorózne) a agilné.

Tradičnými prístupmi sa myslia také prístupy, ktoré majú podrobne a presne definované procesy a pomerne negatívne pristupujú k zmenám v priebehu vývoja softvéru. Presne stanovujú procesy a požiadavky. Predpokladajú, že tvorbu softvéru je možné presne definovať, popísať a opakovane realizovať. Tradičné metodiky je vhodné použiť pri štandardných a veľkých projektoch.

Na rozdiel od tradičných prístupov, kde sa od zákazníka často očakáva špecifikácia požiadaviek už na samotnom začiatku projektu a zmena po odovzdaní produktu býva veľmi nákladná. Agilné prístupy operujú s faktom, že sa preferencie a potreby zákazníkov môžu meniť veľmi rýchlo a snažia sa na to zareagovať. Tieto metodiky je vhodné použiť pre projekty s nejasným alebo meniacim sa zadaním, pre menšie tímy.



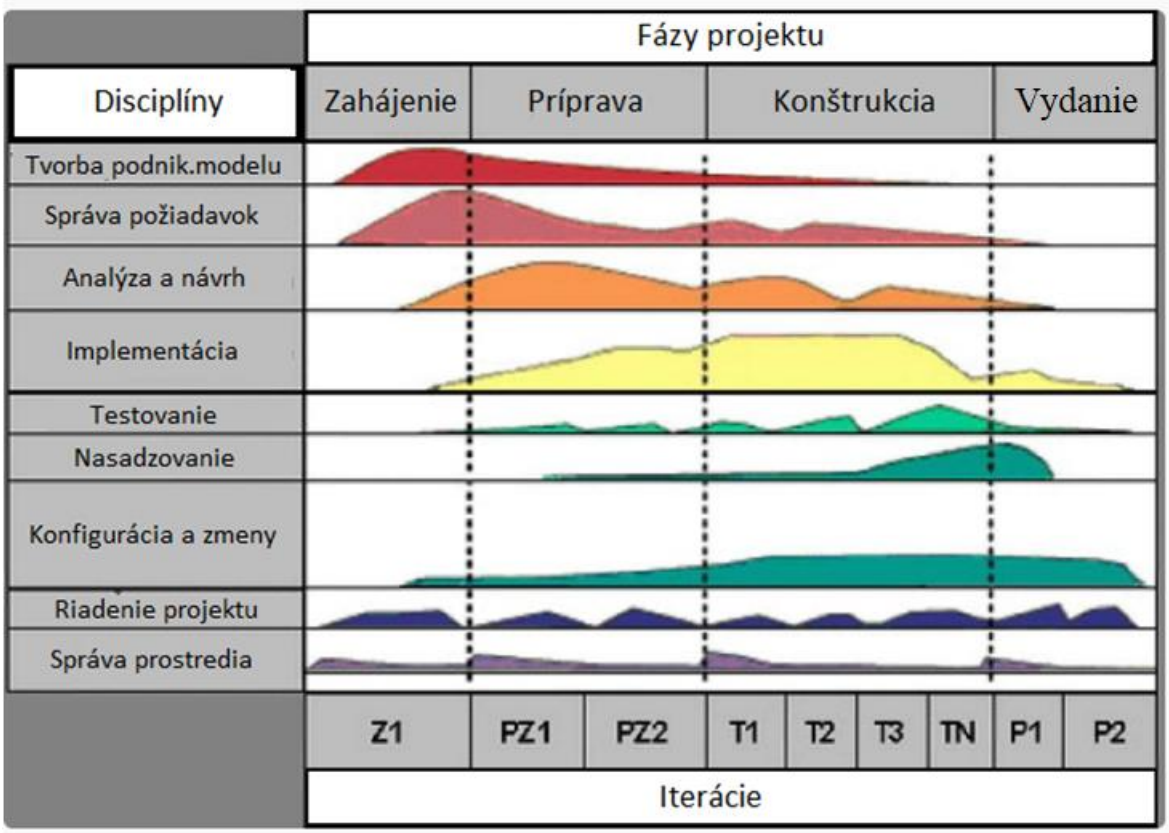
Obrázok 6 Porovnanie tradičného a agilného prístupu [20]

Tradičné projektové riadenie na začiatku definuje požadovanú funkcionalitu, ktorú považuje za fixnú. Podľa nich odhaduje čas a zdroje potrebné na realizáciu, ktoré sú premennými veličinami. Naopak, agilné riadenie projektov považuje za fixný čas a zdroje a premennou veličinou je funkcionalita, ktorá je prispôsobená prioritám zákazníka .

1.7.1 *Tradičné metodiky*

Tradičné metodiky môžeme rozdeliť na dva typy – riešené pomocou štrukturálnej analýzy alebo riešené pomocou objektovo orientovanej analýzy. Medzi metodiky so štrukturálnou analýzou patria Structured Systems Analysis and Design Method (SSADM) a Yourdon: Modern Structured Analysis (YMSA) a medzi metodiky s objektovo orientovanou analýzou patrí Rational Unified Process (RUP) a Unified Process (UP) [19].

My si bližšie popíšeme Rational Unified Process, keďže je podľa nás najviac využívaní v podnikovej praxi. RUP je objektovo orientovaný iteratívny prístup k životnému cyklu softvéru. RUP je vhodnejší skôr pre rozsiahlejšie projekty a väčšie vývojové tímy, lebo kladie dôraz na analýzu a dizajn, plánovanie, riadenie zdrojov a dokumentáciu. Na modelovanie procesov sa využíva prostriedky jazyka UML.



Obrázok 7 Unified process [21]

Na obrázku č. 7 je zobrazený priebeh práce na projekte. Zvislá os obsahuje vybrané procesy vrátane vyjadrenia objemu pracovných kapacít rezervovaných na danú činnosť v danej chvíli. Na vodorovnej osi sú zanesené štyri fázy metodiky RUP. Prvá fáza je začatie (inception) a definuje účel, rozsah projektu a jeho obchodný kontext. Vo fáze prípravy (elaboration) je potrebné analyzovať požiadavky zákazníka, celého projektu a definovať základy architektúry. Konštrukčná fáza (construction) je najdlhšia, pretože tu prebieha tvorba zdrojových kódov. V poslednej fáze odovzdania (transition) môže byť projekt odovzdaný zákazníkovi alebo do ďalšieho cyklu. Každá z týchto fáz môže byť rozdelená do viac častí zvaných iterácie. Pred začatím novej iterácie musia byť splnené skôr definované kritériá predchádzajúcej iterácie [21].

1.7.2 *Agilné metodiky*

Agilných metodík je niekoľko, patrí medzi ne Scrum, extrémne programovanie, vývoj riadený vlastnosťami (Feature Driven Developement – FDD), vývoj riadený testami (Test Driven Development – TDD) a ďalšie [22]. Na základe konzultácií s manažmentom

a skúseností z podnikovej praxe sa zameriame na metodiku Scrum, ktorú je bližšie popísaná v kapitole 2.2.

1.8 Porovnanie RUP a Scrum vzhľadom na zabezpečenie kvality

Ako sme už spomínali v kapitole 1.7. poznáme tradičné a agilné metodiky. Medzi tradičné metodiky patrí Rational Unified Process (RUP), ktorý sme si podrobnejšie popísali. Medzi agilné metodiky patrí Scrum. Obe tieto metodiky využívajú iteratívne a inkrementálne postupy, ktoré sa ale líšia obsahom iterácií. V scrume sa na začiatku zhromaždia všetky prioritné požiadavky, ktoré následne v každej iterácii zapracovávajú. Každá iterácia obsahuje všetky fázy vývoja ako sú analýza, návrh, implementácia, testovanie a odovzdanie, ktoré sú späté s jednou časťou softvéru. Naproti tomu v Rational Unified Process sa každá iterácia zameriava na konkrétnu fázu vývoja. Ostatné fázy môžu byť tiež zakomponované, ale v menšom rozsahu. Príkladom je fáza prípravy, v ktorej sa čiastočne ešte dolaďujú požiadavky a zároveň sa pracuje na návrhu. Taktiež sa vo fáze prípravy začína s implementáciou, pri ktorej dochádza k testovaniu prvých častí. RUP sa teda pozerá na softvér ako na celok a podľa toho ho tiež posudzuje.

Keďže sa veľmi často stáva, že klient požaduje zmenu zadania v priebehu vývoja, je z hľadiska zaistenia kvality lepšie všeobecne použiť agilnú metodiku, ktorá je na zmeny pripravená. Vie teda reagovať na zmeny ako na strane vývoja, tak na strane testovania a kontroly. Toto zaistenie býva ale často časovo aj finančne náročnejšie [10].

2 Cieľ a metodiky práce a metódy skúmania

2.1 Ciele práce a metódy výskumu

Cieľom diplomovej práce je zmapovanie rôznych prístupov merania kvality softvéru a informačných systémov v podnikovej praxi z rôznych hľadísk s aspektom na vývoj softvéru. Na základe tohto cieľa sa budeme podrobnejšie venovať metrikám merania softvéru z rôznych pohľadov v rámci vývoja softvéru s cieľom zabezpečiť kvalitu vyvíjaného softvéru, pričom sa zameriavame už na konkrétnu metodiku, ktorú si neskôr rozoberieme aj z praktickej časti.

K splneniu hlavného cieľa diplomovej práce bolo potrebné stanoviť čiastkové ciele:

- Porozumieť, čo zaznamená kvalita softvéru a ako k nej firmy pristupujú,
- Prieskum certifikátov, ktoré opisujú riadenie kvality softvéru a charakteristík, ku ktorým firma musí pristúpiť a dodržať ich, aby zabezpečila kvalitu dodávaného produktu,
- Analýza techník zaistenia a riadenia kvality softvéru,
- Analýza súčasných metodík vývoja softvéru,
- Porovnanie metodík a postupov vývoja softvéru,
- Analýza metrík z rôznych oblastí vývoja softvéru z dostupných zdrojov,
- Analýza metrík z rôznych oblastí vývoja softvéru v podnikovej praxi,
- Prieskum a analýza nástrojov slúžiacich na meranie kvality softvéru,
- Na základe syntézy poznatkov z analýz vytvoriť návrh metrík merania kvality softvéru so získanými dátami z podnikovej praxe, ktorými môžeme docieliť vhodnú kvalitu softvéru.

Pre splnenie vopred stanovených cieľov práce boli zvolené nasledujúce metódy výskumu:

- Prieskum a syntéza dostupných internetových a knižných zdrojov, ktoré opisujú kvalitu softvéru a ako k nej pristupovať, respektíve ju zabezpečiť,
- Pre analýzu metrík merania kvality softvéru v podnikovej praxi sme oslovili troch manažérov z troch rôznych vývojových firiem a následne s nimi viedli konzultácie, aby sme lepšie porozumeli ich prístupu ku kvalite softvéru v podnikovej praxi a urobili porovnanie voči poznatkom získaných z prieskumu a analýzy dostupných zdrojov,

- Analýza získaných dát z jednotlivých firiem za cieľom dedukcie a porozumenia metrikám merania kvality softvéru v podnikovej praxi.
- Komparácia získaných metrík merania kvality softvéru a následný návrh metrík pre firmy za účelom zabezpečenia kvalitne dodávaného produktu.

2.2 Metodika práce

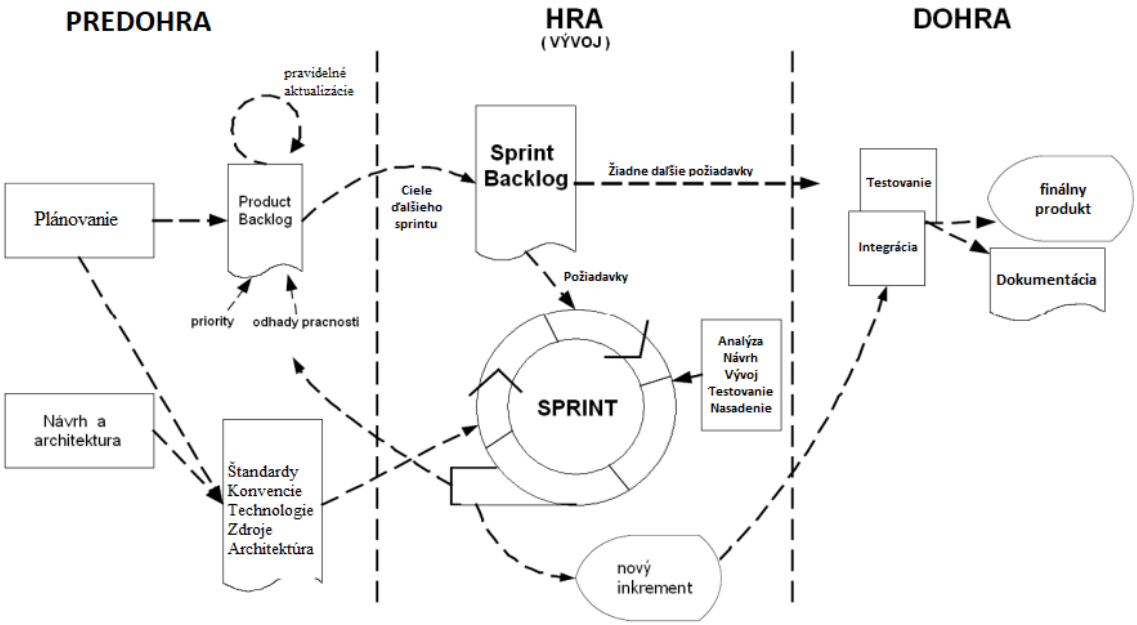
V kapitole 1.7.2 sme uvádzali niekoľko agilných metodík, ktoré tímy môžu využívať pre riadenie vývoju softvéru. Pre metodiku práce sa zameriame na metodiku Scrum, ktorú sme si zvolili na základe analýzy získaných poznatkov z teoretickej časti a konzultácií s manažérmi v podnikovej praxi. Scrum je metodika, ktorá je vhodná pre tímy o menšom počte, ktoré majú každý deň krátku schôdzku, tzv. daily scrum. Tieto stretnutia suplujú centrálne plánovanie. Každý člen tu referuje o svojej činnosti z minulého dňa a na aké problémy narazil. Následne informuje tím o tom, čo bude robiť dnes. Metodika presadzuje iteratívny vývoj. Obdobie iterácie sa nazýva sprint a zvyčajne trvá 2-4 týždne, no jeho dĺžku si každý tím prispôsobuje podľa seba. Výsledkom sprintu je demo vzniknutých úprav, ktoré je predvedené zákazníkovi. Ten poskytne spätnú väzbu, čo umožňuje rýchlo reagovať na zmeny v požiadavkách [22].

Na základe dĺžky sprintov, vytvorenia demo verzie, ktorá sa prezentuje zákazníkom s cieľom získať jeho spätnú väzbu a tým pohotovo reagovať na ďalšiu iteráciu vývoja softvéru môžeme tvrdiť, že výsledky práce vidíme omnoho skôr ako pri vodopádovom modely.

Scrum sa delí na niekoľko artefaktov. Prvým artefaktom je tzv. Product Backlog, do ktorého sa postupne umiestňujú všetky užívateľke príbehy (user stories) bez hierarchického členenia. Jedná sa o požiadavky kladené na samotný softvér zákazníkom. Druhým artefaktom je už spomínaný tzv. Daily scrum. Tretím artefaktom je tzv. Sprint Backlog, ktorý obsahuje už len konkrétne užívateľské príbehy, ktoré sa plánujú v rámci sprintu splniť. V rámci projektového tímu dochádza aj k tzv. sprint retrospektíve. Jedná sa o ďalšie stretnutie, ktorého cieľom je zhodnotiť priebeh ukončeného sprintu. Hodnotia sa slabé a silné stránky. Cieľom je teda zaisťovať kontinuálne zlepšovanie.

Všetky užívateľské príbehy sú ocenené, čo sa týka časovej náročnosti a je im nastavená priorita. S ubúdajúcim časom sprintu by mal aj ubúdať odhadovaný čas práce na jednotlivých položkách. Na konci sprintu by mali byť všetky užívateľské príbehy implementované a plne otestované.

V prípade, že tomu tak nie je, môžeme urobiť nápravné akcie. Každý sprint by mal byť vždy plne funkčný a implementovať všetky požiadavky vybrané pre daný sprint. Jedným z hlavných dôvodov veľkej obľuby scrumu je možnosť dennej vizualizácie priebehu každého sprintu. Graf nazývame Iteračný graf zvyškového trendu (BurnDown Chart). Údaje k jeho vytvoreniu sú dostupné, pretože jednotlivé užívateľské príbehy sú časovo ohodnotené a na konci každého dňa tím vývojárov zaznamená zmeny ich časového plnenia. Jednoducho tak môžeme porovnať plánovaný čas s tým aktuálnym [36]. Na obrázku č. 8 môžeme vidieť riadenie produktového vývoja na základe metodiky Scrum.



Obrázok 8 Scrum [22]

3 Výsledky práce

Na základe získaných poznatkov môžeme tvrdiť, že na zabezpečenie kvality softvéru sa podieľa množstvo aktérov, od dizajnérskeho tímu cez testerov, vývojárov, analytikov, ale aj koncového používateľa a manažment.

V tejto kapitole sa venujeme zmapovaniu metrík merania kvality softvéru z dostupných internetových a knižných zdrojov a návrhu konceptu prístupu ku kvalite softvéru. Na začiatok budeme popisovať akým životným cyklom by sa mala firma uberať, aké certifikáty by si mala zabezpečiť, aké metriky merania kvality sme zmapovali a nakoniec aké metriky merania kvality softvéru navrhujeme v podnikovej praxi, aby firma vytvorila čo najkvalitnejší softvérový produkt. Kvalita softvérového produktu môže byť hodnotená, analyzovaná a vnímaná veľmi odlišnými spôsobmi, a napriek tomu všetky tieto pohľady popisujú ten istý produkt.

3.1 Zvolený životný cyklus pri vývoji softvéru

Za účelom výskumu sme oslovili troch manažérov z troch rôznych firiem, aby sme čo najlepšie porozumeli pohľadu, akým sa firmy pozerajú na kvalitu softvéru a následne na základe získaných poznatkov vytvorili zovšeobecnený koncept, ktorým by sa mala riadiť každá vývojová firma.

Na základe konzultácií s manažérmi a získaných poznatkov z teoretickej časti môžeme tvrdiť, že najviac využívaný životný cyklus firiem je inkrementálny a iteratívny model. Pre jeho dodržiavanie tímy využívajú agilný vývoj, konkrétne metodiku Scrum.

