

Střední průmyslová škola Třebíč

Maturitní práce

Nasazení a hodnocení nástrojů pro bezpečnostní analýzu zdrojového kódu

Profilová část maturitní zkoušky

Studijní obor: Informační technologie

Třída: ITA4

Školní rok: 2025/2026 Simona Havelková

Zadání práce

Cílem maturitní práce je nasadit a ověřit sadu nástrojů pro statickou analýzu za účelem hledání zranitelností ve zdrojovém kódu, tzv. SAST (Static Application Security Testing). Vybrané SAST nástroje budou ověřeny nad zranitelným kódem webové aplikace vyvinuté v programovacím jazyce JavaScript. Výstupem práce bude analýza a hodnocení SAST nástrojů dle předem určených metrik (přesnost, pokrytí, míra falešně pozitivních nálezů atd.). Práce se zaměří také na úspěšnost SAST v identifikaci zranitelností určitého typu (např. injekční útoky), na základě čehož uvede doporučení pro jejich konfiguraci za účelem redukce falešně pozitivních hlášení.

ABSTRAKT

Účelem mé maturitní práce je prozkoumat a vyhodnotit nástroje pro statickou analýzu zdrojového kódu podle definovaných hodnotících ukazatelů (přesnost, pokrytí, míra falešně pozitivních nálezů atd.) nad zranitelným kódem webové aplikace naprogramované v jazyce JavaScript. Teoretická část obsahuje přehled vývoje software od vysvětlení pojmu životního cyklu vývoje software, rozebrání problematiky softwarových zranitelností až po bezpečnostní analýzu zdrojového kódu. V praktické části je proveden výběr technologií, které se následně přichystaly pro jejich použití. Poté začalo testování vybraných SAST nástrojů. Získané výsledky jsou následně vyhodnoceny. Závěrečná část práce obsahuje shrnutí získaných výsledků analýz. Práce tak poskytuje komplexní přehled možností a omezení vybraných SAST nástrojů a jejich přínosů ke zvýšení bezpečnosti softwaru.

KLÍČOVÁ SLOVA

Statická analýza; bezpečný vývoj; softwarové zranitelnosti; SAST nástroje

ABSTRACT

The aim of my graduation thesis is to explore and evaluate tools for static source code analysis based on defined evaluation criteria (such as accuracy, coverage, and the rate of false positives) using a vulnerable web application written in JavaScript. The theoretical part provides an overview of software development, starting with an explanation of the software development life cycle, discussing the issue of software vulnerabilities, and concluding with the security analysis of source code. In the practical part, suitable technologies were selected and prepared for use. Afterwards, testing of the chosen SAST tools was conducted, and the obtained results were evaluated. The final section summarizes the findings of the analyses. The thesis thus provides a comprehensive overview of the capabilities and limitations of the selected SAST tools and their contribution to improving software security.

KEYWORDS

Static analysis; secure development; software vulnerabilities; SAST tools; software development life cycle

PODĚKOVÁNÍ

Děkuji Mgr. Petru Novotnému za cenné připomínky a rady, které mi poskytl při vypracování maturitní práce.

V Třebíči dne 16. října 2025 podpis autora

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval/a samostatně a uvedl/a v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil/a.

V Třebíči dne 16. října 2025

podpis autora

Obsah

[Úvod 5](#_Toc2130715949)

[1 Teoretická část 6](#_Toc1086292808)

[1.1 Životní cyklus vývoje software 7](#_Toc776636142)

[1.1.1 Verzování vývoje 8](#_Toc1319577843)

[1.1.2 Proces DevOps 8](#_Toc1055380097)

[1.1.3 Bezpečný vývoj 9](#_Toc157953512)

[1.2 Softwarové zranitelnosti 10](#_Toc259920018)

[1.2.1 Známe zranitelnosti (CVE) 10](#_Toc1655416579)

[1.2.2 Kategorizace zranitelností (CWE) 10](#_Toc1100154409)

[1.2.3 Hodnocení zranitelností (CVSS) 11](#_Toc2043311482)

[1.3 Bezpečnostní analýza zdrojového kódu 11](#_Toc584401387)

[1.3.1 Statická analýza 11](#_Toc1491016337)

[1.3.2 SAST 11](#_Toc832150965)

[1.3.3 Srovnání vybraných nástrojů 12](#_Toc569457689)

[1.4 Závěr 12](#_Toc888773035)

[1.5 Seznam použitých zdrojů 13](#_Toc1027055236)

[1.6 Seznam použitých symbolů a zkratek 13](#_Toc1420254010)

[1.7 Seznamy použitých obrázků a tabulek 13](#_Toc2013592283)

[1.8 Seznam příloh 13](#_Toc599572802)

[Závěr 13](#_Toc191610483)

[Seznam použitých zdrojů 14](#_Toc1403892123)

[Seznam použitých symbolů a zkratek 17](#_Toc659856044)

[Seznam obrázků 18](#_Toc853737939)

[Seznam tabulek 19](#_Toc719251975)

[Seznam příloh 20](#_Toc143617415)

Úvod

V dnešní době jsou kybernetické útoky mnohem častější než útoky fyzické, a právě proto je nezbytné jim předcházet. Jednou z příčin může být veliké množství zranitelností ve zdrojovém kódu. Jedním z účinných opatření je bezpečný vývoj softwaru, který pomáhá minimalizovat riziko zneužití zranitelností. K dispozici je řada nástrojů a metod, které tento proces podporují – mezi nimi například statickou analýzu zdrojového kódu.

Hlavním cílem této práce je porovnat různé nástroje pro statickou analýzu zdrojového kódu, a to z hlediska jejich vlastností, jako je podpora programovacích jazyků, rychlost odezvy či možnosti integrace do vývojového procesu.

Práce je rozdělena do několika částí. V teoretické části je popsán životní cyklus vývoje softwaru spolu s procesy, které s ním úzce souvisejí – verzování vývoje, DevOps a zásady bezpečného programování. V další části jsou vysvětleny softwarové zranitelnosti, jejich význam a představuje související klasifikační a hodnotící systémy CVE, CWE a CVSS. Závěr teoretické části se zaměřuje na problematiku bezpečnostní analýzy zdrojového kódu, obecný princip statické analýzy, fungování SAST nástrojů, a nakonec i na srovnání vybraných SAST nástrojů z pohledu jejich vlastností a praktického využití.

