Text ke 3. přednášce do IVS 2019/2020 na téma

Týmová spolupráce, komunikace, sdílení dat v týmu, základy návrhu a plánování projektu

Obsah

- Cílem této přednášky je seznámit se se základními postupy, jak sestavit tým pro práci na malém projektu, dohodnout se na komunikačních kanálech a prostředcích pro sdílení dat v týmu a jak tyto kanály a prostředky efektivně využít.
- Hned na začátku zmíním rozdíl mezi prostředky pro komunikaci a prostředky pro sdílení dat v týmu. I když je tento na 1. pohled intuitivní, půlsemestrální zkoušky v předchozích letech ukázaly, že stojí za zmínku. Zatímco komunikací myslíme výměnu zpráv – tedy to, co bychom si při práci v jedné kanceláři řekli ústně, pod sdílení dat spadá sdílení zdrojových textů programů, dokumentace apod. – tedy toho, na čem a s čím společně pracujeme.
- V závěru budou zmíněny základní informace o plánování projektu a návrhu vytvářeného software.

Komunikace v týmu

Sestavení týmu a prvotní komunikace v týmu

- Ve škole i v budoucím zaměstnání se můžete dostat do situace, kdy musíte sestavit tým lidí k řešení projektu.
- Ne vždy si však můžete zvolit členy svého týmu podle svých představ a ne vždy máte možnost ihned zjistit, který člen je zkušený a pracovitý a který nikoliv.
- Tým lidí je navíc často ustaven pomocí elektronických prostředků a členové nikdy nebudou sedět ve společné kanceláři.
- Vhodným přístupem a komunikací můžete omezit negativní dopad činnosti „nepříliš dobrých“ členů týmu.
- Základem je, že ihned po sestavení týmu zjistíte informace o jeho neznámých členech.

Komunikace

- Ke komunikaci existuje mnoho přístupů a nástrojů. Velké firmy mají svoje komunikační standardy a pravidla, která je nutné dodržovat. Kdybychom měli probírat příklady z jednotlivých firem, překročili bychom čas na tuto prezentaci. Firma vás navíc při nástupu vždy řádně zaškolí, takže o nic nepřijdete.
- Nebudeme se tedy zabývat komplexním prostředím ve velkých firmách, ale situací, která nastává ve škole při práci na projektech do jednotlivých kurzů či v praxi při práci na živnostenský list.
- Situace je obvykle taková, že dostanete za úkol sestavit tým. Ve škole z lidí, kteří chodí do daného kurzu, v praxi z lidí, se kterými jste už někdy spolupracovali, a z lidí, na které dostanete v rámci zakázky kontakt. Každý však může mít jiné komunikační zvyklosti a využívat jiné komunikační kanály. Např. grafička může preferovat telefon a e-mail, pomocný programátor Jabber a specialista na danou oblast je zvyklý na diskusní fóra.
- Pokud Vy preferujete diskutování na nějakém fóru, obvykle nejste v pozici, kdy můžete někoho dalšího nutit, aby se na toto fórum registroval a poskytl tak jeho správci nějaké osobní údaje, byť jich chce pouze nezbytné minimum.
- Dohoda na komunikačních kanálech a způsobu komunikace je tedy v této situaci především otázkou kompromisu, na kterém je nutné se dohodnout.

Příklad

- Pro další pokračování v tomto tématu si stanovme příklad situace, které se tyto informace týkají. Mohl bych sice zvolit třeba příklad z práce na živnostenský list, ale raději zvolíme situaci, která Vám bude bližší a do které se můžete dostat třeba už v průběhu příštího nebo některého z následujících semestrů.
- Mějme povinný předmět studia, ve kterém dostanete zadaný týmový projekt. Každý tým musí mít 4 nebo 5 osob, ale vy máte pouze 2 kamarády, takže máte 3-členný tým. Nebo máte 5 kamarádů a jednoho přece nevyhodíte, tak se rozdělíte na 2 týmy.

- V každém případě ve Vašem týmu zůstanou volná místa a na ta se Vám někdo zapíše přes IS. Řekněme, že se přihlásí 2 lidé, o kterých nic nevíte.
- Také předpokládejme, že kamarády znáte z přednášek či ze střední školy, ale v týmu jste s nimi nikdy nepracovali. Komunikaci tedy bude nutné dohadovat se všemi.

Prvotní komunikace

- Pokud se chceme dohodnout na komunikačních kanálech, základních pravidlech pro komunikaci a dalších věcech, které je v týmu potřeba vyřešit, jako např. rozdělení zodpovědností a volba vedoucího, vždy je nejlepší uspořádat osobní schůzku. Pokud však má každý člen týmu jiný (časový) rozvrh, může být dohoda na společném místě a termínu schůzky problematická. Navíc často známe pouze e-mailové adresy či telefonní čísla ostatních členů týmu.
- Pokud má tým více než 5 lidí, začíná být dohadování pomocí telefonátů a e-mailů zdlouhavé a nepřehledné. Ne každý využije „odpovědět všem“ a ne všichni jsou tak informovaní o aktuálním stavu. Ne každý je ihned ochotný všem ostatním poslat, které dny v týdnu má čas a na která místa je ochoten přijít. I pokud se tyto informace vymění, stále je třeba, aby někdo zjistil, kde se časová okna překrývají a jedno z nich vybral tak, aby byli ostatní spokojení. Pokud se navíc okna nepřekryjí, bude někdo muset ustoupit a udělat si čas. Pokud se toto nepodaří dostatečně rychle, bude další týden a dohadování může začít znovu od začátku s jinými rozvrhy.
- Dohodu lze výrazně urychlit využitím jednoduché webové služby, která umožní hlasování o termínu schůzky, případně i dalších problémech. Mezi takovéto nástroje patří např. Doodle, Framadate nebo NeedToMeet.
- Pokud je potřeba složitější dohadování, lze využít nějakého fóra (je vysoká pravděpodobnost, že kolegové budou např. na Facebooku, kde lze založit privátní chat), ale ne každý bude ochoten se někde registrovat.
- Pozor na veřejně přístupná fóra či fóra, kde má provozovatel fóra práva k Vašemu obsahu – na těch nelze řešit libovolný problém, protože nikdy nevíte, co předáváte konkurenci.

Doodle

- Zde vidíte ukázku hlasování na Doodle. V záhlaví tabulky jsou nabídnuté termíny a na řádcích jména osob. Pro založení hlasování stačí zadat název události, termíny a svoje jméno a e-mail. Celé hlasování pak lze provést bez zadání jediného citlivého osobního údaje, protože stačí zadat pouze jméno či přezdívku a vybrat si z možností. U každého termínu můžete zvolit, zda je dobrý, přijatelný (2 kliknutí) nebo nevhodný. Pod hlasování lze připisovat i textové komentáře.
- Lze nastavit i striktní hodnocení termínů (ano či ne), ale pak hlasující často raději vybírají „ne“ a může se stát, že žádný termín nebude vhodný.
- Je-li to možné, je třeba vypsat dostatečné množství termínů. Pro 5 lidí výběr ze 2 možností pravděpodobně nedopadne dobře, protože se obvykle najde alespoň 1 člověk, který si z nich nevybere.

Doodle

- Po tom, co již někdo hlasoval, je v zápatí zobrazené vyhodnocení pro jednotlivé termíny.
- Ten, kdo hlasování založil, jej na závěr uzavře, při čemž dle výsledků hlasování zvolí nejvhodnější termín.

Příklad

- Nyní se vrať me k našemu příkladu. Všichni členové týmu mají stejné zadání a nikdo není předem určený jako vedoucí týmu. Začíná tedy jakási hra na nervy, kdy všichni čekají, až se někdo ozve. Ve výhodě můžou být ti členové týmu, kteří se znají, ale stále je pravděpodobné, že budou čekat, až se jim ozvou zbývající 2.
- Toto chování je fatální chyba, protože se nesmyslným čekáním ztrácí čas na práci na projektu. Komunikovat by měli začít všichni a ihned, protože to je jediný způsob, jak rychle začít. Každý by tedy měl poslat e-mail všem ostatním, ve kterém uvede detailnější kontaktní informace, nabídku termínů schůzky či odkaz na Doodle (pokud už jej neposlal někdo z ostatních).
- Lze nabídnout i nějaký lepší komunikační kanál, např. neveřejné fórum pod vaší správou, kde lze provést dohodu na věcech, co se musí rozhodnout před termínem 1. schůzky (např. název týmu pro registraci). Následně by se měl využívat nejlepší nabídnutý komunikační kanál, případně pokud je více stejných, měl by se využít první nabídnutý.

- V našem příkladu máme členy týmu, které si pro anonymitu nazveme písmeny od začátku abecedy. Nejrychleji svůj e-mail poslal pan C, který všem zaslal svůj Jabber, telefonní číslo a časy, ve které by příští týden mohl přijít na schůzku do školy seřazené podle vhodnosti.
- Následně reagoval B, ale ten poslal jenom ICQ a jeden termín, který si ze seznamu pana C vybral. To je od něj sice hezké, ale jeden vybraný termín je k ničemu, protože ostatním se nemusí hodit. Správně měl poslat všechny termíny, ve které může přijít, nebo alespoň ty, ve kterých se shoduje s C.
- Pan A potom kromě jiného poslal i odkaz na Doodle a svůj Facebook – třeba se podaří, ostatní se mu tam ozvou a bude schopen založit společný chat. Pokud někdo z Vás přemýšlí, proč posílal časy i odkaz na Doodle, tak to není proto, že je aktivní blbec, ale proto, že nechce zdržovat v případě, že ostatní budou posílat e-maily a Doodle z nějakého podivného důvodu nevyužijí. Navíc může být využita jiná služba než Doodle a může vyžadovat zadání nějakého osobního údaje, na které nemusí přistoupit každý.

Příklad

- Každý by měl nyní odpovědět na první přijatý e-mail a pak na každý další, na který je třeba reagovat.
- Vždy by měla být využita funkce „odpovědět všem“, aby se nestávalo, že někdo neví, co ostatní řeší. Pokud už někdo ostatní členy týmu do seznamu příjemců nezahrne, je potřeba to co nejdříve napravit – když pošlete odpověď všem, získají s ní i původní e-mail (pokud ho nesmažete – viz dále).
- Když už je možnost přejít na lepší, tedy především přehlednější komunikační kanál, všichni by to měli udělat. Měli by tedy hlasovat na Doodle a sledovat tam, který termín bude vybrán.

