Nástroje CASE a ich využitie v reverznom inžinierstve*

Šimon Ukuš
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xukus@stuba.sk

9. december 2021

Abstrakt

Článok sa zaoberá problematikou softvérového inžinierstva, kontkrétne ako je možné proces vývoja softvéru automatizovať pomocou nástrojov CASE - Computer-Aided Software Engineering (Počítačom podporované softvérové inžinierstvo). V práci sa uvádza klasifikácia týchto nástrojov. Článok ďalej skúma softvérové inžinierstvo a použitie CASE nástrojov z inej perspektívy. Na rozdiel od vnímania vývoja softvéru klasicky, teda smerom vpred (Forward Engineering) sa sústredí na tzv. spätné inžinierstvo (Reverse Engineering). Tu je skúmaná kompletnosť a presnosť spätne navrhnutých UML diagramov generovaných nástrojmi CASE. Predmetom porovnania bolo celkom 8 nástrojov (z toho 6 open source a 2 komerčné). Tieto nástroje boli hodnotené na základe toho, aké typy vstupov podporujú, aké typy diagramov dokážu rekonštruovať a v akej kvalite.

Kľúčové slová: nástroje CASE, softvérové inžinierstvo, reverzné inžinierstvo, UML model, modelovanie v softvérovom inžinierstve

1 Úvod

Pojem softvérové inžinierstvo môže byť chápaný ako uplatňovanie metód, postupov a nástrojov na riadenie a vývoj počítačových systémov [3]. Ide o komplexný postup, na ktorom sa zúčastňuje mnoho odborníkov z rôznych oblastí, ako napríklad projektový manažér, team líder, softvérový developer, tester, UI dizajnér a mnoho ďalších. Časť ich práce je možno automatizovať či uľahčiť využitím nástrojov CASE. Tieto nástroje sú špeciálne vyvinuté pre podporu vývoja softvéru, automatizujú proces vývoja. Cieľom ich implementácie je ušetrenie času a nákladov pri vývoji softvéru a zvýšenie jeho kvality [1].

Využitie nástrojov CASE a ich klasifikácia sú uvedené v časti 2., kde je popísané delenie podľa toho, ktorú časť tzv. životného cyklu vývoja softvéru

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

(Software Development Life Cycle, ďalej len SDLC) pomájhajú automatizovať. V časti 3 sa čitateľ zoznámi s pojmom *reverzné inžinierstvo*. Nachádza sa tu prehľad, v ktorom je popísaný rozdiel medzi tzv. *forward* (dopredným) a *reverse* (spätným) inžinierstvom. Samotné nástroje, ich predstavenie a následné porovnanie rozoberajú časti 4 a 5. Záverečné hodnotenie týchto nástrojov prináša časť 7.

2 Klasifikácia nástrojov CASE

Nástroje CASE sú odpoveďou na stále narastujúce nároky a zvyšujúcu sa komplexnosť počítačových systémov. Podporujú vývoj softvéru a dajú sa aplikovať v niektorých, niekedy vo všetkých fázach SDLC, ktorého podpora je čoraz žiadúcejšia. Náklady na vývoj softvéru každým rokom narastajú, a preto čo i len skromné vylepšenia pri vývoji a automatizácii môžu znamenať veľké úspory. Sú cielené na riešenie ťažkostí pri vývoji vysokokvalitného a komplexného softvéru načas a v súlade s rozpočtom [7].

Ako sa vyššie spomína, nástroje CASE slúžia na podporu rôznych fáz SDLC, prípadne dokážu automatizovať všetky z nich. Zvyknú sa preto klasifikovať podľa toho, ktoré štádium životného cyklu podporujú. Takéto rozdelenie vyzerá nasledovne:

- Upper CASE nástroje
- Lower CASE nástroje
- Integrated CASE nástroje

Upper CASE, niekedy označované aj ako *front end CASE* slúžia na podporu skorých fáz životného cyklu softvéru, napríklad pri analýze a dizajne.
Lower CASE, tiež nazývané aj *back end CASE* zase nachádzajú využitie v neskorších fázach životného cyklu softvéru, najmä pri testovaní a vytváraní kódu.
Integrated CASE, sú schopné pokryť obe časti SDLC [4].