Dôvodom využívania práve tohto prístupu je fakt, že v dnešnej dobe sa musia firmy neustále prispôsobovať požiadavkám zákazníka. Z dlhodobého hľadiska je vývoj softvéru na základe vodopádového modelu citlivý na zmeny a môže ľahko dospieť do situácie, kedy firma už nevie garantovať zákazníkovi kvalitu dodávaného softvéru. Naopak Scrum bol vytvorený pre zvládanie týchto situácií a vďaka nemu sa vie firma jednoducho prispôsobiť zmenám požiadaviek zákazníka.

3.1.1 *Riadenie tímu prostredníctvom metodiky Scrum*

Výhoda scrumu spočíva v každodennej komunikácii medzí tímami prostredníctvom denných stretnutí (daily scrums). To zabezpečuje, že jednotlivý členovia tímu prezentujú výsledky predchádzajúceho dňa a ich plány na daný deň. Okrem toho spomínajú aj problémy

s akými sa stretli a tie následne spracúva a rieši vedúci tímu (scrum master). Túto charakteristiku scrumu vnímame ako dôležitý aspekt pri zabezpečovaní kvality softvéru. Ďalšou dôležitou súčasťou scrumu sú aj už spomínané sprinty. Sprint môžeme chápať ako časový úsek počas ktorého sa vyvíja a následne dodáva verzia produktu. Táto verzia je následne prezentovaná koncovému zákazníkovi, aby mohol posúdiť, či boli splnené jeho očakávania a zhodnotiť doterajší priebeh vývoja. Ako ďalšiu kľúčovú charakteristiku scrumu je nutné spomenúť retrospektívu sprintu v rámci tímu. V rámci nej jednotliví členovia komunikujú svoje postrehy a problémy zachytené v rámci sprintu vedúcemu tímu s cieľom zlepšovať ich proces fungovania a tým zabezpečiť lepšiu kvalitu vyvíjaného produktu.

3.2 Certifikácie firmy odzrkadľujúce kvalitu vývoja softvéru

Na základe analýzy zdrojov z teoretickej časti a konzultácií v podnikovej praxi môžeme tvrdiť, že je dôležité, aby firmy okrem meraní kvality softvéru spĺňali kritéria pre certifikáciu ISO normami. Ako sme už spomínali v kapitole 1.2, existujú rôzne normy zamerané na kvalitu softvéru. Certifikácia pre ISO normy - ISO 9000, ISO 9001 a ISO/IEC 25010 je kľúčová pre dôveru zákazníka voči firme a firma by mala klásť dôraz na ich prísne dodržiavanie. Ako sme už spomínali, ISO 9000 definuje systém manažérstva kvality. Firma certifikovaná touto normou indikuje pre zákazníka, že je schopná preukázať schopnosť riadenia vývoja a distribúcie softvéru. ISO 9001 ako súčasť normy ISO 9000 hovorí o tom, že firma má správne nastavené základné riadiace procesy v organizácii, ktoré neustále pomáhajú zlepšovať kvalitu poskytovaného softvéru, ich schopnosť sa adaptovať zákazníkovi a strategicky riadiť tím a prácu s rizikami. V neposlednom rade je potrebné spomenúť aj normu ISO/IEC 25010, ktorá je súčasťou rady štandardov s názvom Software product Quality Requirements and Evaluation. Predstavuje model kvality produktu pozostávajúci z faktorov ovplyvňujúcich kvalitu softvéru. V práci sa budeme zaoberať metrikami merania kvality softvéru, a preto je táto norma, ktorá sa zameriava na kvalitu produktu pre nás najdôležitejšia. Z tohto dôvodu považujeme za potrebné vysvetliť fungovanie ISO/IEC 25010 v praxi. Ako už bolo spomenuté v kapitole 1.2.2 táto norma pozostáva z ôsmich charakteristík, ktoré sa ďalej členia na indikátory kvality softvéru. Medzi tieto charakteristiky patria:

- **Funkčnosť –** (Functional suitability) – Z praktického hľadiska táto charakteristika hovorí o tom, do akej miery produkt alebo systém poskytuje funkcionalitu zodpovedajúcu stanoveným a vyplývajúcim potrebám zákazníka za stanovených

podmienok. Môžeme si to predstaviť ako dokumentáciu medzi zákazníkom a firmou, ktorá definuje biznis požiadavky, z ktorých sa počas vývoja tvorí detailná funkčná špecifikácia. Funkčná špecifikácia ďalej slúži na to, aby sa voči nej pri záverečnom testovaní overovalo, či systém spĺňa všetky požiadavky na funkčnosť a na špecifikáciu funkčnosti.

- **Výkonnosť–** (Performance efficiency) – V praxi znamená výkon potrebný k množstvu použitých zdrojov na vykonávanie funkcií za stanovených podmienok (zdroje sa myslia napr. hardvérové a softvérové vybavenie a ich konfigurácie). Zákazník napríklad definuje koľko používateľov bude pristupovať k systému a firma dodávajúca systém musí splniť túto požiadavku. Na jej overenie sa využíva záťažové testovanie (perfomance testing).
- **Kompatibilita –** (Compatibility) - Z praktického hľadiska táto charakteristika hovorí o tom, do akej miery produkt, systém, alebo jeho súčasť dokáže vykonať výmenu informácií s ostatnými produktmi, systémami, alebo jeho časťami a vykonať požadované funkcie, zatiaľ čo zdieľa rovnaké hardvérové a softvérové prostredie,
- **Použiteľnosť –** (Usability) – Pri použiteľnosti sa naopak pozeráme na to, do akej miery môže byť produkt, systém použitý efektívne, účinne a uspokojivo na dosiahnutie definovaných cieľov v špecifikovanom kontexte pre daného zákazníka. Z praktického hľadiska firma k tejto charakteristike pristupuje napríklad formou dizajnu. V rámci dizajnu sa kladie dôraz na užívateľskú skúsenosť (user experience) a užívateľsky priateľské prostredie (user friendly).
- **Spoľahlivosť –** (Reliability) – Spoľahlivosť nám hovorí, do akej miery systém, produkt, alebo jeho časť vykonáva špecifikované funkcie za stanovených podmienok. Z praktického hľadiska si môžeme predstaviť príklad, kde firma má nastavené infraštruktúralne prostredie, čiže nastavené prístupy efektívneho overovania (formou testov), aby zabezpečila spoľahlivosť vyvíjaného produktu.
- **Bezpečnosť –** (Security) – Bezpečnosť nám hovorí, do akej miery produkt, alebo systém chráni svoje informácie a dáta, aby osoby, či ostatné systémy nemohli s nimi neoprávnene pracovať, ak by mali neoprávnený prístup. V praxi túto charakteristiku pokrýva bezpečnostne testovanie (etické hackovanie).
- **Udržateľnosť –** (Maintability) – Táto charakteristika nám hovorí o tom, do akej miery efektívnosti a účinnosti môže byť produkt, alebo systém pozmenený a jeho schopnosť sa zmenám adaptovať. Znamená to, že systémová databáza musí byť

zabezpečená a dáta ukladať tak, aby brala do úvahy predpoklad, že tieto dáta budú časom rásť. Systém sa tak stane udržateľný (nebude spomalený a nevyskytnú sa časté výpadky).

- **Prenositeľnosť –** (Portability) – Prenositeľnosť hovorí o tom, do akej miery efektívnosti a účinnosti môže byť produkt, systém, alebo jeho časť prenesená z jedného hardvérového, softvérového, či iného operačného prostredia na iné. Z praktického hľadiska to znamená, že sa snažíme systém vytvoriť čo najuniverzálnejšie. Ako príklad môžeme uviesť systém udržiavajúci databázu na platforme Oracle, ktorý prechádza na Microsoft SQL Server na žiadosť zákazníka. Firma musí zabezpečiť, aby prenositeľnosť dát medzi dvoma platformami bola bezproblémová.

3.3 Prehľad metrík na meranie kvality softvéru

Kvalita softvéru musí byť posudzovateľná nielen empiricky, ale aj pomocou parametrov, ktoré je možné merať. Tieto spôsoby a nástroje merania sa nazývajú metriky. Metrika vyjadruje stav určitého systému, napríklad jeho kvality, efektívnosti a nadobúda pri tom rôzne hodnoty. Pre tento termín sa používajú špecializované pojmy ako Performance Indicator alebo Key Performance Indicator (KPI). Pri každom softvéri je potrebné špecifikovať konkrétne metriky, ktoré budú merané. [13].

3.3.1 *Kľúčové ukazovatele výkonnosti*

KPI je jedna z dôležitých súčasti k nadobudnutiu kvality softvéru. Vo všeobecnosti tieto metriky hovoria o tom, či sa organizácia približuje alebo vzďaľuje od stanovených cieľov. Dôležitou podmienkou pre správne nastavenie KPI je určenie kritických faktorov úspechu pre produkt.

Výsledkom praxe je, že mnohí manažéri a zamestnanci sledujú výsledky produktu, porovnávajú hodnoty a čísla. Význam týchto meraní a následne určenie toho, ako čo najlepšie zabezpečiť kvalitu softvéru sa neskôr stráca v zhluku čísel a grafov.

KPI musia byť merateľné, to znamená, že by mali hovoriť o výsledkoch a nie o činnostiach. Veľmi dôležité je to, aby boli definované zrozumiteľne, pretože by mohlo dôjsť k ich nepochopeniu.

Pokiaľ sú KPI nastavené správne, môže to zefektívniť prácu a tým ju aj uľahčiť. Na tvorbu KPI existujú podmienky známe pod skratkou SMART.

- **S** - Specific: Špecificky odrážať daný cieľ a umožniť sledovanie jeho dosiahnutia,
- **M** - Measurable: Keďže KPI zaisťujú merateľnosť cieľov, musia byť zvolené tak, aby sa v určitom časovom intervale dala sledovať ich hodnota,
- **A** - Acceptable: Ciele ako aj ich ukazovatele by mali byť prijateľné alebo dosiahnuteľné všetkými jednotkami podniku. Tým sú myslené nielen jednotlivé oddelenia, ale aj jednotliví zamestnanci.
- **R** - Realistic: Vízia by mala byť zvolená tak, aby bola zároveň dosiahnuteľná, ale aby k jej naplneniu bolo zároveň potrebné vyvinúť úsilie.
- **T** - Time Specific: Zasadenie do časového rámca je kľúčové. Ciele musia byť merateľné v čase. Malo by byť navrhnuté a zabezpečené, ako bude časovo prebiehať meranie KPI a reportovanie týchto hodnôt [24].

3.3.2 *Štandardné metriky kvality softvéru CISQ*

Spoločnosť CISQ, ktorá sa venuje metrikám kvality softvéru, definovala primárne charakteristiky pre kvalitu softvéru:

Spoľahlivosť softvéru je možnosť, že softvér bude fungovať bezchybne v určitom čase v určitom prostredí [30]. Spoľahlivosť môžeme merať spočítaním počtu chýb s vysokou prioritou nájdených v produkcii. Môžeme tiež použiť testovanie záťaže, ktoré hodnotí, ako dobre softvér funguje za bežných podmienok používania. Je dôležité poznamenať, že „bežné podmienky používania“ sa môžu meniť medzi nízkou a vysokou záťažou – ide o to, že takéto prostredia sa očakávajú [29].

Použiteľnosť sa zameriava na to, ako sa používatelia učia a používajú produkt na dosiahnutie svojich cieľov. Táto charakteristika platí aj pre spokojnosť koncového užívateľa s procesom.

Na zhromažďovanie týchto informácií, softvéroví profesionáli používajú rôzne metódy na získavanie spätnej väzby od používateľov, buď na existujúce webové stránky alebo na akékoľvek plány súvisiace s novým webovým umiestnením.

Ľahko použiteľné hodnotenie môže zahŕňať dva typy údajov: kvalitatívne údaje a kvantitatívne údaje. Prvé opisujú myšlienky a názory účastníkov hodnotenia, zatiaľ čo druhé ukazujú, čo sa deje [30].

Bezpečnosť je myšlienkou vývoja softvéru, ktorá môže stále normálne fungovať aj napriek škodlivým útokom. Cieľom zabezpečenia softvéru je vyhnúť sa bezpečnostným

zraniteľnostiam a poskytovaním bezpečnosti na začiatku životného cyklu vývoja softvéru. Bezpečnosť je v podstate manažment rizík [29].

Bezpečnosť môžeme merať tak, že zhodnotíme, ako dlho trvá oprava softvérových zraniteľností. Môžeme tiež skontrolovať skutočné bezpečnostné incidenty z predchádzajúcich verzií softvéru, vrátane toho, či bol systém narušený a či nejaké porušenia nespôsobili používateľom výpadky. Všetky predchádzajúce bezpečnostné problémy by sa samozrejme mali riešiť v budúcich vydaniach.

Efektívnosť je vlastnosť, ktorá vyjadruje pomer množstva vyvíjaného softvéru a použitých zdrojov (ako je čas, pracovné zaťaženie atď.).

Medzi niektoré metriky efektívnosti použité pri vývoji softvéru patria:

- **Čas stretnutí** - vedúci tímov alebo scrum master by mali porovnať plánovaný čas stretnutia a skutočný čas stretnutia,
- **Dodacia lehota** - množstvo času medzi začiatkom a koncom procesu. Závisí od kvality tímu a zložitosti projektu, ktoré priamo ovplyvňujú náklady na projekt,
- **Stredný čas na opravu a Stredný čas medzi poruchami** - Obe metriky merajú výkon softvéru v produkčnom prostredí. Keďže zlyhaniam softvéru sa takmer nedá vyhnúť, tieto softvérové metriky sa pokúšajú kvantifikovať ako dobre softvér obnovuje a uchováva údaje,
- **Miera zlyhania aplikácie** - Vypočítava sa vydelením počtu zlyhaní aplikácie počtom jej použití,
- **Účinnosť odstraňovania defektov (DRE)** - Používa sa na kvantifikáciu počtu chýb zistených koncovým používateľom po dodaní produktu v porovnaní s chybami zistenými pred dodaním produktu,
- **Záťažové testovanie (Soak testing)** - Slúži na meranie konkrétnych časov načítania stránky a schopnosti načítania kľúčových funkcií [30].

Udržateľnosť je jednoduchým meradlom počítania počtu riadkov kódu. Softvér s viacerými riadkami kódu sa ťažšie udržiava, čo znamená, že akákoľvek zmena povedie s väčšou pravdepodobnosťou ku chybám.

Na kontrolu zložitosti kódu sa používa niekoľko podrobných testovacích metrík, ako je napríklad cyklomatická zložitosť, ktorá počíta množstvo lineárne nezávislých ciest cez zdrojový kód programu.

Odporúčanie vydané NIST (National Institute of Standards and Technology) pre cyklomatickú zložitosť je, že hodnota nad 10 znamená potenciálne rizikovú kódovú základňu z hľadiska možných defektov. Nástroje na testovanie softvéru, ako je napríklad Visual Studio, môžu metriku cyklomatickej zložitosti merať za nás.

Rýchlosť dodavky môžeme skontrolovať spočítaním počtu vydaní softvéru. Ďalším meradlom je počet „príbehov“ alebo požiadaviek používateľa odoslaných používateľovi.

3.3.3 *Metriky kvality softvéru pre agilný vývoj*

Metriky agilného vývoja sa zameriavajú na to, ako agilné tímy prijímajú rozhodnutia a plánujú. V rámci agilného vývoja si vysvetlíme tri kľúčové metriky kvality softvéru, ktoré možno zabudovať do agilného rámca v každej fáze.

- **Iteračný graf zvyškového trendu (Burndown Charts)**, ktorý ukazuje koľko užívateľských príbehov bolo dokončených v každom sprinte a koľko zostáva do dokončenia funkcie alebo produktu. Ide o skvelý vizuálny nástroj na sledovanie postupu a kvality dokončenej práce v dlhšom časovom rámci,
- **Priemerná agilná rýchlosť (Average Agile Velocity)** meria priemer odovzdania taskov v Jire, tiketov alebo eposov, ktoré je tím schopný dokončiť v sprinte,
- **Miera otváranie/zatvárania (Open/Close Rate)** sa meria sledovaním produkčných incidentov, ktoré sa vyskytnú počas daného časového obdobia a ako rýchlo sa riešia,
- **Čas cyklu (Cycle time)** popisuje, ako dlho trvá zmena funkcionality systému a implementácia zmeny do produkcie [28].

3.3.4 *Metriky používané pri tvorbe kódu*

Súbory softvérových metrík sa často oznamujú tímom vývoja softvéru ako ciele. Medzi ciele patrí:

- **Redukcia riadkov kódov,**
- **Zníženie počtu hlásených chýb,**
- **Zvýšenie počtu iterácií softvéru,**
- **Urýchlenie dokončenia úloh.**

Zameranie sa na metriky ako ciele pomáha vývojárom softvéru dosiahnuť dôležitejšie ciele, ako je zlepšenie užitočnosti softvéru a používateľskej skúsenosti.

Napríklad softvérové metriky založené na veľkosti často merajú riadky kódu na označenie zložitosti kódovania alebo efektívnosti softvéru. V snahe znížiť zložitosť kódu môže manažment obmedzovať počet riadkov kódu, ktorý sa má napísať na dokončenie funkcií. V snahe zjednodušiť funkcie by vývojári softvéru mohli napísať viac funkcií, ktoré majú menej riadkov kódu, aby dosiahli svoj cieľ, ale neznížili tým celkovú zložitosť kódu a ani nezlepšili efektivitu softvéru. Medzi metriky používaní pri kódovaní môžeme zaradiť:

- **Aktívne dni** (Activity days) sú mierou toho, koľko času vývojár softvéru prispieva kódom do vývoja. Účelom tejto softvérovej metriky je posúdiť skryté náklady na prerušenia,
- **Rozsah úloh** (Assignment scope) je množstvo kódu, ktoré môže programátor udržiavať a podporovať za rok. Túto softvérovú metriku môžeme použiť na plánovanie počtu ľudí potrebných na podporu softvérového systému a porovnávanie tímov,
- **Kód churn** (Code churn) predstavuje počet riadkov kódu, ktoré boli upravené, pridané alebo odstránené v určenom časovom období.
- **Dopad** (impact) meria vplyv akejkoľvek zmeny kódu na projekt vývoja softvéru. Zmena kódu, ktorá ovplyvňuje viacero súborov môže mať väčší vplyv ako zmena kódu ovplyvňujúca jeden súbor,
- **Kvalita kódu** - Pri tejto metrike sa merajú kvantitatívne metriky, ako je počet riadkov, zložitosť, funkcie, miera generovania chýb a kvalitatívne metriky ako čitateľnosť, jasnosť kódu. Využívajú sa tu konkrétnejšie metriky merania ako cyklomatická zložitosť, Halsteadové metriky, hĺbka dedičnosti, spojenie tried, vnorenie, počet chýb na 1000 riadkov kódu atď.,
- **Pokrytie kódu** - Táto metrika vraví o pokrytí kódu jednotkovými (unit) testami. Vďaka vysokému pokrytiu kódu jednotkovými testami, znižujeme riziko vzniku defektu vo fáze testovania a potom následne na produkciu, čo nám zaručí vyššiu kvalitu softvéru [31].

