

Univerzita Karlova
Pedagogická fakulta

Katedra informačních technologií a technické výchovy

BAKALÁŘSKÁ PRÁCE

Porovnání metodik vývoje software: Waterfall vs. agilní přístupy
Comparison of Software Development Methodologies: Waterfall vs. Agile
Approaches

David Sochor

Vedoucí práce: PhDr. Jiří Štípek, Ph.D.
Studijní program: Informační technologie se zaměřením na vzdělávání

2025

Odevzdáním této bakalářské práce na téma Porovnání metodik vývoje software: Waterfall vs. agilní přístupy potvrzuji, že jsem ji vypracoval pod vedením vedoucího práce samostatně za použití v práci uvedených pramenů a literatury. Prohlašuji, že jsem při její tvorbě nepoužil nástrojů umělé inteligence jiným způsobem, než je uvedeno ve vyjádření, které je součástí textu práce. Dále potvrzuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

Praha 14.4.2025

Rád bych poděkoval vedoucímu své bakalářské práce PhDr. Jiřímu Štípkovi, Ph.D., který mi byl velkou oporou po celou dobu jejího psaní. Velmi si vážím jeho ochoty konzultovat i ve večerních hodinách a toho, že si vždy našel čas, kdykoli jsem o pomoc požádal. Jeho vstřícnost, cenné rady a podpora pro mě byly neocenitelné a výrazně přispěly k úspěšnému dokončení této práce.

ABSTRAKT

V oblasti softwarového vývoje existují různé metodiky řízení projektů, které ovlivňují efektivitu práce, kvalitu výsledného produktu a schopnost reagovat na změny požadavků. Tato bakalářská práce se zaměřuje na analýzu a porovnání tradičního vodopádového modelu (Waterfall) s agilními přístupy, jako jsou Scrum, Kanban, XP, TDD a Lean. Hlavním cílem je objasnit jejich silné a slabé stránky, vhodnost použití v různých typech projektů a klíčové faktory ovlivňující jejich výběr.

Práce diskutuje systematický manuál pro rozhodování o výběru metodiky na základě konkrétních kritérií, jako jsou flexibilita, řízení rizik, délka vývojového cyklu a velikost projektu. Manuál je následně aplikován na případovou studii, která prakticky demonstruje jeho využití. Výsledky ukazují, že neexistuje univerzální řešení – každá metodika má své specifické uplatnění. Správný výběr metodiky je klíčový pro úspěch softwarového projektu, a proto je důležité nejen rozumět jednotlivým přístupům, ale i umět je strategicky aplikovat.

KLÍČOVÁ SLOVA:

Metodika vývoje software, Softwarový vývoj, Lineární vývojový proces, Iterativní vývoj, Produktivita týmu, Plánování v softwarovém vývoji, Agilní metodiky, Vodopádový model, Strukturovaný vývoj software

ABSTRACT

Various project management methodologies in software development affect work efficiency, product quality, and the ability to respond to changing requirements. This bachelor thesis focuses on analysing and comparing the traditional Waterfall model with agile approaches such as Scrum, Kanban, XP, TDD, and Lean. The primary goal is to clarify their strengths and weaknesses, suitability for different types of projects, and key factors influencing their selection.

The thesis provides a systematic manual for decision-making in methodology selection based on specific criteria such as flexibility, risk management, development cycle length, and project size. This manual is then applied to a case study, practically demonstrating its use. The results indicate no universal solution—each methodology has its specific application. Choosing the proper methodology is crucial for the success of a software project, making it essential to understand different approaches and to apply them strategically.

KEYWORDS:

Software development methodology, Software development, Linear development process, Iterative development, Team productivity, Software development planning, Agile methods, Waterfall model, Structured software development

Obsah

1	ÚVOD	7
2	CÍL PRÁCE	7
3	TEORETICKÁ ČÁST	9
3.1	Teoretická analýza	9
3.1.1	Historický kontext metodik softwarového vývoje	9
3.1.2	Principy a základní charakteristiky metodik softwarového vývoje	10
3.1.2.1	Waterfall	10
3.1.2.2	Scrum	13
3.1.2.3	Kanban.....	17
3.1.2.4	Extrémní programování (XP).....	19
3.1.2.5	Test-driven Development (TDD)	21
3.1.2.6	Lean Software Development.....	23
3.1.3	Výhody a nevýhody jednotlivých metodik	25
3.2	Srovnávací kritéria	26
3.3	Srovnání metodik dle stanovených kritérií	27
3.3.1	Hodnocení metodik.....	27
3.3.2	Srovnání jednotlivých kritérií a metodik.....	31
3.4	Vhodnost použití jednotlivých metodik v různých typech projektů	32
4	PRAKTICKÁ ČÁST – MANUÁL PRO VOLBU METODIKY	34
4.1	Doporučený postup výběru metodiky	34
4.2	Shrnutí.....	36
5	PŘÍPADOVÁ STUDIE – VOLBA METODIKY NA REÁLNÉM PROJEKTU	37
5.1	Zadání projektu	37
5.1.1	Funkcionalita aplikace:.....	37
5.2	Volba metodiky	37
6	ZÁVĚR	39

1 Úvod

V dnešním dynamickém prostředí softwarového vývoje se obchodní společnosti i státní organizace neustále snaží nalézt optimální způsob řízení svých projektů. Efektivní řízení projektů je klíčové pro dosažení úzce definovaných cílů, jako jsou dodržení rozpočtu, časových harmonogramů a kvality výsledného produktu. V průběhu let vzniklo mnoho různých metodik, které se snaží tyto aspekty optimalizovat a přizpůsobit se různým typům projektů i organizačním kulturám.

Metodiky projektového řízení jsou souborem principů, postupů a praktik, jejichž cílem je zajistit efektivní vývoj softwaru. Správně zvolená metodika může podpořit hladký průběh projektu, minimalizovat rizika a usnadnit adaptaci na změny v zadání. Naopak nevhodně zvolená metodika může způsobit celou řadu komplikací, včetně prodloužení doby vývoje, nárůstu nákladů a snížení kvality výsledného produktu. Proto je důležité dobře porozumět jednotlivým metodikám a jejich vhodnosti pro různé typy softwarových projektů.

Tradiční přístupy, jejichž příkladem je waterfall, nabízejí pevně definovanou strukturu a důraz na důkladné plánování, avšak mohou někdy postrádat flexibilitu při změně požadavků. Naopak agilní metodiky, jako jsou například Scrum a Kanban, se zaměřují na iterativní vývoj, spolupráci a rychlou adaptaci na změny. Každá z těchto metodik má své přednosti i omezení a jejich vhodnost se liší dle konkrétních požadavků a charakteru projektu.

Tato práce se zaměřuje na analýzu a porovnání těchto přístupů s cílem poskytnout ucelený pohled na jejich vlastnosti, silné a slabé stránky a jejich vhodnost pro různé typy softwarových projektů.

Vzhledem k tomu, že text práce obsahuje odborné termíny z oblasti softwarového inženýrství a projektového řízení, které nemusí být čtenáři vždy známé, je součástí dokumentu také slovníček pojmů umístěný v příloze. Pokud se během čtení objeví neznámý výraz, lze jeho základní vysvětlení vyhledat právě tam. Slovníček slouží jako doplňkový nástroj pro lepší porozumění obsahu práce a orientaci v používané terminologii.

2 Cíl práce

V oblasti softwarového vývoje se organizace i jednotlivé týmy často potýkají s otázkou, jakou metodiku řízení projektu zvolit. Tradiční waterfall i agilní přístupy mají své výhody i omezení, avšak špatná volba metodiky může vést k řadě problémů – zpoždění projektu, vyšším nákladům, komplikované komunikaci mezi vývojáři a zadavateli, či dokonce k neúspěchu celého projektu. V praxi se často stává, že IT odborníci metodiky sice znají, ale jejich výběr neprobíhá systematicky. Často nejsou zohledněny všechny klíčové aspekty projektu, což vede k volbě nevhodné metodiky. Důvodem pro volbu metodiky může být

například již předchozí zkušenost týmu, bez hlubší analýzy skutečných potřeb. Chybí jasná pravidla výběru metodiky, která by umožnila rozhodování na základě konkrétních kritérií, reflektujících povahu zakázky.

Hlavním cílem této práce je analyzovat a porovnat dvě hlavní kategorie metodik softwarového vývoje – tradiční vodopádový (waterfall) přístup a agilní metodiky (Scrum, Kanban, XP, TDD, Lean).

Práce bude systematicky hodnotit tyto přístupy z hlediska jejich principů, silných a slabých stránek, aplikovatelnosti na různé typy projektů a efektivity v praxi. Důležitou součástí bude návrh kritérií pro výběr metodiky podle povahy zakázky či projektu, který pomůže IT týmům při rozhodování.

Výstupem práce bude nejen analýza jednotlivých metodik, ale i návrh systematického postupu pro jejich výběr, tedy jakási „metodika výběru metodiky“. Za účelem jednoduchosti a snazšího porozumění budeme nadále nazývat tuto metodiku pro výběr metodiky manuálem. Tento manuál bude následně aplikován na konkrétní ilustrativní příklad z praxe, čímž dojde k jeho lepšímu pochopení a představě o jeho využití v reálném prostředí.

Práce bude postupovat následujícím způsobem:

1. Nejprve bude provedena podrobná analýza jednotlivých metodik softwarového vývoje.
2. Na základě této analýzy budou identifikována klíčová kritéria, podle kterých lze volit vzájemně metodiky srovnávat.
3. Bude navržen systematický postup pro výběr metodiky, který zohlední daná kritéria – manuál pro volbu metodiky.
4. Pro snazší pochopení navrženého přístupu bude tento postup aplikován na konkrétní případovou studii.

Tento nově vytvořený manuál pro volbu metodiky poskytne přehledný a prakticky využitelný postup pro softwarové týmy a organizace, které hledají optimální metodiku pro své projekty.

3 Teoretická část

Metodika v kontextu projektového managementu a softwarového vývoje představuje soubor pravidel, postupů a technik, které jsou aplikovány k efektivnímu řízení a realizaci projektů. Cílem metodik je strukturovat práci tak, aby byla maximálně efektivní, předvídatelná a vedla k očekávaným výsledkům. V projektovém managementu je metodika nástrojem, který pomáhá řídit proces od plánování přes realizaci až po ukončení projektu, a to se zohledněním zdrojů, času, nákladů a kvality výsledného produktu. V oblasti softwarového vývoje je metodika ještě důležitější, protože tento obor často řeší komplexní a dynamické projekty, které vyžadují jasný rámec pro řízení vývoje, testování a nasazení softwaru.

Všechny tyto metodiky mají za cíl nejen zajistit, že projekt bude dokončen včas a v rámci rozpočtu, ale také, že bude schopný reagovat na změny v zadání a přinášet maximální hodnotu zadavateli. Výběr vhodné metodiky závisí na konkrétních potřebách projektu, povaze týmu a prostředí, ve kterém je projekt realizován.

Pro srovnání metodik vývoje softwaru bude použita kombinace teoretické analýzy, srovnávacích kritérií a analýzy odborných zdrojů.

3.1 Teoretická analýza

3.1.1 Historický kontext metodik softwarového vývoje

Historie metodik softwarového vývoje sahá do poloviny 20. století, kdy se objevily první snahy o strukturovaný přístup k tvorbě softwaru. V 50. a 60. letech se softwarové inženýrství začalo formovat jako samostatná disciplína, avšak používané metodiky byly často neformální a přizpůsobovaly se aktuálním potřebám (DYADE, 2024). První strukturované přístupy byly inspirovány tradičními inženýrskými obory a kladly důraz na důkladné plánování a pevně stanovené fáze vývoje.

V 70. letech byl poprvé popsán model waterfall (ROYCE, 1970), který se stal standardem pro řízení softwarových projektů. Tento lineární a sekvenční model předpokládal, že jednotlivé fáze vývoje – analýza, návrh, implementace, testování, nasazení a údržba – probíhají postupně za sebou, přičemž každá fáze musí být dokončena před zahájením následující. Tento přístup vycházel z přesvědčení, že software lze vyvíjet obdobně jako fyzické inženýrské produkty s jasně definovanými etapami (A BRIEF HISTORY OF SOFTWARE DEVELOPMENT METHODOLOGIES, 2021).

V 80. a 90. letech se ukázalo, že rigidní struktura modelu waterfall nevyhovuje rychle se měnícím požadavkům zákazníků během samotného vývoje. V reakci na to byly vyvinuty iterativní modely, jako například Spiral Model představený Barrym Boehmem v roce 1986, který umožňoval cyklický vývoj s průběžnou zpětnou vazbou a opakovaným procházením jednotlivých fází. (A BRIEF HISTORY OF SOFTWARE DEVELOPMENT METHODOLOGIES, 2021)

Na přelomu tisíciletí, v roce 2001, byla publikována Agilní deklarace (Agile Manifesto), která definovala nové principy vývoje softwaru. Tyto principy kladly důraz na spolupráci, adaptabilitu a časté dodávání funkčních verzí produktu. Metodiky jako Scrum, Kanban, Extreme Programming (XP) a Test-Driven Development (TDD) se staly populárními díky své schopnosti rychleji reagovat na změny v požadavcích zákazníků a zlepšovat efektivitu vývoje. (AHIGHSMITH, 2001)

V současnosti jsou stále častější hybridní přístupy kombinující prvky modelu waterfall a agilních metodik, přičemž organizace hledají rovnováhu mezi strukturou a flexibilitou. Čistě agilní přístupy mohou někdy vést k nedostatečné dokumentaci, která je však nezbytná pro interní i externí audity a zajištění souladu s interními nařízeními a normami (KIRPITSAS a PACHIDIS, 2022). Proto se například využívá plánovací fáze a definice požadavků z modelu waterfall jako prvního kroku, zatímco samotný vývoj probíhá například pomocí kombinace metodik Scrum a Kanban.

