Údržba SW systémů

- údržba SW = aktivity, které jsou prováděny po uvolnění programu, resp. po jeho dodání zákazníkovi

- zahrnuje především:
  - opravu chyb SW (corrective maintenance)
  - přizpůsobení SW změnám prostředí, ve kterém běží - OS, periferie apod. (adaptive maintenance)
  - přidávání nové funkčnosti nebo změna funkčnosti na základě požadavků uživatele (perfective maintenance)

- v číslech - cca 17% údržby se týká opravy chyb, 18% přizpůsobení SW změnám prostředí a 65% přidávání nebo změny funkčnosti

- tvoří cca 50% až 75% celkového vývoje, pro obtížně změnitelné systémy (např. zapouzdřené systémy reálného času) až 80%

- cena údržby systému postupně stoupá -> je efektivní systém navrhnout a implementovat tak, aby se cena údržby snížila

- proces údržby je spuštěn množinou požadavků na změny od uživatelů systému, managementu nebo od zákazníka
    - provedeme vyhodnocení ceny a dopadu změn
    - navržené změny jsou ne/schváleny, některé jsou odloženy (rozhodnutí o schválení/neschválení změn je do určité míry ovlivněno udržovatelností komponenty, ve které změnu provádíme)
    - změny jsou implementovány a ověřeny (v ideálním případě: změna specifikace systému, návrhu, implementace, otestování systému)
    - dodána nová verze

Nestrukturovaná údržba

- výše uvedené v ideálním případě
- pracnost a cenu údržby ve skutečnosti zvyšují tyto faktory:

  - po dodání produktu je tým obvykle rozpuštěn, lidé jsou přiřazeni jiným projektům; údržba je přenechána jinému týmu nebo jednotlivcům, kteří systém neznají => většina jejich úsilí musí být věnována pochopení existujícího systému

  - smlouva bývá obvykle pouze na dodání systému, o údržbě se nemluví => tým nemá motivaci vytvářet udržovatelný SW (zvláště pokud se předpokládá, že údržbu bude provádět někdo jiný)

  - údržba je obvykle přenechána méně zkušeným programátorům, navíc systémy mohou být vytvořeny ve starších programovacích jazycích, které se tým provádějící údržbu musí teprve naučit

- změnami se snižuje strukturovanost kódu - tím roste "entropie" SW
- dokumentace starých systémů může být ztracena nebo může být nekonzistentní atd.

- v mnoha případech je jediným dostupným prvkem SW konfigurace zdrojový text -> proces údržby začíná (obtížným) procesem vyhodnocení kódu (ten obsahuje pouze implementaci, nikoli záměr) => obtížná interpretace

- pokud neexistují testy, není možné regresivní testování

-> celkově: produktivita klesá na 40:1 oproti vývoji (podle Boehma)

A co s tím?

- předpokládat vodopádový model - systém vyvinout, udržovat, dokud se údržba stane ekonomicky neúnosná, pak nahradit novým systémem

- nebo evoluce systému - předpokládat např. spirálový model - systém by měl být navržen tak, aby ho bylo možné přizpůsobovat novým požadavkům

Metriky udržovatelnosti systému

- počet požadavků na opravy chyb - pokud počet požadavků na opravy stoupá, může to znamenat, že do programu je procesem údržby vnášeno více chyb, než je jich odstraňováno údržbou, tj. indikuje snížení udržovatelnosti

- průměrná doba potřebná pro vyhodnocení dopadu změny - odráží rozsah (např. počet komponent), které jsou ovlivněny požadavkem na změnu, pokud naroste, udržovatelnost se snižuje

- průměrná doba implementace požadavku na změnu
- počet nevyřízených požadavků na změnu

A co ze špatně udržovatelným programem?

- konverze do moderního programovacího jazyka nebo jeho varianty

- zpětné inženýrství = analýza programu s cílem najít specifikaci a návrh SW (ze zdrojových textů, v horších případech ze spustitelného kódu)

- vylepšení struktury programu a zlepšení jeho srozumitelnosti (automatické nástroje)

- modularizace programu = sdružení souvisejících částí programu, v některých případech včetně transformace architektury SW

Martin Fowler "Refactoring: Zlepšení existujícího kódu. Grada, Praha 2003."

