UNIVERSITY OF NOVI SAD

FACULTY OF TECHNICAL SCIENCES NOVI SAD

Nikola Luburić

Integration of Software Security Design Analysis to the Agile Development Process

- Ph. D. Thesis -

Supervisor
Goran Sladić, PhD, associate professor

Novi Sad, 2019.

Nikola Luburić: *Integration of Software Security Design Analysis to the Agile Development Process*

SERBIAN TITLE:
Integracija bezbednosne analize dizajna softvera u proces agilnog razvoja

SUPERVISOR:
Goran Sladić, PhD, associate professor

LOCATION:
Novi Sad, Serbia

DATE:
September 2019

UNIVERZITET U NOVOM SADU ● **FAKULTET TEHNIČKIH NAUKA**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, **RBR**:	
Identifikacioni broj, **IBR**:	
Tip dokumentacije, **TD**:	Monografska dokumentacija
Tip zapisa, **TZ**:	Tekstualni štampani materijal
Vrsta rada, **VR**:	Doktorska disertacija
Autor, **AU**:	Nikola Luburić
Mentor, **MH**:	dr Goran Sladić, vanredni profesor
Naslov rada, **NR**:	Integracija bezbednosne analize dizajna softvera u proces agilnog razvoja
Jezik publikacije, **JP**:	engleski
Jezik izvoda, **JI**:	srpski
Zemlja publikacije, **ZP**:	Srbija
Uže geografsko područije, **UGP**:	Vojvodina
Godina, **GO**:	2019
Izdavač, **IZ**:	Fakultet tehničkih nauka
Mesto i adresa, **MA**:	Trg Dositeja Obradovića 6, 21000 Novi Sad
Fizički opis rada, **FO**: (poglavlja/strana /citata/tabela/slika/grafika/priloga)	6/159/154/13/13/0/0
Naučna oblast, **NO**:	Elektrotehničko i računarsko inženjerstvo
Naučna disciplina, **ND**:	Bezbednost softvera
Predmetna odrednica/Ključne reči, **PO**:	bezbednosna analiza dizajna, modelovanje pretnji, razvoj bezbednog softvera, životni ciklus razvoja bezbednosti, bezbednosna ekspertiza, bezbednost softvera
UDK	
Čuva se, **ČU**:	Biblioteka Fakulteta tehničkih nauka, Trg Dositeraj Obradovića 6, 21000 Novi Sad
Važna napomena, **VN**:	

UNIVERZITET U NOVOM SADU ● **FAKULTET TEHNIČKIH NAUKA**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Izvod, **IZ**:		U sklopu disertacije izvršeno je istraživanje u oblasti razvoja bezbednog softvera. Razvijene su dve metode koje zajedno omogućuju integraciju bezbednosne analize dizajna softvera u proces agilnog razvoja. Prvi metod predstavlja radni okvir za konstruisanje radionica čija svrha je obuka inženjera softvera kako da sprovode bezbednosnu analizu dizajna. Drugi metod je proces koji proširuje metod bezbednosne analize dizajna kako bi podržao bolju integraciju spram potreba organizacije. Prvi metod je evaluiran kroz kontrolisan eksperiment, dok je drugi metod evaluiran upotrebom komparativne analize i analize studija slučaja, gde je proces implementiran u kontekstu dve organizacije koje se bave razvojem softvera.
Datum prihvatanja teme, **DP**:		11.07.2019.
Datum odbrane, **DO**:		
Članovi komisije, **KO**:	Predsednik:	dr Branko Milosavljević, redovni profesor, FTN, Novi Sad
	Član:	dr Silvia Gilezan, redovni profesor, FTN, Novi Sad
	Član:	dr Gordana Milosavljević, vanredni profesor, FTN, Novi Sad
	Član:	dr Žarko Stanisavljević, docent, ETF, Beograd
	Član, mentor:	dr Goran Sladić, vanredni profesor, FTN, Novi Sad