# Teoretická část

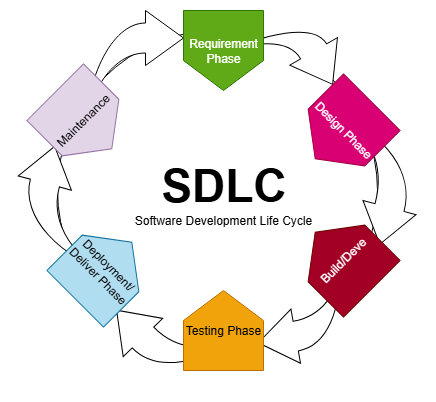
Teoretická část se zaměřuje na bezpečný vývoj software prevenci jeho zranitelností. Úvodem se zaměřuje na životní cyklus vývoje, verzování, DevOps a principy bezpečného kódu. Dále se věnuje softwarovým zranitelnostem, kategorizaci a hodnocení. Závěrem se zaměřuje na bezpečnostní analýzu zdrojového kódu, statickou analýzu a srovnání vybraných SAST nástrojů pro odhalování chyb a rizik.

## Životní cyklus vývoje software

Životní cyklus vývoje softwaru, tzv. SDLC (Software Development Life Cycle) je strukturovaný a systematický proces, který vede vývoj softwaru od jeho návrhu až po nasazení a údržbu. SDLC poskytuje jasný rámec pro plánování, tvorbu a správu aplikací, a zajišťuje, že vývoj probíhá efektivně a v souladu s požadavky projektu a očekáváními uživatelů.

Proces SDLC se obvykle dělí do několika fází:

1. Plánování – Identifikovat rozsah, cíl a požadavky projektu
2. Analýza – Shromáždit a zkontrolovat data o požadavky projektu
3. Návrh – Definovat architektury projektu
4. Kódování – Napsat počáteční kód
5. Testování – Otestovat kód a odstranit chyby
6. Nasazení – Nasadit kód do produkčního prostředí
7. Údržba – Průběžně provádět opravy a vylepšení

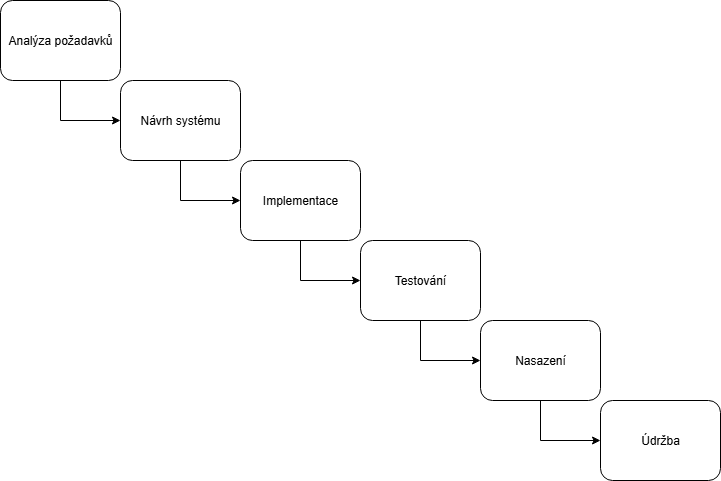


Obrázek – SDLC věc

SDLC má i mnoho modelů:

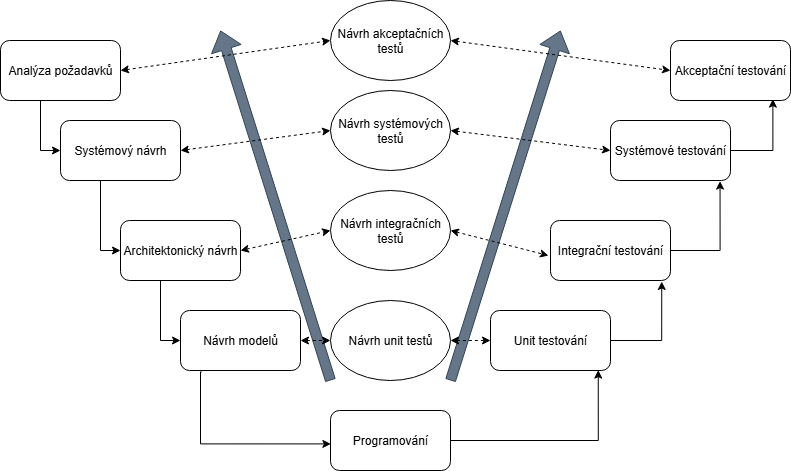
1. Vodopád

Tento model byl prvním procesním modelem. Označuje se jako lineárně-sekvenční model životního cyklu. Každá fáze musí být dokončena před zahájením další fáze. V modelu Vodopád se fáze nepřekrývají.



2. V-model

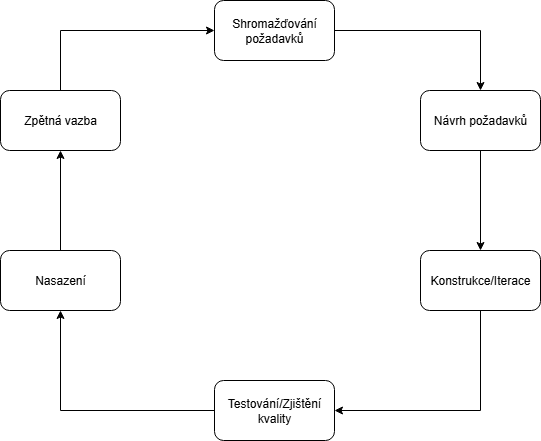
Dalším modelem SDLC je V-model, kde provádění procesů probíhá sekvenčně ve tvaru písmene V. Tento model je rozšířením modelu vodopádu a je založen na propojení testovací fáze s každou odpovídající fází vývoje. Jedná se o vysoce disciplinovaný model a další fáze začíná až po dokončení předchozí fáze.



