**UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE**

FAKULTA PRÍRODNÝCH VIED

Systém pre on-line publikovanie študijných materiálov a kníh

bakalárska práca

Študijný odbor: 9.2.9 Aplikovaná informatika

Študijný program: Aplikovaná informatika

Školiace pracovisko: Katedra informatiky

Školiteľ: Mgr. Martin Drlík, PhD.

Nitra 2015 Peter Baláž

# Obsah

[1 Analýza súčasného stavu skúmanej problematiky 3](#_Toc444895590)

[1.1 Vodopádový model 3](#_Toc444895591)

[1.2 Špirálový model 4](#_Toc444895592)

[1.3 Inkrementálny prístup 5](#_Toc444895593)

[1.4 Prototypový prístup 5](#_Toc444895594)

[1.5 Rapid application development 5](#_Toc444895595)

[1.6 Agílne metodiky 6](#_Toc444895596)

[1.6.1 Extrémne programovanie 7](#_Toc444895597)

[1.6.2 Lean development 9](#_Toc444895598)

[1.6.3 Vývoj riadený testami 10](#_Toc444895599)

[1.6.4 SCRUM Development process 10](#_Toc444895600)

[2 Ciele bakalárskej práce 12](#_Toc444895601)

[Zoznam bibliografických odkazov 13](#_Toc444895602)

# 1 Analýza súčasného stavu skúmanej problematiky

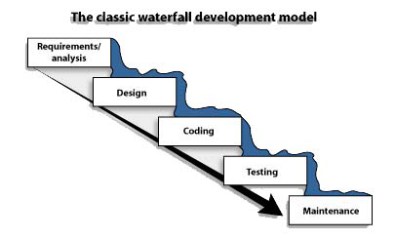
S rozmachom informatizácie neustále stúpa potreba kvalitnejších, ekonomickejších, rozšíriteľnejších a nezávislejších informačných systémov dodaných v čo najkratšom čase. Pre uspokojenie týchto požiadaviek je však potrebné zefektívniť procesy počas vývoja informačného systému, a preto vzniklo odvetvie informatiky nazývané softvérové inžinierstvo.

Softvérové inžinierstvo je štúdium a aplikácia inžinierstva na návrh, vývoj, implementáciu a údržbu softvéru systematickým spôsobom(Laplante, 2007). Sommerville(2007) softvérové inžinierstvo definuje ako inžiniersku disciplínu, ktorá sa zaoberá všetkými aspektami vývoja softvéru od raných fáz špecifikácie systému až po údržbu po zavedení softvéru do prevádzky. Softvérové inžinierstvo prešlo dlhým vývojom, čo malo za následok vznik a vývoj rôznych metodík tvorby softvéru a jej plánovania. Medzi najznámejšie prístupy plánovania vývoja softvéru patria:

* vodopádový prístup,
* špirálový prístup,
* inkrementálny prístup,
* prototypový prístup,
* Rapid Application Development,
* agílne prístupy.

## 1.1 Vodopádový model

Vodopádový model je pomenovaný podľa podobnosti grafického znázornenia priebehu s vodopádom. Jeho etapy na seba kaskádovito nadväzujú v presne určenom poradí. Každá fáza je ukončená kompletnou dokumentáciou a ďalšia môže začať až po ukončení predchádzajúcej.



Obrázok Grafické znázornenie vodopádového modelu[[1]](#footnote-1)

