Udržovateľnosť a Open Source Software

PAVOL GONO

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava

Abstrakt. Cieľom tejto eseje je poukázať na udržovateľnosť softvérových produktov a faktory vplývajúce na udržovateľnosť softvéru. Porovnáva sa udržovateľnosť medzi softvérom s otovreným zdrojovým kódom a softvérom s uzavretým zdrojovým kódom. Analyzujú sa možnosti merať udržovateľnosť, existujúce metriky, kompozitné metriky. Hodnotia sa vykonané merania v reálnych softvéroch v tejto oblasti.

Úvod

Podnetom pre túto esej bola práca autorov Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis a Apostolos Oikonomou s názvom „Open source software development should strive for even greater code maintainability“ [1]. Autori sa venujú problematike merania udržovateľnosti Open Source Software, konkrétne piatich populárnych projektov.

V tejto eseji chcem bližšie rozobrať možnosti merania udržovateľnosti, uviesť vhodné metriky a priblížiť ďalšie merania konkrétnych softvérových projektov. Tiež uvediem všeobecné zásady na tvorbu dobre udržovateľných programov.

V prvej časti eseje je definovaná udržovateľnosť a údržba softvéru. Druhá časť rozoberá Open Source Software. V tretej časti sa hovorí o rozdieloch medzi Open Source Software a Closed Source Software. Štvrtá časť analyzuje možnosti merania softvéru. V piatej časti sú uvedené výsledky merania softvérov a v šiestej časti sú zásady tvorby dobre udržovateľných programov.

Čo je to udržovateľnosť

Jedna z definícií udržovateľnosti je: Úsilie, ktoré treba vynaložiť na ďalší vývoj a údržbu výrobku podľa meniacich sa potrieb zákazníka a aj meniaceho sa okolia. [2]

S udržovateľnosťou úzko súvisí samotná údržba softvéru. Ak je dobrá udržovateľnosť, ľahšie a rýchlejšie sa vykonáva údržba. Údržbu rozdeľujeme na [3]:

- opravnú (corrective maintenance): odstraňovanie všetkých chýb, ktoré pretrvávajú v systéme (chyby špecifikácie, návrhu, implementácie, dokumentácie).
- prispôsobovaciu (adaptive maintenance): menenie softvéru v závislosti od zmien okolitého prostredia, tiež zmeny časti softvéru v dôsledku modifikácie inej časti.
- zlepšovaciu (perfective maintenance): vylepšovanie vlastností produktu, ktoré neboli zahrnuté do špecifikácie pôvodného systému, tiež zmeny v závislosti od zmien problémovej oblasti a požiadaviek používateľa.
- preventívnu (preventive maintenance): modifikácie produktu s cieľom zlepšenia ďalšej údržby.

Aký softvér zaraďujeme k Open Source Software

Existuje aj viac definícií systémov s otvoreným zdrojovým textom programu (ďalej OSS – Open Source Software). Najčastejšie používaná definícia sa nachádza na stránke Open Source Iniciatívy [4]. Je rozpísaná v bodoch, ktoré sa venujú voľnej redistribúcii, zdrojovému kódu, odvodenej práci, integrite zdrojového kódu autora a pod. Tu sú však zdôraznené iba požiadavky na licenciu OSS (umožnenie bezplatného prístupu k danému softvéru a jeho zdrojovým textom, každému záujemcovi umožniť modifikáciu daného softvéru, ...). Príkladom OSS licencií je GPL, BSD, apache license, Mozilla Public License. Pre účely tejto eseje bude vhodnejšie chápať OSS ako softvér, pri ktorého vývoji sa použil špecifický proces, najčastejšie nazývaný ako Katedrála a Bazár [5].

Rozdiely medzi OSS a CSS z pohľadu udržovateľnosti

Obyčajný začiatok OSS projektu: vývojár alebo malá skupinka vývojárov napíše prvú verziu softvéru, táto sa poskytne na Internet, a pozvú sa ďalší vývojári, aby prispievali kúskami kódu. Niektorí vývojári majú hlavné slovo, ktorí riadia, akým smerom sa bude vývoj uberať, aké úpravy kódu sa uskutočnia, časový plán releasov.

Obyčajný začiatok CSS projektu: manažéri softvérovej spoločnosti sa rozhodnú vyvíjať určitý produkt (napr. na základe ponuky trhu, požiadaviek zákazníka), vytvorí sa tím vývojárov, dohodne sa časový plán a s určitým balíkom peňazí sa začne vyvíjať projekt.

