Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Bezpečnost vývoje aplikací při použití agilní metodiky

Adam Hruška

Bakalářská práce

2020

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Adam Hruška**
Osobní číslo:	**I17108**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Bezpečnost vývoje aplikací při použití agilní metodiky**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem této práce je vytvoření popisu procesů, pravidel a doporučení, které by měli zajistit vysokou míru bezpečnosti vyvíjené aplikace od počátku jejího vývoje až po doručení do produkčního prostředí.
Teoretická část se bude věnovat rešerši literatury a odborných publikací zabývající se všemi fázemi aplikačního vývoje, od tvorby kódu, až po nasazení aplikace, včetně automatizace procesů.
Praktická část práce bude obsahovat popis a postup moderního způsobu agilního vývoje aplikací, porovnání s „Waterfall" modelem, shrnutí výhod Agilu, techniky a procesy, které jsou při této metodě praktikovány a způsob dodávek aplikačních funkcionalit v rámci pravidelných sprintů až po funkční produkční aplikaci. Součástí praktické části bude popis výše zmíněných kroků v rámci reálně vyvíjené JAVA aplikaci.

Rozsah pracovní zprávy: **30**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

JAQUITH, Andrew. Security metrics: Replacing Fear, Uncertainty, and Doubt. Boston: Addison-Wesley Professional 2007. 306 s. ISBN: 978-0-321-34998-9.
SHOSTACK, Adam, Andrew STEWART. The New School of Information Security. Boston: Addison-Wesley Professional 2008. ISBN: 978-0-321-50278-0.
HOWARD, Michael, LEBLANC, David. Bezpečný kód. Brno: Computer Press a.s., 2008. 895 s. ISBN: 978-80-251-2050-7.
HUSEBY, Sverre H. Zranitelný Kód. Brno: Computer Press a.s., 2006. 207 s. ISBN: 80-251-1180-6.

Vedoucí bakalářské práce: **Ing. Miloslav Macháček, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. listopadu 2019**
Termín odevzdání bakalářské práce: **7. května 2020**

Univerzita Pardubice
Česká republika
L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 17. prosince 2019

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 17. 12. 2020

Adam Hruška

PODĚKOVÁNÍ

Tímto bych rád poděkoval panu Ing. Miloslavu Macháčkovi, Ph.D. za vedení práce a rady při vypracovávání práce. Dále děkuji Bc. Filipu Štěrbovi za cenné rady při zpracování práce a vedení studentské stáže ve společnosti Škoda Auto. Taktéž bych velice rád poděkoval celé své rodině za nedocenitelnou podporu během celého mého studia.

ANOTACE

Tato bakalářská práce se věnuje otázce zajištění zvýšení bezpečnosti při vývoji aplikací. Cílem práce je vytvoření popisu procesů, pravidel a doporučení, které by měly zajistit vysokou míru bezpečnosti vyvíjené aplikace od počátku jejího vývoje až po doručení do produkčního prostředí. Cíle práce je dosaženo metodou studia dostupných zdrojů a jejích analýzy. Praktická část bakalářské práce se zabývá popisem procesů a technik zvýšení bezpečnosti vývoje softwaru na reálně vyvíjené aplikaci ve společnosti Škoda Auto.

KLÍČOVÁ SLOVA

Aplikace, bezpečnostní chyby, bezpečnostní testování, modelování hrozeb, životní cyklus aplikace, agilní metodiky, Scrum, Extrémní programování, vývoj softwaru

TITLE

Security of application development using agile methodology

ANNOTATION

The topic of the thesis the issue of increasing security during the application development. The goal is to create a description of processes, rules and recommendations, which should ensure a high level of security of the developed application from the beginning of its development to delivery to the production environment. The objectives of the work are achieved by study of available resources and their subsequent analysis. Practical part of the bachelor thesis deals with the description of processes and techniques of increasing the security of software development on a real application in the company Škoda Auto.

KEYWORDS

Applications, security flaws, security testing, threat modelling, application lifecycle, agile methods, Scrum, Extreme programming, software development

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SLOVNÍK POJMŮ

Security Operation Centre – Tým nepřetržitě monitorující systémy Škoda Auto.

Computer Emergency Response Team – Tým informační bezpečnosti, který se stará o ochranu před kybernetickými incidenty, jejich odhalování a reakci na ně.

WebSeal – Vysoce výkonný webový server, který aplikuje bezpečnostní politiku do prostoru webových objektů.

LDAP – Multiplatformní protokol používaný pro ukládání a přístup k datům v adresářovém serveru.

Hash – Digitální otisk textu, který je výsledkem hashovací funkce.

SAP – Centralizovaná správa dat.

LDAPWS – Webová služba LDAP, která provádí ověřování a vyhledávání vůči Active Directory.

HTTP – Internetový protokol určený pro komunikaci v počítačové síti.

HTTPS – Internetový protokol umožňující zabezpečenou komunikaci v počítačové síti.

Microsoft Team Foundation Server – Systémem spravující vývojový proces (sestavení, nasazení, testování).

CI / CD pipeline – Seznam kroků, které je potřeba provést, aby bylo možné dodat novou verzi softwaru.

SEZNAM ZKRATEK

OWASP	The Open Web Application Security Project
TFS	Microsoft Team Foundation Server
SOC	Security Operation Centre
CERT	Computer Emergency Response Team
ACL	Access Control List
JCC	Java Competence centre
SDL	Secure Development Lifecycle
ASA	Application Security Assessment
LDAP	Lightweight Directory Access Protocol
AES	Advanced Encryption Standard
AMS	Application Management Services
CI / CD	Continuous Integration / Continuous Delivery
LCO	Life Cycle Objectives
LCA	Life Cycle Architecture
IOC	Initial Operational Capability
RUP	Rational Unified Process
XP	Extreme Programming
DOS	Denial of service
HTTPS	Hypertext Transfer Protocol Secure
HTTP	Hypertext Transfer Protocol
B2C	Business To Customer
SAP	Systems – Aplication – Products in data processing
LDAPWS	Lightweight Directory Access Protocol Web Service
OU	Organization Unit
CI	Corporate identity

1 ÚVOD

V dnešní době se klade velký důraz na bezpečnost, jinak tomu není ani v oblasti vývoje softwaru. Internetové připojení je dostupné po celém světě a s tím se pojí počet lidí připojených k internetu. Pokud jde o firmy, tak již snad neexistuje firma, která by neměla připojení k internetu. U většiny firem je internet klíčový pro fungování jejich byznysu. Prostřednictvím internetu firmy nabízejí své služby či produkty a k tomu využívají různé typy aplikací. Může se jednat o čistě webové aplikace nebo o aplikace, které komunikují se vzdálenými zdroji. Některé aplikace si firmy vyvíjejí sami, jiné nakupují od ostatních společností zabývajících se tímto odvětvím.

S rostoucí počtem používaných aplikací roste také možnost, že nějaká z nich bude obsahovat bezpečnostní chybu. Tyto bezpečnostní chyby mohou být skvělou příležitostí pro případné útočníky, kteří chtějí odcizit data, nebo dokonce vyřadit aplikaci z provozu. Důsledky takového útoku mohou být obrovské, může se jednat o velké finanční ztráty nebo o poškození dobrého jména firmy. Ztráta dobrého jména může mít vetší následky než odcizení dat nebo finanční ztráty. Dobré jméno se totiž těžko získává zpět, proto spousta společností tají na ně provedené útoky nebo opravené bezpečnostní chyby právě z důvodu obavy o ztrátu důvěry zákazníků.

Bezpečnost ve vývoji softwaru by měla otázkou číslo jedna. Týmy, které se zabývají vývojem aplikací, by měly mít dobré povědomí o bezpečnostních problémech, aby jim mohly efektivně předcházet. Není pravidlem, že otázka bezpečnosti aplikací bývá řešena od počátku vývoje. To může mít za následky zvýšení nákladů na vývoj nebo velké množství bezpečnostních chyb ve výsledné aplikaci. Proto je nutné zajistit, aby otázka bezpečnosti aplikací byla řešena už od začátku projektu a ve všech jeho fázích.

2 METODIKY VÝVOJE SOFTWARU

Metodiky vývoje softwaru můžeme označit jako komplexní postupy a návody pro vývoj softwarové aplikace. Tyto postupy a znalosti jsou používány ke kontrolování a plánování vývoje informačních systémů či jiných softwarových projektů. Na světě existuje mnoho metodik vývoje softwaru. Každá metodika má svoje silné i slabé stránky, a proto není snadné vybrat tu správnou. Je těžké jednotlivé metodiky srovnávat, protože se mnohé metodiky zaměřují jen na některé fáze či aspekty samotného vývoje. Důvodů pro velké množství metodik je mnoho – velké vývojářské týmy, jednotlivci, různé technologie a organizace, které se vývojem zabývají. Proto je možné metodiky vývoje klasifikovat podle různých kritérií, které mohou pomoci vybrat tu správnou.

Kritérium zaměření metodiky:

- globální – určené pro vývoj SW v rámci celého podniku,
- projektové – určené pro vývoj části SW.

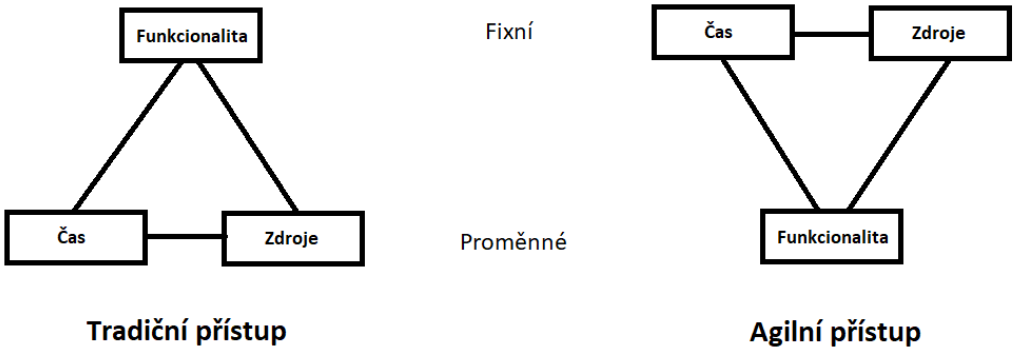
Kritérium rozsahu:

- metodiky zabývající se celým životním cyklem vývoje,
- metodiky, které se zaměřují pouze na určité etapy vývoje.

Kritérium váhy:

- těžké metodiky – obsahují podrobný popis, přesně definované procesy, činnosti a vytvořené produkty,
- lehké metodiky – definují principy a praktiky pro úspěšné zvládnutí projektu.

Podle posledního kritéria je možné metodiky rozdělit na tradiční (rigorózní) a agilní. Agilní metodiky patří mezi lehké a kladou důraz na produkt, komunikaci se zákazníkem a jsou tolerantní k změnám požadavků. Kdežto tradiční metodiky bývají podrobné, objemné a těžké. Obecně platí, že při používání agilních metodik je pevně daný čas a zdroje, ale funkcionalita se může v průběhu vývoje měnit. Naopak u tradičních metodik je pevně daná funkcionalita dosažená v proměnném čase a s proměnnými zdroji. [1], [2]



Obrázek 1: Porovnání tradičního a agilního přístupu[1]

[1] Zdroj: Autor

3 TRADIČNÍ METODIKY VÝVOJE SOFTWARU

Tradiční metodiky jsou mnohdy nazývány jako rigorózní, jsou často velice rozsáhlé, protože podrobně popisují činnosti a procesy vývoje softwaru, bývají zakládány na sériovém vývoji, proto je jasné, jak jednotlivé fáze půjdou za sebou. V této kapitole budou popsány tradiční metodiky založené na sériovém (vodopádovém) vývoji, ale i tradiční metodiky založené na iterativním a inkrementálním vývoji. [2]

Na úvod je dobré této kapitoly je nutné uvést základní pojmy, které se budou vztahovat k modelům životních cyklů uváděných v této kapitole.

- **Rizika** – základní pojem, jako rizika se považují všechny situace a události, které mohou způsobit, že projekt nebude úspěšně dotažen do cíle. U každého rizika se musí stanovit jeho závažnosti a pravděpodobnost výskytu. Proces stanovování rizik probíhá v každé fázi vývoje a jeho cílem je odstranit události, které mohou ohrozit hladký průběh vývoje. [2]
- Typy rizik:
- **Projektová rizika** – Jedná se například o snížení rozpočtu určeného pro vývoj a odchod některých členů vývojového týmu. [2]
- **Technická rizika** – Problémy s operačními systémy, hardwarem, či s vývojovými nástroji. [2]
- **Obchodní rizika** – Špatně zvládnutý marketing. [2]

- **Prototyp** – jedná se o nejpoužívanější metodu odstranění technických rizik. Dostatečně dobře vyvinutý prototyp dokáže odstranit nejasnosti ve specifikaci a může být dobrým základem pro další cyklus.

  Typy prototypů:

  - **Ilustrativní prototyp** – Tento prototyp se používá při komunikaci se zákazníkem, klade důraz na uživatelské rozhraní a jeho vzhled. [2]
  - **Funkční prototyp** – Jedná se o zmenšeninu celého systému s omezeným počtem funkcí. Další funkce jsou postupně přidávány v souvislosti s analýzou rizik. [2]
  - **Ověřovací prototyp** – Vždy se provádí implementace jen určité části systému, cílem je potvrdit zvolené technologie a ujasnit si požadavky na systém. [2]

- **Plánování** – Provádí se kontrola projektu a klade se důraz na naplánování další fáze projektu. [2]

- **Cyklický**, **iterativní vývoj** – Probíhá opakování fází vývoje. Po každé iteraci je možné zkontrolovat výsledek a porovnat ho s požadavky, tím se vývoj urychlí a lépe se odhalí jednotlivé chyby ve specifikaci. [4]

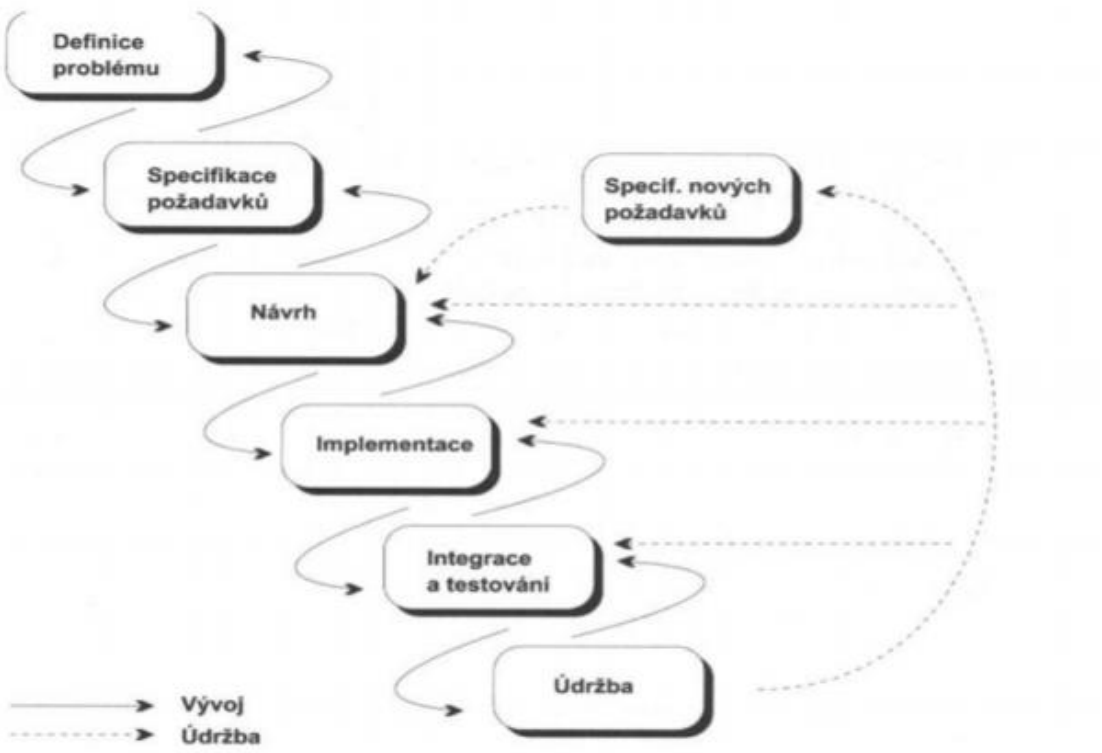
3.1 Vodopádový model životního cyklu

Vodopádový model je prvním modelem, který přinesl základní členění procesu vývoje na jednotlivé aktivity, které na sebe sériově navazují. Jeho vznik se datuje do roku 1970. V době jeho vzniku byl opravdovým průlomem ve světě vývoje softwaru. [2]

Ve vodopádovém modelu se přesně za sebou dodržují jednotlivé fáze s minimálními nebo žádnými iteracemi. V průběhu vývoje probíhá posouvání v jednotlivých fázích, takže vzniká dojem vodopádu, odtud také vznikl název této metodiky. Dokud není předchozí fáze dokončena, tak není možné přejít na další, to znamená, že není možné provádět více fází najednou. V každé fázi vzniká obsáhlá dokumentace, takže je možné dobře sledovat postup vzhledem k plánu vývoje. Nejsou zde ani žádné prototypy a při vývoji neprobíhá žádná průběžná komunikace se zákazníkem. Vodopádový model se stal inspirací mnohým autorům a existuje velké množství jeho verzí. [2]

Hlavní fáze vodopádového modelu:

1. Definice problému, poznání zákazníka
2. Analýza a specifikace požadavků
3. Návrh systému
4. Implementace
5. Integrace a testování
6. Provoz a údržba



Obrázek 2: Vodopádový model životního cyklu[2]

Obrázek číslo 2 znázorňuje posouvání se v jednotlivých fázích vývoje při použití vodopádového modelu. Je zřejmé, že nalezení chyby může vývoj velice prodloužit a zdražit, protože chyba se může nacházet v jedné z předchozích fází a je nutné se vrátit o několik fází zpět, nebo dokonce na samotný začátek, zde opravit chybu a s přihlédnutím ke změnám absolvovat znovu stejné fáze. Aby nedocházelo k takovýmto chybám, je nutné při vývoji dbát na ověřování a testování jednotlivých fází. Z toho důvodu také vzniká již zmíněná obsáhlá dokumentace, která se vypracovává v přechodech mezi jednotlivými fázemi.

[2] Obrázek převzat z [2].

3.1.1 Fáze definice problému

Hlavním cílem této fáze je správně pochopit zákazníka. Je důležité s ním mluvit, pochopit, jak mu daný systém bude sloužit, na jakou činnost ho potřebuje a jaké jsou jeho očekávání. Výstupem této fáze je dokument Úvodní studie. Tento dokument by měl obsahovat všechny zjištěné informace od zákazníka, součástí by měl být i přibližný popis požadavků, ale ne jejich přímá specifikace. [2]

3.1.2 Fáze analýzy a specifikace požadavků

V této fázi se zkoumá, jak přesně bude systém fungovat a jaký účel bude plnit. Popis požadavků už by neměl být přibližný, ale pomocní přesné terminologie. Je velmi důležité přesně pochopit účel systému, špatné pochopení zákazníka může totiž vyjít najevo až při předání projektu. Výsledný dokument této fáze se nazývá Specifikace požadavků. Řeší se zde, jak systém bude fungovat a jaký úkol bude plnit, nikoliv jaká bude implementace, ani se zde nehovoří o vybraném programovacím jazyce. [2]

3.1.3 Návrh systému

Zde je hlavním cílem projít specifikaci požadavků a následně navrhnou vhodnou architekturu systému, tedy rozdělení aplikace, vhodné technologie a vývojové prostředí.

Postup návrhu architektury:

1. Určení programovacího jazyka, vývojového prostředí a ostatních technologií,
2. logické rozdělení systému na jednotlivé subsystémy,
3. návrh implementačních modulů a přenesení logického návrhu do fyzické implementační struktury,
4. specifikování práce s daty, jejich uchovávání a formát. Definování chování modulů,
5. upřesnění, diskuse a hodnocení výsledků. [2]

3.1.4 Implementace

Ve fázi implementace probíhá v ideálním případě přepsání jednotlivých specifikací do zdrojového kódu. Vývojový tým by sám neměl vymýšlet nové funkce, nebo cokoliv měnit. Pokud je nutné provést jakoukoliv změnu v návrhu nebo specifikaci, měl by znovu proběhnout návrhový a schvalovací proces. [1]

3.1.5 Integrace a testování

Testovací fáze by měla být prováděna za cílem zjištění správné a zákazníkem očekávané funkčnosti aplikace. Testovací tým by se měl zaměřit na testování základní funkčnosti, dále by měla být otestována každá kombinace vstupních dat a také každý kousek zdrojového kódu. [2]

3.1.6 Provoz a údržba

Aplikace se předává zákazníkovi k praktickému užití. Provádí se oprava chyb, které nebyly zjištěny v dřívějších fázích, ladění a vylepšování aplikace. [3]

3.1.7 Shrnutí

Vodopádový model životního cyklu určitě v době svého vzniku znamenal veliký pokrok ve vývoji softwaru. Vývoj podle tohoto modelu je přímočarý, v každé fázi je přesně dané, co bude následovat. Je ideální pro řízení, protože je vyžadováno, aby po skončení každé fáze byla dokončena úplná dokumentace. Jsou jasně dány všechny dokumenty a snadno lze stanovit harmonogram. Lze jednoduše kontrolovat, jestli je projekt v časové tísni, nebo zda není překročen rozpočet.