3. Agilní model

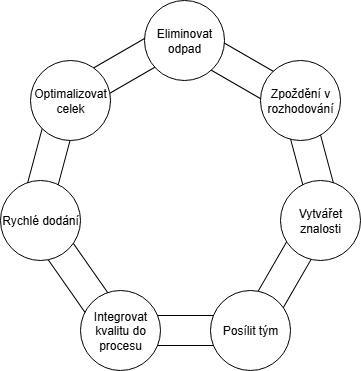
Tento model je velmi odlišný od předchozích modelů. Pomocí tohoto modelu rozdělujeme projekt do několika dynamických fází, běžně známých jako sprinty. Po každém sprintu týmy reflexují a ohlížejí se zpět, aby zjistily, zda by se dalo něco vylepšit, aby mohly případně upravit svou strategii pro další sprint.

Mezi nejběžnější metodologie patří Kanban a Scrum.



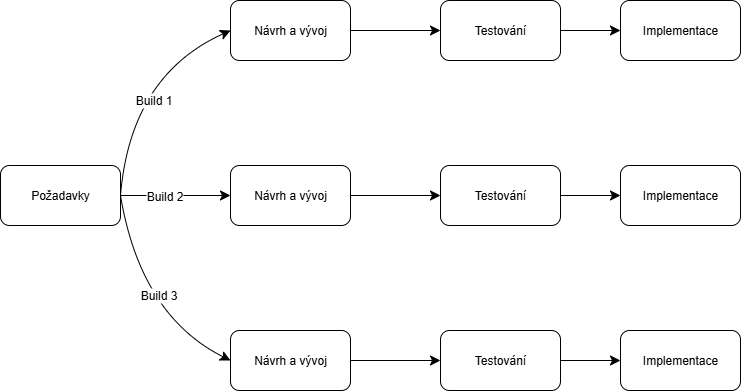
4. Lean

Lean model je založený na optimalizaci času a zdrojů vývoje, eliminaci plýtvání a v konečném důsledku dodávání pouze toho, co produkt potřebuje.



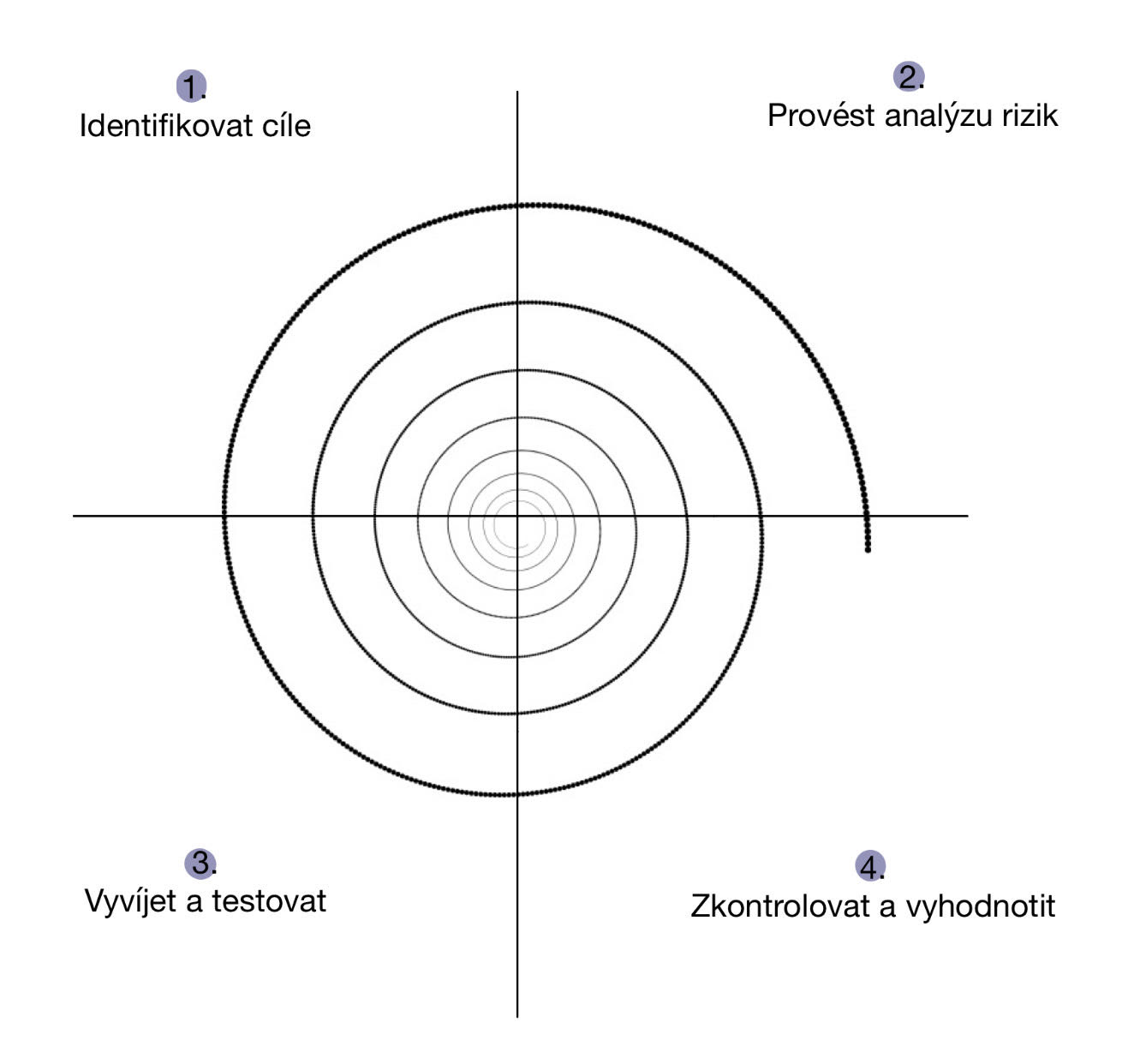
5. Iterativní model

Model, ve kterém se počáteční vývojové práce provádějí na základě dobře stanovených základních požadavků a k tomuto základnímu software se prostřednictvím iterací přidávají následná vylepšení, dokud není vytvořen finální systém.