3.1.2 Principy a základní charakteristiky metodik softwarového vývoje

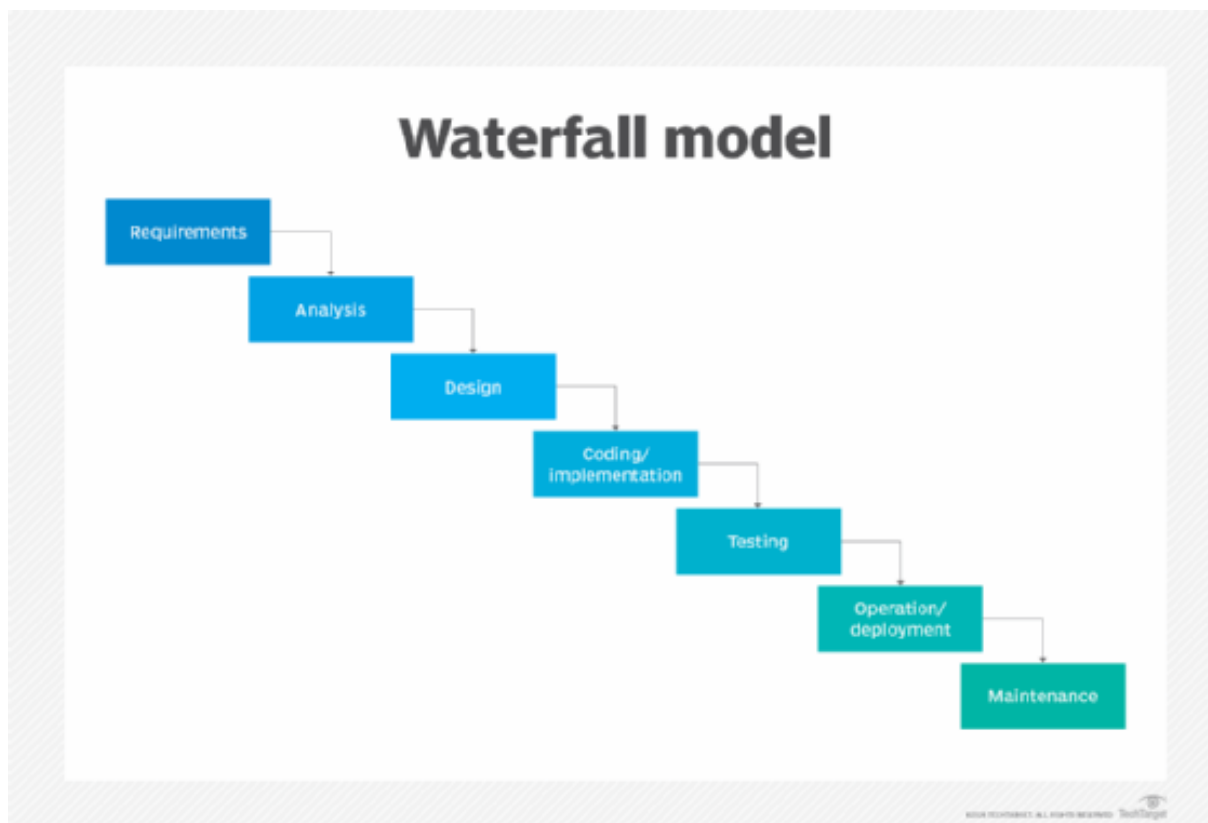
Metodiky ve své práci rozdělují na dvě základní kategorie, podle jejich přístupu k plánování a realizaci projektu.

Waterfall (vodopádový model) je tradiční metodika projektového řízení a vývoje softwaru, která postupuje lineárně, přičemž každá fáze projektu musí být dokončena před přechodem do další. Tento přístup vychází z myšlenky, že projektové fáze na sebe navazují, podobně jako voda teče dolů po vodopádu, a jednou překonaná fáze již není snadno dostupná pro změny. Waterfall je vhodný především pro projekty, kde jsou požadavky na začátku dobře definovány a nebudou se měnit v průběhu vývoje.

Agilní metodiky jsou zaměřeny na flexibilitu, rychlou odezvu na změny v zadání projektu či potřeby uživatelů a průběžné doručování funkčních částí produktu. Namísto dlouhých fází a pevně daného plánu, jak je tomu u waterfallu, agilní metodiky rozdělují vývoj do krátkých iterací s důrazem na spolupráci, testování a možnost rychle upravit plán vývoje podle potřeb zadavatele.

3.1.2.1 Waterfall

Waterfall model byl formálně definován v roce 1970 Winstonem W. Roycem. Metodika je detailně popsána například v knihách "Software Engineering: A Practitioner's Approach" (PRESSMAN, 2001) a "The Mythical Man-Month" (BROOKS, 1995).



Obrázek 1: Fáze modelu waterfall

Základní fáze waterfall přístupu (MCCORMICK, 2012):

1. **Plánování a analýza požadavků:** Tato fáze zahrnuje sběr a analýzu požadavků od zadavatele nebo zúčastněných stran. Výsledkem je dokumentace, která definuje všechny funkce a požadavky na projekt.
2. **Návrh:** Na základě analýzy požadavků se vytváří architektonický a detailní návrh řešení, který zahrnuje technické specifikace, datové modely a návrh uživatelského rozhraní. Tento návrh slouží jako podklad pro vývoj.
3. **Implementace (vývoj):** V této fázi se návrh převádí do zdrojového kódu. Tým vývojářů pracuje na vytváření funkcionalit podle specifikace z předchozí fáze.
4. **Testování:** Po implementaci následuje fáze testování, kde je ověřováno, zda software splňuje všechny požadavky definované v první fázi. Testování zahrnuje jednotkové testy, integraci systému, testování výkonu a další metody ověření kvality.
5. **Nasazení a implementace:** Po úspěšném testování je produkt připraven na nasazení do provozního prostředí. Tato fáze zahrnuje instalaci softwaru u uživatele, školení uživatelů a přípravu na jeho plné využívání.
6. **Údržba:** I po nasazení softwaru do provozu je nutné jej udržovat. Údržba zahrnuje opravy chyb, aktualizace a úpravy systému na základě nových požadavků.

Waterfall model je založen na lineárním přístupu k vývoji softwaru, což znamená, že každá fáze musí být dokončena, než se přistoupí k následující (OBRÁZEK 1). Tento model je

ideální pro projekty s jasně definovanými a stabilními požadavky, což je časté v regulovaných oblastech, jako je zdravotnictví, bankovníctví nebo státní správa (PRESSMAN, 2001). V těchto oblastech je kladen důraz na detailní dokumentaci a přísnou kontrolu kvality, což waterfall umožňuje díky své přehledné struktuře.

Každá fáze je nejprve pečlivě dokumentována, což zajišťuje přehlednost, ale může zpomalovat adaptaci v případě změny zadání či situace zadavatele. V praxi se ukazuje, že některé projekty zůstanou velmi dlouhou dobu ve fázi návrhu a plánování, protože je téměř nemožné dokumentaci dostat do finálního stavu před tím, než přijde změnový požadavek. Každá fáze vyžaduje formální schválení a vytváření detailní dokumentace, jako jsou detailní design, systémové požadavky, architektonický návrh, testovací plány a další.

Přestože tento model poskytuje vysokou úroveň kontroly a sledovatelnosti, jeho rigidita může být jeho největší slabinou. Jakýkoliv požadavek na změnu během vývoje může znamenat nutnost vrátit se k předchozím fázím, což je nákladné a časově náročné (MOKHTAR & KHAYYAT, 2022). Tato nevýhoda je obzvláště patrná v prostředí, kde se požadavky vyvíjejí a mění v průběhu projektu. Projekty, které se musí přizpůsobovat rychle změněným podmínkám, mohou být takto limitovány.

V rámci tohoto modelu existují jasně definované role, které zajišťují, že každý krok je prováděn správně a ve správném pořadí. Projektový manažer je zodpovědný za celkové řízení projektu, včetně plánování časového harmonogramu a monitorování dodržování metodiky. Business analytik se zaměřuje na sběr a definování požadavků od zadavatele, což slouží jako základ pro další fáze vývoje. Vývojář vytváří software podle předem definovaných specifikací a tester se zaměřuje na ověření kvality výstupů během testovací fáze. Tester testuje produkt jak během vývoje, tak i ve finální verzi během fáze testování. Poskytuje zpětnou vazbu vývojářům a jeho práce slouží jako podklad pro rozhodnutí, zda je produkt dokončen a splňuje zadání (SERGEEV, 2016). Každý z těchto členů týmu má svou specifickou odpovědnost a jejich spolupráce je nezbytná pro úspěšný vývoj.

Praktické použití:

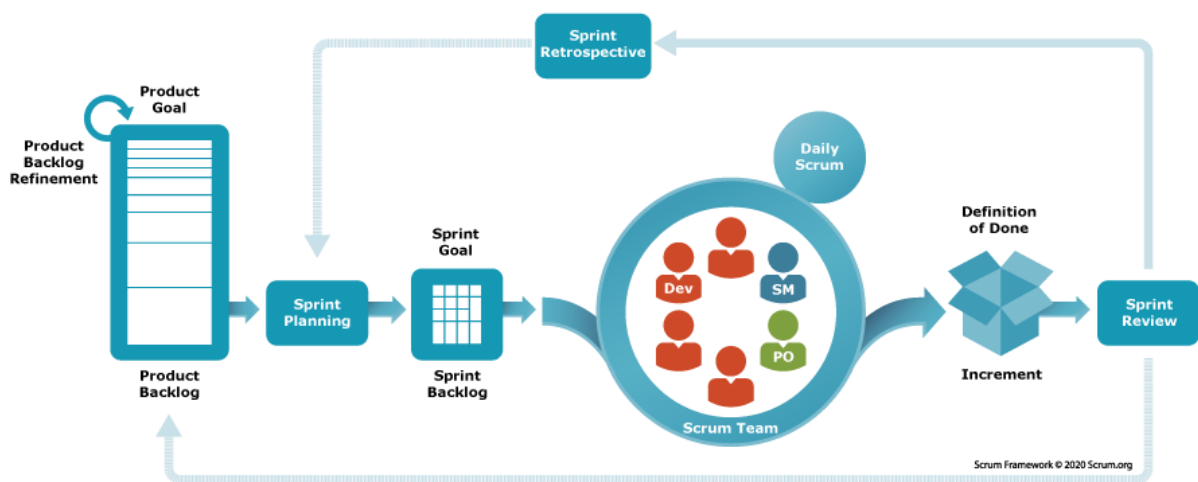
- Vhodné pro projekty s pevnými požadavky (např. ERP systémy, vládní software).
 - Tento přístup je vhodný pro projekty, kde jsou na začátku jasně definovány požadavky, které se v průběhu projektu nemění. V případě, že se některé požadavky změny v průběhu projektu, může být nutné zahájit projekt znovu od první fáze.
- Nevhodné pro rychle se měnící prostředí (např. startupy, webové aplikace).
 - Waterfall není flexibilní a není vhodný pro projekty, které vyžadují rychlé změny a přizpůsobení.

3.1.2.2 Scrum

Scrum je agilní metodika zaměřená na řízení projektů prostřednictvím opakujících se cyklů nazývaných **sprinty** (obvykle trvají 1–4 týdny). Na konci každého sprintu je doručena funkční část softwaru, která může být nasazena do produkce nebo dále iterována.

Scrum je formálně popsán v dokumentu Scrum Guide (SCHWABER a SUTHERLAND, 2020) a je jedním z nejrozšířenějších agilních přístupů v softwarovém inženýrství a produktovém řízení. Tento dokument byl poprvé vydán v roce 2010 jako oficiální definice scrumu. Současná verze je šestou revizí tohoto dokumentu.

Scrum definuje základní role, artefakty a události – ceremonie (OBRÁZEK 2). Pod artefakty si můžeme představit nástroje, které využívá ke svému fungování a ceremonie jsou schůzky, které provází celý proces jednotlivého sprintu.



Obrázek 2: Vizualizace pracovního procesu metodiky Scrum

Principy Scrumu:

- Krátké iterace (sprinty) s jasně definovanými úkoly.
- Pravidelné setkání týmu - Daily Scrum (OBRÁZEK 2).
- Průběžná zpětná vazba od zadavatele a zúčastněných stran – Sprint Review a Sprint Retrospective (OBRÁZEK 2).

Role ve Scrumu

Product Owner:

Product Owner je zodpovědný za maximalizaci hodnoty produktu, která vzniká prací Scrum Teamu. Jak toho dosáhne, závisí na konkrétní organizaci, týmu a individuálním přístupu. Klíčovou odpovědností Product Ownera je také efektivní správa Backlogu –

určuje cíle produktu, vytváří a jasně komunikuje požadavky, řadí je podle priority a zajišťuje jejich srozumitelnost a transparentnost.

I když může část těchto úkolů delegovat na jiné členy týmu, konečná odpovědnost za správu Backlogu zůstává vždy na něm. Pro úspěch Product Owenera je zásadní, aby celá organizace respektovala jeho rozhodnutí, která se odrážejí v obsahu a pořadí backlogu i v podobě výstupů prezentovaných na Sprint Review.

Product Owner je vždy jedna konkrétní osoba, nikoliv komise. Reprezentuje zájmy různých stakeholderů a rozhoduje o směru vývoje produktu. Ti, kdo chtějí změnit backlog, musí Product Owenera přesvědčit o vhodnosti svého návrhu. Jeho role je klíčová pro strategický rozvoj produktu a sladění priorit s potřebami zákazníků i organizace.

Scrum Master:

Scrum Master je klíčovou rolí v rámci Scrum týmu, která zajišťuje správné pochopení a zavedení Scrumu v týmu i celé organizaci. Jeho hlavním úkolem je pomáhat týmu se sebeřízením, podporovat jeho multifunkčnost a usnadňovat vytváření hodnotných přírůstků, které splňují definici hotového produktu (Definition of Done).

Zároveň odstraňuje překážky, které by mohly bránit postupu týmu, a zajišťuje, že všechny Scrum události probíhají efektivně, produktivně a v rámci stanoveného časového rámce. Podporuje také Product Owenera, například při definování cílů produktu, správě Backlogu a zajištění jasně formulovaných požadavků.