První schůzka týmu

- První schůzka týmu by měla být vždy osobní. Přítomnost přes telekonferenci je stále částečně neosobní, což má negativní vliv na důvěru k ostatním členům týmu. Navíc se přes telekonferenci typicky příliš neseznámíte. Je snadné bez varování opustit tým lidí, které jste nikdy neviděli a nic o nich nevíte. Ale pokud víte, že když tento tým opustíte, někdo nebude mít možnost jít ke státnicím, někdo s malými dětmi se dostane do exekuce apod., budete přemýšlet ...
- Na schůzce je třeba dohodnout alespoň následující:
  - Každý tým musí mít vedoucího. Jedná se o osobu, která zodpovídá především za dodržování termínů a popohání ty, kteří je nedodržují. Vedoucí týmu musí mít stále přehled o tom, kdo a na čem pracuje a zda vůbec někdo pracuje. Je zodpovědný i za komunikaci se zákazníkem, aby neprobíhala chaoticky a nebyly mu opakovaně pokládány stejné dotazy. Místo vedoucího není nějaký privilegovaný post, ale spíše velice nepříjemná práce, kdy někoho popoháníte, aby neměl problémy s termínem, a on Vám za to nadává. Měl by to být nejspolehlivější člen týmu, který má největší motivaci a který je ochoten se této práce ujmout. Pokud nikdo nechce být vedoucím, je třeba losovat mezi spolehlivými a motivovanými členy týmu.
  - Po volbě vedoucího je třeba dohodnout stabilní komunikační kanály, pravidla pro komunikaci a prostředky pro sdílení dat v týmu. Těmito se budeme zabývat v následujících částech této přednášky.
  - Nakonec lze předběžně rozvrhnout práce na projektu, rozdělit počáteční úkoly apod. Bude záležet především na tom, kolik bude na schůzku času a jak rychle se podaří dohodnout na základních věcech.

Komunikační kanály

Komunikační kanály

- Komunikační kanály jsou kanály využívané pro komunikaci, tedy výměnu zpráv, resp. diskusi (to, co bychom si při práci v 1 místnosti řekli ústně). Neřešíme zde výměnu zdrojových textů programů, dokumentace, ani testovacích či jiných dat – tímto se budeme zabývat později.
- Mezi základní nejčastěji využívané komunikační kanály patří telefon, e-mail, instant messaging, diskusní fóra, chaty a další webové služby.

Telefon

- Nejrychlejším kanálem je obvykle mobilní telefon. Ne každý je však ochoten poskytnout ostatním členům týmu své číslo, protože je nezná. Je to určitá prevence před obtěžováním, která však ihned na začátku zavádí do týmu nedůvěru a nepříjemnou atmosféru. Úspěch projektu je přitom závislý především na dobré týmové spolupráci.

- Motivací ke sdělení telefonního čísla může být i to, že kolegové Vás mohou upozornit na nějakou náhlou událost či na to, že zapomenete na nějaký důležitý termín.
- Už jsem byl několikrát v situaci, kdy se na někoho čekalo kvůli tomu, že si spletl nebo špatně zapamatoval termín. Dokonce i u státnic se nejednou stalo, že měl někdo za 15 min. přijít na řadu a stále nebyl ve škole. Obvykle se jednalo o posledního člověka, který počítal svůj čas od konce, na kterém je rezerva, místo od začátku. Kdybychom neměli k dispozici jeho telefonní číslo, jeho pozdní příchod pro něj mohl mít fatální následky (k tomuto je nutno poznamenat, že dnes již telefonní čísla studentů v IS FIT nejsou dostupná kvůli ochraně osobních údajů, takže kdo si tam číslo dobrovolně nevyplní, tomu už nepomůžeme).
- Využití telefonu by však mělo být minimální a měl by se využívat pouze pro neodkladné záležitosti. Neobtěžovat ostatní častými a zbytečnými telefonáty patří k ohleduplnému a serióznímu jednání člena týmu.

E-mail

- E-mail je pravděpodobně nejdostupnější komunikační prostředek. Bohužel je však také jedním z nejpomalejších. I přesto, že mají obě komunikující osoby spuštěného poštovního klienta, může výměna e-mailů trvat i desítky minut. Pokud je někdo zvyklý vybírat poštu pouze ráno nebo náhodně 2x týdně, může být doba čekání na odpověď nepřijatelná.
- Proto je u důležitých zpráv výhodné využívat doručenku, která umožní zjistit, zda už daná osoba e-mail přijala. Zaslání doručenky lze sice přeskočit, nicméně při týmové spolupráci je toto jednání zcela nevhodné.
- Pokud řešíme týmový projekt, je navíc vhodné zasílat každou zprávu v kopii všem ostatním členům týmu a při odpovídání využívat funkci „Odpovědět všem“, která zašle kopii odpovědi i příjemcům kopie původní zprávy. Předejdeme tím zdržování a různému přeposílání zpráv se slovy „tenhle řekl tohle a tenhle tamto“. Ve větších projektech se využívají mailing listy, což jsou seznamy příjemců a specializované nástroje, které s nimi pracují (např. http://www.gnu.org/software/mailman/index.html). Když pak pošlete e-mail na speciální adresu, následně jej dostanou všechny ostatní osoby na seznamu a nemůže se stát, že na někoho zapomenete.
- Musím zde upozornit také na to, že reakci na e-mail vždy píšeme nad citovaný text e-mailu, na který reagujeme, nebo daný e-mail rozčleníme a na jednotlivá témata reagujeme přímo pod nimi. Pokud svoji reakci připíšete na konec, autor původního e-mailu musí skrolovat přes svůj text až na konec k Vaší odpovědi. Pokud si vyměníte několik e-mailů a oba budete psát na konec, budete se zbytečně otravovat skrolováním. Pokud byste se chtěli skrolování vyhnout tím, že vymažete původní text e-mailu, je to ještě mnohem horší, protože je třeba hledat, čeho se vlastně reakce týká. Zachování celé historie odpovědí je tedy také součástí dobrých zvyklostí.

Instant messaging

- Existují různé sítě jako ICQ, Jabber, WhatsApp, Discord apod. Do různých sítí se lze připojovat různými klienty a některé sítě navíc umí přenos do jiných sítí. Pokud obě komunikující osoby nevyužívají stejnou síť a stejného klienta, mohou nastávat různé problémy. Autorizace se nemusí zdařit, může docházet k její ztrátě a nutnosti opakování. Může to také vypadat, že je někdo offline, ale ve skutečnosti má jen rozdílného klienta, který jinak vysílá status. Místo smajlíka se zobrazuje nesrozumitelný text, přenosy souborů často nefungují apod,
- Některé sítě či klienti nepodporují chat 3 a více osob současně, nebo jej mají nějak omezen (např. musíte mít všechny členy přidané mezi přátele apod.), a při chatu s různými klienty mohou nastávat různé problémy s vypadáváním osob z místnosti apod. Jednou jsme při řešení projektu diskutovali a když jsem se jednoho z členů týmu zeptal na jeho názor, zjistil jsem, že zprávy z chatu mu už asi 5 minut vůbec nechodí. Jenom mu bylo divné, že tak dlouho nikdo nic nepíše. Instant messaging lze tedy využít spíše při komunikaci 2 osob, které společně řeší nějaký podúkol, nebo při dotazech na konkrétní osobu, které se netýkají celého projektu. V těchto případech má velké výhody v rychlosti a jednoduchosti.
- Pokud je tímto kanálem uskutečněno nějaké důležitější rozhodnutí, je vhodné si jej poznamenat a zpřístupnit ostatním přes jiný komunikační kanál, např. fórum nebo e-mail. Obvyklým problémem totiž je, že historie komunikace se ukládá pouze do klienta, který toto podporuje a je pro uchování záznamů komunikace nastaven. Potom někdo přijde k jinému PC a k potřebným informacím se nedostane. A i když se jedná o něco, co je online jako Discord, stejně se v dlouhé nestrukturované historii důležité věci špatně hledají.

VoIP / Skype / Google Hangouts

- VoIP, Skype či Google Hangouts nám umožňují telefonovat přes Internet. Kromě standardních hovorů jsou zde i možnosti videohovorů, audiokonferencí s více účastníky a videokonferencí.

- Není-li možná osobní schůzka, lze ji částečně nahradit i schůzkou přes Skype. První schůzka týmu by však měla být vždy osobní.
- Nevýhodou však je, že některý člen týmu nemusí mít potřebné vybavení, tedy mikrofon, sluchátka, webkameru a dostatečně rychlé připojení k Internetu, nebo jej má pouze doma či pouze v práci.
- Zde bych chtěl upozornit na to, jak strašně nepříjemné a otravné je, když má někdo mikrofon a reproduktory, tedy nepoužívá náhlavní soupravu se sluchátky. Daný člověk má sice pohodlí a neslyší žádný problém, ale ten, kdo je na druhé straně, slyší svoji ozvěnu, což jej po chvíli totálně otráví. Pokud nepoužívá náhlavní soupravu žádný z účastníků hovoru, přes ozvěny a pískání často nelze vůbec hovořit. Využití náhlavní soupravy tedy jednoznačně patří k ohleduplnému chování. Je-li u jednoho PC více lidí, nebo z jiného důvodu nelze využít náhlavní soupravu, je potřeba ve chvílích, kdy nemluvíte, ztlumit mikrofon. Ušetří se tím i šířka pásma pro přenos, což může u více účastníků na pomalejších linkách znamenat opravdu znatelné zlepšení. Ve více účastnících to tedy lze doporučit vždy. Ztlumené mikrofony však zpomalují reakční dobu a musíte tedy počítat s větší délkou hovoru. Také je potřeba dávat ostatním prostor a více času na reakci.
- Velmi důležité je dělat si poznámky. Je třeba si vypsat důležitá rozhodnutí, úkoly a další potřebné informace. Po 2 hodinách dohadování už si jinak na polovinu dohodnutých věcí nevzpomenete a záznam se obvykle nepořizuje. I když však záznam budete mít, není v něm možné jednoduše vyhledávat a jeho kompletní procházení je časově náročné. Málo kdo má navíc nervy na to se 2 hodiny poslouchat ze záznamu.