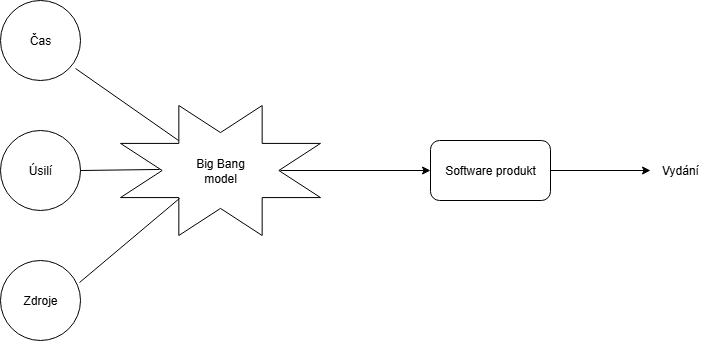
6. Spiral

Spirálový model je kombinace iterativního modelu a vodopádového modelu. Používá se především softwarovými inženýry a je preferován pro velké, drahé a složité projekty. Projektový manažer určuje počet smyček, které se liší v závislosti na projektu. Každá smyčka spirály je fází v modelu procesu vývoje software.



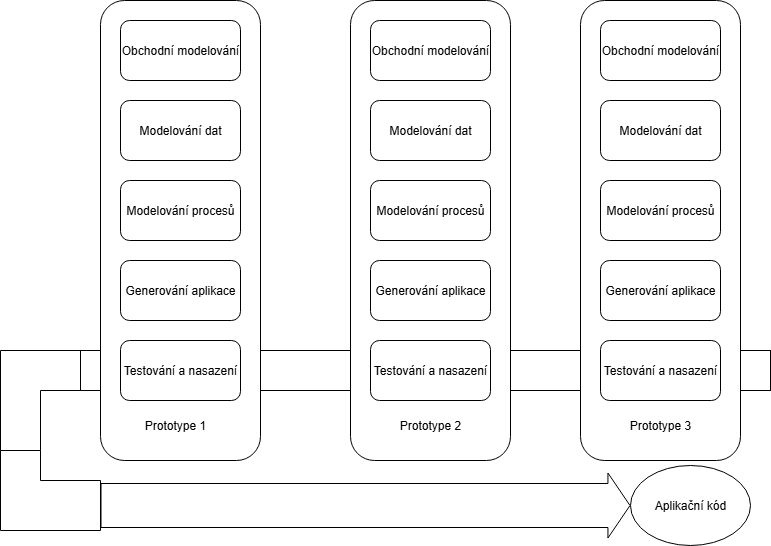
7. Big Bang

Big Bang je nejjednodušší model, protože nevyžaduje téměř žádné plánování. Vyžaduje však spoustu finančních prostředků a kódování a zabere více času. Tento model kombinuje čas, úsilí a zdroje pro vytvoření produktu. Produkt je postupně vytvářen podle požadavků zákazníka, nicméně konečný produkt nemusí splňovat skutečné požadavky.



8. RAD (Rapid Application Development)

Rad neboli v překladu rychlý vývoj aplikací je vývoj software založený na prototypování a rychlé zpětné vazbě s menším důrazem na specifické plánování. Obecně upřednostňuje vývoj a tvorbu prototypu před plánováním.



### Verzování vývoje

Verzování vývoje je proces sledování a řízení změn v softwarovém kódu. Systémy pro verzování vývoje jsou nástroje, které umožňují vývojovým týmům efektivně spravovat a uchovávat historii změn během celého vývojového procesu. Tyto nástroje usnadňují spolupráci více vývojářů na jednom projektu, umožňují sledovat jednotlivé úpravy a v případě potřeby se vrátit k předchozím verzím.

Systémy pro správu verzí (VCS) se stále vylepšují a některé jsou lepší než jiné. Tyto systémy jsou někdy známe jako SCM (Source Code Management) nástroje nebo RCS (Revision Control Systém). Git je jedním z nejpopulárnějších VCS nástrojů, který se dnes používá. Git je bezplatný a open source. Ať už je použit jakýkoliv systém, hlavní výhody jsou následující.

1. Kompletní dlouhodobá historie změn souboru. Vytváření, mazání a také úpravy jejich obsahu, to vše zahrnují změny. Historie by měl být také zaznamenán autor, datum a poznámky o účelu změny. Návrat k předchozím verzím v kompletní historii neschází, což pomáhá vyhledávat příčiny chyb, a je potřebná při opravě problémů ve starších verzích softwaru.
2. Větvení a slučování. Systém podporuje jak souběžnou práci týmů, tak i práci jednotlivce, který má možnost pracovat na nezávislých proudech změn. V VCS nástrojích lze vytvořit tak zvané „větve“ pro udržení více proudů práce na sobě nezávislých a zároveň poskytuje možnost sloučit práce zpět dohromady.
3. Sledovanost. S analýzou hlavních příčin a dalšími forenzními analýzami může pomoci schopnost sledovat každou změnu, která byla provedena na softwaru a propojit ji se softwarem pro řízení projektů a sledování chyb, jako je Jira, a schopnost anotovat změny zprávou popisující účel a záměr provedené změny. Mít anotovanou historii může vývojářům umožnit provádět správné harmonické změny.

Verzovacích systémů je mnoho, ovšem zde se zaměříme na systém Git. Je to nejrozšířenější moderní systém na světě. Má distribuovanou architekturu. Každý vývojář má zde pracovní kopii kódu, která může obsahovat historii všech změn projektu, což je výhodnější než v kdysi populárních systémech pro správu verzí, jako je CVS . Každou verzi projektu můžeme organizovat do tzv. větví, které můžou být jakkoliv pojmenovány. Je mnoho důvodů, proč požívat Git pro správu projektů:

1. Sledování změn: Systém Git umožňuje ukládat veškeré změny a sledovat je. Tedy pokud něco v projektu není plně funkční nebo nefunguje správně a je třeba se vrátit k předchozí verzi, historie změn to umožňuje.
2. Spolupráce: Usnadňuje spolupráci vývojářů na jednom projektu. Každý člen týmu může pracovat na vlastní verzi a změny lze poté snadno sloučit.
3. Větvení: Vytvořením větví máte možnost pracovat nezávisle. To je užitečné pro opravu chyb a vývoj nových funkcí, aniž by se změny promítly do hlavního projektu.
4. Zpětné vrácení změn: Pokud se stane, že nové funkce obsahují problémy nebo provedete něco špatně, je možné se jednoduše vrátit k předchozímu stavu projektu.
5. Ochrana dat: Kdyby došlo k chybě nebo selhání disku, data jsou v bezpečí uložena v tomto systému.
6. Podpora pro týmy: Jelikož Git umožňuje sledování, kdo a kdy provedl změny, pomáhá tak efektivně koordinovat práci větších týmů.
7. Open source a popularita: Systém je open source, což znamená, že zdrojový kód je veřejně dostupný a bezplatný. Má širokou komunitu uživatelů a aktivní vývoj.
8. Flexibilita: Git není využíván pouze na vývoj softwaru, ale také třeba i při psaní prací, správu obsahu webu a dalších.
9. Rychlost: Velikou výhodou je i rychlost a efektivita systému. Záznamy změn jsou úsporné, tudíž zabírají méně místa na úložišti.