Potpis mentora

Obrazac **Q2.HA.06-05**- Izdanje 1

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monograph documentation
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	Ph.D. thesis
Author, **AU**:	Nikola Luburić
Mentor, **MN**:	Goran Sladić, Ph.D., Associate Professor
Title, **TI**:	Integration of Software Security Design Analysis to the Agile Development Process
Language of text, **LT**:	English
Language of abstract, **LA**:	Serbian
Country of publication, **CP**:	Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2019
Publisher, **PB**:	Faculty of Technical Sciences
Publication place, **PP**:	Trg Dositeja Obradovića 6, 21000 Novi Sad
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	6/159/154/13/13/0/0
Scientific field, **SF**:	Electrical engineering and computing
Scientific discipline, **SD**:	Software security
Subject/Key words, **S/KW**:	security design analysis, threat modeling, secure software engineering, security development lifecycle, security expertise, software security
UC	
Holding data, **HD**:	Library of Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000
Note, **N**:	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Abstract, **AB**:		This thesis presents research in the field of secure software engineering. Two methods are developed that, when combined, facilitate the integration of software security design analysis into the agile development workflow. The first method is a training framework for creating workshops aimed at teaching software engineers on how to perform security design analysis. The second method is a process that expands on the security design analysis method to facilitate better integration with the needs of the organization. The first method is evaluated through a controlled experiment, while the second method is evaluated through comparative analysis and case study analysis, where the process is tailored and implemented for two different software vendors.	
Accepted by the Scientific Board on, **ASB**:		11.07.2019.	
Defended on, **DE**:			
Defended Board, **DB**:	President:	Branko Milosavljević, PhD, Full Professor, FTN, Novi Sad	
	Member:	Silvia Gilezan, PhD, Full Professor, FTN, Novi Sad	
	Member:	Gordana Milosavljević, PhD, Associate Professor, FTN, Novi Sad	Menthor's signature
	Member:	Žarko Stanisavljević, PhD, Assistant Professor, ETF, Belgrade	
	Member, Mentor:	Goran Sladić, PhD, Associate Professor, FTN, Novi Sad	

Obrazac **Q2.HA.06-05**- Izdanje 1

Abstract

With the rise of attacks on digital systems, organizations have started demanding security from the software they use. To comply with these requirements, software vendors have adopted various secure software engineering practices that enhance the regular development activities. There is a specific class of practices, called security design analysis, that is particularly challenging to integrate into agile development. Security design analysis examines the design of software or its component to assess its security posture and recommends changes and additional work to enhance it.

While cost-efficient, as it resolves vulnerabilities before they are introduced into the software, practitioners and researchers highlight low adoption rates and inefficient execution of these methods. Part of the problem lies in the specific set of expertise required to practice them efficiently, which differs from regular software engineering. Additionally, several applicability issues arise when introducing traditional security design analysis into the agile workflow, including complexity, unaccountability of work, and lack of guidance. Through this research, we address these issues.

We examine different teaching techniques to determine which support better learning outcomes for security design analysis. We find the case study analysis and the hybrid flipped classroom teaching methods suitable for our context and combine them to construct a framework for generating training workshops dedicated to teaching security design analysis. Through a controlled experiment, we evaluate our approach and demonstrate that threat models of higher quality are produced by trainees that attended the framework-formulated workshops, as opposed to the traditional workshops covering the same topics.

To address the applicability issues of complexity, unaccountability, and lack of guidance, we create a process around the security design analysis method. This process enables the incremental development of threat models as the software changes, where organizations can tailor the process according to their needs, to define and prioritize security work accordingly. We evaluate our approach through a comparative analysis with similar methods found in literature, as well as through two case study implementations of the process where we demonstrate the tailoring of the process and its execution on real-world user stories.

Finally, we combine both methods into a process for integrating software security design analysis to the agile development process, to construct the baseline threat models and set up the foundation for the continuous improvement of the security design analysis.

Keywords: security design analysis, threat modeling, secure software engineering, security development lifecycle, security expertise, software security

Rezime

Digitalna revolucija je u proteklim decenijama rezultovala širokom primenom računara i softvera u rešavanju raznih problema u raznim oblastima. Vlade, organizacije i pojedinci sve više zavise od softverskih rešenja koja rade sa osetljivim podacima i funkcijama (Gandhi i drugi, 2011). Dodatno, kriminal i terorizam pronalaze svoje mesto u digitalnom svetu. Krađa podataka, sabotaža softvera i ugrožavanje njegovih korisnika je postala stvar svakodnevnice. Zbog ovoga, vlade i organizacije sve više ulažu u bezbednost svojih sistema, što uključuje i bezbednost softvera koji kupuju (Salini i Kanmani, 2012). Naime, organizacije zahtevaju od proizvođača softvera da razviju bezbedan softver.