Stáva sa, že CSS vývoj sa zmení na OSS, napr. aby sa dosiahli výhody takéhoto vývoja. Taktiež sa stáva, že OSS sa zmení na CSS, zvyčajne však pre iné príčiny.

V CSS nastáva podľa mňa často nasledujúca situácia. Príde zákazka na vývoj softvérového produktu, stanoví sa pomerne krátka doba, za ktorú je potrebné produkt vytvoriť. Programátori v snahe splniť požiadavky vyprodukujú výsledok rýchlo, ale už

sa nemyslí veľmi na nasledujúcu údržbu produktu. Môže existovať podvedomé odmietanie písanie prehľadného kódu – Ja si túto časť programujem, keď bude treba spraviť zmenu, ja ju budem opravovať, tak načo písať nejaké ďalšie komentáre, veď ja tomu rozumiem. Pri OSS projektoch programátori rátajú (alebo by mali rátať) s tým, že ich kód bude určite čítať aj niekto iný, takže je tu väčší predpoklad, že sa kód napíše prehľadnejšie. Ťažko to však presne potvrdiť a už vôbec sa to nedá zovšeobecniť.

V OSS je často nekompletná dokumentácia (možná výhovorka – dajú sa pozrieť zdrojové texty) alebo neexistujúca technická podpora (hlavne u menších projektov sa vyskytuje prístup - dorob si sám). Často sa ale vyskytujú aj opačné prípady, napríklad z vlastných skúseností - ak som mal problémy s používaním niektorých open-source programov, veľakrát som dostal rýchlu odpoveď na mailing-liste.

Merateľnosť udržovateľnosti

Dostatočne presným meraním udržovateľnosti by bolo merať strednú dobu potrebnú na opravy chyby, strednú dobu potrebnú na pochopenie logiky modulu a strednú dobu potrebnú na sprístupnenie príslušnej informácie v dokumentácii.

Takéto meranie by ale bolo v praxi veľmi nepohodlné, pretože by si programátori museli zapínať a vypínať stopky pri bežnej činnosti. Taktiež by sa nedalo takéto meranie urobiť externe nezúčastnenými osobami. Výhodnejším prístupom sa javí hodnotiť samotný zdrojový text softvéru, pretože ten veľa hovorí o samotnom softvéri. Vlastnosti ako štýl programovania, jazyk, komentáre, samoopisnosť, zložitosť vplývajú na samotné vlastnosti softvéru. V modernom softvérovom inžinierstve platí domnienka, že externé charakteristiky kvality sú vo vzťahu k interným charakteristikám kvality. To znamená, že merania v zdrojových textoch môžu do určitého rozsahu predpovedať externé charakteristiky kvality systému, ako sú udržovateľnosť, spoľahlivosť, rozšíriteľnosť, alebo prenosnosť.

Autori McCall a kolektív už v 70 rokoch definovali takéto faktory ovplyvňujúce udržovateľnosť: jednoduchosť, presnosť, samoopisnosť, modularita.

V súvislosti so zlou udržovateľnosťou môžu vznikať problémy pri údržbe [3]:

- Ťažkosti so sledovaním, riadením vývoja softvéru, ktorý existuje v mnohých verziách.
- Problémy s porozumením programov „niekoho iného“.
- Na údržbu sa často nasadzujú neskúsení softvéroví inžinieri, podceňovanie údržby.
- Udržiavané programy vznikli často pred veľa rokmi, nepoužili sa moderné techniky, nie sú štruktúrované, okomentované.
- Zmeny často spôsobujú chyby a následne ďalšie zmeny.
- Degradácie štruktúry meneného softvéru
- Vo všeobecnosti nízka kvalita dokumentácie

- Udržiavaný systém často nemožno vysadiť z prevádzky

Autori článku [1] používajú tieto metriky pre meranie zdrojových kódov:

- počet riadkov textu programu (LOC) – meria fyzickú veľkosť programu bez prázdnych riadkov a komentárov. (Táto metrika môže vyjadrovať celkovú zložitosť systému, čas potrebný na pochopenie systému)
- Percento riadkov komentárov vzhľadom na počet riadkov textu programu (PerCM). Vyjadruje „samoopisnosť“ zdrojového textu.
- Metrika „Halstead Volume“ (V). n1 = počet odlišných operátorov, n2 = počet odlišných operandov, N1 = celkový počet operátorov, N2 = celkový počet operandov, n = n1 + n2 (slovník programu), N = N1 + N2 (dĺžka programu). Nakoniec $V = N * (\log_2 n)$ je výsledná zložená metrika. Poskytuje alternatívny pohľad na veľkosť programu.
- Cyklomatické číslo V(g). Táto metrika sa zakladá na počte nezávislých ciest v grafe riadenia programu. Jeho hodnota závisí na počte vetiev vytvorených vetvením programu (if-else konštrukcie). Meria štrukturálnu zložitosť programu.

