Špirálový model a jeho použitie pri modelovaní softvéru*

Tomáš Andel

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

xandelt1@stuba.sk

30. september 2021

Abstrakt

Špirálový model je metóda na vývoj softvéru. Vývoj pomocou tohto modelu sa vyznačuje tým, že kladie dôraz na redukciu rizík. Článok predstaví vývoj softvéru, niektoré tradičné modely, ich problémy a ako ich špirálový model rieši. Analyzuje postup vývoja pomocou špirálového modelu, porovná jeho výhody a nevýhody. Vysvetlí, prečo je analýza a riešenie rizík dôležité a ako ich tento model rieši. Ukáže ako veľa sa dnes využíva a predstaví konkrétny príklad využitia.

Kľúčové slová : špirálový model, SDLC, vývoj softvéru, softvérové modelovanie, riešenie rizík

1 Úvod

Vývoj softvéru je zložitý proces. Tvorba nápadu, určenie požiadaviek, dizajn, implementácia, testovanie, údržba - tieto všetky a ďalšie procesy nastávajú pri vývoji softvéru, bez jasného plánu môže byť vývoj zdĺhavý a drahý. Už od možnosti vývoja prvého softvéru ľudia hľadajú modely, podľa ktorých budú pracovať, a ktoré im umožnia efektívne vyvíjať komplexné softvéry.

Existuje množstvo vývojových modelov, niektoré sú vhodnejšie na menšie projekty, pričom sú nepoužiteľné vo veľkých, kde by vývoj postupoval neefektívnym spôsobom. Iné sú zasa vhodnejšie na veľké projekty, ale pri malých projektoch by bolo ich použitie zbytočne komplikované. Tento článok ukáže vývoj skôr veľkých projektov a prečo je na to výbornou voľbou práve špirálový model.

V časti 2 sa pozrieme na životný cyklus vývoja softvéru a aké fázy obnáša. V časti 2.1 bude predstavených niekoľko tradičných vývojových modelov a budú ukázané ich nevýhody a ako ich špirálový model rieši. V ďalšej časti (3) prejdeme na samotný špirálový model. Ukážeme si ako funguje, aké sú jeho výhody/nevýhody v časti 3.1, ako prebieha riešenie objavených rizík v časti 3.2, kde a ako veľa sa využíva v časti 3.3. Na záver sa ešte vyjadrím k témam z prednášok (4) a v časti 5 budú zhrnuté získané poznatky.

*Semestrálny projekt v predmete Metódy inžinierskej práce, ak. rok 2021/22, vedenie: Ing. Vladislav Mlynarovič, PhD.

2 Vývoj softvéru

Softvéroví inžinieri sa vždy snažia vyprodukovať kvalitný produkt, ktorý zodpovedá požiadavkám klienta, neprekročí pridelený rozpočet a je vyprodukovaný načas. Bohužial, často tieto ciele nie sú dosiahnuté. Ale dobrý manažment projektu vo vhodnom prostredí, ktoré sa drží zaužívaných metód a modelov môže zaručiť konzistené dosahovanie týchto cielov. [6]

Životný cyklus vývoja softvéru je proces tvorby softvéru, ktorého cieľom je finálny produkt čo najvyššej kvality a čo najnižšej ceny [4]. Väčšinou obnáša tieto fázy:[9]

1. **Iniciácia/Počiatočné plánovanie** - Diskusia o realizovateľnosti projektu, počiatočná estimácia nákladov a časovej náročnosti.
2. **Analýza požiadaviek a špecifikácia** - Identifikácia problémov, ktoré má vyvíjaný softvér riešiť a ich špecifikácia. Riešenie jeho operačných schopností, výkonnostných charakteristík a infraštruktúry potrebnej na jeho údržbu.
3. **Funkčná špecifikácia alebo prototypovanie** - Identifikácia objektov, ich vlastností a vzťahov. Identifikácia obmedzení, ktoré obmedzujú správanie systému ať d.
4. **Rozdelenie systémov (Build vs. Buy vs. Reuse)** - Rozdelenie systému na menšie časti a zistenie, ktoré časti je výhodné vyrobiť, ktoré stačí odkúpiť a nakonfigurovať k požiadavkám systému, a ktoré je možné opäť použiť z predošlých projektov.
5. **Architektúra** - Definuje prepojenia a vytvára rozhrania medzi jednotlivými podsystémami, komponentami a modulmi aby bol možný ich jednotlivý detailný dizajn.
6. **Detailný dizajn komponentov** - Definuje funkciu jednotlivých komponentov, ich interné správanie, ako transformujú vstupy na požadované výstupy.
7. **Implementácia komponentov** - Kodifikuje špecifikácie navrhnuté počas architektúry a detailného dizajnu komponentov do funkčného zdrojového kódu.
8. **Testovanie a kontrola integrity** - Kontrola systému a podsystémov na základe ich požiadaviek. Kontroluje celkovú integritu systému.
9. **Tvorba dokumentácie** - Vytváranie systematických dokumentov a užívateľských príručiek.
10. **Spustenie softvéru pre klienta/užívateľa** - Poskytnutie softvéru užívatelovi a prípadne inštrukcie na inštaláciu a konfiguráciu.
11. **Školenie a používanie softvéru** - Školenie užívateľov systému ako správne a efektívne softvér využívať.
12. **Údržba softvéru** - Udržiavanie operatívnosti softvéru, opravovaním objavených chýb, poskytovaním funkčných alebo výkonnostných vylepšení a údržbou podpornej infraštruktúry.