Razvoj bezbednog softvera se postiže primenom niza bezbednosnih aktivnosti kroz čitav životni ciklus razvoja softvera, od njegovog koncipiranja, preko dizajna i implementacije, pa sve do verifikacije i postavke u produkciju (Howard i Lipner, 2006). Jedan podskup ovih aktivnosti se bavi inženjeringom bezbednosnih zahteva, što podrazumeva prikupljanje i analizu bezbednosnih zahteva i definisanje načina za njihovo ispunjavanje. Bezbednosni zahtevi, za razliku od funkcionalnih, se bave aktivnostima koje korisnik ne bi smeo da vrši kroz softver. Na visokom nivou apstrakcije, bave se zaštitom bezbednosnih svojstava (poput poverljivosti, integriteta i dostupnosti) resursa sistema. Inženjering bezbednosnih zahteva podrazumeva analizu poslovnih zahteva, bezbednosnih standarda i pravnih regulativa koje treba softver da ispuni, kao i razmatranje motivacije i mogućnosti napadača koji bi hteli da ugroze softver. Ova analiza napadača i njihovog uticaja na softver se sprovodi kroz aktivnost koja se naziva modelovanje pretnji.

Modelovanje pretnji uključuje skup tehnika za bezbednosnu analizu dizajna softvera (Shostack, 2014b). Ove tehnike sistematski ispituju dizajn softvera i njegovih celina, razmatrajući načine kako napadači mogu da ga kompromituju. Iako koraci bezbednosne analize dizajna zavise od konkretne tehnike koja se koristi, grubo se mogu grupisati u tri veće celine:

- *Analiza modula* – gde se ispituje modul kako bi se razumeo njegov način rada, koji su tokovi podataka, i način na koji komunicira sa spoljnim entitetima.
- *Analiza pretnji* – gde se određuju pretnje za modul i resurse kojima manipuliše, poput bitnih podataka i servisa. Pretnje se dekomponuju na napade putem kojih se mogu ostvariti, ranjivosti koje dozvoljavaju da napad uspe i bezbednosne kontrole koje regulišu date ranjivosti.

- *Analiza rizika* – gde se razmatraju negativne posledice realizacije pretnje i verovatnoće da se to desi, kako bi se zaključile protivmere i odredila pogodna strategija za regulisanje rizika.

Rezultat ove aktivnosti su zahtevi za izmenu dizajna softvera ili zahtevi za njegovo dodatno obezbeđivanje, na primer kroz dodatan razvoj ili integraciju bezbednosnih alata. Bezbednosna analiza dizajna je aktivnost koja pronalazi ranjivosti u sistemu u ranim fazama razvoja, pre nego što se program napiše i kada je najjeftinije regulisati bezbednosne probleme. Međutim, efikasna primena ove aktivnosti u kontekstu savremenih metodologija razvoja softvera je relativno retka, uprkos sve većoj potražnji za bezbednim softverom (Baca i Carlsson, 2011; Oyetoyan i drugi, 2016; Türpe i Poller, 2017).

Ključni problemi koji su identifikovani za uspešnu primenu bezbednosne analize dizajna u proces agilnog razvoja softvera uključuju:

1. Nedostatak znanja da se efikasno sprovodi. Veština koje je potrebna da se izvrši bezbednosna analiza dizajna se teško prenosi i stiče i ne odgovara mentalitetu koji je potreban za standardan razvoj softvera (Shostack, 2014b; Schoenfield, 2015). Zbog ovoga, timovi koji se bave razvojem softvera uglavnom ne poseduju dovoljno veštine da praktikuju bezbednosnu analizu dizajna (Morrison i drugi, 2017).
2. Neslaganje paradigmi između agilnog razvoja i tradicionalnih tehnika za bezbednosnu analizu dizajna, koje su krojene za vodopad metodologije razvoja softvera. U agilnom razvoju, prvobitni dizajn se brzo menja kako se novi zahtevi za softver definišu. Tradicionalne tehnike bezbednosne analize dizajna podrazumevaju temeljnu analizu prvobitnog dizajna, što rezultuje zastarelim modelima pretnji i rezultatima analize upitne vrednosti. Zbog svoje kompleksnosti, skupo je sprovesti kompletnu bezbednosnu analizu dizajna za svaki novi razvoj. Dodatna problematika se ispoljava i kroz nedostatak opipljivih rezultata analize i jasno definisanog kraja analize, što dovodi do nejasno definisanog posla i njegovog trajanja. Najzad, značajan deo tehnika za bezbednosnu analizu dizajna ne pruža dovoljno smernica kako izvršavati analizu niti kako je integrisati u razvoj softvera (Poller i drugi, 2017; Luburić i drugi, 2018a).