Podľa štúdie 25 miliónov riadkov kódu v National Security Agency [6], metriky na meranie udržovateľnosti a čitateľnosti boli použité tieto:

- Počet riadkov kódu. Nemá prekročiť 62 riadkov kódu na jednu funkciu.
- Počet vykonávateľných príkazov. Nemá prekročiť 50 vykonávateľných príkazov na jednu funkciu.
- Počet riadkov komentárov. Mal by byť najmenej 60% na funkciu. Toto môže trochu udiviť, ale NSA má veľmi veľa zastaraného kódu, ktorý sa niekedy používa aj desiatky rokov. Slabo komentovaný kód je ťažké pochopiť a horšie sa mení.
- Rozpätie odkazu na premennú. Najväčší priemerný počet riadkov medzi priradením hodnoty do premennej a použitím premennej vo funkcii. Nemal by byť väčší ako 10-12 riadkov z rôznych bezpečnostných dôvodov.

Autori článku [7] skúmajú 2 merateľné veličiny

- Globálna zviazanosť. Zviazanosť je miera previazanosti súčiastok, ideálne keď je minimálna. Globálna zviazanosť je jav, keď sa súčiastky (moduly) odvolávajú na spoločné údaje. Je to nepríjemný jav, pretože chyba v jednej súčiastke môže spôsobiť nesprávne správanie v iných súčiastkach. Ťažko sa určuje kto a kedy modifikoval spoločné údaje. Priame väzby medzi súčiastkami sú skryté, z toho vyplývajú problému pri zmenách a údržbe. Je ľahké si predstaviť, že väčšia globálna zviazanosť má nepriaznivý vplyv na udržovateľnosť. Metrika sa určila ako počty odkazov na globálne premenné (nie konštanty), pričom v rámci jedného modulu sa viacnásobný odkaz na tú istú premennú počítal iba raz.

- Počet riadkov kódu (LOC).

Merania zdrojových textov produktov OSS

Autori [1] chceli použiť kompozitnú metriku, ktorá by vyjadrovala jedno číslo, ktorým by sa ľahko porovnávali rôzne softvéry. Použili kompozitnú metriku – Maintainability Index (MI), ktorá bola navrhnutá SEI v [8] ako najviac vhodný nástroj na meranie udržovateľnosti pre systémy, na ktoré sa kladú vysoké nároky na kvalitu. Koeficienty na výpočet tohto indexu boli kalibrované podľa mnohých systémov, ktoré udržoval Hewelt Packard:

MI = 171 – 5.2ln(avgV) – 0.23avgV(g) – 16.2ln(avgLOC) + 50sin(sqrt(2.4avgPerCM))

avgV znamená priemerné Halstead Volume na modul, a podobne aj avgV(g), avgLOC a avgPerCM. Vyššie hodnoty MI znamenajú vyššiu udržovateľnosť. Vzorec berie do úvahy veľkosť, zložitosť a samoopisnosť zdrojového textu.

Sčítanec 50.sin(...) znamená funkciu, ktorá ma lokálne maximum v prípade avgPerCM=1,028. Vyššie hodnoty MI tiež získame, keď avgV bude nižšie (menšia zložitosť podľa Halstead-a), keď avgV(g) bude nižšie (menšie cyklomatické číslo) a keď avgLOC bude nižšie (menšia veľkosť systému).

Predbežné závery autorov [1]:

1. Použitím nástrojov ako MI na meranie, kvalita OSS kódu vyzerá byť rovnaká, prípadne lepšia ako kvalita CSS kódu. Môže to byť zapríčinené motiváciou skúsených OSS programátorov konkurovať CSS programátorom. Vysoká motivácia môže byť považovaná za veľkú výhodu v OSS.
2. OSS projekty potrebujú obozretnú individuálnu analýzu kvôli náhlym zmenám medzi rôznymi verziami softvéru, pre ktoré sa rozhodli koordinátori OSS projektu. Takéto zmeny majú veľký vplyv na usporiadanie OSS softvéru. Štrukturálna analýza kódu môže poukázať na komponenty, ktoré sú najviac rizikové a môžu zapríčiňovať najviac problémov. Takéto komponenty sa dajú identifikovať pomocou uvedených meraní.
3. Kvalita kódu OSS trpí tými istými problémami ako kvalita kódu CSS. Znehodnocovanie udržovateľnosti časom je typickým fenoménom a vznikajú tým zastarané CSS systémy. Je logické, že to isté sa bude diať s OSS systémami. To znamená, že v OSS projektoch je taktiež nutná vhodná preventívna údržba a prípadná reštrukturalizácia.

