Najčastejšie chyby pri vývoji softvéru*

Adam Melikant
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xmelikant@stuba.sk

5. November 2021

Abstrakt

V tomto článku sa budeme zaoberať najčastejšími chybami, ktoré môžu nastať pri komplexnom vývoji akéhokoľvek typu softvéru. Celý tento vývoj rozdelíme na jednotlivé celky popisujúce postupy vývoja softvéru. Tieto celky podrobne rozoberieme a poukážeme na jednotlivé nedostatky, ktoré následne môžu spôsobovať závažné problémy v praxi. Začneme od začiatku: Plánovanie potrebných informácií, zdrojov, cielený účel softvéru, rozšírenia atď. Následne rozoberieme štruktúru softvéru, chyby pri navrhovaní, programovaní či celkovej štruktúre, návrhu a funkcionalite. Dostatočne vysvetlíme zaužívané postupy, ktoré vedú k najčastejším chybám a aj to ako sa im efektívne vyhnúť prípadne čo najrýchlejšie opraviť aby sa následne zamedzilo kaskádovému vzniku ďalších problémov. Po úspešnom dokončení softvéru poukážeme na chyby v distribúcií a rôznych iných aplikáciách, ktoré sú častým nedostatkom hlavne z hľadiska dosiahnutia čo najväčšieho dopytu.

Každý plán vývoja softvéru je navrhnutý s ohľadom na finančné a časové obmedzenia dostupné pre zainteresované strany. Pravidelný výskyt chýb však môže viesť k zbytočným prieťahom v pláne. To môže spôsobiť nadmerné finančné zaťaženie a v najhoršom prípade odmietnutie produktu. Preto si vývojový tím musí byť vedomý bežných chýb, ktoré sa môžu počas procesu stať. Tu uvádzame niektoré bežné chyby, ktorým je potrebné sa vyhnúť, aby mohol pokračovať správny životný cyklus vývoja softvéru.

1 Počiatočný prieskum

Testovanie vierohodnosti nápadov pred vývojom je kľúčovým krokom, ako sa vyhnúť uviaznutiu. V prvej fáze sa vývojári a tvorcovia nápadov stretnú a dosiahnu konsenzus ohľadom vlastností a funkčnosti programu. To zahŕňa aj brainstorming s odborníkmi z odvetvia o aktuálnych trhových trendoch a požiadavkách.

Rozhodovanie o nákladových faktoroch je zároveň dôležitým faktorom životného cyklu vývoja softvéru. Tu vývojári a klient diskutujú o nákladových dôsledkoch rôznych technológií a funkcií. Vo väčšine prípadov sa táto veľmi podstatná počiatočná fáza pri vývoji softvéru jednoducho preskočí.

*Najčastejšie chyby pri vývoji softvéru, ak. rok 2020/21, vedenie: Vladimir Mlynarovic

1.1 Nepochopenie požiadaviek a ich neoverenie

Účelom vytvorenia dokumentu so špecifikáciami softvéru v počiatočnej fáze je neustále kontrolovať proces a výstupy v porovnaní s tým, čo sa očakáva na strane používateľa. Počas vývoja softvéru na zákazku sú požiadavky jedinečné a špecifikované, a preto je dôležité dodržiavať tieto požiadavky. Niektorí vývojári, ktorí v tejto oblasti nemajú veľa skúseností, tento krok v niektorých fázach vynechajú. Umožňuje, aby sa chyby dostali aj k druhej časti a následne aj zvýšili náklady podnikov. Požiadavky by sa preto mali v každej fáze dôkladne pochopiť a overiť so zainteresovanými stranami a koncovými používateľmi.

2 Zber Informácií

Na iternete existuje množstvo zdrojov, ktoré môžu byť použité či už ako inšpirácia alebo ako knižnica alebo framework do nášho softvéru. Mať vedomosť o týchto zdrojov je v počiatočných krokoch vývoja softvéru kľúčová.

Jednou z najhorších programátorských chýb je nedostatok výskumu zo strany vývojárskeho tímu. Mnoho programátorských tímov má tendenciu ignorovať časť výskumu a pokúšať sa dokončiť moduly kódovania počas samotného vývoja. Toto je jedna z hlavných príčin výroby podpriemerných produktov. Dôkladný prieskum potrebných zdrojov tiež zaisťuje, že v pracovnom procese nebudú žiadne medzery.

S vedením by sa mali riadne konzultovať konečné ciele produktu, aby sa vybrali správne technické zdroje pre projekt.