2.1 Metódy vývoja softvéru

Metódy alebo modely na vývoj softvéru popisujú spôsob navigácie medzi vyššie uvedenými procesmi vývoja.[10]

Existuje veľké množstvo modelov a každý z nich je vhodný v inej situácií. Pozrime sa na niekoľko tradičných modelov [6]:

- **Waterfall Model** - Jeho grafická reprezentácia vyzerá ako kaskáda vodopádov. Je to jeden z najstarších modelov a je základom mnohých iných modelov. Pri používaní tohto modelu je každý proces najprv dokončený a až potom sa prechádza na ďalší. Jeho nevýhodou je nízka úroveň flexibility, všetky požiadavky je potrebné vedieť už na začiatku. Nie je vhodný na veľké projekty. Analýza rizík je často zložitá.

- **Sashimi Model** - Waterfall model, pri ktorom je dovolená paralelná práca na viacerých fázach naraz, je časovo výhodnejší.

- **V-Shaped Model** - Je považovaný za rozšírenie Waterfall modelu. Namiesto lineárneho pohybu nadol sa vytvára tvar V s vrcholom, v ktorom je implementačný proces. Model vytvára vzťahy medzi vývojovými procesmi pred implementáciou a ich príslušnými fázami testovania. Výhodou tohto modelu jednoduché použitie rovnako ako pri Waterfall modeli a narozdiel od Waterfall modelu je tu vyššia šanca úspechu kvôli plánovaniu testovania už na začiatku vývoja. Nevýhodou je nízka úroveň flexibility a nemožnosť tvorby skorých prototypov pretože celková produkcia kódu je až vo fáze implementácie.

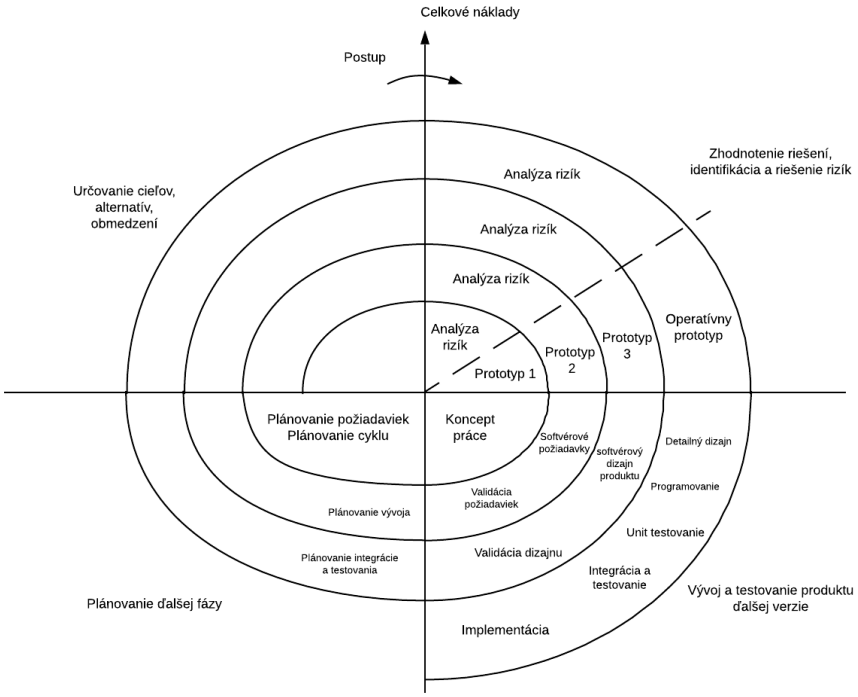
Ako je možné vidieť, nevýhodami týchto modelov sú hlavne nízka flexibilita a zlá analýza rizík. Pri vývoji veľkých projektov často vznikajú rôzne riziká, ktoré sa pri tých menších objavujú menej. Preto veľké firmy hľadajú iné modely, pomocou ktorých tieto riziká vedia odhaliť a uskutočniť určité kroky na ich spracovanie.