Diskusní fóra

- Diskusní fóra jsou asi nejlepším prostředkem pro komunikaci v týmu. Historie komunikace se zaznamenává a informace jsou odkudkoliv přístupné všem členům týmu. Fórum je sice pomalejší než třeba Jabber, ale může být rychlejší než e-mail (pokud jej členové týmu sledují). Některá fóra navíc umožňují upozorňování na důležité zprávy e-mailem či přes mobil, nebo mají přímo samostatné mobilní rozhraní či aplikaci.
- Je však velmi důležité zachovat přehlednost. K tomuto účelu je vhodné využití vláken a moderátora, který dohlíží na dodržování stanovených pravidel pro komunikaci na fóru. Nejsou-li k dispozici vlákna, řešení více problémů se prolíná a rychle se ztrácí kontext a přehled. Není-li fórum moderované, porušení pravidel může snížit přehlednost fóra např. rozprostřením jednoho problému do více vláken, spojením více problémů do jednoho vlákna apod.
- Na fóru je také nutné udržet emoce, vyjádřit svůj názor, respektovat názor druhého, ale nezačít hádku. Hádka, která již nepřináší žádné konstruktivní připomínky, odsunuje do historie a zatemňuje původně řešený problém. Pokud se 2 uživatelé pohádají (typicky zda je lepší Windows nebo Linux), zbylých 5 členů týmu už nemusí být schopných zjistit, co se vlastně řešilo. Při hádce by měl vždy zasáhnout moderátor.
- Pro práci na projektu byste měli vždy využít neveřejné fórum. Využijete-li veřejné fórum, může jej např. pomocí nějakého vyhledávače najít konkurence a využít Vaše řešení, které zde nalezne přehledně s výhodami i nevýhodami.
- Pokud fórum neprovozujete sami, dávejte si pozor i na podmínky použití fóra, zda nedáváte licenci k jeho obsahu provozovateli fóra.

Další webové služby

- Existuje celá řada dalších webových nástrojů pro komunikaci, z nichž některé jsou i velmi sofistikované a specializované na týmovou spolupráci.
- Nevýhodou je, že na využití takového nástroje se musejí dohodnout všichni členové týmu a musejí se s ním seznámit. Seznámení s takovým nástrojem a jeho efektivním využitím také přináší určitou režii, která zkracuje čas na samotnou práci na vytvářeném produktu.

Facebook

- Některé nástroje jsou pouhou kombinací výše uvedených (případně s nějakými vylepšeními). Za zmínku stojí např. Facebook, který kombinuje fórum, chat, úložiště souborů a další jednoduché služby. Pokud jsou všechny nástroje v jednom, může to být velká výhoda.
- Nicméně v případě Facebooku je méně pohodlné a přehledné procházení historie, což může být značná nevýhoda. Jednotlivé příspěvky lze využívat jako vlákna fóra, lze je označovat nějakými značkami, které využijeme při hledání apod. Ale takovéto využívání není nijak vynuceno a nebudeme-li jej sami využívat disciplinovaně, rychle se stane nepřehledným.

- Problematická je i licence k obsahu, který přes něj sdílíte.

Discord

- Poslední dobou studenti čím dál více využívají Discord. Pojd'me se tedy podívat na jeho vlastnosti.
- Jedná se o platformu pro hráče poskytující klasický nestrukturovaný chat. Integrovaná je i možnost videohovorů a vlastnostmi z hlediska týmové spolupráce je tak blízký např. Skype.
- Chceme-li jej využívat na něco, kde nechceme rychle ztrácet veškeré informace v historii, je třeba řešit nějaké minimální strukturování a různé mechanismy, které nám umožňují udržet některé informace déle viditelné. Řadu věcí si musíme doimplementovat sami pomocí robotů, kteří sledují diskusi v chatu a reagují na definované příkazy.
- Velmi zrádné je nastavení oprávnění. Zatímco ve hře nám stačí vytvořit chat mezi pár lidmi, který nebude mít dlouhou životnost, pro delší využití větším množstvím lidí již může být třeba řešit uživatelská oprávnění pro přístup k jednotlivým místnostem, využití robotů apod. K tomuto účelu jsou dostupné hierarchické role a paralelně s nimi nezávislá uživatelská oprávnění.
- Uživatel s nižší rolí nemůže modifikovat práva a nastavení uživatele s vyšší rolí, což by bylo logické, pokud by práva a role souvisely. Naneštěstí je možné systém nastavit tak, že uživatel s vysokou rolí nemá žádná práva a uživatel s nízkou rolí je naopak administrátorem. Administrátor pak nemůže spravovat uživatele s vysokou rolí a tento uživatel sám sobě také nic nenastaví, protože na to nemá práva. Snadno tedy vzniknou různé nežádoucí stavy.
- Možnosti vyhledávání v historii jsou navíc velmi omezené.
- I když se tedy platforma může zdát sofistikovaná a poskytující řadu možností, pro týmovou spolupráci na projektu oproti Skype výraznější výhody nepřináší.

„Zpátky na stromy“

- Když se v naší zemi začal masivněji šířit Internet, většina připojení byla vytáčených přes pevné linky. Platilo se za čas a člověk tak nebyl připojený pořád. Zprávy jsme si četli dávkově s nějakým zpožděním.
- Když je nových zpráv hodně a komunikuje se v nestrukturovaném chatu, je problém se zorientovat v tom, co k čemu patří. Proto jsme se snažili vytvářet dobře strukturovaná fóra a komunikaci kultivovat, aby byla přehledná a rychle jsme se zorientovali a vybrali, co je pro nás důležité.
- Dnes jsme téměř pořád online. Očekává se, že příjemce zprávu dostane okamžitě (min. na mobil) a zprávy čte pořád. Bude tedy mít relativně málo nepřečtených zpráv a bude stíhat sledovat, o čem se píše.
- Hodně lidí je online a jsou zvyklí většinu věcí řešit přes Internet. Množství informací je velké a obtížně zpracovatelné – snažíme se proto optimalizovat např. čtením 1. věty z každého příspěvku, rychlým přeskakováním nezajímavého obsahu apod.
- Strukturování obsahu přináší režie, tak se mu spíše snažíme vyhnout – když vše píšeme do 1 diskuse a nepřetržitě sledujeme, není potřeba přepínat mezi tématy, vlákny apod. – vše nepřečtené je na 1 místě.
- Pro zábavu ve volném čase a 2 lidi, co spolu řeší jeden problém, to funguje perfektně. Ale když je v SW projektu více lidí a řeší více věcí, ne vše udělají hned a občas je potřeba se k něčemu vrátit.
- Všichni jsou však zvyklí na nestrukturovanou komunikaci a až zjistí, že historie komunikace je nepřehledná a nepoužitelná, už je pozdě.
- Od zlepšování se ve strukturování a kultuře komunikace jsme tedy přešli k tomu, co jsme měli na začátku – jeden chat, kde je všechno smíchané. Jediný rozdíl je, že je to dynamické a zprávy vidíme ihned.
- Naučit se odlišit soukromou a profesionální komunikaci a kultivovat tu profesionální, aby se bylo možné podívat na historii, poučit z předchozích rozhodnutí apod. může být obtížné. Ale stojí za to se tím zabývat.

Projektová wiki a web projektu

Projektová wiki

- Projektová wiki je jakousi encyklopedií projektu. Lze v ní shromáždit veškeré informace o projektu a následně z ní čerpat informace při tvorbě dokumentace i zkušenosti s řešenými problémy pro práci na jiných projektech.
- Každý projekt by měl mít hlavní stránku, která bude obsahovat základní informace, jako je název, řešitelé, základní informace o zadání a cílech projektu, informace o celkovém stavu projektu apod.
- Na dalších stránkách by potom měly být detailní informace o jednotlivých řešených problémech, podproblémech, dosažených cílech, vytvořených součástech programu apod.
- Vše by mělo mít dokumentační charakter, tzn. že by se z wiki stránky nemělo stát diskusní fórum, ve kterém se důležité informace ztratí v nerelevantních otázkách a odpovědích.

Příklad wiki stránky bakalářské práce

- Jako ukázku wiki stránky projektu jsem zvolil stránku jedné zdařilé bakalářské práce. Jak vidíte, na začátku jsou nejdůležitější informace, následuje cíl práce, úvod do řešené problematiky, informace o řešení a výsledky.

Web projektu

- U projektů, u kterých je potřeba informovat veřejnost a sponzory, je nutné vytvořit web projektu. Můžeme jej však vytvořit u jakéhokoliv projektu, i když pro příliš malé projekty se jeho tvorba nemusí vyplatit.
- Web projektu však nemusí být pouze stránky s informacemi pro veřejnost. Můžeme vytvořit celý portál, na kterém lze agregovat webové služby pro podporu práce na projektu. Může zde být diskusní fórum, chat, úložiště dokumentů apod. Lze vytvořit i sekce pro zadávání pracovních výkazů a sledování stavu projektu.
- Vždy je však třeba pečlivě zvažovat, které informace budou veřejně přístupné a ke kterým se dostanou pouze přihlášení členové týmu, případně sponzoři či jiní uživatelé.

Příklad

- K webu projektu zde mám obrázek, jak vypadal web jednoho z projektů řešených v magisterském studijním programu. Kromě veřejných informací obsahoval chat, diskusní fórum, ankety pro hlasování o problémech, úložiště dokumentů a pracovní výkazy. Jeho značnou nevýhodou bylo, že diskusní fórum nepodporovalo vlákna.
- Na profesionální web projektu se můžete podívat např. na https://mixedemotions-project.eu/

Nástroje

- Web projektu samozřejmě nemusíme psát manuálně pro každý projekt. Jednoduché fórum, chat a úložiště dokumentů lze napsat rychle (web, který jste viděli na předchozím slajdu jsem vytvořil za cca 40 hod.). S větším množstvím integrovaných služeb by však rozsah prací na webu projektu mohl překonat rozsah práce na samotném produktu.
- Pro vytvoření webu projektu máme k dispozici hotové webové aplikace, které integrují fóra, wiki, systém pro správu verzí, plánovač a další potřebné nástroje.
- Mezi tyto aplikace patří např. Redmine a Trac. Oba tyto systémy jsou vyvíjeny s otevřeným zdrojovým kódem a můžete si je tedy stáhnout, vyzkoušet, případně i začít využívat na školních projektech, i když na školní projekt může být takový nástroj jako kanón na vrabce.