Iný pohľad na kategorizáciu poskytuje autor [5], ktroý nespája fázy SDLC do tzv. skorých a neskorších, ale konkrétne fázy jednotlivo vymenúva a podľa toho delí nástroje CASE ako nástroje na:

- riadenie projektov
- analýzu a návrh
- podporu Objektovo-Orientovaného sw inžinierstva
- testovanie
- nástroje formálnych metód
- Klient/Server nástroje
- webové inžinierstvo
- opätovné inžinierstvo (Reengineering)

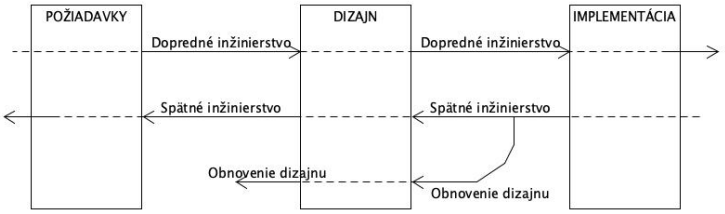
Pri tomto delení sa stretávame s pojmom opätovné inžinierstvo (Reengineering), a keďže v ďalších kapitolách je rozpracovaná téma reverzného inžinierstva a použitie nástrojov CASE v reverznom (spätnom) inžinierstve, je potrebné

uviesť, že je rozdiel medzi spätným inžinierstvom (Reverse Engineering) a opätovným inžinierstvom (Reengineering). Síce oba pojmy odkazujú na ďalšie skúmanie alebo vývoj už hotových produktov, metódy a požadované výsledky sa výrazne líšia. Ako [6] ďalej vysvetľuje, reverzné inžinierstvo sa snaží odhaliť, ako daný systém funguje. Úlohou opätovného inžinierstva je zlepšenie súčasného návrhu skúmaním jeho konkrétnych aspektov.

3 Z dopredného inžinierstva k spätnému

Spätné inžinerstvo, ako už bolo v sekcii 2 spomenuté, sa snaží pochopiť funkciu systému. Obr. 1 ilustruje spôsob vývoja informačných systémov. Pre jednoduchosť boli použité tri fázy životného cyklu, kde prvá - *"požiadavky"* predstavuje špecifikáciu problému, druhá - *"dizajn"* zase špecifikáciu riešenia a tretia - *"implementácia"* predstavuje programovanie a testovanie požadovaného systému.

Pojem dopredné *(forward)* inžinierstvo reprezentuje tradičný proces vývoja softvéru, kedy sa postupuje od prvotných abstraktných modelov, cez dizajn až ku konečnej implementácii systému. Zavedenie názvu pre niečo tak priamočiare sa môže zdať zbytočné, možno aj mätúce . V našom prípade, a tiež v mnoho iných, je to však nevyhnutné pre jeho odlíšenie od spätného inžinierstva. Ako Obr 1 ukazuje, dopredné inžinierstvo prechádza jednotlivými fázami SDLC z ľava do prava.



Obr. 1: Vybrané fázy SDLC a znázornené procesy súvisiace s dopredným a spätným inžinierstvom (prevzaté a preložené z [2])

Na druhej strane, spätné *(reverse)* inžinierstvo postupuje opačným smerom oproti doprednému inžinierstvu, čo ilustruje Obr. 1. Reverzné inžinierstvo je proces, počas ktorého je systém (softvér) analyzovaný, výsledkom čoho je pochopenie fungovania daného systému, identifikácia komponentov, prípadne prinavrátenie stratených informácií [2]. Vďaka reverznému inžinierstvu je možné rekonštruovať UML diagramy, na čo je možné použiť nástroje CASE. Práve problematika generovania UML diagramov zo zdrojového kódu je ďalej popísaná v sekciách 4 a 5, pre ktoré bola podkladom štúdia realizovaná Osmanom et al. [8].