Na úrovni organizace se Scrum Master podílí na školení a koučování zaměstnanců, pomáhá s implementací Scrumu a podporuje empirický přístup k řízení složitých úkolů. Zároveň usnadňuje komunikaci mezi Scrum týmy a stakeholdery. Je skutečným lídrem, který pomáhá vytvářet efektivní prostředí pro agilní vývoj a neustálé zlepšování.

Development Team:

Vývojáři jsou členové Scrum Teamu, kteří se zavazují každý Sprint vytvářet funkční a použitelné části produktu. Jejich konkrétní dovednosti se liší podle oblasti, ve které pracují, ale jejich hlavní odpovědnosti zůstávají stejné.

Vývojáři jsou zodpovědní za plánování práce na Sprint, tedy tvorbu Sprint Backlogu, a za dodržování vysoké kvality výstupů podle Definition of Done. Každý den přizpůsobují svůj plán tak, aby se co nejlépe přiblížili splnění Sprint Goal. Zároveň mezi sebou udržují profesionální zodpovědnost a vzájemně se podporují v dosažení společných cílů.

Jejich role není jen o psaní kódu nebo technické realizaci – jsou klíčovými členy týmu, kteří aktivně přispívají k plánování, zlepšování procesu a celkové efektivitě Scrum Teamu.

Artefakty

Product Backlog

Product Backlog je neustále se vyvíjející a prioritizovaný seznam všeho, co je potřeba k vylepšení produktu. Slouží jako jediný zdroj úkolů pro tým.

Položky v Product Backlogu, které může tým dokončit v rámci jednoho Sprintu, jsou považovány za připravené k výběru během Sprintu. Tento stav obvykle dosáhnou po procesu refinementu, což je průběžná činnost, při které se položky rozdělují na menší a lépe definované části. Refinement zahrnuje doplňování detailů, jako jsou popisy, priority nebo náročnost úkolů, přičemž konkrétní atributy se mohou lišit podle povahy práce.

Za odhady náročnosti úloh odpovídají vývojáři, kteří budou na daných úkolech pracovat. Product Owner může jejich rozhodování ovlivnit tím, že jim pomůže pochopit různé možnosti a kompromisy.

Product Goal popisuje budoucí stav produktu, který slouží jako dlouhodobý cíl pro Scrum Team. Tento cíl je součástí Product Backlogu a všechny jeho položky společně definují co je potřeba k jeho dosažení.

Product je prostředek k dodávání hodnoty – má jasně vymezené hranice, známé stakeholdery a definované uživatele či zákazníky. Může se jednat o službu, fyzický produkt nebo něco abstraktního. Scrum Team se vždy soustředí na splnění (nebo případné opuštění) jednoho Product Goalu, než se pustí do dalšího.

Sprint Backlog

Sprint Backlog obsahuje Sprint Goal (cíl Sprintu), vybrané úkoly z Product Backlogu a konkrétní plán jejich realizace. Slouží jako aktuální přehled práce, kterou vývojáři plánují dokončit během Sprintu.

Sprint Backlog se průběžně aktualizuje, protože se při práci objevují nové informace. Měl by obsahovat dostatek detailů, aby si tým mohl každý den na Daily Scrumu ověřit pokrok a případně upravit plán.

Sprint Goal je hlavní cíl Sprintu, který dává týmu směr a pomáhá udržet zaměření. Definuje se během Sprintu a i když se může upravit rozsah úkolů ve Sprint Backlogu, samotný Sprint Goal zůstává neměnný.

Inkrement

Inkrement je dílčí krok k Product Goalu, který staví na předchozích výsledcích. Každý Inkrement musí být plně funkční a použitelný, aby přinášel hodnotu. Během Sprintu jich může vzniknout více a mohou být dodány kdykoliv, nejen na konci sprintu.

Definition of Done (DoD) určuje, kdy je práce dokončená a splňuje požadovanou kvalitu. Pokud položka Product Backlogu nespĺňuje DoD, nemůže být součástí Inkrementu.

Organizace může mít vlastní standardy DoD, které musí všechny Scrum Teamy dodržovat. Pokud ne, tým si vytvoří vlastní definici. Vývojáři se jí musí řídit a pokud na produktu pracuje více týmů, musí se na ní shodnout.

Ceremonie:

Sprint

Sprint je základní cyklus Scrumu, během kterého se nápady mění v inkrement. Má pevnou délku jeden týden až maximálně jeden měsíc, aby zajistil pravidelnost. Nový Sprint začíná ihned po skončení předchozího.

Všechny klíčové aktivity, jako Sprint Planning, Daily Scrum, Sprint Review a Sprint Retrospective, probíhají uvnitř Sprintu. Během Sprintu nesmí dojít ke změnám, které by ohrozily Sprint Goal, nesmí klesnout kvalita, ale Product Backlog může být průběžně upřesňován.

Skrz pravidelnou inspekci a adaptaci pomáhají Sprints udržet směr k Product Goal a minimalizovat rizika. Kratší Sprints umožňují rychlejší zpětnou vazbu a snižují náklady v případě změnových požadavků. Sprint může být zrušen pouze Product Ownerem, pokud se jeho cíl stane neaktuálním.

Sprint Planning

Sprint Planning zahajuje Sprint tým, že definuje práci, která se v něm má vykonat. Celý Scrum Team společně vytváří plán na základě priorit v Product Backlogu, které určuje Product Owner. Ten zajistí, aby všichni byli připraveni diskutovat o klíčových úkolech. Pokud je potřeba, mohou být přizváni další odborníci.

Během Sprint Planningu se řeší tři hlavní otázky:

1. Jakou hodnotu přináší tento Sprint? – Product Owner navrhuje, jak může produkt v tomto Sprintu získat větší hodnotu. Tým společně definuje Sprint Goal, který musí být určen ještě před koncem plánování.
2. Co se dá během Sprintu stihnout? – Vývojáři vybírají úkoly z Product Backlogu, upřesňují jejich detaily a odhadují, co reálně zvládnou. Pomáhá jim zkušenost s minulými Sprints a znalost Definition of Done.
3. Jak se práce provede? – Vývojáři rozdělují úkoly na menší kroky tak, aby bylo možné průběžně vytvářet hotové Inkrementy. Nikdo jim neříká, jak mají pracovat – to určují sami.

Výstupem Sprint Planningu je Sprint Backlog obsahující Sprint Goal, vybrané úkoly a plán jejich realizace. Sprint Planning trvá maximálně 8 hodin pro měsíční Sprint, u kratších Sprintů bývá kratší.

Daily Scrum

Daily Scrum slouží k rychlé kontrole postupu směrem ke Sprint Goalu a případnému přizpůsobení Sprint Backlogu. Vývojáři si tímto způsobem upravují plán práce na další den.

Jde o krátkou 15minutovou schůzku, která se koná každý pracovní den Sprintu, ideálně ve stejný čas a na stejném místě. Účastní se jí vývojáři, případně i Scrum Master a Product Owner, pokud aktivně pracují na úkolech ve Sprint Backlogu. Jakým způsobem setkání probíhá, si tým určuje sám – důležité je zaměřit se na pokrok a vytvořit jasný plán na další den.

Daily Scrum zlepšuje komunikaci, pomáhá rychle řešit překážky v další práci a usnadňuje v rámci týmu rozhodování. Není to ale jediný moment, kdy mohou vývojáři upravit svůj plán – často se domlouvají i během dne podle aktuálních potřeb.

Sprint Review

Sprint Review slouží k přezkoumání výsledků Sprintu a určení dalších kroků. Scrum Team prezentuje svou práci klíčovým stakeholderům a diskutuje pokrok směrem k Product Goalu.

Během této schůzky tým a stakeholdeři společně hodnotí, co se podařilo dokončit, jaké změny nastaly a co by mělo následovat. Na základě této diskuse se může upravit Product Backlog tak, aby lépe odrážel nové příležitosti a priority. Sprint Review by neměl být jen formální prezentací, ale aktivním pracovním setkáním.

Jde o předposlední událost Sprintu, která trvá maximálně čtyři hodiny u měsíčního Sprintu. U kratších Sprintů bývá i tento meeting kratší.

Sprint Retrospective

Retrospektiva slouží k plánování zlepšení kvality a efektivity týmové práce.

Scrum Team hodnotí, jak probíhal poslední Sprint z hlediska spolupráce, interakcí, procesů, nástrojů a jejich Definition of Done. Tým identifikuje předpoklady, které vedly k problémům, a analyzuje, co se během Sprintu podařilo a co ne. Diskutuje se o tom, co fungovalo, jaké problémy se vyskytly, a jak byly řešeny.

Tým vybere nejdůležitější změny pro zlepšení efektivity a začne je řešit co nejdříve. Některé změny mohou být přidány do Sprint Backlogu pro následující Sprint.

Sprint Retrospective je časově omezená na maximálně 3 hodiny u měsíčního Sprintu. U kratších Sprintů je obvykle kratší.

3.1.2.3 Kanban

Kanban je metodika zaměřená na vizualizaci práce a její plynulý tok. Hlavním nástrojem je Kanban tabule, na které jsou jednotlivé úkoly zobrazeny v různých fázích realizace (např. „k provedení“, „v procesu“, „hotovo“). Často je oblíbená u týmů, které se starají o provoz

IT systémů a jejich práci není možné plánovat dopředu, protože není jasné, jaké situace bude nutné v daném období řešit.

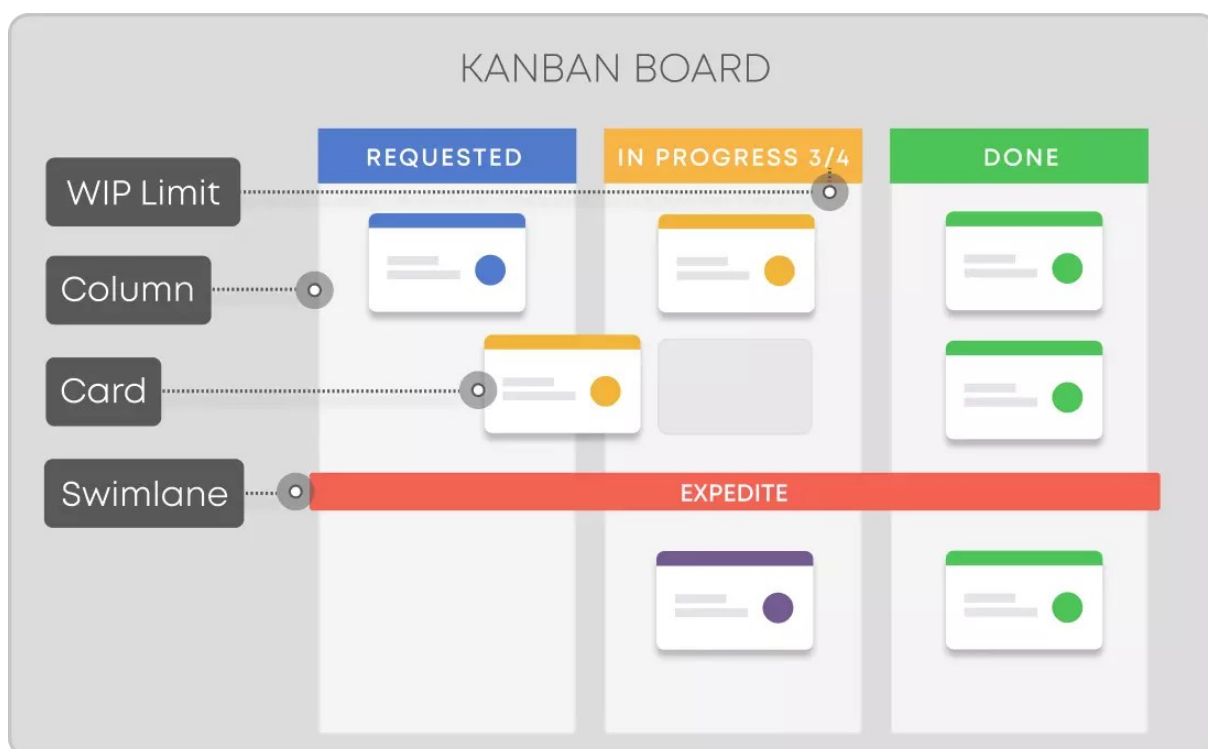
Kanban byl původně vyvinut společností Toyota v 50. letech pro řízení výroby a později adaptován pro software DAVIDEM J. ANDERSONEM v knize Kanban: Successful Evolutionary Change for Your Technology Business (2010). Kanban, na rozdíl od Scrumu, nemá jednu oficiální příručku. Mimo výše citovanou publikaci je často doporučována The Kanban Guide od KANBAN UNIVERSITY. Tento průvodce popisuje základní principy a hodnoty Kanbanu a jak tento systém implementovat v rámci organizačních procesů. KANBAN UNIVERSITY je uznávanou autoritou v oblasti Kanban metodologie, která je často zaměňována za jiné agilní přístupy.

Principy Kanbanu zahrnují několik klíčových prvků, které pomáhají týmu efektivně řídit a optimalizovat pracovní procesy. Prvním z nich je průběžná vizualizace práce a pracovních postupů. Pomocí tabule (Kanban board) a karet (tasks) se úkoly a jejich stav zobrazují v reálném čase. Tento vizuální přístup poskytuje týmu okamžitý přehled o všech úkolech, což umožňuje snadněji identifikovat problémy v procesu a efektivně řídit tok práce. Vizualizace práce také pomáhá členům týmu rychle zjistit, kde je potřeba věnovat více úsilí a kde je proces zpomalený.

Dalším klíčovým principem je omezení množství úkolů v jednotlivých fázích procesu vývoje, známé jako WIP limit (Work in Progress limit). Tento princip znamená stanovení maximálního počtu úkolů, které mohou být současně v jednotlivých fázích pracovního procesu. WIP limit pomáhá zabránit přetížení týmu, zajišťuje plynulý tok práce a zamezuje tomu, aby tým začal nové úkoly, aniž by dokončil stávající. Když je dosažen limit pro danou fázi, nový úkol nemůže být zahájen, dokud neprobíhá nějaká dokončená práce.