3 Plánovanie procesu

Kým je fáza výskumu dokončená a potrebné máme potrebé zdroje , ďalším krokom je plánovanie, kde delegovanie povinností vykonáva vývojový tím. Softvér sa môže líšiť v rôznych metrikách, ako je frontend a backend, a preto musia byť funkcie v súlade so zručnosťami, ktoré majú vývojári k dispozícii. V tejto fáze je tiež potrebné rozhodnúť o pracovnom modeli životného cyklu vývoja softvéru.

Kvalita a zručnosť vývoja je najdôležitejším pilierom, na ktorom projekt stojí.Často sa stáva, že organizácie majú tendenciu najímať neskúsených začiatočníkov, ktorí môžu v nepriaznivých situáciách ľahko zlyhať. Ak máte po boku skúsených vývojárov, môžete organizácii jednoducho pomôcť ušetriť finančné zdroje a čas. Najímanie skúsených vývojárov softvéru môže vyzerať ako nadmerné náklady, aj keď v skutočnosti poskytuje oveľa vyššiu návratnosť investícií ako zamestnávanie začiatočníkov.

Plán je ako cesta pre vývojárov. Funguje ako sprievodca, keď je veľkosť projektu veľká. Plánovanie nám umožňuje pracovať špecifikovaným spôsobom a dosiahnuť cieľ v rámci časovej osi. Správne testovanie v rôznych časoch je tiež dobrý spôsob, ako sa uistiť, či je tím pripravený alebo nie v prípade veľkého vývojového projektu.

Je tiež rozumné vopred rozhodnúť, že na konci každej etapy sa umiestnia náležité kontrolné opatrenia. Príliš prísny plán však môže viesť k výpadkom a problémom v budúcnosti. Preto musia byť plány dostatočne flexibilné na to, aby začlenili nové zmeny.

3.1 Zlé rozdelenie práce medzi tímom

Zmätok ohľadom rozdelenia povinností je to posledné, čo by si každý vývojársky tím želal. Na odstránenie chýb vo vývoji súvisiacich s delegovaním povinností by mal existovať riadny kontrolný bod v hierarchii tímu. Tiež správne delegovanie povinností dáva vývojárom dostatočný priestor na uplatnenie ich schopností a zručností.

Ďalej, nevenovanie pozornosti delegovaniu vedie k zmätku a prekrývaniu právomocí v tíme. To môže viesť k zahrnutiu a vylúčeniu prvkov, ktoré majú vplyv na budúce fázy vývoja. Je však tiež dôležité, aby nedochádzalo k príliš veľkému delegovaniu vedúcemu k spomaleniu procesu vývoja

4 Návrh architektúri

Architektúra programu je štruktúra, ktorá definuje vzťah medzi rôznymi prvkami. Toto zahŕňa

- Vlastnosti kvality
- Dizajnové prostredie
- Obchodná stratégia
- Ľudská dynamika
- IT prostredie

Mizerná štruktúra kódu je hlavným problémom, ktorý poškodzuje kód. Keď je definovaná zlá kompozícia, bude sa ňou pravdepodobne riadiť každý vývojár. Vždy je dobré mať štruktúru kódu, pretože v budúcnosti ušetrí veľa času. Výhody dobrej štruktúry kódu pomôžu vývojárom implementovať menej kódu. Kód sa bude jednoduchšie aplikovať, systém bude prístupnejší a chyby sa budú dať ľahšie vystopovať.

4.1 Nekomentovanie a štruktúrovanie Kódu

Ak by existoval zoznam najbežnejších chýb, ktorým sa dá vyhnúť, tento by bol na vrchole. Bolo to mnohokrát opakované a stále mnohí vývojári majú tendenciu ignorovať štruktúru a komentáre. Nekomentovanie kódu spôsobuje veľké problémy pri budúcej aktualizácii softvéru. Preto je nanajvýš dôležité, aby poskytovateľ služieb vývoja softvéru uvádzal správne komentáre vždy, keď je to potrebné.

Navyše, ak kódu neposkytnete správnu štruktúru, vyzerá to chaoticky. Čistota štruktúry kódu je rovnako dôležitá ako prehľadnosť používateľského rozhrania softvéru

5 Vývojový proces

Fáza vývoja je aktuálna fáza, v ktorej sa realizujú všetky plány podľa ich požiadaviek na zdroje. Vývojová fáza si vyžaduje intenzívne testovanie jednotiek v každej fáze, aby sa zabezpečilo, že akákoľvek chyba alebo chyba bude zachytená

už na začiatku. Okrem toho sa najväčšia pozornosť venuje komunikácii medzi rôznymi oddeleniami tímu vývoja softvéru, ako aj zainteresovanými stranami.