Metriky

- jakékoli měření atributů SW produktu nebo SW procesu

- nejdůležitější metriky a oblasti využití:
  - metriky analytického modelu (např. kvalita specifikace)
  - metriky návrhu (např. složitost komponent)
  - metriky zdrojových textů (např. počet řádků kódu a cyklomatická složitost)
  - metriky pro testování (např. pokrytí logiky programu, efektivita testování)

Počet řádek kódu (Lines of Code, LOC)

- co máme počítat jako řádek kódu?
- Nepočítáme komentáře, prázdné řádky a automaticky generovaný kód - někdy se nazývá NCLOC (Non-comment LOC) nebo ELOC (Effective LOC) - základ metodiky COCOMO pro odhad ceny SW

McCabeho metrika "cyklomatická složitost"

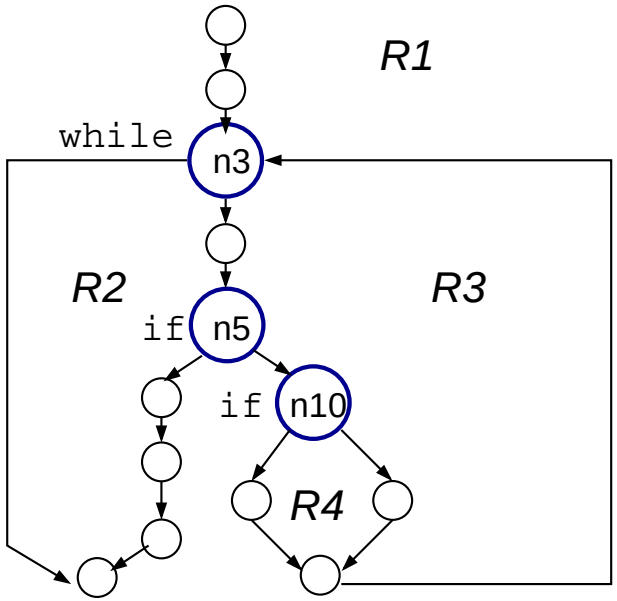
- počítá rozhodovací body v podprogramu
- vysoká cyklomatická složitost má korelaci s chybovostí, s obtížností číst a testovat podprogram (prostá délka podprogramu nemá korelaci s chybovostí)

- počítání cyklomatické složitosti:

1. Pro přímou cestu podprogramem započteme 1.
2. Pro každé z následujících klíčových slov nebo jejich ekvivalentů přičteme jedničku: if, while, do-while, for, a pro "and a "or" ve složených podmínkách.
3. Pro každý případ ve switch/case přičteme 1.

- pokud je >10, je příznak, že je vhodné podprogram zjednodušit (např. část podprogramu vložit do dalšího podprogramu volaného z původního)

- pokud v rozhodovacích příkazech nejsou složené podmínky, odpovídá cyklomatická složitost počtu regionů v grafu toku řízení podprogramu (region = oblast ohraničená hranami a uzly grafu; oblast mimo graf se počítá jako samostatný region).



- viz obrázek: přímá cesta, while, if a if – cyklomatická složitost 4 , počet regionů také 4.

Další metriky

- velikost zdrojových textů
  - celkový počet řádek (LOC)
  - celkový počet komentářových řádek (CLOC)
  - počet deklarací dat
  - počet prázdných řádek

- produktivita
  - E-faktor (environmental factor) = počet nepřerušených hodin/celková pracovní doba; optimální je cca 0.4
  - pracovní doba strávená na projektu
  - pracovní doba strávená na každém podprogramu
  - počet změn v podprogramu
  - náklady projektu
  - náklady projektu na řádek kódu
  - náklady na opravu defektu

- sledování defektů
    - vážnost defektu
    - místo defektu
    - způsob opravy defektu
    - osoba zodpovědná za defekt
    - počet řádek změněných při opravě defektu
    - pracovní doba strávená opravou defektu
    - průměrná doba potřebná na nalezení defektu
    - průměrná doba potřebná na opravu defektu
    - počet pokusů opravit defekt
    - počet nových chyb zanesených opravou defektu

- celková kvalita
- udržovatelnost
- objektově orientované metriky ad.

Konfigurační management

- sw projekty - změna požadavků na systém, design systému, kód, dokumentaci systému atd., tj. ve všech fázích tvorby SW

- v průběhu času vědí všichni zúčastnění více (o tom co potřebují, jak by se to nejlépe udělalo, atd.)

