Text k 5. přednášce do IVS 2021/2022 na téma

Typy dokumentace, generování programové dokumentace z kódu, identifikace existujících komponent a využívání knihoven dostupných na různých platformách

Obsah

- V této přednášce se nejprve podíváme na jednotlivé typy dokumentace. Pak se zaměříme na programovou dokumentaci, kde od dobrých praktik při komentování kódu přejdeme k tomu, jak z komentářů vygenerovat dokumentaci. Blíže se podíváme na nástroj Doxygen, po čemž bude následovat rychlá ukázka, jak jej nakonfigurovat a použít.
- Ve druhé části přednášky se budeme zabývat identifikací existujících komponent, které můžeme využít, abychom „znovu neobjevovali kolo“ a neplýtvali lidskými zdroji ani financemi. Blíže se podíváme na licence k využití cizího kódu, protože jejich porušení může vést k nepříjemným postihům.

Dokumentace

Typy dokumentace

- Uživatelská příručka popisuje základní použití programu. Je určena pro uživatele, což může být sekretářka, která absolvovala třeba zemědělskou univerzitu. Pokud jí program vypíše chybové hlášení, že pole pro číslo musí vyplnit, „protože hodnota jeho parametru required je nastavena na true“, nebude vědět, co má dělat. Tento příklad jsem si nevymyslel, ale pochází z jednoho staršího programu od Ministerstva kultury. Pokud se takovéto texty vyskytnou i v uživatelské příručce, je to špatné. Také nemůžeme předpokládat, že uživatel již obdobné programy používal a lehce se zorientuje – program ve skutečnosti může nahrazovat listinnou formu řešení, nějaké mechanické zařízení apod.
- Pokud bude mít uživatelská příručka 800 stran, v rámci pracovní doby ji není možné přečíst. Obvykle se jí každý lekne a ani ji neotevře a když bude tištěná, ještě bude zavazet, dokud neskončí někde v archivu. Měla by být stručná, výstižná a ideálně doplněná vhodnými obrázky. Uživateli ji dodáme ve formě malé brožurky nebo PDF.
- Referenční manuál již nemůže být stručný, protože jeho cílem je popsat všechny detaily programu z hlediska uživatele a správce, který má program instalovat a konfigurovat. V dnešní době už se typicky netiskne, ale dodává se v PDF nebo v podobě webových stránek. Uživatel ani správce však obvykle nepotřebují vědět, jak je program napsaný. To je často součástí firemního tajemství, které by se nemělo zveřejňovat.
- Co by však v referenčním manuálu mělo být, jsou závislosti na externích knihovnách a programech, včetně uvedení konkrétních verzí (často se uvádí konkrétní otestovaná verze doplněná textem „a vyšší“). Pokud pak něco nefunguje, lze nahlédnout do referenčního manuálu a zjistit, co je třeba doinstalovat. U programů s otevřenými zdrojovými texty je vhodné uvádět i verzi využitého překladače, protože pro novější verzi mohou být nutné úpravy.
- Programová dokumentace slouží pro vývojáře, kteří budou program udržovat či vylepšovat. Ti potřebují vědět, k čemu slouží která část programového kódu či ve které části je implementovaná daná funkce.
- Když už jsou v programovém kódu komentáře, manuální psaní programové dokumentace by znamenalo vypisovat vybrané části komentářů. Obvykle navíc nemá smysl dokumentovat každý řádek kódu, ale stačí vybírat komentáře větších bloků, funkcí nebo dokonce celých souborů. Takový výběr komentářů lze provést i automaticky a dokumentaci nechat vygenerovat strojově.
- Poslední typ dokumentace, co zde zmíním, je projektová dokumentace. Ta popisuje proces tvorby programu. Typicky začíná analýzou požadavků, teoretickými základy, návrhem programu (např. v UML), popisem postupu implementace, testování apod. Příkladem může být technická zpráva k bakalářské práci.

Uživatelská příručka

- Dnes je na webu velká spousta programů, ale zdaleka ne všechny umí přesně to, co člověk zrovna potřebuje.
- Instalace bývá často jednoduchá, ale čistá odinstalace není samozřejmostí – zůstávají klíče v registrech, konfigurační soubory . . . V horším případě v PC po programu zůstane virus.

- Některé programy jsou k dispozici zdarma, některé lze zdarma vyzkoušet a některé člověk bez zaplacení ani nevyzkouší.
- Typicky nechceme instalovat (a odinstalovávat) několik desítek programů, než se nám podaří najít ten správný. Potřebujeme tedy něco, podle čeho zjistíme, jestli je program pro nás vhodný, dříve, než si ho nainstalujeme. A právě v tom nám pomůže uživatelská příručka.
- Uživatelská příručka by měla obsahovat:
  - Závislosti programu a HW nároky.
  - Návod k instalaci programu.
  - Stručný popis použití doplněný vhodnými snímky obrazovek.
  - Informace o autorech aplikace.
- Uživatelská příručka by neměla obsahovat:
  - Zadání – to bylo součástí specifikace.
  - Teoretické základy využité k tvorbě programu – ty patří do projektové dokumentace.
  - Informace o implementaci – ty patří do programové dokumentace.
  - Informace o tom, co se nepodařilo – na to zákazníka nenalákáme.
- Kde skladovat uživatelské příručky ke všem programům, které máme nainstalované, tak, abychom je snadno našli? Složka plná PDF asi nebude to pravé ... Proto by program měl poskytovat nápovědu.

