Časté chyby pri vývoji softvéru*

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

14. December 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.

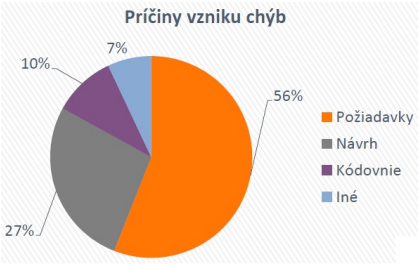
1 Úvod

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.Diagram 1 opisuje životný cyklus sú vyzačené fázy, v ktorých sa najčastejšie vyskytujú chyby.

Obr. 1: Životný cyklus softvéru.

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

Graf 2 znázorňuje precentuálny výskyt chýb v rôznych fázach vývoja softvéru.



Obr. 2: Príčiny vzniku chýb pri vývoji softvéru, prevzaté zo zdroja [1].

2 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.

2.1 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.

Typy požiadaviek:

- Požiadavky na prevádzku systému (počet používateľov, ...)
- Požiadavky na softvér (efektívnosť, bezpečnosť, hardvérové prostredie, OS, iný spolupracujúci SW)
- Externé požiadavky (legislatívne, etické požiadavky)

[1]

3 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.

4 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.

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.

4.1 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

5 Návrh architektúri

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

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

Fáza návrhu definuje fungovanie softvérovej aplikácie. V tejto fáze práce tím robí rozhodnutia o návrhu softvéru o architektúre a mení softvér na riešenie. Táto fáza zahŕňa tvorbu návrhových dokumentov, programovacích pokynov, diskusiu o nástrojoch, postupoch, časoch spustenia a rámcov, ktoré tímu pomôžu splniť požiadavky na softvér, špecifikácie a ciele definované vo fáze práce. [5]

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.

5.1 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

6 Vývojový proces

Proces vývoja softvéru je množina aktivít, ktorých koneným výsledkom je softvérový produkt. V každej organizácii je tento proces iný. V každej „píšu softvér“ trochu iným spôsobom. Rozdiely môžu by napríklad v intenzite dokumentovania alebo v intenzite testovania. V každom prípade však vývoj softvéru prechádza vždy viac-menej rovnakými etapami: analýza a špecifikácia požiadaviek, návrh, implementácia a testovanie, integrácia, údržba. Vývoj je podporovaný aktivitami, ako sú: riadenie projektu, zabezpeovanie kvality, správa konfigurácií a riadenie zmien. [5] 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.

6.1 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.

6.2 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.

6.3 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.

7 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. Aktuálne zaznamenávame chyby v rozmedzí od 15 do 65 chýb na 1000 riadkov. Tabuľka 1 znázorňuje priemerná počet cháb na 1000 riadkov pre rôzne distribúcie softvérov počas niekoľkých rokov.

Tabuľka 1: Priemerný počet chýb na 1000 riadkov v rôznych softvéroch

Rok	Softvér	Počet riadkov [tisíc]	Počet chýb/1000 riadkov
2003	Linux kernel 1.3.13	52	32
2009	Linux kernel 2.6.29	75.5	21
2009	Linux kernel 2.6.32	79	14
2010	Linux kernel 2.6.35	92.6	34
2012	Linux kernel 3.6	115	25
2015	Linux kernel pre-4.2	127.5	17
2021	Linux kernel 5.15	150	18

7.1 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.

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

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. [7]

7.3 Žiadna analýza pamäte

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]

8 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.

9 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.

Spoločenské súvislosti Táto prednáška bola veľmi zaujímavá. Odohrala sa s dekanom fakulty, čož bola pre mňa osobne sama o sebe dosť vzrušujúca udalosť. Pán dekan nám ponúkol možnosť opýtať sa na rôzne udalosti, ktoré sa stali na tejto fakulte a bol ochotný zareagovať v podstate na všetky naše otázky. IT sféra sama o sebe je veľmi rozliahlá. Uvedenie tejto témy ma fascinovalo a začal som premýšľať o svojej budúcnosti v IT. Aktuálne stále nie som rozhodnutý, čo presne ma najviac zaujalo, však minimálne som si upresnil niektoré podstatné rozhodovacie faktory a verím, že celá táto prednáška mi bola nie len nápomocná ale aj veľmi zaujímavá.

Historické súvislosti Táto prednáška mi dala mnoho nových informácií. Boli to hlavne historické informácie o veľmi zaujímavých ľudoch, ktorých si celý svet bude ešte dlhu dobu pamätať. To všetko čo dokázali si určite budeme vážiť a pamätať ešte dlhú dobu. Za skutočne krátku dobu dokázali posunúť ich výckum, prácu na nový level a rozšíriť svoje meno po celom svete. Bola to skutočne zaujímava prednáška, na ktorej som sa dozvedel množstvo nových vecí a tak-

mer určite by som ocenil aj ďalšie podobné prednášky o inováciach či známych objavov v minulosti.

Technológia a ľudia

Udržateľnosť a etika

10 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 pokročilejších častiach 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 obrovské množstvo času a zdrojov určených pre podnikanie.

Literatúra

[1] Proces vývoja softvéru. http://www.kiwiki.info/index.php/Proces_v%C3%BDvoja_softv%C3%A9ru.

[2] Mária Bieliková . Peter Dolog. Tvorba softvéru v treťom tisícročí. http://www2.fiit.stuba.sk/~bielik/courses/msi-slov/kniha/2002/hobiti.pdf.

[3] Henry P Stevenson edy. Proceedings of a codasyl programming language committee symposium on structured programming in cobol $\frac{3}{4}$ future and present. http://users.mct.open.ac.uk/mj665/COBOLSP.pdf.

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

[5] Pavol Mederly. Softvérové inžinierstvo. https://fmfi-uk.hq.sk/Informatika/Principy%20Tvorby%20Software/2001/p1-uvod-studmat.pdf.

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

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

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