Tvorba textové hry

Životní cyklus libovolné aplikace se skládá z následujících fází:

- **analýza** ohraničení, co bude vstupem a výstupem aplikace. Pokud je popis vágní, vyplatí se také výslovně specifikovat, co aplikace řešit nebude. Součástí analýzy mohou být UML diagramy aktivit a užití.
- **návrh** plán implementace aplikace, jednotlivé úkoly natolik podrobné, že je možné určit, jak dlouho bude trvat jejich vývoj. Neměl by obsahovat žádné nejistoty. Čas je vhodné vynásobit individuální konstantou (např. ×2). Návrh by měl obsahovat také UML diagramy tříd.
- **implementace** vlastní programování dle návrhu. Cca třetina až polovina času.
- **testování** procházením všech cest a možných vstupů diagramů aktivit. Některá vývojová prostředí toto automatizují generováním jednotkových testů (*unit testing*), někteří doporučují tímto dokonce začít (*test driven development*)

Základní metodiky vývoje

Agilní metodiky

Používají se tam, kde je zadání příliš vágní. Předem se počítá s tím, že životní cyklus aplikace se bude v průběhu vývoje několikrát opakovat (tj. bude vyvíjen iteračně). V části analýzy se snažíme ponechat stranou vše, co je prozatím nejasné a vzdálené a soustředit se na přiměřeně krátkou část, která je definovaná dílčími úkoly, které aplikace po dokončení iterace bude splňovat (*feature driven development*). Po dokončení cyklu opět proběhne analýza zúčastněných o tom, co, jak a jestli vůbec dělat dále.

Waterfall

Vodopád patří mezi rigorózní metodiky, které nepočítají se změnou. Hodí se tudíž tam, kde je analýza velmi konkrétní. Předpokládá se, že se vývoj uskuteční v průběhu jediné iterace. Pokud v některé fázi nastanou problémy, vracíme se o jednu fázi zpět, ale ne více. Tedy pokud se při implementaci zjistí, že je něco příliš obtížné, vytvoří se alternativní návrh, ale nezasahuje se už do analýzy.

Extrémní programování

Hodí se tam, kde je velmi krátký a jednoduchý úkol, u kterého se očekávají připomínky (např. při vývoji uživatelského prostředí "posuňte prosím to tlačítko o 1 pixel dolů", "přidejte k tomu formuláři ještě jednu volbu"). V tom případě je vhodné posadit zadavatele rovnou k programátorovi a všechny fáze splynou do jedné, čímž se ušetří byrokracie.

Antivzory vývoje

Antivzory jsou často opakované postupy, které se ukázaly být jako neefektivní, zejména:

Extinct by instinct

Podcenění analýzy a návrhu, kdy se rovnou začíná kódovat na základě vágní intuitivní představy o aplikaci. Je téměř zaručeno, že dříve či později se kód stane nepřehledný, těžkopádný a plný chyb, takže ho bude nutno přepsat. A pak nejspíš ještě jednou a tak dále, dokud se neudělá pořádná analýza.

Analysis paralysis

Přecenění analýzy a návrhu, zejména u rigorózních metodik. Vzniká zejména tehdy, když do analýzy promlouvá více lidí a nikdo nemá poslední slovo. Mnoho dní se sbírají všechny podklady, řeší se příliš velké detaily, vytváří se alternativy, diagramy se komplikují a vše končí u dohadů, aniž je napsána byť jediná řádka kódu.

Konzolová hra Šibenice

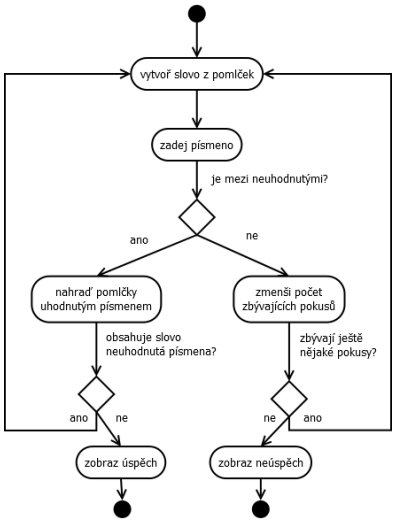
Při hodině padl návrh vytvořit hru Šibenici, na které bychom si procvičili práci s kolekcemi.