3.3.5 *Postup merania kvality softvéru z hľadiska tvorby kódu*

Aby sme lepšie porozumeli postupu tvorby metrík merania kvality softvéru, použijeme príklad z hľadiska tvorby kódu, ktorý uvádza Flavien Huynh [27].

Kvalita softvéru je viac ako predchádzanie chybám. Dáva nám to tiež príležitosť dosiahnuť lepší kód. Od rôznych úrovní kvality až po rôzne stimuly na hodnotenie kvality existuje mnoho ciest ku kvalite softvéru.

Našťastie existuje mnoho nástrojov, ktoré vytvárajú metriky kvality softvéru: analýzou štruktúry a zložitosti kódu, kontrolou súladu s pravidlami zo štandardu, overovaním pokrytia, meraním časov vykonávania atď.

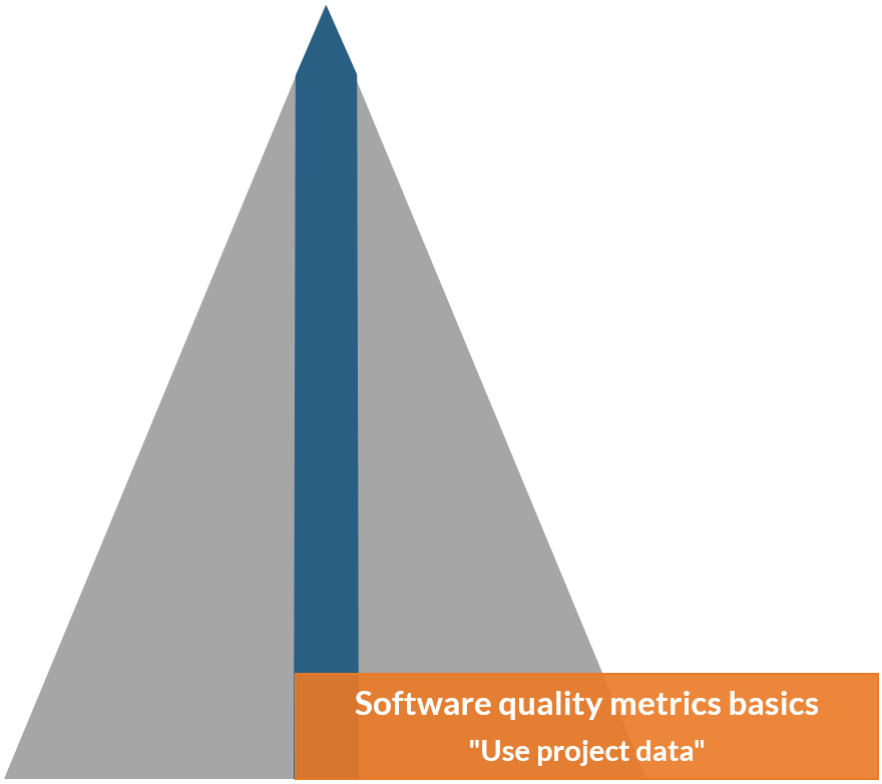
Pozrime sa, ako môžu byť tieto metriky základnými kameňmi na výpočet agregovaných cieľov kvality, ktoré pomôžu zamerať sa na kód, ktorý si vyžaduje pozornosť, a definovať kontroly kvality na dosiahnutie zdravých návykov kvality [27].

Úroveň 1: Metriky kvality softvéru

Prvým a najpriamejším spôsobom hodnotenia kvality je meranie kvality softvéru. V podstate ide o projektové merania pochádzajúce priamo z analytických nástrojov spustených na zdrojovom kóde. Jedná sa o analyzátory statických alebo dynamických kódov, kontroly programovacích pravidiel, alebo riešenia pokrytia testov. Tieto nástroje generujú presné informácie pre každú funkciu alebo súbor v projekte a využívajú sa s veľkým prínosom.

Všetko, čo musíme urobiť, je analyzovať denníky (logy) alebo prehľady (dashboards) týchto nástrojov a overiť, či metriky spĺňajú očakávané prahové hodnoty.

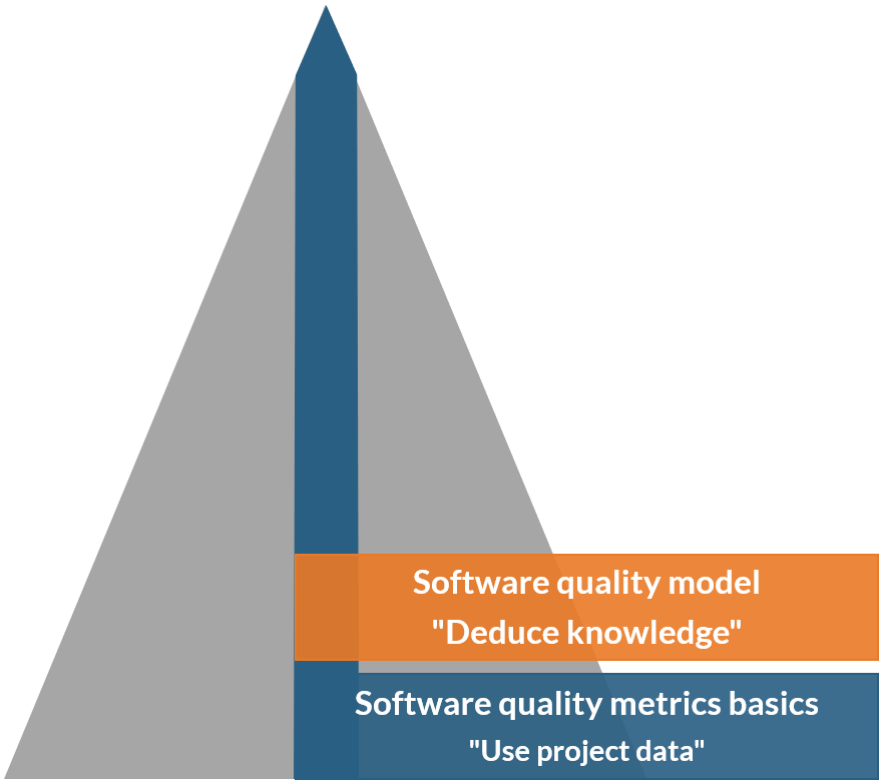
Dá sa to celkom ľahko uviesť do praxe, ale pri veľkých projektoch sa to môže stať skľučujúcou úlohou. Desiatky tisíc riadkov kódu môžu produkovať obrovské množstvo metrík.



Obrázok 9 Základy metrík kvality softvéru „Použité projektové dáta" [27]

Úroveň 2: Model kvality softvéru

Primárnou funkciou modelu kvality je poskytovať ukazovatele kvality na vysokej úrovni, ktorých výpočet je založený na metrikách kvality. To znamená, že tieto ukazovatele premietnu kvalitné osvedčené postupy do schválených a spoľahlivých výpočtov. Indikátor dobrej kvality pomôže zistiť, ktorá časť kódu si vyžaduje pozornosť, bez toho, aby sme museli analyzovať všetky metriky pri hľadaní odľahlých hodnôt.



Obrázok 10 Model kvality softvéru „Vyvodenie poznatkov“ [27]

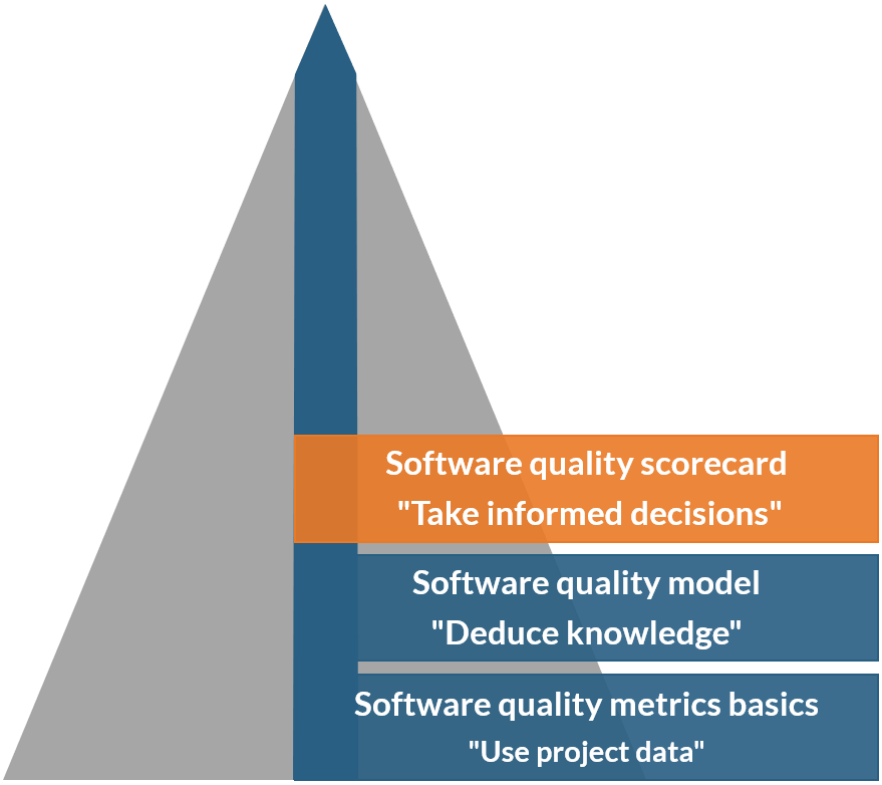
Úroveň 3: Prehľad kvality softvéru

Karta hodnotenia (scorecard) kvality je ako konečné hodnotenie odborníka po dôkladnej analýze. Ak dôverujeme odborníkovi, toto hodnotenie nám poskytne hodnotenie kvality kódu ako celku. Prehľad výsledkov by však nemal kvalifikovať len tip softvérového projektu, ale musí kvalifikovať každý súbor, triedu, funkciu. Keď hovoríme o dôvere, hodnotiaca karta je plne založená na výsledkoch modelu kvality softvéru, ktorý je sám o sebe implementáciou postupov dobrej kvality.

Výhoda hodnotiacej karty kvality je v tom, že poskytuje syntetizovanú víziu projektu a kontroluje či je v dobrom zdravotnom stave z hľadiska kvality. Umožňuje tiež rýchlu detekciu častí kódu, ktoré si vyžadujú pozornosť z dôvodu zlého alebo zhoršujúceho sa hodnotenia.

Vyššie sme popísali, že motiváciou kvality softvéru už nie sú len chyby, ale lepšia kvalita kódu. Okrem tohto zámeru je kvalita očakávaným výsledkom reťazca výroby softvéru. A toto očakávanie sa nekončí pri vývojárskom tíme. Niekoľko úrovní zainteresovaných strán môže vyžadovať špecifické úrovne kvality na overenie softvérového produktu.

V dôsledku toho sa musí kvalita softvéru sledovať, monitorovať, hlásiť a odôvodňovať.

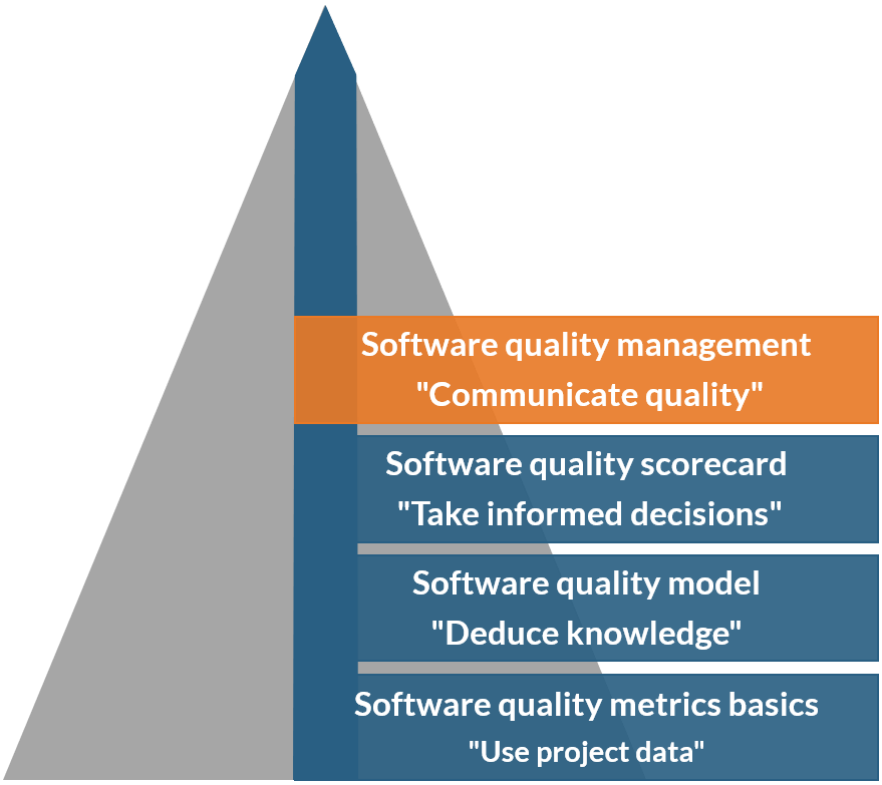


Obrázok 11 Prehľad kvality softvéru „Prijímať informované rozhodnutia“[27]

Úroveň 4: Manažment kvality

Kvalita je tiež v očiach diváka. Z pohľadu vývojového tímu, projektového manažéra, spoločnosti alebo externého klienta je kvalita kódu niečo iné. Všetky tieto atribúty kvality musia byť súčasťou kvalitného projektu a musia byť riadené ako každý iný projekt, od presného testovania na úrovni funkcií až po štandardnú zhodu, úroveň zrelosti alebo špecifické ukazovatele v odvetví.

Bez ohľadu na úroveň (metriky, model alebo hodnotiace karty), riadenie kvality softvéru odpovedá na potrebu sledovať a komunikovať aktuálny stav kvality projektu stranám, ktoré nie sú nevyhnutne zapojené do každodenného životného cyklu projektu. Ide o citlivú úlohu, ktorá môže míňať cieľ, ak vytvára príliš veľké a veľmi podrobné správy.

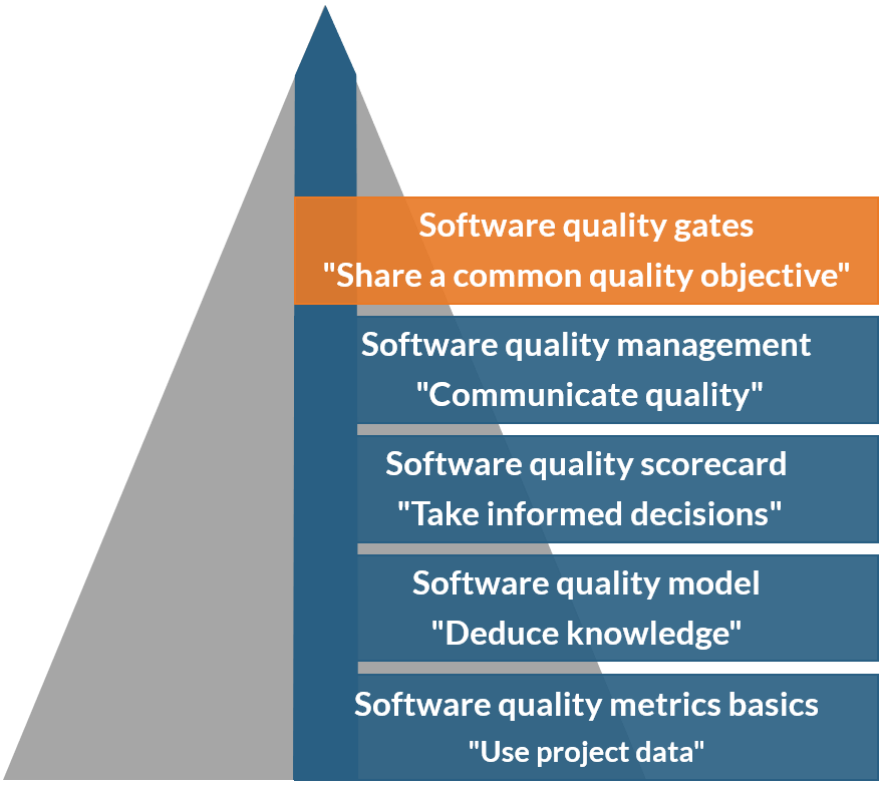


Obrázok 12 Manažment kvality softvéru „Komunikovať kvalitu“ [27]

Úroveň 5: Kvalitné brány

Kvalitu softvéru môžeme vidieť ako dohodu medzi dvoma stranami a rýchlo si predstaviť príklady. Projektový manažér skontroluje, či vývojový tím dodržiava programové pokyny, a potom overí súlad so štandardom. Audítor má k dispozícii kontrolný zoznam ukazovateľov na overenie. Klient potrebuje vedieť, či sú splnené požiadavky na kvalitu projektu.

Softvérové brány kvality riešia všetky tieto prípady ako špecifické overenia hodnotenia kvality projektu. Nenahrádza ani neruší bežnú správu o kvalite, ale funguje ako kontrolný bod na rýchlu odpoveď na otázku „Spĺňa projekt dohodu?“. Samozrejme, dohoda musí byť definovaná od začiatku, ale aj na to slúži brána kvality. Ak chceme od začiatku zdieľať to, čo sa kontroluje.