V ďalšej časti sa pozrieme na špirálový model a ako dokáže tieto problémy riešiť.

3 Špirálový model

Špirálový model, pôvodne navrhnutý Barrym W. Boehmom, je evolučný softvérový procesný model, ktorý spája iteračnú vlastnosť prototypovania s riadenými a systematickými aspektmi lineárneho sekvenčného modelu. [5]

Jeho diagram vyzerá ako špirála s mnohými sľučkami (viď. obrázok 1). Každá sľučka špirály sa nazýva fáza vývojového procesu softvéru. Presný počet fáz závisí od zložitosti projektu. Projektový manažér analyzuje riziká a iné faktory, podľa ktorých dynamicky určuje počet týchto fáz.



Obr. 1: Diagram špirálového modelu - upravené podľa [1]

Čím je väčší počet fáz, tým sú väčšie náklady projektu - polomer špirály predstavuje celkové náklady projektu. Uhol medzi pozorovaným miestom a začiatkom fázy predstavuje dosiahnutý postup v danej fáze. [7]

Každá fáza je rozdelená do štyroch kvadrantov, ako je znázornené na obrázku. Kvadranty sú [7]:

1. **Určovanie cieľov, alternatív, obmedzení** - V tomto kvadrante prebieha zbieranie požiadaviek od klientov, analýza a identifikácia cieľov a potom diskusia ohľadom alternatívnych riešení, ktoré by mohli byť použité v ďalších častiach súčasnej fázy.

2. **Zhodnotenie riešení, identifikácia a riešenie rizík** - V druhom kvadrante sú všetky možné riešenia zhodnotené a vyberie sa najlepšie riešenie.

Ďalej prebieha analýza rizík spojených s vybraným riešením, tie sú následne riešené pomocou najvhodnejšej stratégie. Na konci tohto kvadrantu je zhotovený prototyp, ktorý je v súlade s vybranou najlepšou stratégiou.

3. **Vývoj a testovanie produktu ďalšej verzie** - V treťom kvadrante prebieha všetka implementácia a testovanie. Určené špecifikácie sú vytvárané a testované. Na konci tohto kvadrantu je pripravená ďalšia verzia produktu.

4. **Plánovanie ďalšej fázy** - Klient zhodnotí vytvorený produkt a prebieha plánovanie ďalšej fázy.

3.1 Výhody a nevýhody špirálového modelu

Ako je zjavné z popisu vyššie, tento model je vhodný najmä pre rozsiahle, komplexné projekty. Progresívna povaha tohto modelu umožňuje vývojárom rozdeliť veľký projekt na menšie časti a venovať sa plne jednej funkcii, čo zaručuje lepšie prepracovanie, menší počet chýb atď. [2]

Medzi ďalšie **výhody** špirálového modelu patrí [11] :

- Prvé verzie softvéru sú dostupné už na začiatku vývoja.
- Riešenie rizík - je jednou z najdôležitejších predností špirálového modelu. Vďaka analýze a riešeniu rizík v každej fáze tento model minimalizuje riziko neúspechu rozsiahlych projektov.
- Flexibilný vzhľadom na nové požiadavky - pri vývoji týmto modelom, môžeme kedykoľvek v neskorších fázach vývoja zmeniť požiadavky na vyvíjaný softvér a ľahko a správne ich zapracovať do softvéru. Rovnako môže byť kedykoľvek pridaná aj ďalšia funkcionalita.
- Vývoj týmto modelom je otvorený spolupráci so zákazníkmi/užívateľmi. Zákazníkov môžeme, najmä vďaka tvorbe skorých a pravidelných prototypov, zapájať už od začiatku vývoja. Toto redukuje množstvo nedorozumení a tým šetrí čas a náklady na prerábanie nevhodnej funkcionality.
- Kontrola postupu práce a tvorba dokumentácie je na vysokej úrovni.
- Je vhodný pre vysoko rizikové projekty. Pomocou tohto modelu je možné vytvoriť veľmi prispôsobený produkt.