U sklopu ovog istraživanja, fokus se stavlja na razvoj podržan *Scrum* radnim okvirom, kao najrasprostranjenijim pristupom agilnog razvoja (CollabNet VersionOne, 2019). Iz prethodno navedenih problema identifikovana su istraživačka pitanja koje teza obrađuje:

1. Kako efikasno naučiti *Scrum* timove da vrše bezbednosnu analizu dizajna?
2. Kako integrisati bezbednosnu analizu dizajna u proces agilnog razvoja softvera baziranog na *Scrum*-u?

Spram istraživačkih pitanja, formirana je centralna hipoteza disertacije koja glasi:

- *Timovi koji razvijaju softver po Scrum agilnoj metodologiji mogu praktikovati bezbednosnu analizu dizajna softvera tokom čitavog njegovog razvoja, čime dokazuju da je bezbednost adekvatno razmotrena tokom dizajna, pod uslovom da:*
    - *Dobiju prikladnu obuku kako bi efikasno praktikovali bezbednosnu analizu dizajna.*
    - *Tehnika bezbednosne analize dizajna je kompatabilna sa načinom rada u Scrum procesu, i ne zahteva uvođenje novih uloga u razvojni tim, niti konstrukciju kompleksne dokumentacije.*
    - *Posao je opipljiv, merljiv, konačan, te mu se može odrediti prioritet i formulisati plan realizacije kao i za standardan razvoj.*
    - *Postoji dovoljno uputstva kako usvojiti, koristiti, i adaptirati tehniku bezbednosne analize dizajna za različite organizacione kontekste.*

Polazna hipoteza je razložena na tri očekivana doprinosa disertacije:

1) Konstrukcija radnog okvira za pomoć pri kreiranju radionica, kroz koje inženjeri softvera efikasno uče kako da sprovode bezbednosnu analizu softvera.
2) Definisanje procesa bezbednosne analize dizajna koji rešava probleme koji postoje u interakciji procesa agilnog razvoja i tradicionalnih tehnika za bezbednosnu analizu dizajna.
3) Definisanje procesa za integraciju bezbednosne analize dizajna u agilan razvoj putem upotrebe radnog okvira za konstrukciju radionica kako bi se uspostavilo znanje za integraciju, upotrebu i kontinualno unapređenje procesa bezbednosne analize dizajna, opisan pod tačkom dva. Kombinacija ove dve tehnike treba da podrži agilan razvoj softvera i da odgovori na zahteve za bezbednosnu analizu dizajna koje definišu standardi na temu razvoja bezbednog softvera.

U cilju razmatranja prvog dela hipoteze i prvog istraživačkog pitanja, istražuju se metode obuke namenjene učenju tehnika razvoja bezbednog softvera, što je tema sekcije 2.1. Sa rastom interesovanja za razvoj bezbednog softvera, formirana je istraživačka grana koja se bavi ispitivanjem tehnika obuke inženjera softvera da razvijaju bezbedniji softver. Tradicionalna učionica, gde predavač izlaže znanje dok polaznici pasivno slušaju, se u literaturi i praksi ističe kao neadekvatna za obuku na temu razvoja bezbednog softvera. Zbog ovoga, alternativne tehnike za obuku se ispituju.

Varma i Garg (2005) su razmatrali upotrebu različitih tehnika učenja u polju softverskog inženjerstva. Zaključili su da je analiza studija slučajeva (engl. *Case study analysis*) efikasna tehnika za obuku inženjera softvera, postavljajući pogodan teren na kom se može ispoljiti kompleksnost i zamršenost koja postoji u softverskim proizvodima. Meneely i Lucidi (2013) su primenili ovu tehniku u sklopu univerzitetskog kursa čija tema je razvoj bezbednog softvera. Na početku svakog predavanja, autori analiziraju „ranjivost dana", gde ispituju primere bezbednosnih problema iz stvarnog sveta i analiziraju posledice ranjivosti, način na koji bi se mogla eksploatisati, programski kod njene implementacije, kao i način na koji bi se ranjivost rešila. Tokom ove aktivnosti, autori zapažaju visok nivo interesovanja među studentima. Analiza studija slučajeva postavlja polaznike u realistične situacije, gde je potrebno da se izbore sa nepotpunim informacijama, ciljevima koji su u koliziji i ograničenjima poput novca i vremena (Andersen i Schiano, 2014) i stimuliše kritičko razmišljanje (Dunne i Brooks, 2004).