Kanban také klade důraz na kontinuální zlepšování pracovního procesu. Týmy pravidelně analyzují svůj proces, hledají problémy a hledají způsoby, jak jej optimalizovat. Pomáhá to týmu přizpůsobit se měnícím se podmínkám a zajišťuje dlouhodobou udržitelnost fungování týmů v proměnlivém prostředí. Celý systém vizualizace jednotlivých fází podněcuje tým k vizuální kontrole pracovního procesu a snazší představě, kde vznikají zásadní problémy. To týmu umožňuje neustále přehodnocovat a optimalizovat pracovní proces. Kanban nedefinuje konkrétní schůzky tak jako Scrum, kde by docházelo k revizi, ale předpokládá, že si členové týmu ve chvíli, kdy takovou příležitost identifikují sami řeknou a proces upraví. Tento přístup neustálého zlepšování umožňuje týmům reagovat na nové výzvy a přizpůsobovat procesy tak, aby byly efektivnější a produktivnější.

Na rozdíl od metodiky Scrum, která používá pevně stanovené sprinty, je Kanban flexibilní. Týmy mohou upravit svůj pracovní proces bez potřeby zásadních změn nebo definování konkrétních časových rámců. Tento flexibilní přístup je vhodný pro týmy, které chtějí optimalizovat pracovní tok postupně, bez nutnosti zásadních změn. Kanban je ideální pro prostředí, kde je důležitá schopnost rychlé reakce na nové požadavky a efektivní řízení toku práce bez přísně stanovených termínů.



Obrázek 3: Vizualizace kanbanové tabule

3.1.2.4 Extrémní programování (XP)

První projekt, využívající Extreme Programming (XP), byl zahájen 6. března 1996. Od té doby se XP stalo jednou z nejpoblárnějších agilních metodik a prokázalo svou úspěšnost v různých firmách a odvětvích po celém světě. Jeho hlavním cílem je zajistit spokojenost zákazníka prostřednictvím pružného a efektivního vývoje softwaru. Namísto dlouhého vývoje s nejistým výsledkem XP umožňuje dodávat funkční software průběžně a přizpůsobovat ho aktuálním potřebám.

Klíčovým prvkem XP je týmová spolupráce mezi vývojáři, zákazníky a manažery, kteří fungují jako rovnocenní partneři. Metodika vytváří jednoduché a efektivní pracovní prostředí, které podporuje samoorganizaci týmu a vysokou produktivitu. Mezi základní principy patří komunikace, jednoduchost, zpětná vazba, respekt a odvaha.

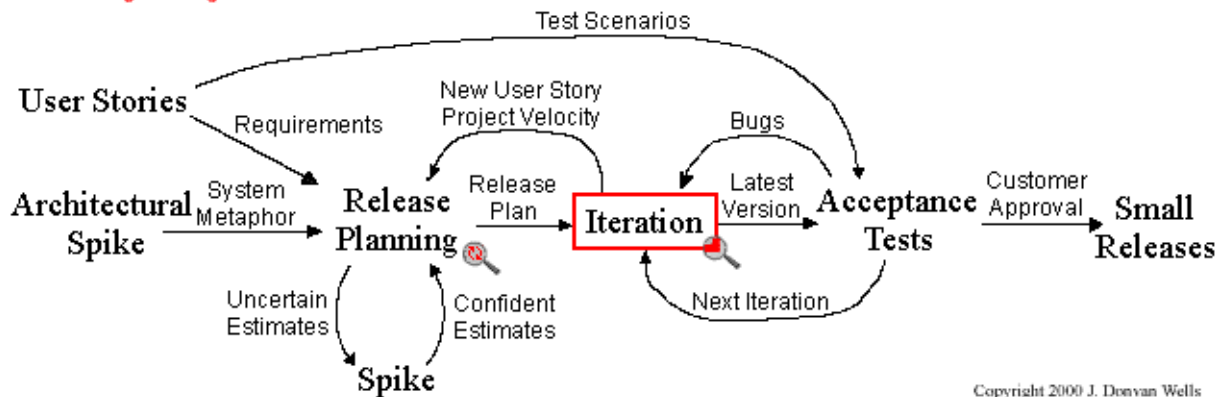
XP klade důraz na neustálou komunikaci, udržování co nejjednoduššího designu a testování softwaru od prvního dne. Software je dodáván postupně, což umožňuje rychlou reakci na změny a vylepšování produktu podle zpětné vazby. Tento přístup zvyšuje kvalitu softwaru i efektivitu práce týmu.

Metodika XP je založena na jednoduchých, vzájemně propojených pravidlech, která společně tvoří efektivní proces vývoje (WELLS 1999). Podporuje aktivní zapojení zákazníků, umožňuje vývojářům přispívat bez ohledu na zkušenosti a minimalizuje zbytečné aktivity, čímž snižuje náklady i frustraci všech zúčastněných.

XP bylo formálně definováno na webu „www.extremeprogramming.org“ (WELLS 1999) a v knize "Extreme Programming Explained" (BECK, 2000).



Extreme Programming Project



Copyright 2000 J. Donovan Wells

Obrázek 4: Způsob práce pomocí metodiky Extrémní programování

Pravidla XP (WELLS 1999)

Plánování

- Uživatelské příběhy (user story) – uživatelské příběhy jsou srozumitelně popsány.
- Plán vydání (release plan) – plánování vydání stanovuje harmonogram vydávání nových verzí.
- Časté vydání – software je vydáván často v malých verzích.
- Iterativní vývoj – projekt je rozdělen do iterací.
- Plánování iterace – každá iterace začíná plánovací schůzkou.

Řízení

- Optimalizace pracovního prostoru – tým má k dispozici prostor pro setkávání a spolupráci.
- Udržitelný pracovní rytmus – pracovní tempo v týmu je dlouhodobě udržitelné.
- Denní setkání – každý den začíná krátkým statusovým meetingem.
- Měření rychlosti vývoje – měří se, jak rychle vznikají nové funkcionality (Project Velocity).
- Rotace členů týmu – lidé se přesouvají mezi úkoly a rolemi v týmu pro co nejsnazší sdílení znalostí a zkušeností.
- Oprava XP – pokud XP přestane fungovat, je třeba ho opravit.

Návrh

- Jednoduchost – návrh systému by měl být co nejjednodušší.
- Metafora systému – tým si volí sdílenou metaforu, pro jejíž popis se používají známé koncepty a analogie a slouží pro co nejsnazší pochopení systému.
- CRC karty – při návrhu se používají CRC karty (Class-Responsibility-Collaborator).

- Řešení typu „spike“ – experimentální řešení s co nejjednodušším kódem pro vyřešení daného problému.
- Žádná funkcionalita předem – funkce se přidávají jen tehdy, když jsou potřeba.
- Refaktoring – průběžně se zlepšuje kvalita kódu.

Programování

- Zákazník v dosahu – zákazník či zadavatel je neustále dostupný pro konzultaci a řešení problémů.
- Standardy kódu – kód se píše podle dohodnutých standardů.
- Vývoj řízený testy (TDD) – nejprve se píší testy, poté teprve probíhá implementace funkcionalit.
- Programování ve dvojici – veškerý produkční kód se píše ve dvojici.
- Sekvenční integrace – změny v kódu integruje vždy jen jedna dvojice programátorů.
- Průběžná integrace – změny v kódu se často integrují do sdíleného repozitáře.
- Vyhrazený integrační počítač – pro integraci kódu je k dispozici zvláštní počítač.
- Sdílené vlastnictví kódu – každý může upravovat jakoukoli část kódu.

Testování

- Unit testy – každý kus kódu musí mít vlastní testy.
- Úspěšné testy před vydáním – kód lze vydat pouze tehdy, pokud projde všemi testy.
- Testování chyb – pokud se objeví nová chyba, nejprve se vytvoří testy pro její odhalení.
- Akceptační testy – akceptační testy se spouštějí často a jejich výsledky jsou dostupné jak týmu, tak zákazníkům či zadavatelům.

3.1.2.5 Test-driven Development (TDD)

Test-Driven Development (TDD) je agilní vývojová metoda, která se občas považuje za součást Extrémního programování. V rámci této metodiky se nejprve píší testy a až poté samotný kód. Tento přístup zajišťuje, že software je od počátku navržen s ohledem na testovatelnost, což snižuje chybovost a zvyšuje kvalitu kódu.

TDD se skládá ze tří hlavních kroků:

1. **Napsání testu.** Vývojář nejprve napíše test, který definuje očekávané chování funkce. Tento test zpočátku selže, protože ještě neexistuje žádná implementace.
2. **Implementace kódu.** Následně se napíše minimální množství kódu potřebné k úspěšnému průchodu testem.
3. **Refaktorizace.** Kód se upraví tak, aby byl čistší, efektivnější a snadno udržovatelný, přičemž se neustále ověřuje, že testy zůstávají úspěšné.

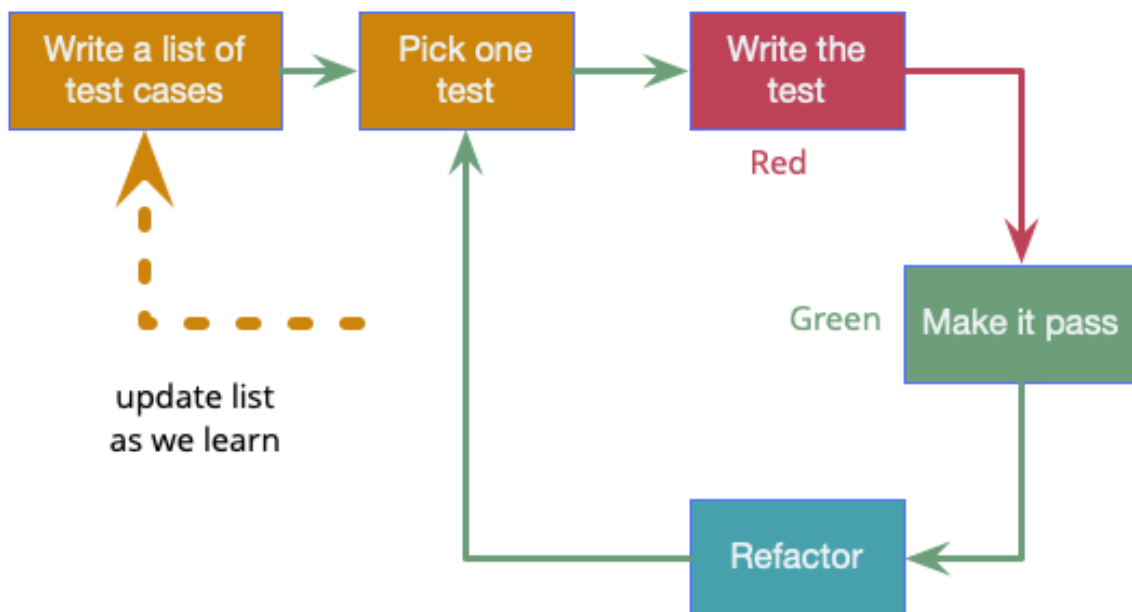
TDD bylo popsáno v knize "Test-Driven Development: By Example" (BECK, 2003). Tento přístup je často využíván v rámci Extreme Programming (XP) a je základem mnoha moderních CI/CD (Continuous Integration / Continuous Deployment) procesů.

Podstatou TDD je, že vývoj začíná psaním testu, který nejprve selže, poté se implementuje minimální množství kódu nutného k jeho splnění a následně se tento kód optimalizuje – tento cyklus je známý jako Red-Green-Refactor.

Mezi hlavní techniky TDD patří jednotkové testy (unit tests), které automaticky ověřují funkčnost malých, izolovaných částí kódu. TDD se často propojuje s nástroji pro nepřetržitou integraci (CI), které umožňují pravidelné spouštění testů a nasazování změn v krátkých vývojových cyklech. Díky tomu je možné rychle odhalit chyby a zajistit vysokou stabilitu aplikace v průběhu celého vývoje.

Úspěšná implementace TDD vyžaduje jasně definované role v týmu. Vývojáři nesou primární odpovědnost za psaní testů ještě před samotnou implementací funkce a jejich následnou údržbu. QA inženýři (Quality Assurance) mohou pomoci s návrhem testovacích scénářů a ověřením správnosti chování systému. Dále jsou důležití DevOps specialisté, kteří nastavují testovací prostředí a připravují automatické nasazení kódu, zajišťují automatizaci a integraci testů do CI/CD pipeline.

TDD nachází své uplatnění zejména u projektů, kde je kladen důraz na vysokou kvalitu kódu, stabilitu a bezpečnost. Typickými příklady jsou finanční, zdravotnické a další kritické systémy, kde i malá chyba může mít závažné důsledky. Naopak TDD nemusí být vhodné pro rychlé prototypování nebo projekty s velmi nejasnými a proměnlivými požadavky. V těchto případech může být psaní testů zbytečně náročné a omezující. Stejně tak může být problematické v menších týmech s omezenými zdroji, protože TDD často zvyšuje počáteční náročnost vývoje.



Obrázek 5: Diagram jednotlivých kroků Test-driven Development

3.1.2.6 *Lean Software Development*

Lean Software Development (LSD) je agilní metodika vycházející z principů štlíhlé výroby (Lean Manufacturing), konkrétně z filozofie Toyota Production System. Přestože byly tyto principy původně určeny pro optimalizaci výroby automobilů, jsou překvapivě dobře použitelné i v prostředí vývoje softwaru (POPPENDIECK a POPPENDIECK, 2003). Lean klade důraz především na rychlé dodávání hodnoty zákazníkovi, odstraňování plýtvání a neustálé zlepšování.

