

Prototypování

Softwarové prototypování ([anglicky](#) *Software prototyping*) je vytváření [prototypů](#) softwarových aplikací, tj. neúplných verzí [softwarového programu](#). Je to činnost používaná v [procesu vývoje softwaru](#) a je srovnatelná s prototypováním v jiných oborech, např. ve [výrobě](#) nebo [strojírenství](#).

Prototyp obvykle simuluje pouze některé aspekty výsledného produktu a může se od něj výrazně lišit.

Prototypování má několik výhod: softwarový návrhář a implementátor mohou v časných stádiích projektu získat cennou zpětnou vazbu od uživatelů. Klient a kontraktor mohou zjistit, zda software vyhovuje [softwarovým standardům](#), které má program splňovat. Umožňuje softwarovým inženýrům určitý odhad přesnosti počátečních odhadů a reálnosti dohodnutých termínů a [mílníků](#). Míra nasazení a techniky používané v prototypování se vyvíjejí a diskutují od jeho návrhu na začátku 70. let 20. století.^[6]



Obsah

- [1 Úvod](#)
- [2 Náčrt procesu prototypování](#)
- [3 Dimenze prototypů](#)
 - [3.1 Horizontální prototyp](#)
 - [3.2 Vertikální prototyp](#)
- [4 Typy prototypování](#)
 - [4.1 Zahazovací prototypování](#)
 - [4.2 Evoluční prototypování](#)
 - [4.3 Inkrementální prototypování](#)
 - [4.4 Extrémní prototypování](#)
- [5 Výhody prototypování](#)
- [6 Nevýhody prototypování](#)
- [7 Nejlepší projekty používající prototypování](#)
- [8 Odkazy](#)
 - [8.1 Poznámky](#)
 - [8.2 Reference](#)
 - [8.3 Související články](#)

Úvod

Původním účelem prototypů je umožnit budoucím uživatelům softwaru vyhodnotit návrhy designu výsledného produktu skutečným používáním namísto interpretace a vyhodnocování jeho popisů. Prototypování je také použitelné pro koncové uživatele k popisu a doplnění požadavků, které byly původně opomenuty, a které mohou být klíčovým faktorem v komerčním vztahu mezi vývojáři a jejich klienty^[1]. S tímto cílem využívá intenzivně prototypování tak zvaný [Interakční design](#).

Tento přístup je v protikladu s monolitickým vývojovým cyklem vytváření kompletních programů v 60. a 70. letech 20. století, kdy byl nejdříve vyvinut kompletní program a pak se řešily nalezené odchylky mezi návrhem a implementací, což obvykle vede k vyšším nákladům a špatným odhadům času a ceny. Monolitický přístup byl přezdíván „Slaying (software) Dragon“ technika, protože předpokládá, že softwarový designer a vývojář je jediný hrdina, který má zabít draka sám. Prototypování může také snížit náklady a potíže při změnách hotového softwarového produktu.

Prototypování je jedním z přístupů, které popisuje [Frederick P. Brooks](#) ve své knize *Mythical Man-Month* z roku 1975 a článku *No Silver Bullet* vydaném o 10 let později.

K prvním příkladům rozsáhlého softwarového prototypování byla implementace překladače NYU Ada/ED pro [programovací jazyk Ada](#)^[2] implementovaný v [SETL](#) s úmyslem vytvořit spustitelný sémantický model pro jazyk Ada, který upřednostňoval jasnost návrhu a uživatelského rozhraní před rychlostí a efektivitou. NYU Ada/ED byla první potvrzená implementace jazyka Ada, která byla certifikována v 11. dubna 1983.^[3]

Náčrt procesu prototypování

Proces prototypování zahrnuje tyto kroky:

1. Identifikace základních [požadavků](#)

Stanovení základních požadavků včetně požadovaných vstupních a výstupních informací. Detaily jako například bezpečnost se v této fázi obvykle ignorují.