Medzi **nevýhody** špirálového modelu patrí [11] :

- Nie je vhodný na malé projekty. Vývoj by bol moc nákladný a nedokázali by využiť jeho plný potenciál.
- Vysoká komplexita - je komplexnejší ako väčšina ostatných vývojových modelov.
- Dôraz na analýzu a riešenie rizík vytvára potrebu neštandardnej špecifickej expertízy.

- Neurčité časové estimácie. Kedže počet fáz býva na začiatku vývoja neznámy, nie je jednoduché určiť časovú dĺžku vývoja softvéru, preto sa využíva skôr na dlhodobý vývoj.

- Špirála môže pokračovať do nekonečna.

- Nie je vhodný na nízko rizikové projekty. Pre takéto projekty nie je dôvod tento model využívať.

- Definícia jasných, overiteľných miľníkov môže byť zložitá. Častá zmena funkcionality si vyžaduje nadmernú dokumentáciu.

3.2 Riešenie rizík pomocou špirálového modelu

Riziko je akákoľvek nepriaznivá situácia, ktorá môže ovplyvniť úspešné dokončenie softvérového projektu. Najdôležitejšou vlastnosťou špirálového modelu je analýza a riešenie týchto rizík. Takéto riešenia rizík sa dajú ľahšie vykonať vývojom prototypu. Špirálový model má na tento proces vyhradený jeden kvadrant v každej fáze vývoja.

Aj "prototypovací model"podporuje analýzu a riešenie rizík, ale všetky riziká musia byť identifikované už pred začatím vývoja. Avšak často sa riziká naskytávajú aj počas vývoja, takže v tomto je špirálový model flexibilnejší a celkovo vhodnejší. [5]

3.3 Využitie špirálového modelu

Dáta o popularite špirálového modelu sú ťažko dohľadateľné, takže v tejto téme som sa odrazil iba od jednej štúdie spoločnosti GoodFirms, preto môžu byť výsledky nepresné.

V tejto štúdii GoodFirms skúmali 150 firiem, ktoré vyvíjajú softvér. Jedným z faktorov, ktorý skúmali je aké vývojové modeli tieto firmy využívali. Výsledky tohto prieskumu sú uvedené v tabuľke 1

Tabuľka 1: Využitie vývojových modelov firmami [8]

Model	Počet firiem	Podiel
Agile	92	61%
Scrum	35	23%
Waterfall	14	9%
Špirálový a iné	9	6%
Spolu	150	100%

Ako je možné vidieť v tabuľke, táto štúdia naznačuje minimálne využitie špirálového modelu. Jedným z dôvodov môže byť nevhodná vzorka skúmaných firiem - v štúdii sa vyskytovali najmä menšie firmy, ktoré sa len zriedka púšťajú do vývoja rozsiahlych projektov. Preto sa teraz pozrieme na konkrétny príklad využitia špirálového modelu.

GanttPRO - Aplikácia, ktorá poskytuje nástroje na projektový manažment, podobná aplikáciám ako Notion alebo MS Project.

Vývojári využívali pri vývoji špirálové a scrum princípy. Napríklad kratšie iterácie, čo im dovolilo častejšie produkovať prototypy a tým získali častejšiu spätnú väzbu. Pre každú iteráciu (fázu) vytvorili detailný plán práce.

Hlavnými problémami, ktorým vývojári čelili, boli:

- Rôzne nové požiadavky, ktoré sa časom naskytovali. Z iterácie do iterácie ich bolo potrebné objasniť a zakomponovať.
- Potreba analýzy a riešenia rizík, ktoré sa vývojom naskytovali.
- Možné zvýšené náklady spôsobené dlhým plánovaním.

Zvolili teda kratšiu frekvenciu vývoja prototypov - dvojtýždňové iterácie. V dôsledku toho sa im podarilo zredukovať množstvo rizík a maximalizovať zisky vďaka rýchlemu prispôsobeniu sa potrebám používateľov a trhu. [3]

4 Reakcia na témy z prednášok

V tejto časti zareagujem na niekoľko tém, ktoré boli obsahom prednášok na predmete MIP.