Gamifikacija (engl. *Gamification*) predstavlja moderan metod učenja, gde se kroz igru u kojoj učestvuju polaznici formiraju pozitivni ishodi učenja povećanjem angažmana i interaktivnosti među polaznicima. Denning i drugi (2013) su definisali kartašku igru *Control-Alt-Hack*, kao alat za učenje računarske bezbednosti kroz igru. Još jednu kartašku igru, *Elevation of Privilege*, je definisao Shostack (2014a) sa ciljem da nauči inženjere softvera kako da vrše bezbednosnu analizu dizajna softvera. Kroz ovu igru, učesnici diskutuju ranjivosti u sistemu, mesta odakle napadač može da sprovede napad i resurse koje bi želeo da ugrozi. U kontekstu agilnog razvoja softvera, Williams i drugi (2010) su definisali *Protection Poker*, igru koja pomaže razvojnim timovima da odrede i rangiraju bezbednosne rizike koji postoje u softveru i koje novi korisnički zahtevi mogu doneti.

Kassicieh i drugi (2015) ispituju korporativne radionice iz domena računarske bezbednosti i ističu njihovu neefikasnost. Autori ističu da je deo problema to što tradicionalna tehnika učenja ne podržava razvoj „mentaliteta" koji je potreban za bezbednost. Nekoliko autora ističe da je za aktivnosti poput bezbednosne analize dizajna neophodan „napadački pogled" na problem (Shostack, 2014a; Schoenfield, 2015; Carranza i DeCusatis, 2015). Carranza i DeCusatis (2015) ispituju hibridnu invertovanu učionicu (engl. *Hybrid flipped classroom*) kao pogodniju tehniku za razvoj napadačkog pogleda u odnosu na tradicionalnu učionicu. Kod hibridne invertovane učionice polaznici dobijaju materijale, poput video predavanja, članaka ili poglavlja u knjizi, koje treba samostalno da prođu pre radionice. Tokom radionice, polaznici diskutuju materijale sa predavačem kako bi dodatno razumeli gradivo i stekli novi pogled na izneseno znanje. Potom rade na zadacima i sprovode aktivnosti koje zahtevaju znanje iz pripremnih materijala. Invertovanu učionicu ističu i Kassicieh i drugi (2015), kao jedan način da se unaprede radionice iz domena računarske bezbednosti.

Elektronski-podržano učenje (engl. *E-learning*) je pronašlo specifičnu primenu kod učenja tehnika razvoja bezbednog softvera. Nekoliko autora je formiralo platforme za učenje o softverskoj bezbednosti kroz ranjive softverske pakete. Pohl i drugi (2015) ističu *BREW*, ranjiv softver koji studenti koriste kao metu za napada. Walden (2008) razvija *OWASP WebGoat* projekat, što predstavlja kolekciju malih aplikacija, gde svaka ima jednu ili više ranjivosti u vidu izazova koje treba rešiti. Ovi softverski paketi donose visok nivo angažmana među polaznicima zbog svoje interaktivnosti i sistema izazova (Pohl i drugi, 2015; Kimminich, 2019). Putem njih, polaznici savladavaju ranjivosti, napade i bezbednosne kontrole.

Prethodno navedene tehnike su ispitivane u sklopu predmeta na završnoj godini osnovnih akademskih studija na Fakultetu tehničkih nauka u Novom Sadu, koji pokriva temu razvoja bezbednog softvera. Spram preporuka istaknutih u literaturi, kombinuju se tehnike hibridne invertovane učionice (NSF, 2008; Andersen i Schiano, 2014) sa analizom studija slučaja (Carranza i DeCusatis, 2015) kako bi se formirao radni okvir spram kog se kreiraju laboratorijske vežbe. Efikasnost radnog okvira se evaluira i utvrđuju se bolji rezultati učenja u odnosu na tradicionalnu učionicu. Kako bi se rešio problem kompleksnosti radnog okvira, formira se proširenje koje se zasniva na upotrebi gamifikacije i ranjivih softverskih paketa (Kimminich, 2019). Rezultujući radni okvir je tema poglavlja 3.