Nápověda

- Nápověda je vlastně uživatelskou příručkou zabudovanou přímo do programu, abychom ji při použití měli vždy snadno a rychle k dispozici.
- Nápověda může být nejen ve formě klasické textové nápovědy, která se otevře v samostatném okně přímo v programu, v externím prohlížeči ve formátu PDF či HTML nebo třeba online ve webovém prohlížeči, ale i ve formě kontextové nápovědy, která se zobrazí při najetí myší na jednotlivé ovládací prvky programu apod.
- Na rozdíl od uživatelské příručky v nápovědě nemáme popis instalace programu, protože dokud program není nainstalovaný, nápovědu si v něm nezobrazíme, a až už je nainstalovaný, je na studování postupu instalace pozdě.
- Při tvorbě uživatelské příručky můžeme recyklovat nápovědu, nebo naopak vytvořit nápovědu úpravou uživatelské příručky.

Doxygen

- Nyní se budeme zabývat programovou dokumentací, kterou lze vygenerovat ze zdrojových textů programu. K tomuto účelu můžeme využít Doxygen. Je to univerzální zdarma dostupný nástroj, který podporuje celou řadu programovacích jazyků a podpora dalších přibývá i ve formě různých vstupních filtrů, které některé konstrukce jednoho jazyka převedou do jiného podobného jazyka tak, aby se správně vyextrahovaly komentáře.
- Výstupem je dokumentace, která může být v jednom či více z podporovaných formátů. Nejčastěji se využívá HTML nebo PDF.
- Pokud si nainstalujete i nástroj dot z balíku Graphviz, Doxygen může s jeho využitím vygenerovat grafy závislostí, volání funkcí apod. (nemáme-li návrhovou dokumentaci, částečně si ji dogenerujeme z kódu).

Další nástroje

- Existuje i celá řada dalších nástrojů pro generování dokumentace.
- Mezi nejčastěji využívané patří např. Javadoc, který je de facto standardní nástroj pro jazyk Java. Drobnou nevýhodou je, že podporuje pouze angličtinu a japonštinu – pro jiné jazyky nezbývá než si doplnit lokalizační soubor (cca 300 řádků). Nicméně pokud napíšeme komentáře pro Javadoc, lze je zpracovat i Doxygenem. Naopak to však neplatí zcela – ne všechny komentáře pro Doxygen zpracuje i Javadoc.
- Pro jazyk PHP můžeme využít phpDocumentor, pro Python PyDoc, pro JavaScript JSDoc apod. Tyto nástroje jsou specializované na konkrétní jazyky a existuje jich celá řada.

- Univerzálnější je ROBODoc, ale jeho nevýhodou je formát komentářů nekompatibilní s ostatními nástroji.
- Těmito nástroji se zde nebudeme zabývat. Pro zbytek přednášky jsme zvolili Doxygen, protože je nejuniverzálnější a když někdo zvládne využít Doxygen, velmi pravděpodobně během krátké doby zvládne i libovolný podobný nástroj, který je specializovaný na konkrétní jazyk. Základní prvky komentářů jsou ve většině případů stejné či podobné.

Komentáře

- Jak jsem již říkal, dokumentace se generuje především z komentářů. Pokud bychom měli kód bez komentářů, vygenerovaly by se sice seznamy funkcí a nějaké grafy, žádná rozumná dokumentace by však nevznikla.
- Navíc je třeba vzít v úvahu to, že počítač sice pozná, že komentář je v daném souboru, ale ne vždy je schopen rozpoznat, k čemu patří a zda je pro dokumentaci důležitý. Pokud komentář umístíme několik řádků nad funkci, už není jasné, zda patří k této funkci, nebo jestli je to jenom nějaká poznámka, např. k celému souboru, popis chybějící funkcionality apod. Počítač navíc v bloku textu nerozpozná, co je popis funkce, co je popis jejích parametrů apod. Je tedy potřeba komentáře upravit tak, aby byly pro strojovou analýzu vhodné. Protože pro komentáře není dostupný všeobecný standard, musíme se vždy držet pravidel pro generátor dokumentace, který chceme využít – v našem případě Doxygen.

Komentáře

- Než se však začneme zabývat přesnou formou komentářů pro Doxygen, je vhodné říci něco málo o dobrých zvycích při komentování. Velké firmy či projekty mají obvykle stanovena svoje striktní pravidla pro psaní komentářů, kde je řečeno, jak co komentovat a o kolik mezer odsazovat. Já zde tolik do detailů nepůjdu, ale řeknu pouze několik obecných pravidel, která je vždy dobré dodržovat.
- Jedním ze základních pravidel je, že komentářů by mělo být raději více, než méně. Některé konstrukce, které se mohou zdát jasné, jiný vývojář může chápat jenom obtížně. Nemá však smysl psát až příliš mnoho komentářů, protože kód potom může být nepřehledný.
- Komentář by měl mít vždy přínos k pochopení řešeného problému. Pokud se na řádku přičítá jednička, můžeme okomentovat k čemu a proč. Pokud bychom však napsali pouze že se jedná o přičtení jedničky, je to k ničemu a spíše to znepřehledňuje.
- Komentování každého řádku kódu by bylo zbytečné, ale celý soubor je třeba komentovat vždy a to jak hlavičkou, která říká, co obsahuje, tak i zápatím, které umožňuje zjistit, že je soubor celý. Komentovat je třeba i každou třídu v objektově orientovaném programování, proceduru, funkci apod.
- Názvy proměnných by měly být samopopisné, což znamená, že z názvu ihned poznáme, k čemu slouží. Příliš dlouhé názvy však znepřehledňují kód a u pomocných lokálních proměnných a počítadel jsou často zbytečné. Pokud však využijeme proměnné s nejasnými názvy, je třeba jejich deklaraci či první výskyt komentovat.
- Komentovat bychom měli také všechno, co není zcela jasně zřejmé a pochopitelné z názvů volaných funkcí, využitých proměnných apod.