5.1 Komunikácia nie je jasná

Komunikačný kanál v projekte vývoja softvéru možno rozdeliť na dve časti, a to vnútrotímovú a medzitímovú komunikáciu. Oba tieto kanály by sa mali efektívne využívať, aby sa predišlo akýmkoľvek nejasnostiam. Ak to neurobíte, môže to viesť k narušeniu rýchlosti projektu, čo môže mať za následok nadmerné náklady.

Rovnako dôležitá je správna komunikácia s manažérskym a marketingovým tímom. Pomáha to pri overovaní faktorov životaschopnosti v každej fáze, čo uľahčuje rozhodovanie o budúcich postupoch. Toto sa stáva ešte dôležitejším v procese vývoja softvéru na zákazku, pretože každá funkcia musí byť správne koordinovaná.

5.2 Snaha o dosiahnutie dokonalosti

Každý projekt je nejakým spôsobom obmedzený finančnými a časovými limitmi a zbytočné detaily do dokonalosti tomu môžu brániť. Pri zabezpečení toho, aby nedochádzalo k žiadnym kompromisom s kvalitou konečného produktu, je tiež potrebné, aby nedochádzalo k narušeniu kvôli detailovaniu. Lesknutie do detailov môže vyžadovať tvrdú prácu, ktorú možno nahradiť už dostupnými riešeniami. Vždy tu bude niečo, čo bude vynechané, a preto by sa vývojár mal najprv zamerať na dokončenie hlavných častí. To pomáha pri poskytovaní vtáčej perspektívy prvkov, ktoré si vyžadujú ďalšie zmeny, čo zase pomáha pri odhade času dokončenia.

5.3 Preskočenie testovacej časti produktu

Testovanie softvéru je dôležité tak v čase vývoja, ako aj na konci procesu vývoja. To zahŕňa testovanie jednotiek aj záťaže, umiestnenie správnych testovacích systémov na konci požadovaných intervalov vedie k včasnej identifikácii chýb. Na druhej strane absencia týchto systémov môže z dlhodobého hľadiska spôsobiť nenapraviteľné škody.

Preskočenie alebo unáhlenie procesu testovania je teda z dlhodobého hľadiska kontraproduktívne, a preto si v počiatočnom pláne vyžaduje primerané časové vyčlenenie na testovanie. Aj keď je potrebné mať na pamäti, že testy sú v súlade s predpokladanými vzormi používania.

6 Programatorské chyby

Programátorská chyba je druh softvérovej chyby, ktorú urobil programátor pri vytváraní počítačového programu. Programátorskú chybu, ktorá v softvéri spôsobí bezpečnostný problém, označujeme ako zraniteľnosť. Program využívajúci zraniteľnosť je exploit. V realtite sa musíme pokúsiť týmto chbám vyhnúť. V dnešnej dobe existuje mnoho dobrých, ale aj zlých hackerov a teda musíme dostatočne zabepečiť softvér proti akémukoľvek vniknutiu a nežiaducemu správaniu.

Chyby v programoch sú dôsledkom ľudského faktora. Vznikajú prehliadnutím, alebo vzájomným nepochopením vo vývojovom tíme počas špecifikácie kódovania a dokumentácie.

6.1 Ladenie/Debugovanie

Nájsť a opraviť chybu, čiže "debugovanie", bola vždy dôležitá časť programovania. So zvyšujúcou sa zložitosťou programov stúpa počet chýb a tiež obtiažnosť ich zachytiť a opraviť. Často sa stáva, že programátori strávia viac času hľadaním a opravovaním chýb, než písaním nového kódu. Softvéroví testeri sú profesionáli, ktorých jedinou úlohou je nájsť a opraviť chyby, alebo napísať kód pre testovanie. Pri niektorých projektoch je viac prostriedkov vynaložených na testovanie ako na vývoj.

Obvykle najťažšia časť ladenia je nájsť chybu v kóde. Akonáhle je odhalená, jej oprava nebýva obvykle problém. Jednou z pomôcok pri debugovaní je tzv. krokovanie programu, pri tomto procese je program prechádzaný po jednotlivých príkazoch pri neustálom sledovaní premenných. Bez tejto možnosti sa používa iný postup, v rôznych častiach programu sú premenné vypisované napr. do konzoly, čo napomáha k lokalizácii chýb.

Avšak aj týmito pomôckami je niekedy lokalizácia chýb umenia, spravidla sa stáva, že chyby v jednej časti programu spôsobia pád v úplne inej časti. Niektoré chyby sú spôsobené zlým myslením alebo plánovaním zo strany programátora. Takéto chyby vyžadujú prepísanie časti kódu.