Inžinierska gramotnosť - V tejto prednáške sme sa dozvedeli kto je vlastne inžinier. Inžinier pracuje najmä s technickými informáciami - mal by vedieť pochopiť, interpretovať, aplikovať a ďalej odovzdať informáciu. Spomedzi viacerých zdrojov na rovnakú tému, by mal vedieť zhodnotiť čím a prečo sa odlišujú.

Ďalej sme sa dozvedeli o inžinierskej gramotnosti. Obnáša presné písomné alebo ústne vyjadrovanie, pričom používa prostriedky a nástroje, ktoré mu takéto vyjadrovanie umožnia. Jedným z takýchto nástrojov je LaTeX, o ktorom sme sa dozvedeli v zápätí. Tento nástroj sa zdá menej praktický ako WYSIWYG editory, ale po jeho používaní pri písaní článku som názoru, že nie je vôbec na zahodenie.

Načo budem inžinierom? - Túto prednášku prezentoval dekan fakulty, profesor Ivan Kotuliak. Rozprával napríklad prečo je vysokoškolské vzdelanie dôležité alebo o dôležitosti dokumentácie práce, s čím súhlasím. Pokračoval v predstavovaní rôznych možností zamestnania v IT a rôznych IT aplikáciách. Táto prednáška bola podľa môjho názoru zaujímavá a inšpiratívna.

Udržateľnosť a etika - Táto prednáška sa zaoberala udržateľnosťou a pokladala otázku, či je možné udržať návštevnosť študentov na prednáškach. Myslím si, že do určitej miery to je možné. Najskôr sú tu radikálnejšie riešenia ako je povinná účasť alebo povinné skúšanie/testy na prednáške, tieto sú efektívne, ale vrámci udržania návštevnosti študentov z vlastnej vôle sú nevhodné. Iné riešenia môžu byť napríklad prednes dôležitých informácií, ktoré sa mimo prednášky nenachádzajú, interakcia so študentami alebo rôzne zaujímavé obmeny obsahu prednášky.

Ďalej sa prednáška zaoberala etikou - myslím si, že etické správanie je dôležité. Zvyšuje životnú úroveň a je dôležitou súčasťou modernej spoločnosti. Rovnako dôležité je aj v IT.

5 Záver

Článok stručne predstavil vývoj softvéru a životný cyklus vývoja softvéru. Ďalej, na porovnanie, priblížil pár základných metód (modelov) vývoja softvéru. V ďalších častiach bol spodrobnený špirálový model, práca s ním a riešenie rizík. Prieskum využitia špirálového modelu ukázal nízky záujem s čím nesúhlasím. Iné zdroje naznačujú rozsiahlejšie využitie tohto modelu, ale bez potrebných dát, ktoré by tieto tvrdenia potvrdzovali.

V závere teda článok poskytol prehľad o špirálovom modeli vývoja softvéru, ukázal jeho prednosti a kedy ho je vhodné využívať.

Literatúra

[1] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[2] K. Damore. Definition - spiral model. `https://searchsoftwarequality.techtarget.com/definition/spiral-model`, 2019.

[3] D. Gurendo. Software development life cycle (sdlc). spiral model. `https://xbsoftware.com/blog/software-development-life-cycle-spiral-model/`, 2015.

[4] Harness. Understanding the phases of the software development life cycle. `https://harness.io/blog/software-development-life-cycle/`, 2021.

[5] S. Jaiswal. Spiral model. `https://www.javatpoint.com/software-engineering-spiral-model`, 2019.

[6] G. Kumar and P. K. Bhatia. Comparative analysis of software engineering models from traditional to modern methodologies. In *2014 Fourth International Conference on Advanced Computing Communication Technologies*, pages 189–196, 2014.

[7] S. K. Pal. Software engineering — spiral model. `https://www.geeksforgeeks.org/software-engineering-spiral-model/`, 2021.

[8] M. Raymond. Remarkably useful stats and trends on software development. `https://www.goodfirms.co/resources/software-development-research`, 2019.

[9] W. Scacchi. Process models in software engineering. In *J.J. Marciniak (ed.), Encyclopedia of Software Engineering, 2nd Edition, John Wiley and Sons, Inc, New York, December 2001.*, Institute for Software Research, University of California, Irvine, Oct. 2001.

[10] B. Shiklo. Software development models: Sliced, diced and organized in charts. `https://www.scnsoft.com/blog/software-development-models`, 2019.

[11] R. K. Upadhyay. Advantages and disadvantages of using spiral model. `https://www.geeksforgeeks.org/advantages-and-disadvantages-of-using-spiral-model/`, 2020.