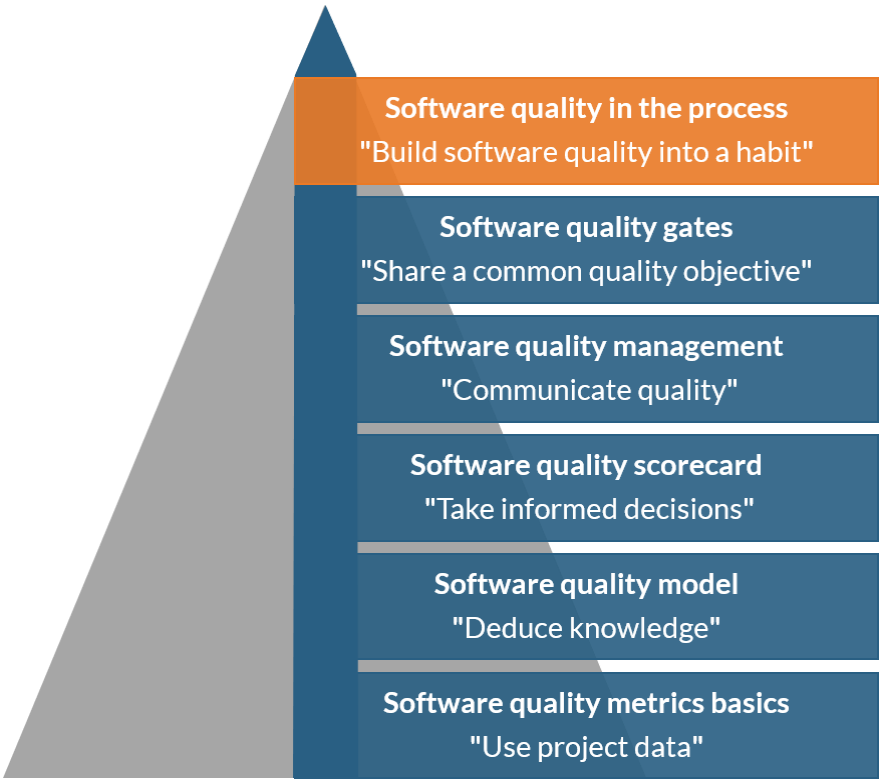


Obrázok 13 Brány kvality softvéru „Zdieľať spoločný cieľ kvality“ [27]

Najvyššia úroveň: Kvalita v procese

Kvalita softvéru môže byť nákladná, ak sú tímy mobilizované pri každom audite kvality alebo ak proces stanovuje fázy hodnotenia kvality pred každým vydaním. Budú sa vytvárať správy o kvalite, budú sa prekračovať brány, ale cenou môže byť strata motivácie a nedôvera vo výhody riadenia kvality softvéru.

Neexistuje žiadne tlačidlo „Dosiahnuť kvalitu teraz“, ale zavedenie postupov kvality do procesu vývoja, udržiavanie povedomia o kvalite pravidelným informovaním o hodnotení kvality je dobrý spôsob, ako dosiahnuť vstavený scenár kvality [27].



Obrázok 14 Kvalita softvéru v procese „Postaviť kvalitu softvéru do zvyku“ [27]

3.3.6 *Metriky používané pri testovaní*

Z hľadiska testovania existujú metriky, ktoré zabezpečujú kvalitu softvéru. Metriky sa môžu zbierať počas a na konci testovacích činností s cieľom posúdiť:

- pokrok prác voči rozpočtu a plánovanému harmonogramu,
- aktuálnu kvalita testovaného objektu,
- vhodnosť zvoleného prístupu k testovaniu,
- efektívnosť testovacích činností v súvislosti s cieľmi.

Všeobecné testovacie metriky zahŕňajú:

- percentuálny podiel plánovanej a vykonanej práce v rámci prípravy testovacích dát,
- percentuálny podiel plánovanej a vykonanej práce pri príprave testovacieho prostredia,
- pokrok vo vykonávaní testovacích prípadov,
- informácie o defektoch,
- mieru pokrytia požiadaviek, užívateľských scenárov, akceptačných kritérií, rizík alebo kódu testami,

- mieru dokončenia úloh, alokácia a využitie zdrojov, prácnosť,
- náklady na testovanie vrátane pomeru nákladov a prínosov zistenia ďalšieho defektu alebo pomeru nákladov a prínosov vykonania ďalšieho testu [23].

3.3.7 *Metrika unikátneho návštevníka*

Z hľadiska navrhnú dizajnu softvéru je podstatné spomenúť aj metriku merania unikátneho návštevníka ako jedného z ukazovateľov kvality softvéru pokiaľ sa jedná o webovú aplikáciu (napríklad e-shop).

Nástroje webovej analýzy obvykle spresňujú pojem návštevník (visitor) na unikátneho návštevníka (unique visitor) alebo absolútne unikátneho návštevníka (absolute unique visitor). Nástroje, ktoré sa využíva na toto meranie sú napríklad Google Analytics alebo Piwik.

Spôsob počítania návštevníkov spočíva v tom, že pokiaľ sa pri počítaní unikátnych návštevníkov používa metóda značkovania stránok pomocou JavaScriptu, potom nástroj pre webovú analýzu umiestni do prehliadača návštevníka jedinečný súbor cookie. Tento súbor zostáva uložený, aj keď návštevník stránku opustí. Pokiaľ si návštevník znovu vyžiada sledovanú stránku, je identifikovaný ako vracajúci sa návštevník. Je dôležité si uvedomiť, že metriku unikátny návštevník nemožno spájať s konkrétnou osobou. Pokiaľ si z toho istého prehliadača na tom istom počítači vyžiada sledovanú stránku iná osoba, bude identifikovaná ako vracajúci sa návštevník (bude použitá identifikácia z už existujúceho súboru cookie). V takom prípade nemožno osoby rozoznať.

Sledovanie pomocou súboru cookie so sebou nesie riziko, že návštevníci môžu súbory cookie blokovať a mazať. Návštevníkovi, ktorému je pridelené jednoznačné označenie a zapísané do súboru cookie, je po zrušení súboru cookie pridelené označenie nové a údaj o unikátnom návštevníkovi je skreslený. Už nejde o jedného návštevníka, ale dvoch návštevníkov. Preto je vhodné pozerať na počet unikátnych návštevníkov ako na približné číslo skôr ako absolútne.

Medzi ďalšie metriky, ktoré sa zároveň popri meraní unikátneho návštevníka sledujú zaradzujeme:

- **Návštevy** - Jedná sa o časový interval medzi prvým a posledným požiadavkám označovaní ako relácia (session),

- **Čas strávený na stránke a celkovo na webovej aplikácií** - Jedná sa o meranie času, ktorý návštevníci strávia na jednotlivej stránke a o čas, po ktorý sa pohybujú po celej webovej aplikácií v rámci jednej relácia,
- **Miera opustenia** - Vyjadruje percento návštev, ktoré opustili webovú aplikáciu prehliadnutím jedinej stránky,
- **Miera odchodu** - Udáva percento návštevníkov, ktorí opustili webovú aplikáciu z určitej stránky,
- **Miera záujmu** - Meria kladný vzťah alebo záujem prostredníctvom napríklad dotazníkov umiestnených na stránkach alebo pri odchode z webovej aplikácie,
- **Miera konverzie** - Vyjadrenie v percentách podiel počtu konverzií delene počtom unikátnych návštevníkov. Pod pojmom konverzia môžeme chápať napríklad objednávka na webovej aplikácií.

Tieto metriky slúžia prioritne na komerčné účely, no na základe nich môžeme aj posúdiť, či je softvér kvalitný.

V nadväznosti k týmto metrikám je vhodné spomenúť aj metriku **spokojnosti zákazníka.**

Spokojnosť zákazníkov sa často meria údajmi z prieskumu zákazníkov prostredníctvom päťbodovej stupnice:

- **Veľmi spokojný,**
- **spokojný,**
- **Neutrálne,**
- **Nespokojný,**
- **Veľmi nespokojný.**

Spokojnosť s celkovou kvalitou produktu a jeho konkrétnymi rozmermi sa zvyčajne získava prostredníctvom rôznych metód zákazníckych prieskumov. Na základe údajov päťbodovej stupnice je možné zostaviť a použiť niekoľko metrík s malými odchýlkami v závislosti od účelu analýzy. Napríklad:

- **Percento úplne spokojných zákazníkov,**
- **Percento spokojných zákazníkov,**
- **Percento nespokojných zákazníkov** [13].

3.4 Nástroje slúžiace na meranie kvality softvéru

Nástrojov slúžiacich na meranie kvality softvéru existuje v dnešnej dobe veľké množstvo [35]. Pre našu prácu sme ale vymedzili nástroje - JIRA, Kibana, SonarQube, ktoré sme zvolili na základe konzultácií s manažérmi, a ktorými tvoríme návrh metrík v podnikovej praxi. Dôvod ich analyzovania spočíva v tom, aby sme lepšiu porozumeli tvorbe meraní kvality softvéru a získavania metrík, a aby sme následne mohli v rámci nášho návrhu demonštrovať ich využitie.

3.4.1 *JIRA*

JIRA je softvér od spoločnosti Atlassian určený pre podporu všetkých členských tímov, ktorí sa podieľajú na vývoji softvéru. Poskytuje nástroje na plánovanie a sledovanie úloh, evidenciu problémov a zaznamenávanie chýb pri vývoji softvéru. Z projektového hľadiska je Jira využívaná ako hlavný pracovný nástroj [32]. Jedným z prvkov tohto sofistikovaného nástroja je prehľad (Dashboard). IT manažérom dashboard slúži na zobrazenie si tabuliek a grafov, ktoré zobrazuje výsledky respektíve kritické informácie, z ktorých následne manažéri vyvodzujú rozhodnutia pre riadenie vývoja produktu. Z prehľadu môžeme vyvodiť hneď niekoľko metrík na meranie kvality softvéru z oblasti riadenia agilného vývoja a testovania.

3.4.2 *Kibana*

Kibana je bezplatný, otvorený softvér, ktorý poskytuje používateľské rozhranie. Umožňuje vizualizovať dáta z Elasticsearch, požiadavky na vyhľadávané dáta a efektívne sledovať aplikácie [33]. Pomocou nástroja Kibana môžeme získať niekoľko metrík na meranie kvality softvéru z oblasti integračný, záťažových a automatizovaných testov, ktoré si bližšie popíšeme v nasledujúcich kapitolách a ktoré sme využili pri tvorbe návrhu pre podniková prax.

3.4.3 *SonarQube*

Nástroj slúžiaci na automatickú kontrolu tvorby kódu, ktorý ponúka prehľadné používateľke rozhranie. SonarQube je ľahko konfigurovateľný, čo znamená, že stačí pár kliknutí, aby si manažér vedel spraviť jasný prehľad o jednotlivých metrikách z oblasti programovania. Podporuje jazyky kódu Scala, Java, JavaScript, Python, Ruby, PHP, Apex, JSP, XML, Velocity, VisualForce, C#, JSON a Kotlin a mnoho ďalších [37].

3.5 Návrh metrík pre prax, aby bola dosiahnutá vhodná kvalita softvéru

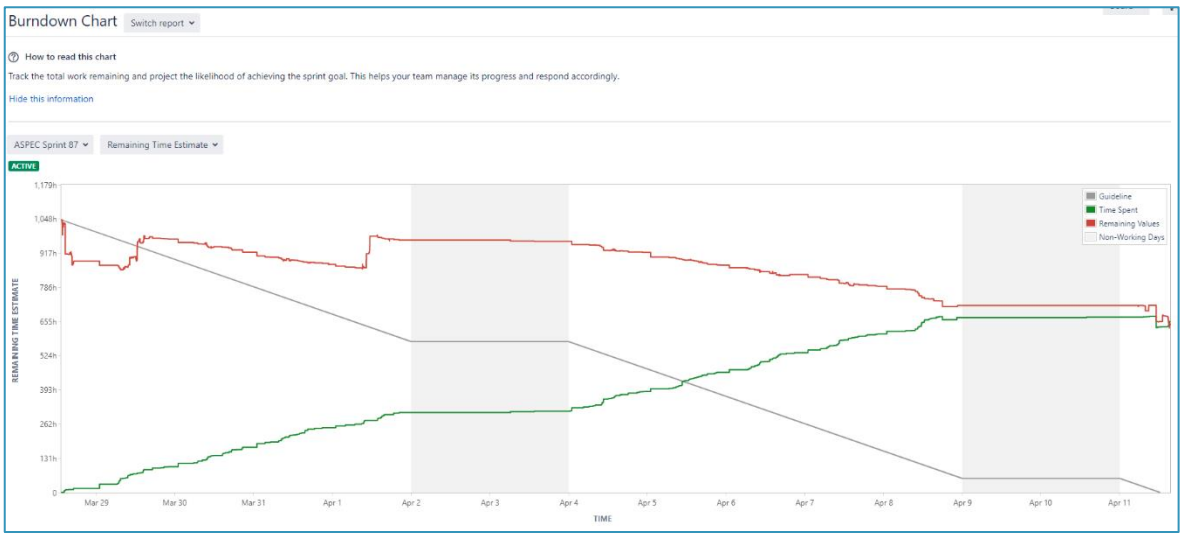
Na základe zmapovaných metrík a konzultácií s manažérmi, ktorí nám poskytli dáta a príklady metrík, ktoré využívajú v podnikovej praxi, sme vybrali metriky, ktoré by mal používať každý agilný tím, aby zabezpečil efektivitu práce a kvalitu dodávaného softvéru.

3.6 Metriky merania kvality softvéru z oblasti riadenia agilného tímu

Metriky merania kvality softvéru z oblasti riadenia agilného tímu využíva hlavne scrum master, aby získal prehľad o efektívnosti jednotlivých sprintov a mohol na základe toho lepšie riadiť tím a zabezpečiť kvalitu dodávaného produktu.

3.6.1 *Iteračný graf zvyškového trendu (Burndown Charts)*

Táto metrika meria koľko užívateľských príbehov (napríklad požiadavka od zákazníka) bolo dokončených v každom sprinte a koľko zostáva do dokončenia funkcionality alebo produktu. Ide o skvelý vizuálny nástroj na sledovanie postupu a kvality dokončenej práce v dlhšom časovom horizonte.



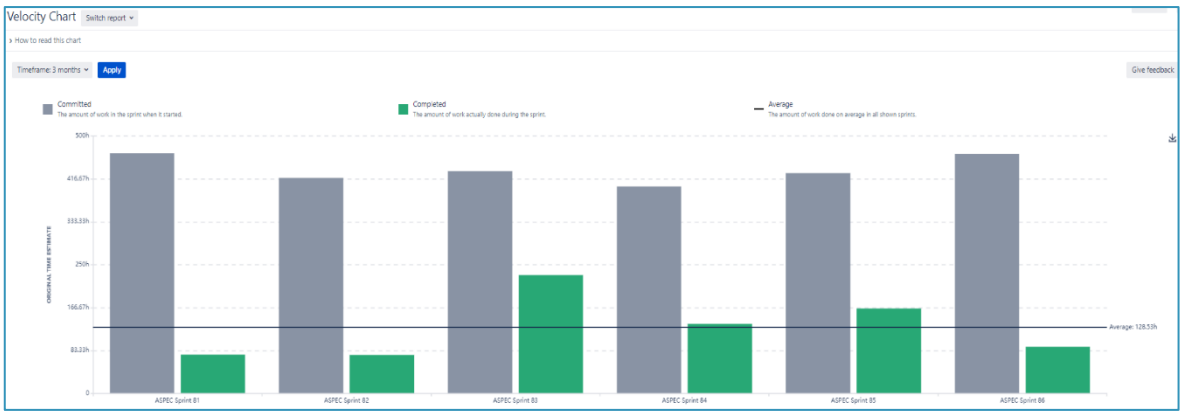
Obrázok 15 Iteračný graf zvyškového trendu [Táto práca]

Na obrázku č. 15 môžeme vidieť porovnanie plánovaného vykonávania (burndown) užívateľských príbehov (story points) v priebehu sprintu, ktoré je zobrazené pomocou sivej krivky a reálneho vykonávania užívateľských príbehov (story pointov) počas sprintu, zobrazené pomocou zelenej krivky. Uvedený príklad Burndown Chart poukazuje červenou krivkou na dva zvýšené nárasty hodnoty „zostávajúci odhad“ (remaining estimate) pre plánované prírastky funkcionality softvérového produktu počas prvej polovice sprintu, čo

zrejme predstavovalo rozšírenie rozsahu práce v sprinte bez odobrania zo sprintového backlogu v adekvátnom rozsahu pre menej prioritné práce. Zároveň graf poukazuje na nedostatočné kapacitné pokrytie rozsahu prác sprintu, čo vidieť zo zelenej krivky, keďže výkaz prác nedosiahol rozsah plánovej pracnosti prírastkovej funkcionality v sprinte. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja Jira. Na základe metriky je manažér schopný predikovať čas dokončenia sprintu a tým zaručiť efektívnosť tímu, ktorá odzrkadľuje kvalitu dodávaného produktu. Pokiaľ je spotrebovaný čas (time spent) vyšší ako plánovaný čas (Guidline), tak všetky stanovené ciele (user stories) budú dokončené v predstihu.

3.6.2 *Priemerná agilná rýchlosť (Average Agile Velocity)*

Metrika meria priemerné množstvo práce, ktorú tím dokončí počas sprintu. Tímy môžu použiť rýchlosť na predpovedanie toho, ako rýchlo dokážu prepracovať nevybavené úlohy, pretože metrika sleduje predpokladanú a dokončenú prácu počas niekoľkých sprintov. Čím viac sprintov, tým presnejšia predpoveď.