Redmine

- Na tomto slajdu vidíte snímek obrazovky Redmine. Pro jednotlivé úkoly je přehledně zobrazeno, v jakém stavu daný úkol je, jakou má prioritu, kdo jej má přiřazen apod.

Pravidla pro komunikaci

Pravidla pro komunikaci

- V týmu je také třeba stanovit určitá pravidla pro komunikaci, která budou všichni členové dodržovat. Tato pravidla zahrnují především to, které komunikační kanály budou využívány k jakým účelům a minimální frekvence využívání těchto kanálů. Je třeba stanovit minimální frekvenci kontroly pošty a prohlížení diskusního fóra, např. 1x denně v pracovní dny a pokud je využíván instant messaging, tak i nějaké minimální časové rozsahy, kdy budou všichni dostupní online.
- Bez stanovených pravidel pro komunikaci se může stát, že bude komunikace váznout. Navíc nebude jasné, jak dlouho čekat na něčí reakci, a můžeme ztratit hodně času.

Příklad

- Nyní se vrat'me k našemu příkladu. Protože A, B a C jsou kamarádi, jako vedoucí by měl být zvolen jeden z nich, protože je menší pravděpodobnost, že projekt vzdá a kamarády v tom nechá. Nikdo z nich však o vedení nemá výraznější zájem. Protože A na začátku nabídl Doodle, B a C ho zvolí vedoucím. Pokud by vedení nepřijal, mohli by losovat.
- A nabídl, že vytvoří web projektu (či třeba zajistí účet na Redmine) a umístí na něj chat a fórum. Ostatní souhlasili a následně se dohodli na těchto pravidlech:
- Hlavním komunikačním kanálem bude fórum na webu projektu. Každý jej navštíví nejméně 1x za každý pracovní den, přičemž se stejnou frekvencí se bude vybírat i pošta. Moderátorem fóra bude vedoucí projektu.
- Pro elektronické diskuse mezi více členy současně bude využíván chat na webu projektu, termíny diskusí budou domlouvány na fóru.
- Pro komunikaci 2 osob pracujících na podúkolech bude využit Jabber. Každý člen týmu, který zrovna pracuje na nějakém úkolu, by měl být při práci online.
- Telefon se bude využívat pouze výjimečně. Pouhé prozvonění telefonu je považováno za výzvu k příchodu na Jabber.
- Pokud někdo bude nemocný, nebo z jiného důvodu nebude moci daná pravidla dodržet, na fóru se omluví a uvede, proč a do kdy nebude komunikovat, případně proč nějakou dobu nekomunikoval. Pokud je to možné, omluva by měla být zveřejněna předem. Pokud by někdo byl v nemocnici a hrozilo, že se již projektu nezúčastní, měl by se ozvat vedoucímu, např. prostřednictvím telefonu či vzkazu přes třetí osobu.
- Jistě Vám neuniklo, že zde není uveden Skype. Je to proto, že pánové D a E řekli, že Skype nevyužívají a ne vždy je v projektu potřeba mít k dispozici všechny typy kanálů.
- Všimněte si také nutnosti omluvy. Některé z Vás jistě napadlo, že pravidlo je příliš přísné a nikomu není nic do toho, proč jste nemohli komunikovat. Za jeden den nepřítomnosti se samozřejmě není třeba omlouvat, ale pokud někdo jeden nebo dva týdny nekomunikuje, může to znamenat, že projekt vzdal, nebo že mu na něm nezáleží a moc času mu nevěnuje. Při práci na jednom projektu se mi stalo, že jeden kolega řešil svůj podúkol 4 týdny s tím, že po prvních dvou týdnech teprve odpověděl na e-mail s tím, že na tom pracuje a polovinu už má hotovou. Čtvrtý týden, když už se opět dlouho neozýval, jsem mu telefonoval a s hrůzou jsem zjistil, že celou dobu nedělal nic a projekt vzdal.

Prostředky pro sdílení dat v týmu

Prostředky pro sdílení dat v týmu

- Od komunikace, která se zaměřuje na domlouvání a výměnu zpráv, přejdeme ke sdílení dat v týmu. Data jsou vytvářené dokumenty, zdrojové texty programů, testovací data a další soubory, které jsou při práci na projektu potřebné.
- Pomineme-li sdílení dat přenášením po fyzických nosičích, které je už spíše nereálné, a různé peer-to-peer sítě a další řešení, která nejsou k tomuto účelu nejběžnější, máme 4 různé možnosti pro sdílení dat v týmu: FTP, využití webových služeb, sdílený disk a systémy pro správu verzí.

(S)FTP

- (S)FTP je zcela jistě nejdostupnějším řešením. Stačí založit účet na nějakém hostingu zdarma, který nechce příliš mnoho osobních údajů, rozeslat přihlašovací údaje a výměna dat může začít.
- Pokud však pomocí FTP sdílíme něco, co se v průběhu práce na projektu mění, brzy nastanou problémy. Ztrácíme totiž předchozí verze souborů a pokud někdo něco pokazí, již nemusí být cesta zpět. Navíc si 2 členové týmu mohou stáhnout soubor, jeden z nich jej upraví a aktualizuje a potom jej aktualizuje druhý, čímž zahodí změny od toho prvního. FTP také není vhodné pro souběžný přístup, takže pokud budou zapisovat 2 osoby současně do 1 souboru, výsledek může být různý.
- I přes všechny nevýhody lze FTP při sdílení dat využít. Hodí se především pro soubory, které jednorázově někdo vytvoří a dále už je ostatní jenom stahují, jako např. zápisy a záznamy ze schůzek, sady testovacích dat apod. Vzhledem k efektivnímu způsobu přenosu se hodí i pro velké soubory a výhodou může být i možnost stažení souboru přes HTTP, tzn. možnost někomu zaslat odkaz, který si rychle bez hledání otevře v prohlížeči.
- Možnost přístupu přes HTTP je však současně i velkou nevýhodou. Pokud využijete nějaký webhosting zdarma a u souborů si nenastavíte vhodná oprávnění, můžete veřejně zpřístupnit i utajované informace, které se mohou hodit konkurenci. To, že konkurence nezná adresu, nemusí být překážkou, protože pokud daný web nemá žádnou indexovou stránku, webserver může automaticky zobrazovat seznam souborů, který následně zaindexuje vyhledávač.
- I když správně nastavíte oprávnění, přenášení dat v podobě otevřeného textu představuje další riziko. Vždy proto preferujte zabezpečený přenos – tedy SFTP.

Web

- Na webu lze nalézt řadu služeb pro sdílení souborů. Soubor si uložíte a rozesíláte odkazy, nebo využijete nějaké sdílené úložiště, které funguje jako FTP přes web. Takové úložiště lze vytvořit i na webu projektu.
- Sice existují i sofistikovanější řešení, ale obvykle se skutečně jedná o nějakou obdobu FTP přes web. Z toho plynou i stejné nevýhody jako u FTP, ke kterým se navíc často přidává výrazné omezení velikosti souboru.
- Takové úložiště lze vytvořit i ve webu projektu.
- Vždy je však potřeba dávat pozor na bezpečnost dat.

Příklad

- Zde mám opět příklad z reálného studentského projektu. Jedná se o velice jednoduché úložiště, které však pro daný účel bylo dostačující. Všimněte si odkazů pro vymazání souboru. Nepříjemná situace může nastat ve chvíli, kdy někdo použije stahovač na všechny odkazy se záměrem stažení celého obsahu úložiště, aniž by přemýšlel o tom, že jsou tam i odkazy pro smazání souboru. Tuto situaci jsem v daném projektu zažil a viníka usvědčil až denní záznam z logu webserveru.

Sdílený disk

- Dalším prostředkem pro sdílení dat v týmu je sdílený disk v cloudu. Mezi tyto patří např. Dropbox, Google Drive, TeamDrive, Microsoft OneDrive nebo Cesnet ownCloud. Některé umějí uchovat předchozí verze souborů apod., nicméně pro sdílení zdrojových textů programu mají stále různé nevýhody – např. obtížnější získání předchozí verze souboru či prohlížení změn, chybí popisy změn apod. Uchování historie také může být omezené, případně se za její delší uchování více platí.
- Např. pokud dojde ke konfliktu – tedy k přepsání novější verze upravenou starší, lze v historii dohledat 2 verze, z nichž ani jedna není správná (neobsahuje obě sady změn) a nic nás neupozorní na to, že tyto 2 verze existují a které to přesně jsou (bude nutné hledat a pak je nějak porovnat a zkombinovat).
- Vždy je potřeba dát pozor na podmínky využití (jaká má provozovatel úložiště práva k uloženému obsahu) a na to, zda v průběhu řešení projektu nemůže vzniknout nutnost další platby za nějakou službu (přesažení maximální kapacity úložiště, návrat ke starší verzi souboru apod.).

Systémy pro správu verzí

Systémy pro správu verzí

- FTP, jednoduchá webová služba ani sdílený disk tedy nejsou vhodné ke sdílení souborů, které se často mění, zejména ke sdílení zdrojových textů programu. K tomuto účelu existují mnohem lepší a sofistikovanější nástroje, kterými jsou systémy pro správu verzí.
- Anglicky tyto systémy označujeme jako RCS (Revision Control System) nebo VCS (Version Control System). RCS je však současně název jednoho staršího nástroje, takže je lepší využívat označení VCS.
- Tyto systémy umožňují vyhledání libovolné předchozí verze souboru, přičemž u jednotlivých verzí máme i popisy změn a můžeme snadno prohlížet rozdíly.
- Umožňují i řešení konfliktů, přičemž pokud neměníme stejnou část souboru, některé nástroje toto do značné míry vyřeší automaticky (není třeba obě verze spojovat manuálně).
- Umožňují mít více větví vývoje, tedy verzí se kterými se pracuje souběžně (např. údržba stávající a vývoj nové verze).
- Systém pro správu verzí však typicky není vhodný pro velké objemy dat, u kterých není potřeba uchovávat předchozí verze. Pokud do adresáře se zdrojovými texty programu uložíte 1000 souborů s 2 GB testovacích dat, kolega, který je zrovna připojený přes mobil, kde platí za přenesená data nebo má datový limit, nejprve začne aktualizovat svoji verzi a stahovat. Za chvíli zjistí, co se děje, a začne nadávat. Potom bude muset vybírat, co si chce aktualizovat a co ne a jak se s tím vypořádat, aby si tu hrůzu nemusel stahovat.
- Některé starší systémy pro správu verzí navíc s velkými soubory neumějí pracovat, takže uložení souboru se nemusí zdařit a celý systém pro správu verzí může zkolabovat.