2. Vývoj počátečního prototypu

Je vyvinut počáteční prototyp, který obsahuje pouze uživatelské rozhraní. (Viz [horizontální prototyp](#) níže)

3. Revize

Zákazníci včetně koncových uživatelů zkontrolují prototyp a poskytnou zpětnou vazbu pro zapracování změn nebo doplnění vlastností.

4. Úprava a vylepšení prototypu

Využitím zpětné vazby lze vylepšit standardy i prototyp. Obvykle je nutné vyjednávání a upřesnění smlouvy a produktu. Kroky 3 a 4 se opakují, dokud je potřeba.

Dimenze prototypů

[Jakob Nielsen](#) rozlišil ve své knize *Inženýrství použitelnosti* tyto dimenze prototypů:

Horizontální prototyp

Obvyklý termín pro prototyp uživatelského rozhraní je **horizontální prototyp**. Poskytuje podrobný náhled na celý systém nebo subsystém zaměřený spíše na uživatelské interakce než na nízkourovňovou funkčnost systému, jako je například přístup k databázi. Horizontální prototypy jsou užitečné pro:

- potvrzení požadavků na uživatelské rozhraní a rozsah funkcí systému,
- ukázka systému pro získání zpětné vazby (případně i objednávek) od potenciálních zákazníků,
- předběžný odhady času, ceny a množství práce potřebné na vývoj systému.

Vertikální prototyp

Vertikální prototyp ([anglicky](#) *vertical prototype*) je úplnější propracování jediného subsystému nebo funkce. Je užitečný pro získání podrobných požadavků na danou funkci s následujícími výhodami:

- [Zjemnění](#) návrhu databáze,
- Získání informací o objemu dat a potřebách na systémové rozhraní, pro určení kapacity sítě a provedení inženýrství,
- vyjasnění složitých požadavků by rozebráním skutečné funkčnosti systému.

Typy prototypování

Softwarové prototypování má mnoho variant. Všechny však vycházejí ze dvou hlavních typů: zahazovací prototypování a evoluční prototypování.

Zahazovací prototypování

Také nazývané uzavřené prototypování. Zahazovací nebo [rapidní prototypování](#) znamená vytvoření modelu, který bude nakonec zahozen a nestane se součástí výsledného dodaného softwaru. Po shromáždění předběžných požadavků je zkonstruován jednoduchý pracovní model systému, aby bylo možné názorně ukázat uživatelům, jak budou jejich požadavky implementovány do výsledného systému.

Rapidní prototypování zahrnuje vytváření pracovního modelu různých složek systému ve velmi rané fázi, po relativně krátkém výzkumu. Metoda používaná při vytváření je obvykle dost neformální, nejdůležitějším faktorem je rychlost, kterou je model vytvořen. Model se pak stane východiskem, na kterém mohou uživatelé opětovně kontrolovat svá očekávání a vyjasňovat své požadavky. Když je tento cíl dosažen, prototyp model je 'zahozen', a systém je formálně vyvinut podle identifikovaných požadavků. [\[7\]](#)

Nejzjevnější důvod pro použití zahazovacího prototypování je, že může být provedeno rychle. Dostanou-li uživatelé rychlou zpětnou vazbu na své požadavky, mohou vývoj softwaru včas zkorrigovat. Provádění změn v časných fázích vývojového cyklu je velice cenově efektivní, protože nevyžaduje rozsáhlé změny. V pozdějších fázích projektu může i malá změna vyžadovat velké úsilí, protože softwarové systémy mají mnoho závislostí. Rychlost je klíčovou kvalitou při implementaci zahazovacího prototypu, protože s omezeným množstvím času a peněz lze malou část vynaložit na prototyp, který bude zahozený.

Další silnou stránkou zahazovacího prototypování je jeho schopnost zkonstruovat rozhraní, které uživatelé mohou testovat. Uživatelské rozhraní je tím, jak se uživateli systém jeví, a jeho včasným předvedením lze mnohem snáze porozumět, jak bude systém pracovat.