4 Predstavenie nástrojov

Táto sekcia zobrazuje nástroje, ktoré boli predmetom skúmania, realizovaného autorom [8]. Všetkých 8 posudzovaných nástrojov je uvedených v Tabuľke 1. Pri evaluácii komerčných nástrojov bola použitá verzia pre účely testovania, prípadne verzia na vzdelávacie účely.

Tabuľka 1: Zoznam posudzovaných nástrojov (prevzaté a preložené z [8])

č	**nástroj CASE**	**informácie**	**Predajca**	**Typ Licencie**
1	Visual Paradigm 8.1	Visual Paradigm	Visual Paradigm	Komerčné
2	MagicDraw 17.0	MagicDraw	No Magic	Komerčné
3	Altova Umodel 2011	Altova Umodel	Altova	Komerčné
4	Enterprise Architect 8.0	Enterprise Architect	Sprax System	Komerčné
5	Rational Software Architect 8.0.1	Rational Software	IBM	Komerčné
6	MyEclipse 8.6	MyEclipse	Genuitec	Komerčné
7	StarUML 5	StarUML	StarUML	Open Source
8	ArgoUML	ArgoUML	Tigris.org	Open Source

Pri porovnávaní nástrojov boli zisťované nasledovné parametre:

- ktoré programovacie jazyky sú podporované jednotlivými nástrojmi
- ktoré nástroje sú schopné rekonštrukcie ktorých diagramov (diagramy balíčkov - 'package'; diagramy tried - 'class'; sekvenčné diagramy - 'sequence')
- či sú rekonštruované diagramy správne a úplne

5 Výsledky porovnania nástrojov

V tejto sekcii sú uvedené výsledky porovnania nástrojov zo sekcie 4. Výsledky sú zhrnuté v Tabuľke 2.

Nástrojmi podporované programovacie jazyky sú uvedené v Tabuľke 2, z ktorej je možné vyčítať, že Enterprise Architect podporuje všetky jazyky uvedené v hlavičke tabuľky. Celkovo väčšina nástrojov dokáže rekonštruovať UML modely zo súborov zdrojových kódov typu .java, .cpp, .cs. Dokonca všetky porovnávané nástroje dokážu spätne generovať modely zo zdrojových kódov napísaných v jazyku Java.

V pravej časti Tabuľky 2 sú uvedené schopnosti nástrojov z hľadiska podporovaných typov diagramov. V tejto časti tabuľky sa vyskytujú tri rôzne znaky, ktoré predstavujú tri úrovne hodnotiacej škály. Vysvetlivky k danej škále sú nasledovné [8]:

- "+" - nástroj dokáže rekonštruovať daný typ diagramu aj keď s malou/malými chybami, ako napríklad nesprávne uvedenie/neuvedenie vzťahov pri diagramoch tried (agregácia a kopozícia často uvedená ako asociácia)
- "0" - nástroj síce dokáže rekonštruovať daný typ diagramu, ale neposkytuje všetky informácia o diagrame
- "-" - nástroj nie je schopný rekonštrukcie daného typu diagramu

Väčšina nástrojov je schopná spätnej rekonštrukcie diagramov balíčkov *(package)*. Všetky porovnávané nástroje dokážu generovať diagramy tried *(class)*, ArgoUML je však v tomto najmenej spoľahlivý, keďže nedokáže rekonštruovať vzťahy medzi triedami (iné ako dedičnosť) [8]. Čo sa týka sekvenčných diagramov, ani pri jednom z porovnávaných nástrojov sa nedá hovoriť o priamej

Tabuľka 2: Zoznam podporovaných programovacích jazykov a diagramov jednotlivými nástrojmi (prevzatá a upravená od [8])