Komentáře

- Pokud využíváme vnořené závorky, které ohraničují velké bloky kódu, pouhé odsazení pro přehlednost nestačí. Je-li otevírací závorka o 3 obrazovky výš a nad ní je obdobná struktura ještě několikrát, nejen že může být zdlouhavé hledání otevírací závorky, kdy dlouze jedeme kursorem, ale můžeme i přejet a udělat zbytečnou chybu, kdy např. připíšeme řádek kódu do jiného bloku či funkce. Takové závorkové struktury, jako např. tato (viz slajd) je tedy vhodné komentovat, a to buď zkrácenou kopií řádku, který závorku otevírá (např. název funkce či podmínka cyklu), nebo komentářem k tomuto řádku.
- Rád bych zdůraznil ono zkrácení při využití kopie řádku, který závorku otevírá. Zejména u funkcí zde opravdu nemá smysl ponechávat návratový typ a parametry – stačí název funkce a závorky, aby bylo jasné, že se jedná o konec funkce (více informací opět zbytečně znepřehledňuje).

Komentáře pro Doxygen

- Jak již bylo řečeno na začátku, pro generátor dokumentace je nutné komentáře upravit, resp. psát je už od začátku tak, abychom jej potom mohli využít. Protože v kódu máme celou řadu komentářů, nejprve je třeba určit, které komentáře má Doxygen extrahovat. Tyto jsou určeny tím, že jsou umístěny v tzv. dokumentačních blocích. Podporovaných syntaxí bloku je více, jednou z nejvyužívanějších je varianta pro C a Javu.

- Uvnitř bloku potom můžeme využívat různé příkazy pro Doxygen, které vysvětlím dále.
- Abychom u deklarací proměnných nemuseli využívat zdlouhavé komentářové bloky, existuje k tomuto účelu i jednořádková zkrácená varianta komentáře.
- Tato jednořádková varianta je stále blokový (tedy nikoliv řádkový) komentář.
- Častou chybou studentů je nesprávné použití tohoto typu komentáře pro komentování něčeho, co do generované dokumentace nemá smysl dávat (např. komentář k pomocné proměnné použité pouze na několika následujících řádcích uvnitř funkce).

Komentování souborů (1/2)

- Každý soubor by měl mít komentář, ze kterého bude patrné k jakému projektu patří, kdo jej vytvořil, co obsahuje, o jakou verzi se jedná a kdy byl naposledy změněn. Některé z těchto informací Doxygen získá z konfigurace a bylo by tedy zbytečné a nepřehledné je do dokumentace umist'ovat vícekrát. Proto můžeme hlavičku rozdělit na 2 části, kde 1. není korektní dokumentační blok a Doxygen ji tedy ignoruje. Druhou část hlavičky již upravíme pro Doxygen. Příkazem `@file` řekneme, že daný blok komentáře patří k celému souboru. `@brief` je potom stručný komentář. Pokud by byl potřeba i delší komentář, uvedli bychom jej volně dovnitř dokumentačního bloku. Můžeme uvést i další příkazy, pomocí kterých uvedeme autory a další informace relevantní k souboru.
- Proč vlastně vkládat 1. část hlavičky, když ji Doxygen ignoruje? Motivačním příkladem, k čemu lze tuto hlavičku využít, může být případ z praxe, kdy došlo k poškození souborového systému diskového pole. Miliony souborů sice zůstaly nepoškozené, ale adresářová struktura byla značně poškozená a vše tak bylo pouze v jediném adresáři a často pojmenované pouze číslem. Zdrojové soubory s dobře napsanými hlavičkami bylo možné automatizovaně přejmenovat, vytřídit a obnovit složky různých projektů. Pro zbytek souborů pak bylo možné pouze manuálně hledat podle něčeho uvnitř (po paměti, podle obsahů již nalezených souborů apod. – tedy místo 5 min. třeba 3 dny práce a to pouze pro jediný z desítek programů, co zde byly uloženy). Často se jako jednodušší a levnější ukázalo vytvořit kód znovu než jej najít. Investice do hlaviček by se při této události mnohonásobně vyplatila.
- Co se týče rozsahu informací v hlavičce, záleží i na tom, co lze snadno zjistit i bez ní – např. pokud používáme GIT, datum vytvoření a poslední změny snadno zjistíme z něj a když bude i v hlavičce, typicky jej budeme zapomínat aktualizovat. Pokud programujeme v jazyce Java a vhodně pojmenujeme balíčky s využitím doporučených konvencí (např. `package cz.vutbr.fit.knot.annotations.entity.attribute;`), nepotřebujeme název projektu ani balíčku, protože nám pro jednoznačné určení stačí přečíst 1 řádek kódu. Takto můžeme dospět k tomu, že stačí hlavička pro Doxygen se stručným popisem obsahu souboru.

Komentování souborů (2/2)

- Konec souboru je rovněž vhodné označit komentářem, aby bylo zřejmé, že je soubor kompletní a nedošlo k nějakému poškození při přenosu apod.
- Toto samozřejmě neděláme tam, kde je např. 1 třída a konec souboru je jasně daný uzavírací závorkou (kterou pravděpodobně komentovat budeme).

Komentování funkcí

- Komentáře k funkcím či metodám jsou často ze všech nejdůležitější.
- Funkce komentujeme dokumentačním blokem nad jejich deklarací. Vždy je třeba komentovat, k čemu funkce slouží, její parametry a návratovou hodnotu. Volitelně můžeme uvést i autora a další informace, mezi které patří např. zjištěné neopravené chyby a nedořešené problémy. Doxygen nám potom umožňuje generovat seznam chyb a seznam úkolů k řešení.
- Podívejme se detailněji na jednotlivé části tohoto komentáře, protože zde studenti velmi často chybují.
  - Za příkazem `@param` následuje mezera, název parametru, mezera a popis. Datový typ může generátor přečíst z kódu, takže se neuvádí. Když např. místo „`radek`“ uvedete „`int radek`“, bude v dokumentaci jako název parametru datový typ, což je zjevně špatně. Jako oddělovače se využívají mezery, takže různé dvojtečky, pomlčky apod. tam nemají co dělat.
  - Pozor je potřeba dávat i na ukazatele, kde je typický zápis např.: „`char *retezec`“, přičemž „`char *`“ je datový typ (ukazatel na znak) a „`retezec`“ je název proměnné.