Historie

- Systémy pro správu verzí prošly dlouhým vývojem. První byl SCCS (Source Code Control System) z roku 1972, který umožňoval spravovat verze 1 vývojáři v 1 adresáři.
- V roce 1986 vznikl CVS (Concurrent Versions System), který má centrální server a umožňuje sdílení kódu mezi více vývojáři.
- V roce 1999 pak vznikl SVN (Subversion), který provádí atomické commity a na rozdíl od CVS tedy nehrozí nekonzistence při chybě přenosu dat. Kromě toho umí zamykání souborů, zachování historie při přejmenování či přesunu souboru a nabízí i řadu dalších vylepšení.
- I přes to, že řada starších firem SVN stále používá a nelze vyloučit, že se s ním ještě setkáte, je i tento nástroj značně zastaralý a jeho využití je z mnoha důvodů nevýhodné. Mezi hlavní nevýhody patří složitější vytváření větví, jeden bod selhání na centrálním serveru, obtížnější zálohování, pracnější řešení konfliktů (na úrovni souborů) apod.
- V dalších letech vznikly systémy jako GNU Arch, Monotone a Darcs. Jejich výhodou oproti SVN je distribuovaný přístup, tedy nepotřebují centrální server. Rovněž zavádějí využití kryptografických algoritmů a využití hashů k identifikaci souborů. Nicméně tyto systémy se z různých důvodů příliš neujaly. Např. Monotone byl tvořen s důrazem na integritu za cenu nízkého výkonu, kdy je kvůli kontrolám a ověřování řada operací pomalých.
- Od roku 1995 se na trhu objevilo i komerční řešení Perforce Helix. V roce 2016 mělo podíl na trhu 2 % (GIT měl v té době 87 % s rostoucí tendencí a SVN 6 % s klesající tendencí – viz https://rhodecode.com/insights/version-control-systems-2016). Vzhledem k jeho novému rozhraní, se kterým jej lze využívat jako GIT, se jím zde nebudeme zabývat.
- V roce 2005 vytvořil Linus Torvalds GIT jako nástroj pro verzování jádra Linuxu. Je distribuovaný, využívá hashe pro identifikaci verzí (inspirace v Monotone), umožňuje nelineární vývoj s vytvářením velkého množství větví, podporuje spojování větví se změnami ve stejných souborech a má i řadu dalších výhod.
- Ve stejném roce k němu Matt Mackall vytvořil alternativu – Mercurial. Základní principy jsou podobné – existují převodní tabulky příkazů, takže když umíte využít jeden, snadno zvládnete i druhý. Repozitář GITu lze konvertovat na repozitář Mercurialu a zpět bez ztráty dat. V některých aspektech se však liší, takže pokud chcete využívat pokročilejší funkce, vytvářet zásuvné moduly apod., je třeba je srovnat a zvolit dle konkrétního projektu.

- Od roku 2005 byl vyvíjen i GNU Bazaar, jehož 1. stabilní verze vyšla v roce 2007. Vznikl evolucí GNU arch, má komerční podporu od firmy Canonical a poměrně hodně funkcí. Umí pracovat distribuovaně, s centrálním serverem, nebo obě možnosti kombinovat. Umí importovat/exportovat historii z GITu a Mercurialu a má i řadu dalších funkcí. Oproti GITu je však výrazně pomalejší. Opět nalezneme převodní tabulky, takže kdo umí používat GIT, relativně snadno zvládne i Bazaar.
- Vzhledem k tomu, že nejpoužívanějším systémem pro správu verzí je momentálně GIT, a k tomu, že kdo jej umí využívat, nebude mít problém ani s ostatními nejvyužívanějšími moderními systémy, bude mu věnována celá příští přednáška.

Základní pojmy

- Abychom se mohli blíže zabývat GITem, je však třeba znát základní pojmy z oblasti systémů pro správu verzí a základní principy a postupy jejich využití.
- Prvním z těchto pojmů je „pracovní strom“ (working tree) – je to adresářová struktura se zdrojovými texty, ve které pracujeme. Tuto bychom měli i kdybychom žádný systém pro správu verzí nevyužívali.
- Dalším pojmem je „vyčkávací prostor“ (staging area), do kterého umist'ujeme změny které chceme uložit do repozitáře. Nemusí v něm být všechny změny v pracovním stromě – můžeme vybrat pouze některé soubory, které byly změněny, přejmenovány či smazány. Tyto změny spolu tvoří celek jako např. opravu chyby, novou funkci apod.
- Po tom, co umístíme změněné soubory k uložení do vyčkávacího prostoru, provedeme tzv. commit. Tím sadu změn ve vyčkávacím prostoru pojmenujeme a uložíme do repozitáře, čímž vznikne další verze.
- Sada po sobě následujících commitů pak tvoří větev. Větve umožňují paralelní vývoj. Jak se správně využívají uvidíme za chvíli.
- Samotný repozitář je pak databáze, která obsahuje historii projektu. Např. u GITu je uložena ve složce `.git` (u SVN na serveru).
- Klon repozitáře je kopie repozitáře s pracovním stromem.
- Holý (bare) repozitář je kopie repozitáře bez pracovního stromu. Typicky se jedná o tu kopii, která je umístěna na serveru a sdílena více vývojáři, kteří si ji odsud naklonují.
- Na sdílený server byste měli vždy umist'ovat holý repozitář. Pracovní kopie ve sdíleném repozitáři totiž svádí k nesprávným postupům a často dochází k zákeřným operacím, kdy si někdo místo naklonování repozitáře nakopíruje pracovní kopii a následně vrátí soubory zpět a provede commit, kterým vrací změny ostatních, protože pracovní kopie již není aktuální (pro repozitář je to commit, kterým děláte změny oproti aktuálnímu stavu ve větvi).

Pracovní strom (working tree)

- Nyní si to projděme s obrázky.
- Zdrojové soubory máme v pracovním stromě.

Repozitář (repository)

- Repozitář je databáze, která obsahuje sadu obsahů souborů.

Repozitář (repository)

- Sady těchto obsahů souborů společně s popisy změn tvoří commity.

Repozitář (repository)

- Commity obsahují i ukazatele na své předchůdce. Vzniká tak graf rodové historie commitů.

Repozitář (repository)

- Commity postupně přibývají.

Repozitář (repository)

- A každý obsahuje ukazatele na své přímé předchůdce.

Repozitář (repository)

- Pojd'me obrázek zmenšit a podívat se, jak může probíhat vývoj.

Repozitář (repository)

- Vytvoříme si novou větev a v ní naprogramujeme novou funkcionalitu, opravíme chybu apod.

Repozitář (repository)

- Až svoji práci dokončíme, je potřeba ji přidat do větve, ve které je aktuální verze programu, se kterou pracují ostatní. Spojení větví se anglicky nazývá „merge“.

Repozitář (repository)

- Pak můžeme vytvořit další větev.

Repozitář (repository)

- Nejmladší commit ve větvi je tzv. hlava (head či tip).

Repozitář (repository)

- Vícehlavový repozitář (multi-head) je takový, který má více větví, které nejsou spojené.

DAG (Directed acyclic graph)

- Commity v repozitáři tvoří orientovaný acyklický graf.

Reference

- Reference jsou ukazateli do tohoto grafu.

Reference

- Máme více typů referencí.
- Značky (tags) ukazují na konkrétní commit a obvykle se nemění. Využíváme je např. k pojmenování verzí programu, které jsou zveřejněny zákazníkům.

Reference

- Dalším typem referencí jsou větve (branches).

Reference

- Ukazatel větve ukazuje vždy na nejmladší commit ve větvi.

Reference

- Posledním typem reference je hlava (HEAD), což je ukazatel na aktuální checkout, tedy na verzi, kterou máme momentálně v pracovním stromě.

Workflows

- Workflow (pracovní tok) označuje postup práce s repozitářem. Jedná se o systematický přístup k využití repozitáře, ve kterém se jednotlivé činnosti typicky opakují.
- Základní tok je udělat změny v pracovním stromě, přidat tyto změny do vyčkávacího prostoru (lze využít škaredě zkomolené sloveso stageovat) a commitnout. Toto děláme opakovaně.
- Základní tok pak rozšiřujeme při práci s větvemi. Obvyklé postupy jsou:
  - Tradiční workflow, což je starší základní přístup, a
  - Tématické větve, které jsou novějším přístupem, který umožňuje dynamičtější vývoj a je velkou výhodou při práci ve více lidech.

Tradiční workflow

- Začneme tradičním workflow.

Tradiční workflow

- Při tradičním workflow máme vždy jednu hlavní vývojovou větev, tzv. „trunk“. V této větvi postupně děláme změny a vytváříme tak další verzi programu.

Tradiční workflow

- V určité chvíli vždy prohlásíme, že ve vývojové větvi máme vše, co potřebujeme pro vydání další verze. Pro tuto verzi vytvoříme novou větev.

Tradiční workflow

- Takto postupně vznikají jednotlivé verze.

Tradiční workflow

- Ve větvích jednotlivých verzí pak provádíme pouze opravy chyb. Tedy poskytujeme pro ně podporu. V určité chvíli pak vyhlásíme ukončení podpory a zákazníka tak donutíme přejít na novější verzi.

Tradiční workflow

- V hlavní vývojové větvi při tom pořád vyvíjíme další verzi, do které zařazujeme jak opravy chyb, tak i novou funkcionalitu.