Jednoduchost vodopádového modelu je zařazována jako výhoda, ale může být považována také jako nevýhoda, protože model nedokáže pokrýt velké a komplexní projekty. Další nevýhodou je, že do vývoje není možné vstupovat, fáze jsou přesně dané a není je možné nijak měnit, pokud si zákazník rozmyslí nějakou funkcionalitu, celý proces analýzy a návrhu musí proběhnout znovu. Jako další nevýhodu je vhodné uvést, že zákazník, se setká s dodavatelem pouze na začátku a následně na konci, při předávání již hotové aplikace. Zákazník může mít pocit, že se vývojový tým celou dobu „nic nedělá“. Také se mohou změnit požadavky zákazníka a zjistí se, že je vše jinak, než by potřeboval. Může také dojít k situaci, že analytik špatně

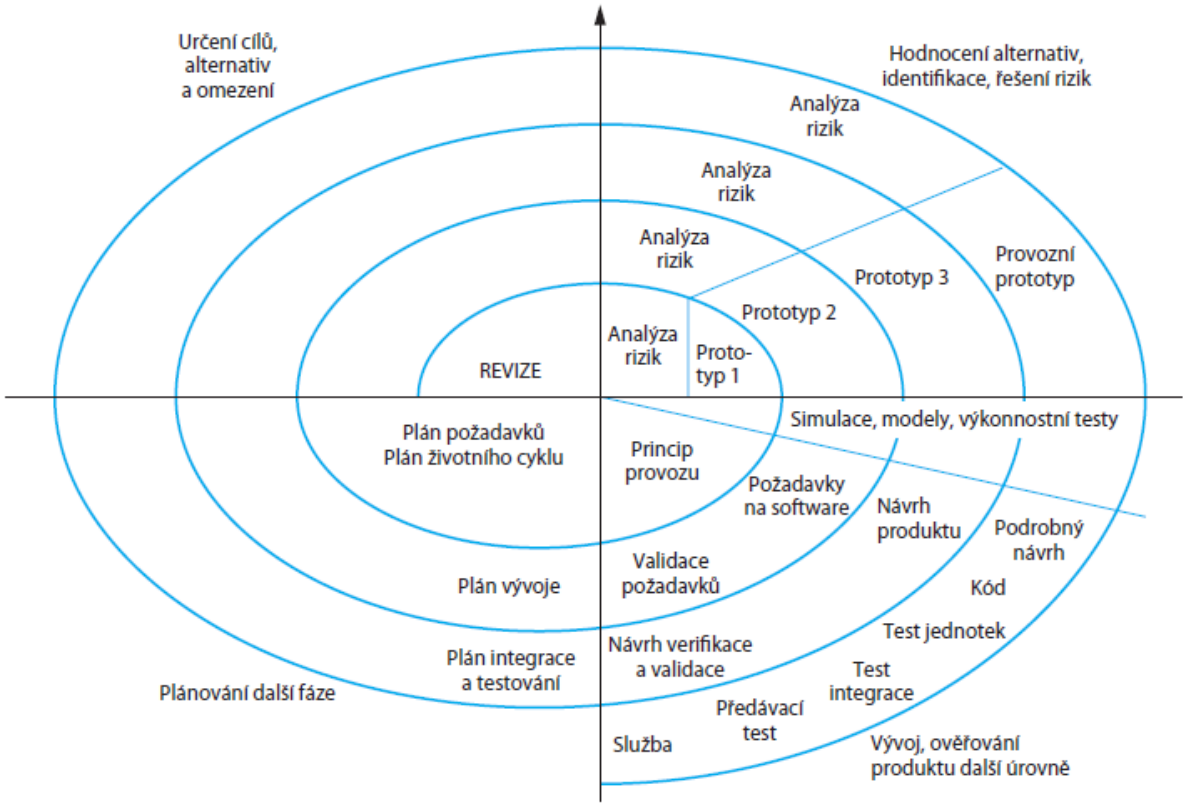
pochopí zadání, třeba i kvůli špatné formulaci zákazníka, a tím pádem vývojový tým pracoval na něčem jiném, než zákazník požadoval.

Vodopádový model životního cyklu je spíše určen pro menší a specifikačně jednodušší projekty, které jsou vyvíjeny v menších týmech či jednotlivcích. V dnešní době se vodopádový model již moc nevyužívá a je určen spíše k výukovým účelům. [1], [2]

3.2 Spirálový model životního cyklu

Krátce po svém vzniku čelil vodopádový model řadě připomínek a poznámek. Nevýhody vodopádu si postupně uvědomila i softwarově-inženýrská komunita. Barry Boehm navrhl v roce 1988 spirálový model životního cyklu. Tento model měl za úkol odstranit některé nedostatky vodopádového modelu, ale přinést i některé novinky jako je iterativní přístup a opakující se analýzu rizik. [3]

Spirálový model patří do skupiny tzv. riziky řízených přístupů, to znamená, že další postup ve fázích vývoje je založen na již zmíněné důsledné analýze rizik. Na základně výsledků této analýzy může být upraven další směr vývoje, postup, či dokonce může být zastaven celý projekt. Vývoj je realizován od středu směrem ven, proto model nese název spirálový. Obecně lze říct, že čím více iterací model má, tím víc bude vývoj nákladnější. [2]



Obrázek 3: Spirálový model životního cyklu[3]

3.2.1 Fáze vodopádového modelu

Spirálový model je rozdělen do čtyř sektorů:

- **Nastavení cílů** – (levý horní) Jsou určeny cíle pro následující iteraci, také jsou identifikována rizika a omezení. [2]
- **Hodnocení a omezení rizik** – (pravý horní) U každého zvoleného rizika projektu se provádí důsledná analýza a kroky k jeho omezení. [2]
- **Vývoj a ověřování** – (pravý dolní) Vývoj produktu a kontrola očekáváných výsledků, testování. [2]
- **Plánování** – (levý dolní) Probíhá rozhodování o pokračování do další smyčky, následně se plánuje postup pro další iteraci. [2]

[3] Obrázek převzat z [3].

Spirála může obsahovat tolik cyklů, kolik je potřeba, tím je ale prodlužováno dodání aplikaci a také prodražen vývoj. V základu je ale spirála rozdělena na čtyři cykly:

- **První cyklus** – Úkolem toho cyklu je najít rizika ohrožující vývoj. Po úspěšném nalezení všech rizik se následně zpracovává základní koncept vývoje a rozhoduje se o použitých metodách.
- **Druhý cyklus** – Zde probíhá specifikování požadavků.
- **Třetí cyklus** – Cílem je vytvořit návrh systému.
- **Čtvrtý cyklus** – Tento cyklus má za úkol implementování, testování a nasazení systému.

Všechny zmíněné cykly obsahují povinné fáze určení cílů, alternativ, plánování, důslednou analýzu rizik a omezujících podmínek. [2]

3.2.2 Dokumentace

Po vytvoření spirálový model neobsahoval formální náležitosti ani dokumenty. Toto bylo vnímáno jako nedostatek, proto v roce 1996 Boehm vydal tři mezníky, které měly „*udržovat přehled o provedeném vývoji, o aktuálních parametrech produktu a prototypů a o požadavcích a jejich změnách*.“ [2]

Mezníky:

- Cíle životních cyklu (Life Cycle Objectives, LCO).
- Architektura životního cyklu (Life Cycle Architecture, LCA).
- Počáteční funkční vlastnosti (Initial Operational Capability, IOC).

Součástí těchto mezníků může být také dokument či prototyp, na kterém spolupracuje zákazník. Standardně bývají mezníky od sebe odděleny jednou iterací (obrátkou v cyklu), ale není to pravidlem. Počet oddělujících cyklů se může měnit v závislosti na potřebách projektu, dále je také možné sloučení jednotlivých mezníků. [2]

3.2.3 Shrnutí

Velkou výhodou tohoto modelu je, že vývojový tým dokáže včas vyloučit nevhodná řešení, a to především díky důsledné analýze rizik. Díky prakticky neomezenému počtu iterací je vhodný pro velké a rozsáhlé projekty. Jako nevýhodu je možné brát, že změny požadavků jsou možné pouze po dokončení iterace, tento fakt může také celý projekt prodloužit a zdražit. Další nevýhodou bezpochyby je, že celý systém je zákazníkovi dodán až po dokončení poslední iterace. Pro malé projekty spirálový model nemá žádný přínos oproti vodopádovému modelu.

Pro moderní aplikace dnešní doby již není spirálový model dostačující, v době svého vzniku byl však tento model velikým přínosem pro softwarové inženýrství. Spirálový model byl inspirací pro mnohé další metodiky a modely. [3], [2]

3.3 Metodika Rational Unified Process

Metodika Rational Unified Process (RUP) je objektově orientovaná iterativní metodika a bývá zařazována mezi klasické metodiky, později však byla doplňována o agilní přístupy. Byla vyvinuta firmou Rational Software a jedná se o komerční produkt. RUP metodika patří do skupiny přístupů řízených případy (use-case-driven approach). Základním prvkem je chápán případ užití – „*posloupnost akcí prováděných systémem či uvnitř systému, která poskytuje určitou hodnotu uživateli systému.“* [2] Metodika RUP je rozdělena do čtyř fází vývoje, přičemž každá fáze je rozdělena do iterací, jejichž počet může být přizpůsoben velikosti a potřebám projektu. [2]

RUP popisuje šest postupů, které jsou optimální pro vývoj systémů.

- **Iterativní vývoj**
- **Správa požadavků** – Doporučení dokumentovat požadavky zákazníků a jejich změny. Dále zkoumat dopady nových požadavků.
- **Použití komponentové architektury** – Je vhodné architekturu systému rozdělit do jednotlivých komponent, to může zajistit efektivnější vývoj.

- **Vizuální modelování softwaru** – Slouží k jednodušší komunikaci uvnitř týmu a k lepšímu porozumění systému za použití modelovacího jazyka UML.
- **Průběžné ověřování kvality** – Odstraňování možných chyb a nedostatků v průběhu vývoje.
- **Řízení změn** – Časté změny ve vývoji mohou vést k chaosu, proto je dobré tyto změny přehledně spravovat ve vhodném systému. [3]

3.3.1 Základní elementy RUP

RUP definuje čtyři hlavní prvky, na kterých stojí celý vývoj softwaru v této metodice.

- **Pracovníci** – „*Pracovník určuje odpovědnost skupiny nebo jednotlivce a jejich chování.“* [4] Pracovník není vyloženě fyzická osoba, ale jedná se spíš o roli, která může být přiřazována v průběhu projektu různým osobám. Chování tohoto elementu je popsáno pomocí činností a odpovědnost pomocí vztahů s meziprodukty. Pracovník je element, který odpovídá na otázku KDO? Možné příklady pracovníků: tester, analytik. [4]
- **Činnosti** – Tyto elementy odpovídají na otázku JAK? Jedná se o jednotku práce. Každá činnost má přesně definované kroky vedoucí k úspěšnému dokončení, bývá vykonávána skupinou či jednotlivcem a má přesně definovaný cíl, tím je vytvořit nebo upravit meziprodukt (třídu, model, plán). Činnosti se dále dělí na tři jednotlivé kroky:
  - úvahy,
  - provádění,
  - přezkoumání. [4]
- **Meziprodukty** – Odpověď na otázku CO? Jedná se o část informace, která je použita nebo zpracována v rámci procesu. Meziprodukty se považují jako výsledky projektů. Používají se jako vstup či výstupy z činností. Příklady meziproduktů:
  - zdrojový kód,
  - spustitelná aplikace,
  - element modulu (třída). [4]

RUP obsahuje velké množství meziproduktů, proto se rozděluje do skupin. Některé příklady skupin meziproduktů:

- meziprodukty týkající se testování,
- meziproduktu týkající se implementace,
- meziprodukty týkající se nasazení produktu a mnohé další. [4]

- **Pracovní procesy –** Posloupnost činností, která vede ke splnění cíle. Tento element odpovídá na otázku KDY? Pracovní proces bývá tvořen obvykle deseti činnostmi, které odpovídají skupinám uvedených v meziproduktech. [4]

3.3.2 Fáze

Model RUP je rozdělen na čtyři fáze.

- **Zahájení –** Cílem této fáze je, aby se všichni účastníci (zákazník, vývojáři, uživatelé) shodli, jak má daný projekt vypadat, určit náklady a základní rizika. Dále je nutné se dohodnout, jestli je možné za daných podmínek (rozpočet, dostupné technologie) projekt zrealizovat. [3], [4]
- **Projektování –** Ve fázi projektování se navrhuje architektura, provádí se částečná analýza rizik a vytváří se prototyp. [3], [4]
- **Realizace –** Provádí se realizace projektu, v této fázi také probíhá paralelní vývoj komponent a jejich následná paralelní integrace do systému. Dále se produkt průběžně testuje. Na konci této fázi by měl být projekt správně otestovaný a připravený k předání zákazníkovi. [3], [4]
- **Zavedení –** Produkt je předán zákazníkovi. Provádí se školení uživatelů, kteří budou se systémem pracovat. Dále se počítá s údržbou systému a podporou pro uživatele. Fáze zavedení může být velice krátká, ale také velice zdlouhavá a nákladná, záleží totiž na požadavcích zákazníka a jeho spokojenosti s výsledným produktem. [3], [4]
- Uvnitř každé fáze se může provádět více iterací, to však závisí na velikost a náročnosti projektu. [3], [4]

3.3.3 Shrnutí

Model RUP je vhodný pro veliké spektrum projektů, nejvíce se však hodí pro velké a náročné projekty, protože nastudování a také zavedení této metodiky stojí poměrně dost času. Velikou výhodou modelu RUP je iterativní vývoj, protože je možné dřív odhalit chyby nebo lépe reagovat na změny požadavků. Jako výhodu lze uvést také fakt, že výrobce průběžně pracuje na nových verzích tohoto modelu. Rozsáhlost a časová náročnost může být také brána jako nevýhoda, a to pro malé týmy vyvíjející velké projekty. Možnou nevýhodou tohoto modelu je, že je komerční a některé společnosti či studenti nemusejí mít dostatečné finance na pořízení. [2]

4 AGILNÍ METODIKY VÝVOJE SOFTWARU

Na úvod této kapitoly je nutné vysvětlit význam slova agilní. Mnoho lidí, kteří se v životě s tímto slovem nesetkali, si vůbec nedokážou vyložit, co toto slovo znamená. Pod slovem agilní si tedy můžeme představit pojmy jako rychlý, flexibilní, dynamický nebo mít schopnost reagovat na změnu. Přesně takovéto by měly být metodiky, které budou popisovány v této kapitole. Agilní metodiky začaly být tvořeny na přelomu 20. a 21. století jako reakce na tradiční metodiky, které byly velice zdlouhavé, těžké a mnohdy vývoj nebyl zcela efektivní. Je důležité zmínit, že agilní metodiky nejsou přesně daným postupem, jak postupovat při vývoji softwaru. Jedná se o soubor doporučení a rad, jak vyvíjet, vždy je možné si dané metodiky přizpůsobit vlastnímu projektu a jeho potřebám. Agilní metodiky jsou založené na iterativním a inkrementálním vývoji. Kladou veliký důraz na komunikaci ve vývojovém týmu, blízkou komunikaci se zákazníkem a jeho spokojenost a jsou velice schopné reagovat na změny. Hlavní rozdíl mezi agilními a klasickými metodikami je, že klasické metodiky považují funkcionalitu za fixní, čas a zdroje jsou naopak proměnné. U agilních metodik je tomu naopak, čas a zdroje jsou fixní a funkcionalita je proměnná. Pokud dojde k nějakým komplikacím nebo změnám požadavků klienta, klasické metodiky potřebují čas a zdroje navíc. Projekt, který se vyvíjí podle klasických metodik, se vyvíjí jako celek, naopak u agilního přístupu se funkcionality implementují podle priority, po každé implementaci by projekt měl být stále schopný provozu, toto by nemělo ohrozit včasné dodání projektu, byť i s malými nedostatky, které by neměly být tak závažné, vzhledem k určeným prioritám implementace. S postupem času se začaly objevovat další agilní metodiky, proto vznikla nezisková organizace „Agile Alliance“, která se zabývá propagací agilního vývoje. V roce 2001 vznikl také manifest agilních metodik. Autoři manifestu dospěli k tomu, že upřednostňují:

- jednotlivce před procesy a nástroji,
- fungující software před úplnou dokumentací,
- spolupráce se zákazníkem před vyjednáváním smluv,
- reakce na změny před dodržování plánu. [10]

Dále za agilním manifestem stojí 12 principů agilního vývoje:

- Naší nejvyšší prioritou je uspokojit zákazníka prostřednictvím včasného a nepřetržitého doručování cenného softwaru.
- Vítáme měnící se požadavky, dokonce i v pozdních fázích vývoje. Agilní přístupy využívají změny pro vyšší konkurenceschopnost zákazníka.
- Doručujeme software často, od pár týdnů do pár měsíců, dáváme přednost kratšímu časovému období.
- Vývojový tým a zákazník musí během projektu vzájemně spolupracovat každý den.
- Budujeme projekty s motivovanými jednotlivci. Poskytujeme jim prostředí a podporu, kterou potřebují, a důvěřujeme jim, že práci dokončí.
- Nejúčinnější a nejefektivnější metoda předávání informací vývojovému týmu a uvnitř něj je osobní rozhovor.
- Funkční software je primárním měřítkem pokroku.
- Agilní přístup podporuje udržitelný rozvoj. Sponzoři, vývojáři a uživatelé by měli být schopni udržovat stálé tempo po neomezenou dobu.
- Neustálá pozornost na technickou dokonalost a dobrý design podporuje agilní přístup.
- Jednoduchost – umění maximalizovat množství nevykonané práce – je zásadní.
- Nejlepší architektury, požadavky a návrhy vznikají ze samoorganizujících se týmů.
- V pravidelných intervalech tým přemýšlí nad tím, jak být více efektivní a následně podle toho upraví a přizpůsobí své chování. [11] [překlad autor]

Mezi agilní metodiky se řadí například Lean Development, Crystal metodiky, Extrémní programovaní nebo Scrum Development Process. Některé z těchto metodik budou popsány v následujících podkapitolách této práce. [6], [5]

4.1 Extrémní programování

4.1.1 Historie

Metodika Extrémní programování (XP) byla vytvořena Kentem Beckem v roce 1999. Kent Beck měl již dřívější zkušenosti s různými metodikami vývoje a také s vývojem samotným. Před vydáním své metodiky pracoval ve společnosti Chrysler, kde vyvíjený projekt nebyl dlouhou dobu implementován, byl si jistý, že za neúspěchem stál špatný vývojový proces. Právě pro to se to rozhodl změnit. [2]

4.1.2 Charakteristika

Extrémní programování se zařazuje do agilních metodik. Využívá známé postupy a myšlenky, ale dovádí je do extrémů jako například neustálé revize kódu, neustálé testování, jak vývojáři, tak zákazníky nebo velice krátké iterace. Velký důraz je kladen na komunikaci v týmu, do kterého patří také zákazník a dokáže pružně reagovat na změny. Při používání této metodiky může být výsledek velice rychlý a stabilní. [3], [6]

4.1.3 Základní principy XP

Extrémní programování se řídí čtyřmi základními hodnotami.

- **Komunikace** – Komunikace je základní hodnotou, na které si extrémní programování zakládá, správná komunikace totiž může zabránit problémům nebo zpoždění projektu. Problém může například nastat, když se špatně naloží s informací od zákazníka, tedy osoba, která tuto informaci dostala, ji nepředá dále. Je tedy velice důležité o všem informovat ostatní členy týmu, i když zpráva nemusí být zcela pozitivní. [2]
- **Jednoduchost** – Extrémní programování považuje za důležité, aby se vývojový tým zaměřoval na to, co zákazník potřebuje teď a nemyslel na to, co bych mohl potřebovat v budoucnosti, protože to se může změnit. Z tohoto důvodu je podle XP lepší udělat jednoduší věc v současnosti, a pokud bude potřeba, tak v budoucnu za změnu zaplatit trochu více. [2]

- **Zpětná vazba** – Zpětná vazba informuje vývojový tým o tom, v jakém stavu se aktuálně nachází projekt. Informuje také o požadavcích či potřebách zákazníka. Důležitou metodou pro získání zpětné vazby je testování, protože když vše projde důsledným testovacím procesem, je to známka, že vývoj se ubírá správným směrem, v opačném případě to značí, že jsou potřeba opravy či změny. Zpětná vazba může být k dispozici ve velice krátké době, protože samotní programátoři si píšou testy vlastními silami, nebo za delší dobu, kdy se jedná o testy funkcionality, které provádí zákazník. [2]
- **Odvaha** – Odvaha je velice důležitý prvkem v extrémním programování. Vývojový tým musí za každou cenu systém dokončit, i když se zdá, že vývoj dál nemůže pokračovat, nebo v případě, že velká část již vytvořeného projektu nemůže být použita a musí se vše předělat. [2]

4.1.4 Postupy

- „*Kent Beck definoval dvanáct konkrétních postupů, které by měly vést k vytváření kvalitních softwarových produktů pomocní metodiky XP.*“ [2] Těchto 12 postupů se dále rozděluje do tří oblastí.