- Příkaz `@return` slouží ke komentování návratové hodnoty. Jak jistě všichni víte, tato hodnota se předává přes zásobník. Název lokální proměnné, kterou jste využili pro výpočet a dočasné uložení této hodnoty, za příkaz `@return` nepatří.
- Komentář `@bug` vznikne, když vývojář (při ladění) objeví chybu, ale její oprava není triviální a nemá čas ji opravit hned. Označí místo, kde chyba je, a umožní tak její snadné nalezení ve chvíli, kdy se bude řešit.
- `@todo` se využívá pro vyznačení místa, kde něco není dokončené. Nejde o chybu. V souboru má být např. 10 funkcí, ale my jich stihneme implementovat jen 9 a pak jdeme dělat něco jiného (nebo končí pracovní doba). Aby se na tu 10. nezapomnělo, připíšeme tam před commitem komentář. Často se používají tam, kde se neošetří nějaké chybové stavy apod., protože nejprve rychle vytváříme prototyp a pak potřebujeme vše doladit, abychom to mohli předat zákazníkovi.

Další příkazy

- Doxygen podporuje i celou řadu dalších příkazů pro komentování prvků objektově orientovaného programování, vytváření odkazů v rámci dokumentace apod. Tyto si můžete podle potřeby najít v manuálu.
- Za zmínku možná stojí příkaz `@package`, protože ním komentujeme několik souborů, ale uvádí se pouze v jednom z nich – typicky jej umístíme do souboru se vstupním bodem, rozhraním apod., ale někdy prostě jen do 1. souboru, který v balíčku vytvoříme.

Generování dokumentace

- Doxygen je konzolová aplikace, což znamená, že se spouští z příkazové řádky. Nejprve vytvoříme tzv. Doxyfile, což je soubor s konfigurací, a následně program spustíme jednoduchým příkazem. Pokud se zdrojový kód programu změní, vygenerování nové verze dokumentace lze snadno provést opakovaným spuštěním.
- Pokud soubor nazveme přímo „Doxyfile“, název souboru při spuštění nemusíme uvádět.

Doxyfile

- Šablonu konfiguračního souboru lze vygenerovat příkazem `doxygen -g` a následně ji upravit pro svůj projekt.
- K vytvoření konfiguračního souboru existuje i nástroj s GUR (grafickým uživatelským rozhraním). V tomto nástroji si konfiguraci můžete naklikat, uložit výsledný Doxyfile i spustit Doxygen.
- Starší verze tohoto nástroje neměly všechny volby do detailů, čímž byla negativně ovlivněna kvalita vygenerované dokumentace.
- V aktuální verzi lze využít průvodce se základními volbami a následně doladit nastavení v záložce Expert. Dokumentace jednotlivých položek se v této záložce zobrazuje při najetí myší.
- Oproti manuální exitaci Doxyfile je tento postup pomalejší, protože popisy konfiguračních voleb nevidíte hned, ale musíte jezdit myší a nahlížet. Pokud provedete jen základní nastavení a neprojdete položky v záložce „expert“, některé výchozí volby způsobí, že vygenerovaná dokumentace nebude nejlepší.

Doxywizard

- Průvodce základními volbami nastaví pouze malé množství voleb a některé výchozí volby nejsou příliš dobré. Po využití tohoto průvodce bez následného doladění je kvalita dokumentace znatelně nižší.

Doxywizard

- Nevýhodou je, že oproti procházení Doxyfile, kde jsou komentáře ihned nad položkami a lze rychle procházet celým souborem, je zde nutné jezdit myší, aby se nám komentáře postupně po jednom zobrazovaly, a přepínat sekce. Výhodou je, že soubory lze vybírat pomocí systémového dialogu a neuděláme zde syntaktickou chybu v Doxyfile.

Sekce Doxyfile

- V šabloně souboru vygenerované Doxygenem je obsažena spousta komentářů, které Vám pomohou se v něm zorientovat. Je rozdělen na řadu sekcí, které lze rozdělit do 3 skupin: základní nastavení, nastavení výstupu a další sekce, což bychom mohli nazvat i „ostatní“.
- Mezi další sekce patří např. volby spojené s preprocesorem, což je předzpracování pro programovací jazyky, které nejsou přímo podporovány.

Základní nastavení (1/3)

- Než začneme procházet jednotlivé sekce se základními nastaveními, zmíním nastavení kódování, které je velice důležité. Nastavit lze kódování zdrojových textů programu i samotného Doxyfile. U zdrojových textů se nejedná jenom o možnost využití diakritiky v komentářích, ale i o textové řetězce vypisované programem a převzaté do dokumentace v rámci možnosti procházení zdrojovým textem.
- A nyní již volby spojené s projektem. Mezi ty základní patří název projektu a verze, která se skrývá pod označením `PROJECT_NUMBER`.
- Nastavit je potřeba také umístění pro generovanou dokumentaci a jazyk komentářů, abychom v dokumentaci neměli např. „class obrazek“ ale „třída obrazek“ apod.
- U jazyka se na chvíli zastavím, protože mezi časté dotazy studentů patří, v jakém jazyce komentáře a identifikátory (názvy proměnných, funkcí apod.) psát.
- Plně české zdrojové texty omezují možnost znovupoužití cizími programátory a obecně nejsou doporučovány ani preferovány. Lze je využít spíše u tvorby pro vlastní potřebu, u které jste si jistí, že je nikdy nebudete chtít sdílet s programátory z ciziny, nebo u aplikací, které nikdy (lze-li tohle slovo vůbec říci jistě) nebudou spravovat cizinci (např. česká webová aplikace pro malé zahradnictví XY). Při cvičných školních projektech je to také často tolerováno a v programech určených pro výuku, kde studentům ukazujete zdrojové texty, to může být i žádoucí, protože lépe pochopitelné jsou texty v mateřském jazyce.
- Pokud se nejedná o mezinárodní projekt, neumíte dobře anglicky, a čeština není explicitně zakázaná, můžete psát identifikátory v angličtině a komentáře v češtině. Cizinec to pak může číst jako zdrojový text bez komentářů, což je při vhodně zvolených názvech funkcí a proměnných často možné, a tam, kde se v kódu nezorientuje, může využít překladač.