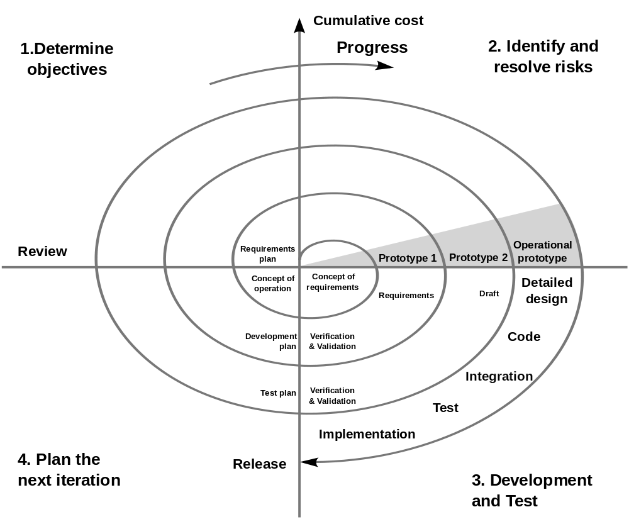
Zmeny sa vykonávajú len veľmi ťažko a po určení zmien zákazníkom je potrebné celý proces zopakovať. Ďalšou nevýhodou je, že zákazník nie je s vývojom softvéru takmer vôbec spätý, nemôže takmer vôbec ovplyvniť ani zadanie ani funkcionalitu. Častokrát sa teda stáva, že zákazník nedostane softvér vypracovaný podľa svojich predstáv čo vedie k pripomienkam a predlžovaniu času potrebného na vývoj takéhoto softvéru. Kvôli týmto nevýhodám je aj napriek svojej intuitívnosti a jednoduchosti vodopádový model takmer nepoužiteľný v skutočných projektoch, takže ho môžeme pokladať len za teoretický spôsob vývoja softvéru(Rajib, 2009). Napriek tomu je však často dodržiavaný pri písaní softvérovej dokumentácie. Vývoj softvéru využitím vodopádového modelu sa skladá z týchto etáp:

* štúdia uskutočniteľnosti,
* analýza požiadaviek a špecifikácia,
* dizajn,
* programovanie a testovanie,
* integrácia a systémové testovanie,
* údržba.

(Rajib, 2009).

## 1.2 Špirálový model

Špirálový model bol navrhnutý tak, aby zahŕňal najlepšie vlastnosti modelov prototypovania a vodopádu doplnené o nový komponent hodnotenia rizík. Termín špirála sa používa na opis procesu, ktorý prebieha pri vývoji softvéru.



Obrázok Špirálový model[[2]](#footnote-2)

V tomto modeli, podobne ako pri modeli prototypovania, je vyvinutá počiatočná verzia systému, ktorá je následne modifikovaná na základe výsledkov hodnotenia zákazníkom. Na rozdiel od modelu prototypovania, je však vývoje každej verzie narhnutý pomocou krokov zapojených do vodopádového modelu. Následne sa s každou iteráciou systém nabaľuje až pokým nie je kompletný a pripravený na prebratie zákazníkom.(State University of New York, 2016)

## 1.3 Inkrementálny prístup

Inkrementálny prístup je vhodný na kombináciu sekvenčných a iteračných metodík softvérového vývoja. Cieľom je obmedziť projektové riziká rozdelením projektu na menšie časti a zjednodušiť zavedenie zmien počas vývoja. Základné princípy inkrementálneho vývoja sú:

* sú robené série malých vodopádov, kde každý vodopád je určený pre určitú časť softvéru a je dokončený pred pokračovaním na ďalší prírastok, alebo
* všeobecné požiadavky sú definované skôr než sa pristúpi k evolučnému vývoju pomocou malých vodopádov, alebo
* prvotný koncept je definovaný pomocou vodopádu, nasleduje iteratívny prototypový prístup, ktorý vrcholí inštaláciou finálneho prototypu ako funkčného systému(CMS,2008).

## 1.4 Prototypový prístup

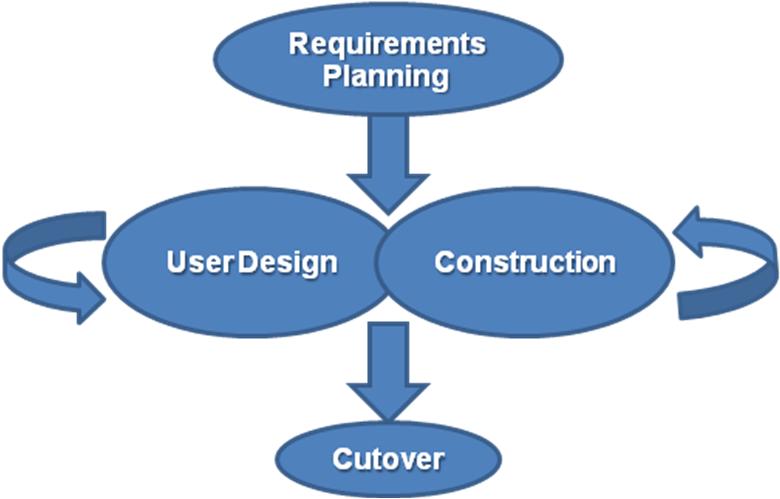
V prípade prototypového prístupu sa počas vývoja vytvára množstvo neúplných softvérových riešení tzv. prototypov ktoré postupne nadobúdajú finálnu podobu. Prototypový prístup nie je kompletnou metodikou vývoja, ale skôr prístupom k jednotlivým častiam väčších tradičných metodík. Používateľ je zapojený do celého procesu vývoja, čo znižuje riziko, že softvér bude treba prerábať a taktiež rozdelením projektu na menšie časti znižuje závažnosť projektových rizík. Na druhú stranu sa však súčasne vyvíja niekoľko prototypov, z ktorých veľká časť neprejde do samotného vývoja(CMS,2008).