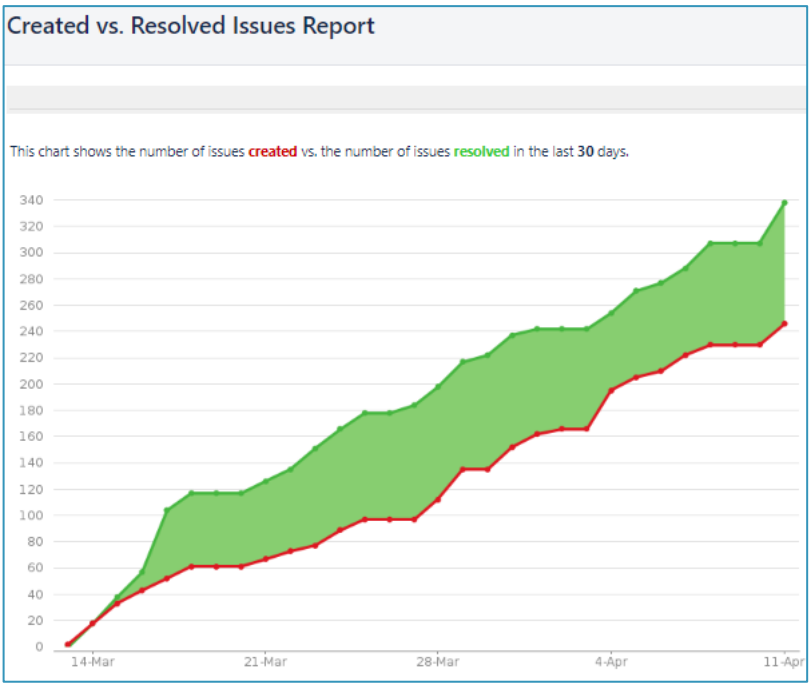


Obrázok 16 Priemerná agilná rýchlosť [Táto práca]

Na obrázku č. 16 môžeme vidieť počet užívateľských príbehov, ktoré boli prijaté do konkrétneho sprintu (sivý stĺpec), a ktoré boli vykonané v rámci sprintu (zelený stĺpec). Následne čiara, ktorá prechádza cez sprinty zobrazuje ich priemernú agilnú rýchlosť. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja Jira.

3.6.3 *Mieria otváranie a zatvárania incidentov (Open/Close rate)*

Metrika sleduje produkčné incidenty, ktoré sa vyskytnú počas daného časového obdobia a ako rýchlo sa vybavia. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja Jira.

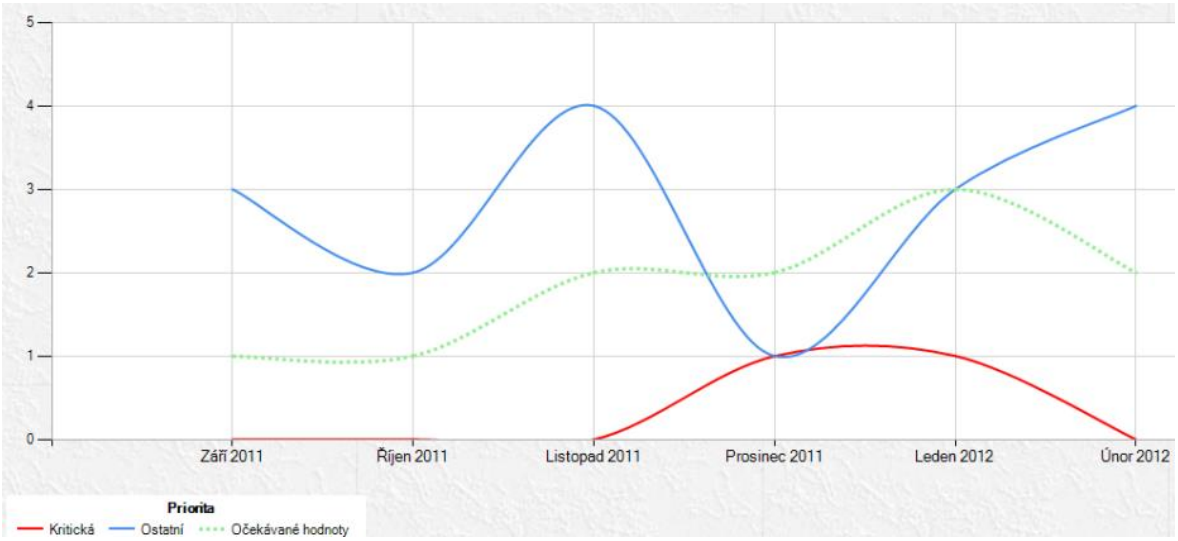


Obrázok 17 Miera otvárania a zatvárania incidentov [Táto práca]

Na uvedenom obrázku č. 17 môžeme vidieť počet vytvorených incidentov (červená krivka) a počet vyriešených incidentov (zelená krivka) za obdobie 30 dní. Sledovaním tejto metriky môže manažér získať prehľad do akej miery je tím vývojárov schopný spracovávať, riešiť a hlavne opravovať chyby v softvéri. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja Jira.

3.6.4 *Metrika vyriešených požiadaviek*

Metrika vyriešených požiadaviek udáva počet požiadaviek, ktoré sme v projekte vyriešili za určité obdobie podľa ich priority. Ide o požiadavky, ktoré sme dostali zadané a už sme ich boli schopní vybaviť. Vysoká miera vyriešených požiadaviek je indikátorom, ktorý vyjadruje, že tím vývojárov pracuje efektívne. Vždy by sme však mali porovnať aktuálny počet aktívnych požiadaviek s tými vyriešenými. Môže totiž nastať situácia, že počet vyriešených požiadaviek bude nulový a môžeme tak nadobudnúť dojem, že tím pracuje neefektívne. Záleží teda na vyššie spomínaných okolnostiach a tiež na fakte, že počas vývojového cyklu softvérového diela sa charakteristika požiadaviek bude meniť.

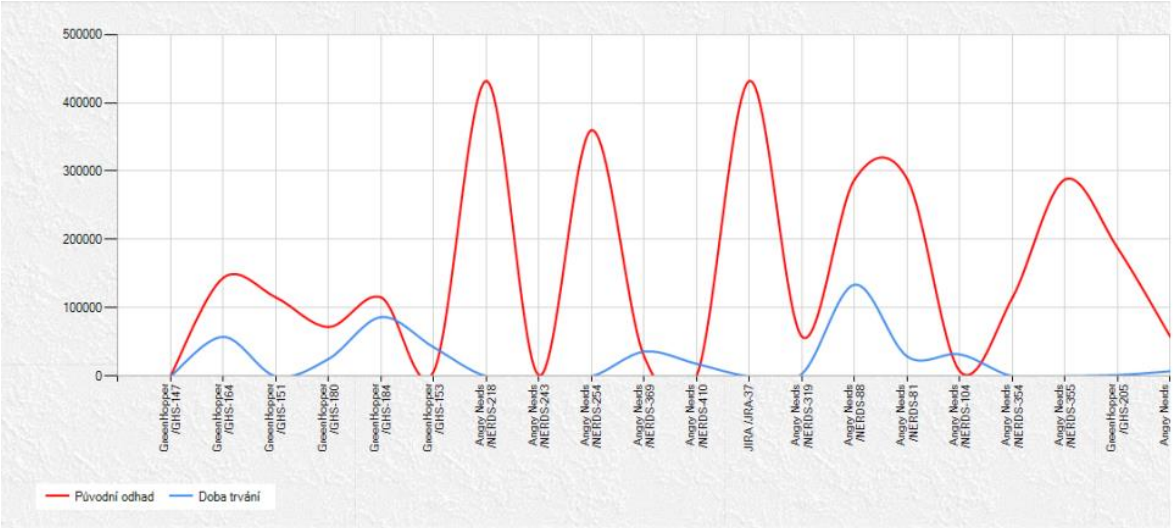


Obrázok 18 Metrika vyriešených požiadaviek [Táto práca]

Na obrázku č. 18 sú zaznamenané kritické (prioritné) a neprioritné požiadavky. Kritickým požiadavkám prislúcha červená farba, pretože majú vyššiu prioritu a sú pre tím dôležitejšie. Do zvyšných požiadaviek (modrá farba) zaraďujeme problémy s triviálnou, malou alebo strednou prioritou. Z grafu je možné vyčítať aj ideálne hodnoty, ktoré reprezentuje zelená prerušovaná krivka. Môžeme tak porovnať aktuálne dáta voči očakávaným. V našom prípade rieši tím vývojárov požiadavky nad očakávania. V decembri sa môže zdať, že bolo očakávanie vyššie ako počet vyriešených požiadaviek. Treba však mať na pamäti, že sa požiadavky delia podľa priority a musia sa brať ako súčet, ktorý je v tomto prípade zhodný s očakávanými hodnotami. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja Jira.

3.6.5 *Odhad plnenia úloh*

Táto základná myšlienka každého procesu merania spočíva v porovnaní doby plnenia úlohy s jeho odhadom. Alebo inými slovami porovnanie reality a ideálneho stavu.

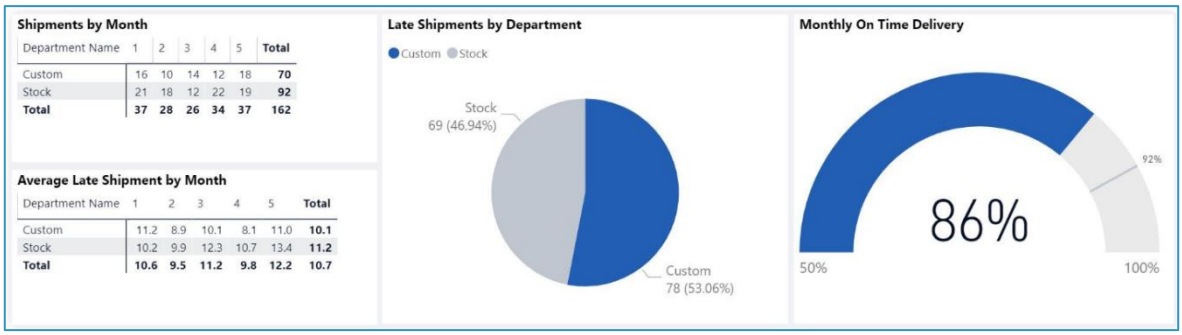


Obrázok 19 Odhad plnenia úloh [Táto práca]

Na obrázku č. 19 zobrazujeme v grafe porovnanie odhadu plnenia úlohy (červená farba) s reálnou dobou (modrá farba) plnenie v určitej iterácii. Z obrázku je možné vyvodiť niekoľko skutočností. Predovšetkým je zvláštne, že pri niektorých úlohách je doba trvania nulová. Možno z toho usudzovať, že plnenie týchto úloh ešte nezačalo. Ale pochopiteľne je potrebné zistiť, či nie je zodpovedná osoba s prácou pozadu alebo ,v horšom prípade, ešte nezačala, či dokonca zabudla. Samozrejme, môže sa stať, že je všetko v najlepšom poriadku a v systéme len nie sú dáta, ktoré zodpovedajú aktuálnemu stavu. Celkovo možno povedať, že v tejto iterácii prebieha plnenie úloh v norme. Po celú dobu je odhad prevažne väčší ako reálna doba trvania. Čo je určite veľmi prínosné. Celkovo by nemalo dôjsť k situácii, kedy bude pôvodný odhad dlhodobo nižší ako doba plnenia. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja Jira.

3.6.6 *Včasné dodanie (delivery on time)*

Včasné dodanie vyjadruje pomer úspešne dodaných funkcionalít v rámci vydaní softvérového produktu voči pôvodnému plánu pre dané vydanie produktu. Jeho prínos spočíva v zobrazovaní úspešnosti dodávať sľúbenú funkčnosť včas.

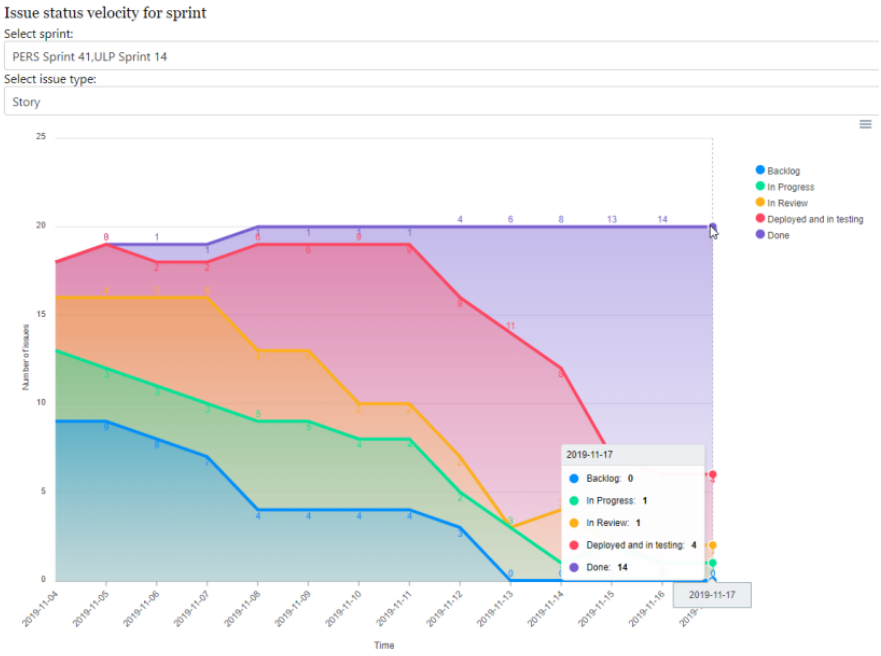


Obrázok 20 Včasné dodanie [Táto práca]

Na druhom grafe na obrázku č. 20 môžeme vidieť, že v rámci meraného času bolo úspešne dodaných 86% funkcionalít v rámci vydania (release) softvérového produktu voči pôvodnému plánu pre dané vydanie (release) produktu. Z obrázku tiež môžeme vyčítať, že tím si stanovil minimálne 92% funkcionalít, ktoré budú odovzdané načas, avšak tento cieľ sa nepodarilo naplniť. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja Jira.

3.6.7 *Stav úloh v čase v aktuálnom sprinte*

Táto metrika je prezentovaná formou skladaného plošného grafu. Merané hodnoty sa vzťahujú vždy k sprintu, a to buď k aktuálnemu alebo historickému podľa výberu. Podľa ďalšej vybranej možnosti ukazuje priebeh úloh. Na obrázku č. 21 vidíme graf, ktorý zachytáva prechod úloh medzi jednotlivými stavmi počas sprintu. Z obrázku je možné vyčítať tri indikátory prezentované za posledný sprint. Prvým je začiatok sprintu. Predpokladom je, že všetky úlohy sú v stave *Backlog*, no z obrázku môžeme vidieť, že na začiatku sprintu, boli štyri úlohy v stave *In Progress*, tri úlohy v stave *In Review* a dve úlohy v stave *Deployed and in testing*. Druhým indikátorom je koniec sprintu, kedy vychádzame z predpokladu, že všetky úlohy sú v stave *Done*. Namiesto toho sú štyri požiadavky v stave *Deployed and in testing* a po jednom v *In Progress* a *In Review* stave. To mohlo byť spôsobené napríklad pomalým procesom revidovania zmeny v kóde, nedostupnosťou testera. Tretím indikátorom je nárast celkového počtu úloh v prvom a štvrtom dni sprintu. Každý sprint by mal mať vopred definovaný a nemenný plán. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja Jira.



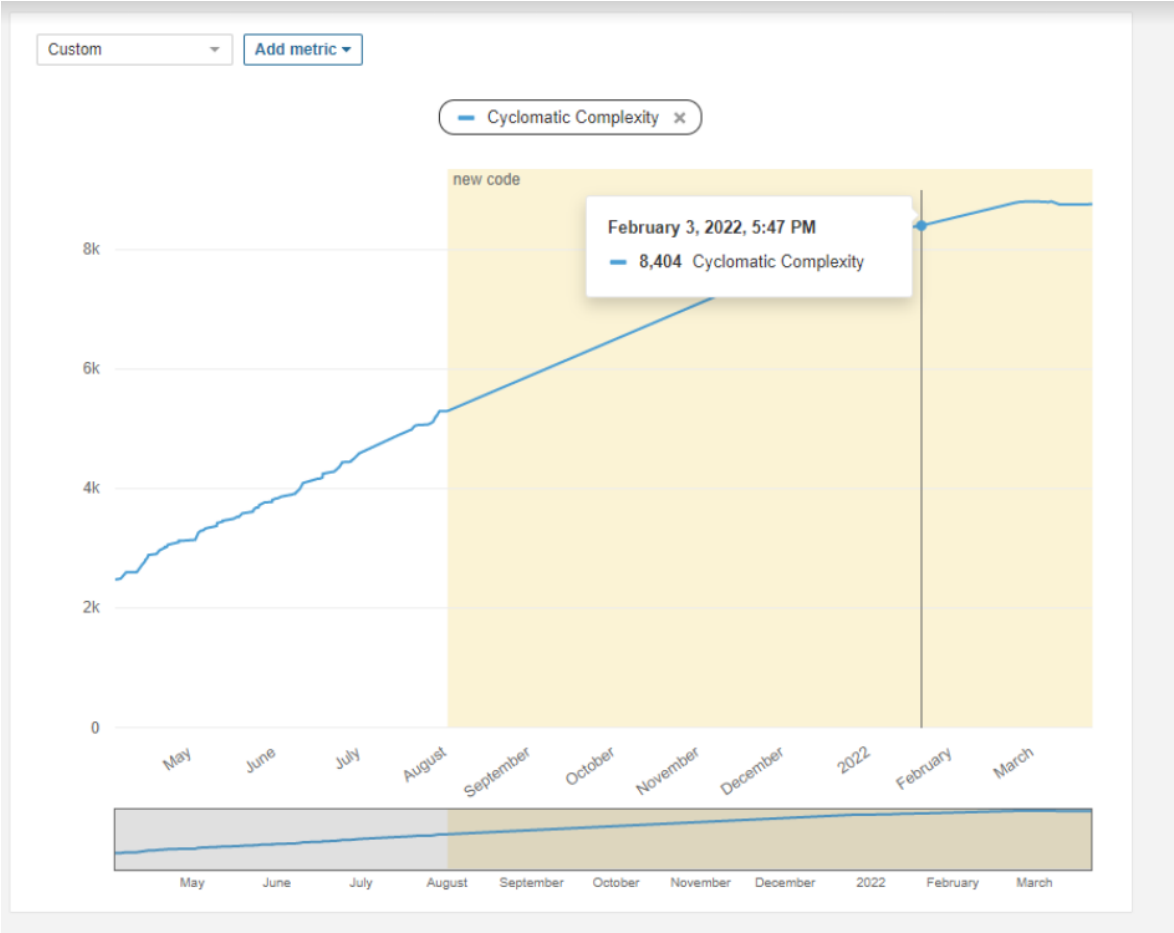
Obrázok 21 Stav úloh v čase v aktuálnom sprinte [Táto práca]

3.7 Metriky merania kvality softvéru z oblasti programovania

Metriky merania kvality softvéru z oblasti programovania sa zameriavajú na efektívnosť programátorov a kvalitu kódu. Pre nás návrh sme vybrali tri metriky, ktoré dobre odzrkadľujú vyššie spomenuté ciele a ktoré v podnikovej praxi využívajú IT manažéri.