Posledným typom chýb sú chyby, ktoré nemajú s kódom nič spoločné. Pokiaľ sa programátor spolieha na dokumentáciu k hardvéru a tá nie je presná, môže byť program napísaný dobre s ohľadom na dokumentáciu ale už nie s ohľadom na skutočný hardvér.

6.2 Dokumentácia napísaná až za implementáciou

Veľa organizácií sa snaží o komplexnú dokumentáciu. Uistit sa, že je všetko zdokumentované, ale dokumentácia nie je vždy urobená v správnom čase. Problém je v tom, že dokumentácia sa často robí až po napísaní kódu. Dokumentácia musí byť vykonaná pred a počas kódovania, nikdy potom. Pred začatí implementácie by sme mali začať podrobnú špecifikáciu a projektovú dokumentáciu. kedykoľvek je dokument nejednoznačný, najprv ho upravme. Nielenže to zaisťuje, že dokument zostáva aktuálny, ale zabezpečuje, že programátor implementuje to, čo dokument špecifikuje. Aktualizácia dokumentácie aj počas implementácie slúži ako kontrola kódu. Programátori často nachádzajú chyby v ich kóde, keď o tom píšu. Napríklad programátor môže napísať: "Po úspechu táto funkcia vráti 1."Programátor si potom myslí: „Ak nie je úspech, čo je sa vráti?"Pozrie sa na svoj kód a možno si uvedomí, že ten nedostatok, scenár úspechu nebol riadne implementovaný.

6.3 Opätovné použitie kódu, ktorý nie je určený na opätovné použitie

[4] Kód, ktorý nie je určený na opätovné použitie, nebude vo forme abstraktný dátový typ alebo objekt. Kód môže mať vzájomnú závislosť s iným kódom, takže ak sa vezme celý, tam je viac kódu, ako je potrebné. Ak sa odoberie len

časť, treba ju dôkladne vypreparovať, čím sa zvyšuje riziko nevedomky vystrihnutie niečoho, čo je potrebné, alebo neočakávaná zmena funkčnosti. Ak kód nie je určený na opätovné použitie, je lepšie analyzovať, čo robí existujúci kód, a potom ho prepracovať, znovu implementovať kód ako dobre štruktúrovaný softvér na opakované použitie komponentov. Odtiaľ je možné kód znova použiť. Prepísanie tohto modulu zaberie menej času ako vývoj a čas ladenia potrebný na opätovné použitie pôvodného kódu. Bežná mylná predstava je, že pretože softvér je definované v samostatných moduloch, je prirodzene opakovane použiteľné. Toto je samostatná chyba a súvisiaca s tvorbou softvéru s príliš veľa závislosťami.

6.4 Žiadna analýza pamäte

[4] Množstvo pamäte vo väčšine vstavaných systémov je obmedzené. Napriek tomu väčšina programátorov netuší, čo je to za pamäť dôsledky sú pre akýkoľvek ich dizajn. Keď sú požiadaní koľko pamäte využíva určitý program alebo dátová štruktúra, bežne sa mýlia rádovo. V mikrokontroléroch a DSP je významný rozdiel výkon. Môže existovať medzi prístupom k ROM, interná RAM a externá RAM. Kombinovaná analýza pamäte a výkonu môže pomôcť pri čo najlepšom využití efektívnej pamäťe umiestnením najpoužívanejších segmentov kódu a dát do najrýchlejšej pamäte. Pridáva sa procesor s vyrovnávacou pamäťou ešte ďalší rozmer výkonu. Analýza pamäte je u väčšiny dnešných veľmi jednoduchá pomocou vývojových prostredí. Väčšina prostredí poskytuje mapovací súbor počas fáz kompilácie a prepojenia s použitými pamäťovými údajmi. Kombinovaná analýza pamäte a výkonu je však oveľa náročnejšia, ale určite sa oplatí ak je výkon kritickou zložkou.

7 Testovanie

Po dokončení vývojovej fázy sa vykoná testovanie konečného produktu. Jedným z najdôležitejších testov, ktoré je potrebné vykonať, je záťažové testovanie. V rámci toho je softvér vystavený rôznym umelým stresovým situáciám, aby sa zabezpečilo bezproblémové fungovanie v prípade zaťaženia serverov alebo v prípade nadmernej prevádzky.