...je uváděno, že revoluční [rapidní prototypování](#) je efektivnější způsob, jak pracovat s uživatelskými požadavky, a proto je významným vylepšením celkové produktivity při

vytváření softwaru. Požadavky lze identifikovat, simulovat a testovat mnohem rychleji a levněji, pokud jsou problémy možnosti vývoje, údržby a struktury softwaru ignorovány. To pak vede k přesnému popisu požadavků a následné konstrukci funkčního a z uživatelského pohledu použitelného systému obvyklými metodami vývoje softwaru.^[8]

Prototypy lze klasifikovat podle věrnosti, s jakou zachycují skutečný produkt s ohledem na vzhled, interakce a časování. Jednou z metod vytváření zahazovacího prototypu s nízkou věrností je [papírové prototypování](#). Prototyp je implementován pomocí papíru a tužky, takže napodobuje funkci skutečného produktu, ale vůbec jako skutečný produkt nevypadá. Jinou metodou snadného vytvoření zahazovacího prototypu s vysokou věrností je použití [GUI Builderu](#) a vytvoření *klikací náhražky* - prototypu, který vypadá jako cílový systém, ale neposkytuje žádnou funkčnost.

Používání [storyboardů](#), animatik nebo kreseb není přesně totéž jako zahazovací prototypování, ale patří do stejné skupiny. Jde o nefunkční implementace, které ukazují, jak bude systém vypadat.

Souhrn: Při tomto přístupu je zkonstruován prototyp, který bude zahozen, a výsledný systém bude vytvořen z nuly. Kroky v tomto přístupu jsou:

1. Sepsání předběžných požadavků
2. Návrh prototypu
3. Uživatelské ověření prototypu, definice nových požadavků
4. Opakování postupu pokud je třeba
5. Sepsání výsledných požadavků

Evoluční prototypování

Evoluční prototypování ([anglicky](#) *Evolutionary prototyping*, *breadboard prototyping*) se zásadně liší od [zahazovacího prototypování](#). Jeho hlavním cílem je strukturovaná tvorba velmi robustního prototypu, který slouží jako jádro budoucího systému, a jeho postupné vylepšování. Principem tohoto přístupu je postupné budování a vylepšování nového systému a implementace dalších požadavků.

Při vývoji systému pomocí evolučního prototypování je systém neustále zjemňován a přestavován.

„...evoluční prototypování uznává, že nerozumíme všem požadavkům a implementujeme pouze ty, které dobře chápeme.“^[9]

Tato technika umožňuje, aby vývojový tým přidával vlastnosti nebo prováděl změny, které nemohly být formulovány ve fázi vytváření požadavků a návrhu.

Aby byl systém užitečný, musí být vyvíjen používáním ve svém pracovním prostředí. Výrobek není nikdy „hotov“; je vyhrává jak se mění prostředí jako použití... často se snažíme definovat systém pomocí nejznámějšího referenčního rámce - kde právě jsme. Činíme předpoklady o způsobu, jak bude věc fungovat a technologickém základě, na kterém bude implementována. Je ustanoven plán na vývoj funkčnosti, a dříve nebo později vznikne něco co se podobá požadovanému systému.^[9]

Výhodou evolučních prototypů oproti zahazovacím prototypům je, že se jedná o funkční systém. I když nemají všechny plánované vlastnosti, mohou být používány jako dočasné řešení, dokud nebude dodán výsledný systém.

„V rámci prototypování není neobvyklé dát uživateli počáteční prototyp k praktickému používání, dokud nebude k dispozici pokročilejší verze... Uživatel totiž často zjistí, že 'nedokonalý' systém je lepší než žádný.“^[7]

U evolučního prototypování se mohou vývojáři zaměřit na vývoj těch složek systému, kterým rozumějí, místo toho, aby pracovali na vývoji celého systému.

Aby se minimalizovalo riziko, vývojář neimplementuje vlastnosti, které zatím nejsou dobře pochopeny. K nasazení u zákazníka se posílá částečný systém. Uživatelé mohou při práci se systémem odhalit potřebu nových vlastností a vyžádat si jejich implementaci od vývojářů. Vývojáři pak shromažďují požadavky na vylepšení spolu se svými vlastními a používá spolehlivé metody pro správu konfigurace pro změnu standardu pro úpravu požadavků na software, aktualizaci návrhu, úpravu kódu a nové testování.^[10]

Inkrementální prototypování

Produkt se skládá z dílčích prototypů, které jsou na závěr sloučeny do výsledného řešení. Inkrementálním prototypováním lze překlenout časovou propast mezi uživateli a vývojáři.