č.	nástroj CASE	programovacie jazyky								UML Diagramy		
		PHP 5	C++	Java	Delphi	Python	V.B	C#	celkovo	Diagramy balíčkov (Package)	Diagramy tried (Class)	Sekvenčné diagramy (Sequence)
1	Visual Paradigm 8.1	Y	Y	Y	N	Y	N	N	4	+	+	0
2	MagicDraw 17.0	N	Y	Y	N	N	N	Y	3	0	+	-
3	Altova Umodel 2011	N	N	Y	N	N	Y	Y	3	+	+	0
4	Enterprise Architect 8.0	Y	Y	Y	Y	Y	Y	Y	7	0	+	0
5	Rational Software Architect 8.0.1	N	Y	Y	N	N	Y	Y	4	0	+	0
6	MyEclipse 8.6	N	N	Y	N	N	N	N	1	0	+	-
7	StarUML 5	N	Y	Y	N	N	N	Y	3	0	+	-
8	ArgoUML	N	Y	Y	N	N	N	Y	3	0	0	-

podpore, no niektoré nástroje dokážu s malou pomocou používateľa generovať takéto diagramy.

6 Reakcia na témy z prednášok

Udržateľnosť a etika. O udržateľnosti, z hľadiska softvéru, by sa dalo povedať ako o schopnosti pretrvať. To znamená, že softvér bude v budúcnosti dostupný (napr. nová platforma). Na to je samozrejme potrebné ho udržiavať. Práve nástroje, schopné reverzného inžinierstva dokážu veľmi pomôcť pri údržbe softvéru, kde pomáhajú s pochopením samotného systému. Vďaka nim je možné extrahovať štruktúru programu.

Táto metóda môže byť využitá aj v kontexte etiky, pri detekcii plagiátorstva, kedy štruktúra programu môže slúžiť ako ukazovateľ príliš veľkej podobnosti (alebo zhodnosti). Pri skopírovaní cudzej práce a prezentovaní ju za svoju by bolo možné získať podobnú (prípadne rovnakú) štruktúru. Na identifikáciu plagiátorstva určite existuje veľa nástrojov, ktoré ponúkajú presnejšie výsledky, ale je zrejmé, že výsledky reverzného inžinierstva môžu slúžiť ako podnet k ďalšiemu prešetreniu.

Technológia a ľudia. Technológia ako pojem navodzuje pocit hmotného objektu. Významovo ale môže predstavovať aj softvérové riešenie. Je dôležité vedieť technológie využívať vo svoj prospech, a teda tak aby slúžili ľuďom a nie naopak. Technológie umožňujú ľuďom pracovať efektívnejšie prostredníctvom automatizácie, ktorú im ponúkajú. Rovnako tak aj nástroje CASE ponúkajú automatizáciu v rôznych fázach SDLC. Sú odpoveďou na stále narastajúce nároky a zvyšujúcu sa komplexnosť počítačových systémov. Náklady na vývoj softvéru každým rokom narastajú, a preto, ako aj autor [7] uvádza, už len skromné vylepšenia pri vývoji a automatizácii môžu znamenať veľké úspory. Vďaka automatizácii, ktorú ponúkaju, dokážu pomôcť pri vývoji vysokokvalitného softvéru načas a v súlade s rozpočtom.

Historické súvislosti. V dnešnej dobe veľa vecí berieme ako samozrejmosť a neuvedomujeme si, koľko času a úsilia ľudia venovali týmto pre nás maličkostiam. Iné to nie je ani v informatike. Zoberme si napríklad pre nás úplne bežnú vec, že na jednom zariadení dokáže bežať niekoľko programov súčasne. V začiatkoch informatiky to nebolo také samozrejmé. Softvér na počítačoch bol minimálny a často si ho používateľ musel vytvoriť sám. Takto špecificky naprogramovaný počítač dokázal pracovať iba s jedným programom. V 50. rokoch minulého storočia boli počítače nanjvýš tak vo veľkých spoločnostiach, výskumných centrách a na univerzitách. Na jednej takej univerzite, konkrétne na MIT, videl Fernando Corbató potenciál drahého prístroja, ktorý zatiaľčo je zaneprázdnený nenáročnými výpočtami zostáva nevyužitý, a preto sa rozhodol vytvoriť akýsi operačný systém, ktorý by dovoľoval pracovať súbežne viacerým používateľom na jednom počítači. V roku 1961 predstavil takýto operačný systém pod názvom Compatible Time-Sharing System (CTSS). Z neho sa vyvinul operačný systém Multics, ktorý bol neskôr inšpiráciou pre vytvorenie operačného systému Unix.