Tématické větve

- Při využití tématických větví vytváříme jednu větev pro každé téma – tedy pro každou novou funkci, opravu chyby apod.

Tématické větve

- Protože na každý problém vytvoříme zvláštní větev, ve které typicky pracuje jeden člověk nebo málo lidí, kteří spolu úzce spolupracují, můžeme dělat časté commity a některé verze v těchto větvích nemusí být možné zkompilovat a spustit. Pokud něco pokazíme, máme spoustu dílčích verzí, ke kterým se můžeme vrátit, časté zálohy a je lepší přehled o stavu vývoje (v repozitáři se objevují menší jednotky práce).

Tématické větve

- Tématické větve mají krátkou životnost – slouží k řešení malých podproblémů, takže se v nich pracuje pouze omezený čas.

Tématické větve

- Po dořešení problému větev spojíme do master.

Tématické větve

- Podívejme se na animaci. Máme repozitář, kde vyvíjíme program. Produkční verze je v hlavní větvi označované master. Je to vlastně ekvivalent větve trunk v tradičním workflow.

Tématické větve

- Do programu chceme zavést novou funkcionalitu, tak pro ni vytvoříme větev.

Tématické větve

- Na této funkcionalitě nějakou dobu pracujeme.

Tématické větve

- Při tom objevíme chybu v produkční verzi, která nám brání ve vývoji. Tak ji rychle opravíme – vytvoříme si na to větev přímo z master.

Tématické větve

- Chybu rychle opravíme.

Tématické větve

- Vrátíme se do větve s vývojem naší funkce.

Tématické větve

- Připojíme si do ní opravu chyby a můžeme pokračovat ve vývoji.

Tématické větve

- Ale máme problém – přijde nám hlášení, že upstream padá. Upstream je globálně sdílená verze, kde probíhá vývoj projektu na nejvyšší úrovni (typicky u původního autora SW). U Linuxu je to místo, odkud si program stáhnou tvůrci distribučních balíčků, kteří si jej následně přizpůsobí pro danou distribuci.

Tématické větve

- Nejprve si musíme aktualizovat master z upstreamu, abychom měli vše v nejnovější verzi.

Tématické větve

- Vytvoříme větev pro opravu chyby.

Tématické větve

- Opravíme chybu.

Tématické větve

- Přepneme se do master.

Tématické větve

- Integrujeme/připojíme opravu chyby a když už jsme u toho, rovnou připojíme i opravu chyby, kterou jsme objevili dříve.

Tématické větve

- A můžeme se vrátit k vývoji naší funkce.

Tématické větve

- Dokončíme její vývoj.

Tématické větve

- Přepneme se do master.

Tématické větve

- A integrujeme.

Tématické větve

- Hotovo!

Tématické větve

- Hlavní výhody tématických větví jsou:
  - přehlednost (víme, kde co je),
  - možnost revizí nových funkcí před jejich připojením do master,
  - spojení můžeme odkládat dokud funkce není úplně hotová.

Systémy pro správu verzí

- Nyní se podíváme na základní rozdělení systémů pro správu verzí na centralizované a distribuované.

Centralizovaná správa verzí

- U centralizované správy verzí máme jeden sdílený server.

Centralizovaná správa verzí

- Na tomto centrálním serveru je celý repozitář.

Centralizovaná správa verzí

- Klient si načte pouze pracovní strom.

Centralizovaná správa verzí

- Veškeré operace s repozitářem pak logicky vyžadují přístup k serveru. A to jak čtení ...

Centralizovaná správa verzí

- ... tak i zápis.

Centralizovaná správa verzí

- Všichni klienti pracují s jednou instancí repozitáře. Každý má pouze pracovní strom.

Centralizovaná správa verzí

- Centrální server je pak tzv. „jeden bod selhání“ (single point of failure), protože když selže, ztratíme celou historii projektu i některé verze, které zrovna nikdo neměl v pracovním stromě.
- Centrální server je současně úzkým místem z hlediska výkonu, protože všichni klienti pracují s jedinou instancí databáze.

Decentralizovaná správa verzí

- U decentralizovaného (distribuovaného) systému pro správu verzí máme opět sdílený server s repozitářem.

Decentralizovaná správa verzí

- A klienta.

Decentralizovaná správa verzí

- Klient provede operaci „clone“.

Decentralizovaná správa verzí

- A nakopíruje si celý repozitář s kompletní historií, ze kterého si pak načte pracovní strom.

Decentralizovaná správa verzí

- Veškeré operace pak dělá lokálně.

Decentralizovaná správa verzí

- Když potřebuje získat změny od ostatních vývojářů, provede operaci „pull“.

Decentralizovaná správa verzí

- Touto operací stáhne aktuální změny z větve na serveru do svojí lokální větve.

Decentralizovaná správa verzí

- Následně provede „push“ ...

Decentralizovaná správa verzí

- ... čímž odešle změny na server.

Decentralizovaná správa verzí

- Další klienti mají také klony repozitáře.

Decentralizovaná správa verzí

- A rovněž přijímají a odesílají změny.

Decentralizovaná správa verzí

- Protože má každý klient celou kopii repozitáře, může se chovat i jako server a sdílet jej někomu dalšímu. Centrální server ani nikdo další pak nemusí všem povolit zápis.

Decentralizovaná správa verzí

- Kdokoliv si pak může stáhnout projekt, ale pouze stanovený tým vývojářů může měnit stav na centrálním serveru. Osataní mohou posílat změny přes ně.

Decentralizovaná správa verzí

- Vývojář nemusí nasdílet svůj repozitář a čekat, co mu tam kdo udělá. Může čekat, až mu někdo napíše, že něco udělal a je to připravené k integraci (tzv. „pull request“), a může si to od něj stáhnout.

Decentralizovaná správa verzí

- Případně mohou mezi sebou vzájemně provádět všechny operace jako se serverem.

Výhody decentralizované správy verzí

- Hlavními výhodami decentralizované správy verzí jsou:
  - rychlost (hodně operací se dělá lokálně),
  - můžeme dělat malé časté commity a i když v naší verzi není funkční kód, nepokazíme to nikomu jinému,
  - můžeme pracovat i offline,
  - nemáme jeden bod selhání,
  - v podstatě nemusíme řešit zálohování, protože každý vývojář má kompletní zálohu.

Topologie distribuovaných VCS

- U distribuovaných systémů pro správu verzí pak můžeme mít různé topologie. Typické jsou 4:
  - Jeden vývojář
  - Centralizovaná
  - Integrátor
  - Diktátor a poručíci

Jeden vývojář

- Jeden vývojář založí projekt a pracuje na něm lokálně u sebe.

Jeden vývojář

- Má svůj pracovní strom.

Jeden vývojář

- A lokální repozitář.

Zveřejnění kódu

- Pak se rozhodne svůj kód zveřejnit a sdílet s ostatními.

Zveřejnění kódu

- A natlačí jej na server.

Zveřejnění kódu

- Na serveru máme holý repozitář, do kterého může vývojář tlačit další verze.

Zveřejnění kódu

- Přijde druhý vývojář.

Zveřejnění kódu

- Naklonuje si repozitář.

Zveřejnění kódu

- A může pracovat ...

Zveřejnění kódu

- Patch (sadu změn) pak může původnímu autorovi poslat e-mailem (to příliš nedoporučuji – GIT to sice umí, ale bývají s tím problémy – např. s kódováním).

Zveřejnění kódu

- Nebo si koupí vlastní server či založí účet na serveru.

Zveřejnění kódu

- Natlačí na něj svoje změny.

Zveřejnění kódu

- A pošle „pull request“ původnímu autorovi, který si změny zkontroluje, integruje a natlačí na svůj server.

Centralizovaná topologie

- U centralizované topologie všichni vývojáři pracují s jedním sdíleným serverem, na kterém je holý repozitář.

Centralizovaná topologie

- K tomu můžeme mít speciální server, kde se provádí automatické sestavování a testování.

Centralizovaná topologie

- Můžeme mít i zvláštní server pro zálohování apod.

Integrátor

- Ne vždy však chceme, aby si každý mohl dělat co chce a integrovat libovolné změny do master. Toto řeší topologie nazvaná Integrátor.

Integrátor

- Opět máme centrální server ...

Integrátor

- ... ke kterému můžeme mít specializovaný server pro sestavování a testování apod.

Integrátor

- Z centrálního serveru mohou všichni číst.

Integrátor

- Ale pouze 1 vývojář, tzv. „strážce brány“ (gate keeper) může zapisovat.

Integrátor

- Tedy pouze tento vývojář má právo udělat push.

Integrátor

- Ostatní pracují se svými klony repozitářů.

Integrátor

- Mohou je sdílet přes svoje účty na serveru.

Integrátor

- A to i mezi sebou.

Integrátor

- Po dokončení nějakého úkolu vždy pošlou pull request integrátorovi, který si jejich práci stáhne, zkontroluje a integruje.

Integrátor

- Výsledek pak natlačí do hlavního repozitáře.

Integrátor

- Automatický systém pak celý program sestaví a může spustit testy.

Diktátor a poručíci

- Nejsložitější topologie je Diktátor a poručíci. Tato se využívá např. pro vývoj jádra Linuxu.

Diktátor a poručíci

- Linus Torvalds vytvořil 1. verzi jádra.

Diktátor a poručíci

- Všichni si naklonují jeho repozitář.

Diktátor a poručíci

- Poručíci (lieutanants) jsou zodpovědní za jednotlivé podsystémy, pro které si vytvoří repozitáře.

Diktátor a poručíci

- S repozitáři podsystémů mohou pracovat jednotliví vývojáři.

Diktátor a poručíci

- Změny buď tlačí přímo do těchto repozitářů.

Diktátor a poručíci

- Nebo sdílejí přes Github a posílají pull requesty poručíkům.

Diktátor a poručíci

- Poručíci posílají diktátorovi pull request.

Diktátor a poručíci

- A diktátor spravuje centrální repozitář.
- Tímto končíme s obecnými principy systémů pro správu verzí a na příští přednášce se blíže podíváme na konkrétní systém pro správu verzí – GIT.