- Když už komentáře píšete česky, využívejte diakritiku – psaní bez diakritiky omezuje možnosti automatického překladu textu a v dnešní době, kdy typicky využíváme kódování UTF-8, nemá žádnou výhodu.
- Ve firemní praxi to však často nemusí být tolerováno (sice v některé ryze české firmě může být, ale slyšel jsem několik historek typu „ryze česká firma – tak si uděláme přijímací pohovor v angličtině“, takže bych na to nespoléhal) a musíte psát komentáře i identifikátory anglicky. Česky jsou pak jen texty, které může vidět uživatel, a ty jsou obvykle umístěny v samostatném souboru, který bývá hlavní součástí lokalizačního balíčku.
- Můžeme zde povolit i optimalizaci pro některý programovací jazyk apod.

Základní nastavení (2/3)

- Konfigurační volby spojené se sestavením (dokumentace) umožňují zvolit, které programové konstrukce chceme dokumentovat a zda chceme generovat i seznam úkolů, chyb apod.
- Nastavení varování a zpráv o zpracování umožňuje vygenerovat s dokumentací i soubor, ve kterém budou obsaženy problémy zjištěné při generování. Tento bychom měli využívat, abychom např. vlivem překlepu neměli v dokumentaci chybu nebo v ní něco nechybělo. Odhalí však pouze omezené množství chyb, takže dokumentaci je vhodné vždy alespoň přehledově zkontrolovat.
- Pokud soubor není prázdný, v dokumentaci pravděpodobně bude chyba. Pokud je prázdný, neznamená to, že tam žádná chyba není (často tam chyby jsou).
- Typickou chybou jsou chybějící stručné komentáře k souborům, třídám, balíčkům apod. – v seznamech pak chybí položky v pravém sloupci. Nejčastěji je problém u jazyků, kde je v 1 souboru 1 třída a vývojář si neuvědomí, že tam mají být 3–4 komentáře (1. pro celý soubor, 2. pro celý soubor pro Doxygen, 3. pro třídu v tom souboru a případně i 4. pro balíček, který se dává do 1 souboru v balíčku).

Základní nastavení (3/3)

- Mezi poslední důležitou část základních nastavení patří nastavení vstupu, kde zvolíme zkoumané přípony souborů, jejich umístění, výjimky apod.
- Výchozí nastavení může zahrnovat spoustu přípon na pokrytí různých programovacích jazyků. Když však chceme dokumentovat program v jazyce C, nemá smysl, aby se zkoumalo 20 přípon. Pak se na to zapomene, generátor dokumentace omylem přečte nějaké pomocné datové soubory a máme problém.

Nastavení výstupu

- Druhou skupinou nastavení je nastavení výstupu. Některé volby jsou zde společné pro různé výstupní formáty. Mezi tyto volby patří např. možnost umístit zdrojové kódy se zvýrazněnou syntaxí přímo do dokumentace, takže se od popisu funkce můžete proklikat až k jejímu obsahu.
- Doxygen může vygenerovat i abecední rejstřík tříd apod.
- Pro jednotlivé výstupní formáty jsou v Doxyfile umístěny samostatné sekce, které umožňují povolit daný formát a případně nastavit některá jeho specifika.
  - Dokumentaci nemá smysl generovat ve spoustě formátů – typicky povolíme právě jednu sekci a vygenerujeme obvykle HTML (díky odkazům často přehlednější) nebo PDF.
- Máme-li nástroj dot, můžeme povolit a nastavit i generování grafů. Je však důležité zvolit správný formát výstupních obrázků, aby dokumentace neměla desítky MB.

Ukázka výstupu (jednotlivé slajdy)

- Nyní zde mám ukázku vygenerované dokumentace. Záměrně jsem vybral program, jehož komentáře jsou dobře napsané, ale mohly by být i lépe.
- Nyní se podívejme na hierarchii tříd. Seznam vypadá pěkně, ale hodily by se stručné popisy jednotlivých tříd. Ty však bohužel v komentářích chybí.
- Pokud se podíváme na konkrétní třídu, vidíme seznam jejích metod. Stručné komentáře opět chybí, takže účel metody se dozvíme až když na ni klikneme a podíváme se na detaily.

- V detailním popisu už je vše v pořádku. Pokud by nám však uvedené informace nestačily, je zde ještě detailnější zobrazení. Můžeme totiž využít toho, že Doxygen umí udělat ze zdrojových textů součást dokumentace. Jistě Vám neuniklo, že zde chybí komentář pro celý soubor, protože ten, který zde je, není pro Doxygen.
- Nyní se vrat'me k detailnímu zobrazení informací o metodě. Jistě jste si všimli grafů volání, na kterých je názorně zobrazeno, které metody tato metoda volá či může volat a kterými metodami je volána.
- Kromě grafů volání Doxygen vygeneroval i graf hierarchie tříd, který však pro tento program a jazyk C++ není příliš dobrý.
- Nyní zde mám ještě obrázky z dokumentace lépe komentovaného programu, který je v jazyce Java.
- Na úvodní stránce nebylo vidět nic zvláštního, ale v seznamu tříd ihned vidíme, že jsou uvedeny i stručné popisy.
- Pro jednotlivé třídy zde máme vygenerovány i diagramy tříd. Tyto diagramy znáte z Úvodu do softwarového inženýrství, takže je nebudu více popisovat. Jak vidíte, nemáte-li návrhové dokumenty, můžete k získání diagramů pro neznámé zdrojové texty s výhodou využít Doxygen.