## 1.5 Rapid application development

Rapid Application Development je odpoveďou na iné metodiky založené na vodopádovom prístupe. Jedným z problémov týchto metodík je, že sú založené na tradičných inžinierskych postupoch, ktoré sú používané na dizajnovanie a stavbu mostov alebo budov. Softvér je však úplne iným druhom subjektu. Softvér môže radikálne zmeniť celý proces použitý na riešenie problému. RAD sa však snaží tuto chýbajúcu flexibilitu pridať, a preto obsahuje procesy ktoré dovoľujú použiť získané poznatky z životného cyklu softvéru na pretvorenie riešenia(Brooks, 1986).

Martinov(1991) prístup k RAD rozdeľuje proces na štyri casti:

* fáza plánovania požiadaviek, ktorá zabezpečuje určenie potrieb, záberu, obmedzení a systémových požiadaviek,
* fáza používateľského dizajnovania, ktorá zabezpečuje návrh modelov a prototypov reprezentujúcich všetky procesy, vstupy a výstupy softvéru,
* konštrukčná fáza, ktorá sa zameriava na vývoj softvéru do ktorého naďalej zostávajú zapojení aj používatelia,
* a čistiaca fáza, počas ktorej sa vykonáva konverzia dát, testovanie a školenie používateľov.



Obrázok Fázy RAD podľa Martina[[3]](#footnote-3)

## 1.6 Agílne metodiky

Vo februári roku 2001 sa v rezorte Snowbird v americkom štáte Utah stretlo 17 softvérových vývojárov aby prediskutovali ľahké metodiky vývoja. Následne vydali publikáciu Manifesto for Agile Software Development v ktorej hovoria o uprednostnení individualít a interakcií pred procesmi a nástrojmi, funkčný softvér pred obsiahlou dokumentáciou, zákazníkovu spoluprácu pred vyjednávaním o kontrakte a reakciu na zmenu pred dodržiavaním plánu(Beck, 2001). Agílny manifest zakladá na dvanástich princípoch:

* Zákazníkova spokojnosť vďaka skorej a priebežnej dodávke softvéru.
* Vítanie zmien požiadaviek, a to aj v neskorých fázach vývoja.
* Funkčný softvér je dodávaný často.
* Blízka denná spolupráca medzi predajcami a vývojármi.
* Projekty sú postavené na motivovaných jedincoch.
* Konverzácia tvárou v tvár je najlepšia forma komunikácie.
* Funkčný softvér je hlavným meradlom pokroku.
* Udržateľný rozvoj, ktorý je schopný udržať konštantnú rýchlosť.
* Trvalá pozornosť k dobrému dizajnu a technickej dokonalosti.
* Jednoduchosť, ako umenie maximalizovať prácu ktorá ešte nebola spravená, je nevyhnutná.
* Najlepšie architektúry, požiadavky a návrhy pochádzajú zo seba organizovaných tímov.
* Tím pravidelne uvažuje o tom, ako byť efektívnejší, a podľa toho sa upravuje(Kent, 2001).

Podľa Larmana(2004) sú agilné metodiky porovnaní s tradičným softvérovým inžinierstvom, cielené na komplexné systémy a projekty s dynamickou, nedeterministickou a nelineárnou charakteristikou, kde presné odhady, stabilné plány a predpovede je v raných fázach ťažké presne určiť, a zároveň veľké dopredu navrhnuté dizajny a usporiadania by mohli spôsobiť veľké množstvo odpadu. To znamená, že nie sú ekonomicky stabilné. Tieto základné argumenty a skúsenosti v odvetví pomohli tvarovať agilné metodiky do ich súčasného tvaru.

