Zkušenosti z tvorby Softwarového projektu

0. Nejdůležitější pravidlo

Je velmi **dobrý nápad si pročíst zkušenosti ostatních týmů z tvorby projektů**. Sice se to třeba před započetím projektu může zdát jako hloupost, protože „každý tým měl dost specifický projekt a zadání“, není tomu tak. Většina popsaného jsou důležité body, které mohou leckdy zachránit týmu kůži. Po neobhájení bude pozdě na „Já vám to říkal“ :)

1. Vypsání projektu

- Na počátku je student, který nemá tušení, co bude dělat za projekt, ale ví, že si nějaký musí zapsat. Možností, **jak najít vhodný projekt**, je mnoho:
    - Zeptat se pár spolužáků a dát dohromady tým, dále si vymyslet téma a přijít za člověkem z katedry, který má k danému tématu nejblíže a/nebo se kterým mají dobré zkušenosti třeba z cvičení/bakalářky. Seznámit ho s návrhem, a pokud vše bude OK tak do měsíce se může pracovat.
    - Projít si vypsaná zadání na webu projektové komise/SISu/diskusním fóru, obeslat maily se zájmem připojit se do týmu.
- **Téma volte rozumně obsáhlé**, aby bylo dosti zajímavé pro vypsání, ale volte jej zase minimalistické v tom smyslu neslibovat to, co nejde splnit. To, co se napíše do specifikace, musí být splněno, to, co tam není, může být přidaná hodnota.
- Po zadání máte **9 kalendářních měsíců na zhotovení** a poté jste povinni se přihlásit na první možný termín obhajoby, myslete na to.

2. Příprava práce

- Řekněme, že už máme tým a téma, teď je třeba vyřešit další důležité body. **Jazyk a cílová platforma**. Zde je třeba dát mimořádný pozor, protože špatný výběr je cestou do pekel. V současné době to bude nejčastěji asi C# a .NET, Java, popř. C++. U výběru platformy je třeba zohlednit především zkušenosti jednotlivých členů týmu, dále vhodnost pro dané zadání (web/klient+server/vlákna/distribuovanost?) a také vyspělost dané technologie. Je velmi na váženou pouštět se do něčeho, co v době zahájení projektu není ve stabilní verzi, ale jen třeba jako CTP/beta. Opravdu se do toho nepouštějte, ač to může znít lákavě! Bývá to většinou přítěží než přínosem.

V našem projektu jsme se pouštěli do Silverlightové aplikace psané částečně v C# a tehdy vznikajícího jazyka F#. Silverlight byl tehdy ve verzi 2 beta, F# jen jako CTP a vzájemná podpora oficiálně neexistovala, řešila se pomocí manuální editace kompilačních souborů na základě rad ve fórech. Při odevzdání projektu náš projekt běžel nad Silverlightem 3 a o něco novější CTP stále nehotové verzi F# jazyka. Po většinu doby vývoje projektu např. neexistovala možnost debugovat naše F# knihovny a to nebylo zdaleka vše…

- Určete si **systém pro verzování dat a také spolehlivý bugtracker**. Jako verzovací systém je dobrou volbou klasické Subversion, nabízí se i TFS systém ve Visual Studiu, popř. GIT. Druhým bodem je bugtracker, který

taktéž není radno podceňovat. Spoléhat se na seznam chyb v nějakém texťáku je hloupost. Otázkou také je, kde toto hostovat. Nabízí se několik možností zdarma, které ale většinou vyžadují otevřenost zdrojových kódů, třeba Assembla, placené varianty poté nabízejí uzavřené úložiště.

My jsme nejprve používali TFS běžící na Malé straně, ale ten se ukázal jako nepoužitelný v možnostech přizpůsobení a komunikaci se správcem. Dále jsme používali TFS na Codeplexu, ale ten byl dosti pomalý a jak se později ukázalo, tak přidružený bugtracker byl také prakticky nepoužitelný. Nakonec jsme přešli na SVN a bugtracker Trac umístěný na Assemble, se kterým nebyl problém.

- **Dohodněte se a dodržujete společný codestyle!** Podceňované, ale důležité pravidlo. Dost to kazí morálku a čitelnost kódu, když každý píše závorky a názvy proměnných jinak a navzájem si to lidé opravují.

U nás se osvědčil plugin ReSharper do Visual Studia, který umožňuje nastavit společný codestyle a ten sdílet v projektu mezi uživateli.

- **Týmová komunikace je základ**. Dobré jsou častě schůzky s vedoucím min. 1x týdně. Navrhněte si den a hodinu v týdnu, kdy se probere progress a problémy. Na schůzkách pořizujte zápisy, které se budou archivovat na webu na dohodnutém místě. Dohodněte se na společném komunikačním kanálu pro řešení problémů během týdne.