1. Business praktiky

- **Plánovací hra –** Jedná se o plánovací fázi projektu. Tato fáze je velice důležitá, protože je nutné určit náročnost úkolů, plán vývoje, rozpočet a tak podobně. Na začátku projektu je doporučené vytvořit stručný a jednoduchý plán vývoje, který se vytvoří na základě požadavků sepsaných zákazníkem. Tento plán se následně zpřesňuje s každou další iterací. Požadavky a představy zákazníka se dále přetaví do plánu verze. Podle tohoto plánu se odhadnou náklady na vývoj a harmonogram. Požadavky se uplatňují v jednotlivých iteracích podle priority určenou zákazníkem. [6]
- **Zákazník na pracovišti –** Součástí vývojového týmu by měl být i zákazník nebo uživatel, ten by měl být k dispozici na plný úvazek, aby mohl určovat priority, řešit problémy a pomáhat s testováním. XP uvádí, že takový člověk může být drahý, protože mnohdy musí být k dispozici řadu měsíců, zároveň uvádí, že je důležitější funkční software před prací jedné osoby. [6]

- **Malé verze –** Při vývoji podle XP probíhají iterace ve velice krátké době za cílem dodávat důležité funkce co nejdříve, tím pádem se vývojovému týmů dostává rychlá a kvalitní zpětná vazba. [2]
- **Metafora –** Provádí se definice vize, jak má systém fungovat, jak má vypadat a také jeho popis. Pomáhá všem účastníkům sjednotit pohled na systém a zlepšit celou komunikaci o systému mezi zákazníkem, managementem a vývojovým týmem. [6]

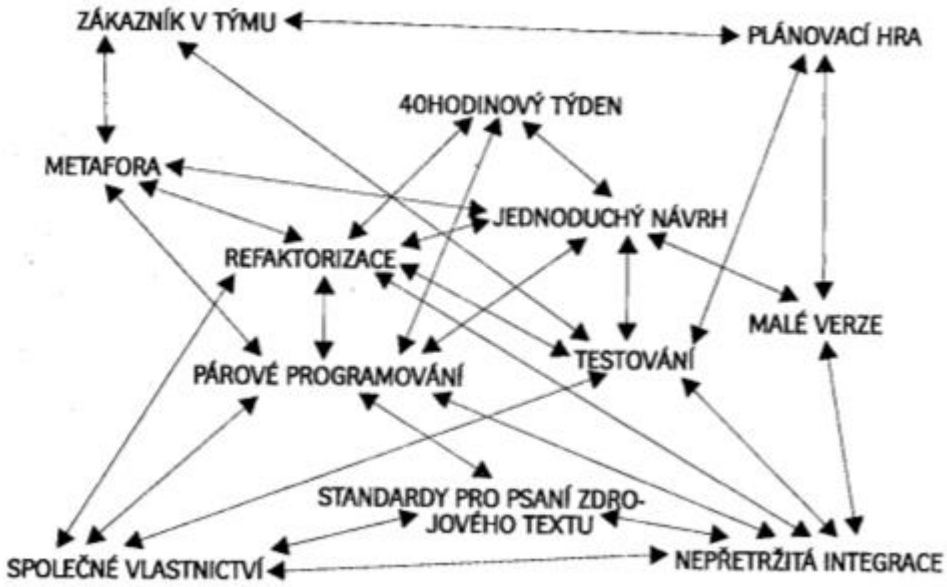
2. Týmové praktiky

- **Párové programování –** Znamená, že dvojice vývojářů sedí u jedné pracovní stanice, jeden z dvojice přemýšlí nad co možná nejlepší implementací, druhý z dvojice přemýšlí nad tím, jestli je implementace správná, jak ji případně zlepšit nebo jaké další testy vytvořit. Klíčová je komunikace v páru. Role si v průběhu dne programátoři střídají, a také se střídají různě páry. Tento způsob dává lepší zpětnou vazbu. [6]
- **Společné vlastnictví –** V Extrémním programování platí, že žádný kus kódu či třída nemá svého majitele, je tomu naopak, každý může měnit či upravovat jakýkoli zdrojový kód kdekoli v systému. Snižuje se tím množství chyb a zvyšuje se kvalita kódu. Tato praktika souvisí s párovým programováním a testováním. [6]
- **Udržitelné tempo –** Obecně ideální pracovní dobou je čtyřicet hodin za týden, tuto dobu se snaží dodržet i metodika extrémní programování, protože čím delší pracovní doba, tím jsou vývojáři unavení a tvoří více chyb. Snahou je, aby se nepracovalo přesčas, ale když je potřeba, tak je dodržováno pravidlo pracovat přesčas maximálně jednou za týden. [6]
- **Standardy psaní kódu –** Jelikož se v extrémním programování užívá společné vlastnictví kódu, je tedy nutné psát kód stejným standardem, aby byl snadno pochopitelný celým týmem. [6]

3. Programovací praktiky

- **Průběžná integrace –** „*Integrace a sestavení se provádí několikrát denně. Každý pár je zodpovědný za integraci svého kódu, kdykoliv se v něm vyskytne významnější změna.*“ [6]
- **Jednoduchý návrh –** Snaha, aby systém byl navržen jako co nejvíc jednoduchý. Jakmile je nalezena nějaká složitost, je snaha o okamžité zjednodušení. [6]

- **Refaktorizace kódu –** Vývojáři se neustále snaží zlepšovat nebo zjednodušovat svůj kód. Může se jednat například o odstranění duplicit. Při všech těchto změnách nesmí být změněna struktura či chování celého systému. [6]
- **Testování –** Testování je důležitým aspektem vývoje při použití metodiky extrémního programování. Testování omezuje produkci chyb a zajišťuje zpětnou vazbu. Programátoři píšou testy až po vytvoření funkčního kódu, další testy tvoří zákazníci z pohledu funkčnosti systému. [6]



Obrázek 4: Vzájemný vztah dvanácti postupů v XP[4]

Aby bylo dodržováno Extrémní programování, je nutné, aby všechno výše zmíněné postupy byly striktně dodržovány.

4.1.5 Role v XP

Extrémní programování nejvíce z agilních metodik pracuje s lidským faktorem, tedy složení týmu, rolí a uspořádáním pracovního prostředí. V XP by tým měl pracovat nejlépe v jedné místnosti, která by měla obsahovat několik počítačů pro párové programování, ale také jakési „kóje“ pro případ, že vývojář potřebuje soukromí. Role v XP jsou následující:

[4] Obrázek převzat z [2].

- **Programátor –** Tato role má za úkol komunikovat s ostatními členy v týmu, dále musí umět programovat, psát testy a refaktorovat kód. V XP je programátor brán jako srdce týmu. [2]
- **Zákazník –** Zákazník je brán jako role týmu a součást vývoje v XP. Spolupracuje s programátory, aby věděli, co je potřeba programovat. Zákazník se musí naučit některé zásady XP, ovlivňovat projekt a spoluzodpovídat za vývoj. [2]
- **Tester –** Role testera je trochu jiná než v ostatních metodikách. V extrémním programování klasické testování zajišťuje programátor, tester tedy pomáhá zákazníkovi s psaním testů funkcionality. [2]
- **Stopař –** Stopař analyzuje zpětnou vazbu a sbírá z ní informace. Má největší přehled o celém projektu, například z jakého důvodu je projekt zpožděn. [2]
- **Kouč –** Kouč pomáhá týmu sledovat proces vývoje a stará se o dodržování zásad extrémního programování. Dalším úkolem je pomáhání při nastalých problémech. Důležitou činností kouče je také udržování funkční komunikace uvnitř týmu. [2]
- **Konzultant –** Většinou se v týmu, který vyvíjí podle zásad extrémního programování, nevyskytují experti, kteří má konkrétní zaměření na určitou oblast, proto je občas nutné najmout takovou osobu, tedy nějakého odborného konzultanta. [2]
- **Velký šéf –** „*Osoba vedoucí organizaci.*“ [2] Pokud tým funguje bez problému, tak nezasahuje. V případě problémů musí umět motivovat a podpořit členy týmu. [2]

4.1.6 Shrnutí

Tato metodika je vhodná spíše po menší vývojové týmy, které mají mezi deseti až dvaceti členy, naopak pro velké společnosti, s velkým počtem vývojářů, je v zásadě nepoužitelná. Mezi agilními metodikami se jedná o možná nejznámější metodiku. Extrémní programování používá již známé a zavedené postupy, které však dovádí do extrémů. Výhody extrémního programování nepochybně plynou z iterativního přístupu a inkrementálního vývoje, tato metodika je schopná rychle reagovat na změny požadavků. Klíčovým principem je jednoduchost. Nevýhodou extrémního programování může být praktická realizace a doba, než si na tuto metodiku celý tým (zákazníci, programátoři, management) zvykne a adaptuje.

4.2 Lean Development

4.2.1 Historie

Další metodikou patřící mezi agilní je Lean Development, jejímž autorem je Robert Charrete. Kořeny této metodiky sahají do 80. let 20. století a nepatří do světa vývoje softwaru. Odvětvím, ze kterého čerpají, je automobilový průmysl v Japonsku, konkrétně firma Toyota. Cílem bylo zvýšit efektivitu a flexibilitu výroby automobilů, protože bylo potřeba uspokojit veliké množství zákazníků. Část těchto byla buď úplně, nebo po nepatrných úpravách převzata k vývoji softwaru. [5]

4.2.2 Charakteristika

Lean Development se snaží identifikovat a eliminovat plýtvání se zdroji v průběhu celého procesu vývoje, tato metodika se snaží vyvíjet kvalitní software za třetinu obvyklého času, použít třetinu rozpočtu a snížit celkové množství chyb na třetinu. Metodika Lean Development není přesným postupem, jak software vyvíjet, je však souborem pravidel a doporučení, podle kterého je možné lépe a efektivně software vyvíjet. [5]

4.2.3 Pravidla

Vývoj v metodice Lean Development se řídí deseti základními pravidly. Mezi těchto deset pravidel patří:

- **Odstranění zbytečností** – Základním pravidlem je odstranit vše, co není nutně důležité pro potřeby projektu. Může se jednat například o dokument, prototyp nebo meziprodukt a je důležité si odpovědět na otázku, zda tento element přinese něco do výsledného produktu. Pokud je odpověď váhavá, nebo zní ne, tak můžeme element považovat za zbytečný. Je nutné si uvědomit, že za vším je nějaká práce a vše stojí nějaké peníze, a pokud dané činnosti či prvky v projektu nepoužitelné, pak se jedná o vyhozené peníze. Příklady druhů plýtvání:
    - výroba více produktů, než je potřeba,
    - čas ztracený čekáním,
    - plýtvání neefektivní prací. [2], [6]

- **Minimalizace zásob** – Toto pravidlo hovoří o tom, že je zbytečné tvořit jakékoliv elementy, které nejsou ihned potřeba. Jedná se především o dokumentaci, která nepřináší hodnoty k výslednému produktu. Je samozřejmě složité určit, co v dokumentaci je zbytečně a může se vynechat, může to záležet na konkrétním projektu nebo vývojovém týmu. [2], [6]
- **Maximalizace toku** – Málokdo si dokáže představit, co maximalizace toku znamená. V klasických metodikách vývoje je používán lineární proces vývoje, tedy nejdříve se vyvíjí jedna část produktu a následně další. Tento způsob může být zdlouhavý a také drahý. Metodika Lean Development to řeší tak, že rozdělí procesy na menší části, tím pádem některé části mohou být vyvíjeny paralelně a také se může zapojit iterativní způsob vývoje. V rámci každé iterace se zpracuje a nasadí určitá část systému, která se následně otestuje. [2], [6]
- **Vývoj je tažen poptávkou** – Požadavky na software se můžou velice rychle měnit. Vývojový tým by se s tím měl smířit a být na to připravený, protože velice pravděpodobně se požadavky dřív nebo později opravdu změní. Proto je dobré provádět rozhodnutí co nejpozději, aby nebyl prováděn vývoj něčeho, co po změnění požadavků bude nepoužitelné. [2], [6]
- **Pracovníci mají pravomoc rozhodovat** – V klasických metodikách vývoje provádí veškerá rozhodnutí management společnosti. V metodice Lean Development je snaha zařadit i „obyčejné“ pracovníky do rozhodování. Pracovníci musí vědět, jakým způsobem přispívají k dokončení vývoje a k finálovému produktu, tím pádem stoupá jejich motivace. Samozřejmě by měla probíhat určitá kontrola od vedení, aby nevznikla moc velká volnost rozhodování. [2], [6]
- **Cílem je uspokojit požadavky zákazníka** – Častým neúspěchem projektů bývá fakt, že proběhla nesprávná specifikace požadavků či nekompletní zadání. Problémem je, že zákazník mnohdy neví, co chce, dokud to nedostane. Klasické metodiky tento problém řeší analýzou a tvorbou specifikace, kde zákazník podpisem souhlasí, že toto chce. Lean Development bere zákazníka jako partnera, který je v kontaktu po celou dobu vývoje, tím se zvyšuje pravděpodobnost reagovat na změny požadavků v samotném vývoji. [2], [6]
- **Zpětná vazba** – Lean Development pracuje s tím, že ne úplně vždy se podaří definovat všechny požadavky na začátku projektu, z tohoto důvodu je zavedena zpětná vazba.

Požadavky jsou doplňovány postupně v průběhu vývoje, to ale může přinést změnu zdrojového kódu nebo architektury. Je dobré si uvědomit, že každá další změna může zavléct do projektu nové chyby. Stejně jako v Extrémním Programování klade Lean Development veliký důraz na testování. Testy se píšou před samotným zdrojovým kódem a testování by mělo probíhat velice často. [2], [6]

- **Odstranění lokálních optimalizací** – Pravidlo číslo osm hovoří o tom, že je zbytečné provádět jakoukoliv optimalizaci, která nemá efekt na výsledný produkt. Může totiž dojít k zpomalení vývoje. [2], [6]
- **Partnerství s dodavateli** – Je důležité mít dobré vztahy s kvalitními dodavateli, kteří mohou dodávat určité části projektu, dobré vztahy a komunikace totiž zajistí podporu spolupracující společnosti. Opačný postoj, tedy odmítavá podpora a odmítaní komunikace, může vést k neúspěchu projektu. Dokonce je doporučeno využívat dodavatele, kteří nabízí hotové komponenty (knihovny či moduly). Lean Development staví na dodávání kvalitního produktu zákazníkovi a je jedno, jak je tohoto výsledku dosaženo. [2], [6]
- **Neustálé zlepšování** – Všichni účastníci projektu, od vedení po vývojáře, by měli být motivovaní se zdokonalovat ve vývoji kvalitního softwaru. [2], [6]

4.2.4 Shrnutí

Metodika Lean Development patří mezi agilní metodiky, tedy využívá iterativního vývoje, dokáže pružně reagovat na změny požadavků, také se snaží rychle a kvalitně dodávat funkční software. Cílem této metodiky je vytvářet software s třetinovým časem, lidskou prací a rozpočtem. Definuje deset základní pravidel podporujících vývoj podle této metodiky. Výhodou je snaha o eliminaci zbytečného, což má za následek větší efektivnost a nižší náklady na vývoj. Jako riziko je možné brát snahu o pozdní rozhodování, které je jedním z deseti pravidel, ale ne vždy se musí vyplatit. Tak jako v případě ostatní agilních metodik se nejedná o přesný postup vývoje softwaru, ale pouze o jakýsi soubor pravidel a doporučení, jakým směrem by se měl vývoj ubírat a co dodržovat. [2]

4.3 Scrum

4.3.1 Historie

Vznik této metodiky se datuje na přelom 20. a 21. století, autory jsou Ken Schwaber, Jeff Sutherland a Mike Beedle. Všichni tito muži měli zkušenosti s vývojem softwaru a práci v týmech, proto chtěli vyvinout metodiku, která bude mít efektivnější proces vývoje a lépe bude reagovat na změny požadavků. Termín Scrum má spojitost s formací, která je používána v rugby a to proto, že celý tým v této formaci funguje jako jeden celek, tedy i celý tým se snaží vyvinout kvalitní a funkční software. [2]

4.3.2 Charakteristika

Scrum je rámec pro agilní vývoj a patří mezi nejpoužívanější a nejoblíbenější. Scrum tým by měl obsahovat zhruba pět až jedenáct členů. Scrum je založen na iterativním a inkrementálním vývoji a je určen pro řízení komplexních prací, například vývoj nového systému či produktu. Vývojový proces se rozděluje na takzvané sprinty (iterace), které bývají dlouhé 30 dní, ale některé týmy si tento čas zkracují, takže je možné říct, že doba jednoho sprintu trvá dva až čtyři týdny. Scrum plní vizi agilního manifestu a snaží se tedy o maximální spolupráci v týmu, časté dodávky a pružné reagování na změnu požadavků. Během vývojového procesu se Scrum snaží rozdělit velké celky na menší činnosti, které mohou být vývojovým týmem dokončeny v kratším čase. [5]

4.3.3 Role

Scrum definuje tři základní role, které se dohromady nazývají jako takzvaný Scrum Team. Mezi základní role patří:

- **Vlastník Produktu (Product Owner)** – Definuje, jaké vlastnosti má produkt mít a co má obsahovat. Stará se o to, aby vývojový tým pracoval během každého sprintu na položkách, které mají vysokou prioritu a smysl. Vlastník má přehled o nevyřízených položkách a zajišťuje, aby vývojový tým měl přehled o prioritě v jednotlivých položkách, také by měl znát trh, zákazníka a obchod. Měl by být snadno dostupný vývojovému týmu, ale i ostatním zúčastněným stranám, se všemi těmito stranami by měl být schopný velice dobře komunikovat. [12]

- **Vývojový tým (Team)** – Vývojový tým by měl být složen zhruba z pěti až jedenácti členů a je samoorganizující se, tedy týmy si samy organizují a řídí práci. Je zodpovědný za vytvoření produktu. Všichni členové by měli navzájem úzce spolupracovat a fungovat opravdu jako jeden tým. [12]
- **Scrum Master** – Pomáhá týmu dodržovat principy metodiky Scrum, aby byl dodáván kvalitní software. Scrum Master není klasický vedoucí, ale spíše podpora týmu, která se stará o odstranění překážek bránícím vývoji. Pomáhá vývojovému týmu lépe komunikovat, organizuje schůzky a motivuje tým. [12]

4.3.4 Artefakty

Scrum používá tři základní artefakty k řízení práce a vývoje, všechny artefakty budou popsány níže.

- **Product Backlog** – Product backlog je seznam požadavků seřazených podle priority a je určený pro vývojový tým. Backlog se neustále vyvíjí a nikdy není úplně hotový. Nejvíce prioritní požadavky či vlastnosti jsou uvedeny na vrcholu tohoto seznamu, to značí, že musí být implementovány co nejdříve. Product backlog tvoří vlastník produktu. [12]
- **Product Backlog** – Product backlog je seznam požadavků seřazených podle priority a je určený pro vývojový tým. Backlog se neustále vyvíjí a nikdy není úplně hotový. Nejvíce prioritní požadavky či vlastnosti jsou uvedeny na vrcholu tohoto seznamu, to značí, že musí být implementovány co nejdříve. Product backlog tvoří vlastník produktu. [12]
- **Sprint Backlog** – Sprint backlog je seznam činností, které je potřeba vykonat v rámci plánovaného sprintu. Vzniká při plánování sprintu a je výběrem nejvíce prioritních položek z product backlogu. Po vytvoření sprint backlogu do něj nemůže přidávat položky nikdo jiný než vývojový tým. [12]
- **Sprint Burndown Chart** – Jedná se o graf, který zobrazuje, kolik zbývá práce v daném sprintu, také pomáhá vývojovému týmu měřit, zda stihne dokončit zadanou práci ve sprintu. Obvykle se aktualizuje denně, po schůzce daily scrum. S postupujícím sprintem se množství práce snižuje a mělo by směřovat k tomu, aby poslední den sprintu bylo vše hotovo. Pokud graf znázorňuje rostoucí práci, jde o známku, že sprint neprobíhá dobře. [12]

4.3.5 Schůzky

Scrum definuje čtyři základní schůzky, ke kterým dochází při každém sprintu.

- **Sprint Planning (**plánování sprintu) – Během této schůzky celý Scrum tým podle priority plánuje práci, která má být splněna během následujícího sprintu. Vlastník produktu popisuje cíl daného sprintu a případně zodpovídá dotazy vývojového týmu. Plánování sprintu by mělo trvat maximálně 8 hodin. Obecně se hovoří o dvou hodinách plánování na každý jeden týden sprintu. [12]
- **Daily Scrum** – Po dobu sprintu by se měl vývojový tým scházet každý den, a to zhruba na patnáct minut, aby proběhla kontrola pokroku v daném sprintu a dále dohodnutí se na dalším postupu. Každý člen vývojového týmu popíše svoji práci, posoudí, zda je na správné cestě k cíli sprintu a v případě potřeby požádá o pomoc. Povinnými účastníky jsou členové vývojového týmu, účast scrum mastera a vlastníka produktu není povinná. [12]
- **Sprint Review** – Tato schůzka se koná na konci každého sprintu a účastní se vývojový tým, vlastník produktu a scrum master a je zaměřená na produkt. Jedná se o kontrolu, co vše bylo úspěšně a neúspěšně provedeno v uplynulém sprintu. Na základě zpětné vazby vlastníka produktu se z product backlogu odstraní vlastnosti, které již byly implementovány. Celá tato schůzka by měla maximálně trvat čtyři hodiny pro měsíční sprint, pro kratší sprinty by měl být čas kratší. Výsledkem je aktualizovaný product backlog, který by měl být výchozím bodem pro plánování dalšího sprintu, které by mělo proběhnout co nejdříve. [12]
- **Sprint Retrospective** – Celý scrum tým diskutuje o tom, co probíhalo správně, a kde je možné se zlepšit. Vytvářejí se plány, jak zlepšit procesy, nástroje a vztahy, které by se měly projevit v dalším sprintu. Sprint restrospective by měl probíhat po sprint review, ale před sprint planning. Časový rámec této schůzky by měl být 45 minut na každý týden sprintu. [12]

4.3.6 Shrnutí

Scrum je rámec pro agilní vývoj softwaru. Je založen na inkrementálním a iterativním způsobu vývoje. Cílem je rychle a kvalitně dodávat software. Iterace se nazývá sprint a jednotlivé sprinty trvají od dvou do čtyř týdnů. Scrum tým se dělí na tři základní role – vlastník produktu, scrum master a vývojový tým. Vývojový proces je postaven na třech základních artefaktech. Product backlogu – základní seznam vlastnosti, které produkt má obsahovat. Sprint backlogu – seznam vlastnosti, které mají být dodány v rámci jednoho konkrétního sprintu. Burdown chart – graf znázorňující zbývající práci ve sprintu. Dále se definují čtyři základní schůzky. Sprint planning – určení nejprioritnější vlastnosti, které mají být implementovány v daném sprintu. Daily scrum – denní schůzka, kde se informuje, jak daný sprint probíhá. Sprint review – koná se na konci sprintu a jedná se o hodnocení sprintu. Sprint retrospective – schůzka, která se zaměřuje na zlepšení, které by se mělo projevit do dalšího sprintu. Scrum je založen na řízení projektu, nejde přímo o postup a návod, jak software vyvíjet. Scrum tým by měl mít zhruba pět až jedenáct členů.