### 1.6.1 Extrémne programovanie

Extrémne programovanie je jednou z najrozšírenejších agilných techník, nakoľko predstavuje predvídateľný, účinný a flexibilný spôsob vývoja softvéru no je vhodná najmä pre malé tými skladajúce sa z maximálne desiatich členov, ktoré sa musia rýchlo vysporiadať s nejasným a často sa meniacim zadaním. Základom extrémneho programovania je presvedčenie, že jediným zmerateľným, jednoznačným a overiteľným zdrojom informácií je zdrojový kód(Kadlec, 2004). Podľa Becka(1999) je extrémne programovanie o sociálnych zmenách. Ide v ňom o opustenie návykov a vzorov ktoré boli adaptívne v minulosti, ale dnes bránia vo vykonávaní práce čo najlepšie. Ide o to vzdať sa ochrany, ktorá interferuje s  produktivitou.

Extrémne programovanie vychádza z princípov a postupov bežných pri vývoji softvéru no privádza ich do extrémov. Pri extrémnom programovaní sa kód neustále reviduje vďaka využitiu párového programovania, kedy sa programátori navzájom kontrolujú, čo pomáha predísť chybám priebežne počas celej doby vývoja. Následne sa kód necháva testovať vývojármi v podobe jednotkových automatizovaných testov, a tiež používateľmi v podobe funkčných testov. Ďalším typickým znakom XP je refaktorizácia počas ktorej všetci každodenne upravujú a vylepšujú návrh softvéru. XP sa zároveň vyznačuje krátkymi iteráciami, ktoré nie sú týždne, mesiace či roky, ale sekundy, minúty, hodiny.

Extrémne programovanie upravuje tri tradičné premenné vývojového cyklu a pridáva štvrtú:

* kvalita,
* čas,
* náklady,
* šírka zadania(Shore, 2007).

Dôležitou myšlienkou XP je, že zákazníci, zadávatelia a manažéri zadávajú len ľubovoľné tri z týchto premenných a štvrtú, poslednú, premennú určuje vývojový tím. Ak zadávatelia s výslednou premennou nie sú spokojní, majú možnosť vstupné premenné predefinovávať do tej doby, kým nebudú spokojní výslednou hodnotou(Kadlec, 1999).

Ideovým základom XP je komunikácia, jednoduchosť, spätná väzba, odvaha a rešpekt čo však nestačí na vytvorenie funkčnej metodiky, kvôli čomu tiež obsahuje dvanásť základných postupov, ktoré vedú k vytvoreniu kvalitného softvéru:

* Plánovacia hra, ktorej cieľom je zapojiť celý tým do plánovania.
* Malá verzia, čo znamená, že nová verzia sa vydáva často.
* Metafora, ktorá slúži na popis funkcionality softvéru pomocou príbehu, čo zabezpečuje, že jej rozumejú všetky zainteresované strany.
* Jednoduchý návrh, teda najjednoduchší návrh, ako je danom momente možné.
* Testovanie, čiže spôsob ako udržať kvalitu aj napriek absencii explicitne definovaného návrhu.
* Refaktorizácia, ktorá pomáha zvyšovať kvalitu kódu jeho zjednodušovaním bez akejkoľvek zmeny chovania softvéru.
* Párové programovanie, pri ktorom zdrojový kód píše navzájom sa kontrolujúca dvojica programátorov.
* Spoločné vlastníctvo, hovoriace o myšlienke, že každý môže meniť ktorúkoľvek časť kódu v akomkoľvek čase.
* Nepretržitá integrácia, čo znamená, že systém je integrovaný, zostavený a testovaný minimálne raz denne.
* Štyridsať hodinový pracovný týždeň, nakoľko dlhodobé nadčasy vedú k znižovaniu efektivity.
* Skutočný zákazník alebo používateľ sa počas vývoja stáva súčasťou tímu.
* Dodržiavanie štandardov písania zdrojového kódu, čím je dosiahnutá vyššia čitateľnosť a kvalita zdrojového kódu.