Úvod do plánování projektu

- A nyní se podíváme na rychlý úvod do plánování projektu. Tedy jak si rozvrhnout a rozdělit práci na projektu.

Časové rozložení práce na projektu

- U každého projektu máme obvykle daný termín, do kterého jej musíme dokončit nebo dosáhnout určitého dílčího cíle. Předběžné rozvržení prací na projektu nám umožní získat přehled o tom, kolik času máme na kterou činnost a jak intenzivně je třeba pracovat.
- Obvyklé je, že v prvních fázích projektu jde práce pomaleji a ke konci není dostatek času. Ve všech fázích je proto třeba uvažovat i rezervy, přičemž u prvních fází projektu by tyto rezervy měly být menší a u posledních větší. Při plánování rezervy je třeba zohlednit i složitost dané fáze projektu.

- Velmi důležité je to, aby byl plán ve všech částech dostatečně podrobný. Pokud plán nevytváří specialista, který má s touto činností zkušenosti, je dobré, když plán vytvářejí a případně průběžně upřesňují všichni členové týmu. Lze tak předejít situaci, kdy plán vytvoří vedoucí, který si svoji práci, které dobře rozumí, rozvrhne podrobně a ostatním jen velmi zhruba. Pro ostatní členy týmu je pak obtížnější sledovat stav práce na jejich částech projektu i stav práce na těch částech, na které svou prací navazují.
- Samotné plánování sice u malých projektů můžeme vyřešit dohodou nebo na papíře, ale u větších projektů je již třeba využít nějaké podpůrné prostředky. Jedním z nejčastěji využívaných programů pro podporu plánování prací na projektu je Microsoft Project.
- Ať už však plán vytvoříte jakkoliv, není to něco, co se odloží bokem a začne se pracovat. Plán je potřeba průběžně sledovat a v případě potřeby (např. když už je jasné, že je nereálný) jej upravit. Nemělo by smysl se dívat na plán, kde je 1/2 termínů v minulosti nesplněných a nevíme, co máme do kdy udělat – termíny je potřeba posunout tak, aby byly reálné. Nelze-li to, projekt se zpozdí a musíme počítat s problémy.
- Častou chybou je také to, že plán sleduje pouze vedoucí. Členové týmu potom nevidí, kdy mají začít pracovat a kdy mají mít práci hotovou. Když pak někomu vedoucí pošle e-mail, že už je v polovině času na svou práci, a daný člověk teprve z tohoto e-mailu zjistí, že už měl dávno pracovat, už může být pozdě. Plán by proto měl být vystavený někde, kde jej jednotliví členové týmu vidí, a každý by měl sledovat jeho aktuální stav. U větších projektů jsou pak kromě plánu projektu i plány jednotlivých fází či menších částí a řešitelé mohou mít přístup pouze k částem, které se jich týkají. Využívají se také plánovací kalendáře vygenerované z plánu projektu apod.
- Nyní Vám ukážu pouze velmi krátkou ukázku využití Microsoft Project a pokud by Vás tato problematika zaujala, můžete si ve 3. ročníku nebo v magisterském studiu zapsat předmět Management projektů, ve kterém se probírá detailně.

MS Project

- Zde vidíte vytváření nového projektu. Vytvoříte soubor MS Project, vložíte projekt mezi úkoly, aby se stal kořenem stromu úkolů, a nastavíte, od kdy se bude plánovat.
- Projekt plánujeme od začátku (nakonec se dozvíme, kdy jej zvládneme dokončit a jestli dodržíme termín), nebo od konce (nakonec se dozvíme, kdy musíme začít, abychom to stihli do stanoveného termínu).

Ganttův diagram

- Na tomto obrázku je Ganttův diagram, který je základním diagramem pro plánování projektů. Osa x je časová osa a na ose y jsou jednotlivé úkoly. V diagramu vidíte termíny, dobu trvání jednotlivých úkolů, vztahy mezi úkoly, přiřazené osoby a prostředky apod. Kromě úkolů jsou v diagramu i milníky, což jsou body v čase a mají tedy nulovou dobu trvání. Jsou to okamžiky, ve kterých se má zkontrolovat splnění nějaké podmínky, dokončení úkolu nebo fáze projektu.

Kalendář

- K dispozici je také plánovací kalendář.

Seznam zdrojů

- Pokud je někdo přetížený, zjistíte to i v seznamu zdrojů. Kromě lidských zdrojů mohou být také materiální a investiční. Lidské zdroje potom mohou být abstraktní či konkrétní. Abstraktní zdroj může být skupina lidí určitého zaměření.

Diagram zdrojů

- V diagramu zdrojů vidíte vytížení jednotlivých zdrojů v jednotlivých dnech. Přehledně zde vidíte přesčasy a chybné přiřazení úkolů, kdy má někdo pracovat na 200 a více procent.

Sledování stavu projektu

- MS Project umožňuje také sledování stavu projektu. V základním zobrazení vidíte, které úkoly jsou splněny.

Sledovací Ganttův diagram

- Při sledování projektu obvykle nestačí vědět, které úkoly jsou splněné, ale je potřeba vědět, v jakém stavu jednotlivé úkoly jsou. K tomu slouží sledovací Ganttův diagram.

Rozdělování práce a zodpovědností v týmu

- Ve firmě je rozdělení práce často dané managementem nebo znalostmi jednotlivých členů týmu. Tým má obvykle určité jádro, což je skupina dobře spolupracujících lidí s aktivním přístupem a ostatní členy, kteří byli k týmu přiděleni a jejich přístup k práci není předem známý.
- Máte-li možnost si práci rozdělit mezi sebou a jste-li v jádře týmu, vždy je dobré aby práce na prvních fázích projektu dostali členové z jádra týmu. I když se může jednat o jednodušší podúkoly, začátek, kdy ještě nic není hotové, bývá nejtěžší a slabšímu členovi týmu může trvat dlouho. Rychlý začátek projektu je důležitý, aby se nenabíral skluz do dalších fází projektu.
- Pokud slabší člen týmu svoji práci neudělá, někdo z jádra týmu ji obvykle musí udělat za něj. Pokud se jedná o pozdější fázi projektu, kdy už má daný člen týmu svoji práci hotovou a v projektu není časový skluz, je to zvládnutelnější, než když má někdo dělat svoji práci a současně práci z předchozí fáze, na kterou měl navazovat.

Plánování problémů

- Při práci na projektu mohou nastat různé problémy: někdo onemocní, dá výpověď, nebude se mu chtít do práce a bude zdržovat apod. Problémy mohou být i technického charakteru, kdy někdo nebude mít na čem pracovat (shoří mu PC) nebo dojde ke ztrátě dat.
- Na tyto problémy je třeba být připraven a počítat s nimi již od prvotního plánování prací na projektu. V této fázi se však zatím nejedná o problémy, ale o pouhá rizika.
- Rizika navíc v průběhu projektu vznikají a zanikají, takže nestačí je pouze jednou naplánovat, ale je třeba s nimi stále pracovat. Touto problematikou se zabývá disciplína nazvaná řízení rizik, kterou se můžete naučit v rámci předmětu Management projektů.

Jednoduché řešení pro malý tým a projekt

- Pokud neumíte s riziky pracovat, nebo se to v rámci malého projektu pro 4–6 lidí nevyplatí, můžete si pouze vytvořit obecné časové rezervy. Rezervu vytvoříte např. tak, že na začátku některý spolehlivý člen týmu nedostane žádnou práci. Ostatní tak mají více práce a daný člen týmu může pomáhat tomu, kdo to zrovna potřebuje. Pokud někdo onemocní, automaticky převezme jeho práci (pokud neonemocní sám).
- Také je výhodné počítat rezervu na celý projekt a při plánování stanovit termín dokončení před oficiálním termínem.
- Pokud se v některé fázi projektu dostanete do předstihu, je dobré posunout následující termíny o tento předstih a zvětšit tak rezervu na konci projektu. Pokud se další fáze zpozdí, můžete termíny posunout zpět na výchozí. Takto lze korigovat chyby v plánování rezerv na jednotlivé fáze projektu.

Úvod do návrhu software

- A nyní přejdeme k poslední části této přednášky, kterou je úvod do návrhu software.
- Nepůjdeme zde do hloubky – na to jsou jiné předměty. Podíváme se pouze na rychlý přehled toho, jak začít a co asi by takový návrh měl obsahovat. Zájemci najdou další informace v literatuře a v předmětech jako Analýza a návrh informačních systémů.

Úvod do návrhu software

- Začnete-li živelně programovat bez jakéhokoliv návrhu, typicky děláte i věci, které nejsou potřeba, musíte hodně věcí předělávat a ve více lidech se Vám může stát, že jednotlivé části nepůjde spojit.
- Libovolně malý program je vhodné promyslet a navrhnout, abyste pokud možno dělali přesně to, co je potřeba, a neztráceli čas předěláváním. Oprava chyby v návrhu je mnohem jednodušší než předělávání hotové práce. Dobrý návrh rozhraní pak při správné implementaci zajistí bezproblémové spojení jednotlivých částí.
- U velkých programů a systémů je návrh nutností.

Prerekvizity návrhu

- Než začneme pracovat na návrhu software, musíme vědět, co budeme navrhovat – potřebujeme specifikaci. Zákazník nám však nedodá přesnou specifikaci, ale pouze svoje požadavky a přání. Nemusí mít dostatečnou představu o celém výsledku, ale soustředí se pouze na klíčové vlastnosti, které pro něj představují motivaci k nákupu nového software. Požadavky mohou být i protichůdné nebo chybné, protože si zákazník nedokáže představit, jak přesně to bude fungovat.
- Požadavky je tedy třeba analyzovat, upřesnit a vytvořit specifikaci, kterou nám zákazník následně schválí.
- Specifikace musí obsahovat zejména role uživatelů, popis funkčnosti, popis uživatelského rozhraní, výkonnostní požadavky (počet uživatelů a HW, na kterém musí program fungovat) apod.
- Po analýze požadavků musí následovat analýza existujících řešení. Nemá smysl znovu objevovat kolo a vytvářet něco, co už existuje. Pokud zákazník neví o konkurenčním řešení, v průběhu vývoje se o něm může dozvědět a bude srovnávat cenu a funkcionalitu. Pak typicky musíme být schopní dodat lepší či levnější produkt, jinak budeme mít problémy a negativní ohlasy.
- Často má smysl pouze rozšířit existující řešení. Vyhledáme-li vhodné knihovny, můžeme ušetřit čas a peníze.