Kako bi se odgovorilo na drugo istraživačko pitanje i drugi deo hipoteze, u sekciji 2.2 se razmatraju postojeće tehnike za bezbednosnu analizu dizajna i njihovi problemi, kao i zahtevi koje standardi propisuju za ove tehnike. Pogodan izvor ovih tehnika predstavlja skorašnja sistematična studija mapiranja koja izlaže tehnike za inženjering bezbednosnih zahteva u agilnom razvoju (Villamizar i drugi, 2018). Ovo uključuje *Abuser Story* (Peeters, 2005), *SEAP* (Baca i drugi, 2015), *Secure Scrum* (Pohl i Hof, 2015), *Security Backlog* (Azham i drugi, 2011), *S-Scrum* (Mougouei i drugi, 2013), *Agile Security Framework* (Singhal, 2011), *Security assurance case* (Othmane i drugi, 2014a), i *VAHTI-Scrum* (Rindell i drugi, 2015). Navedene tehnike se analiziraju kako bi se utvrdilo kako odgovaraju na probleme kompleksnosti, neopipljivosti posla i nedostatka uputstva za upotrebu i adaptaciju. Iz ove analize proizilaze sledeći zaključci:

1. Nedostatak uputstva za rad, izvršavanje i prilagođavanje tehnike predstavlja čest problem.
2. Posao je opipljiv i merljiv kada se u proces uključi ekspert iz domena softverske bezbednosti, no to ugrožava zahtev za jednostavnost tehnike jer je za svaku analizu potrebno njegovo angažovanje.
3. Tehnike koje ispunjuju zahtev za jednostavnost ne objašnjavaju kada je analiza gotova, te nije moguće izmeriti posao i ispuniti zahtev jasno definisanog posla.

U sklopu iste sekcije se ispituje i sistematičan pregled literature na temu tehnika bezbednosne analize dizajna koju sprovode Tuma i drugi (2018). Autori ispituju 26 tehnika i razmatraju njihovo usvajanje i primenu u industriji savremenog razvoja softvera. Istaknuto je nekoliko problema koji ograničavaju usvajanje ovih tehnika, uključujući nedovoljnu upotrebu automatizacije, nedostatak definicije kada je posao završen i nedostatak smernica kako da se metoda praktikuje.

Galvez i Gurses (2018) ispituju izazove koji postoje prilikom modelovanja pretnji u agilnom razvoju. Uočeni su problemi u održavanju i ažuriranju kompleksnih modela pretnji, u određivanju pogodne apstrakcije za modelovanje pretnji, kao i u nedostatku znanja i smernica za vršenje ove aktivnosti. Sličnu studiju sprovode Cruzes i drugi (2018) koji ispituju upotrebu bezbednosne analize dizajna u kontekstu određenog proizvođača softvera. Autori su ustanovili da razvojni timovi ne žele da proizvode modele pretnji jer oduzimaju previše vremena u odnosu na vrednost koju donose, kao i da postoji nedostatak ekspertize, znanja i smernica kako da efikasno izvršavaju ovu aktivnost.

Nakon pregleda predloženih tehnika za bezbednosnu analizu dizajna iz literature, analizirani su industrijski standardi koji propisuju zahteve za proces razvoja bezbednog softvera, stavljajući fokus na aktivnosti koje se tiču bezbednosne analize dizajna. *IEC 62443-4-1:2018* standard definiše zahteve za razvoj bezbednih softverskih i hardverskih proizvoda (IEC, 2018a). Odavde se dodatno ispituju prakse vezane za inženjering bezbednosnih zahteva i konstrukciju bezbednog dizajna. Po sličnom pristupu, analizira se *NIST SP 800-160v1*, standard koji definiše proces za inženjering bezbednih sistema (Ross i drugi, 2016).

Iz navedenog istraživanja se izdvajaju principi koji vode formulisanje procesa za bezbednosnu analizu dizajna, sa ciljem definisanja procesa koji ispunjava zahteve industrijskih standarda (IEC, 2018; Ross i drugi, 2016) i rešava neke od problema u postojećim tehnikama (Luburić i drugi, 2018a; Tuma i drugi, 2018; Cruzes i drugi, 2018; Galvez i Gurses, 2018):

- Visok stepen složenosti procesa je u kontradikciji za zahtevom agilnosti u razvoju softvera.
- Neophodan je određen nivo bezbednosne ekspertize kako bi se posao mogao definisati i rangirati. Potrebno je obučiti razvojni tim ili uvesti domenskog eksperta. Najzad, usled nemogućnosti dokazivanja da su sve bitne pretnje pronađene i razložene, potrebno je vremenski ograničiti izvršavanje procesa bezbednosne analize dizajna.
- Organizacijama su potrebna uputstva i usmeravanja kako bi usvojili i prilagodili metod za bezbednosnu analizu dizajna. Ovo podrazumeva opis samog metoda, primere njegove upotrebe i uputstvo kako ga prilagoditi različitim potrebama organizacije.