- pro řízení změn v projektech byly vyvinuty procesy, nazývané souhrnně konfigurační management (angl. software configuration management, SCM)

- úkolem SCM je definovat procedury pro provádění změn, eliminace některých problémů vznikajících zejména pokud je mnoho vývojářů a mnoho verzí SW

- úspěšný SW => tisíce požadavků na opravy a vylepšení

  - kód ve sdíleném adresáři - co když dva programátoři provádějí změnu ve
  - stejném modulu?
  - oprava chyby v produkční verzi, měla by se promítnout zároveň ve vývojové verzi, do které jsou ale mezitím přidávány další vlastnosti
  - vylepšení zavleklo chyby - jak se vrátit ke staré verzi?
  - jak zjistit, z čeho se která verze skládá?
  - obvykle kombinace těchto + dalších problémů

-> pro úspěch projektu podstatná schopnost řídit změny tak, aby si systém mohl zachovat integritu v čase

Tradiční konfigurační management

- založený na vodopádovém modelu – sw předán SCM týmu po dokončení vývoje a po otestování jednotlivých komponent
- SCM tým přebírá odpovědnost za sestavení úplného systému a za vedení testů
- chyby objevené při testu systému jsou předány zpět k opravě vývojovému týmu
- tento přístup ovlivnil tvorbu standardů; např. IEEE Std. 828 nevysloveně předpokládá vodopádový model, tj. obtížně se přizpůsobují např. inkrementálnímu vývoji
- definuje 4 procedury pro SW projekt
    - identifikace konfigurace
    - řízení konfigurace
    - vytváření záznamů o stavu konfigurace
    - autentizace konfigurace

Konfigurační management pro inkrementální vývoj

- příklad procesu
    - celý systém se sestavuje často (např. denně)
    - organizace určí čas, do kdy musí vývojáři doručit své komponenty (např. 16h)
    - komponenty mohou být neúplné, ale musejí poskytovat základní funkčnost, která může být otestována

- z komponent se sestaví nová verze systému
    - systém je předán testovacímu týmu, který provede předdefinované testy systému
    - vývojáři zatím dále pracují na svých komponentách, přidávají funkčnost a opravují chyby objevené v předchozích testech
    - testovací tým zdokumentuje objevené chyby, předá je vývojářům – vývojáři chybu opraví v další verzi komponenty

- brzo se najdou problémy vzniklé interakcí komponent
- vývojáři pociťují tlak, aby jejich komponenty nezpůsobily havárii systému - důsledkem je lepší testování jednotek

- SCM proces musí někdo řídit, ustanovit podrobné procedury, zajistit, aby všechny změny proběhly správně

Verzování

- udržení přehledu o podobách konfiguračních jednotek
- verze popisuje stav konfiguračních jednotek, nebo postup jeho změn

- extenzionální verzování: každá verze má jednoznačné ID (např. 1.5.1)
    - přístup v často používaných nástrojích (rcs/cvs)
    - jednoduchá implementace
    - nepoužitelné při větším počtu verzí

- intenzionální verzování: verze je popsána souborem atributů např. OS=DOS and UmiPostscript=YES
    - nutné pro větší prostory verzí
    - potřeba vhodných nástrojů

- prostor verzí je často reprezentován grafem
    - uzly = verze, hrany = vazby mezi verzemi, nejčastěji relace následnictví
    - větvení (branch) - operace vytvoření větve (branch-off, split) a spojení (merge)

Nástroje pro verzování

- ruční verzování = dohody a konvence o značení verzí v názvech (dokument, soubor, adresář), baseline pomocí zálohy všech souborů v daném čase

- základní - správa verzí souborů
    - obvykle extenzionální verzování modulů
    - ukládání všech verzí v zapouzdřené úsporné formě
    - příklady nástrojů: rcs, cvs

- pokročilé - integrované do CASE
    - obvykle kombinace extenzionálního a intenzionálního verzování
    - automatická podpora pro ci/co prvků z repository do nástrojů
    - příklad nástroje: ClearCase, Adele

Revision Control System (RCS)

- správa verzí pro textové soubory; UNIX, Windows, DOS
- extenzionální stavové verzování komponent
- ukládá historii všech změn v textu souboru, informace o autorovi a času změn, textový popis změny zadaný uživatelem, další informace (stav prvku, symbolická jména)
- funkce
    - zamykání souborů, poskytování R/O kopií
    - symbolická jména revizí, návrat k předchozím verzím
    - možnost větvení

Concurrent Versioning Systém (CVS)

- nadstavba nad rcs => umí vše, co umí rcs
- optimistický přístup ke kontrole paralelního přístupu - místo zamkni-modifikuj-odemkni (RCS) pracuje systémem zkopíruj-modifikuj-sluč
- práce s celými konfiguracemi (projekty) najednou
- sdílené repository + soukromé pracovní prostory
- možnost definovat obsah a strukturu konfigurace
- zjišťování stavu prvků, rozdílů oproti repository
- příkazová řádka, grafické nadstavby (UNIX, Windows, web)

- rcs i cvs jsou volně šířené
- existují další podobné nástroje, např. PRCS, Subversion apod.