5 BEZPEČNÝ VÝVOJ SOFTWARU

Mnoho vývojářských týmů stále bere bezpečnost jako něco, co přináší překážky a nutí je k přepracování jejich vykonané práce. Software, který není bez zcela bezpečný, však přináší velké riziko pro firmy, které jej využívají, může se jednat například o zranitelnost lehce využitelnou útočníky, tedy snadnou ztrátu citlivých dat či osobních dat zákazníků, což může znamenat ztrátu důvěry firmy, velké finanční újmy a následnou ztrátu pozice na trhu. V moderním vývoji softwaru by měla mít bezpečnost tu nejvyšší prioritu. Při tlaku dodávat software v co nejrychlejším čase, za co nejnižší cenu a mnohdy s nedostatečnou kapacitou bezpečnost není vždy na prvním místě, nebo je řešena až na poslední chvíli. Bezpečnost vývoje by tedy měla být řešena v průběhu všech fází vývoje softwaru. Pokud se bezpečnosti hrozba odhalí v raných fázích vývoje, odstraní se tím riziko, že prostoupí celým projektem a způsobí finanční ztráty. Existuje celá řada metodik a příruček k správnému zabezpečení softwaru ve všech fázích jeho vývoje. Jedná například o metodiky od společnosti Microsoft nebo od organizace The Open Web Application Security Project (OWASP).

5.1 Bezpečnostní rizika

OWASP je nezisková organizace, která pracuje na zlepšení zabezpečení softwaru. Tato organizace definovala deset největších rizik pro webové aplikace, vývojářské firmy by měly být seznámené s tímto dokumentem a snažit se eliminovat tyto rizika ve svých produktech.

1. **Injection –** K chybám může dojít při vsunutí škodlivého kódu například při SQL Injection. Chybná data jsou odeslána jako součást dotazu nebo příkazu, toto může vést ke ztrátě citlivých dat nebo spuštění nežádoucího kódu. [13]
2. **Broken Authentication –** Špatně nebo nedostatečně implementované funkce ověřování mohou pomoct útočníkům získat hesla, klíče nebo tokeny. Další chyby implementace pak například mohou vést k vydávání se za identity zaměstnanců firmy. [13]
3. **Sensitive Data Exposure –** Špatná ochrana citlivých dat, jako například nepoužívání šifrování při přenosu nebo uchování dat, může vést ke snadnému ukradení nebo upravení těchto dat útočníkem. Útočník tato data může použít ke krádežím identity, podvodům s kreditní kartou nebo jiným trestným činům. [13]

4. **XML External Entities** – K ohrožení může dojít při zpracování XML pomocí zastaralých procesorů. Externí entity v XML dokumentech může útočník použít k odhalení tajných interních souborů, skenování portů či spuštění škodlivého kódu. [13]
5. **Broken Access control** – Mnohdy nejsou ověřovány přístupy uživatelů, této chyby může využít útočník, který tak může získat přístup k citlivým datům, uživatelským účtům či provést změnu přístupových práv. [13]
6. **Security misconfigurations** – Rizikem je používání výchozí, chybné nebo neúplné konfigurace zabezpečení. Útočník může hledat nepoužívané stránky, neopravené chyby, nedostatečně zabezpečené soubory k získání přístupu nebo citlivých informací. Všechny operační systémy, aplikace a podobně by měly být včas aktualizovány. [13]
7. **Cross Site Scripting** – Útočník používá webové stránky k odeslání škodlivého kódu. Většinou se jedná o skript, který se spustí v prohlížeči oběti. Prohlížeč nepozná, zda skript není důvěryhodný a spustí ho. Útočník může získat citlivé informace z prohlížeče či přesměrovat uživatele na škodlivé webové stránky. [13]
8. **Insecure Deserialization** – Tato zranitelnost je těžko zjistitelná a zneužitelná. Pokud aplikace přijímá serializované objekty, může mít deserializace objektů z externích zdrojů fatální následky. Útočník může podstrčit škodlivý kód, který se provede po deserializaci. Útočník tím pádem může být schopen provádět DoS útoky nebo obcházet ověřování. [13]
9. **Using Components with known vulnerabilities** – Útočník může využít známé zranitelnosti v komponentách a frameworcích, které jsou v aplikaci používány. Může dojít ke ztrátě závažných dat nebo převzetí serveru, proto je nutné používat vždy aktualizované verze všech komponent a frameworků. [13]
10. **Insufficient logging and monitoring** – Nedostatečné protokolování a monitorování útoků, spojené s chybějící automatickou notifikací umožňuje útočníkům vytrvale provádět útoky, hledat zranitelnosti a krást data bez odhalení. [13]

5.2 Testování softwaru

Testování je proces, který slouží k odstranění chyb nalezených během vývoje aplikací či systémů. Testování by mělo probíhat při celém cyklu vývoje softwaru. I při sebepečlivějším testováním se nemusí podařit opravit všechny chyby, nemusí to ale znamenat, že se na trh vypustí nekvalitní software. Existuje celá řada druhů testů a testovacích metodik. Testy se dělí například podle způsobu provedení, kam patří manuální a automatizované testy, dále se testy dělí podle znalosti o testovaném systému, kam patří testování černé skřínky a testování bílé skříňky. [7]

5.2.1 Testování černé a bíle skříňky

Testování lze rozdělit podle znalosti vnitřní struktury testovaného systému.

- **Testování černé skříňky** – V případě testování černé skříňky nemá tester přístup ke zdrojovému kódu, není tedy přesně jasné, jak systém pracuje s daty. Tento typ testování se zaměřuje na chování softwaru a zahrnuje testování z pohledu koncového uživatele. [7]
- **Testování bíle skříňky** – V tomto případě tester má přístup ke zdrojovému kódu, tedy testování se zaměřuje zejména na zdrojový kód a logiku uvnitř aplikace. Tester musí mít znalost konkrétního programovacího jazyku, ve kterém je aplikace vyvíjena. [7]

Jako příklad testování černé i bílé skříňky je možno uvést:

- **Penetrační testy** – Jedná se o druh testování, při kterém se simuluje útok hackera. Při tomto útoku se ověřuje, jak je na tom testovaná z hlediska bezpečnosti, hledají se zranitelnosti, chyby v zabezpečení nebo dodržování požadavků. Penetrační testy lze provádět ručně či pomocí automatický nástrojů. Mnoho firem využívá externích společností k provádění penetračních testů. [5]

5.2.2 Manuální a automatické testování

Testování se rozděluje podle subjektu, který jej provádí, tedy manuální a automatické.

- **Manuální testování –** Testování, které provádí osoba tomu určená, většinou se jedná o roli testera a nepožívá žádné nástroje a skripty. Účelem ručního testování je

identifikovat chyby, problémy a vady aplikací či systémů. Tester ví, co a jak testuje, a proč zvolil takový postup, dokáže pak lépe předat a vysvětlit výsledky testů osobám, které se v tomto odvětví neorientují, může se jednat o management či zákazníka. Každá nová aplikace musí být před automatizací otestována ručně. Ruční testování bývá časově náročnější než automatické, ale testy mohou být přizpůsobeny na míru testovanému softwaru. [8]

- **Automatické testování –** Tyto testy probíhají pomocí nástrojů a jejich výhodou je rychlost a reprodukovatelnost. Tester se musí naučit zacházet s nástrojem určeným k testování. Automatické testy jsou spíše obecné, tedy nedají se zcela specifikovat zranitelná místa v daném vyvíjeném softwaru. [8]

5.2.3 Dynamické a statické testování

Testování se dá dále rozdělit podle toho, zda je nutné, aby testovaný software byl spuštěn.

- Statické testování – Jedná se o typ testování, které probíhá, aniž by se skutečně spustil kód softwarové aplikace. Testování může probíhat ručně či automaticky pomocí nástrojů. Při statickém testování je snaha nalézt chyby, chyby v kódu a potenciálně škodlivý kód. [7] Příklady statické techniky:
    - Statická analýza – Zkoumání zdrojového kódu, a to z hlediska bezpečnosti, programovacích chyb a syntaxe. Statická analýza může být prováděna manuálně (code review) nebo pomocí automatických nástrojů. [5]
- Dynamické testování – V tomto typu testování je potřeba, aby byl testovaný software spuštěn. Kontroluje se, zda testovaný software funguje v souladu s požadavky. Dynamické testování se používá spíše v pozdějších fázích vývoje, kdy je k dispozici spustitelný prototyp či hotová aplikace. [7]

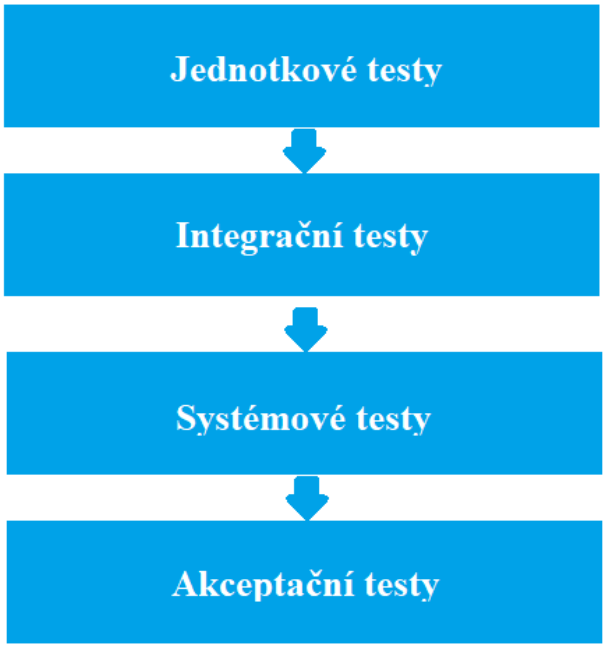
5.2.4 Fáze testování

Testování probíhá během celého životního cyklu vývoje softwaru. Existuje mnoho testů, podle fáze, ve které se zrovna vývoj nachází. Testy provádějící se v různých fázích se rozdělují následovně:

- **Testování jednotek** (Unit testing) – Testují se jednotlivé komponenty zdrojového kódu aplikace, účelem je zjistit, zda dané moduly a jednotky fungují podle očekávání.

Testování provádí sám vývojář. Jednotkou může být například metoda či funkce. Tento typ testování se provádí v brzké fázi vývoje. 0

- **Integrační testování** (Integration testing) – Projekty se většinou skládají z menších částí – modulů, které je po nutné po dokončení spojit dohromady. Účelem tohoto testování je ověřit, zda tyto části společně dohromady fungují. [5]
- **Systémové testování** (System testing) – Tento typ testů probíhá v pozdější fázi vývoje. Aplikace se testuje jako celek a ověřují se všechny vstupy a výstupy dané aplikace a celkově probíhá ověřování aplikace z pohledu zákazníka. [5]
- **Akceptační testování** (Acceptance testing) – Pokud všechny výše zmíněné testy proběhnou bez problému, tak je následně možné aplikaci předat zákazníkovi, ten následně testuje, zda splňuje jeho požadavky a funkce. [5]



Obrázek 5: Typy testů[6]

[6] Zdroj: Autor

5.3 Secure development lifecycle (SDL)

Existuje mnoho metodik ke zvýšení bezpečnosti vývoje softwaru (ochranu osobních údajů, minimalizování chyb v návrhu a kódu, menší zranitelnost aplikací), jedná se například o metodika od neziskové organizace OWASP, metodika od společnosti Cisco a také od společnosti Microsoft. Všechny tyto metodiky se zabývají bezpečností při všech fázích vývoje softwaru a jejich cílem je snížit počet zranitelných míst a hrozeb ve vyvíjeném softwaru. V této podkapitole bude popsán Microsoft SDL, který je rozdělen na pět základních fází a jednu pre-fázi.

5.3.1 Trénink (Training)

Jedná se o jakousi SDL pre-fázi. Každý člen vývojového týmu by měl alespoň jednou za rok absolvovat školení o základech bezpečnosti a aktuálních trendech v zabezpečení a ochraně osobních údajů. Školení totiž může zajistit, že software bude vyvíjen s vyšší mírou bezpečnosti. Dále je doporučeno, aby členové týmu z vlastní iniciativy vyhledávali další školení a vzdělávání, které odpovídají jejich kvalifikaci a znalostem. Zhruba 80 % zaměstnanců, kteří se vývoje zúčastní, by měli mít absolvované školení již před začátkem projektu či v jeho rané fázi. [9]

5.3.2 Požadavky (Requirements)

V prvotní fázi vývoje softwaru probíhá:

- **Identifikace bezpečnostních požadavků** – Identifikace bezpečnostních požadavků ze všech příslušných pokynů a bezpečnostních konceptů, ale také požadavků specifických pro danou aplikaci, může se jednat například o ochranu osobních údajů. Všechny identifikované požadavky musí být zdokumentovány a zahrnuty do systémové specifikace pro danou aplikaci. [9]
- **Brána kvality** (Quality Gate) - Dále by mělo proběhnout definování Brány kvality, což je minimální přijatelná úroveň zabezpečení a soukromí. Díky definování těchto kritérii na začátku vývoje je umožněno lepší pochopení rizik spojených s bezpečnostními problémy a umožňuje týmu lépe identifikovat a následně opravit chyby zabezpečení během vývoje. [9]

- **Posouzení rizik** – Následně by mělo proběhnout posouzení bezpečnostních rizik a rizik soukromí, které identifikují části vyvíjené aplikace vyžadující důkladnou kontrolu. [9]

V této fázi také probíhá výběr vhodné metodiky pro vývoj dané aplikace či systému, výběr provádí tomu určená oddělení (projektová kancelář a jiné IT útvary).

5.3.3 Návrh (Design)

- **Stanovení požadavků na návrh** – Provádí se vytvoření specifikací návrhu zabezpečení, ochrany osobních údajů a vytvoření minimálních kryptografických požadavků. Specifikace návrhu by měly popisovat funkce, které bude používat přímo uživatel, může se jednat například o ověřování přístupu či přístup k citlivým datům. [9]
- **Redukovat zranitelnosti** – Tato činnost úzce souvisí s modelováním hrozeb, ale bere bezpečnostní problémy trochu z jiné perspektivy. Redukování zranitelností znamená nabídnout útočníkovi méně prostoru na provádění jeho útoků, a to například pomocí omezení přístupu k systémovým službám nebo vypnutím zbytečných funkcí. [9]
- **Modelování hrozeb** – Pomocí této činnosti se identifikují potencionální hrozby ve vyvíjeném softwaru, posouzení jejich rizik a rozhodnutí, jak tato rizika odstranit nebo zmírnit jejich dopad. Modelování hrozeb lze provést v jakékoli fázi vývoje, nejvhodnější je však začít co nejdříve. [9]

5.3.4 Implementace (Implementation)

Účelem této fáze je dohlédnout na to, že je bezpečně implantován kód, použity bezpečné a aktualizované nástroje a také vypracovat vhodnou dokumentaci k bezpečnému nasazení a následnému používání vyvíjeného softwaru.

- **Používat schválené a aktualizované nástroje** – Vývojový tým by měl publikovat seznam vybraných nástrojů a jejich bezpečnostních kontrol, tento seznam by měl být následně schválen bezpečnostním poradcem projektu. Dále by měla být vytvořena dokumentace, která bude zaměřena na koncové uživatele a bude pojednávat o bezpečném používání daného softwaru. Dokumentace by měla pojednávat i o bezpečném nasazení dané aplikace, tedy používat nové technologie, bezpečnostní standardy a další. [9]

- **Nepoužívat nezabezpečené funkce** – Nejdříve by měla proběhnout identifikace funkcí a API rozhraní, které mají být při vývoji použity a následně zakázat ty, které byly identifikovány jako nebezpečné. [9]
- **Provádění statické analýzy –** Následně by měla být prováděna statická analýza, která slouží ke kontrole zdrojového kódu, dodržování bezpečnostních zásad psaní kódu a identifikaci chyb zabezpečení jako například SQL Injection, Cross Site Scripting a podobně. Statická analýza může být prováděna pomocí automatických nástrojů či manuálně testerem. Výstupy ze statické analýzy by měly být analyzovány a zjištěné zranitelnosti by měly být evidovány v příslušném systému. Existuje mnoho nástrojů určených pro statickou analýzu, jedná se například o:
  - Code Analysis Tool .NET,
  - FxCop / Visual Studio Code Analysis,
  - Microsoft Source Code Analyzer for SQL Injection,
  - Black Duck software,
  - Check Marks CX SAST,
  - SonarQube. [9]

5.3.5 Verifikace (Verification)

V této fázi se provádí ověřování, zda implementovaný kód splňuje požadavky na bezpečnost a ochranu osobních údajů, které byly definovány v předchozích fázích. Provádí se:

- **Dynamická analýza** – Ověření funkčnosti programu za jeho běhu. Ověřuje se, zda všechny funkce fungují tak, jak byly navrženy. Dále se hledají chyby programu jako problémy s oprávněním uživatelů, pády aplikace a jiné bezpečnostní chyby. [9]
- **Fuzz testování** – Tento typ testování se používá k testu chování a bezpečnostních zranitelností aplikace, když jsou na vstup vkládána neočekávaná nebo poškozená data. Probíhá monitorování vzniklých výjimek, pádu programu nebo paměťových úniků. [9]
- **Penetrační testování** – Provádí se simulace útoku na aplikaci, tak jak by to případně mohl provést hacker. Testují se všechny možné zranitelnosti, přístupy do aplikace a bezpečnostní chyby. Cílem tohoto útoku je prověřit a zhodnotit úroveň zabezpečení aplikace, a nakonec zpracovat shrnující zprávu z penetračního testu. [9]

- **Code Review** – Je provedena manuální code review, tedy zkoumání zdrojového kódu z pohledu bezpečnosti a kvality. Hledají se programátorské chyby a chyby syntaxe, čímž se zvyšuje celková kvalita kódu. [9]
- **Kontrola analýzy hrozeb** – Protože se aplikace může odchylovat od funkčních a návrhových specifikací, které jsou vytvořeny v brzkých fázích vývoje, tak je nutné provést kontrolu analýzy hrozeb, případně její aktualizace, když je aplikace kompletní. Zjišťuje se, zda byly zohledněny všechny změny návrhu či implementace, a to z hlediska bezpečnosti a redukce zranitelností. [9]

Aby byl software schválen, musí být ověřeno, zda byl software testován podle definovaných bezpečnostních požadavků, zda byly odstraněny chyby zjištěné během testování, a jestli jsou implementované bezpečnostní požadavky adekvátní úrovni klasifikace dat.

5.3.6 Vydání (Release)

- **Vytvořit plán zvládání incidentů** – tento plán musí obsahovat: vhodný tým, který bude vždy připraven komunikovat se zákazníkem, plány pro záplaty kódů od jiných vývojových skupin a kontakt na rozhodovací autoritu, která je dostupně 24 hodin denně, sedm dní v týdnu. [9]
- **Provést závěrečnou kontrolu bezpečnosti** – provedení ověření všech bezpečnostních činností a aktivit pro bezpečný vývoj, provádí se tedy zkoumání modelů hrozeb, požadavků a ověření brány kvality. Výsledkem této činnosti jsou 3 výstupy: Úspěšná kontrola, kontrola s výjimkami a kontrola s eskalací – nedodržení požadavků či jiné vážné nedostatky. [9]
- **Certifikace a archivace** – Před uvolněním aplikace do produkce, musí být splněny bezpečnostní požadavky, následně může být aplikace opatřena firemním klíčem (certifikátem), aby byla zajištěna její integrita. Dále musí být archivovány všechny specifikace, zdrojové kódy, modely hrozeb a celá dokumentace k dané aplikaci, aby bylo možné provádět další vývoj či opravy. [9]

5.3.7 Podpora (Response)

Jedná se o post-fázi Microsoft SDL, která se zabývá řešením případných chyb po vydání a nasazení aplikace. Vývojový tým musí být schopný řešit chyby zabezpečení a problémy s ochranou osobních údajů. V této fázi by mělo stále probíhat pravidelné testování aplikace, protože i po nasazení se mohou objevovat nové zranitelnosti jednotlivých komponent. Jelikož vývojový tým může být dost vytížený, tak nemusí dokázat rychle opravit případné nedostatky či chyby aplikace, což může mít fatální následky. K tomuto účelu se využívají Application Management Services (AMS). Využívání AMS znamená, že se podpora, údržba a oprava chyb aplikací předá tomu určenému oddělení či externí společnosti, tím pádem se zvýší spokojenost zákazníku a je umožněno, že vývojový tým se může věnovat jiným projektům. [9]0

6 PRAKTICKÁ ČÁST

6.1 Úvod

V praktické části této bakalářské práce bude popsán primárně proces bezpečného vývoje softwaru ve společnosti Škoda Auto a. s., od prvotního nápadu na vznik aplikace, až po úspěšné nasazení do produkce. Některé kapitoly, které nesouvisejí přímo s bezpečným vývojem, budou popsány stručně, důraz je zde kladen zejména na popsání bezpečnostního testování nikoliv testování provozu. Celý proces bude popsán na vyvíjené aplikaci Škoda Veletrh, která má sloužit jako rezervační systém pro prostory konferenčních místností.