Pokud se zaměříme na principy Gitu, základním stavebním prvkem je repozitář. Tedy místo, kde jsou uloženy a spravovány veškeré soubory, historie a metadata projektu. Navíc v něm můžou být uvedeny kompletní záznamy o změnách kódu, souborech, složkách a dalších součástí projektu.

Repozitář má hlavní funkci ukládat a udržovat celou historii projektu. Při uložení stavu projektu v určitém okamžiku vývoje provedeme tzv. commit. Commit je zaznamenán společně s autorem změny a kdy ke změně došlo v repozitáři. Což umožňuje sledovat vývoj projektu v čase, pohybovat se mezi verzemi a zjišťovat, kdo a kdy danou změnu provedl.

Důležitou součástí Gitu je jakým způsobem systém ukládá soubory. Každý soubor je uložen pouze jednou a poté se ukládají tzv. snapshoty. Veškeré soubory jsou v commitu uloženy jako snapshot.

Do Gitu je také možné ukládat i například obrázky či jiné netextové soubory díky tomu, že jsou uloženy všechny soubory binárně. Velikost repozitáře ovšem zůstává téměř identická.

Veškeré operace nejprve probíhají lokálně. Může se to zdát jako nevýhoda, protože se přidává krok mezi serverem a uživatele, ve skutečnosti to ale má spoustu výhod. Před tím, než se kód publikuje je snadné upravovat chyby na lokální práci. Jakmile se kód publikuje na server a někdo jiný ho stáhne, už není možné ho jakkoliv zpětně měnit.

Jelikož lze kód ukládat lokálně, není potřebný přístup k internetu. Téměř všechny operace lze provádět lokálně a data přidat na sever později, když je k dispozici síťové připojení.

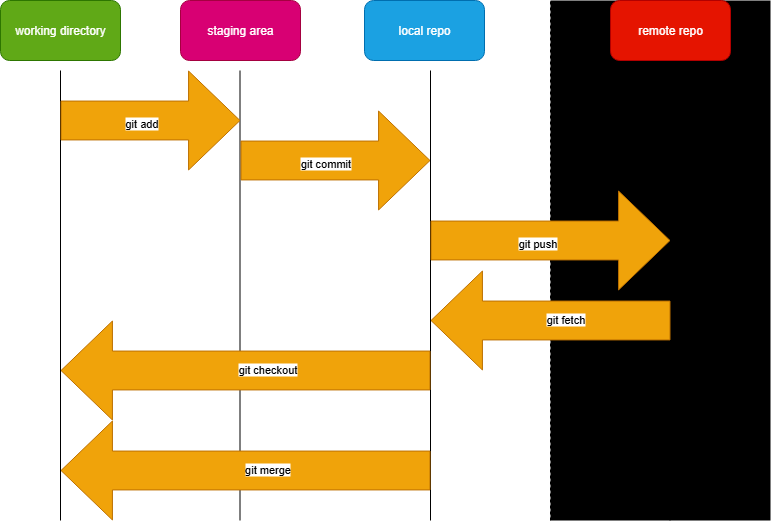
Git také klade velký důraz na integritu dat. Jakákoliv změna či poškození souboru je okamžitě detekováno. Systém ukládá soubory podle kontrolního součtu SHA-1, nikoli podle jména. Tento kontrolní součet provede sám Git pro každý soubor nebo složku.

Git funguje na principu přidávání dat. Tedy pokud bude odstraněn řádek, Git zaznamená informaci o odstranění a původní data zůstanou zachována. Je velmi obtížné poškodit data po provedení commitu natolik, aby už nebyla obnovitelná. Pokud nedojde k externí události, jsou soubory v repozitáři bezpečně uloženy.

Také je nutné zmínit, že celý proces správy verzí se skládá ze čtyř fází. Pomocí těchto fází se zvyšuje efektivita a koordinovanost spolupráce při verzování projektů. Jedná se o tyto čtyři fáze:

1. Pracovní složka: Jedná se o místo, kde provádíme změny a editujeme projekt. Lze zde vytvářet, mazat a upravovat soubory dle svých potřeb.
2. Staging area: V tomto prostoru vybíráme, které změny budou obsahovat následující commit.
3. Lokální repozitář: Po vybrání souboru ve staging area provedeme commit, který uloží změny do lokálního repozitáře. V tomto repozitáři se vyskytují veškeré změny a commity, které byly provedeny.
4. Vzdálený repozitář: Pokud chceme, aby změny, které byly uloženy na lokální repozitář, viděli i ostatní spolupracovníci, nahrajeme tyto změny na vzdálený server.

Následující graf zobrazuje tento proces:



Na trhu je mnoho platforem systému Git. Mezi nejpoužívanější patří:

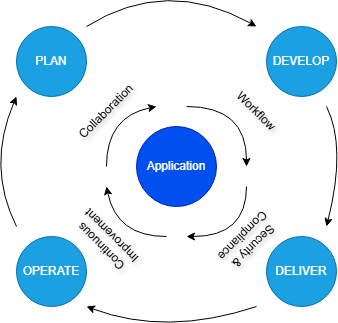
* GitHub
* Bitbucket
* GitLab
* Azure DevOps
* SourceForge
* Beanstalk
* FramaGit

### Proces DevOps

DevOps představuje propojení lidí, procesů a technologií s cílem zajistit kontinuální dodávání kvalitních produktů a služeb zákazníkům. Název vznikl spojením slov vývoj (development, Dev) a provoz (operations, Ops). Vývojáři, správci IT, testeři a specialisté na bezpečnost – dříve oddělené role a dnes pomocí DevOps spolu úzce spolupracují a koordinují své činnosti. Zavedením této kultury je umožněna rychlejší reakce na potřeby zákazníků, spolehlivost aplikací a přispívá k efektivnějšímu dosažení obchodních cílů organizace.