Více konfigurací

- V některých případech vytvoříme i více souborů Doxyfile a vygenerujeme více dokumentací s různou mírou detailu a pokrytí kódu dle účelu:
  - pro uživatele naší knihovny (zákazníky) vygenerujeme dokumentaci rozhraní,
  - pro vývojáře (zaměstnance) vygenerujeme detailní dokumentaci, ve které budou i privátní prvky, celé zdrojové texty v dokumentaci k rychlému zorientování, seznamy chyb a úkolů k dořešení apod.

Identifikace a využití existujících komponent

- V dnešní době málo kdy budeme programovat pouze se základními konstrukcemi zvoleného programovacího jazyka a několika standardními knihovnami v distribuci. K dispozici máme velké množství frameworků a knihoven, které můžeme využít. Nechceme-li „znovu objevovat kolo“ a chceme-li produkt vytvořit v rozumném čase, za rozumnou cenu a s adekvátní funkcionalitou, jejich volbě a využití se nevyhneme.
- Ve druhé části této přednášky se podíváme na to, jaké jsou typy znovupoužitelných komponent, co bychom při jejich volbě měli zohlednit a jak správně využívat cizí kód.

Dostupné komponenty

- K dispozici máme 3 různé typy komponent, které můžeme využít. Liší se v tom, jakým způsobem budeme naše řešení s jejich využitím programovat.
- Prvním typem jsou hotové produkty. Ty můžeme využít dvěma způsoby:
  - Vezmeme kompletní řešení, jako je např. webový prohlížeč, a vytvořením doplňku (add-on) či zásuvného modulu (plugin) jej rozšíříme o potřebnou funkcionalitu.
  - Hotové řešení využijeme jako součást našeho tak, že využíváme jeho služby přes API. Např. pro webovou aplikaci využijeme databázový server, nebo v rámci ní využijeme jinou webovou aplikaci jako např. mapy.
- Mimochodem rozdíl mezi doplňky a zásuvnými moduly není zcela jasný, ale v prostředí webových prohlížečů je zásuvný modul typicky větší a připojuje SW 3. strany přes nějaké API, zatímco doplněk je spíše menší a přidává typicky prvky uživatelského rozhraní, mění téma vzhledu apod.
- Druhým typem znovupoužitelných komponent jsou frameworky. Framework určuje chování aplikace, které programátor dle potřeby upravuje. Typicky se používají pro webové aplikace a aplikace s grafickým uživatelským rozhraním.
- U frameworku obvykle nadefinujeme obrazovky uživatelského rozhraní a co se má stát při jaké události v tomto rozhraní. Framework pak zajišt'uje běh aplikace a při příslušných událostech volá naše funkce.
- Posledním typem komponent jsou knihovny. Chování aplikace má plně pod kontrolou programátor a volá funkce z knihovny, které provádějí nějaké dílčí výpočty apod.

- Ne vždy lze jasně určit, zda se jedná o knihovnu nebo o framework. Někdy navíc můžeme z frameworku využít jen pomocnou funkcionalitu a volat jej pouze jako knihovnu.
- Při volbě komponent samozřejmě postupujeme shora. Pokud stačí jednoduše rozšířit existující řešení, obvykle nemá smysl programovat vlastní. Pokud můžeme využít framework, který nám práci výrazně usnadní, nemá smysl, abychom vše programovali sami. A všude, kde je potřeba funkcionalita, co už je k dispozici hotová a neřeší ji přímo využitý framework, použijeme knihovny.

Identifikace existujících komponent

- Při volbě konkrétních komponent musíme zohlednit celou řadu faktorů. Zde máme ty nejdůležitější.
- Hned na prvním místě máme to nejzásadnější – licenci. Licence udává jak, kde a za jakých podmínek můžeme řešení využít. To zahrnuje nejen zda a kolik za to musíme zaplatit, ale také zda si dané řešení můžeme přizpůsobit a jaká bude licence našeho výsledného produktu. U některých licencí nám vzniká povinnost svůj kód, který dané řešení využívá, zdarma zveřejnit. Volba řešení s nevhodnou licencí nebo porušení licenčních podmínek může mít velmi negativní až fatální následky.
- V době, kdy i kadeřnice musí platit za to, že si při své práci pustí rádio, je licenci skutečně potřeba věnovat náležitou pozornost – proto se k ní za chvíli znovu vrátíme.
- Dalším faktorem, který je třeba zohlednit, je výkon a využití zdrojů. Pokud bude mít aplikace tisíce uživatelů a využijeme pomalou knihovnu s velkou spotřebou paměti, uživatelé budou nespokojení s čekáním a může snadno dojít k vyčerpání systémových prostředků.
- Velmi důležitá je rovněž dostupnost a míra detailu dokumentace. Pokud si vybíráme mezi dobře zdokumentovanou knihovnou, která neposkytuje vše, co potřebujeme, a knihovnou, která toho sice umí hodně, ale dokumentace je nekvalitní, často raději sáhneme po té první, protože je jednodušší něco málo doprogramovat než u veškeré funkcionality experimentovat, abychom zjistili, jak ji správně využít.
- Zohlednit musíme rovněž přenositelnost. Musíme si dobře rozmyslet, na jaké zákazníky cílíme a jaký HW a SW používají. Před určitou dobou bylo téměř nemyslitelné, aby webová aplikace nefungovala v Internet Exploreru. Určitý zlom přišel po zveřejnění článku, jak startup ušetřil 100 tis. dolarů tím, že se rozhodl jej nepodporovat https://techcrunch.com/2012/04/01/bootstrapped-startup-saves-over-100k-by-dropping-ie/. A dnes už Microsoft vývoj Internet Exploreru opustil. Vždy tedy musíme zvážit, zda podporou další platformy vyděláme více než kolik nás ta podpora bude stát. Při volbě knihoven pak samozřejmě musí mít na všech zvolených platformách podporu i zvolená knihovna, nebo ji tam musíme být schopni nahradit.
- Pokud s knihovnou dlouhodobě pracujeme, nemusíme studovat dokumentaci a jsme schopní pracovat výrazně rychleji. Pokud využijeme knihovnu, kterou neznáme, musíme se s ní seznámit, což stojí nemálo času a úsilí. Může se tedy vyplatit použít známou knihovnu i za cenu, že má méně funkcí a něco si budeme muset sami doprogramovat.
- Knihovnu pochopitelně vybíráme pro zvolený programovací jazyk, případně tak, abychom ji s využitím nějakého rozhraní mohli na program ve zvoleném jazyce napojit.
- Kromě knihoven musíme vhodně zvolit i kompilátor či více kompilátorů pro různé platformy. Kompilátor může ovlivňovat i výkon výsledné aplikace.