Analýza

Vzhledem k omezenému času uděláme pouze jednu iteraci vývoje, kde bude v kódu natvrdo zadané slovo, které bude hráč hádat. Bude mít natvrdo 5 pokusů. Vývoj by pak mohl probíhat v dalších iteracích, kde bychom mohli slovo načíst z nějakého slovníku a zvolit obtížnost, ale to prozatím nebudeme řešit.

Neuhodnuté znaky se zobrazí pomlčkami, uhodnuté se zobrazí na příslušné pozici. V každém kole je hráč vyzván, aby zadal písmeno. Pokud je toto na neuhodnuté pozici, zobrazí se na všech neuhodnutých pozicích, v opačném případě se zmenší počet pokusů. Hra končí uhádnutím všech písmen (úspěch) nebo vyplýtváním všech pokusů (neúspěch). Výsledek hry se před zavřením vypíše na konzoli.

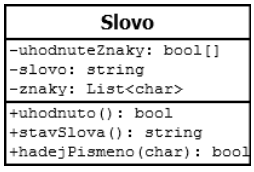
Diagram užití má smysl v případech, kdy systém využívá více lidí. Diagram aktivit by mohl vypadat následovně:



Návrh

Z diagramu aktivit bychom mohli udělat procedurální kód: jeden cyklus, který se opakuje, dokud obsahuje slovo neuhodnutá písmena nebo zbývají pokusy. Aktivita `Vytvoř slovo z pomlček` vyžaduje udržování seznamu uhodnutých písmen a funkce pro hádání písmena, vygenerování slova s pomlčkami a možná další. Tyto údaje spolu souvisí a tvoří samostatný útvar v kódu, který je účelné oddělit od zbytku kódu řešícího vstup a výstup (místo konzole bychom pak mohli v jiné verzi hry použít jiný způsob zadávání). Tyto útvary jsou třídy, funkcím v nich říkáme metody a datům položky (*fields*).

Vytvoříme tedy návrh třídy Slovo pomocí standardizovaného UML diagramu tříd:



Z těchto diagramů tedy máme dva nezávislé úkoly, z nichž každý má podúkoly:

1. třída Slovo
    - metoda `uhodnuto`: projde všechny znaky a vrátí true, pokud jsou všechny uhodnuté, jinak false
    - metoda `stavSlova`: vrátí slovo písmeno po písmenu: pomlčku, pokud ještě nebylo uhodnuto, jinak písmeno
    - metoda `hadejPismeno`: vrátí true, pokud je hádané písmeno ve slovu a dosud neuhodnuté, jinak false. Doplní písmeno mezi hádané znaky a upraví pole uhodnuteZnaky
2. cyklus pro vstup a výstup
    - vypsání slova (`stavSlova`), zobrazení počtu pokusů, vyzvání uživatele k zadání písmena, volání `hadejPismeno` a zmenšení počtu pokusů, pokud vrátí false, opakování cyklu dokud zbývají pokusy a není `uhodnuto`, vypsání výsledku

Takto sestavený seznam si projdeme a srovnáme si, jestli máme ke všem úkolům potřebné znalosti (práce s konzolí, cykly a podmínky). Pokud např. nemáme dost zkušeností s konzolí, vyhledáme zdroje, které jsou pro nás srozumitelné a odhadneme čas, který budeme potřebovat na studium, přičteme časy každého z dílčích podúkolů, přičteme plánované přestávky, přičteme nějakou rezervu, to celé vynásobíme dvěma. To je kvalifikovaný odhad času (a tedy ceny) na implementaci (přičteme čas za analýzu a návrh, který bývá tak třetinový až poloviční). Spočítáme také čas na testování:

- metoda `uhodnuto`: test, pokud není uhodnuto žádné písmeno, pokud jsou uhodnuta některá, pokud jsou uhodnuta všechna
- metoda `stavSlova`: totéž

- metoda `hadejPismeno`: test hádání nepísmenného znaku, malého a velkého písmena, písmena, které už bylo hádáno a nachází se nebo nenachází ve slově, písmena, které nebylo hádáno a nachází se nebo nenachází ve slově