Kód bez testovania nie je spoľahlivý a bráni vývojárom používať ho. Refaktorovanie sa stáva oveľa zložitejším, pretože akákoľvek zmena a vývojári nebudú vedieť, či určitá funkcia funguje alebo nie. Šanca na nájdenie nesprávneho kódu je vážna v systémoch, ktoré nie sú testované. Pri testovaní sa kód stáva oveľa spoľahlivejším a bezpečnejším pre vývojárov pri hľadaní chýb.

8 Distribúcia a Propagácia softvéru

Po ukončení vývojovej a testovacej fázy je nasadzovacia fáza určená na to, aby sa konečný produkt dostal k jeho užívateľovi. To zahŕňa aj dostupnosť včasných aktualizácií softvéru. Dnes je možné softvér nasadiť v živých prostrediach vďaka dostupnosti cloudových služieb, ako sú Microsoft Azure a Amazon Web Services

Okrem toho vývojové spoločnosti používajú viacero modelov na prácu na projekte vývoja softvéru. Tieto modely sú užitočné, aby ste sa vyhli chybám

vývojárov softvéru. Tieto modely sa vyberajú na základe počiatočného prieskumu požiadaviek, ako je uvedené vyššie. Waterfall, Agile a Spiral sú niektoré z najpoužívanejších modelov v procese vývoja softvéru.

Nemať životný cyklus vývoja softvéru je ako snažiť sa dostať do cieľa bez použitia akejkoľvek mapy. SDLC pomáha pri znižovaní nepotrebných akcií, čo zase pomáha znižovať náklady a čas potrebný na vývoj. Pri dodržaní úplného kontrolného zoznamu zaistíte, že v programe nechýbajú žiadne prvky.

9 Záver

Vývoj softvéru je dlhý proces a chyby sa môžu pri malej nedbanlivosti kedykoľvek vkradnúť. Počas prvých dní vývoja sú vyššie uvedené chyby medzi vývojármi najčastejšie. Niekedy, keď sú chyby identifikované v druhej časti životného cyklu vývoja softvéru, sú pre podniky veľmi nákladné. Skúsení vývojári alebo etablované vývojárske spoločnosti vedia, ako sa s týmito problémami vysporiadať a svoj proces majú vybudovaný tak, aby zabezpečili kvalitnú prácu s minimálnou alebo žiadnou stratou. Tieto začiatočnícke programátorské chyby sa dajú ľahko identifikovať, keď máte niekoho skúseného v projekte.

Identifikácia problémov a úprava v správnom čase môže ušetriť obrovský čas a zdroje pre podnikanie. Každý krok alebo stupeň v SDLC má rôzne výstupy, ktoré možno testovať podľa dokumentu špecifikácie požiadaviek. Tento jediný krok zastaví akúkoľvek chybu, ktorá sa rozšíri až do konca a pokazí celý produkt. Existuje mnoho ďalších metód na odstránenie chýb vo vývojovej časti.

Pri dôležitých projektoch by sa týmto chybám malo za každú cenu vyhnúť. Tieto bežné chyby pri vývoji softvéru sú často najhoršími programátorskými chybami pre podniky, pretože vedú k obrovským stratám. Ako obchodná spoločnosť je najlepším krokom konzultácia s odborníkom na projekt a najatie softvérovej spoločnosti s dobrým portfóliom projektov. Tieto IT spoločnosti majú dobré skúsenosti s riešením bežných problémov a dodávaním rozsiahlych projektov v časovom rámci. Consulting Whiz je jednou z takýchto spoločností na vývoj softvéru v USA. Spoločnosť Consulting Whiz, ktorá pracuje viac ako 12 rokov, umožnila značkám zvýšiť svoje príjmy pomocou bezchybných služieb vývoja softvéru. Ich portfólio má niektoré z projektov, ktoré sú dnes úžasne úspešné.

Literatúra

[1] Henry P Stevenson edy. Proceedings of a codasyl programming language committee symposium on structured programming in cobol ¾ future and present. http://users.mct.open.ac.uk/mj665/COBOLSP.pdf.

[2] Fabio Silva Lima. 21 common lifetime mistakes about software development. https://www.c-sharpcorner.com/article/21-common-lifetime-mistakes-about-software-development/.

[3] Victor Osetskyi. 10 common mistakes in agile software development. https://dzone.com/articles/10-common-mistakes-in-agile-software-development.

[4] David B. Stewart. Twenty-five most common mistakes with real-time software development. https://webpages.uncc.edu/~jmconrad/ECGR4101-2005-08/Twenty-Five.pdf.

[5] Aspire Blog Team. The 10 most common mistakes made in software development. https://blog.aspiresys.pl/technology/mistakes-in-software-development/.