Základní ideou Lean přístupu je, že vývoj softwaru není výroba v tradičním slova smyslu – je to spíš proces objevování. V neustále se měnícím a často nejistém prostředí softwarových projektů není možné vše naplánovat dopředu. Lean proto podporuje postupné učení, testování hypotéz a rychlou adaptaci. Vývoj se děje v krátkých, iterativních cyklech (obvykle 1 až 3 týdny), které umožňují časté dodávání funkčního softwaru, získání zpětné vazby a její rychlé zpracování.

Lean se soustředí na eliminaci všeho, co nepřináší přímou hodnotu zákazníkovi – ať už jsou to zbytečné schůzky, nadměrná dokumentace, nevyužité funkce nebo příliš složitá rozhodovací struktura. Tato orientace na hodnotu zákazníka zároveň přirozeně snižuje rizika. Namísto budování rozsáhlých systémů bez ověření, že o ně někdo skutečně stojí, Lean doporučuje „dodávat co nejrychleji“ – tedy vystavit produkt realitě co nejdřív, v jednoduché podobě, a na základě reálné zpětné vazby rozhodovat o dalším směru vývoje.

Zásadní roli hraje také rozhodování v pravý čas – tedy až ve chvíli, kdy máme dostatek informací. V Lean přístupu nejde o to vše naplánovat předem, ale o schopnost odkládat rozhodnutí do poslední chvíle, kdy jsou rizika menší a důsledky rozhodnutí jasnější. Tento přístup dává prostor změnám a zvyšuje odolnost vůči nejistotě. Lean přiznává, že změna zadání není selhání, ale přirozený důsledek poznání během vývoje. Důležité je být připraven se změnám přizpůsobit, místo aby byly vnímány jako narušení plánu.

Na rozdíl od některých tradičních přístupů Lean neklade důraz na formální dokumentaci nebo přesně definované role, ale na spolupráci a sdílené vlastnictví. Posiluje týmovou autonomii a důvěru ve schopnosti jednotlivců – ti, kteří pracují na řešení, mají zároveň kompetenci rozhodovat. Klíčem k efektivitě je jednoduchá a transparentní komunikace, průběžné testování, automatizace a viditelnost aktuálního stavu projektu pro všechny členy týmu.

Jedním z důsledků tohoto přístupu je i kratší délka vývojových cyklů. Místo toho, aby tým pracoval měsíce na jedné velké verzi, preferuje se časté dodávání menších, ale hodnotných inkrementů. Tento způsob nejenže zrychluje zpětnou vazbu a snižuje náklady na případné opravy, ale také umožňuje rychle reagovat na změny v prostředí nebo požadavcích zákazníka. Takové cykly bývají v praxi řádově týdenní až čtrnáctidenní, ale konkrétní délka závisí na povaze týmu a produktu.

Mezi silné stránky Lean patří schopnost efektivně reagovat na změny v zadání, minimalizace plýtvání zdroji a důraz na týmovou spolupráci. Lean vytváří prostředí, kde je možné přizpůsobovat se okolnostem, zlepšovat produkt na základě reálné zpětné vazby a budovat software, který má skutečnou hodnotu. Umožňuje flexibilní plánování, adaptaci a důraz na kvalitu bez zbytečné byrokracie.

Na druhou stranu slabší stránkou může být to, že úspěšná aplikace Lean vyžaduje vysokou míru disciplíny, důvěry a zralosti v týmu. Není to návod „krok za krokem“, ale spíše sada principů, které je nutné přizpůsobit konkrétnímu kontextu. Pokud tým postrádá schopnost sebereflexe, komunikace nebo ochoty měnit zaběhnuté zvyky, může zavedení Lean narazit. Také ve velmi regulovaných prostředích (např. ve zdravotnictví nebo bankovníctví) může být složitější aplikovat Lean v plném rozsahu kvůli požadavkům na dokumentaci a kontrolu.

Lean Software Development tedy není jen metodika, ale spíš způsob myšlení – orientovaný na hodnotu, respekt k lidem, adaptabilitu a neustálé zlepšování. Je ideální tam, kde je vysoká nejistota, kde zákazník neví přesně, co chce, nebo kde se požadavky rychle mění. Funguje nejlépe v kombinaci s dalšími agilními přístupy – například s Kanbanem, který usnadňuje řízení toku práce, nebo s DevOps, který podporuje automatizaci a hladké nasazení výsledků.

7 principů Lean Software Development definovaných v knize Lean Software Development: An Agile Toolkit" (POPPENDIECK a POPPENDIECK, 2003):

1. **Eliminace plýtvání.** Za plýtvání se považuje vše, co nepřináší zákazníkovi přímou hodnotu. Ať už jsou to nevyužité funkce, přebytná dokumentace, zbytečné předávání práce mezi týmy nebo dlouhé čekání na rozhodnutí – vše, co zpomaluje dodání požadovaného výsledku, je plýtvání a mělo by být odstraněno.
2. **Podpora učení.** Vývoj je proces objevování – podobný vytváření nového receptu. Nelze čekat, že se vše povede na první pokus. Proto je důležité vytvářet prototypy, testovat varianty a z výsledků se učit. Iterativní přístup a rychlé zpětné vazby umožňují neustálé zlepšování.
3. **Rozhodování co nejpozději.** V nejistém prostředí je výhodné odkládat klíčová rozhodnutí na později, kdy už máme více informací. Místo spekulací se rozhodujeme na základě faktů. Udržování otevřených možností a schopnosti reagovat na změny je efektivní strategií v komplexním vývoji.
4. **Dodávání co nejrychleji.** Rychlost je klíčem k získání zpětné vazby, zrychlení učení a vyšší spokojenosti zákazníka. Kratší vývojové cykly umožňují rychlejší reakce na změny požadavků a zajišťují, že software odpovídá aktuálním potřebám, ne těm z minulosti.
5. **Posilování týmu.** Nejlepší rozhodnutí dělají lidé, kteří práci skutečně vykonávají. Lean podporuje týmy, aby byly samostatné, měly přístup ke všem potřebným informacím a společně rozhodovaly. Práce je řízena podle skutečných potřeb

(např. pomocí vizualizace procesu, denních schůzek, testů a průběžné integrace kódu).

6. **Budování integrity.** Skvělý software není jen funkční, ale i logicky uspořádaný, přehledný, snadno se používá i rozšiřuje. Takový systém působí jako „přesně to, co jsem chtěl“. Toho lze dosáhnout díky jasné architektuře, dobré komunikaci a vedení – nikoli jen procesními pravidly.
7. **Vnímejte celek.** Je důležité sledovat celý systém a neoptimalizovat jen jednotlivé části. Když se každý zaměřuje pouze na svůj úsek (např. databázi, UI), vzniká riziko neefektivního celku. Úspěšný vývoj vyžaduje spolupráci napříč disciplínami a společný cíl – funkční celek, ne perfektní jednotlivosti.

Tato metodika se často využívá v kombinaci s **DevOps, Continuous Delivery a Kanbanem**.

3.1.3 Výhody a nevýhody jednotlivých metodik

Tabulka shrnutí silných a slabých stránek jednotlivých metodik

Tabulka 1: Shrnutí silných a slabých stránek jednotlivých metodik

Metodika	Silné stránky	slabé stránky
Waterfall	jasně definovaný plán a postup; vhodné pro projekty s pevnými požadavky	nízká flexibilita vůči změnám; dlouhá doba dodání
Scrum	flexibilita; rychlé dodání; zaměření na týmovou spolupráci	potřeba dobře zvoleného Product Ownera; nevhodné pro velké týmy
Kanban	flexibilita; průběžné dodání; snadné přizpůsobení, vizualizace toku práce	nedostatečně strukturované pro velké a složité projekty
XP	velmi vysoká kvalita kódu; silná týmová spolupráce; rychlé dodání	nevhodné pro velké týmy; vyžaduje vysoké technické dovednosti
TDD	velmi vysoká kvalita kódu; přehledný kód; snadné testování	pomalejší start vývoje; vysoké nároky na kvalitu již při psaní testů
Lean	eliminace plýtvání; rychlé dodání hodnoty; zaměření na zákazníka	vyžaduje důkladné porozumění procesu; může být obtížné implementovat bez zkušeností

Je důležité poznamenat, že agilní metodiky se často částečně překrývají a mohou být kombinovány tak, aby co nejlépe vyhovovaly specifikům konkrétního projektu. Tento přístup, nazývaný hybridní metodiky, vede k tvorbě názvů jako SCRUMBAN nebo LEAN XP, které naznačují fúzi dvou a více agilních přístupů. Takové kombinace se ukázaly jako efektivní v mnoha případech, kdy je třeba využít výhod více metodik zároveň, čímž se maximalizuje flexibilita, kvalita a efektivita vývoje.

Výběr metodiky by tedy měl být pečlivě zváženo s ohledem na velikost týmu, složitost projektu, požadavky na rychlost dodání a kvalitu, přičemž je nutné být připraven na adaptaci a případné přizpůsobení metodiky v průběhu vývoje.

3.2 Srovnávací kritéria

Software se stal nepostradatelnou součástí téměř každého odvětví, od financí přes zdravotnictví až po zábavu. Správně zvolená metodika vývoje zásadně ovlivňuje efektivitu práce týmu, kvalitu výsledného produktu a jeho schopnost přizpůsobit se měnícím se požadavkům. Neexistuje univerzálně nejlepší metodika – každá má své výhody i nevýhody a je vhodná pro jiný typ projektu. Malý startup s nejasnou vizí bude potřebovat jiný přístup než korporace s rozsáhlým systémem, kde je důraz kladen na stabilitu a bezpečnost. Správná volba metodiky tedy umožňuje optimalizovat vývojový proces, minimalizovat rizika a efektivně alokovat zdroje.

Na základě zkušeností z praxe se jedním z nejčastějších problémů softwarových projektů jeví financování a s ním spojené tlakové řízení nákladů. Projekty často začínají s omezeným rozpočtem a očekáváním investorů, že produkt bude co nejdříve použitelný a přinese návratnost. Dalším problémem jsou změnové požadavky – během vývoje se často ukazuje, že původní zadání nebylo dostatečně specifikované nebo se potřeby uživatelů změnilly. Vznikají tak dodatečné úpravy, které mohou projekt významně zdržet a případně i prodražit. Časové požadavky a termíny dodání jsou dalším klíčovým faktorem. Vývojový tým je často pod tlakem splnit pevně stanovené milníky, což může vést k přetížení a snížení kvality výsledného produktu. Mezi další problémy patří například nejasnosti v zadání, kdy vývojáři dostávají nedostatečně definované požadavky, což vede k neefektivnímu vývoji a častým změnám.

Na základě těchto problémů se při porovnání metodik vývoje jeví jako klíčová následující témata:

1. **Rychlost vývoje** – jak efektivně metodika umožňuje dodat funkční produkt v co nejkratším čase.
2. **Řízení rizik** – jak metodika pomáhá předcházet nečekaným problémům, například chyby či nejasnosti v požadavcích, zpoždění nebo překročení rozpočtu.
3. **Reakce na změny zadání** – jak pružně dokáže metodika reagovat na nové požadavky a přizpůsobit vývoj aktuálním potřebám.

Na základě těchto faktorů byla zvolena čtyři kritéria pro srovnání metodik vývoje softwaru:

- **Flexibilita a reakce na změny zadání** – klíčové pro projekty, kde se požadavky často mění.
- **Řízení rizik** – je klíčové pro minimalizaci nečekaných problémů a zajištění stability projektu. Stabilita projektu znamená schopnost udržet projekt na správné cestě k dosažení stanovených cílů, termínů a rozpočtu i v případě vzniku problémů. Zahrnuje efektivní řízení změn, kontrolu nad rozsahem projektu, udržení kvality

výstupů, zachování pozitivní dynamiky týmu a minimalizaci neplánovaných nákladů. Důsledné řízení rizik zajišťuje, že projekt i přes výzvy pokračuje podle plánu a dosahuje požadovaných výsledků.

- **Délka vývojového cyklu** – rozhodující pro projekty, kde je kladen důraz na rychlé dodání první verze produktu.
- **Velikost projektu** – některé metodiky jsou vhodnější pro malé projekty s týmy o pár jednotlivců, jiné pro velké projekty s cenou několik milionů korun a dopadem na statisíce zákazníků.

Tato kritéria umožní objektivně posoudit silné a slabé stránky jednotlivých metodik a vybrat tu nejvhodnější pro daný projekt.