6.2 Tailoring

Jako první v procesu vývoje softwaru probíhá ve Škodě Auto Tailoring, tedy inicializační schůzka mezi byznysem, který přijde s nápadem na realizaci projektu, a příslušnými protistranami ze stran provozu, bezpečnosti, architektury a projektové kanceláře. K realizaci toho projektu byla vybrána agilní metodika vývoje, konkrétně metodika Scrum, vůči tomu jsou přizpůsobeny požadavky a proces bezpečnostního testování aplikace. Při schůzce Tailoring jsou zodpovězeny následující základní otázky:

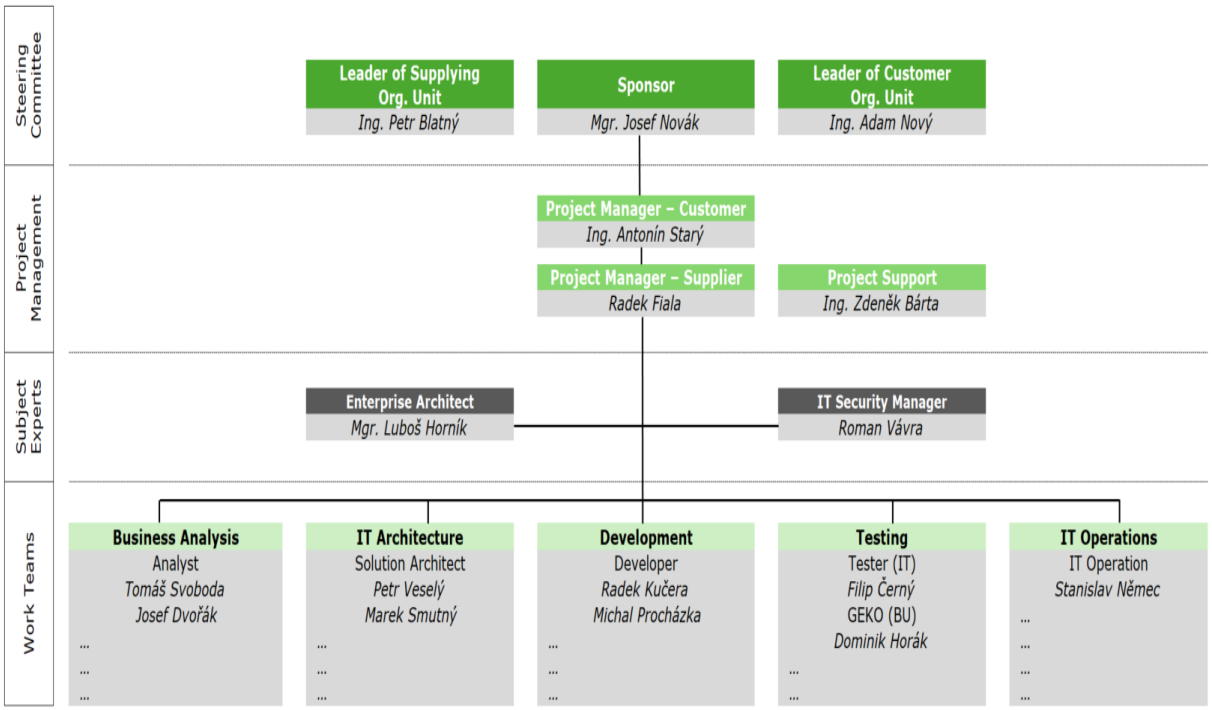
Otázka	**Odpověď**
Business case	
Spadá projekt do některé z následujících kategorií?	Projekty do 100 000 EUR
Financování projektu	
Jaké plánujete náklady na celý projekt?	65 000 EUR
IT Risk and Licence management	
Je součástí projektu nějaký SW, aplikace?	Ano
Je součástí projektu pořízení licencí? Pokud ano, jakých (nákup standardních licencí nebo se jedná o vývoj SW na zakázku)?	Ne, jedná se o vývoj SW na zakázku
Na jakém prostředí bude SW nasazen?	Java

Vznikají realizací projektu nové vazby na jiné aplikace, rozhraní?	SAP – fakturace
Je projekt opatřením, které řeší nějaké IT riziko?	Ne
Jsou vstupy nebo výstupy v projektu důvěrná nebo tajná data?	Ne – pouze interní
User Experience and interface	
Do jaké oblasti projekt spadá?	B2C
Bude řešení nástavbou nějakého stávajícího SW? Pokud ano, jakého.	Ne

Tabulka 1: Odpovědi na otázky v rámci Tailoringu[7]

Po vyplnění odpovědí na předchozí otázky, je v rámci Tailoringu dále nutné vytvořit organizační strukturu osob, které budou pracovat na daném projektu. Organizační struktura projektu Škoda Veletrh je znázorněna na obrázku číslo 6.

[7] Zdroj: [22]



Obrázek 6: Organizační struktura aplikace Škoda Veletrh[8]

[8] Zdroj: Autor

Po úspěšném absolvování schůzky Tailoring začne vývojový tým zpracovávat všechny potřebné dokumenty k projektu. Mezi tyto dokumenty se řadí:

- **Aplication Security Assessment** – Nástroj pro stanovení a dodržování základních bezpečnostních požadavků pro danou aplikaci.
- **Systémová specifikace** – Jde o dokument, který obsahuje popis aplikace, která má být vyvinuta. V systémové specifikaci je popsáno, co aplikace bude dělat a co se od ní očekává, používá se k poskytování důležitých informací všem týmům účastnících se projektu – vývoji, provozu, bezpečnosti.
- **Zápisy z jednání**
- **Servisní manuál**
- **Závěrečná zpráva**

6.3 Application Security Assessment

Application Security Assessment (ASA) je nástroj pro stanovení a dodržování minimálních bezpečnostních požadavků, může být využit již o začátku vývoje softwaru, až do jeho nasazení do produkce. Nástroj ASA je nutnou součástí projektu a používá se, když je dokončena schůzka Tailoring. Pro stanovení minimálních bezpečnostních požadavků aplikace Škoda Veletrh je nutné odpovědět na následující otázky v rámci nástroje ASA:

Otázka	Odpověď
Datum vyplnění ASA:	27. 11. 2020
Název aplikace:	Škoda Veletrh
ID aplikace:	54604
Popis aplikace:	Rezervační systém pro prostory konferenčních místností
Kolika lidmi bude aplikace využívána?	<100
Kde bude aplikace hostována / provozována?	Externě, Azure cloud
Základní architektura:	Webová aplikace
Důvěrnost: (Jak jsou informace zpracovávané aplikací klasifikovány z hlediska důvěrnosti)	Interní
Integrita:	Vysoká

(Jak jsou informace zpracovávané aplikací klasifikovány z hlediska správnosti, neočekávaným změnám)	
Dostupnost: (Jak jsou informace zpracovávané aplikací klasifikovány z hlediska dostupnosti)	Vysoká
Z jakých sítí bude umožněn přístup do aplikace?	Internet, intranet Škoda Auto (admin část)
Které kategorie osobních dat budou aplikací zpracovávána?	soukromé kontaktní a identifikační údaje, údaje týkající se osobních údajů, data o platbách a organizaci času, údaje o spolehlivosti a financích, kontaktní a organizační údaje týkající se zaměstnání

Tabulka 2: Odpověď na otázky v nástroji Application Security Assessment[9]

Následkem odpovědí na poslední otázku bylo, že zpracování dat v aplikaci Škoda Veletrh bylo vyhodnoceno jako důvěrné. Výstupem z nástroje Application Security Assessment je stanovení minimálních bezpečnostních požadavků pro aplikaci Škoda Veletrh. Celý seznam základních bezpečnostních požadavků bude obsažen v Tabulka 3: Seznam nutných základních bezpečnostních požadavků, která obsahuje bezpečnostní požadavek, jeho popis, prioritu (čím menší číslo, tím větší priorita) a komentář, který obsahuje vzájemné slazení návrh byznysu a vývojového týmu na implementaci požadovaných opatření, které je konzultováno s oddělením IT bezpečnosti. Opatření, která musí byznys popsat do ASA, nemusí být popsána do největšího detailu, podrobně se tyto informace popisují v systémové specifikaci, v příslušných kapitolách.

[9] Zdroj: [14]

Bezpečnostní požadavek	Popis	Priorita	Komentář
Silné ověření uživatelů	Jde o ověřování nejméně dvou faktorů jako například PIN nebo heslo, PKI karta, biometrická metoda.	1	Autentizace uživatelů do administrační části je zajištěna prostředky WebSeal tak, že vytvořená junction\tomcat\veletrh má nastavena ACL pouze pro vybrané LDAP skupiny Veletrh.xxx a cesta \tomcat\veletrh\secure je konfigurovaná pouze pro povolení autentizace dvou faktorově (RSA, token) – pouze zde jsou agendy obsahující osobní údaje. Dále je nastaveno, že WebSeal odesílá informaci o přihlášeném uživateli v HTTP hlavičce v atributu IV-USER. Aplikace atribut využívá při volání LDAPWS a získání členství uživatele v odpovídajících skupinách a zajištění autorizace v aplikaci. Externí uživatelé / zákazníci se musí registrovat pomocí mailové adresy standardním registračním formulářem (vše probíhá pomocí šifrované HTTPS komunikace), nebo je možné využít poskytovatele identity třetích stran jako například Google nebo Facebook účet.
Zabránění přístupu k heslům technických uživatelů	Technické uživatelské profily jsou využívány systémy, pro různé systémové akce jako například přihlášení do databáze.	1	Ano, technické účty budou využívány pro automatické operace na serverové části systému. Hesla k technickým účtům budou znát pouze vybraní administrátoři systému. Hesla nebudou nikde uložena v nešifrované podobě.
Používání schválených šifrovacích algoritmů pro ukládání dat do databáze	K šifrování uložených dat je možné použít pouze schválené šifrovací algoritmy.	1	Ano, počítáme s použitím šifrovacích algoritmů, jako například AES 256 – Advanced Encryption Standard, a dalšími algoritmy doporučenými národním institutem pro standardy a technologie.

Používání schválených šifrovacích algoritmu pro ukládání dat mimo databáze	Jedná se o ukládání dat lokálně či v síťových jednotkách. K šifrování uložených dat je možné použít pouze schválené šifrovací algoritmy.	1	Ano, počítáme s použitím šifrovacích algoritmů, jako například AES 256 – Advanced Encryption Standard, a dalšími algoritmy doporučenými národním institutem pro standardy a technologie.
Představení opatření ke zvýšení integrity při ukládání dat	Musí být definována vhodná opatření k zajištění integrity při ukládání dat. Může jednat například o vytváření a ověřování hash hodnot, kontrolu věrohodnosti, automatickou či manuální kontrolu vstupů a výstupů.	1	Ano, počítá se zpětnou kontrolou hashů při práci se soubory, ať už na vstupu či výstupu a to automaticky. Manuálně pouze v případě potřeby ručního zásahu, například při bezpečnostním incidentu, kdy je potřeba ověřit správnou funkci automatických procesů. Jinými slovy budeme v souladu s doporučením tohoto bodu.
Používání schválených šifrovacích algoritmů pro přenos dat	K šifrování přenosu dat lze použít pouze schválené šifrovací algoritmy.	1	Ano, počítáme s použitím šifrovacích algoritmů, jako například AES 256 – Advanced Encryption Standard, a dalšími algoritmy doporučenými národním institutem pro standardy a technologie.
Oddělení externího přístupu k aplikaci	Přístupy k aplikaci z externích sítí musí být odděleny. Oddělovacími komponentami mohou být například Citrix Server či reverzní proxy.	1	Ano, přístup do administrační části je řízen skrze proxy server z intranetu vnitřní sítě Škoda Auto. Přístup pro běžné uživatele / zákazníky je napřímo skrze HTTPS komunikaci.
Pravidelné získávání / vyžádání si informací o chybách aplikace	IT manažer aplikace musí získávat pravidelné informace (obvykle jednou za měsíc, maximálně jednou za tři měsíce) o chybách zabezpečení, které ovlivňují programování a používání SW komponent (např. Knihoven).	1	Vnitřní procesy jsou nastaveny na takzvané automatické notifikace skrze CERT – Computer Emergency Response Team.

Okamžitá oprava kritických IT zranitelností	Pokud má zranitelnost či chyba zásadní dopad na provoz aplikace, je potřeba naplánovat a neprodleně provést její opravu.	1	Okamžitá oprava vychází z procesu CERT. Který zaúkoluje AMS tým, s požadavkem okamžitého implementace bezpečnostních záplat dle příslušné kritičnosti zranitelnosti.
Pravidelně opravovat slabiny	Pokud slabina nemá zásadní dopad na provoz aplikace nebo přidružené aplikace, musí být slabina opravena pomocí aktualizace nebo plánu oprav. Slabina musí být do té doby sledována, aby se zajistilo, že nedojde k žádným dopadům na bezpečnost IT.	2	O pravidelnou údržbu a instalaci aktualizací a bezpečnostních záplat se stará AMS tým.

Dokumentace konceptu rolí a práv	Koncept rolí a práv aplikace by měl zajistit, aby každý uživatel dostal pouze ta práva, která jsou skutečně potřebná pro splnění jeho úkolů. Díky tomu je možné se vyhnout zneužití práv. Do konceptu rolí a práv musí být zahrnuty tyto informace: Popis rolí včetně matice oprávnění, proces přiřazování rolí, proces odvolání rolí, proces testování a nastavení nových rolí.	2	Práva budou granulárně dle následujících rolí: • Administrátor systému • Průvodce • Technik • Management veletrhu • Pracovník recepce • Uživatel
Přiřazování uživatelských oprávnění v principu dvojí kontroly	Uživatelská oprávnění musí být přidělena pomocí souhlasu na principu dvojí kontroly, tj. do procesu přiřazování by měli být zapojeni alespoň dva zaměstnanci (kromě osoby, která práva obdržela). Tento postup má zabránit náhodnému přidělení oprávnění.	1	Přidělení požadované role podléhá vždy principu čtyř očí, žádost o členství schvaluje vedoucí oddělení žadatele a byznys vlastník systému.

Dokumentace autorizačních procesů	Celý proces autorizace (od žádosti o práva na užívání až po vydání práv) musí být zdokumentován. Na základě dokumentace lze prokázat přípustné přiřazení oprávnění. Vlastníkovi informací (nebo jeho zástupci) musí být dána příležitost zobrazit přidělená oprávnění.	2	Detailní proces bude popsán v rámci systémové specifikace.
Omezení přístupu na požadované minimum	Každý uživatel musí mít přístup pouze k informacím a funkcím, které jsou vyžadovány pro výkon jeho úkolů. Role musí být odpovídajícím způsobem definovány a autorizace omezena.	1	Bude aplikován systém omezeného přístupu, takzvaný přístup „pouze tolik, kolik je potřeba“.
Používání osobních uživatelských účtů	Musí být možné přiřadit uživatelský účet v aplikaci jedinečně jedné osobě.	1	Na základě principů fungování LDAP identity je zaručeno, že do administrativní části se dostane vždy jeden člověk s jednou identitou. U zákazníků / běžných uživatelů toto není možné zajistit, pokud se zaregistrují pod jiným emailem / účtem.

Zabránění přístupu k aplikaci prostřednictvím výchozích nebo testovacích účtů	Některé aplikace jsou dodávány s výchozím účtem. Může se také stát, že budou vytvořeny testovací účty pro testování nových uživatelských rolí. Tyto účty nesmí být přiřazeny jednoznačně jedné osobě. Výchozí účty musí být vždy odstraněny, deaktivovány nebo blokovány, totéž platí pro testovací účty.	1	Do produkčního prostředí se tyto účty nesmí dostat, správnou implementaci a odstranění těchto účtu potvrdí penetrační testování a code review.
Používání procesu kontroly a odebírání práv	Musí být dodržováno minimální přidělení oprávnění, proto je nutné kontrolovat, zda uživatelé ke své činnosti stále potřebují přidělená práva. Je-li to nutné, pak je možné uživatelům práva odebrat či je omezit.	2	V rámci vnitřních systémů administrační části se přístup přiděluje na jeden rok, čímž dochází k automatické sanitaci prostředí. Systém automaticky měsíc předem upozorní na blížící se vypršení přístupů a uživatel musí znovu zažádat o prodloužení platnosti oprávnění. Běžní uživatelé / zákazníci jsou v případě neaktivity delší, jak jeden rok automaticky odstraněni z databáze. Účty pracovníků Škoda Auto jsou automaticky mazána při jejich odchodu ze společnosti nebo změně pracovního místa.
Zaznamenávání přihlášení, odhlášení uživatelů	Data by měla jasně ukazovat, kdy se uživatel do aplikace přihlásil. Proces přihlášení a odhlášení nemusí nutně monitorovat samotná aplikace, ale mohou to zajišťovat jiné komponenty.	1	Aplikace je napojena na logovací systém Škoda Auto, veškeré požadované logy jsou dle klasifikace dat odeslány do centrálního logovacího systému dle rozsahu, který udávají interní normy.

Zaznamenávání akcí spojených se správou uživatelů	Aby byl zajištěn sledovatelný návrh konceptu rolí a práv, musí být zaznamenány všechny akce, které se týkají správy uživatelů v rámci aplikace. V protokolu musí být zaznamenáno, který správce aplikace vytvořil uživatele nebo mu přidělil oprávnění a kdy.	1	Systém je napojený na logovací systém Škoda Auto, veškeré požadované logy jsou dle klasifikace dat odeslány do centrálního logovacího systému, dle rozsahu, který udávají interní normy.
Zaznamenávání přístupu pro zápis dat (přidávání, změna, mazání)	Pokud existují nějaké požadavky (právní, provozní či technické), které vyžadují, aby bylo provedeno zaznamenávání přístupu pro zápis dat ze strany uživatelů, tak tomu musí být učiněno v souladu se specifikacemi.	1	Systém je napojený na logovací systém Škoda Auto, veškeré požadované logy jsou dle klasifikace dat odeslány do centrálního logovacího systému, dle rozsahu, který udávají interní normy.
Definování incidentů souvisejících s IT bezpečnostní v souvislosti s aktivitami uživatelů	Data lze plně vyhodnotit, pouze pokud jsou známa kontrolní kritéria. Události související s bezpečností IT jsou pro každou aplikaci individuální a musí být identifikovány odpovědnými technickými osobami. Mohou být relevantní například následující události: - Časté neúspěšné pokusy uživatele o přihlášení - Přihlášení v neobvyklých časech - Neoprávněné nastavení uživatelů - Neschválené změny konfigurace	2	Systém je napojený na logovací systém Škoda Auto, veškeré požadované logy jsou dle klasifikace dat odeslány do centrálního logovacího systému, dle rozsahu který, udávají interní normy. Z těchto logů poté Security Operation Centre (SOC) vytváří v případě porušení přednastavených pravidel bezpečnostní incidenty.

Provádění pravidelných anonymních analýz dat za účelem kontroly, zda nedošlo k incidentům souvisejících s IT bezpečností	Data musí být pravidelně analyzována, aby se zkontrolovalo, zda došlo k incidentům souvisejícím s bezpečností IT. Pokud je zjištěn incident, je nutné o tom informovat příslušné kontaktní partnery a incident vyjasnit.	1	Systém je napojený na logovací systém Škoda Auto, veškeré požadované logy jsou dle klasifikace dat odeslány do centrálního logovacího systému, dle rozsahu který, udávají interní normy. Z těchto logů poté Security Operation Centre (SOC) vytváří v případě porušení přednastavených pravidel bezpečnostní incidenty.
Vytvoření konceptu logování	Koncept logování definuje organizační a technické aspekty procesu logování včetně: definice incidentů, které jsou relevantní pro bezpečnost IT, definice kontaktů, které musí být informovány, když dojde k příslušným incidentům, definice rolí, které mají přístup k datům protokolování, včetně definice úrovně autorizace (čtení, zápis)	2	Systém je napojený na logovací systém Škoda Auto, veškeré požadované logy jsou dle klasifikace dat odeslány do centrálního logovacího systému dle rozsahu, který udávají interní normy.
Zajištění filtrování vstupů	Aby se předešlo ztrátám dat v aplikaci, nebo ve zpracovaných datech, je nutné před předáním uživatelských vstupů na rozhraní zkontrolovat povolené hodnoty.	1	Při vývoji budeme vycházet z doporučení OWASP Top Ten.

Zajištění filtrování výstupů	Chybové zprávy aplikace by měly být co nejobecnější, aby z nich nebylo možné vyvodit závěry o podrobnostech interní implementace. Například žádné vypisování chyb z databáze.	1	Při vývoji budeme vycházet z doporučení OWASP Top Ten.
Provedení kontroly zabezpečení aplikace pomocí penetračních testů a analýzy zdrojového kódu	Kontrola zabezpečení aplikace musí být provedena odpovědnou osobou zabezpečení IT nebo společností tomu pověřenou. Kontrola probíhá za pomocí penetračních testů a analýzy zdrojového kódu aplikace.	1	Ano, v rámci uvedené risk class budou provedeny penetrační testy a code review (tyto testy a jejich průběh bude popsán dále v této bakalářské práci).
Odstranění slabých míst nalezených v kontrole zabezpečení aplikace	Zranitelnosti odhalené během kontroly zabezpečení aplikace musí být opraveny důsledně. Zranitelnosti s vysokou prioritou je třeba rychle odstranit.	1	Vychází se z doporučení, která budou popsána v závěrečné zprávě z testování (Doporučení k opravě).
Opětovná kontrola zabezpečení aplikace pomocí penetračních testů a analýzy zdrojového kódu	Musí být ověřena náprava slabých míst nalezených v aplikaci, a to opětovným penetračním testováním a analýzou zdrojového kódu.	1	V případě nálezů bude prováděno opětovné testování.