Poglavlje 3 prikazuje detalje radnog okvira za konstrukciju laboratorijskih vežbi (Luburić i drugi, 2019a) čija svrha je obuka na temu bezbednosne analize dizajna. Kroz sekciju 3.1 se opisuju komponente radnog okvira i proces njegove upotrebe. Radni okvir se sastoji iz četiri komponente:

1. Odabrane tehnike za bezbednosnu analizu dizajna, što predstavlja glavni cilj učenja i glavni ulaz za korišćenje radnog okvira,
2. Pripremni materijali koji sadrže potrebno znanje koje polaznici treba da usvoje kako bi učestvovali u laboratorijskoj vežbi,
3. Jedna ili više studija slučaja koje će biti meta bezbednosne analize dizajna,
4. Laboratorijske vežbe kao glavni rezultat radnog okvira, što podrazumeva skup pripremnih materijala, opis studija slučaja, skup zadataka za polaznike i tok vežbi u vidu uputstva za predavača.

Radni okvir definiše sledeće korake potrebne za definisanje laboratorijskih vežbi ili radionica:

1. Odabir tehnike za bezbednosnu analizu dizajna;
2. Dekomponovanje tehnike na segmente sa idejom da jedan segment bude tema jedne radionice, odnosno laboratorijske vežbe;
3. Ispitivanje svakog segmenta, tako da se:
    a. Odrede relevantni bezbednosni koncepti (napadi, ranjivosti i protivmere);
    b. Konstruišu pripremni materijali (npr. u vidu prezentacija, video materijala, tekstualnih dokumenata);

   c. Definišu zahtevi koje studija slučaja treba da ispuni kako bi se mogao uspešno sprovesti segment bezbednosne analize dizajna i primeniti prateći pripremni materijali;
   d. Odredi tok laboratorijske vežbe, uključujući i zadatke;
   e. Proceni da li je laboratorijska vežba previše kompleksna uzimajući u obzir vremensko ograničenje i izdeli u više vežbi ukoliko jeste.
4. Nakon ispitivanja svih segmenata, formuliše se jedna ili više studija slučaja spram skupa svih zahteva, gde kreator vežbi treba da se trudi da stvori realistične studije koje će biti dovoljno bliske polaznicima.
5. Uzimajući u obzir definisane studije slučaja, formulišu se konačni tokovi laboratorijskih vežbi, gde se u ovom koraku mogu spajati i razdvajati vežbe spram organizacionih ograničenja i procene kreatora vežbi.

Sekcija 3.2 ističe primere izvršavanja radnog okvira i njegova unapređenja, sa ciljem pružanja dodatnih smernica za upotrebu radnog okvira. Ovde se opisuje upotreba radnog okvira za stvaranje šest laboratorijskih vežbi za potrebe predmeta na osnovnim akademskim studijama koji pokriva temu razvoja bezbednog softvera. Za metod bezbednosne analize dizajna odabran je STRIDE (Shostack, 2014b), gde se prvobitnim razlaganjem formira skup od šest segmenata, jedan za svaku klasu pretnji koje definiše STRIDE. Za svaku od metoda definišu se bezbednosni koncepti koji se tiču date klase pretnji, stavljajući fokus na napade i odbrane koji su relevantni za domen. Spram segmenata se formira šest vežbi koje se opisuju srednjim nivoom detalja. Ovde je u grubim crtama istaknuto koje su teme koje pripremni materijali pokrivaju, šta su zahtevi za studije slučajeva i kako izgledaju sami tokovi vežbi. Najzad, za studiju slučajeva koja se koristi u svim vežbama je opisan informacioni sistem moderne bolnice kao sistem koji sadrži značajnu količinu vrednih podataka i resursa i otvara dovoljno prostora gde se mogu primeniti tehnologije poznate studentima.

U sklopu iste sekcije se detaljnije opisuje jedna laboratorijska vežba koja se tiče analiziranja pretnji neporecivosti i dizajniranja mehanizma za beleženje događaja koji ispunjava kako bezbednosne tako i funkcionalne zahteve. Ovde su opisani pripremni materijali koji se koriste. Potom se razrađuje studija slučaja informacionog sistema bolnice, stavljajući fokus na osetljive podatke i operacije u sistemu. Najzad, ističe se struktura zadataka i aktivnosti koje se sprovode tokom date vežbe koje predavač i polaznici izvršavaju.