3.7.1 *Cyklomatická zložitosť zdrojového kódu*

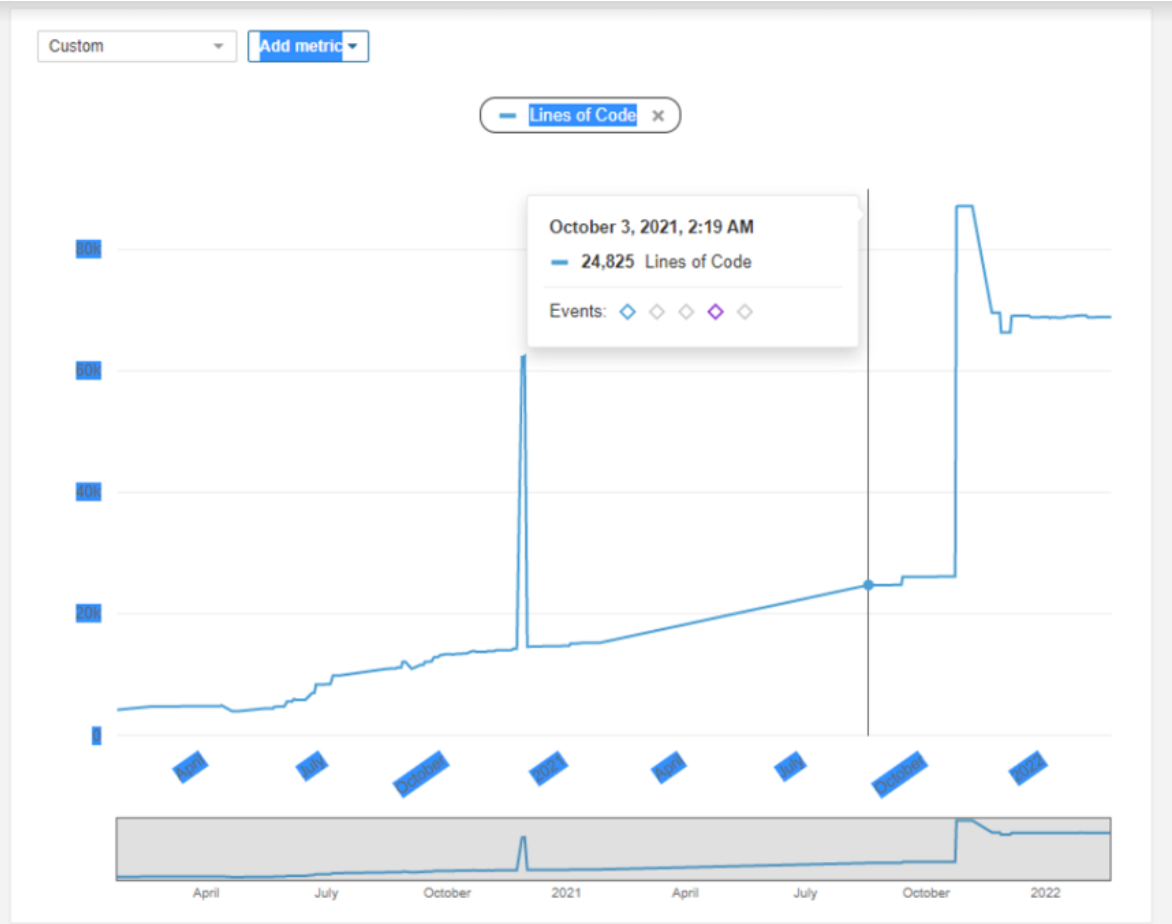
Metrika slúži na označenie zložitosti kódu. Počíta množstvo lineárne nezávislých ciest cez zdrojový kód programu. V praxi to znamená, že ak zdrojový kód neobsahuje príkaz radiaceho toku, jeho cyklomatická zložitosť bude 1 a zdrojový kód obsahuje jednu cestu. Podobne, ak zdrojový kód obsahuje jednu podmienku IF, potom bude cyklomatická zložitosť 2, pretože budú existovať dve cesty, jedna bude pravda (true) a druhá bude nepravda (false). Na obrázku č. 22 nižšie môžeme vidieť ilustráciu tejto metriky, ktorej vizualizácia sa vykonáva prostredníctvom nástroja SonarQube.



Obrázok 22 Cyklomatická zložitosť zdrojového kódu [Táto práca]

3.7.2 *Počet pridaných riadkov kódu (Lines of code)*

Táto metrika predstavuje počet pridaných riadkov kódu v rámci vývoja softvéru. Metriku využívajú manažéri na meranie veľkosti softvéru a zároveň monitorovanie činnosti programátorov.

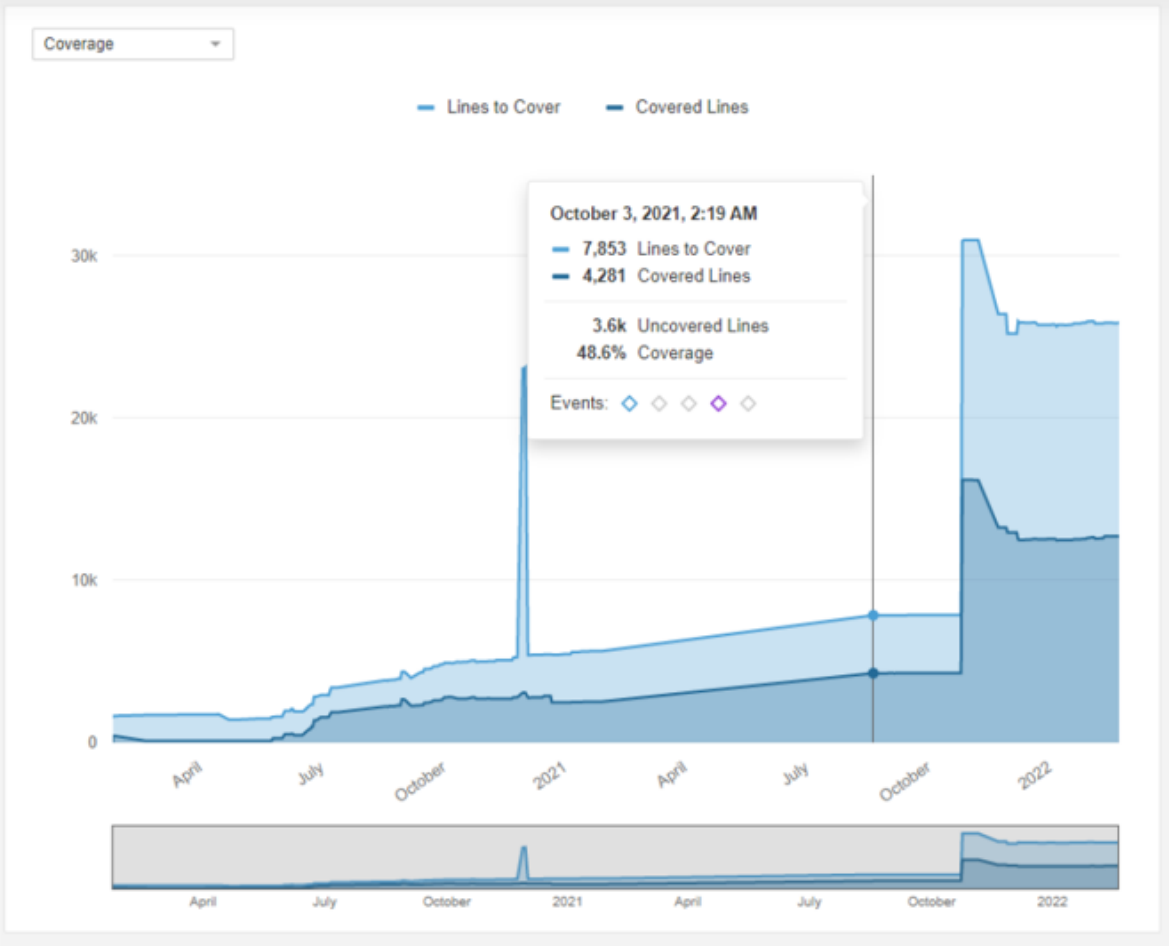


Obrázok 23 Počet pridaných riadkov kódu [Táto práca]

Na obrázku č. 23 môžeme vidieť počet pridaných riadkov kódu za sledované obdobie do systému. Na grafe môžeme napríklad vidieť, že pred koncom roka 2021 nám krivka prudko vzrástla. Dôvod tohto prudkého rastu bol v dôsledku, že sa do systému v jednom čase implementovalo veľké množstvo kódu v rámci vyvíjanej funkcionality, ktorú programátori tvorili mimo systém, a ktorá sa po implementácií do systému stala jeho súčasťou. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja SonarQube.

3.7.3 *Jednotkové (Unit) testy*

Metriku automaticky počítajú podporné nástroje pre správu zdrojových kódov, a to ako percento kódu, ktoré je pokryté jednotkovými (unit) testami. Jeho prínos spočíva v percentuálnej metrike určujúcej kvalitu a udržateľnosť kódu. Hoci nie je definitívnym ukazovateľom kvality kódu, je výhodné s ňou pracovať a snažiť sa ju držať na vysokej úrovni.



Obrázok 24 Jednotkové testy [Táto práca]

Na obrázku č. 24 môžeme vidieť počet riadkov kódu v sledovanom čase, ktoré je potrebné pokryť a počet riadkov kódu, ktoré sa v danom čase pokryté jednotkovými (unit) testami. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja SonarQube.

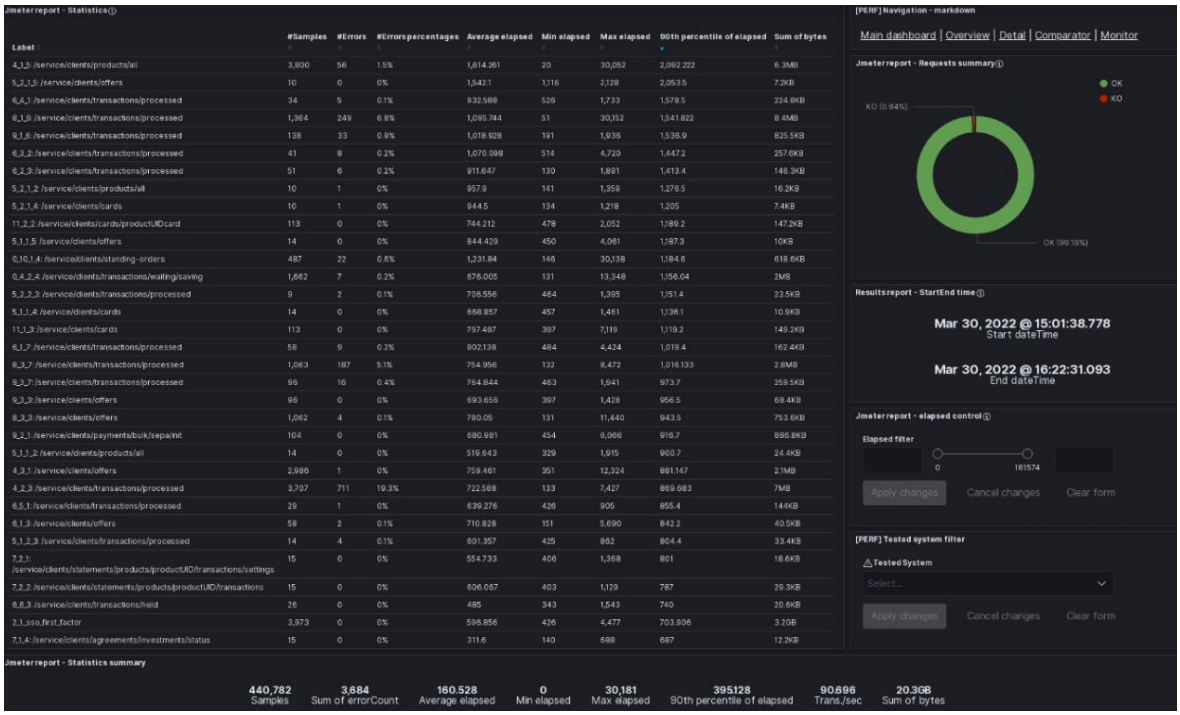
3.8 Metriky merania kvality softvéru z oblasti testovania

Ako sme už spomínali v kapitole 1.5, z nášho pohľadu je testovanie softvéru najväčšia záruka jeho kvality. Pri vývoji softvéru testujeme dodávaný produkt z rôznych hľadísk, aby sme v čo najväčšej miere pokryli všetky funkcionality a mohli identifikovať rôzne druhy defektov, ktoré by sa neskôr mohli dostať do produkcie a tým znížiť dôveru zákazníka v spoľahlivosť softvérovej firmy. Pred vydaním verzie softvéru do produkcie je potrebné, aby test manažéri navrhli plán a realizáciu testovania, následne test analytici vytvorili sadu scenárov, ktoré pokryjú v čo najväčšej miere funkcionalitu softvéru a následne testeri mohli prechádzať vytvorené test scenáre a testovať produkt. Po prejdení všetkých scenárov z rôznych oblastí testovania získavajú test manažéri výsledky, na ktorých

využívajú metriky merania softvéru a tie následne prezentujú vyššiemu manažmentu s cieľom vytvoriť strategické rozhodnutia. Medzi ne patrí napríklad rozhodnutie, či môžeme danú verziu vydať zákazníkovi alebo nastaviť princípy ako vylepšiť tieto výsledky merania v ďalších vydaniach, a tým do väčšej miery zabezpečiť kvalitu softvéru. Metriky merania kvality softvéru z oblasti testovania delíme na meranie výsledkov z oblasti záťažového testovania, integračného testovania, automatizovaného testovania a manuálneho testovania.

3.8.1 *Meranie výsledkov pri záťažovom testovaní softvéru*

Výkonové testy, známe aj ako performačné testy, sú testy používané na meranie času odpovedi a výkonových limitov systému pod generovanou záťažou. Testované sú rozhrania na úrovni aplikácie. Na generovanie záťaže a meranie výsledkov je používaný nástroj Kibana. Vyhodnotenie výsledkov vykonáva IT analytik testov. Vyhodnocované sú výsledky časov odpovedí testovanej aplikácie, chybovosť testu a celkový priebeh testu v HTML reporte. Výsledky sú zapísané v prípade potreby v Jire a z každého testu je vytvorený manuálne report v Confluence (doplnok nástroja Jira, ktorý slúži na ukladanie dokumentov). Výsledky testov sú pravidelne reportované projektovému test manažérovi. Na vizualizáciu výsledkov sa používa nástroj Kibana. Pri takomto type testovania navrhujeme napríklad metriku stavu úspešných, neúspešných testov záťaže.



Obrázok 25 Metrika stavu úspešných a neúspešných testov záťaže [Táto práca]

Na obrázku č. 25 môžeme vidieť zoznam služieb, kde prvý stĺpec obsahuje počet volaní daných služieb, následne počet chýb, percentuálne vyjadrenie celkového počtu chýb zaznamenaných v rámci služby, priemerný čas volaní v milisekundách, minimálny čas v milisekundách, maximálny čas v milisekundách a percentil, ktorý udáva, koľko zhruba bude trvať beh služieb na produkcií. Graf, ktorý sa nachádza na pravej strane znázorňuje percentuálnu úspešnosť a neúspešnosť vykonaných testov záťaže z poskytnutých dát za sledované obdobie a slúži na vizuálne zobrazenie navrhovanej metriky.

Obrázok 26 Metrika úspešných a neúspešných testov záťaže v časových rozmedzeniach [Táto práca]

V rámci poskytnutých dát, môžeme metriku doplniť aj o informácie o počte úspešných a neúspešných volaní v daných časových rozmedziach, kde na obrázku č. 26 v prvom grafe vidíme počet úspešných a neúspešných volaní v daných rozmedziach (0-500 milisekúnd, 500-1000 milisekúnd atď.). A na druhom grafe môžeme vidieť už iba chybové volania, ktoré boli v daných rozmedziach. Tieto grafy pre manažéra môžu indikovať, že väčšina chýb nebola spôsobená výkonnosťou (rýchlosťou) aplikácie, ale funkčnosťou, keďže väčšina bola zachytená v rozmedziach do 1000 milisekúnd.

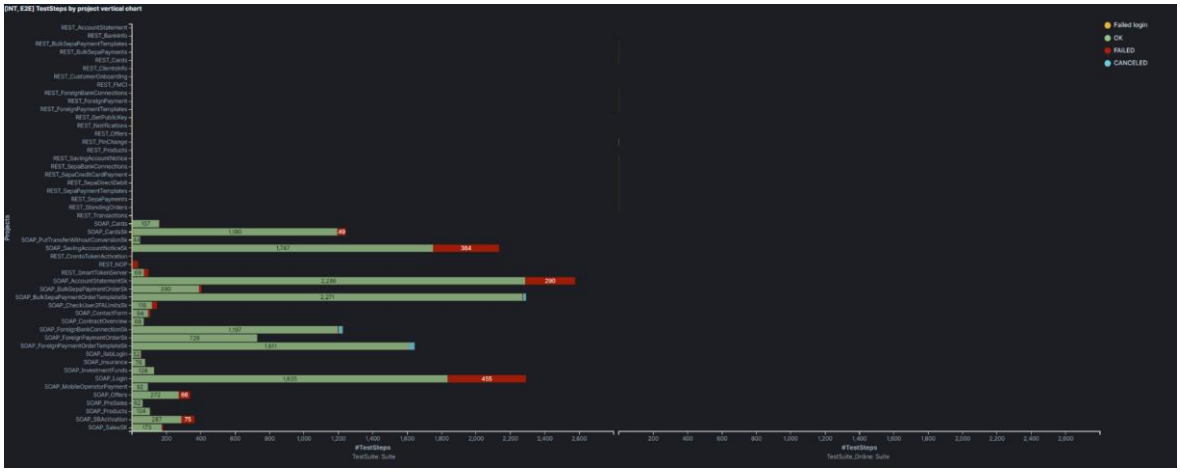
3.8.2 *Meranie výsledkov pri integračnom testovaní softvéru*

Testované sú aplikačné webové rozhranie a vystavené webové služby. Nástroj na testovanie a opakované volanie služieb je SoapUI. Vyhodnocované sú odpovede systémov za konkrétne požiadavky, určené podľa navrhnutých testovacích scenárov. Testovacie scenáre ďalej rozlišujeme na pozitívne a negatívne, podľa toho čo testujeme, respektíve akú očakávame odpoveď. Vyhodnocované sú detailné odpovede aplikácií a všeobecné výsledky testovacích scenárov. Celkové vyhodnotenie výsledkov vykonáva IT analytik testov. Výsledky sú zapísané manuálne do Jira a je vytvorený detailný report na Confluence. Výsledky testov sú pravidelné reportované projektovému test manažérovi. Na vizualizáciu

výsledkov sa používa nástroj Kibana. Pri takomto type testovania navrhujeme metriku stavu úspešných, neúspešných a nevykonaných volaní za systém.