Extrémní prototypování

Extrémní prototypování se používá zvláště pro vývoj WWW aplikací. Vývoj webu rozkládá do tří fází, z nichž každá buduje na předchozí. První fáze je statický prototyp, který sestává hlavně z HTML stránek. Ve druhé fázi jsou naprogramovány obrazovky a plně funkční rozhraní využívající simulovanou vrstvu služeb. Ve třetí fázi jsou implementovány služby. Název extrémní prototypování má zacílit pozornost na druhou fázi procesu, kdy je vyvíjeno plně funkční uživatelské rozhraní s minimálním zřetelem na jiné služby, než ty, které jsou uvedeny ve smlouvě.

Výhody prototypování

Prototypování má při vývoji softwaru mnoho výhod – některé jsou hmatatelné, jiné abstraktní.^[11]

Snížení času a nákladů: Prototypování může zlepšit kvalitu požadavků a standardů, které budou mít k dispozici vývojáři. Protože s pozdějším odhalením potřeby změn roste jejich cena exponenciálně, časné stanovení toho, *co uživatel skutečně chce* může vést k rychlejšímu a méně nákladnému vývoji softwaru.^[8]

Zlepšené a zvýšené zapojení uživatele: Prototypování vyžaduje zapojení uživatele, kterému umožňuje vidět a komunikovat s prototypem, což přináší lepší a úplnější zpětnou vazbu a přesnější popis požadavků a systému.^[7] Existence prototypu, který je vyzkoušen uživatelem, zabráni mnoha nedorozuměním a vzájemným nepochopením, ke kterým může dojít, když obě strany nekriticky věří, že druhá strana rozumí tomu, co řekli. Protože uživatelé znají problémovou oblast lépe než kdokoli ve vývojovém týmu, zvětšená interakce by měla vést k produktu, který má větší hmatatelnou i nehmotnou kvalitu. Výsledný produkt pravděpodobně bude lépe vyhovovat představám uživatele na vzhled, funkčnost a provedení.

Nevýhody prototypování

Prototypování může přinášet také nevýhody.

Nedostatečná analýza: Koncentrace na omezený prototyp může odvádět vývojáře od náležité analýzy kompletního projektu. To může vést k přehlédnutí lepších řešení, tvorbě neúplných popisů nebo přeměně omezených prototypů na špatně navržené výsledné produkty, které lze obtížně [udržovat](#). Pokud má prototyp omezenou funkčnost, nelze jej dobře škálovat, pokud se prototyp používá jako východisko výsledného produktu, což může zůstat nepovšimnuto, pokud jsou vývojáři příliš zaměřeni na vytváření prototypu jako modelu.

Uživatelské zmatení prototypu a výsledného systému: Uživatelé se mohou mylně domnívat, že prototyp který bude zahozen, je nevyladěným cílovým systémem. (Často si například nejsou vědomi, jaké úsilí je třeba vynaložit na doplnění chybových kontrol a bezpečnostních vlastností, který prototyp nemůže mít.) Kvůli tomu mohou předpokládat, že prototyp přesně modeluje výsledný systém, i když to není úmysl vývojářů. Uživatelé také mohou mylně spoléhat na vlastnosti, které byly obsaženy v prototypu pro jejich zhodnocení, a které nakonec byly z výsledného systému odstraněny. Pokud jsou uživatelé vyžadují, aby všechny navrhované vlastnosti byly obsaženy ve výsledném systému, může to vést ke konfliktu.