7 Zhrnutie

Tento článok skúmal automatizáciu SDLC pomocu nástrojov CASE. Boli uvedené dve rôzne klasifikácie, na základe ktorých možno nástroje CASE rozdeliť. Obe síce vychádzajú z jednotlivých fáz životného cyklu, no jedno delí nástroje podľa konkrétnych fáz a druhé ich zase stručnejšie delí do ranných a neskorých fáz. V práci je ďalej opísané tzv. *dopredné* inžinierstvo a potreba zavedenia tohto prídavného mena. Neskôr je v článku popísané aj reverzné inžinierstvo a stručne uvedená potreba jeho využitia. V posledných častiach sa článok zameriava na ohodnotenie 8 konkrétnych nástrojov CASE (z toho 6 komerčných a 2 open source) v reverznom inžinierstve. Pri porovnávaní bolo sledované, aké programovacie jazyky sú nástrojom podporované, a tiež ktoré typy diagramov sú podporované. Pri typoch diagramoch boli nástroje hodnotené aj na základe toho, ako úspešne dokázali jednotlivé typy diagramov rekonštruovať. Nasledujúce závery sú prevzaté od [8].

Z výsledkov porovnania vyplýva, že všetky nástroje dokážu rekonštruovať diagramy tried a diagramy balíčkov zo zdrojového kódu. Niektoré nástroje produkujú aj sekvenčné diagramy, no pri nich sa nejedná o úplne automatické generovanie (používateľ musí vybrať metódu v triede).

Celkovo neboli pozorované veľke rozdiely medzi schopnosťami jednotlivých nástrojov. Všetky majú relatívne rovnaké silné a slabé stránky. Jedna slabina bola pozorovaá naprieč všetkými porovnávanými nástrojmi. Ide o nesprávne zobrazenie vzťahov medzi triedami, kde agregáciu a kompozíciu často prezentujú ako asociáciu (prípadne neprezentujú vôbec). Takýto informačný chaos môže viesť k nesprávnej interpretácii. Používatelia, ktorí chcú využívať tieto nástroje v reverznom inžinierstve by si mali byť vedomí tohoto nedostatku.

Do budúcna sa odporúča zopakovať toto porovnanie, prípadne porovnať nástroje prezentované v tejto práci s inými nástrojmi, nakoľko výsledky štúdie, ktoré dosiahli Osman a Chaudron sa môžu zdať zastarané. Vďaka nim ale ich nasledovníci vedia, aké parametre treba detailne odsledovať.

V závere článku možno nájsť reakcie na témy z predmetu Metódy inžinierskej práce. Témy *Udržateľnosť a etika* a *Technológia a ľudia* nadväzujú na článok. Pri téme *Historické súvislosti* je vyjadrenie ku konkrétnej osobnosti z prednášky.

Literatúra

[1] O. I. A. Ashour and T. Pusatli. Adoption of case tools & UML. ACM, Nov. 2020.

[2] E. Buss and J. Henshaw. A software reverse engineering experience. ACM Press, 2010.

[3] A. F. Case. Computer-aided software engineering (CASE). 17(1):35–43, Sept. 1985.

[4] N. L. Chervany and D. Lending. CASE tools. 19(2):13–26, Apr. 1998.

[5] G. Dias. Evolvement of computer aided software engineering (case) tools: A user experience. *International journal of computer science and software engineering*, 6(3):55, 2017.

[6] L. J. Fagleman. What is the difference between reverse engineering and re-engineering?, Mar 2019.

[7] L. Fowler, J. Armarego, and M. Allen. CASE tools: Constructivism and its application to learning and usability of software engineering tools. 11(3):261–272, Sept. 2001.

[8] M. H. Osman and M. Chaudron. Correctness and completeness of case tools in reverse engineering source code into uml model. *The GSTF Journal on Computing (JoC)*, 1, 01 2012.