Životní cyklus aplikací je ovlivněn DevOps prostřednictvím jednotlivých fází. Každá fáze se spoléhá na ostatní, tedy každá role je v určité míře zapojená ve všech fázích. Těmito fázemi jsou:

* Plánování: V této fázi týmy vymýšlejí, definují a popisují funkce a možnosti projektů, které vytvářejí. Vytvářejí backlogy, sledují chyby a vizualizují průběh práce prostřednictvím řídících panelů.
* Vývoj: Fáze vývoje zahrnuje veškeré kódování – od samotného psaní kódu, testování, kontrolu a integraci mezi členy týmu, až po tvorbu build artefaktů.
* Doručování: Doručování je proces spolehlivého a konzistentního nasazování aplikací do produkčního prostředí. Fáze doručování také zahrnuje nasazení a konfiguraci plně spravované infrastruktury, která zajišťuje funkčnost jednotlivých prostředí. Během této fáze týmy definují postup pro správu verzí s jasně definovanými postupy ručního schvalování. Také nastavují automatizované brány, které přesouvají aplikace mezi jednotlivými fázemi, před tím, než se zpřístupní zákazníkům.
* Provoz: V provozní fázi se vývojáři zaměřují na údržbu a monitorování aplikací a řešení problémů s aplikací, která je již v provozu.



S DevOps souvisí proces CI/CD (Continuous Integration/Continuous Deployment). Kontinuální integrace (CI) je automatizovaný proces využívaný vývojovými týmy ke slučování a testování změn v kódu. Automatizované testy jsou spouštěny v rámci procesu CI, aby byla zajištěna kvalita kódu. Výstupy procesu CI jsou následně používány v procesech vydávání k častému nasazování nových verzí softwaru.

Průběžné doručování (CD) zahrnuje sestavení, testování a nasazení kódu do jednoho nebo více testovacích a produkčních prostředí. Postupné nasazování a testování ve více prostředích přispívá ke zvýšení kvality aplikace. Artefakty vytvořené v rámci CD, včetně infrastruktury a aplikací, jsou využívány automatizovanými procesy k vydávání nových verzí a oprav stávajících systémů. Nepřetržité monitorování a zasílání výstrah zajišťuje dohled nad celým procesem CD.

### Bezpečný vývoj

Bezpečný vývoj softwaru je metodologie pro tvorbu softwaru, která zahrnuje zabezpečení do každé fáze životního cyklu vývoje softwaru. Zabezpečení je integrováno do kódu od samého začátku, nikoli řešeno až po odhalení kritických nedostatků produktu testováním. Zabezpečení se stává součástí fáze plánování a začleňuje se dlouho před napsáním jediného řádku kódu.

Bezpečnost softwaru lze posílit prostřednictvím několika základních zásad vývoje. Jednou z nich je validace vstupu, která zajišťuje, že do systému vstupují pouze správná a očekávaná data. Tímto opatřením se nejen předchází ukládání nesprávných dat, ale zároveň se snižuje riziko útoků, jako jsou XSS, SQL injection a další.

Další klíčovou zásadou je sanitizace, tedy proces, při kterém se ověří vstup uživatele, aby se zjistilo, zda je bezpečný pro užití ve webové aplikaci.

Součástí bezpečného vývoje je také integrační testování, při kterém jsou jednotlivé softwarové moduly propojeny a testovány jako funkční celek.

Také je nezbytné zmínit penetrační testování v před produkčním nasazení software. Jedná se o testování v izolovaném a kontrolovaném prostoru, kde týmy ověřují software před jeho spuštěním.

## Softwarové zranitelnosti

Zranitelnost je jakákoli slabina v informačních systémech, vnitřních kontrolách nebo procesech systému organizace, kterou mohou útočníci zneužít. Zranitelnost sama o sobě představuje potenciální riziko, nikoli přímou hrozbu.

Hrozba nastává tehdy, kdy má protivník nebo útočník příležitost, schopnost a úmysl způsobit negativní dopad na systém. Mezi příklady hrozeb patří malware, injekce, phishingové útoky a další škodlivé aktivity.

Pokud je hrozba realizována, může být využit exploit – specifický typ škodlivého kódu, který zneužívá zranitelnosti systému a způsobuje nežádoucí či neočekávané chování. Útočníci takto často distribuují různé formy malwaru do napadených systémů.

### Známe zranitelnosti (CVE)

CVE (Common Vulnerabilities and Exposures) je systém číselného označení záznamů v databázi, která obsahuje definice veřejně známých zranitelností. Cílem CVE je usnadnit sdílení informací o zranitelnostech napříč různými platformami a nástroji pro etické hackery i bezpečnostní specialisty. Každý záznam CVE obsahuje identifikační číslo, stručný popis zranitelnosti a alespoň jeden veřejný odkaz na další relevantní informace.

Příkladem může být CVE-2023-23397. Jedná se o zranitelnosti Microsoft Outlook umožňující zvýšení oprávnění. Nebo zranitelnost CVE-2025-55177, která pojednává o neúplné autorizaci zpráv synchronizace propojených zařízení v aplikaci WhatsApp pro iOS mohla umožnit nesouvisejícímu uživateli spustit zpracování obsahu z libovolné adresy URL na cílovém zařízení.

Jednou z nejznámějších zranitelností je CVE-2017-0144. Umožňuje vzdáleným útočníkům spouštět libovolný kód. Známý je především proto, že tato zranitelnost byla zneužita skupinou kyberzločinců v květnu roku 2017. Byl to jeden z prvních ransomwarových útoků s názvem WannaCry. Šířil se jako červ, byl rychlý a díky tomu zvládl nakazit přes 230 000 počítačů ve 150 zemích během jednoho dne. Obětem zablokoval přístup nebo používaní, dokud nebylo zaplaceno výkupné. Původní výkupné bylo 300 dolarů v Bitcoinech, později se cena zvýšila na 600 dolarů. Marcus Hutchins dokázal tento incident nejdříve neutralizovat a následně zastavit.