Obrázok 27 Metrika úspešných, neúspešných a nevykonaných volaní za systém [Táto práca]

Na obrázku č. 27 môžeme vidieť percentuálne výsledky stavu volaní za systém (steps status by system) v rámci testovacieho scenára. Zelená časť nám hovorí o úspešných výsledkoch volania. Červená časť nám hovorí o neúspešných výsledkoch volania a modrá časť o výsledkoch nevykonaných volaní za obdobie 5 dní. Toto meranie bolo vykonané v bankových systémoch mojacsob, public, smartbanking.



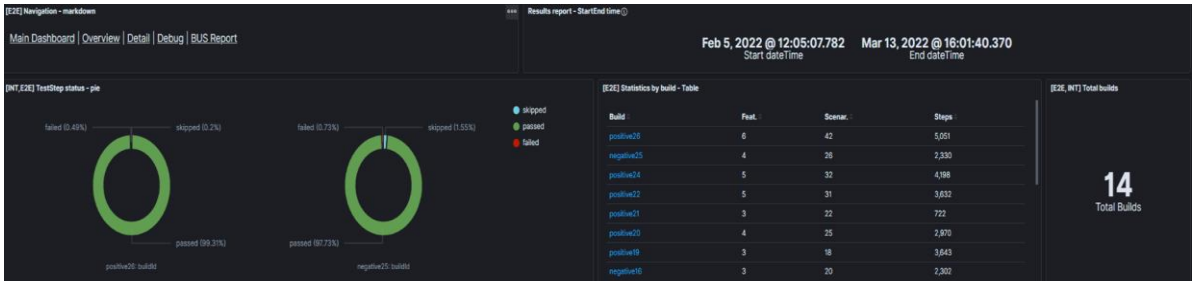
Obrázok 28 Počet úspešných, neúspešných a nevykonaných volaní za jednotlivé projekty

Naopak obrázok č. 28 nám hovorí o počte úspešných, neúspešných a nevykonaných volaní za jednotlivé projekty (Test steps by project vertical chart) za obdobie 5 dní.

3.8.3 *Meranie výsledkov pri automatizovanom testovaní softvéru*

Na automatizované testovanie sú použité frameworky Selenium, zapísané v jazyku Java, ktoré využívajú Selenium WebDriver API. Ich výsledky sú zapísané automatizované do nástroja Jira a je vytvorený detailný report na Confluence. Výsledky testov sú pravidelné

reportované projektovému test manažérovi. Jednou z dôležitých metrík pre meranie výsledkov pri automatizovanom testovaní softvéru slúži metrika počtu úspešných a neúspešných automatizovaných testov v čase, ktorými sa pokrýva nasadená funkcionalita.



Obrázok 29 Počet úspešných a neúspešných automatizovaných testov

Na obrázku č. 29 môžeme vidieť sumár výsledkov 14 rôznych spustených zhlukov (buildov) za jeden a pol mesiaca, ktoré obsahujú testovacie kroky, ktorými bola pokrytá funkcionalita. Grafy reprezentujú percentuálnu úspešnosť, neúspešnosť a vynechanie vykonaných testovacích krokov za merané obdobie. Na základe získaných výsledkov môžeme usúdiť, aké percento testov prešlo úspešne v rámci sledovaného času a aké percento neprešlo. Veľká chybovosť môže byť dôsledkom toho, že testy boli vykonané prostredníctvom Selenium, pričom testy boli neúspešné z dôvodu dlhého načítania stránky a testovacích krokov, a to tento framework vyhodnotil ako neúspešný test. Na vizualizáciu výsledkov sa používa nástroj Kibana.

Obrázok 30 Počet úspešných a neúspešných automatizovaných testov v rámci projektu [Táto práca]

Na obrázku č. 30 môžeme naopak vidieť počet úspešných a neúspešných automatizovaných testov v rámci projektov (features), testovacích scenárov (scenarios) a testovacích krokov. Metriku využívame na uistenie, že regresné testy sú spúšťané každú iteráciu a že nedošlo ku zavedenie chyby do už existujúcej funkcionality. V agilnom

prostredí je automatizácia hlavným nástrojom na zabezpečenie dodávky fungujúceho produktu. Metriky spojené s pokrytím automatizovanými testami by nemali byť prehliadané.

3.8.4 *Meranie výsledkov pri manuálnom testovaní softvéru*

Metriky slúžiaci pre meranie výsledkov pri manuálnom testovaní softvéru sledujú hlavne test manažéri. V rámci testovania softvéru manuálnymi testami získavame výsledky, ktoré môžeme merať rôznymi pohľadmi a preto pre túto oblasť navrhujeme najvyšší počet metrík. Vizualizácia týchto metrík je vykonávaná prostredníctvom nástroja Jira.

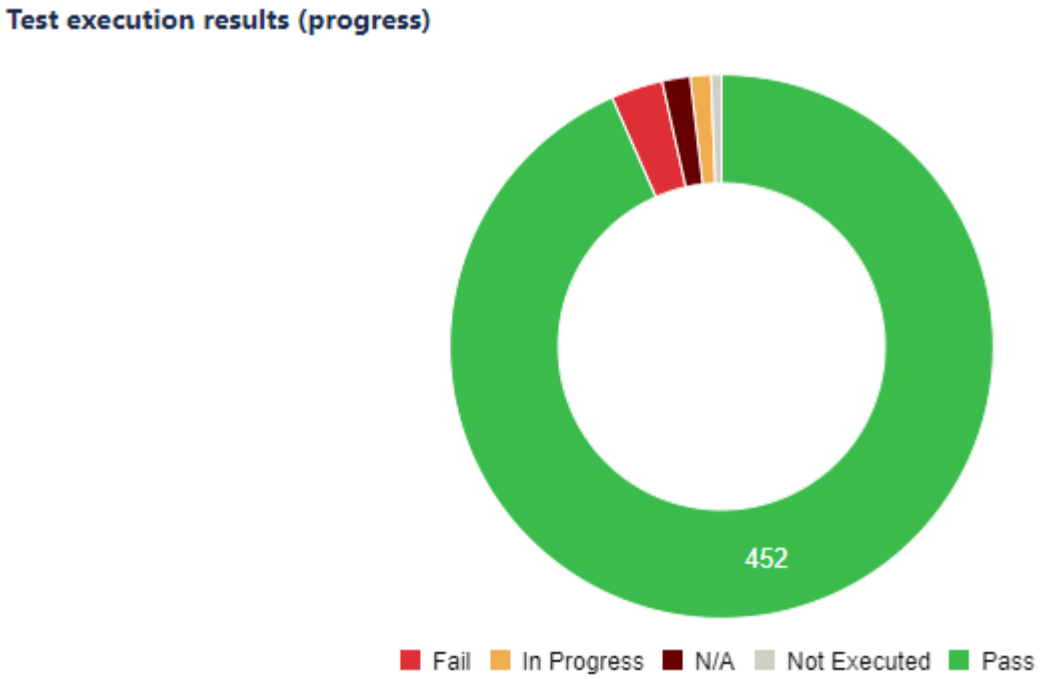
Pre nasledujúce metriky navrhované v našej práci budeme vychádzať z poskytnutých dát na obrázku č. 31.

Test execution scorecard by test cycle

	Test Execution Results										Effort				Issues		
Test Cycle	Total	Not Executed	In Progress	Pass	Fail	Blocked	N/A	Completed	Remaining	Progress	Estimated	Actual	Remaining	Variation	Open	Closed	Total
Názov test cyklu	484	0.62% (3)	1.24% (6)	93.39% (452)	3.1% (15)	0% (0)	1.65% (8)	475	9	98.14%	62:00	19:16	42:43	68.91%	0	15	15
Total	484	0.62% (3)	1.24% (6)	93.39% (452)	3.1% (15)	0% (0)	1.65% (8)	475	9	98.14%	62:00	19:16	42:43	68.91%	0	15	15

Obrázok 31 Zaznamenaná výsledky manuálneho testovania v rámci testovacieho cyklu [Táto práca]

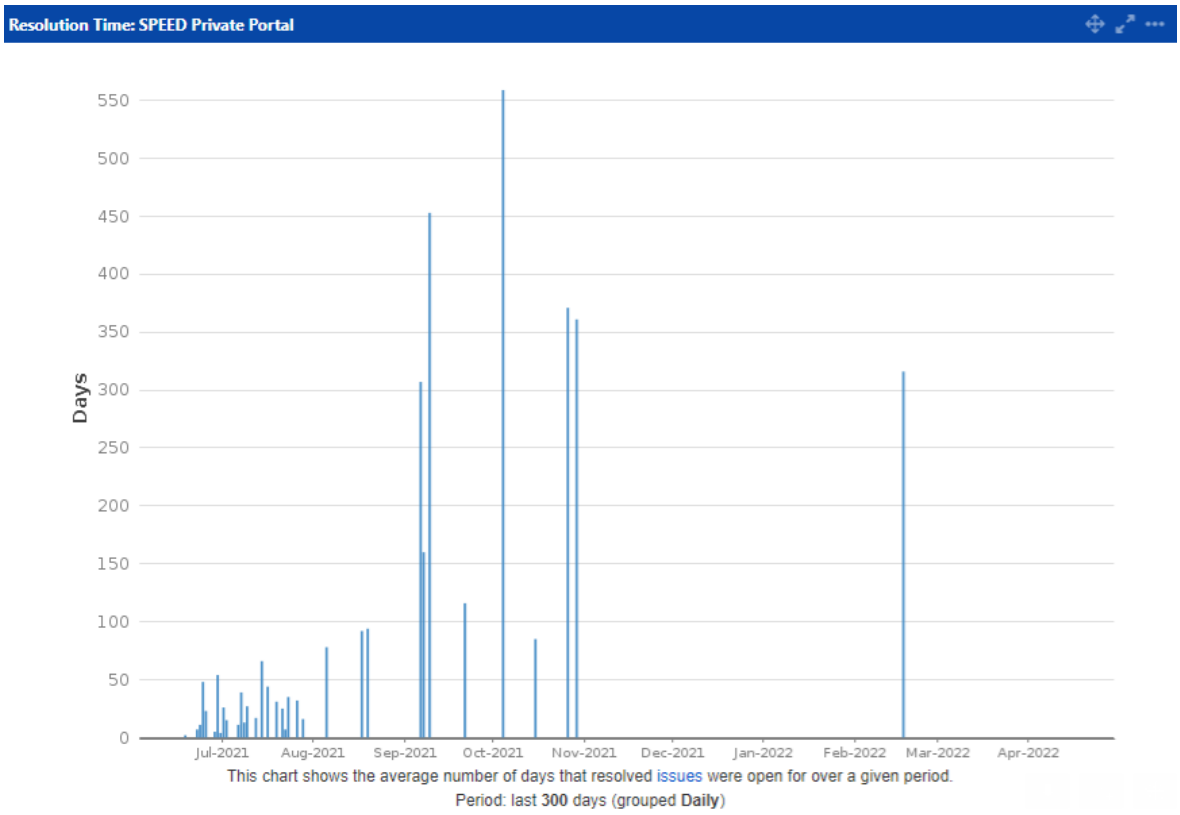
Obrázok nám hovorí o zaznamenaných výsledkov manuálneho testovania v rámci jedného testovacieho cyklu. Na základe vydania (release-u) boli pripravené testovacie prípady pre pokrytie celej novej funkcionality. Regresnou sadou testovacích prípadov zahrnutou v testovacom cykle sa zároveň potvrdila bezdopadovosť na funkcionality nasedené v produkcii. Na základe 93,5% pass rate-u môžeme prehlásiť, že bola splnená požadovaná kvalita (quality gate) pre dané vydanie (release). Neprítomnosť kritických defektov zabezpečuje, že môžeme túto dodávku z pohľadu testovania akceptovať bez výhrad. Testovacie prípady v stave N/A (not executed) sa týkali funkcionalít, ktoré nebolo možné na prostredí v čase testovania otestovať. Pre tieto výsledky môžeme použiť metriku úspešných, neúspešných vykonaných testovacích scenárov a metriku počtu nespustených testovacích scenárov, ktorej vizualizáciu môžeme vidieť na obrázku č. 32.



Obrázok 32 Metrika počtu úspešných, neúspešných a nespustených testovacích scenárov

Následne si uvedieme ďalšie metriky v rámci nášho návrhu, ktoré už nesúvisí s dátami prezentovanými vyššie, ale týkajú sa oblasti manuálneho testovania. **Metrika pokrytia funkcionalít regresnými testami/testami.** Dôležitosť tejto metriky spočíva vo vzťahu k formovateľnosti požiadaviek a testov. Každá požiadavka by mala byť viazaná minimálne na jeden test a zároveň by nemal existovať test, ktorý nie je priradený k požiadavke. Nízke pokrytie požiadaviek testami by malo byť pre tím alarmujúce. V metodike Scrum sa často používa pojem HLAC – High level acceptance criteria. Ide o súbor kritérií, ktoré musia byť minimálne uspokojené, aby bola požiadavka považovaná za dokončenú. V nadväznosti na metriku pokrytia požiadaviek testami by malo byť každé z týchto kritérií pokryté minimálne jedným testom.

Metrika merania času opravy defektu v kóde od zaevidovania po opravu nám hovorí o absolútnom počte dní od založenia nového defektu po jeho označení za vyriešený (či je opravený alebo zneplatnený). Vývoj softvéru nie je len o nových funkcionalitách, ale aj zachovaní existujúcich funkcionalít bez defektov, teda o dodržaní istej úrovne kvality produktu. Je teda podstatné udržiavať prehľad o tom, či (a ako rýchlo) sa tímu darí defekty opravovať. Táto metrika slúži ako indikátor, či tím má dosť priestoru venovať sa oprave defektov. Graf reprezentujúci túto metriku môžeme vidieť na obrázku č. 33. Pre jeho vizualizáciu je využívaný nástroj Jira.



Obrázok 33 Metrika merania času opravy defektu v kóde od zaevidovania po opravu [Táto práca]

Vzhľadom na fakt, že rýchlosť dodania je v agilnom prostredí kľúčová, je potrebné zisťovať, ktoré faktory môžu rýchlosť ovplyvniť. Začiatok merania je moment, kedy došlo k priradeniu defektu vývojárovi. Býva zvykom, že pre každú prioritnú triedu defektu je nastavená maximálna doba pre jeho opravu. Napríklad pri nájdení defektu s najvyššou prioritou je doba na jeho opravu maximálne 1 deň. Pokiaľ dosahuje tím príliš vysoké hodnoty tejto metriky, je nutné pátrať, čo tento stav zapríčinil.

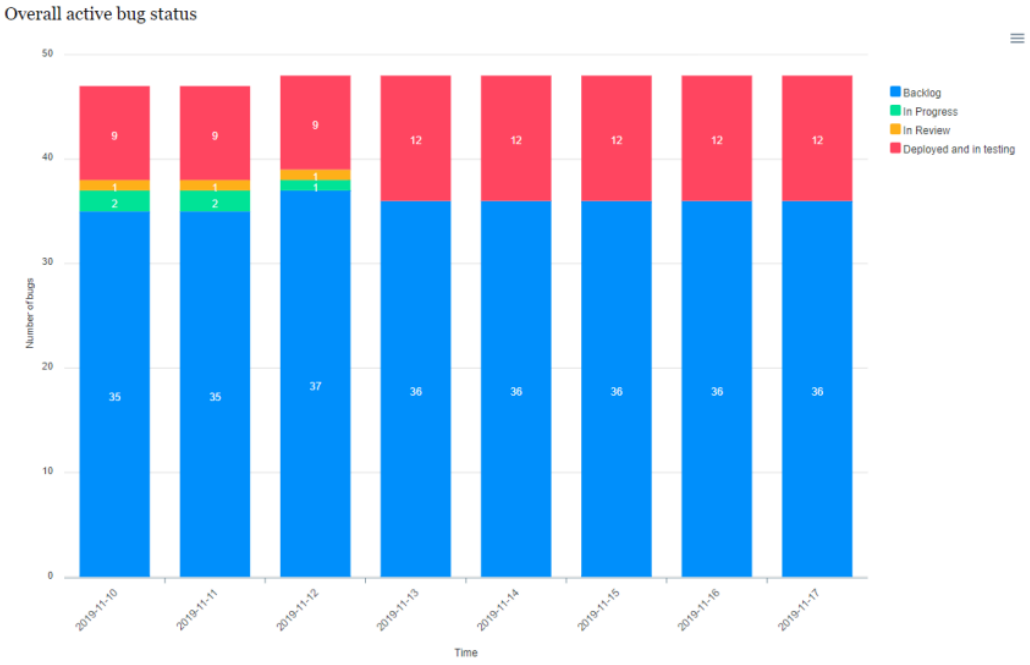
Metrika dostupnosti testovacieho prostredia. Nedostupnosť testovacieho prostredia nás môže dokonale obmedziť pri akejkoľvek snahe testovať. Metrika nadobúda na význame vo chvíli, keď sa nestabilita testovacieho prostredia začína odrážať na rýchlosti dodávania produktu. Väčšinou táto nepríjemnosť nie je zvažovaná pri odhadovaní prácnosti testovania funkcionality a nestála situácia môže zablokovať testovací proces až na niekoľko hodín. Ak táto metrika nadobúda vyššie hodnoty, je potrebné venovať dostatočnú kapacitu na prešetrenie príčiny nestability prostredia.

Metrika času potrebného k testovaniu. Táto metrika je kľúčová pre vytváranie odhadov trvania testovania. Odhad času potrebného na testovanie zahŕňa prípravu na

testovanie, exekúciu testovania a reportovanie testovania. V prostredí agilného vývoja je tento odhad zahrnutý v rámci odhadovania prácnosti požiadaviek, keďže dokončené testovanie je súčasťou definície dokončeného produktu. Pri sledovaní skutočného času potrebného na testovanie, je možné následne porovnať odhadovaný a skutočný čas potrebný na testovanie funkcionality,

Metrika počtu nevyriešených defektov v rámci sprintu, ktoré sú v produktovom backlogu v stave - v procese (In progress), na revíziu /(In review), nasadené a testované (Deployed and in testing).