Tabulka 3: Seznam nutných základních bezpečnostních požadavků[10]

[10] Zdroj: [15], [21]

6.4 Systémová specifikace

V této podkapitole bude popsána struktura systémové specifikace, tedy dokumentu, který je nutnou součástí projektu. Systémová specifikace se začne vytvářet po úspěšné schůzce Tailoring. Jedná se o dokument, který obsahuje popis aplikace, která má být vyvinuta. V systémové specifikaci je popsáno, co aplikace bude dělat a co se od ní očekává, používá se k poskytování důležitých informací všem týmům účastnících se projektu – vývoji, provozu, bezpečnosti a architektury. Systémová specifikace se doplňuje postupně, tak jak probíhá a postupuje agilní vývoj.

Struktura systémové specifikace [16]:

- **Historie dokumentu** – Zde se zapisuje, v jakém stavu a verzi se projekt a nachází k jakému datu. Popisují se zde i změny, které proběhly.
- **Cíle projektu** – V této části je popsáno, co je cílem tohoto projektu – jaká aplikace či systém mají být vyvinuty. Součástí je i popis cílového stavu, tedy jakou funkci má aplikace plnit, když bude nasazena. Dále se zde nachází harmonogram akcí, jak přesně po sobě mají jít, jedná se například o revizi systémové specifikace, vývoj verze 1.0 a testy a pilotní provoz aplikace.
- **Business model**
- **Procesní integrace**
- **Model návrhu a analýzy – Případy užití** – V této části jsou řešeny jednotlivé případy užití, tedy jaké činnosti bude provádět administrátor, zákazník, pracovník, recepční a další pozice v rámci aplikace. Dále je zde popsáno, jakým způsobem bude zákazník vyplňovat objednávku, jak ji administrátor bude schvalovat a další případy. Případy užití jsou znázorněny pomocí Use Case diagramů.
- **Vývoj aplikace** – V této podkapitole jsou definovány technologie, které budou používány v projektu. Může se jednat o programovací jazyk, ve kterém se bude aplikace vyvíjet, vývojové prostředí či databáze.
- **Model nasazení** – Zde je popsána logická a fyzická struktura.
- **Bezpečnost** – V kapitole bezpečnosti je popsáno, jakým způsobem je klasifikováno zpracování dat (důvěrné, tajné…), také jestli se na aplikaci vztahují nějaké zákonné či

jiné interní nařízení a předpisy. Dále se zde řeší požadavky na logování, jsou zde popsány typy uživatelů (anonymní, autentizovaní) a uživatelské role a oprávnění.

- **Nefunkční požadavky** – V této části je definováno, kolik zhruba uživatelů bude v jeden čas aplikaci používat, stanoví se nutná přibližná odezva a webové prohlížeče, které budou podporovány na různých operačních systémech. Dále je zde popsáno, jakým způsobem bude probíhat zálohování dat a jak často.
- **Akceptace projektu** – V této kapitole se stanoví akceptační kritéria, pravidla akceptace – za jakých pravidel bude projekt akceptován zákazníkem po splnění akceptačních testů.
- **Plán testů** – V této kapitole je sepsán plán všech testů, které budou prováděny na projektu. Jedná se například o funkční testy, zátěžové testy, integrační testy, testy bezpečnosti a uživatelské akceptační testy.
- **Nasazení řešení** – Popisuje se, jak dlouho bude potřeba pilotní provoz a kde bude prováděn, například jakými odděleními a podobně. Dále jsou stanoveny požadavky na předání aplikace do Application Management Support (AMS). Požadavky se týkají například od kdy, do kdy musí být aplikace provozována, jaká je požadovaná dostupnost, kdy jsou okna pro odstávky a další.
- **Školení** – V kapitole Školení je uvedeno, jaká oddělení budou zodpovědná za provádění školení k dané aplikaci.
- **Provoz a servis** – Jakým způsobem bude prováděn provoz a servis aplikace. Může se jednat o podporu, která má více úrovní – základní telefonická, pokročilejší a expertní.
- **Cena a náklady** – V kapitole jsou popsány investiční náklady na aplikace a náklady na její následný provoz.
- **Licence** – Je řešeno, zda jsou potřeba nové serverové licence, a také jaké licence bude uživatel potřebovat pro využívání aplikace.
- **Další postup** – Zde jsou definovány úkoly pro dodavatele a zadavatele. Úkolem pro dodavatele může například být nutnost implementace aplikace dle harmonogramu a úkolem pro zadavatele může být provedení akceptačních testů.

6.5 Zahájení vývoje

Došlo k dokončení všech nutných přípravných fází tohoto projektu, takže je možné začít samotný vývoj aplikace. Jak již bylo zmíněno, tak aplikace se bude vyvíjet agilně, v jazyce Java. Na vývoj této aplikace dohlíží Java Competence centre, které se stará o dodržování moderních standardů při vývoji v jazyce Java. Délka jednotlivých sprintů byla stanovena na čtrnáct dní. Systémem spravující vývojový proces (sestavení, nasazení, testování) je Microsoft Team Foundation Server (TFS). Vytvoří se potřebný projekt v TFS, nastaví se oprávnění pro vývojový tým a veškeré zdrojové kódy jsou ukládány do Škoda Auto repozitáře (GIT) a veškeré další návazné operace v rámci vývojového procesu (sestavení aplikaci, nasazení do požadovaného prostředí) probíhají taktéž skrze TFS. V každém Scrum sprintu je provedená určitá práce, která je dohodnuta na schůzce plánování sprintu. Po každém sprintu je provedena statická analýza, která má za úkol odhalit bezpečnostní chyby a nedostatku v kódu. Statická analýza bude prováděna pomocí nástrojů:

- SonarQube – Nástroj pro automatické provádění statické analýzy zdrojového kódu za účelem detekce nedostatků a chyb v zabezpečení. Nástroj umí pracovat s více než 20 programovacími jazyky jako například Java, C#, C.
- Black Duck – Nástroj, který pomáhá identifikovat rizika zabezpečení, kvality a dodržování licencí ve zdrojovém kódu.
- CheckMarks CX SAST – Nástroj určený k provádění statické analýzy, sloužící k identifikaci bezpečnostních chyb ve zdrojovém kódu. Tento nástroj identifikuje zranitelná místa a poskytuje užitečné informace k jejich nápravě. Podporuje více než 20 programovacích jazyků.

V pozdějších fázích vývoje se provádí manuální code review a penetrační testy.

Při vývoji si aplikace projde následujícími prostředími:

- Vývojové prostředí,
- testovací prostředí,
- prostředí kvality – zdrojový kód je 1:1 s tím, který jde do produkce. Toto prostředí slouží pro finální testování aplikace, která být nasazena do produkce,
- produkce.

V každém z těchto prostředí může být provedeno tolik iterací, kolik jich je potřeba. Pro lepší představu a pochopení procesu bude v této bakalářské práci popsána vždy úvodní a závěrečná iterace v každém prostředí.

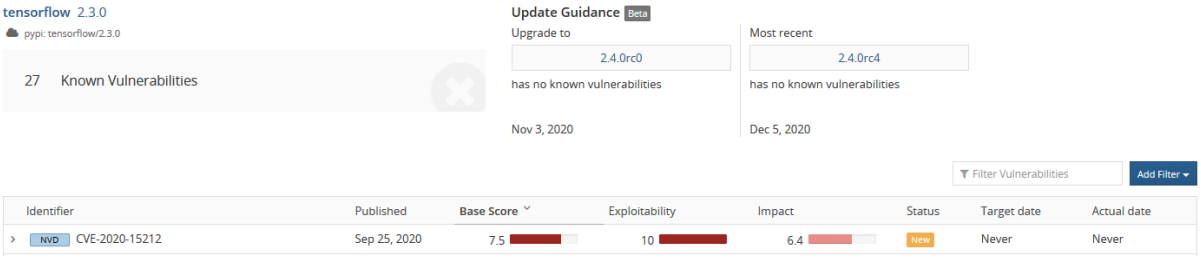
Všechny Scrum artefakty a schůzky (viz oddíly 4.3.4 a 4.3.5) jsou prováděny a řešeny vývojovým týmem, budou tedy popsány jen stručně, tato bakalářská práce se v praktické části zaměřuje zejména na popsání bezpečného vývoje aplikace.

6.5.1 První iterace – Vývojové prostředí

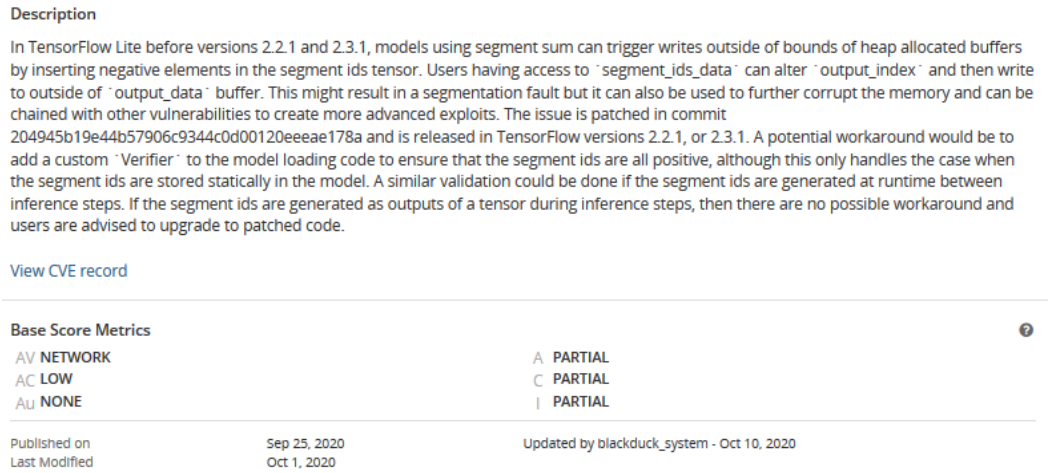
Vývojový tým vytvoří product backlog, ve kterém jsou popsány veškeré funkce, které mají být v aplikaci Škoda Veletrh obsaženy, tyto funkce jsou seřazeny podle priority, tak jak budou postupně implementovány. Následně je svolána schůzka plánování sprintu, kde se vytvoří sprint backlog, tedy seznam funkcí s největší prioritou, které budou implementovány v následujícím sprintu. Následně započne samotný vývoj v rámci sprintu, který trvá 14 dní.

Vývojový tým po konci sprintu odevzdá všechny zdrojové kódy do repozitáře GIT. Nad TFS CI/CD pipeline se provede sestavení aplikace, paralelně se sestavením aplikace se zdrojové kódy odešlou do nástrojů statické analýzy SonarCube, Black Duck HUB, CheckMarx CX SAST. Tyto nástroje automaticky vygenerují reporty o stavu aplikace. Vývojový tým se spojí s oddělením IT bezpečnosti a vyhodnotí se reporty. Výstupy z jednotlivých nástrojů statické analýzy jsou vyobrazeny níže.

Reporty z nástroje Black Duck:

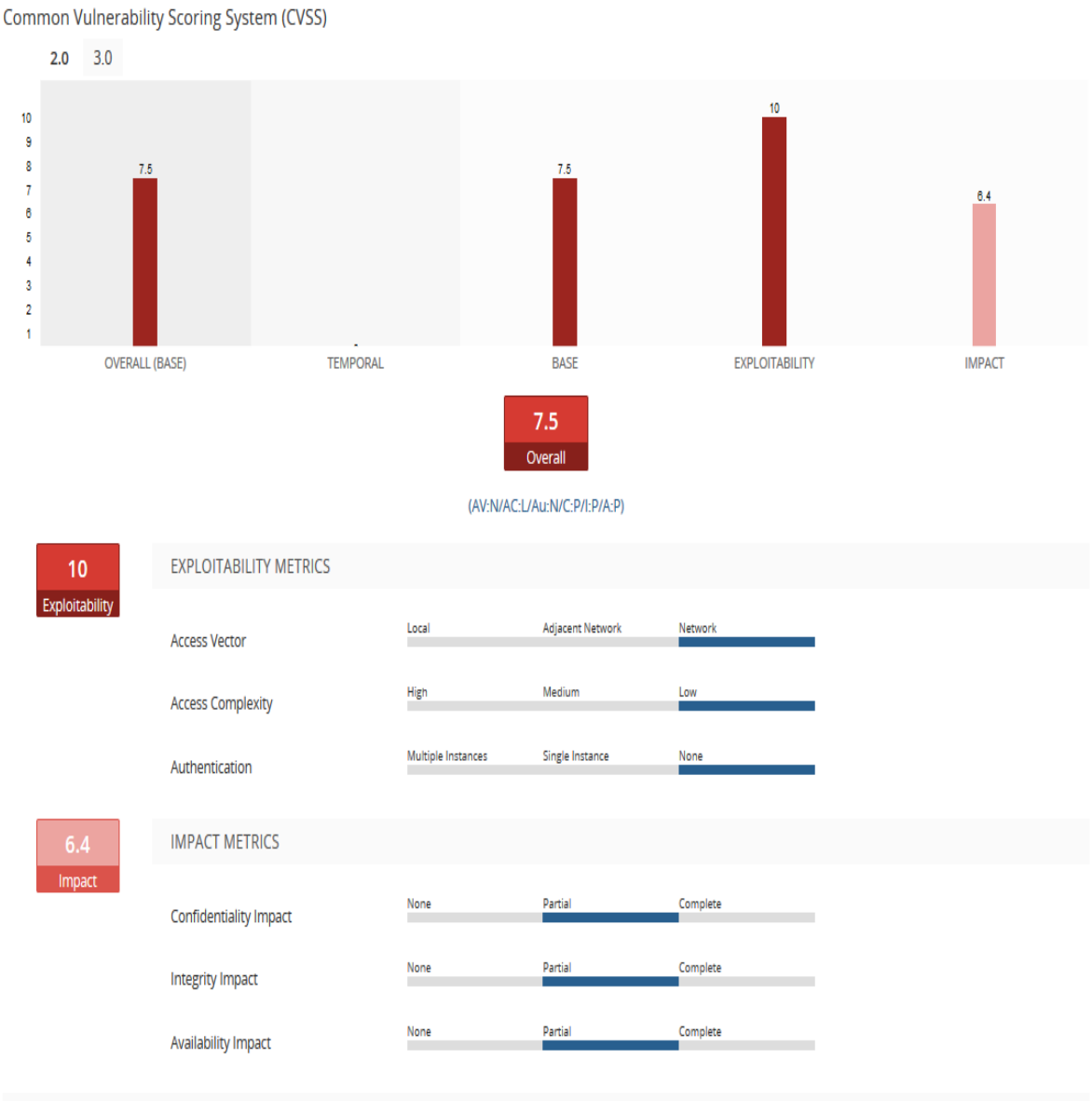


Obrázek 7: Ukázka nalezené vysoké zranitelnosti komponenty tenserflow v nástroji Black Duck[11]



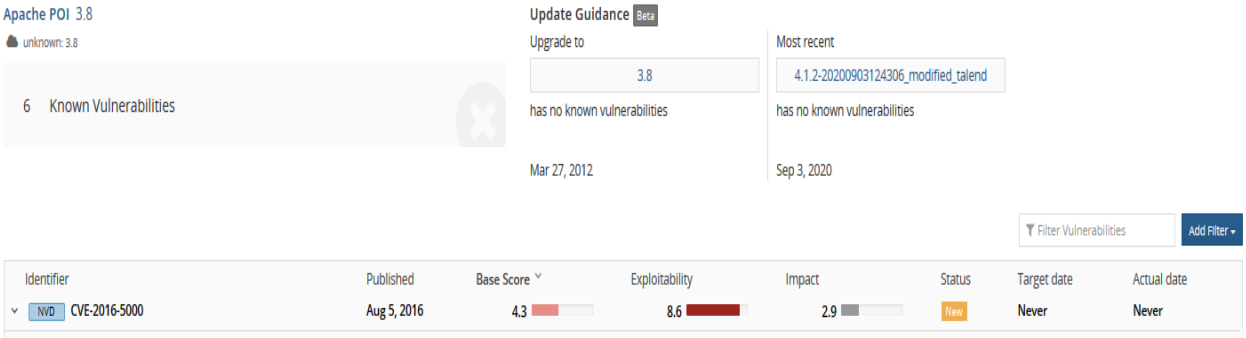
Obrázek 8: Ukázka popisu zranitelnosti komponenty tenserflow v nástroji Black Duck[12]

[11] Zdroj: Autor
[12] Zdroj: Autor

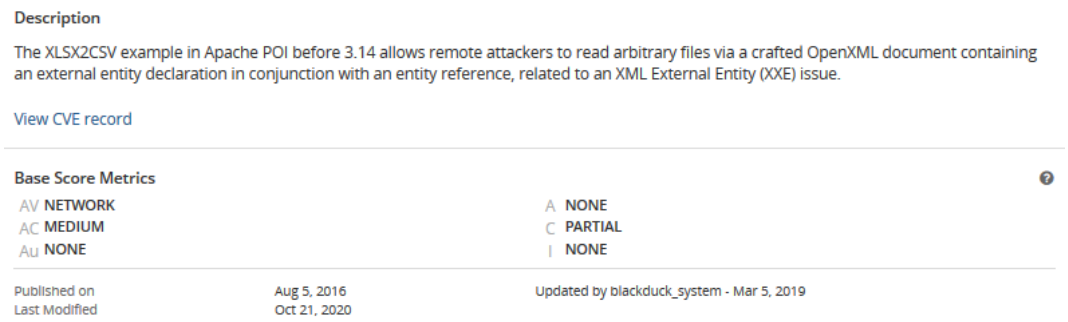


Obrázek 9: Ukázka detailu nalezené vysoké zranitelnosti komponenty tenserflow v nástroji Black Duck[13]

[13] Zdroj: Autor

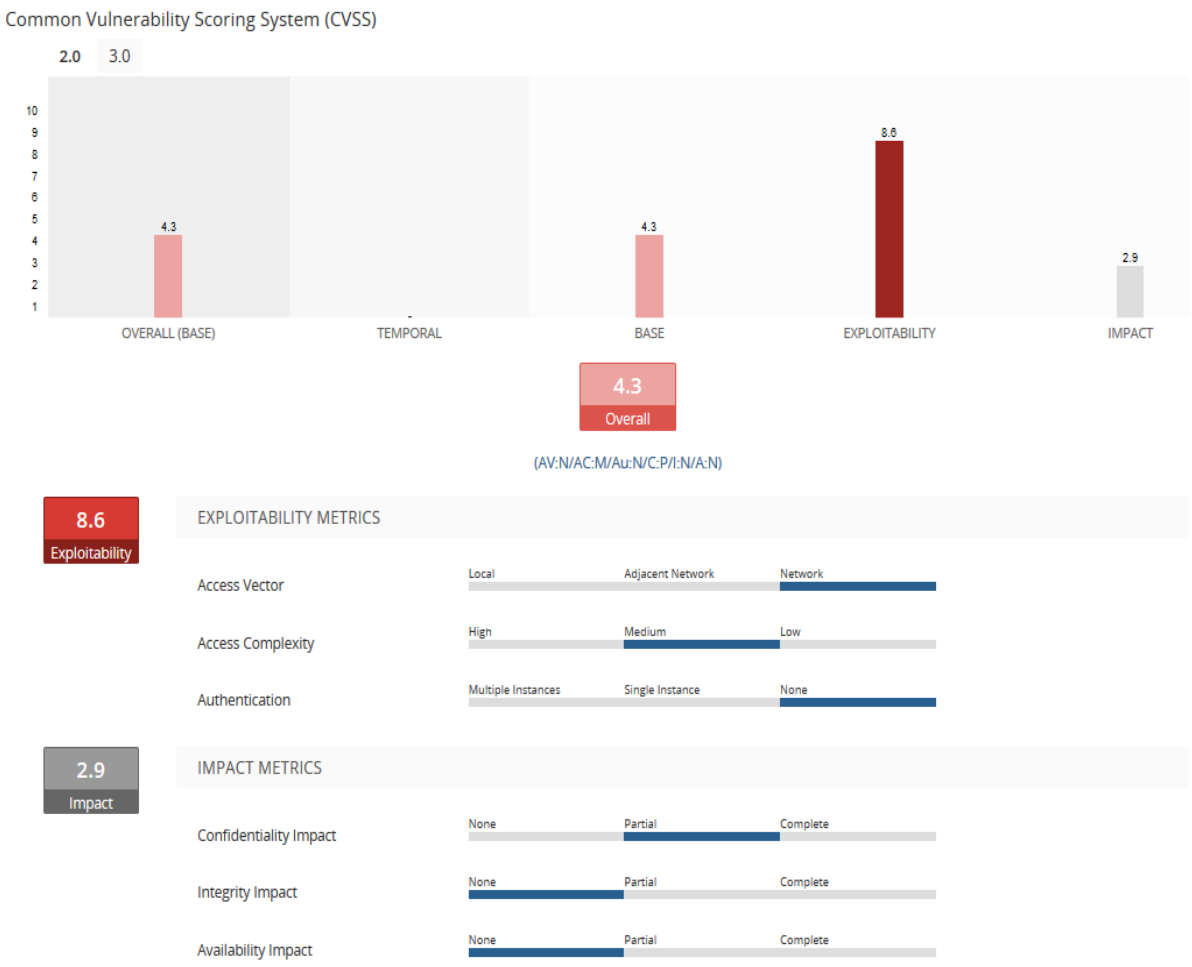


Obrázek 10: Ukázka nálezu střední zranitelnosti komponenty Apache POI v nástroji Black Duck[14]



Obrázek 11: Ukázka popisu zranitelnosti komponenty Apache POI v nástroji statické analýzy Black Duck[15]