### Kategorizace zranitelností (CWE)

CWE(Common Weakness Enumeration) je katalog běžných softwarových a hardwarových slabin, které tvoří hlavní příčiny bezpečnostních zranitelností. Zaměřuje se na typy chyb, jako je nesprávná validace vstupu, nebezpečné navrhové vzory nebo přetečení bufferu. Každé slabině je přiřazen jedinečný identifikátor (např. CWE-89 – SQL Injection), doplněný podrobným popisem, konkrétními příklady a informacemi o možných opatřeních ke zmírnění rizik.

Mezi nejvážnější slabiny momentálně patří například CWE-79, nebo-li Cross-site Scripting. Pomocí této slabiny může útočník manipulovat nebo ukrást soubory cookie, vytvářet požadavky vydávající se za legitimního uživatele, ohrozit důvěrná data či spustit škodlivý kód v prohlížeči oběti. Je tedy nutné se před touto slabinou bránit například pomocí validace a sanitizace vstupů nebo escapování výstupů. Nebo CWE-89 – SQL Injection, která umožňuje útočníkovi zadat vstup, pomocí kterého může získat přístup l datum, která by vidět neměl a v mnoha případech je i změnit či smazat. Většině případů můžete zabránit pomocí parametrizovaných dotazů namísto zřetězení řetězů v rámci dotazu.

### Hodnocení zranitelností (CVSS)

CVSS (Common Vulnerabiliti Scorint System) je standardizovaný rámec pro hodnocení závažnosti bezpečnostních chyb v informačních systémech. Každé zranitelnosti je přiřazeno skóre v rozmezí 0–10, přičemž vyšší hodnota označuje závažnější problém. Systém CVSS organizacím umožňuje efektivně prioritizovat bezpečnostní hrozby a rozhodnout, které zranitelnosti vyžadují okamžitou pozornost na základě jejich potenciálního dopadu.

Následující tabulka znázorňuje rozdělení jednotlivých úrovní závažnosti podle CVSS verze 4.0:

|  |  |
| --- | --- |
| Skóre | hodnocení na základě závažnosti |
| 0.0 | Žádná |
| 0.1–3.9 | Nízká |
| 4.0–6.9 | Střední |
| 7.0–8.9 | Vysoká |
| 9.0–10.0 | Kritická |

## Bezpečnostní analýza zdrojového kódu

Bezpečnostní analýza kódu je proces, při kterém se analyzuje zdrojový kód aplikace s cílem odhalit bezpečnostní chyby či zranitelnosti. Kontrola se zaměřuje například na logické chyby, správnou implementaci specifikací nebo dodržování zásad psaní kódu.

Automatizovaná kontrola kódu využívá nástroje, které provádějí revizi zdrojového kódu podle předem definovaných pravidel. Tento přístup umožňuje rychlejší identifikaci potenciálních problémů než manuální kontrola.

Manuální kontrola kódu spočívá v pečlivém procházení zdrojového kódu člověkem, řádek po řádku, s cílem odhalit bezpečnostní slabiny. Manuální revize dokáže zohlednit kontext rozhodnutí vývojáře a obecnou logiku aplikace, což automatizované nástroje často nedokážou. Tento přístup je strategičtější a umožňuje řešit konkrétní problémy cíleně.

### Statická analýza

Statická analýza hledá zranitelnosti, které jsou náchylné ke kyber útokům. Tato analýza skenuje kód ještě před kompilací. Je také známé jako white box testing, tudíž testování se znalostí kódu, struktury a fungování software. Metoda poskytuje pohled na vnitřní procesy a umožňuje najít konkrétní oblasti, které by mohly obsahovat chyby.

Oproti white box testing existuje i black box testing, kde znalosti vnitřních mechanismů neznáme, tudíž se nemůže jednat o statickou analýzu. Nezaměřujeme se tedy na kód, ale pouze pozorujeme její výkon. Zaměřujeme se na vstupy a výstupy a zajišťujeme, aby se software choval podle očekávání.

### SAST

Bla bla

### Srovnání vybraných nástrojů

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SAST nástroj | Jazyky | Open source | GUI | CI/CD | Generování reportů | Cloud | IDE |
| CodeQL | 12 | Ano | Ano | Ano | Ne | Ano | Ano |
| GitLab SAST | 19 | Ano | Ano | Ano | Ne | Ano | Ano |
| SonarQube | 27 | Ano | Ano | Ano | Ne | Ano | Ano |
| Snyk | 17 | Ano | Ano | Ano | Ne | Ano | Ano |
| Bearer | 10 | Ano? | Ne | Ano | Ne | Ano | Ne |
| Aikido | 16 | Ano | Ano | Ano | Ano | Ano | Ano |
| Horusec | 19 | Ano | Ano | Ano | Ano | Ne | Ano |

## Závěr

Závěr obsahuje stručné shrnutí získaných poznatků, uvedení dalších možných postupů či řešení, hodnocení dostupné odborné literatury, ze které bylo čerpáno. Závěr by měl obsahovat kritické porovnání záměru práce a dosažených výsledků, srovnání dosažených výsledků s dosud známými poznatky a popis odlišností od doposud známých skutečností. Závěr může naznačit praktické uplatnění výsledků práce.

## Závěr

Vytvořená šablona maturitních prací obsahuje formální požadavky maturitních prací na SPŠT Třebíč. Jedná

se zejména o upravené styly v dokumentu, podrobný popis jednotlivých částí maturitní práce a jejího obsahu, snadno editovatelné záhlaví a zápatí s automatickým číslováním stránek a propojení stylů se seznamy a obsahem.