Návrh

- Máme-li schválenou podrobnou specifikaci, můžeme začít s návrhem. Návrh můžeme dělat pouze v textové formě, ale to často není nejvhodnější řešení, protože jak se říká: „obrázek vydá za tisíc slov“ a grafická reprezentace je typicky mnohem přehlednější. Je vhodné držet se nějakého zavedeného formalismu, protože pak je v odborné komunitě obecně srozumitelný. Doporučuji Vám využívat jazyk UML.
- Jazyk UML jste již probírali v Úvodu do softwarového inženýrství a budete s ním pracovat i v dalších předmětech. Tím se zde tedy zabývat nebudeme. Spíše se podíváme na to, jak jej při návrhu prakticky využít.
- Začneme tím, co vůbec v jazyce UML můžeme modelovat. Základní jsou 2 modely.
- Model obchodního (business) systému vytváříme zejména u velkých projektů. Popisuje organizační model firmy a procesy ve firmě, přičemž zahrnuje pouze ty aspekty, které budeme později potřebovat pro návrh IT systému. Umožňuje nám dobře porozumět tomu, kým a jak bude výsledný systém využíván. Při jeho tvorbě můžeme objevit chybějící či nesprávné části specifikace a vyhnout se různým problémům, které na 1. pohled nejsou zřejmé. Některé procesy lze také optimalizovat, protože ne vždy má smysl vše v elektronické podobě řešit tak, jak se to řešilo bez využití počítačů s papíry.
- Druhým modelem je pak model IT systému, který popisuje vytvářený software a jeho nasazení na daném hardware. Tento model se zabývá pouze činností výpočetních prostředků – tedy nepopisuje další činnosti osob, které se software pracují (např. ne „prodavač vezme zboží z pásu, zadá do pokladny a vloží do košíku“, ale „prodavač zadá množství a EAN kód“).

Model obchodního systému

- Nyní se blíže podíváme na model obchodního systému. Na nejvyšší úrovni má 2 pohledy – externí a interní.
- Externí pohled zahrnuje především:
  - diagram případů použití – popisuje, kdo co dělá,
  - diagram aktivit – popisuje postupy jednotlivých činností (případů použití) a
  - diagram sekvence – také popisuje případy použití, ale ve formě popisu určité sekvence interakcí. Umožňuje tak ukázat spolupráci jednotlivých objektů, které na daném případu použití participují, v čase.
- Interní pohled zahrnuje:
  - diagram balíčků – zobrazuje jednotlivé součásti systému a jejich zařazení do organizačních jednotek (např. údržbář a uklízečka patří pod správu areálu),
  - diagram tříd – vyobrazuje vztahy mezi jednotlivými zaměstnanci (pracovními místy), objekty (např. že prodejka je podtypem účetního dokladu) a vnějšími systémy,

  - diagram aktivit – popisuje činnosti zaměstnanců, zařízení, informačních systémů a jiných součástí organizace, přičemž se zaměřujeme na interní procesy. Zatímco diagram aktivit v externím pohledu popisuje např. postup zákazníkova nákupu, v interním se zabývá činnostmi pokladní (založí nový doklad, ...).

Model IT systému

- Model IT systému pak popisuje vytvářený SW. Existuje více přístupů k jeho vytváření – podíváme se na ten z knihy UML 2.0 in Action, který obsahuje 5 pohledů.
- Externí pohled je pohledem zákazníka. Jeho základem je diagram případů použití (role uživatelů a co mohou dělat), který lze doplnit popisy jednotlivých případů použití, případně doplněnými i diagramy sekvence.
- Strukturální pohled popisuje vnitřní strukturu systému. Jeho základem jsou diagram balíčků (součástí systému) a diagram tříd.
- Behaviorální pohled popisuje vnitřní chování systému stavovým diagramem (např. sekvence obrazovek při určité činnosti a přechody mezi nimi).
- Interakční pohled popisuje interakce jednotlivých komponent uvnitř systému. Najdeme zde diagram sekvence, diagram komunikace, přehledový diagram interakce či diagram časování.
- Pohled nasazení pomocí diagramu nasazení popisuje, který SW bude nasazen na kterém HW.
- V UML najdeme i další diagramy, které u výše uvedených pohledů nejsou přímo uvedeny a slouží typicky k dovysvětlení detailů:
  - Diagram objektů popisuje stav systému v určitém časovém okamžiku pro lepší vysvětlení konkrétní situace, co může nastat.
  - Diagram složené struktury zobrazuje sestavu instancí objektů existujících za běhu programu a spolupracujících na dosažení určitého cíle.
  - Diagram kolaborace zobrazuje strukturu spolupracujících prvků, z nichž každý provádí nějakou specializovanou funkci, které společně vytvářejí nějakou požadovanou funkcionalitu.
  - Diagram komponent ukazuje komponenty systému a rozhraní mezi nimi.
- Zmíním zde i ER (Entity relationship) diagram, který je velmi oblíbený, ale není součástí UML. Tento využijeme typicky při návrhu databáze.

Literatura

- Více o jednotlivých pohledech a diagramech se dočtete v této knize. Stručný učební text o Praktickém využití UML 2.0, který jsem s využitím této knihy dělal jako projekt do předmětu SID, najdete na serveru ivs.

Jak začít?

- Teď jsme viděli, jak udělat velké modely, využít spoustu diagramů, ... Ale kde začít? A potřebujeme tohle všechno, když jdeme dělat malý program?
- Vše asi potřebovat nebudeme, ale návrh bychom měli vytvořit vždy. Je tedy nutné určit, co budeme potřebovat.

Volba potřebných částí návrhu

- Diagram případů použití je vhodné vytvořit vždy. Popisuje, jak se náš SW bude používat, a nemá smysl vytvářet něco, co nelze použít.
- Pokud program pracuje se složitými datovými strukturami, měli bychom vytvořit strukturální pohled. Začneme diagramem balíčků, kde si systém rozvrhneme na jednotlivé části. V diagramu tříd pak popíšeme, z čeho budou složené. Pokud z diagramu tříd není jasné, jak se budou používat, můžeme jej doplnit i nějakými diagramy objektů.
- Pokud program provádí složité činnosti, měli bychom vytvořit behaviorální pohled. Pokud bude složitější postup, kterým prochází uživatel, můžeme využít stavový diagram (obrazovky a přechody mezi nimi). Pokud jsou složité interní procesy a algoritmy, co se provádějí automaticky, lze s výhodou využít diagramy aktivit.
- Je-li v procesech zapojeno více SW komponent, je vhodné vytvořit interakční pohled a zabývat se tím, jak budou spolupracovat.

- Když je program složený z více částí, které budou vyvíjeny odděleně, je potřeba navrhnout rozhraní jednotlivých komponent a to, jak budou komponenty propojeny. Zde využijeme zejména diagram komponent a diagram složené struktury. Všechna rozhraní musí být předem dobře definovaná, protože dodatečné předělávání je velmi problematické (i s ohledem na to, že nemusí být jasné, na čí straně je zodpovědnost za vzniklý problém).
- Bude-li složité nasazení systému (různý HW, různý SW dle umístění, ...), vytvoříme pohled a diagram nasazení.

Návrh shora dolů

- Návrh můžeme dělat shora dolů – tedy nejprve vytvořit přehledové diagramy, abychom získali celkový pohled na systém, a pak tam, kde diagram na nejvyšší úrovni nestačí k pochopení, z čeho se SW bude skládat, jak má fungovat a jak mají jednotlivé části spolupracovat, vytváříme diagramy popisující detaily.

Návrh zdola nahoru

- Druhým přístupem je návrh zdola nahoru. Nejprve vytvoříme pouze základní přehledové diagramy, abychom věděli, co jdeme vytvářet, ale následně se zaměříme na menší části, které lze brzy implementovat a otestovat.
- Vytvoříme tedy nejprve diagramy na nižší úrovni jako např. diagram tříd a poté příbuznou funkcionalitu seskupíme do balíčků a doplníme abstraktnější diagramy.

Míra detailu

- Jak detailní návrh vytvářet? Návrh musí být dostatečně detailní, aby pokryl vše důležité, co není jednoznačné. Vývojář obvykle pracuje tak, aby se vydal cestou nejmenšího odporu z hlediska psaní kódu. Z hlediska uživatele pak funkcionalita může být nelogická. Návrhář se musí zaměřit na použití a nesmí při tom příliš uvažovat o implementaci, přičemž návrh by měl být dostatečně detailní, aby implementace nezměnila zamýšlený průběh případů použití.
- Nelze očekávat, že se dva vývojáři dohodnou na vhodném rozhraní, takže rozhraní musíme navrhnout vždy a dostatečně detailně.
- Nelze očekávat, že vývojář správně navrhne část databáze tak, aby ji bylo možné efektivně využít i mimo jím vytvářené komponenty. Veškeré sdílené datové struktury je tedy nutné kompletně navrhnout.
- ...
- Nesmíme to však přehnat a tvořit zbytečně detailní návrh. Některé věci vývojáři zvládnou udělat i bez návrhu a jejich navrhováním bychom tak zbytečně plýtvali drahocenným časem.
- Nemá smysl popisovat, že se připojíme k databázovému serveru, vybereme databázi, sestavíme dotaz, provedeme dotaz apod. – to vývojář zvládne sám.
- Obdobně pokud navrhneme detailní rozhraní komponenty a její vnější chování, nemá smysl se zabývat návrhem jejích vnitřních procesů.

Shrnutí

- A na závěr několik hlavních myšlenek z této prezentace:
  - Dobrá a efektivní komunikace v týmu je základem týmové spolupráce.
  - Pro sdílení dat v týmu je třeba zvolit vhodné prostředky a využívat je k vhodným účelům.
  - Vhodné naplánování projektu umožní předejít problémům s jeho dokončením v daném termínu.
  - Návrh je důležitou součástí procesu vývoje SW. Pro vytvoření kvalitního a použitelného SW je důležité tuto fázi nepodcenit.