Závery merania udržovateľnosti linuxového jadra [7]:

- Počet riadkov kódu narastá so vzrastajúcimi verziami jadra lineárne. Každá ďalšia verzia jadra ponúka novú funkcionalitu, preto je logické, že sa počet riadkov zväčšuje. Fakt, že veľkosť jadra vzrastá iba lineárne, by mohla byť indikácia, že jadro a jeho moduly sú dobre navrhnuté – iba malé množstvo

prídavného kódu je nutné vložiť do rozhrania jadra a modulov, aby sa dosiahla nová funkcionalita.

- Množstvo výskytu globálnej zviazanosti narastá so vzrastajúcimi verziami jadra exponenciálne. Dá sa predpokladať, že takýto rast sa nezastaví pokiaľ sa linux kompletne nezmení štruktúru na minimálne používanie globálnej zviazanosti. V opačnom prípade sa dá očakávať časom zhoršujúca sa udržiavateľnosť.

Pre programátorov

Existujú všeobecné zásady, ktorými by sa mal riadiť vývojári, aby dosiahli dobrú udržovateľnosť zdrojových textov:

- ak je telo funkcie niekoľko desiatok riadkov dlhé, treba uvažovať nad rozdelením funkcie do viacerých
- aj je ten istý kód napísaný len s miernymi modifikáciami dvomi alebo viacerými spôsobmi, pravdepodobne sa to urobilo nevhodne (treba pouvažovať nad jednou funkciou s parametrami)
- používať šablóny namiesto opakovania kódu (ak to programovací jazyk dovoľuje)
- používať konštanty v čo najväčšej miere (ak to programovací jazyk dovoľuje)
- používať objekty polí namiesto polí definovaných na mieste použitia
- ak sa viaceré premenné používajú často spoločne, zapuzdriť ich do objektu
- ak viaceré funkcie pracujú iba s jedným typom objektov, urobiť v objekte tieto funkcie metódami
- už len základná refaktorizácia urobí zdrojový text čitateľnejším a organizovanejším

Záver

V tejto eseji sa bližšie rozobrali možnosti merania udržovateľnosti, uviedli sa vhodné metriky a priblížili sa ďalšie merania konkrétnych softvérových projektov. Verím, že čitateľom ponúkli lepší prehľad v tejto problematike.

Čo sa týka udržovateľnosti OSS projektov verzus CSS projektov, podľa mňa závisí hlavne na konkrétnom projekte, na konkrétnej situácii a na konkrétnych ľuďoch. Existuje veľa projektov, ktoré majú mizernú kvalitu a udržovateľnosť a sú projekty, ktorých zdrojové kódy je radosť čítať a človek má po chvíli dojem, že tomu rozumie. Rovnako pre OSS, rovnako pre CSS.

Použitá literatúra

1. Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis, and Apostolos Oikonomou. Open source software development should strive for even greater code maintainability. *Communications of the ACM*, 47(10):83-87, October 2004.
2. Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.
3. Bieliková, M.: *Softvérové inžinierstvo. Princípy a manažement*. Vydavateľstvo STU, Bratislava, 2000.
4. The Open Source Initiative. Open source definition, version 1.9; www.opensource.org/docs/definition.php
5. Raymond, Eric S.: The Cathedral and the Bazaar. www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/cathedral-bazaar.ps.
6. Drake, T. Measuring software quality: A case study. *IEEE Computer 29,* 11 (1996), 78–87.
7. Schach S.R., Jin B., Wright D.R., Heller D.Z., and Offutt A.J. Maintainability of the Linux kernel. In *IEE Proceedings—Software Engineering 149,* 1 (2002), 18–24.
8. Maintainability index technique for measuring program maintainability. Software Technology Review, SEI; www.sei.cmu.edu/str/descriptions/mitmpm_body.html.

Annotation

Maintainability and Open Source Software

The aim of this essay is to mention maintainability of software products and factors which influence maintainability of software. I compare maintainability between Open Source Software and Closed Source Software. I analyse possibilities of measuring maintainability, existing metrics and composite metrics. Measurements of real software products are evaluated in this section.