Nepochopení uživatelských potřeb ze strany vývojářů: Vývojáři mohou předpokládat, že uživatelé sdílí jejich cíle (například doručit základní funkčnost včas a dodržet rozpočet), bez porozumění širším komerčním otázkám. Například představitelé zákazníka účastníci se schůzek o [firemním softwaru](#) mohou pod vlivem ukázek „auditování transakcí“ (při kterém se zaznamenávají změny, které mohou být zobrazovány v tabulce rozdílů) objednat tuto vlastnost, aniž by si byli vědomi, že vyžaduje dodatečné kódování a obvykle i silnější hardware kvůli zpracování dodatečných přístupů k databázi. Uživatelé se mohou domnívat, že mohou požadovat auditování ve všech oblastech, zatímco vývojáři to mohou považovat za zbytnou nebo zbytečnou vlastnost, protože si již vytvořili představy o rozsahu uživatelských požadavků. Pokud se vývojáři zavázali k dodání této funkčnosti před provedením revize požadavků uživatelů, ocitnou se mezi dvěma mlýnskými kameny, obzvláště pokud má zákazník dojednány sankce při nesplnění implementace požadavků.

Lpění vývojářů na prototypu: Vývojáři mohou příliš lpět na prototypu, na jehož vytvoření vyvinuli velké úsilí; to může vést k problémům jako je snaha převést omezený prototyp na výsledný systém, přestože nemá vhodnou podkladovou architekturu. (To může napovídat, že by mělo být použito zahazovací prototypování místo evolučního.)

Nadměrný čas vývoje prototypu: Klíčovou vlastností prototypování je fakt, že se předpokládá, že bude uděláno rychle. Pokud vývojáři tento fakt ztratí ze zřetele, mohou se snažit vyvinout prototyp, který je příliš složitý. Když je prototyp zahozen přesně vyvinut požadavky, že poskytuje nemůže dává dostačující zvyšuje v produktivity vytvořit up pro čas strávený vývojem prototypu. Uživatelé mohou uvážnout v diskuzích o detailech prototypu, což zdržuje vývojový tým a opoždí dodání výsledného produktu.

Náklady na implementaci prototypování: počáteční náklady pro vytvoření vývojového týmu zaměřeného na prototypování mohou být vysoké. Mnoho společností má zavedenou určitou metodologii vývoje a její změna může vyžadovat přeškolení a změnu nástrojů. Mnoho firem se proto snaží zavést prototypování bez přeškolení svých zaměstnanců.

Obvyklým problémem s přijetím prototypování jsou vysoká očekávání produktivity s nedostatečným úsilím skrytým v učící křivce. Kromě tréninku pro používání prototypovacích technik, existuje často přehlížená potřeba pro firemní vývoj a projekt určitých podkladových struktur podporovat technologie. Pokud jsou tyto podkladové struktury vynechány, výsledkem může být nižší produktivita.^[13]

Nejlepší projekty používající prototypování

Přestože bývá uváděno, že určitá forma prototypování může být použita v každém projektu, prototypování přináší největší užitek u systémů, které mají mnoho interakcí s uživateli.

Bylo zjištěno, že prototypování je velmi efektivní pro analýzu a návrh online systémů, zvláště pro [transakční zpracování](#), kde používání obrazkových dialogů je mnohem více v evidenci. Čím je větší interakce mezi počítačem a uživatelem, tím větší je výhoda rychlého vytvoření systému, se kterým si uživatel může hrát.^[7]

U systémů s minimální uživatelskou interakcí, jako například [dávkové zpracování](#) nebo systémy, které většinou provádějí výpočty, nepřináší prototypování velké výhody. Někdy může být kódování potřebné pro vytvoření systémových funkcí příliš intenzivní a potenciální zisky, které by prototypování mohlo přinést, jsou příliš malé.^[7]

Prototypování je zvláště vhodné pro návrh dobrých rozhraní pro [Human-computer interaction](#). „Dosud je jedním z dosud nejproduktivnějších použití rapidního prototypování iterativní inženýrství uživatelských požadavků a návrh rozhraní člověk-počítač.“^[8]