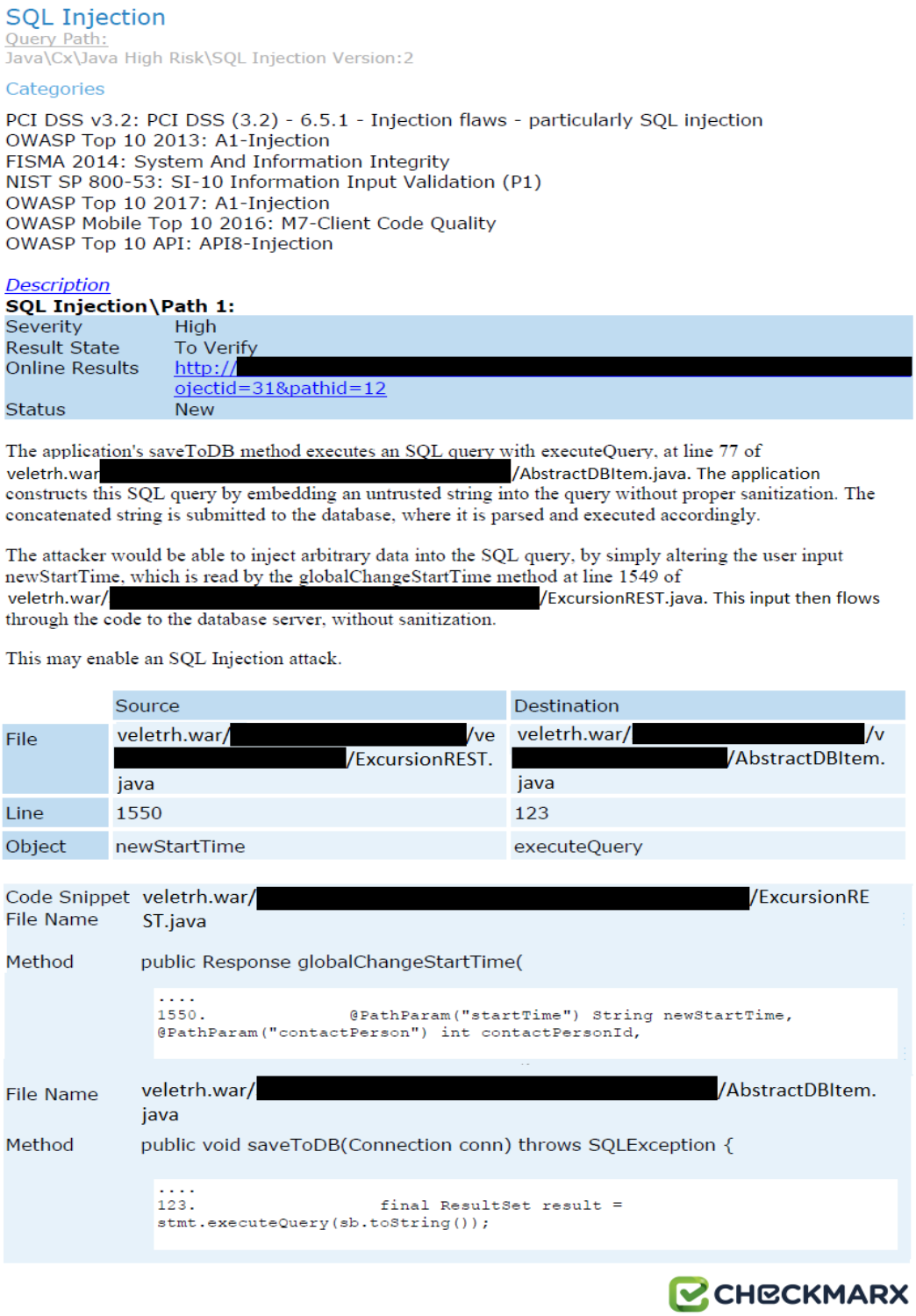
[14] Zdroj: Autor
[15] Zdroj: Autor



Obrázek 12: Ukázka detailu nalezené střední zranitelnosti komponenty Apache POI v nástroji Black Duck[16]

[16] Zdroj: Autor

Reporty z nástroje CheckMarx:



SQL Injection

Query Path:
Java\Cx\Java High Risk\SQL Injection Version:2

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.1 - Injection flaws - particularly SQL injection
OWASP Top 10 2013: A1-Injection
FISMA 2014: System And Information Integrity
NIST SP 800-53: SI-10 Information Input Validation (P1)
OWASP Top 10 2017: A1-Injection
OWASP Mobile Top 10 2016: M7-Client Code Quality
OWASP Top 10 API: API8-Injection

Description

SQL Injection\Path 1:

Severity	High
Result State	To Verify
Online Results	http://[redacted]ojectid=31&pathid=12
Status	New

The application's saveToDB method executes an SQL query with executeQuery, at line 77 of veletrh.war[redacted]/AbstractDBItem.java. The application constructs this SQL query by embedding an untrusted string into the query without proper sanitization. The concatenated string is submitted to the database, where it is parsed and executed accordingly.

The attacker would be able to inject arbitrary data into the SQL query, by simply altering the user input newStartTime, which is read by the globalChangeStartTime method at line 1549 of veletrh.war/[redacted]/ExcursionREST.java. This input then flows through the code to the database server, without sanitization.

This may enable an SQL Injection attack.

	Source	Destination
File	veletrh.war/[redacted]/ve[redacted]/ExcursionREST.java	veletrh.war/[redacted]/v[redacted]/AbstractDBItem.java
Line	1550	123
Object	newStartTime	executeQuery

Code Snippet
File Name veletrh.war/[redacted]/ExcursionREST.java

Method public Response globalChangeStartTime(

```
....
1550.              @PathParam("startTime") String newStartTime,
@PathParam("contactPerson") int contactPersonId,
```

File Name veletrh.war/[redacted]/AbstractDBItem.java

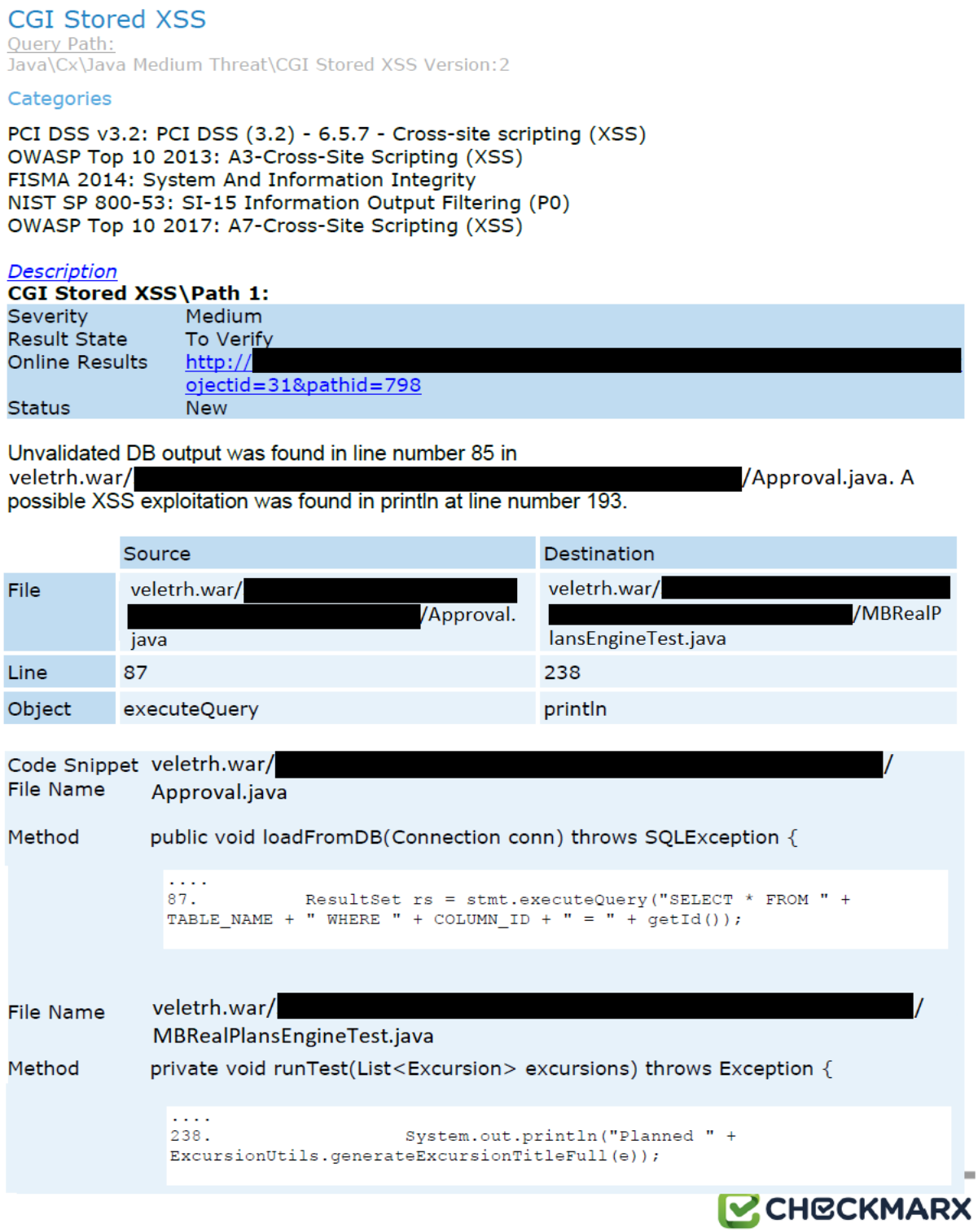
Method public void saveToDB(Connection conn) throws SQLException {

```
....
123.                  final ResultSet result =
stmt.executeQuery(sb.toString());
```

CHECKMARX

Obrázek 13: Ukázka výstupu z nástroje statické analýzy CheckMarx – nalezení vysoké zranitelnosti[17]

[17] Obrázek převzat z [17].



CGI Stored XSS

Query Path:
Java\Cx\Java Medium Threat\CGI Stored XSS Version:2

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.7 - Cross-site scripting (XSS)
OWASP Top 10 2013: A3-Cross-Site Scripting (XSS)
FISMA 2014: System And Information Integrity
NIST SP 800-53: SI-15 Information Output Filtering (P0)
OWASP Top 10 2017: A7-Cross-Site Scripting (XSS)

Description
CGI Stored XSS\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://[redacted] ojectid=31&pathid=798
Status	New

Unvalidated DB output was found in line number 85 in veletrh.war/[redacted]/Approval.java. A possible XSS exploitation was found in println at line number 193.

	Source	Destination
File	veletrh.war/[redacted]/Approval.java	veletrh.war/[redacted]/MBRealPlansEngineTest.java
Line	87	238
Object	executeQuery	println

Code Snippet
File Name veletrh.war/[redacted]/Approval.java

Method public void loadFromDB(Connection conn) throws SQLException {

```
....
87.          ResultSet rs = stmt.executeQuery("SELECT * FROM " +
TABLE_NAME + " WHERE " + COLUMN_ID + " = " + getId());
```

File Name veletrh.war/[redacted]/MBRealPlansEngineTest.java

Method private void runTest(List<Excursion> excursions) throws Exception {

```
....
238.                  System.out.println("Planned " +
ExcursionUtils.generateExcursionTitleFull(e));
```

CHECKMARX

Obrázek 14: Ukázka výstupu z nástroje statické analýzy CheckMarx – nalezení střední zranitelnosti[18]

[18] Obrázek převzat z [17].

Nástroje statické analýzy pomáhají testerům a vývojářům odhalit chyby a zranitelnosti ve vyvíjené aplikaci. Nástroje odhalí místo, kde se konkrétní chyba a zranitelnost nachází, vývojový tým ji může následně rychleji odstranit.

Paralelně k těmto činnostem probíhají všechny typy testů jako například unit, integrační a další, které si vyhodnocuje vývojový tým s byznys vlastníkem.

Na konci provedeného sprintu je svolána schůzka sprint review, na které vývojový tým prezentuje svojí vykonanou práci, dále následuje schůzka sprint retrospective, kde tým probírá, jaké věci se podařily a co je nutné zlepšit a změnit.

6.5.2 Druhá iterace – Vývojové prostředí

Na začátku další iterace opět probíhá schůzka plánování sprintu, na které se plánují činnosti, které budou provedeny v tomto sprintu. Na základě zranitelností nalezených nástroji statické analýzy bylo rozhodnuto, že do sprint backlogu budou přidány úkoly vedoucí k jejich odstranění.

Na základě seznamu činností ze sprint backlogu druhé iterace provádí vývojový tým odstranění odhalených zranitelností, které byly nalezeny příslušnými nástroji v rámci statické analýzy v prvním sprintu, a dále pokračuje v dalším vývoji aplikace. Na konci sprintu se opět provádí statická analýza za pomocí nástrojů SonarQube, Black Duck HUB, Check Marks CX SAST. Z výsledků statické analýzy je patrné, že se vývojovému týmu podařilo odstranit téměř všechny nalezené zranitelnosti z provedeného testování v první iteraci, nepodařilo se odstranit zranitelnost komponenty Apache POI (viz obrázky číslo 10, 11 a 12). Druhá iterace je závěrečná v rámci vývojového prostředí, tudíž následná iterace bude probíhat již v testovacím prostředí. Opět probíhají schůzky sprint review a sprint retrospective.

6.5.3 Třetí iterace – Testovací prostředí

Probíhá plánování sprintu s vytvořením sprintu backlogu. Jelikož se v rámci třetí iterace vývoj aplikace Škoda Veletrh přesunul do testovacího prostředí, tak do sprint backlogu byly přidány pouze činnosti s menší prioritou a dále činnosti vedoucí k odstranění zbylých zranitelnosti nalezených v rámci statické analýzy v předchozím sprintu. Jak již bylo zmíněno, v tomto sprintu se implementují funkce s menší prioritou, tedy už se nejedná o stěžejní funkce, ty byly vyřešeny v předchozích iteracích ve vývojovém prostředí. Zároveň probíhají akceptační, integrační a další testy, o které se stará vývojový tým. Na konci sprintu se provádí manuální code review, které probíhá za pomocí externí specializované firmy. Nejdříve je ale nutné projít procesem podání žádosti o provedení služby security code review [22]:

1. Kontaktovat provozovatele služby za IT bezpečnost podepsaným mailem obsahujícím následující:
    - Název projektu,
    - veškerou dostupnou dokumentaci k projektu,
    - odkaz na zdrojové kódy.
2. Oddělení IT bezpečnosti založí JIRA záznam.
    - Založení záznamu pro celou aplikaci,
    - založení pod záznamů pro jednotlivé platformy.
3. Aplikace se zařadí do fronty pro testování.
    - Priorita testování je nastavena vůči byznys kritičnosti aplikace.
4. Test aplikace a naplánování prezentace výsledků.
    - S blížícím se ukončením testování informuje oddělení IT bezpečnosti žadatele o nejbližším termínu pro prezentaci výsledků.
    - Žadatel zajistí účast zástupců vývojářů pro testované platformy, zástupce z byznysu a vlastníka produktu.
5. Prezentace výsledků.
    - V rámci prezentace se budou nálezy na příslušných platformách konzultovat pro případné určení negativních či falešně pozitivních nálezů.

HIGH Hardcoded hodnoty

KATEGORIE

OWASP Top 10 2013: A6-Sensitive Data Exposure

FISMA 2014: Media Protection

NIST SP 800-53: SC-4 Information in Shared Resources (P1)

OWASP Top 10 2017: A3-Sensitive Data Exposure

PŘÍČINA / JAK SE TO DĚJE

Při testu jsme nalezli následující hardcodované hodnoty:

Soubor: Web.config

Obsah:

Uživatelské jméno a heslo pro připojení do DBA:

```
<add name="CPODEVEntities" connectionString="metadata=res://*/Database.Database.csdl|res://*/Database.Database.ssdl|re
          Database.Database.msl;provider=System.Data.SqlClient;provider
connection string=&quot;data source=cpodev.database.windows.net;initial catalog=CPODEV;persist security info=True;
user id=[redacted];password=[redacted];
encrypt=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" /></
          connectionStrings>
```

Heslo pro certifikát:

```
<add key="Saml2:SigningCertificatePassword" value="[redacted]"/>
```

API klíče:

```
Web.config:42:    <add key="SearchServiceAdminApiKey" value="[redacted]" />
Web.config:44:    <add key="KontentWebhookSecret" value="[redacted]" />
```

RIZIKO / CO SE MŮŽE STÁT

Útočník může uložená hesla získat přímo ze zdrojového kódu a následně je zneužít pro přístup k serverové části, backendovým systémům nebo citlivým údajům uživatele aplikace.

DOPORUČENÍ / JAK SE TOMU VYHNOUT

Nikdy neukládat hesla, šifrovací klíče tak, aby je bylo možné ze zdrojového kódu získat. Pro zabezpečení těchto částí aplikace doporučujeme použít asymetrické šifrování.

Obrázek 15: Ukázka nalezené vysoké zranitelnosti v provedené manuální code review[19]

Na obrázku číslo 15 je možné vidět, že nebyl dodržen požadavek z ASA, aby se hesla k technickým účtům nezapisovala do zdrojového kódu v nezašifrované podobě.

[19] Obrázek převzat z [18].

MEDIUM **Heap Inspection**

KATEGORIE

OWASP Top 10 2013: A6-Sensitive Data Exposure

FISMA 2014: Media Protection

NIST SP 800-53: SC-4 Information in Shared Resources (P1)

OWASP Top 10 2017: A3-Sensitive Data Exposure

RIZIKO / CO SE MŮŽE STÁT

Všechny proměnné uložené aplikací v nešifrované paměti může potenciálně získat uživatel s privilegovaným přístupem k zařízení (vývojář s výjimkou). Například by privilegovaný útočník mohl načíst paměť procesu z odkládacího souboru nebo z tzv. crash dumpu.

PŘÍČINA / JAK SE TO DĚJE

Hesla jsou ukládána na více místech v aplikaci v paměti v proměnných typu String.

Po použití nejsou nijak odstraňována z paměti, což je navíc těžko proveditelné - proměnné řetězce jsou neměnné - jinými slovy, jakmile je přiřazena řetězcová proměnná, její hodnotu nelze změnit ani odstranit.

Tento problém jsme identifikovali na následujících místech aplikace:

```
Models/AccountViewModels.cs:59:        public string Password { get; set; }
Models/AccountViewModels.cs:76:        public string Password { get; set; }
Models/AccountViewModels.cs:95:        public string Password { get; set; }
```

```
34:    public string NewPassword { get; set; }
39:    public string ConfirmPassword { get; set; }
47:    public string OldPassword { get; set; }
53:    public string NewPassword { get; set; }
58:    public string ConfirmPassword { get; set; }
```

DOPORUČENÍ / JAK SE TOMU VYHNOUT

Použijte šifrované komponenty nebo udržujte hesla v char[] polích a po použití nastavte všechny položky na '0'.

Příklad použití pole typu CHAR pro uložení hesla:

```
char[] chars = new char[10];
chars[0] = 'H';
chars[1] = 'e';
chars[2] = 's';
chars[3] = 'l';
chars[4] = 'o';
```

Obrázek 16: Ukázka nalezené medium zranitelnosti v provedené manuální code review[20]

[20] Obrázek převzat z [18].

Na obrázcích číslo 15 a 16 jsou zobrazeny zranitelnosti, které byly nalezeny při provádění manuální code review a jsou zpracovány v závěrečné zprávě. Závěrečná zpráva byla poté zaslána oddělení IT bezpečnosti. V závěrečné zprávě se nachází jedna zranitelnost s vysokým a jedna se středním rizikem. U každé z těchto zranitelností je popsáno, do jakého typu rizika podle organizace OWASP patří, dále co se může stát, kdyby této zranitelnosti využil útočník, jaká je příčina této zranitelnosti a v neposlední řade je zde popsáno, jak těmto chybám příště vyvarovat. Na konci tohoto sprintu se proběhne schůzka sprint review a sprint retrospective.

6.5.4 Čtvrtá iterace – Testovací prostředí

Na úvod čtvrté iterace probíhá schůzka plánování sprintu, opět se provádí vytvoření sprint backlogu, ve kterém se na základě zranitelností nalezených v code review, provedeného v minulé iteraci, nacházejí činnosti vedoucí k odstranění těchto zranitelností. V rámci poslední iterace v testovacím prostředí již neprobíhá vývoj nových funkcionalit, provádí se ladění detailů a oprava nálezů, které byly objeveny v rámci manuální code review v předchozím sprintu. Probíhá opětovné testování, tedy další manuální code review retest. Ze zprávy vytvořené z tohoto code review vyplývá, že vývojovému týmu se podařilo odstranit nález s vysokou zranitelností (viz obrázek číslo 15). I přes snahu o úpravu kódu byla implementace neúplná a nález se střední zranitelností (viz obrázek číslo 16) stále přetrvává, proto je nutné provést další opravu. Zdrojový kód aplikace Škoda Veletrh by v této fázi vývoje měl být již 1:1 se zdrojovým kódem, který bude přesunut do kvality prostředí. Iterace číslo čtyři je poslední v testovacím prostředí, další iterace bude již probíhat v prostředí kvality. Na konci toho sprintu jsou opětovně prováděny schůzky sprint review a sprint retrospective.

6.5.5 Pátá iterace – Prostředí kvality

Iterace číslo pět opět začíná schůzkou plánování sprintu, kde se do sprint backlogu zařadí činnosti k odstranění zranitelnosti, která byla objevena již ve třetí iteraci. Po začátku sprintu tedy vývojový tým usilovně pracuje na odstranění zranitelností a chyb nalezených v rámci code review v předchozích sprintech, a zároveň s tím je prováděno penetrační testování aplikace, které má za úkol zhodnotit úroveň zabezpečení aplikace, nejdříve je však nutné sepsat dohodu o provedení penetračních testů, která obsahuje [20]:

- **Cíl testování** – Jaké jsou cíle testování dané aplikace, tedy ověření zabezpečení aplikace, související infrastruktury, webové části a API rozhraní.
- **Účel testování –** Jaký je účel testování, například zvýšit úroveň bezpečnosti, pomocí identifikace zranitelností.
- **Metodika testování a seznam testů** – Podle jaké metodiky bude testování probíhat a seznam všech testů, které budou provedeny
- **Harmonogram testování.**
- **Místo testování**.
- **Použité nástroje a technika provedených testů** – Které nástroje a techniky budou použity při penetračním testování dané aplikace.
- **Rozsah a hloubka testování**
- **Závěrečná zpráva –** Co bude výstupem penetračního testu, jaké skutečnosti bude závěrečná zpráva obsahovat.