3.3 Srovnání metodik dle stanovených kritérií

3.3.1 Hodnocení metodik

Waterfall

1. **Flexibilita a reakce na změny.** Vodopádový model je známý svou nízkou flexibilitou, protože vývoj probíhá lineárně v pevně stanovených fázích. Jakmile je jedna fáze dokončena, je obtížné se k ní vrátet a provádět změny. To znamená, že pokud se požadavky na software změní v průběhu vývoje, jejich implementace je časově i finančně náročná. Změny v pozdějších fázích mohou vyžadovat zásadní přepracování analýzy, návrhu i implementace, což prodlužuje čas dokončení projektu a zvyšuje náklady. (MOKHTAR a KHAYYAT, 2022)
2. **Řízení rizik.** Waterfall metodika se snaží řídit rizika především důkladnou analýzou a plánováním na začátku projektu. Předpokládá, že dobře definované požadavky minimalizují pravděpodobnost vzniku problémů v dalších fázích vývoje. Tento přístup však neumožňuje efektivně reagovat na neočekávané problémy, které se mohou objevit až v implementační nebo testovací fázi. Pokud dojde ke kritické chybě v návrhu, její oprava bývá komplikovaná a zdlouhavá. (MOKHTAR a KHAYYAT, 2022)
3. **Délka vývojového cyklu.** Waterfall metodika má obvykle dlouhý vývojový cyklus, protože všechny fáze (analýza, návrh, implementace, testování) jsou pevně stanovené a probíhají sekvenčně. To znamená, že první použitelná verze softwaru je dostupná až na konci projektu. U rozsáhlých projektů může tento model vést k tomu, že software nebude odpovídat aktuálním potřebám uživatelů v době jeho dokončení. (MOKHTAR a KHAYYAT, 2022)
4. **Velikost projektu.** Waterfall se hodí především pro velké projekty s pevně danými požadavky, kde se neočekávají změny během vývoje. Typicky se používá v případech, kde je potřeba důkladná dokumentace a schvalovací procesy. Naopak pro dynamické projekty, kde se požadavky často mění, je tento přístup nevhodný. (MOKHTAR a KHAYYAT, 2022)

Scrum

1. **Flexibilita.** Scrum poskytuje vysokou flexibilitu, protože vývoj probíhá v krátkých sprintech. Na začátku každého sprintu může tým upravit backlog a přizpůsobit se novým požadavkům. To znamená, že pokud se v průběhu projektu změní potřeby zákazníků, změny lze implementovat poměrně rychle a efektivně. (SCHWABER a SUTHERLAND, 2011)
2. **Řízení rizik.** Scrum průběžně monitoruje rizika díky denním stand-up meetingům, kde se identifikují překážky v práci týmu. Díky pravidelným retrospektivám se tým učí z předchozích sprintů a optimalizuje svůj přístup. Tato metoda minimalizuje riziko, že se v pozdějších fázích objeví zásadní problémy, protože software je testován a iterativně vylepšován. (SCHWABER a SUTHERLAND, 2011)
3. **Délka cyklu.** Sprints umožňují pravidelné dodávání funkčních verzí softwaru, což je zásadní pro rychlý vývoj. Krátké iterace umožňují týmu průběžně získávat zpětnou vazbu od zákazníků a přizpůsobovat se jejich požadavkům. Délka sprintu by měla být měsíc či méně. (SCHWABER a SUTHERLAND, 2011)
4. **Velikost projektu.** Scrum je ideální pro menší a střední projekty, kde se očekávají časté změny požadavků. Používá se především v startupech, produktových firmách a dynamických odvětvích, jako je vývoj webových aplikací nebo mobilních služeb. (SCHWABER a SUTHERLAND, 2011)

Kanban

1. **Flexibilita.** Kanban poskytuje velmi vysokou flexibilitu, protože úkoly nejsou rozděleny do pevných iterací (jako ve Scrumu), ale jsou neustále aktualizovány na Kanban tabuli. To umožňuje týmům okamžitě reagovat na změny priorit, aniž by museli čekat na konec sprintu. Hlavním nástrojem je vizualizace pracovního toku, která pomáhá týmu i stakeholderům vidět aktuální stav projektu v reálném čase. (VACANTI a COLEMAN, 2020)
2. **Řízení rizik.** Kanban pomáhá řídit rizika tím, že omezuje počet úkolů ve stavu „rozpracováno“ (tzv. Work in Progress – WIP limit). Tento přístup minimalizuje zahlcení týmu, což snižuje pravděpodobnost vzniku chyb a zlepšuje efektivitu práce. Díky kontinuálnímu monitorování pracovního toku mohou manažeři včas identifikovat úzká hrdla a problémová místa. Nicméně chybí zde jasně definované retrospektivy, což může zpomalit dlouhodobé zlepšování procesů. (VACANTI a COLEMAN, 2020)
3. **Délka cyklu.** Kanban nepracuje s pevně stanovenými iteracemi, což znamená, že vývoj probíhá kontinuálně. Jakmile je úkol dokončen, může být ihned nasazen do produkce. To umožňuje rychlé doručování změn, ale zároveň může vést k neorganizovanému přidávání nových úkolů, pokud tým nedodrží správné procesy řízení backlogu. (VACANTI a COLEMAN, 2020)

4. **Velikost projektu.** Kanban se nejlépe hodí pro týmy s neustálým tokem práce, jako jsou operativní týmy, DevOps, IT podpora nebo zákaznické služby, kde se požadavky neustále mění. U něho tedy nelze mluvit úplně o projektu. Je méně vhodný pro dlouhodobé produktové týmy, které potřebují jasně definované milníky a strukturu, například při vývoji rozsáhlých softwarových řešení. (VACANTI a COLEMAN, 2020)

XP

1. **Flexibilita.** XP poskytuje extrémní flexibilitu díky krátkým týdenním iteracím, které umožňují velmi rychlou zpětnou vazbu od zákazníků i mezi členy týmu. Hlavním rysem je pravidelné přezkoumávání kódu, párové programování a neustálé testování, což umožňuje rychlé změny v požadavcích bez narušení celkového vývoje. (WELLS, 1999)
2. **Řízení rizik.** XP snižuje rizika hlavně díky automatizovanému testování, neustálé integraci a refaktorizaci kódu. Používá techniky jako Test-Driven Development (TDD), párové programování a nepřetržitou integraci, což pomáhá identifikovat problémy dříve, než se stanou kritickými. Přesto může být XP náročný na čas i zdroje, protože vyžaduje silnou disciplínu týmu a zkušené vývojáře, kteří jsou schopni pracovat v rychlém tempu. (WELLS, 1999)
3. **Délka cyklu.** XP pracuje s extrémně krátkými iteracemi, typicky týdenními, což znamená neustálé dodávání malých, ale funkčních částí softwaru. Tento přístup umožňuje neustálé zlepšování a adaptaci kódu, ale zároveň může způsobit, že tým bude zahlcen neustálými změnami a ztratí celkový přehled nad architekturou projektu. (WELLS, 1999)
4. **Velikost projektu.** XP je ideální pro malé, vysoce výkonné týmy, které potřebují rychle iterovat a mají vysoké nároky na kvalitu kódu. Používá se především v prostředích, kde je nutné rychlé dodání softwaru bez kompromisů v kvalitě, například ve startupových firmách nebo při vývoji kritických aplikací s důrazem na bezpečnost a stabilitu. Naopak není vhodný pro velké týmy, kde je obtížné udržet úzkou spolupráci mezi všemi členy. (WELLS, 1999)

TDD

1. **Flexibilita.** TDD umožňuje vysokou flexibilitu, protože každá část kódu je pokryta testy, což umožňuje rychlou úpravu a refaktorizaci bez obav z rozbití stávající funkcionality. Pokud je potřeba změnit požadavky na software, stačí upravit testy a přizpůsobit kód tak, aby jim odpovídal. Nicméně psaní testů před kódem může zpomalit prvotní vývoj, což znamená, že metodika nemusí být vhodná pro projekty s extrémním tlakem na rychlost dodání. (BECK, 2003)
2. **Řízení rizik.** Hlavní výhodou TDD je snížení rizika chyb, protože každá změna v kódu je automaticky testována, což minimalizuje pravděpodobnost vzniku

regresních chyb. Díky důrazu na čistý a testovatelný kód je TDD ideální pro projekty, kde je klíčová stabilita a dlouhodobá udržitelnost. (BECK, 2003)

3. **Délka cyklu.** TDD pracuje s krátkými iteracemi, kdy se nejprve píše test, poté se implementuje kód a následně se provádí refaktorizace. Tento cyklus se neustále opakuje, což zajišťuje průběžnou kontrolu kvality, ale zároveň zvyšuje časovou náročnost vývoje ve srovnání s metodami, kde se testy píšou až po dokončení kódu. (BECK, 2003)
4. **Velikost projektu.** TDD je nejvhodnější pro projekty s vysokými nároky na kvalitu, bezpečnost a dlouhodobou udržitelnost. Často se používá v bankovníctví, zdravotnictví nebo jiných kritických systémech, kde je klíčové minimalizovat chyby. Naopak není vhodný pro rychlé prototypování, kde je třeba rychle iterovat bez ohledu na kvalitu kódu. (BECK, 2003)

Lean

1. **Flexibilita.** Lean přístup je vysoce flexibilní, protože se zaměřuje na neustálé zlepšování procesů a eliminaci neefektivních činností. Hlavním principem je rychlé dodání hodnoty zákazníkovi, což znamená, že týmy neustále analyzují a optimalizují vývojový proces, aby minimalizovaly plýtvání (např. nadbytečná dokumentace, nepotřebné funkce). Lean klade důraz na rychlé rozhodování a přizpůsobování se aktuálním podmínkám, což z něj činí vysoce adaptabilní metodiku, vhodnou pro prostředí s neustále se měnícími požadavky. (POPPENDIECK a POPPENDIECK, 2003)
2. **Řízení rizik.** Lean přístup minimalizuje rizika tím, že prioritizuje nejdůležitější funkcionality a nasazuje je v co nejkratším čase. Namísto čekání na dokončení celého produktu Lean dodává malé inkrementální změny, čímž snižuje pravděpodobnost selhání projektu. Díky principu Build-Measure-Learn (Postav – Změř – Pouč se) se neustále testuje, zda má daná funkcionality skutečně přidanou hodnotu. Pokud se zjistí, že některý požadavek není potřebný, je eliminován, což minimalizuje riziko zbytečných investic do nepotřebných funkcí. (POPPENDIECK a POPPENDIECK, 2003)
3. **Délka cyklu.** Lean Software Development klade důraz na rychlé iterace a plynulý vývojový cyklus, což znamená, že nové funkce jsou dodávány průběžně. Díky eliminaci zbytečných procesů jsou týmy schopné rychle reagovat na zpětnou vazbu zákazníků a neustále vylepšovat software. Lean se často propojuje s DevOps a Continuous Deployment, což umožňuje pravidelné nasazování aktualizací do produkce bez velkých výpadků. (POPPENDIECK a POPPENDIECK, 2003)
4. **Velikost projektu.** Lean je ideální pro týmy a organizace zaměřené na maximalizaci efektivity a dodání co největší hodnoty zákazníkům. Je široce využíván ve startupovém prostředí, kde je potřeba rychle validovat nápady a minimalizovat náklady. Lean se také hodí pro velké společnosti, které chtějí

optimalizovat své vývojové procesy a zkrátit dobu mezi vývojem a nasazením. Naopak nemusí být vhodný pro projekty s pevně danými regulacemi, kde jsou striktní požadavky na dokumentaci a kontrolu změn, jako je například vývoj softwaru pro zdravotnictví nebo bankovníctví. (POPPENDIECK a POPPENDIECK, 2003)

3.3.2 Srovnání jednotlivých kritérií a metodik

V předchozích podkapitolách byla jednotlivá kritéria podrobně analyzována pro každou metodiku zvlášť. Nicméně, jak ukazuje tabulka 2, faktory jako flexibilita, řízení rizik, délka cyklu a vhodnost pro různé typy projektů jsou obtížně měřitelné a jejich vzájemné porovnání může být poměrně subjektivní.

Tabulka poskytuje přehledné shrnutí těchto kritérií pro každou metodiku, což usnadňuje jejich vzájemné porovnání a může výrazně ovlivnit rozhodnutí o volbě metodiky.

Tabulka 2: Porovnání metodik dle zvolených kritérií

Kritérium	Waterfall	Scrum	Kanban	XP	TDD	Lean
Flexibilita	Nízká	Vysoká	Vysoká	Velmi vysoká	Střední	Střední
Řízení rizik	Pevné plánování	Poučení se z předchozích sprintů	WIP - nedostatečné	Časté testování	Testy na začátku	Eliminace rizik dříve než vzniknou
Délka cyklu	Dlouhá – v řádech měsíců	Krátké sprinty – maximálně jeden měsíc	Kontinuální	Krátké iterace – 1 až 3 týdny	Viz XP	Variabilní
Velikost projektu	Pevné požadavky	Dynamické projekty	DevOps, IT podpora nebo zákaznické služby	Malé týmy	Kvalitní kód	Efektivní procesy

Flexibilita je jedním z klíčových faktorů, který ovlivňuje výběr metodiky. Zatímco metodiky jako Scrum a Kanban se vyznačují vysokou vysokou flexibilitou a je tudíž možné je přizpůsobit aktuálním potřebám v rámci daného projektu, waterfall je rigidní a méně přizpůsobivý změnám během projektu. Pokud je projekt velmi dynamický, s možností častých změn požadavků nebo pokud je potřeba rychle reagovat na změny v prostředí, metodiky jako Scrum nebo Kanban budou pravděpodobně vhodnější. Naopak, pro projekty s pevně danými požadavky, kde změny nejsou časté, může být waterfall efektivnější, protože jasně definuje kroky a časový rámeček.

V oblasti řízení rizik se jednotlivé metodiky liší v přístupu k identifikaci a mitigaci rizik. Waterfall spoléhá na pevné plánování a předem definované kroky, což znamená, že rizika jsou obvykle identifikována na začátku projektu a následně se na ně pohlíží z perspektivy

predikce. Scrum naopak vychází z pravidelných sprintů, což umožňuje týmu reagovat na rizika průběžně a učit se z předchozích sprintů. XP a TDD kladou důraz na testování a kvalitní kódování již od začátku, což minimalizuje technická rizika. Pokud je projekt vystaven vysokému riziku změn nebo technologickým problémům, metodiky jako Scrum, XP nebo TDD budou schopny lépe přizpůsobit průběh vývoje aktuálním potřebám.

Délka cyklu a časový rámeček pro dokončení projektu jsou dalšími důležitými faktory. Waterfall je vhodný pro dlouhé projekty, které vyžadují podrobný plán na několik měsíců dopředu. Naopak metodiky jako Scrum a XP se zaměřují na kratší iterace a rychlejší dodávky, což je výhodné v projektech, kde je potřeba pravidelně vyhodnocovat pokrok a případně měnit směr. Kanban zajišťuje kontinuální práci bez pevně daných cyklů, což je ideální pro projekty jako IT podpora nebo DevOps, kde je potřeba průběžně reagovat na požadavky. Výběr metodiky tedy závisí na časovém rámci projektu a jeho potřebě flexibilně reagovat na vývoj.

V oblasti vhodnosti pro různé typy projektů ukazuje tabulka 2, jak každá metodika vyhovuje jiným typům projektů. Waterfall je ideální pro projekty s pevnými požadavky a jasně definovaným cílem, například v oblasti stavebnictví nebo výrobních procesů. Scrum a Kanban jsou více zaměřeny na dynamické projekty, kde se očekává častá změna požadavků, což může zahrnovat vývoj softwaru nebo jiné inovativní projekty. Lean a TDD se hodí pro projekty, které se zaměřují na maximální efektivitu procesů a vysokou kvalitu kódu.