### 1.6.2 Lean development

Shore(2007) definuje Lean Development ako systematický prístup k identifikácii a odstráneniu možného plytvania v priebehu celého vývoja, čím sa pokúša dodať zákazníkovi bezchybný produkt a uspokojiť tým jeho požiadavky čo znamená, že jeho hlavnou myšlienkou je odstránenie všetkého nadbytočného, čo v priebehu vývoja vzniká a môže negatívne ovplyvniť náklady, efektivitu a kvalitu výsledného produktu.

Lean Development sa dá zosumarizovať pomocou siedmych princípov, ktoré sú podobné konceptu štíhlej výroby:

* Odstránenie všetkého, čo zákazník nechce, nepoužije alebo pre neho nemá hodnotu.
* Posilnenie učenia pomocou krátkych iteračných cyklov a získavanie spätnej väzby od zákazníka.
* Odloženie závažných rozhodnutí na čo najpokročilejšiu dobu, kvôli možným rozhodnutiam zákazníka zmeniť požiadavku po dlhšom zvážení.
* Potreba dodať produkt v čo najkratšom čase, nakoľko je tým umožnené dostatočne skoro získavať spätnú väzbu a zmeny získane na jej základe začleniť do procesu vývoja.
* Dodanie potrebnej pozitívnej motivácie a prostriedkov tímu pre jeho posilnenie.
* Budovanie integrity systému, či už architektonickej, vnímanej alebo koncepčnej je ďalším dôležitým princípom, obsiahnutým v koncepte štíhlej výroby.
* Posledný z princípov hovorí o nutnosti vnímať produkt aj ako celok, nakoľko veľké systémy sú zložené z viacerých častí, ktoré nemusia byť vyvíjané tým istým tímom, a preto je potrebné zabezpečiť vzájomnú kompatibilitu súčastí(Poppendieck, M.;Poppendieck, T. 2003).

### 1.6.3 Vývoj riadený testami

Beck(2012), ktorý je považovaný za znovuobjaviteľa tejto metódy napísal, že po tom, čo napísal prvý xUnit framework v programovacom jazyku Smalltalk si spomenul na starú knihu v ktorej bol popísaný postup vývoja riadeného testami. Podľa tohto postupu bolo potrebné vziať pásku so vstupnými údajmi, manuálne napísať pásku s výstupnými údajmi podľa očakávaní a programovať dovtedy, pokým výstupná páska neobsahovala dáta podľa očakávaní.

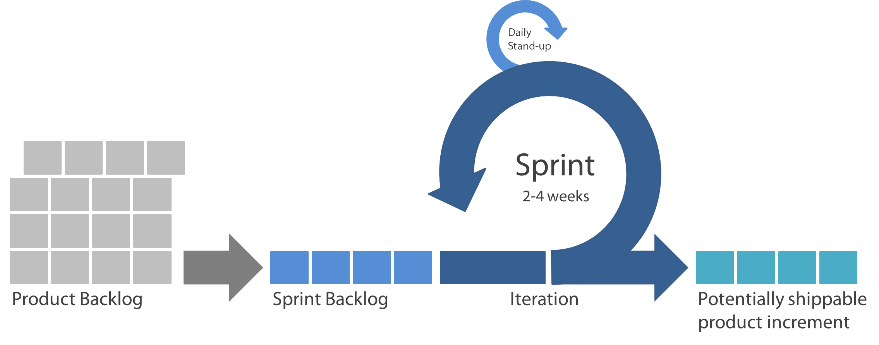
Test Driven Development je metóda, ktorej základom je rýchly cyklus testovania, programovania a refaktoringu. Pri pridávaní funkcionality môžu prejsť desiatky týchto cyklov programovania, zavádzania a zdokonaľovania softvéru po malých krokoch, pokým nezostane nič, čo by bolo potrebné pridať alebo odobrať. Pokiaľ je táto metóda použitá správne, pomáha v zdokonaľovaní dizajnu, dokumentovaní verejných rozhraní a predchádzaní budúcim omylom(Shore, 2007).