Následně je nutné podat žádost o provedení penetračních testů. Zároveň probíhá opětovné provedení code review, které odhalí odstranění zbývající zranitelnosti zobrazené na obrázku číslo 16. Výsledkem penetračních testů je závěrečná zpráva. Nálezy, které jsou popsány v závěrečné zprávě, jsou zobrazeny níže.

Clickjack

SANS	Riziko	Hrozba	Zranitelnost	Dopad
N4	2	Malá	Malá	Střední
CVSS	CVSS:3.1/AV:L/AC:H/PR:H/UI:R/S:U/C:L/I:N/A:L			2.9 Low

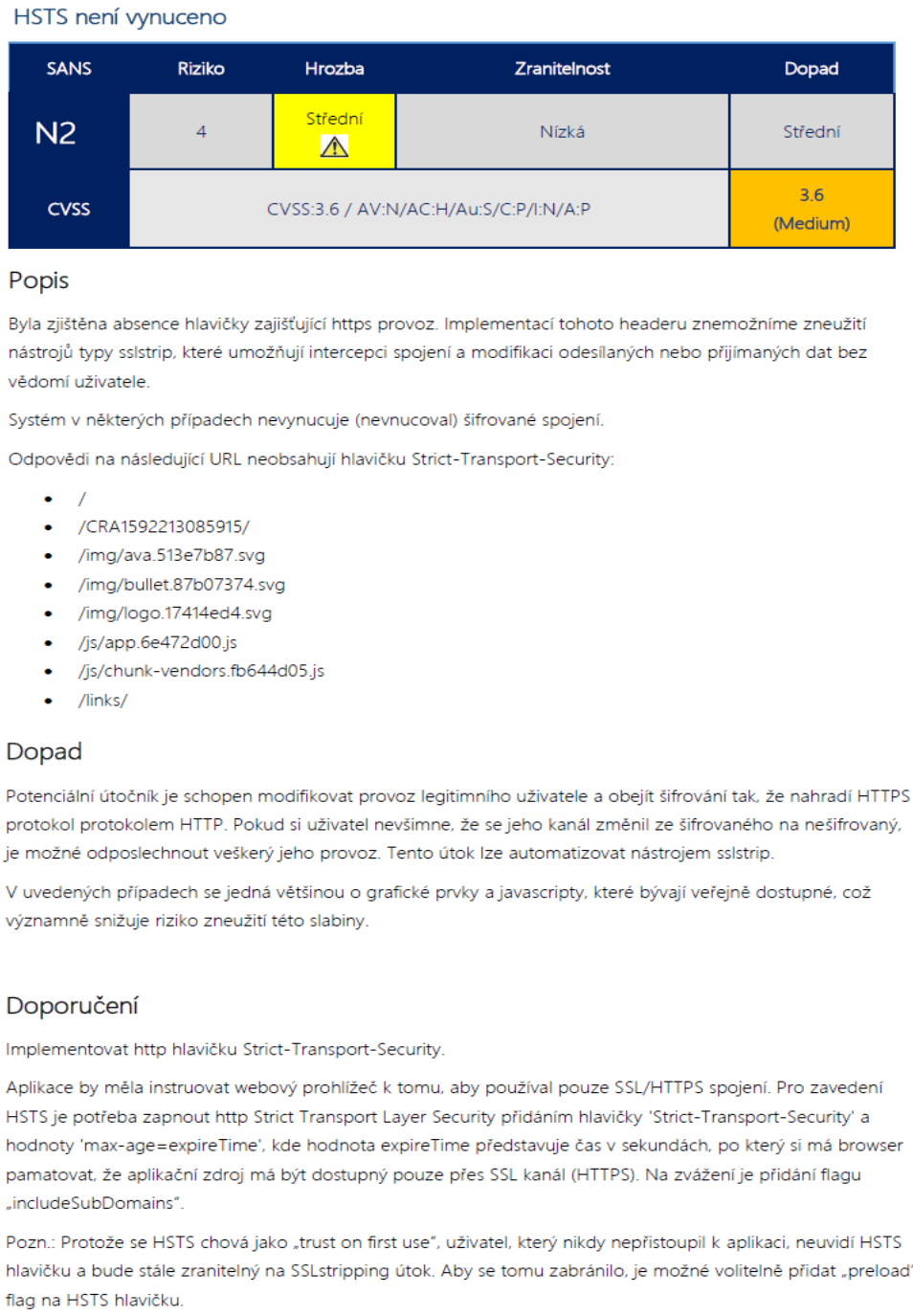
Aplikace vykazuje náchylnost k útokům typu Clickjack.

Při tomto útoku může útočník překrýt původní stránku svou vlastní stránkou a koncový uživatel poté pracuje s podvrženou stránkou útočníka.

Doporučení: Chcete-li tuto chybu zabezpečení odstranit, musíte vynutit v parametru „X-frame-options" s hodnotou „Deny" nebo „Sameorigin" na systémové části webového serveru a určit z jakých externích zdrojů je možné načíst obsah do webové stránky.

Obrázek 17: Ukázka nálezu a jeho detailu ve zprávě z penetračního testu (malé riziko)[21]

[21] Obrázek převzat z [19].



HSTS není vynuceno

SANS	Riziko	Hrozba	Zranitelnost	Dopad
N2	4	Střední ⚠	Nízká	Střední
CVSS	CVSS:3.6 / AV:N/AC:H/Au:S/C:P/I:N/A:P			3.6 (Medium)

Popis

Byla zjištěna absence hlavičky zajišťující https provoz. Implementací tohoto headeru znemožníme zneužití nástrojů typy sslstrip, které umožňují intercepci spojení a modifikaci odesílaných nebo přijímaných dat bez vědomí uživatele.

Systém v některých případech nevynucuje (nevnucoval) šifrované spojení.

Odpovědi na následující URL neobsahují hlavičku Strict-Transport-Security:

- /
- /CRA1592213085915/
- /img/ava.513e7b87.svg
- /img/bullet.87b07374.svg
- /img/logo.17414ed4.svg
- /js/app.6e472d00.js
- /js/chunk-vendors.fb644d05.js
- /links/

Dopad

Potenciální útočník je schopen modifikovat provoz legitimního uživatele a obejít šifrování tak, že nahradí HTTPS protokol protokolem HTTP. Pokud si uživatel nevšimne, že se jeho kanál změnil ze šifrovaného na nešifrovaný, je možné odposlechnout veškerý jeho provoz. Tento útok lze automatizovat nástrojem sslstrip.

V uvedených případech se jedná většinou o grafické prvky a javascripty, které bývají veřejně dostupné, což významně snižuje riziko zneužití této slabiny.

Doporučení

Implementovat http hlavičku Strict-Transport-Security.

Aplikace by měla instruovat webový prohlížeč k tomu, aby používal pouze SSL/HTTPS spojení. Pro zavedení HSTS je potřeba zapnout http Strict Transport Layer Security přidáním hlavičky 'Strict-Transport-Security' a hodnoty 'max-age=expireTime', kde hodnota expireTime představuje čas v sekundách, po který si má browser pamatovat, že aplikační zdroj má být dostupný pouze přes SSL kanál (HTTPS). Na zvážení je přidání flagu „includeSubDomains".

Pozn.: Protože se HSTS chová jako „trust on first use", uživatel, který nikdy nepřistoupil k aplikaci, neuvidí HSTS hlavičku a bude stále zranitelný na SSLstripping útok. Aby se tomu zabránilo, je možné volitelně přidat „preload" flag na HSTS hlavičku.

Obrázek 18: Ukázka nálezu a jeho detailu ve zprávě z penetračního testu (střední riziko)[22]

Na konci této iterace vývojový tým zahájí schůzky sprint review a sprint retrospective.

[22] Obrázek převzat z [19].

6.5.6 Šestá iterace – Prostředí kvality

Šestá iterace začíná schůzkou plánování sprintu, kde jsou do sprint backlogu zařazeny činnosti, které mají vést k odstranění zranitelností nalezených penetračními testy v předchozí iteraci. Po začátku sprintu se vývojový tým stará o odstranění zranitelností. Aby se zjistilo, jak se vývojářům podařilo odstranit chyby a nedostatky, bylo nutné opět provést penetrační testy. Pomocí nich se podařilo zjistit, že zůstal pouze jeden nález, konkrétně nález s malým rizikem (viz obrázek číslo 17). Z důvodu tlaku byznysu a požadavku uvolnění aplikace na trh není možné poslední nález odstranit, z toho důvodu bude požádáno o akceptaci rizika s tím, že se zranitelnost do dohodnuté doby opraví. Po konzultaci s IT bezpečnostní byla maximální doba pro opravu zranitelnosti stanovena na tři měsíce. Pro uvolnění do produkce je nutné evidovat toto riziko do systému IRMA (IT risk management). Je nutné vyplnit tabulku níže, čímž se zajistí evidence rizika do systému.

Schvalovatel	Petr Nový
Oddělení schvalovatele	IT bezpečnost
Popis zranitelnosti	Aplikace vykazuje náchylnost k útokům typu Clickjack.
Výsledný efekt	Při tomto útoku může útočník překrýt původní stránku svou vlastní stránkou a koncový uživatel poté pracuje s podvrženou stránkou útočníka.
Možný výskyt	Jednou za rok
Nemateriální škoda	Krátká ztráta důvěry
Materiální škoda	< 25 000€
Aktuální prohlášení o hodnocení rizika	Na základě udělení výjimky může dojít k tomuto útoku / zneužití zranitelnosti jen jednou. Jakmile se to stane, oddělení IT bezpečnosti aplikaci stáhne z provozu a nepustí ji, dokud nedojde k odstranění zranitelnosti.
Trestní relevance	Mírná

Kontrola rizika	Redukce rizika
Protiopatření	Vývoj nové verze
Předpokládaný výskyt	Jednou za rok
Předpokládaná materiální škoda	< 25 000 €
Důvod pro posouzení jednotlivých rizik	Redukce rizika na základě protiopatření (countermeasure) je paralelní vývoj nové upravené verze s odstraněnou zranitelností, zatím co je původní v produkci.

Tabulka 4: Evidence akceptovaného rizika[23]

[23] Zdroj: [15]

6.5.7 Sedmá iterace – Produkce

Než bude aplikace podepsána certifikátem Škoda Auto a uvolněna do produkčního prostředí, je nutné splnit bránu kvality. Jedná se o schválenou systémovou specifikaci, uživatelské akceptační testy od vlastníka aplikace, korporátní identitu (vzhled aplikace, fonty, logo) a bezpečnostní kontroly (statická analýza, code review a penetrační testy). Všechny procesy jsou popsány níže.

Postup procesu schválení systémové specifikace [22]:

1. Vlastník produktu připraví systémovou dokumentaci, kterou odešle ke schválení na projektový server.
2. Oddělení IT bezpečnosti, architektury a provozu mají dvě možnosti (každá skupina rozhoduje v rámci schvalování za vlastní část v rámci systémové specifikace. Zároveň musí se systémovou specifikací souhlasit všechny strany, aby došlo ke kompletnímu schválení):
   - Zamítnutí systémové specifikace s označením nedostatků a vrácení vlastníkovi produktu k opravě, čímž se proces opakuje.
   - Schválení systémové specifikace za příslušnou roli a posunutí schvalovacího procesu do další fáze (pokud je systémová specifikace za danou roli schválena a jinou zamítnuta, není znovu potřeba posuzování příslušnou rolí, pokud však nedojde k zásadním změnám v projektu).
3. Po schválení systémové specifikace všemi rolemi dojde k předání reportu do hodnocení brány kvality.

Postup procesu schválení UAT (uživatelské akceptační testy) [22]:

1. Vývojář nahraje zdrojové kódy aplikace, která „kandiduje“ na vydání na centrální repozitář GIT. V případě neúspěšného nahrání kódů proces opakuje. Po sestavení aplikace dojde k nahrání do test centra, kde probíhá distribuce testerům.
2. Vlastník produktu provede testování, vedoucí ke třem možnostem:
   - UAT zamítnuto – Závažné chyby.
   - UAT schváleno – Nezávažné chyby (naplánována oprava v dalším sprintu, přičemž proces začíná znovu v rámci dalšího sprintu). Report do hodnocení brány kvality.
   - UAT schváleno – Akceptační testy v pořádku. Report do hodnocení brány kvality.

Postup procesu schválení bezpečnostní kontroly [22]:

1. Vývojář nahraje zdrojové kódy do repozitáře GIT.
2. Z GIT repozitáře se zdrojové kódy přesunou ke službě bezpečnostní kontroly, která má volitelné (dle fáze vývoje, kategorizaci aplikace a dohodnutého testování v rámci systémové specifikace) tři fáze:
   - Povinná automatická kontrola zdrojových kódů pomocí nástrojů na statickou analýzu zdrojových kódů v rámci každé verze buildu aplikace. Report z výsledku kontroly se zapíše do bezpečnostní zprávy.
   - Povinné manuální code review, jenž vychází z fáze vývoje aplikace. Report výsledku kontroly se zapíše do bezpečnostní zprávy.
   - Volitelné penetrační testy aplikace (vycházející z kategorizace aplikace a dohodnutého testování v rámci systémové specifikace). Pro penetrační test musí být splněny určité podmínky. Aplikace musí mít runtime prostředí a případné napojení na backend (pokud ho aplikace má). Takový penetrační test by pak probíhal paralelně s akceptačními testy v rámci testovacího prostředí). Report z výsledku kontroly se zapíše do Bezpečnostní zprávy.
3. Bezpečnostní zpráva má tři možné výsledky:
   - Zamítnuto, závažné chyby – Předání developerovi k opravě v dalším sprintu, čímž se celý proces opakuje.
   - Schváleno, nezávažné chyby – Předání vlastníkovi produktu pro akceptaci rizik > bod 4.
   - Schváleno - Security report je v pořádku. Report do hodnocení brány kvality.
4. Vlastník produktu musí zhodnotit zjištěná rizika z reportu a má dvě možnosti:
   - Odmítnutí akceptace rizik – Předání developerovi k opravě v dalším sprintu, čímž se celý proces opakuje.
   - Akceptace rizik – Report do hodnocení brány kvality.

Postup procesu schválení CI (korporátní identity) [22]:

1. Vývojář připraví grafické návrhy nebo demo aplikace (dle metodiky KM, která je v souladu s firemní CI) a zašle je vlastníkovi produktu na posouzení.
2. Vlastník produktu má dvě možnosti:
   - Zamítnutí návrhu a předání reportu vývojáři k opravě, čímž se celý proces opakuje.

- Schválení návrhu a předání procesu na KM Marketing.

3. KM Marketing má dvě možnosti:
    - Zamítnutí návrhu a předání reportu vývojáři k opravě, čímž se celý proces opakuje.
    - Schválení návrhu a zápis do hodnocení brány kvality.

Všechny body brány kvality byly schváleny, takže aplikace Škoda Veletrh splnila veškeré nutné podmínky. Provede se podepsání aplikace certifikátem Škoda Auto, čímž se zajistí integrita. Následně může být aplikace nasazena do produkce a zpřístupněna zákazníkům. Zároveň je provedeno předání aplikace do Application Management Support, tedy příslušnému oddělení, které se bude starat o podporu zákazníkům, pravidelnou aktualizaci aplikace a opravu nově nalezených chyb.

7 ZÁVĚR

V dnešní době by měl být kladen velký důraz na bezpečnost, jinak tomu není ani ve světě vývoje softwaru. Každá společnost zabývající se vývojem softwaru by měla řešit otázky bezpečnosti softwaru již od začátku jeho vývoje, všem účastníkům procesu vývoje od byznysu až po vývojáře je nutné stále opakovat, že bezpečnost není nic navíc, ale nezbytnou součástí. Cílem této bakalářské práce bylo popsat moderní způsob agilního vývoje aplikací, porovnat agilní vývoj s Vodopádovým modelem a celkově s klasickými metodikami vývoje softwaru. V praktické části bylo cílem vytvořit popis procesů, pravidel a doporučení, které měly zajistit vysokou míru zabezpečení aplikace od počátku vývoje, až po nasazení do produkčního prostředí.

V průběhu zpracování bakalářské práce byly splněny veškeré stanovené cíle. V teoretické části této bakalářské práce byly popsány klasické metodiky vývoje softwaru, konkrétně Vodopádový model, Spirálový model a metodika RUP. V další části byl řešen vývoj softwaru podle agilních metodik. Dále byly porovnány výhody agilního vývoje oproti klasickým metodikám. Poslední kapitola teoretické části se pak věnuje všeobecně známým bezpečnostním rizikům, popsání různých typů testování a v neposlední řade popisuje metodiku ke zvýšení bezpečnosti vývoje softwaru při celém jeho životním cyklu. Praktická část se pak věnuje procesu zabezpečení vyvíjené aplikace, a to od prvotního nápadu, až po nasazení do produkčního prostředí. Při vývoji aplikace Škoda Veletrh byla objevena celá řada bezpečnostní rizik a chyb, které vývojový tým postupně musel řešit a odstraňovat. Jedno riziko se však odstranit nepodařilo, takže bylo nutné přistoupit k akceptaci a evidenci rizika.

Tato práce by mohla sloužit pro začínající vývojáře, jako ukázka procesu bezpečného vývoje aplikace, a také k dodržování standardů v oblasti moderního a bezpečného vývoje aplikací.

Možnost, jak proces zlepšit, je například zapojením interaktivního nástroje pro statické testování, což je automatizovaný nástroj, který kontroluje výstupy ze statické analýzy a pomáhá korelací nálezů ověřovat jejich pravdivost. V neposlední řadě by bylo vhodné vytvořit centrální repozitář schválených open source komponent, díky kterému by vývojáři nepoužívali zastaralé a všeobecně známé zranitelné komponenty. Celkově je proces ve Škoda Auto na celkem vyzrálé úrovni. Aby Škoda Auto zajistila dodávku kvalitních a bezpečných aplikací, investuje do těchto bezpečnostních kontrol a nástrojů v řádech miliónů korun ročně, tímto způsobem chrání Škoda Auto své dobré jméno, a také své zákazníky.

POUŽITÁ LITERATURA

[1] BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky*. Praha: Grada, 2005. Management v informační společnosti. ISBN 80-247-1075-7.

[2] KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press, 2004. ISBN 80-251-0342-0.

[3] SOMMERVILLE, Ian. *Softwarové inženýrství*. Brno: Computer Press, 2013. ISBN 978-80-251-3826-7.

[4] MARTINŮ, Jiří a Petr ČERMÁK. *Metodiky vývoje software: Studijní opora pro kombinované studium* [online]. Olomouc, 2018 [cit. 2020-11-13]. Dostupné z: https://dl1.cuni.cz/pluginfile.php/886974/mod_resource/content/1/Metodiky-vývoje-software-studijní-text.pdf

[5] BELL, Laura, Michael BRUNTON-SPALL, Rich SMITH a Jim BIRD. *Agile Application Security: Enabling security in a continuos delivery pipeline*. Sebastopol: O'Reilly Media, 2017. ISBN 978-1-491-93884-3.

[6] BUCHALCEVOVÁ, Alena. *Metodiky budování informačních systémů*. Praha: Oeconomica, 2009. Vysokoškolská učebnice. ISBN 978-80-245-1540-3.

[7] PATTON, Ron a David KRÁSENSKÝ. *Testování softwaru*. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5.

[8] SELECKÝ, M. *Penetrační testy a exploitace*. Brno: Computer press, 2012. ISBN 80-7226-636-5.

[9] Microsoft Security Development Lifecycle (SDL) Process Guidance – Version 5.2. *Microsoft* [online]. 2012 [cit. 2020-09-20]. Dostupné z: http://www. microsoft.com/en-us/download/details.aspx?id=29884

[10] Manifesto for Agile Software Development [online]. 2001 [cit. 2020-11-15]. Dostupné z: https://agilemanifesto.org

[11] Principles behind the Agile Manifesto [online]. 2001 [cit. 2020-11-16]. Dostupné z: https://agilemanifesto.org/principles.html

[12] The Difference Between Agile and Scrum | Scrum Alliance. Scrum Alliance Certification, Transform your workplace [online]. c2020 [cit. 2020-11-16]. Dostupné z: https://www.scrumalliance.org/about-scrum/definition

[13] OWASP Top Ten. *OWASP Foundation, Open Source Foundation for Application Security* [online]. c2020 [cit. 2020-12-13]. Dostupné z: https://owasp.org/www-project-top-ten/

Interní dokumenty:

[14] *Application Security Assessment: Aplikace Škoda Veletrh*. Mladá Boleslav, 2020.

[15] *Metodický pokyn MP.1.812: Metodika řízení produktů a projektů*. Mladá Boleslav, 2020.

[16] *Systémová specifikace: Aplikace Škoda Veletrh*. Mladá Boleslav, 2020.

[17] *Škoda Veletrh Scan Report: Nástroj CheckMarx*. Mladá Boleslav, 2020.

[18] *Závěrečná zpráva: Code review aplikace Škoda Veletrh*. Mladá Boleslav, 2020.

[19] *Report z penetračního testu: Aplikace Škoda Veletrh*. Mladá Boleslav, 2020.

[20] *Dohoda o provedení penetračních testů: Aplikace Škoda Veletrh*. Mladá Boleslav, 2020.

[21] *Metodický pokyn MP.1.824: Pravidla bezpečnosti informačních technologií (IT) pro systémové vývojáře*. Mladá Boleslav, 2016.

[22] *Metodický pokyn MP.1.806: Softwarové požadavky a bezpečný vývoj aplikací*. Mladá Boleslav, 2020.