Tabulka 2 tedy slouží jako nástroj pro rychlé porovnání metodik, což usnadňuje výběr metodiky na základě specifických požadavků projektu.

3.4 Vhodnost použití jednotlivých metodik v různých typech projektů

Výběr správné metodiky vývoje softwaru je klíčový pro úspěch projektu. Každý projekt má jiné požadavky – některé vyžadují přísnou kontrolu a přesné plánování, jiné zase maximální flexibilitu a rychlou zpětnou vazbu. Při rozhodování záleží na mnoha faktorech: velikosti týmu, očekávané době dodání, finančních možnostech, i na tom, jak přesně jsou definované požadavky už na začátku.

Pokud projekt má jasně dané požadavky, fixní rozpočet a přesně stanovený termín dodání – například rozsáhlý systém pro státní správu nebo firemní ERP – pak je na místě zvážit waterfall. Tato metodika spoléhá na podrobnou dokumentaci, přesné plánování a postupné dokončování jednotlivých fází bez nutnosti větších změn. Nehodí se ale pro situace, kde se požadavky teprve rodí nebo očekáváte, že se budou často měnit.

Pro dynamické projekty, jako jsou například nové mobilní aplikace nebo startupové weby, je daleko vhodnější Scrum. Umožňuje rychle reagovat na změny a pravidelně doručovat funkční části produktu. Týmy pracují v krátkých sprintech, každý sprint končí zpětnou vazbou a plánováním dalšího postupu. Scrum je ideální, pokud nemáte ještě všechno rozmyšlené a chcete vývoj přizpůsobovat podle uživatelských ohlasů.

Kanban je skvělý pro týmy, které pracují kontinuálně na menších úkolech – typicky jde o IT podporu, provozní týmy nebo DevOps. Jeho síla spočívá v jednoduchosti: práce se přehledně zobrazuje na nástěnce, tým si sám hlídá, kolik úkolů má rozpracovaných. Neplánuje se dopředu tolik jako ve Scrumu, ale umožňuje hladký tok práce bez větších skoků.

Extreme Programming (XP) se hodí pro malé, soudržné týmy, které kladou důraz na kvalitu kódu a velmi častou komunikaci. Silně podporuje testování, párové programování a rychlé iterace. Pokud vyvíjíte technicky náročný produkt a zakládáte si na tom, že bude napsaný „čistě a správně“, XP vám může vyhovovat.

Metodika Test-Driven Development (TDD) není samostatným přístupem k řízení projektu, ale může se stát důležitou součástí jiných metodik. Je založena na tom, že nejdříve napíšete testy a až potom samotný kód. Pomáhá snižovat chybovost a vede ke kvalitnějšímu výsledku, ale vyžaduje disciplínu a zkušenosti. Hodí se zejména tam, kde je na prvním místě spolehlivost a udržitelnost řešení.

Lean development klade důraz na efektivitu a minimalizaci plýtvání. Snaží se co nejdříve zjistit, co zákazník skutečně potřebuje, a rychle reagovat na zpětnou vazbu. Pokud pracujete na produktu, kde si nejste jistí, co přesně má umět – typicky MVP aplikace nebo interní nástroje – Lean vám může pomoci rychle ověřit směr a upravovat vývoj podle skutečných potřeb.

Výběr metodiky tedy závisí hlavně na tom, jaký problém řešíte a v jakém prostředí. Čím více nejistoty a změn vás čeká, tím spíš potřebujete agilní přístup. Naopak čím víc se vše dá naplánovat dopředu, tím spíš obstojí klasický přístup jako waterfall. Tabulka 2 vám může napovědět, která cesta bude pro váš projekt ta nejvhodnější.

4 Praktická část – manuál pro volbu metodiky

Volba správné metodiky vývoje softwaru představuje jeden z nejdůležitějších kroků při zahájení softwarového projektu. Nesprávně zvolený přístup může vést k problémům v plánování, komunikaci i samotném dodání výsledného produktu. Tento manuál vychází jak z praxe, tak z poznatků uvedených v předchozích kapitolách této práce a nabízí systematický návod, jak k výběru metodiky přistupovat.

V praxi se velmi často stává, že volba metodiky probíhá na základě zvyklostí nebo předchozích zkušeností členů týmu. Takový přístup však nebere v potaz specifika konkrétního projektu – jako jsou velikost a struktura týmu, dynamika změn požadavků, míra nejistoty, požadavky na dokumentaci, cílové prostředí nebo termín doručení. Tento manuál si klade za cíl nabídnout strukturovaný přístup, který těmto aspektům věnuje odpovídající pozornost.

Při rozhodování o vhodné metodice doporučuji zvážit následující aspekty. Ty neodpovídají přímo srovnávacím kritériím definovaným v teoretické části této práce, kde slouží čistě pro vzájemné porovnání metodik. Zatímco zde už je nutné projekt analyzovat detailněji, a proto je kritérií více:

- **Proměnlivost požadavků:** stabilní vs. proměnlivé zadání.
- **Velikost a zkušenosti týmu:** malý začínající tým vs. zkušený agilní tým.
- **Požadavky na dokumentaci a audit:** interní předpisy, certifikace, právní požadavky.
- **Požadovaná rychlost dodání:** potřeba co nejrychlejšího MVP vs. dlouhodobý vývoj.
- **Termín doručení:** pevně daný (např. smluvně závazný) termín vs. flexibilní časový rámec.
- **Očekávaná úroveň kvality a testovatelnosti:** důraz na stabilitu, bezpečnost, bezchybovost.
- **Typ zákazníka nebo prostředí:** korporát, státní správa, startup, interní produkt apod.

4.1 Doporučený postup výběru metodiky

Na základě výše uvedených kritérií doporučujeme následující postup:

1. **Definujte základní charakteristiky projektu.** V první řadě je důležité pochopit, jaký druh softwaru se má vyvíjet. Může jít o mobilní aplikaci, informační systém pro firmu, veřejně dostupný web nebo třeba prototyp nové technologie. Zohledněte také odvětví (např. zdravotnictví, finance, školství), velikost projektu a kdo bude výsledný software používat.

2. **Zhodnoťte míru proměnlivosti požadavků.** Položte si otázku, zda zadavatel přesně ví, co chce – a zda se tento požadavek pravděpodobně nebude měnit. Pokud jsou požadavky nejasné nebo se často mění podle zpětné vazby, je vhodné volit agilní metodiky, které se s tímto typem nejistoty umí vypořádat.
3. **Zvažte složení a velikost týmu.** Popište, kolik lidí bude na vývoji pracovat, jaké mají zkušenosti s vývojem softwaru a zda již dříve pracovali v agilních týmech. Velké týmy mohou mít potíže se samoorganizací, naopak malé týmy mohou postrádat některé důležité role (např. tester, Scrum Master). Tato úvaha vám pomůže zhodnotit, které metodiky je reálné zavést.
4. **Analyzujte technické požadavky a kvalitu.** Zamyslete se, jak důležitá je spolehlivost a bezchybnost softwaru. U některých aplikací může chyba znamenat jen nepohodlí, u jiných může mít závažné důsledky (např. ve zdravotnictví). Pokud je třeba zajistit vysokou kvalitu a stabilitu, uvažujte o metodikách zaměřených na testování, jako je TDD (vývoj řízený testy).
5. **Zhodnoťte časové požadavky a termín doručení.** Zjistěte, zda má projekt pevně stanovený termín dokončení – například kvůli smluvnímu závazku, marketingové kampani nebo legislativnímu termínu. Pokud je termín fixní a nelze s ním hýbat, je potřeba volit metodiky, které umožní pečlivé plánování a řízení rozsahu – například waterfallový model nebo hybridní přístupy waterfall – agile (KIRPITSAS a PACHIDIS, 2022). Naopak agilní metodiky umožňují iterativní dodávky a jsou vhodné tam, kde je prostor pro průběžné ladění a doručování po částech.
6. **Posuďte dokumentační a auditní požadavky.** Pokud je projekt realizován ve firmě nebo instituci, která podléhá pravidlům auditu (např. banky, státní správa), je důležité, aby dokumentace vývoje byla úplná, přehledná a schvalovaná v jednotlivých fázích. V takovém případě může být vhodnější tradiční waterfallový přístup nebo jeho kombinace s agilními prvky.
7. **Vyberte vhodnou metodiku / kombinaci.** Na základě odpovědí na výše uvedené otázky si vytvořte profil projektu a srovnajte ho s charakteristikami jednotlivých metodik (pomůže vám k tomu následující tabulka). Není nutné volit pouze jednu metodiku – často se osvědčuje jejich kombinace, například použití strukturovaného plánování (Waterfall) na začátku a následný agilní vývoj (Scrum nebo Kanban) v dalších fázích.

Tabulka 3: Přehled kritérií pro výběr vhodné metodiky. Jednotlivá metodika může být z hlediska daného kritéria vhodná (Ano), vhodná s určitými omezeními nebo úpravami (Podmíněně) nebo zcela nevhodná (Ne).

Kritérium / Metodika	Waterfall	Scrum	Kanban	XP	TDD	Lean
Stabilní požadavky	Ano	Podmíněně	Podmíněně	Ne	Podmíněně	Ne
Časté změny požadavků	Ne	Ano	Ano	Ano	Ano	Ano
Malý tým	Ano	Ano	Ano	Podmíněně	Podmíněně	Ano
Velký tým	Ano	Podmíněně	Ne	Ne	Ne	Podmíněně
Důraz na dokumentaci	Ano	Podmíněně	Ne	Ne	Ne	Ne
Potřeba kvality	Podmíněně	Ano	Podmíněně	Ano	Ano	Ano
Rychlé dodání MVP	Ne	Ano	Ano	Ano	Podmíněně	Ano
Flexibilita	Ne	Ano	Ano	Ano	Podmíněně	Ano
Pevně stanovený termín	Ano	Podmíněně	Ne	Ne	Podmíněně	Ne

4.2 Shrnutí

Základním východiskem pro volbu metodiky je analýza povahy projektu. U projektů s jasně definovanými požadavky může být výhodné zvolit tradiční vodopádový model. Naopak tam, kde se očekává častá změna požadavků, jsou vhodné agilní přístupy jako Scrum či Kanban. Důležitým faktorem je rovněž velikost a zkušenost týmu, potřeba dokumentace, časový tlak a rychlost dodání výsledku.

V praxi bývá nejefektivnější kombinace více přístupů. Například strukturované plánování z modelu Waterfall a vývojové sprinty podle Scrumu umožňují skloubit kontrolu i flexibilitu. Tento manuál může sloužit jako vodítko, jak volit a přizpůsobovat metodiky podle konkrétních podmínek projektu. Základním principem však zůstává adaptabilita – tedy ochota metodiku upravit a neustále zlepšovat.

5 Případová studie – Volba metodiky na reálném projektu

5.1 Zadání projektu

Startupová firma vyvíjí interní mobilní aplikaci pro pojišťovací společnost. Aplikace je určena pro prodejce pojištění na pobočkách a umožňuje jim efektivně pracovat s klienty.

Funkce aplikace:

1. Správa klientů – možnost vyhledat existujícího klienta nebo založit nového.
2. Výběr pojištění – doporučení vhodného pojištění na základě informací o klientovi.
3. Příprava smlouvy – vedení uživatele celým procesem vyplňování smlouvy.
4. Podpis smlouvy – umožnění digitálního podepsání dokumentu.

Aplikace bude vyvíjena jako mobilní aplikace, která využívá existující backend a API volání. Během vývoje však dojde ke dvěma zásadním změnám, na které bude nutné reagovat:

1. Sloučení s jinou společností, což vyžaduje kompletní redesign grafického rozhraní. To bylo částečně očekáváno již při plánování projektu, a tudíž je možné očekávat i další problémy.
2. Migrace backendu do cloudu, což znamená nutnost změnit všechna API volání a přizpůsobit autentizaci aplikace.

5.2 Volba metodiky

Pro výběr metodiky pro tento projekt jsem postupoval následovně:

1. Definice základních charakteristik projektu

Typ softwaru: interní mobilní aplikace pro pojišťovací společnost.

Cíl: aplikace pro prodejce pojištění na pobočkách, která umožňuje efektivní práci s klienty.

Technické požadavky: aplikace využívá stávající backend a API, s plánovanými změnami v infrastruktuře (migrování do cloudu a redesign grafiky).

2. Zhodnocení míry proměnlivosti požadavků

Projekt se vyznačuje proměnlivými požadavky, zejména v souvislosti s redesignem aplikace a migrací backendu. Tyto změny nejsou zcela předvídatelné a mohou se měnit na základě zpětné vazby a nových potřeb během vývoje. Proto je důležité zvolit metodiku, která umožňuje flexibilitu při změnách.

3. Zvažování složení a velikosti týmu

Projekt bude pravděpodobně realizován v malém týmu. To znamená, že agilní přístupy, které fungují dobře v menších, více samostatných týmech, mohou být

efektivní. Případně lze použít některé role, jako je Scrum Master, pokud se tým rozroste.

4. Analyzování technických požadavků a kvality

Aplikace musí mít vysokou úroveň kvality, zejména s ohledem na digitální podpisy a citlivé údaje. Vysoký důraz na stabilitu a bezchybnost znamená, že metoda zaměřená na testování, jako je TDD, může být vhodná, ale není nezbytná, pokud budou agilní testy integrovány.

5. Zhodnocení časových požadavků a termínu doručení

Projekt nemá pevně daný termín pro dodání celkové aplikace, ale je potřeba vyvinout rychlý MVP. Agilní metodiky, jako Scrum nebo Kanban, jsou vhodné pro iterativní vývoj a rychlé dodávky verzí aplikace.

6. Posouzení dokumentačních a auditních požadavků

Přestože aplikace bude interní, může být vyžadována nějaká forma dokumentace pro auditní účely. Tato potřeba může být dobře pokrytá kombinací agilního přístupu s některými dokumentačními prvky (např. u Scrumu).