Podľa Beck-a(2003) pozostáva cyklus vývoja použitím tejto metódy z piatich opakujúcich sa krokov:

* Vytvorenie testu, ktorým v TDD začína každá nová funkcionalita.
* Spustenie všetkých testov pre zistenie, či novo vytvorený test neprejde testovaním, čo je v tomto prípade žiaduce.
* Napísanie kódu, ktorý novo vytvoreným testom prejde bez ohľadu na jeho kvalitu.
* Spustenie všetkých testov za účelom zistenia, či napísaná časť kódu testami prejde, a zároveň negatívne neovplyvní ostatné funkcionality.
* Refaktorovanie kódu pre zvýšenie jeho kvality odstránením duplicít, správnym pomenovaním metód a premenných prípadne úprava na tvar vďaka ktorému môže profitovať z rozpoznateľných návrhových vzorov.

### 1.6.4 SCRUM Development process

Pôvod slova scrum je možné nájsť v športe nazývanom ragby, kde sa ním označuje metóda pri reštarte hry keď sú hráči natlačení tesne pri sebe s hlavami dole s cieľom zmocniť sa lopty. Táto metodika sa radí medzi agilné a je definovaná ako flexibilná stratégia pre vývoj produktu, kde celý tím pracuje ako jednotlivec pre dosiahnutie spoločného cieľa(Takeuchi;Nonaka, 1986).



Obrázok Grafické znázornenie SCRUM procesu[[4]](#footnote-4)

Vývoj podľa tejto metodiky prebieha v presne daných časových úsekoch nazývaných šprinty. Každý šprint trvá mesiac alebo menej a po jeho skončení je tím schopný dodať funkčný a otestovaný produkt, následne začína nové plánovanie a cyklus sa opakuje až pokým nie je produkt dokončený.

Hlavným dokumentom SCRUM-u je Backlog, ktorý tvorí utriedený list všetkého, čo môže byť potrebné v produkte a majiteľ produktu zodpovedá za jeho obsah, dostupnosť a utriedenie. Tým ako je produkt používaný, naberá na hodnote a trh ponúka spätnú väzbu, Backlog rastie a je vyťažujúcejší(Schwaber; Sutherland, 2014). Metodika uvádza tri základné role ktoré sa na vývoji podieľajú. Prvou je majiteľ produktu, ktorého hlavnou úlohou je komunikácia, vďaka ktorej je schopný zžiť sa s členmi tímu a ostatnými zúčastnenými stranami, čo vedie k riadeniu projektu správnym smerom(Pichler, 2010). Nasledujúcou rolou je vývojársky tím, ktorý je zodpovedný za dodávku potenciálne publikovateľného produktu na konci každého šprintu. Vývojárske tímy musia byť zostavené tak, aby ich členovia mali všetky potrebné schopnosti na dokončenie celého prírastku. Poslednou rolou je vedúci scrum-u. Jeho úlohou je odstraňovať prekážky, ktoré bránia tímu v dokončení cieľov a zároveň dohliada na to, aby scrum prebiehal tak, ako bolo zamýšľané. Zároveň je jeho náplňou práce aj poskytovať služby ako majiteľovi produktu, tak vývojárskemu tímu ako napríklad:

* nachádza spôsoby efektívnej tvorby produktového Backlogu,
* chápe produktového plánovania v empirickom prostredí,
* zabezpečuje, aby majiteľ produktu vedel zostaviť Backlog tak, aby bola maximalizovaná jeho hodnota,
* upravuje udalosti scrumu podľa požiadaviek alebo potreby,
* organizuje vývojársky tím,
* pomáha tímu vo vytváraní produktov s vysokou kvalitou(Schwaber; Sutherland, 2014).

# 2 Ciele bakalárskej práce

Cieľom bakalárskej práce je navrhnúť a vytvoriť webovú aplikáciu pre elektronické publikovanie publikácií, ktorá umožní autorom postupne zverejňovať jednotlivé kapitoly publikácií, sprístupňovať ich registrovaným čitateľom a získavať spätnú väzbu.

**ČIASTKOVÉ CIELE:**

* filtrovať zdroje,
* analyzovať súčasný stav danej problematiky,
* navrhnúť použitie technológií,
* navrhnúť diagramy podľa UML štandardu,
* navrhnúť databázovú štruktúru,
* navrhnúť používateľské prostredie,
* implementovať navrhnuté riešenie
* odporučiť pre prax

# Zoznam bibliografických odkazov

LAPLANTE,P.A. 2007. What every engineer should know about Software Engineering. CRC Press, 2007. 328 s. ISBN 978-0-8493-7228-5.