Licence

- A nyní se dostáváme k tomu nejdůležitějšímu – k licencím. Komerčních licencí existuje velká spousta a vždy si je musíme důkladně prostudovat. Obdobně existuje celá řada licencí, které nám umožňují bezplatné využití, případně i úpravy poskytnutého řešení. Teď se podíváme na několik nejznámějších licencí pro produkty s otevřeným (zveřejněným) zdrojovým kódem (angl. open source).
- Licence může ovlivnit nejen to, jak můžeme využít daný komponent, ale i to, jaká bude licence našeho výsledného řešení, ve kterém daný komponent využijeme.
- Licence GPL (GNU General Public License) chrání svobodu SW. Pokud využijeme něco, co je distribuováno s licencí GPL, upravený kód musíme opět zveřejnit s licencí GPL a náš produkt musí být distribuován s licencí, která je kompatibilní s GPL. Svoji aplikaci již nemůžeme uzavřít a začít prodávat a nesmíme v ní využít komerční komponenty s nekompatibilní licencí.

- Licence LGPL (GNU Lesser General Public License) povoluje linkování s uzavřenou aplikací, ale úpravy knihovny musí být stále zveřejněny s licencí GPL a možnosti distribuce našeho řešení jsou omezené.
- Licence Apache a BSD pak umožňují daný komponent upravit a komercializovat. Kladou nároky pouze na to, aby zůstalo jasné autorství daných komponent.

Qt

- A nyní se podíváme na příklad konkrétního frameworku – Qt. Jedná se o framework v jazyce C++ pro tvorbu aplikací s grafickým uživatelským rozhraním. Má komponenty pro síťovou komunikaci, přístup k databázím apod.
- Lze jej využít i v jiných jazycích jako Python, Ruby apod. a je přenositlený na Linux, Mac OS X, MS Windows a další platformy.
- Má tzv. duální licenci – komerční a svobodnou. Když naše řešení zveřejníme s licencí GPL, přispíváme k vývoji frameworku a poskytujeme příklady použití – odměnou nám za to je, že jej můžeme využívat zdarma. Když jej chceme využít ke tvorbě uzavřené komerční aplikace, musíme zaplatit za komerční licenci.
- Výhodou frameworku Qt je to, že je modulární. Nemusíme tedy načítat velký balík se spoustou věcí, ale můžeme si vybrat a využít pouze některé moduly podle toho, zda chceme vytvořit jen základní uživatelské rozhraní, nebo zda chceme pracovat i se sítí, multimédii, databázemi apod.

GTK+ (GIMP Toolkit)

- Jako příklad knihovny na pomezí s frameworkem si uvedeme GTK+ (GIMP Toolkit). Tato knihovna původně vznikla při tvorbě grafického editoru GIMP. Je napsána v C, ale lze ji využít i v Pythonu a dalších jazycích a je přenositelná na Linux, Mac OS X i MS Windows.
- Jedná se o svobodný SW distribuovaný pod licencí LGPL a využívá se v řadě volně šiřitelných programů jako Inkscape, Firefox, Meld apod.
- Mimochodem kdo neznáte nástroj Meld pro porovnání obsahů složek, doporučuji se na něj podívat.

Další knihovny

- Jako příklady dalších knihoven zmíním např. libcurl pro přenos dat přes HTTP, FTP a další protokoly, OpenCV pro zpracování obrazu, počítačové vidění apod. nebo SDL pro nízkoúrovňový přístup k HW pro tvorbu A/V přehrávačů.
- Na webu najdete velké spousty knihoven. Doporučuji neskončit vždy u toho prvního, co najdete, ale najít nejprve několik alternativ a pak si z nich vybrat.

Statické × dynamické linkování

- Máme 2 způsoby, jak knihovnu připojit k programu – statické a dynamické linkování.
- U statického linkování se při linkování použité knihovny stávají součástí programu. K provozu pak stačí pouze přeložený program. Nevýhodou je, že přeložený program zabírá hodně místa na disku i v paměti.
- U dynamického linkování je knihovna přeložena zvlášť. Při linkování se pak ukládají pouze odkazy na symboly definované v dynamické knihovně. K provozu pak potřebujeme přeložený program i knihovnu. Výhodou je, že knihovnu sdílíme s dalšími programy, takže se šetří místo na disku i paměť, protože knihovna se do paměti načte pouze jednou. Nevýhodou je problém s různými verzemi knihoven a programů a jejich kompatibilitou.
- Dynamické knihovny mají v MS Windows příponu `.dll` (Dynamic-link library), v Linuxu `.so` (Shared object) a v Mac OS X `.dylib`.