Seznam použitých zdrojů

1. JACKSON, Gita; KOSINSKI, Matthew a HOLDSWORTH, Jim. *What is the software development life cycle (SDLC)?* Online. IBM. 2025. Dostupné z: <https://www.ibm.com/think/topics/sdlc>. [cit. 2025-08-26].
2. TUTORIALSPOINT. *SDLC - Waterfall Model*. Online. Dostupné z: <https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm>. [cit. 2025-09-14].
3. GEEKSFORGEEKS. *SDLC V-Model - Software Engineering*. Online. 2025-08-11. Dostupné z: <https://www.geeksforgeeks.org/software-engineering/software-engineering-sdlc-v-model>. [cit. 2025-09-14].
4. LAOYAN, Sarah. *What is Agile methodology? (A beginner’s guide)*. Online. 2025. Dostupné z: <https://asana.com/resources/agile-methodology>. [cit. 2025-09-14].
5. PRODUCTPLAN. *Lean Software Development*. Online. Dostupné z: <https://www.productplan.com/glossary/lean-software-development>. [cit. 2025-09-14].
6. SCALER. *SDLC - Iterative Model*. Online. 2023-05-22. Dostupné z: <https://www.scaler.com/topics/software-engineering/iterative-model-in-software-engineering>. [cit. 2025-09-14].
7. HASHEMI-POUR, Cameron. *What is the spiral model and how is it used?* Online. 2025. Dostupné z: <https://www.techtarget.com/searchsoftwarequality/definition/spiral-model>. [cit. 2025-09-14].
8. GEEKSFORGEEKS. *Overview of Big Bang Model*. Online. 2025-07-23. Dostupné z: <https://www.geeksforgeeks.org/software-engineering/overview-of-big-bang-model>. [cit. 2025-09-14].
9. KISSFLOW, Team. *What is Rapid Application Development (RAD)? An Ultimate Guide for 2025*. Online. 2025-08-11. Dostupné z: <https://kissflow.com/application-development/rad/rapid-application-development>. [cit. 2025-09-14].
10. *What is version control?* Online. Atlassian. Dostupné z: <https://www.atlassian.com/git/tutorials/what-is-version-control>. [cit. 2025-08-27].
11. ATLASSIAN. *What is Git?* Online. Dostupné z: <https://www.atlassian.com/git/tutorials/what-is-git>. [cit. 2025-10-02].
12. CHACON, Scott a STRAUB, Ben. *Getting Started - What is Git?* Online. Pro Git (Second Edition). 2014. Dostupné z: <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>. [cit. 2025-08-27].
13. GIT. *Git*. Online. Git-SCM. Dostupné z: <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>. [cit. 2025-08-27].
14. VALKOVIČ, Patrik. *Lekce 1 - Git - Historie a principy*. Online. Dostupné z: <https://www.itnetwork.cz/programovani/git/git-tutorial-historie-a-principy>. [cit. 2025-10-07].
15. BENSON, David. *The Top 10 Version Control Tools*. Online. Logit.io blog. 2024, 04.02.2025. Dostupné z: <https://logit.io/blog/post/version-control-tools>. [cit. 2025-08-27].
16. MICROSOFT. *Co je DevOps?* Online. Microsoft Learn. 2023. Dostupné z: <https://learn.microsoft.com/cs-cz/devops/what-is-devops>. [cit. 2025-08-27].
17. HYPERPROOF. *Secure Software Development: Best Practices, Frameworks, and Resources*. Online. 02. 08. 2024. Dostupné z: <https://learn.microsoft.com/cs-cz/devops/what-is-devops>. [cit. 2025-08-27].
18. BEZPEČNÝ KÓD. *Validace vstupu*. Online. Dostupné z: <https://bezpecnykod.cz/cheat-sheety/validace-vstupu>. [cit. 2025-08-27].
19. TAYLLORCOX. *Sanitizace*. Online. Dostupné z: <https://www.tx.cz/slovnik/pci-dss/sanitizace>. [cit. 2025-08-27].
20. DUBOVECKÁ, Klára. *Analýza bezpečnosti #5: Zranitelnost a její role při analýze rizik*. Online. KYBEZ. 2023. Dostupné z: <https://kybez.cz/analyza-bezpecnosti-5-zranitelnost-a-jej-rola-pri-analyze-rizik/?gad_source=1&gad_campaignid=20944701777&gclid=Cj0KCQjw8KrFBhDUARIsAMvIApaSw0mMhksjTjSJMYXxHQPUna6QXX8EtSqTArh60fAtWIGzKRpbLdEaAsBtEALw_wcB>. [cit. 2025-08-27].
21. ESET. *Co je exploit?* Online. Dostupné z: <https://www.eset.com/cz/exploit>. [cit. 2025-08-27].
22. SPLUNK, Inc. *Vysvětlení zranitelností, hrozeb a rizik*. Online. 2024. Dostupné z: <https://www.sands.cz/vysvetleni-zranitelnosti-hrozeb-a-rizik>. [cit. 2025-08-27].
23. RED HAT. *What is a CVE?* Online. 2024. Dostupné z: <https://www.redhat.com/en/topics/security/what-is-cve>. [cit. 2025-08-27].
24. MITRE. *What is CWE?* Online. 05. 06. 2023. Dostupné z: <https://cwe.mitre.org/about/new_to_cwe.html>. [cit. 2025-08-27].
25. GOODMAN, Courtney. *What is the Common Vulnerability Scoring System (CVSS)?* Online. Balbix. 25. 10. 2024. Dostupné z: <https://www.balbix.com/insights/understanding-cvss-scores>. [cit. 2025-08-27].
26. BLACK DUCK. *Secure Code Review*. Online. Dostupné z: <https://www.blackduck.com/glossary/what-is-code-review.html>. [cit. 2025-08-27].
27. IN-COM. *Co je statická analýza?* Online. 2024. Dostupné z: <https://www.in-com.com/cs/blog/what-is-static-code-analysis/#what-is-static-analysis>. [cit. 2025-08-27].
28. JIT. *Static Application Security Testing (SAST): What You Need to Know*. Online. Jit. 2025. Dostupné z: <https://www.jit.io/resources/appsec-tools/static-application-security-testing-sast-what-you-need-to-know>. [cit. 2025-08-27].
29. GITLAB DOCS. *Static Application Security Testing (SAST)*. Online. Jit. 2025. Dostupné z: <https://docs.gitlab.com/user/application_security/sast>. [cit. 2025-08-27].

|  |  |  |
| --- | --- | --- |
|  |  |  |

Seznam obrázků

[Obrázek 1 – SDLC věc 8](#_Toc209087428)

Seznam tabulek

**Nenalezena položka seznamu obrázků.**