Odkazy

Poznámky

1. [^](#) C. Melissa McClendon, Larry Regot, Gerri Akers: The Analysis and Prototyping of Effective Graphical User Interfaces. October 1996. ^[1]
2. [^](#) D.A. Stacy, professor, University of Guelph. Guelph, Ontario. Lecture notes on Rapid Prototyping. August, 1997. ^[2]
3. [^](#) Stephen J. Andriole: Information System Design Principles for the 90s: Getting it Right. AFCEA International Press, Fairfax, Virginia. 1990. Page 13.
4. [^](#) R. Charette, Software Engineering Risk Analysis and Management. McGraw Hill, New York, 1989.
5. [^](#) Alan M. Davis: Operational Prototyping: A new Development Approach. IEEE Software, September 1992. Page 71.
6. [^](#) Todd Grimm: The Human Condition: A Justification for Rapid Prototyping. Time Compression Technologies, vol. 3 no. 3. Accelerated Technologies, Inc. May 1998. Page 1. ^[3]
7. [^](#) John Crinnion: Evolutionary Systems Development, a practical guide to the use of prototyping within a structured systems methodology. Plenum Press, New York, 1991. Page 18.
8. [^](#) S. P. Overmyer: Revolutionary vs. Evolutionary Rapid Prototyping: Balancing Software Productivity and HCI Design Concerns. Center of Excellence in Command, Control, Communications and Intelligence (C3I), George Mason University, 4400 University Drive, Fairfax, Virginia.

9. [^](#) Software Productivity Consortium: Evolutionary Rapid Development. SPC document SPC-97057-CMC, version 01.00.04, June 1997. Herndon, Va. Page 6.
10. [^](#) Davis. Page 72-73. Citing: E. Bersoff and A. Davis, Impacts of Life Cycle Models of Software Configuration Management. Comm. ACM, Aug. 1991, pp. 104–118
11. [^](#) Adapted from C. Melissa McClendon, Larry Regot, Gerri Akers.
12. [^](#) Adapted from Software Productivity Consortium. PPS 10-13.
13. [^](#) Joseph E. Urban: Software Prototyping and Requirements Engineering. Rome Laboratory, Rome, NY.
14. [^](#) Paul W. Parry. Rapid Software Prototyping. Sheffield Hallam University, Sheffield, UK. [\[4\]](#) [Archivováno](#) 13. 2. 2019 na [Wayback Machine](#).
15. [^](#) Dr. Ramon Acosta, Carla Burns, William Rzepka, and James Sidoran. Applying Rapid Prototyping Techniques in the Requirements Engineering Environment. IEEE, 1994. [\[5\]](#)
16. [^](#) Software Productivity Solutions, Incorporated. Advanced Requirements Engineering Workstation (AREW). 1996. [\[6\]](#)
17. [^](#) from GE Research and Development.
https://web.archive.org/web/20061013220422/http://www.crd.ge.com/es/cgsp/fact_sheet/objorien/index.html
18. [^](#) Dynamic Systems Development Method Consortium.
<https://web.archive.org/web/20060209072841/http://na.dsdm.org/>
19. [^](#) Alan Dix, Janet Finlay, Gregory D. Abowd, Russell Beale; Human-Computer Interaction, Third edition

Reference

V tomto článku byl použit [překlad](#) textu z článku [Software prototyping](#) na anglické Wikipedii.

- • Smith MF *Software Prototyping: Adoption, Praxe a Management*. McGraw-Hill, Londýn (1991).
- • DEWAR, Robert B. K.; FISHER JR., Gerald.; SCHONBERG, Edmond; FROELICH, Robert; BRYANT, Stephen. NYU Ada Translátor a Interpreter. *ACM SIGPLAN Notices - Proceedings of ACM-SIGPLAN Symposium na Ada Programming Language*. November 1980. [ISBN 0-89791-030-3](#). [DOI 10.1145/948632.948659](#).
- • SofTech Inc., Waltham, MA. *Ada Compiler Validation Summary Report: NYU Ada/ED, Version 19.7 V-001* [online]. 1983-04-11 [cit. 2010-12-16]. [Dostupné v archivu](#) pořízeném dne 2012-03-12.