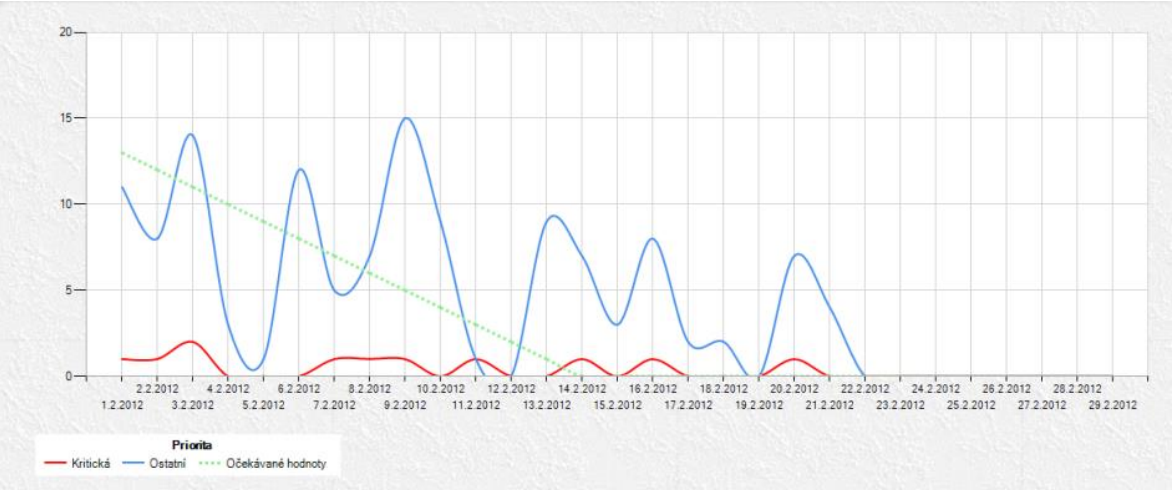
Táto metrika je prezentovaná formou skladaného stĺpcového grafu na obrázku č. 34. Na horizontálnej osi je nanesený čas, ktorého jednotkou je deň. Na vertikálnej osi je zobrazený počet defektov. Skladaný stĺpec predstavuje rozdielne stavy defektov. Zaujímajú nás iba nevyriešené defekty, takže stav hotové (Done) sa v tejto metrike nenachádza. Vďaka zachyteniu defektov v čase je možné zistiť celkový počet defektov. Na grafe reprezentujúcom túto metriku, znázornenom na obrázku č. 34, je jasne vidieť, že na projekte je k 17.9.2021 36 defektov bez začatej práce a 12 sa nachádza v stave testovanie. Zároveň je zrejmé, že stav defektov sa nezmenil od 13.11.2021. To môže byť spôsobené napríklad nedostatočnou kapacitou tímu na opravu defektov a ich pretestovanie.



Obrázok 34 Metrika počtu nevyriešených defektov v rámci sprintu [Táto práca]

Metrika chýb podľa priority. Zrejme najzávažnejšie sú v každom systéme chyby. Ich správna evidencia, vyhodnotenie a predovšetkým riešenie je jedným zo základov úspešného fungovania spoločnosti. Existuje celý rad chýb. Najpodstatnejšia je závažnosť danej chyby vzhľadom k správnej funkcii systému. Napríklad v internet bankingu môže byť ako tá najzávažnejšia chyba napríklad nemožnosť sa do systému vôbec prihlásiť. Zdôrazňovať že táto chyba musí byť odstránená bez zbytočných odkladov, je zbytočné. Pre väčšinu týchto a ním podobných problémov sa navyše vzťahuje záruka na fungovanie softvérového produktu. Tá udáva dodávateľovi povinnosť riešiť chyby podobného rázu bezodkladne. Iná situácia pochopiteľne nastáva, pokiaľ užívateľ takto závažnú chybu spôsobil svojim nezodpovedných prístupom.

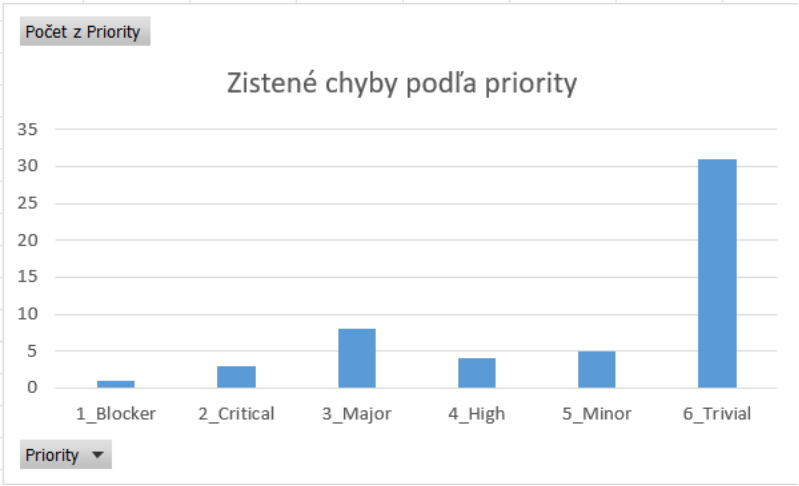
V agilnom prístupe sa množstvo defektov mení s každou iteráciou. Zástupca zákazníka, alebo iná zodpovedná rola biznisu, stanovuje prioritu pre riešenie defektu vzhľadom na závažnosť a vplyvu na dodávanú funkcionalitu. Kritické defekty sú uprednostnené a v optimálnom prípade opravené pred ukončením iterácie, alebo pred vydaním novej verzie produktu do produkčného prostredia. Prítomnosť kritických defektov je nežiaduca a v každom okamihu prebiehajúceho cyklu by sme mali mať prehľad o počte týchto defektov a ako sa vyvíjajú v čase. Nevyriešené defekty sa stávajú súčasťou produktového backlogu a riešia sa v budúcich iteráciách. Avšak nezaujímajú nás iba hlásenia o defektoch, ktoré kriticky ovplyvňujú kvalitu produktu, ale aj počet tých s nižšou prioritou. Narastajúci počet odkladaných „nedôležitých“ defektov môže podstatne ovplyvniť pohľad zákazníka na kvalitu produktu a jeho použiteľnosť.



Obrázok 35 Metrika chýb podľa priority [Táto práca]

Na uvedenom obrázku č. 35 vidíme graf chýb v projekte v určitom sprinte. Výraznejšia farba (červená) predstavuje chyby s kritickou prioritou, modrá ostatné menej závažné chyby. Malo by tu platiť, že počet červených nepresiahne tie modré. Ak nastane stav, že kritické chyby presiahnu nekritické, môže to znamenať, napríklad kompletne zlú základnú analýzu, ktorá spôsobí, že v systéme vznikajú stále nové chyby a nie je možné ich opraviť v dostatočne krátkom čase. Ďalším dôležitým ukazovateľom v grafe je očakávaná hodnota (zelená). Tá predstavuje akýsi ideálny stav. Ten môže vychádzať ako z histórie, ako napríklad z iných projektu, tak aj z jednoduchého odhadu. Pre konkrétne projekty a sprinty ho je možné, samozrejme, meniť podľa aktuálnej potreby. Vizualizácia tejto metriky sa vykonáva prostredníctvom nástroja Jira.

Na vizualizáciu metriky chýb podľa priority môžeme využiť aj graf vytvorený v nástroji Excel, kde vidíme rôzne defekty počas sprintu s ich prioritou.



Obrázok 36 Metrika chýb podľa priority vizualizovaná v Exceli [Táto práca]

Záver

Cieľom diplomovej práce bolo zmapovanie rôznych prístupov merania kvality softvéru a informačných systémov v podnikovej praxi z rôznych hľadísk s aspektom na vývoj softvéru. Na dosiahnutie tohto cieľa sme si stanovili niekoľko čiastkových cieľov. V teoretickej časti sme sa zamerali na porozumenie toho, čo znamená kvalita softvéru. Následne sme vykonali prieskum certifikátov, ktoré opisujú riadenie kvality softvéru a charakteristík, ku ktorým firma musí pristúpiť a dodržať ich, aby certifikát získala. Ďalším čiastkovým cieľom sme sa zamerali na analýzu techník zaistenia a riadenia kvality softvéru a súčasných metodík vývoja. Porovnaním metodík vývoja softvéru sme zistili. že metodika Scrum je v súčasnej dobe najvyužívanejšia v podnikovej praxi,

V praktickej časti na základe analýzy a syntézy poznatkov opisujeme jednotlivé zmapované metriky z rôznych hľadísk, respektíve oblastí v rámci vývoja softvéru. Tieto metriky následne konzultujeme s viacerými manažérmi z rôznych firiem v podnikovej praxi. Na základe konzultácií a získaných dát z jednotlivých firiem respektíve oblasti v rámci vývoju softvéru navrhujeme metriky, ktoré by podľa nás mala využívať každá firma na dosiahnutie kvality softvéru. Môžeme konštatovať. že všetky ciele, ktoré sme si stanovili v práci, sa nám podarilo splniť.

Na základe porovnania uvedených a zmapovaných metrík v rámci nášho návrhu, môžeme konštatovať, že aj keď sú si tieto metriky veľmi podobné, tak v podnikovej praxi sú upravené a prispôsobené zámerom každého manažéra. Manažéri sú si vedomí všeobecných metrík, ktoré pristupujú ku kvalite softvéru, ale väčšinu z nich si ich vyberajú na základe svojich preferencií, meraných dát, či nástrojov, ktoré poskytujú vizualizáciu jednotlivej metriky. Tieto metriky následne potom prezentujú vyššiemu manažmentu alebo zákazníkovi s cieľom poukázať na efektivitu tímu či samotnú kvalitu softvéru.

Taktiež môžeme tvrdiť, že najviac metrík bolo zaznamenaných v oblasti riadenia agilného tímu, pretože odzrkadľujú kvalitu, respektíve efektivitu riadenia tímu. Následne bolo uvedených veľa metrík z oblasti testovania, a to z dôvodu, že práve testovanie najlepšie odzrkadľuje kvalitu produktu.

Prínosom diplomovej práce je detailné zmapovanie a následne poskytnutie návrhu metrík, ktoré by každá vývojová firma či manažér mali využiť v agilnom vývoji. Dôležitou

súčasťou sú taktiež certifikáty, ktorými firma indikuje pre zákazníka, že správne pristupuje ku kvalite softvéru.

Zoznam použitej literatúry

[1] **ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ.** Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013, 208 s. ISBN 978-80-251-3816-8.

[2] Kvalita (jakost). In: ManagementMania [online]. Wilmington (DE) 2011- 2017, 06. 03. 2017 [cit. 2022-02-16]. Dostupné na: https://managementmania.com/cs/kvalita-jakost

[3] **KVETKO, Martin.** Tématický okruh SI11. In: Vrstevnice [online]. [cit. 2021-12-13]. Dostupné na: http://vrstevnice.com/akce/grandaction/vskola/7statnice/otazky/otazkaSW11_36SI2.pdf

[4] **GALIN, Daniel.** Software quality assurance. New York: Pearson Education Limited, 2004, 617 s. ISBN 0-201-70945-7.

[5] **H. ALKHATEEB**, **Jawad**. Quality model based on cots quality attributes. In: Research Gate [online]. 2013 [cit. 2021-12-13]. Dostupné na: https://researchgate.net/figure/260391390_fig2_Figure-2-McCall-SoftwareQuality-Model

[6] ISO/IEC 25010. In: iso25000 [online]. 2017 [cit. 2021-12-13]. Dostupné na: http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[7] **POORNIMA M. CHARANTIMATH**. Total quality management. New Delhi: Pearson Education, 2006, 256 s. ISBN 9788177586473.

[8] **Frederic P. Miller, Agnes F. Vandome, John McBrewster.** PDCA. Alphascript Publishing, 2011, 88 s. ISBN 6134273619.

[9] **PATTON, R.** Testování softwaru. Preklad David KRÁSENSKÝ. Praha: Computer Press, 2002, 313 s. ISBN 80-7226-636-5.

[10] SDLC - V-Model. In: Tutorials point: Simply easy learning [online]. 2017 [cit. 2021-12-14]. Dostupné na: https://tutorialspoint.com/sdlc/sdlc_v_model.htm

[11] **HLAVA, Tomáš**. Fáze a úrovně provádění testů. Testování softwaru. In: testovanisoftwaru [online]. 2011 [cit. 2021-12-28]. Dostupné na: http://testovanisoftwaru.cz/metodikatestovani/druhy-typy-a-kategorie-testu/faze-testu/

[12] Funkční vs nefunkční testování. In: Testování software [online]. [cit. 2021-12-28]. Dostupné na: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=22:funkni-vs-nefunkni-testovani&catid=3:zaklady&Itemid=11

[13] Základní metriky. In: Webová analytika: i pro neprofesionály [online]. 2014 [cit. 2021-12-14]. Dostupné na: http://web-analytics.wikidot.com/zakladni-metriky

[14] **RAGHAV S. NANDYAL**. CMMI: a framework for building world class software and systems enterprises. New Delhi: Tata McGraw-Hill, 2004, 508 s. ISBN 0070528012.

[15] **LACKO, Branislav**. Model zralosti procesů tvorby softwaru. In: Systems on line: S přehledem ve světě informačních technologií [online]. 2005 [cit. 2021-12-14]. Dostupné na: https://www.systemonline.cz/clanky/model-zralosti-procesu-tvorbysoftware.htm

[16] **GRAUZEĽ, Jozef,** STN EN ISO 9000:2016, Systémy manažérstva kvality. . Základy a slovník (ISO 9000: 2015). Bratislava: Slovenský ústav technickej normalizácie, 2016 . 88 s .ISBN 9788097129927

[17] **GOMAA, Hassan**. Software modeling and design: UML, use cases, patterns, and software architectures. New York: Cambridge University Press, 2011, 550 s. ISBN 978-0-521-76414-8.

[18] **RÁČEK, Jaroslav**. Úvod do projektového řízení, plánování projektu. In: IS MU [online]. 2016 [cit. 2021-12-17]. Dostupné na: https://is.muni.cz/el/1433/jaro2016/PA179/um/tp_01_uvod_a_planovani.pdf

[19] **SOMMERVILLE, Ian**. Softwarové inženýrství. Brno: Computer Press, 2013, 680 s. ISBN 978-80-251-3826-7

[20] **BOCK, Jan**. Základní principy. In: Adapma [online]. 2013 [cit. 2021-12-17]. Dostupné na: http://adapma.cz/page11/styled-2/index.html

[21] RUP – Rational Unified Process. In: Testování softwaru [online]. [cit. 2021-12-17]. Dostupné na: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotnihocyklu-softwaru/rup/

[22] **RÁČEK, Jaroslav.** Agilní metodiky vývoje SW. In: IS MU [online]. 2013 [cit. 2021-12-17]. Dostupné na: https://is.muni.cz/el/fi/podzim2013/PA017/um/SWE2_07_agilni.pdf

[23] **KLAUS Olsen, MEILE Posthuma, ULRICH Stephanie.** Certifikovaný tester základnej úrovne. In: ISTQB [online.]. 2018 [cit. 2022-03-14]. Dostupné na https://castb.org/wpcontent/uploads/2020/05/ISTQB_CTFL_Syllabus_SK_2018_3.1-1.pdf

[24] **MARR, B.** Key performance indicators. UK: Pearson, 2012, 376 s. ISBN 978-0-273-75011-6

[25] **ELIAS, Romeo.** The 5 key characteristics of user-friendly software. In: Intellect [online]. 2021 [cit. 2022-03-16]. Dostupné na https://www.intellect.com/blog/the-5-key-characteristics-of-user-friendly-software

[26] **HUYNH, Flavien**. Software quality: Origin story. In: Coders Kitchen [online]. 2020 [cit. 2022-03-20]. Dostupné na: **https://www.coderskitchen.com/software-quality-origin-story/**

[27] **HUYNH, Flavien**. Software quality: From metrics to habits. In: Coders Kitchen [online]. 2022 [cit. 2022-03-20]. Dostupné na: **https://www.coderskitchen.com/software-quality-from-metrics-to-habits/**

[28] **WARD, Patrick.** Software Quality Metrics: Eplained With Examples. In: Rootstrap [online]. 2021 [cit. 2022-03-21]. Dostupné na: https://www.rootstrap.com/blog/software-quality-metrics-explained-with-examples/

[29] **ALTEXSOFT.** What software quality (really) is and the metrics you can use to measure it. In: altexsoft [online]. 2017 [cit. 2022-03-21]. Dostupné na: https://www.altexsoft.com/blog/engineering/what-software-quality-really-is-and-the-metrics-you-can-use-to-measure-it/

[30] **GOHEL, Shivam.** Five key reasons why software quality metrics matter, In: infostretch [online]. 2021 [cit. 2022-03-21]. Dostupné na: https://www.infostretch.com/blog/five-key-reasons-why-software-quality-metrics-matter/

[31] **ALVATER, Alexandra**. What are software metrics and how can you track them? In: Stackify [online]. 2017 [cit. 2022-03-21]. Dostupné na: https://stackify.com/track-software-metrics/

[32] **JIRA** [online]. [cit. 2022-04-03]. Dostupné na: https://www.atlassian.com/software/jira

[33] **Kibana** [online]. [cit. 2022-04-03]. Dostupné na: https://www.elastic.co/kibana/

[34] **BUCKSTEEG, Martin.** ITIL 2011. vy. Brno: Computer Press, 2012, 216 s. ISBN 978.-80-251-3732-1

[35] **G2.com** [online]. [cit. 2022-04-09]. Dostupné na: https://www.g2.com/search?utf8=%E2%9C%93&query=software+quality+metrics&order=popular

[36] **Visual Paradigm.** What are Scrum Artifacts? In: Visual Paradigm [online]. [cit. 2022-04-10] Dostupné na: https://www.visual-paradigm.com/scrum/what-are-scrum-artifacts/

[37] **SonarQube.** SonarQube Documentation. In: SonarQube [online]. [cit. 2022-04-10] Dostupné na: https://docs.sonarqube.org/latest/