Využití cizího kódu

- Ne vždy použijeme celé cizí dílo či produkt – můžeme potřebovat jen několik řádků kódu.
- Než však jakýkoliv kód využijeme, je nutné najít k němu licenci. U různých kousků kódu v článcích a komentářích na webu to může být problematické, ale pokud licenci nenajdeme, rizika jsou vysoká a měli bychom dobře zvážit, zda nám stojí za to takový kód využít. Problém nemusí nastat hned, ale až ve chvíli, kdy je náš produkt úspěšný a někdo se přihlásí o svůj podíl na zisku ...
- Jakýkoliv cizí kód je pak bezpodmínečně nutné správně citovat. Způsob citace může být nařízen i licencí.

Využití cizího kódu – licence

- Nejprve se blíže podíváme na licence.
- Jde-li o kód z nějaké knihovny či celého hotového produktu, licence dané knihovny či produktu musí umožňovat využití části díla. To je u komerčních licencí typicky zakázané či problematické, protože autor vidí riziko ve stanovení rozsahu takovéto části.
- Jedná-li se o kód z webu, vztahuje se na něj minimálně licence daného webu.
- Např. oblíbený Stack Overflow patří do Stack Exchange Network, kde jsou jasně stanovené podmínky využití obsahu a striktně nařízeno, že citace musí obsahovat původ kódu, URL otázky, jména a URL autorů otázky a využitých odpovědí apod. (viz https://stackoverflow.blog/2009/06/25/attribution-required/ či přímo https://stackoverflow.com/legal/terms-of-service).
- Stejně jako při využití celých knihoven i při využití části kódu může tato část ovlivnit licenci výsledného díla. Musíte se tedy dobře zamyslet, zda Vám např. stojí za to použít několik řádků kódu s licencí GPL a zavázat se tak k tomu, že celý Váš produkt bude pod licencí GPL.

Využití cizího kódu – citace

- Nyní se blíže podívejme na to, jak správně citovat kód, není-li to explicitně uvedeno v licenčních podmínkách.
- Základní pravidlo je, že vždy musí být zcela jasné, která část kódu je převzatá a odkud.
- Převezmeme-li (téměř) celý soubor, nebo vyžaduje-li to licence využitého kódu, musí být uvedena hlavička souboru, ve které budou uvedeny všechny potřebné informace – min. odkud byl kód převzat, kdo je jeho autorem a v jakém rozsahu je převzatý – tedy zda se jedná o celý obsah daného souboru, zda byl nějak modifikován apod.
- Jedná-li se o několik řádků kódu, musí před nimi být počáteční komentář s popisem zdroje, autorem a dalšími potřebnými informacemi. Nejedná-li se o funkci, u které je konec jasně daný uzavírací závorkou, musí být uveden i ukončovací komentář, aby byla citovaná část jasně vymezena.
- Vždy je navíc nutné překontrolovat, zda je citace v souladu s licencí – tedy zejména že je kód využit povoleným způsobem a že jsou uvedeny všechny vyžadované informace.

Využití cizího kódu – na co dát pozor

- A na závěr této přednášky upozorním ještě na několik věcí, na které byste měli dát pozor.
- Ve škole se doslovně citovaná část typicky nepočítá do rozsahu Vaší práce, nebo je její rozsah omezen. Vždy je třeba si dobře prostudovat zadání, nicméně v rámci toho, co je povoleno, nemá smysl se citacím vyhýbat. Pokud budete programovat věci, co už jsou dávno hotové a dostupné, stihnete za stejný čas mnohem méně práce a typicky se nedostanete k tomu zajímavému, čemu byste měli věnovat největší pozornost. U bakalářské či diplomové práce se navíc netrestá jen příliš velký podíl převzaté práce, ale i to, když „znovu objevujete kolo“. Malé množství využitých knihoven a citací a velké množství zbytečné práce svědčí o nedostatečném přehledu v dané oblasti.
- Když využijete něčí kód v aplikaci, na které vyděláváte malé desítky korun, málo komu bude stát za to se s Vámi soudit. Ale až se Vám startup rozjede a začnete vydělávat velké částky, skoro každý, kdo bude mít pocit, že by na Vás mohl vydělat, se o to pokusí. Dobře dostupná a jasná licence na kód, který využijete, Vás před tímto chrání. Naopak tam, kde licence chybí nebo ji momentálně neumíte najít, nikdy nevíte, co přijde.
- Studování licencí, závislost na někom dalším apod. jsou sice problémy, které vyžadují určitou opatrnost, ale rozhodně se tím nenechte odradit. Kdo se příliš bojí, podlehne tzv. syndromu NIHS („Not Invented Here“ Syndrome), který postihuje nejen jednotlivé vývojáře, ale i celé firmy. Jeho projevem je preferování vlastního kódu i v případech, kdy by externí řešení bylo lepší technicky i cenově. To, že „znovu objevujete kolo“, však zákazník neocení a dochází ke ztrátě času, peněz, příležitostí na trhu apod.

Odkazy

- Zde jsou zdroje, ze kterých čerpá tato část prezentace, a další zajímavé odkazy.

Shrnutí

- A na závěr několik hlavních myšlenek z této prezentace:
  - Dobré zvyklosti při psaní komentářů nám umožní vygenerovat programovou dokumentaci ze zdrojových textů.
  - Vhodná volba knihoven nám může ušetřit spoustu práce a peněz.
  - Při využití cizího kódu je vždy potřeba řádně prostudovat licenci a správně citovat.