7. Výběr metodiky

Na základě analýzy výše uvedených faktorů jsem vytvořil profil projektu, který jsem porovnal s Tabulkou 3 v tabulce 4 a dospěl k následujícímu závěru:

Agilní metodiky (zejména Scrum) budou ideální pro tento projekt. Scrum umožňuje flexibilitu při změnách požadavků a zároveň podporuje rychlé iterace a dodávky aplikace.

Kanban by mohl být vhodný pro zajištění plynulého toku práce, zejména pokud jde o drobné změny a optimalizace.

Vzhledem k potřebě stabilního a kvalitního kódu by bylo vhodné využít testování řízené vývojem (TDD), což umožní udržet vysokou kvalitu aplikace při rychlém vývoji.

Kombinace Scrum pro agilní přístup a TDD pro zajištění kvality vývoje je ideální volbou pro tento projekt. Zvolené metodiky jsou zvýrazněné v Tabulce 4.

Tabulka 4: Ukázka použití manuálu pro výběr metodiky. Ve druhém sloupci tabulky je příkladový projekt, který následně hodnotíme v rámci jednotlivých kritérií z pohledu porovnávaných metodik.

Kritérium / Metodika	Profil projektu	Waterfall	Scrum	Kanban	XP	TDD	Lean
Stabilní požadavky	Podmíněně (proměnlivé požadavky)	Ano	Podmíněně	Podmíněně	Ne	Podmíněně	Ne
Časté změny požadavků	Ano	Ne	Ano	Ano	Ano	Ano	Ano
Malý tým	Ano	Ano	Ano	Ano	Podmíněně	Podmíněně	Ano
Velký tým	Podmíněně (omezené pro velký tým)	Ano	Podmíněně	Ne	Ne	Ne	Podmíněně
Důraz na dokumentaci	Podmíněně (dokumentace je flexibilní)	Ano	Podmíněně	Ne	Ne	Ne	Ne
Potřeba kvality	Ano	Podmíněně	Ano	Podmíněně	Ano	Ano	Ano
Rychlé dodání MVP	Ano	Ne	Ano	Ano	Ano	Podmíněně	Ano
Flexibilita	Ano	Ne	Ano	Ano	Ano	Podmíněně	Ano
Pevně stanovený termín	Podmíněně (možnost flexibilního termínu)	Ano	Podmíněně	Ne	Ne	Podmíněně	Ne

6 Závěr

Tato práce analyzovala a porovnávala dvě základní skupiny metodiky softwarového vývoje – tradiční vodopádový model (waterfall) a agilní přístupy (Scrum, Kanban, XP, TDD, Lean). Zatímco waterfall poskytuje pevnou strukturu a jasně definované fáze vývoje, agilní metodiky vynikají flexibilitou, možností rychlé adaptace na změny v požadavcích a důrazem na iterativní vývoj.

Každý z těchto přístupů má své silné a slabé stránky. Waterfall je vhodný pro projekty s pevně stanovenými požadavky, kde je kladen důraz na důkladnou dokumentaci a plánování. Naopak agilní metodiky se uplatňují v dynamických prostředích, kde se požadavky často mění a je klíčové rychle reagovat na změny a zpětnou vazbu od zákazníků. V praxi se často ukazuje, že neexistuje univerzálně nejlepší metodika – její výběr závisí na povaze projektu, zkušenostech týmu a specifických požadavcích organizace.

Významným přínosem této práce je návrh systematického manuálu pro výběr metodiky, který umožňuje IT týmům lépe rozhodovat na základě konkrétních kritérií. Díky případové studii bylo možné prakticky demonstrovat aplikaci tohoto manuálu a ukázat, jak může pomoci při volbě optimálního přístupu k vývoji softwaru.

Celkově lze konstatovat, že správný výběr metodiky má zásadní vliv na úspěch projektu. Je důležité nejen dobře porozumět jednotlivým metodikám, ale také umět vyhodnotit jejich vhodnost v konkrétním kontextu. Tato práce tak nabízí užitečný nástroj pro IT týmy a organizace, které chtějí zefektivnit proces vývoje softwaru a zvýšit pravděpodobnost úspěšného dokončení projektu.

Seznam použitých informačních zdrojů

1. *A Brief History of Software Development Methodologies* [online], 2021. 7.7.2021 [cit. 2025-04-11]. Dostupné z: <https://www.growin.com/blog/history-of-software-development-methodologies/>
2. ANDERSON, David J. *Kanban: successful evolutionary change for your technology business*. Sequim, Wash.: Blue hole press, 2010. ISBN 978-0-9845214-0-1.
3. BECK, Kent. *Extreme programming explained: embrace change*. The XP series. Boston: Addison-Wesley, [2000]. ISBN 0-201-61641-6.
4. BECK, Kent. *Test-driven development: by example*. The Addison-Wesley signature series. Boston: Addison-Wesley, [2003]. ISBN 0-321-14653-0.
5. BOEHM, Barry, 2006. A view of 20th and 21st century software engineering. In: *Proceedings of the 28th international conference on Software engineering* [online]. New York, NY, USA: ACM, 2006-05-28, s. 12-29 [cit. 2025-04-11]. ISBN 1595933751. Dostupné z: doi:10.1145/1134285.1134288
6. DYADE, Antosh. *History of Software Development*. DevOps Antosh [online]. 12. června 2024 [cit. 2025-04-07]. Dostupné z: <https://devops.antosh.in/2024/06/history-of-software-development.html>
7. HIGHSMITH, Jim. *History: The Agile Manifesto*. Online. In: *Manifesto for Agile Software Development*. Agile Alliance, 2001. Dostupné z: <https://agilemanifesto.org/history.html>. [cit. 2025-04-11]
8. KANBAN UNIVERSITY. *The Kanban Guide*. Dostupné z: <https://kanban.university/> [cit. 2025-04-11].
9. KIRPITSAS, Ioannis K. a Theodore P. PACHIDIS, 2022. Evolution towards Hybrid Software Development Methods and Information Systems Audit Challenges. *Software* [online]. 1(3), 316-363 [cit. 2025-04-11]. ISSN 2674-113X. Dostupné z: doi:10.3390/software1030015
10. MCCORMICK, Mike, 2012. *Waterfall vs. Agile Methodology*. Revised Edition 8/9/2012. MPCS. Dostupné z: http://mccormickpcs.com/images/Waterfall_vs_Agile_Methodology.pdf
11. MOKHTAR, Renad a Mashaal KHAYYAT. A Comparative Case Study of Waterfall and Agile Management. *SAR Journal - Science and Research* [online]. 52-62 [cit. 2025-04-11]. ISSN 2619-9963. Dostupné z: doi:10.18421/SAR51-07
12. POPPENDIECK, Mary a Thomas David POPPENDIECK, 2003. *Lean software development: an agile toolkit*. Boston: Addison-Wesley. The agile software development series. ISBN 0-321-15078-3.
13. PRESSMAN, Roger S., 2001. *Software engineering: a practitioner's approach*. 6th ed. Boston: McGraw-Hill. McGraw-Hill series in computer science. ISBN 0-07-301933-X.
14. ROYCE, Winston W. *Managing the development of large software systems: concepts and techniques*. In: *Proceedings of the 9th international conference on Software Engineering*. 1987, s. 328-338.

15. SCHWABER, Ken a Jeff SUTHERLAND. *The Scrum Guide*. [online]. 2020 [cit. 2025-04-11]. Dostupné z: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>
16. SERGEEV, Alexander. *Team Roles in Waterfall Methodology*. Hygger [online]. 2016. [cit. 2025-04-11] Dostupné z: <https://hygger.io/blog/team-roles-in-waterfall-methodology/>
17. VACANTI, Daniel a COLEMAN, John. *Kanban Guide*. Kanban Guides, 2020. [cit. 2024-04-12]. Dostupné z: <https://kanbanguides.org/english/>
18. WELLS, Don. *The Rules of Extreme Programming*. *Extreme Programming: A Gentle Introduction* [online]. 1999, October 8, 2013 [cit. 2025-04-07]. Dostupné z: <http://www.extremeprogramming.org/rules.html>

Zdroje obrázků:

Obrázek 1: <https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model> [cit. 2025-04-07]

Obrázek 2: <https://www.scrum.org/resources/what-scrum-module> [cit. 2025-04-07]

Obrázek 3: <https://businessmap.io/kanban-resources/getting-started/what-is-kanban-board> [cit. 2025-04-07]

Obrázek 4: <http://www.extremeprogramming.org/map/project.html> [cit. 2025-04-07]

Obrázek 5: <https://martinfowler.com/bliki/TestDrivenDevelopment.html> [cit. 2025-04-07]

Vyjádření k využití nástrojů umělé inteligence

Při tvorbě této bakalářské práce sloužily nástroje umělé inteligence pouze jako doplňková pomůcka. Konkrétně byly využity k překladu abstraktu a klíčových slov do angličtiny, ke korektuře českého jazyka a kontrole citační normy. Veškerý odborný obsah, provedená analýzy, manuál pro volbu metodiky i závěry jsou výsledkem mé vlastní samostatné práce.

Seznam příloh

Příloha 1 – Slovník IT pojmů

Slovník IT pojmů

Agilní metodiky – Přístupy k vývoji software, které kladou důraz na flexibilitu, rychlé iterace a přizpůsobivost změnám. Mezi nejznámější patří Scrum, Kanban a Extreme Programming (XP).

API (Application Programming Interface) – Rozhraní pro komunikaci mezi různými softwarovými komponentami nebo aplikacemi.

Audit – Nástroj pro kontrolu fungování společnosti, může být jak vnitřní v rámci společnosti, nebo vnější. Může být vyžadován spolupracujícími stranami či zákonem.

Backlog – Seznam úkolů a funkcionalit, které je třeba implementovat. Používá se v agilních metodikách, zejména ve Scrumu.

Backend – Část softwarové aplikace, která běží na serveru a zajišťuje logiku aplikace, zpracování dat a komunikaci s databází.

Cloud computing – Způsob poskytování výpočetních zdrojů a služeb přes internet. Umožňuje škálovatelnost a flexibilitu.

Continuous Delivery (CD) – Vývojová praktika, která automatizuje proces nasazení softwaru, umožňuje průběžné doručování změn do produkčního prostředí a minimalizuje manuální zásahy.

Continuous Integration (CI) – Praktika vývoje, kdy jsou změny kódu průběžně integrovány do hlavní větve projektu a automaticky testovány.

Délka vývojového cyklu – Časový úsek mezi zahájením a dokončením vývoje softwaru. Může se lišit podle použité metodiky.

DevOps – Soubor praktik, který propojuje vývoj (development) a provoz (operations) s cílem zefektivnit a automatizovat proces nasazování softwaru.

Endpoint – Adresa, na kterou aplikace posílá požadavky při komunikaci s API.

ERP systém – ERP systém (Enterprise Resource Planning) je software, který integruje a automatizuje klíčové obchodní procesy organizace, jako je účetnictví, řízení zásob a výroby, čímž zlepšuje efektivitu a usnadňuje správu informací.

Implementace – Proces zavádění a realizace softwarového řešení nebo systému v praxi. Zahrnuje vývoj, konfiguraci, testování a nasazení do provozu.

Iterace – Opakující se vývojový cyklus, ve kterém se postupně vylepšuje software. Používá se v agilních metodikách.

Increment – Pojem z metodiky Scrum definující nově vyvinuté funkcionality v rámci jedné iterace.

Kanban – Agilní metoda vizualizace pracovního postupu, která pomáhá optimalizovat průběh práce.

Kritéria hodnocení metodiky – Parametry, podle kterých lze porovnávat vývojové metodiky, například flexibilita, řízení rizik nebo délka vývojového cyklu.

Lean Software Development – Agilní přístup k vývoji softwaru, který se zaměřuje na eliminaci plýtvání a maximální efektivitu.

Metodika vývoje software – Soubor postupů a pravidel pro organizaci práce při vývoji softwaru. Mezi nejznámější patří waterfall, Scrum, Kanban, XP nebo TDD.

Migrace backendu – Proces přesunu serverové infrastruktury z jednoho prostředí do jiného, například do cloudu.

Product Owner – Osoba ve Scrumu odpovědná za backlog a definování priorit vývoje.

Projektový management – Systém plánování, organizace a řízení projektu.

Refaktorizace kódu – Proces úpravy a optimalizace zdrojového kódu bez změny jeho vnější funkčnosti. Cílem je zlepšit čitelnost, udržitelnost a efektivitu kódu, odstranit duplicitní části a zjednodušit jeho strukturu. Refaktorizace pomáhá snižovat technický dluh a usnadňuje budoucí vývoj a rozšiřování softwaru. Často se provádí v kombinaci s automatizovaným testováním, aby bylo zajištěno, že změny neovlivní správné fungování aplikace.

Retrospektiva – Setkání týmu na konci sprintu, kde se hodnotí, co fungovalo dobře a co je třeba zlepšit.

Roadmapa – Časový plán vývoje či jednotlivých fází vizualizovaný jako diagram.

Řízení rizik – Proces identifikace, hodnocení a minimalizace rizik během vývoje softwaru.

Scrum – Populární agilní metodika založená na iterativním vývoji, sprintu a pravidelných schůzkách.

Sprint – Krátký vývojový cyklus (typicky 1–4 týdny) ve Scrumu, po kterém by měl být dodán funkční software.

Stand-up meeting – Krátká denní schůzka ve Scrumu, kde členové týmu hlásí postup a překážky.

TDD (Test-Driven Development) – Metodika vývoje, při které se nejprve píšou testy a až poté samotný kód.

Technický dluh – Stav ve vývoji softwaru, kdy jsou při implementaci funkcionalit zvolena rychlá, ale neoptimální řešení, která později vyžadují dodatečné úpravy a opravy.

Testování – Proces ověřování, zda software funguje správně a splňuje požadavky.

Vodopádový model (waterfall) – Tradiční lineární přístup k vývoji softwaru, kde se jednotlivé fáze dokončují postupně a nelze se k nim snadno vrátit.

Změnové požadavky – Nové požadavky, které se objevují během vývoje a je třeba je implementovat.