SOMMERVILLE, I. 2007. What is software engineering?. Pearson Education, 2007. 840 s. ISBN 0-321-31379-8.

RAJIB, M. 2009. Fundamentals of software engineering. PHI Learning Pvt. Ltd., 2009. 464 s. ISBN 978-81-203-3819-7.

STATE UNIVERSITY OF NEW YORK. 2016. A survey of system development process models. [online]. 2016. [cit.: 2016-02-29]. Dostupné na internete: <https://www.ctg.albany.edu/publications/reports/survey\_of\_sysdev?chapter=9>.

Centers for Medicare & Medicaid Services. 2008. Selecting a development approach. [online]. 2005. [cit.: 2016-03-01]. Dostupné na internete: <https://www.cms.gov/research-statistics-data-and-systems/cms-information-technology/xlc/downloads/selectingdevelopmentapproach.pdf>.

BECK, K. a kol. 2001. Manifesto for Agile Software Development. [online]. Agile Alliance, 2001. [cit.:2016-03-01]. Dostupné na internete: <http://www.agilemanifesto.org/iso/sk/>

BECK, K. a kol. 2001. Principles behind the Agile Manifesto. [online]. Agile Alliance, 2001. [cit.:2016-03-01]. Dostupné na internete: <http://www.agilemanifesto.org/principles.html>

MARTIN, J. 1991. Rapid Application Development. Macmillan, 1991. 816 s. ISBN 0‑02‑376775‑8.

LARMAN, C. 2004. Agile and Iterative Development: A Manager’s Guide. Addison-Wesley, 2004. 368 s. ISBN 978-0-13-111155-4.

KADLEC, V. 2004. Agilní programování: metodiky efektivního vývoje softwaru. Computer Press, 2004. 278 s. ISBN 80-251-0342-0.

BECK, K. 1999. Extreme Programming Explained. Addison-Wesley, 1999. 224 s. ISBN 978-0-201-61641-5.

Shore, J. 2007. The Art of Agile Development. O’Reilly Media, 2007. 440 s. ISBN 978-0-5965-2767-9.

Beck, K. 2002. Extrémní programování. Grada Publishing, 2002. 160 s.   
ISBN 80-247-0300-9.

TAKEUCHI, H.;NONAKA, I. 1986. New New Product Development Game. [online]. Harvard Business Review, 1986. [cit. 2016-03-04]. Dostupné na internete: <http://www.enterprisescrum.com/publications/The%20New%20New%20Product%20Development%20Game%20-%20Nonaka%20and%20Takeuchi.pdf>

POPPENDIECK, M.; POPPENDIECK, T. 2003. Lean Software Development: An Agile Toolkit. Addison-Wesley, 2003. 240 s. ISBN 978-0-321-15078-3.

BECK, K. 2012. Why does Kent Beck refer to the "rediscovery" of test-driven development?. [online]. 2012. [cit. 2016-03-03]. Dostupné na internete: <https://www.quora.com/Why-does-Kent-Beck-refer-to-the-rediscovery-of-test-driven-development>

BECK, K. 2003. Test Driven Development: By Example. Pearson Education, 2003. 220 s. ISBN 0-321-14653-0

PICHLER, R. 2010. Agile Product Management with Scrum: Creating Products that Customers Love. Addison-Wesley, 2010. 160 s. ISBN 978-0-321-60578-8.

SCHWABER, K.; SUTHERLAND, J. 2014. The Scrum Guide. [online]. Scrum Alliance, 2014. [cit. 2016-03-04]. Dostupné na internete: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf>.

1. Zdroj: https://embeddedsystemsvvce.files.wordpress.com/2013/02/waterfall-copy1.jpg?w=395 [↑](#footnote-ref-1)
2. Zdroj: https://upload.wikimedia.org/wikipedia/commons/thumb/e/ec/Spiral\_model\_(Boehm,\_1988).svg/1000px-Spiral\_model\_(Boehm,\_1988).svg.png [↑](#footnote-ref-2)
3. Zdroj: https://upload.wikimedia.org/wikipedia/commons/5/5f/RADModel.JPG [↑](#footnote-ref-3)
4. Zdroj: http://www.zenexmachina.com/assets/themes/zxm/images/page-artefacts/scrum-process-01.png [↑](#footnote-ref-4)