Automatizace překladu a sestavení projektu:

- make, ant

- make
    - pochází ze systémů UNIX, původně souvislost s jazykem C
    - účelem automatizace překladu a sestavení projektu, minimalizace času spotřebovaného vytvářením aktuálních verzí objektových a dalších "strojově vytvářených" souborů
    - spustitelný program a objektové soubory se typicky vytvářejí ze zdrojových textů
    - popis pravidel překladu v souboru "Makefile" (případně "makefile")
    - překlad příkazem "make" (viz dále KIV/PC)

- ant
    - automatizace překladu projektů v jazyce Java

Etické a právní aspekty tvorby SW

- profesionální organizace (ACM ,IEEE) definovaly "etické zásady SW inženýra" (CS Code of Ethics),
  http://www.acm.org/serving/se/code.htm

- některé základní zásady:
  - při vytváření SW máte možnost být někomu prospěšní nebo mu způsobit škodu (nebo dát možnost jiným, aby byli prospěšní nebo ublížili) – např.pokud jste zodpovědní za vývoj systému kritického pro bezpečnost lidí a čas vás tlačí - bylo by neetické prohlásit systém za otestovaný, pokud není
  - důvěrnost - měli byste respektovat důvěrnost informací o klientech nebo zaměstnavateli
  - způsobilost - měli byste si být vědomi své úrovně a nepřijímat vědomě práci, která je nad vaše schopnosti

- dodržovat autorská práva, patenty apod. - porušováním můžete uvést do potíží nejen sebe
- nezneužívat cizí počítače např. pro provozování programů, se kterými by vlastník nesouhlasil

Autorský zákon

- 121/2000 Sb. "Zákon o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů"

- z § 2 vyplývá, že počítačový program ("je-li původní v tom smyslu, že je autorovým vlastním duševním výtvorem") i jeho jednotlivé vývojové fáze a části jsou předmětem ochrany podle AZ; podle § 65 je chráněn jako dílo literární

- § 5, § 11 a § 26: autorem je fyzická osoba, která dílo vytvořila, autorství nelze převést nebo se ho vzdát

- § 12 a § 46: autor má právo své dílo užít a udělit jiné osobě smlouvou licenci k jednotlivým způsobům nebo ke všem způsobům užití

- § 49: licence může být výhradní nebo nevýhradní (výhradní = autor nesmí
- poskytnout licenci třetí osobě)

- § 49: povinnou náležitostí licence je výše odměny nebo způsob jejího určení

- § 50: není-li v licenci řečeno jinak, platí pouze na území České republiky

- § 50: není-li určeno jinak, platí max. jeden rok

- § 58: zaměstnavatel vykonává svým jménem a na svůj účet autorova majetková práva k dílu, které autor vytvořil ke splnění svých povinností k zaměstnavateli (není-li sjednáno jinak)

- není-li sjednáno jinak, zaměstnavatel může dílo zveřejnit pod svým jménem, upravovat atd.

- počítačové programy se považují za zaměstnanecká díla i tehdy, byla-li vytvořena na objednávku

- licence platná v právním řádu jiné země nemusí být platnou licencí podle českého AZ a naopak (zejména pokud v licenci chybí některé ze zákona povinné ustanovení)

- podle českého AZ vzniká právo autorské k dílu v okamžiku, "kdy je dílo vyjádřeno v jakékoli objektivně vnímatelné podobě" (naproti tomu v jiných jurisdikcích je požadováno uvést informaci o copyrightu ve tvaru: Copyright <rok zveřejnění> <držitel autorských práv>)

- příklad licence platné v ČR:

Copyright 2004 Západočeská univerzita v Plzni

Západočeská univerzita v Plzni tímto poskytuje nabyvateli rozmnoženiny tohoto počítačového programu a jeho dokumentace (dále jen "Software") bezúplatně celosvětovou a časově neomezenou nevýhradní licenci ke všem způsobům užití Software včetně zejména práva Software rozmnožovat a rozšiřovat, s právem upravovat Software a měnit jeho název, spojovat Software s jinými díly a zařazovat Software do děl souborných.