Jako komunikační kanál se nám osvědčil Skype konferenční chat, popřípadě konferenční hovor u vážnějších témat. Na ukládání dokumentů a jako mailová konference se nám hodily Google groups.

3. Práce na projektu

- **Nejdříve přemýšlejte**! Určete si, zda pojedete podle nějaké metodiky programování typu Waterfall, Scrum, agilní programování, test-driven programování, nebo si minimálně o těchto postupech něco pročtěte a nechte se inspirovat! Nehrňte se hned do psaní, věnujte třeba dva-tři týdny návrhu architektury, nebastlete kód „jen aby to fungovalo“.
- **Rozdělte si práce podle logických celků**, tj. třeba GUI, object model, aplikační logika, testy, ale i přesto by každý měl vědět důležité části z ostatních částí. Neměla by se v aplikaci objevit část, které rozumí jen jeden člověk, pak je problém, když tento člověk bude 14 dní pryč a práce bude stát „protože to nefunguje“.
- **Určete si v týmu experta** na danou platformu, který bude mít rozhodovací slovo v architekturních záležitostech
- **Commitování změn by mělo mít svá pravidla**. Mělo by být pravidlem necommitovat to, co nejde zkompilovat a spustit. Veškeré změny by měly být náležitě popsány, dále v komentářích i zmínit vyřešené bugy/tickety.
- **Pište dokumentaci průběžně :)**. Je to otřepaná fráze, ale hodí se. Základem je především komentovat kód tak, jak to nativně podporuje vývojové prostředí, tj. buď formou Doxygenu, nebo xml dokumentace ve Visual Studiu. Mělo by být pravidlem necommitovat kód, který není řádně okomentovaný. Dále se hodí i **psát specifikační dokumenty či náčrtky** o tom, jak která část aplikace funguje, tj. jak funguje práce s filesystémem, jak se volají asynchronní operace, jak funguje model-view-viewmodel aplikace či jiné použité a dohodnuté design patterny.
- Je běžné, že **v týmu budou tahouni i lenoši**. Obvykle se najde člověk, který jen napíše svůj kus a dál se jen veze a někdo, kdo to všechno táhne dopředu. Jak předcházet problémům je složitý proces, základem je ale komunikace, vhodné rozvrhování úkol a kontrola jejich splnění.

4. Obhajoba projektu

- **Hlídejte si termín obhajoby**. Je třeba se přihlásit na obhajobu do daného termínu, dále odevzdat vše potřebné na sekretariátu a samozřejmě připravit se na obhajobu.
- **Programátorskou dokumentaci** byste měli mít v ideálním případě průběžně hotovu - vygenerovanou z kódu. **Doplňková programátorská dokumentace** by měla být psaná ručně tak, aby po přečtení mohl nový člověk přijít do týmu a pokračovat v práci, rozsahem tak minimálně 30-40 stránek.
- **Uživatelská dokumentace je základ**, podle kterého bude váš oponent hodnotit váš projekt. Měla by být dostatečně názorná, aby šlo pochopit k čemu je aplikace dobrá, jak se instaluje, jak se ovládá, co má vliv na co, kam kliknout pro jaký efekt, defakto blbuvzdorná, rozsahem také minimálně 40+ stránek.
- Jakmile se odevzdá projekt, zbývá asi 14 dní do obhajoby. Tato doba je zásadní pro úspěch. Zbývá ještě čas pro doladění drobností do obhajoby, příprava prezentace a především **předvedení projektu oponentovi**. Domluvte si s ním schůzku, ukažte mu, jak váš projekt skvěle funguje, odpovídejte mu na jeho otázky, pamatujte si, co se mu nezdálo, a využijte těchto informací pro finální obhajobu.
- **Prezentace na obhajobě** a její kvalita má asi z poloviny vliv na úspěch vaší obhajoby. Špatná prezentace potopí výborný projekt, povedená prezentace může zachránit nedokonalý projekt. Na předvedení máte 20 minut. Připravte si např. PowerPointovou prezentaci o cca 10 slidech a dále i malé předvedení aplikace. **Vyzkoušejte si prezentaci několikrát předem**, ideálně přímo v místnosti, kde se bude konat. Ověřte si kompatibilitu notebooku a projektoru, jak jsou čitelné slajdy, atd… Ať vás ostatní členové týmu ohodnotí po generálce a ať odhalí případní nejasnosti. Zkuste také říct obhajobu kamarádům, kteří nevědí, o co jde. Připravte si odpovědi na jejich otázky a upravte prezentaci podle jejich připomínek.
- Veškeré hádky a argumenty mezi členy a třeba i vedoucím si nechte až na moment po obhájení a zapsání do indexu :)