Na kraju sekcije se definiše proširenje za radni okvir, gde se primenjuju tehnike gamifikacije i elektronski-podržanog učenja kroz uvođenje javno dostupnog ranjivog softverskog paketa. Uvođenje ovakvog alata značajno smanjuje kompleksnost konstrukcije laboratorijskih vežbi i napor koji je potreban da se one naprave. Proces konstrukcije vežbi se menja tako što kreator vežbi sakuplja zahteve za studiju slučaja, nakon čega pretražuje javno dostupne repozitorijume (Siles i Bennets, 2019) kako bi pronašao pogodnog kandidata da ispuni date zahteve. Ranjivi softverski paketi nude pripremne materijale kako bi se izazovi savladali, te se oni mogu iskoristiti u sklopu izrade pripremnih materijala za laboratorijske vežbe. Bitno ograničenje predstavlja relativno mali broj kvalitetnih ranjivih softverskih paketa što značajno ograničava kreatora vežbi u izboru i integraciji ovih alata u laboratorije. U sklopu ovog dela disertacije opisana je konstrukcija nove laboratorijske vežbe za pretnje koje *Injection* napadi donose, a odabrani su zbog rasprostranjenosti i ozbiljnih negativnih posledica koje mogu prouzrokovati. Nakon definisanja zahteva za studiju slučaja, odabran je *OWASP Juice Shop* alat (Kimminich, 2019), koji ispunjava sve navedene zahteve. Ovaj alat predstavlja realističnu veb-prodavnicu koja sadrži mnoštvo ranjivosti i koja je u značajnoj meri slična aplikacijama koje studenti razvijaju u sklopu predmeta koje ranije slušaju. Najzad, ovde je dat detaljan opis aktivnosti koje se sprovode tokom vežbi i način na koji je alat iskorišćen da doprinese pozitivnim ishodima učenja.

Sekcija 3.3 ističe detalje kontrolisanog eksperimenta, ankete i opservacija predavača koje su sprovedene kako bi se pokazalo da laboratorijske vežbe kreirane uz pomoć radnog okvira stvaraju bolje ishode učenja u odnosu na tradicionalan pristup. Ovde je opisan kurs u okviru kog je primenjen radni okvir kako bi se istakle bitne informacije vezane za same polaznike. U sklopu kursa, studenti konstruišu model pretnji za određen informacioni sistem kao deo projektnog zadatka koji je neophodno uraditi kako bi studenti uspešno završili kurs.

Kroz kontrolisani eksperiment se poredi prosečan kvalitet modela pretnji koji sastavljaju timovi studenata dve generacije, gde jedna generacija prolazi kroz tradicionalne vežbe, a druga kroz vežbe stvorene koristeći radni okvir. Obe generacije formiraju model pretnji za isti informacioni sistem. Za ocenjivanje kvaliteta modela pretnji se koriste formule koje definišu Scandariato i drugi (2015), što podrazumeva:

- Kvalitet dijagrama toka podataka na osnovu kog se vrši identifikacija pretnji, gde se kvalitet meri u tome koliko precizno oslikava tokove podataka u sistemu i u kojoj meri je nivo detalja pogođen.
- Kvalitet identifikacije pretnji, gde se meri broj ispravno navedenih pretnji (*true positive*), neispravno navedenih pretnji (*false positive*) i neidentifikovanih pretnji (*false negative*).

Fokus se stavlja na korak identifikacije pretnji, gde se računaju i porede prosečna preciznost i odziv za identifikovane pretnje na nivou generacije. Četrnaest modela pretnji je razmatrano iz generacije koja je slušala tradicionalne vežbe i sedamnaest iz one koja je slušala vežbe kreirane uz pomoć radnog okvira.

Efikasnost proširenja radnog okvira sa ranjivim softverskim paketom se ispituje anketiranjem polaznika i razmatranjem opservacija predavača. Tokom vežbi, predavači su posmatrali nivo angažovanja polaznika, prirodu njihove interakcije sa ranjivim softverskim paketom, efekat koji je alat imao na vežbe i probleme koji su nastajali zbog njegove upotrebe. Anketa upućena polaznicima je sadržala tvrdnje koje su se ocenjivale po Likertovoj skali sa 5 tačaka (Albaum, 1997), kao i nekoliko otvorenih pitanja, koja su navedena u sekciji 3.3.4.

Rezultati evaluacija su opisani u sekciji 3.4. Sa aspekta kontrolisanog eksperimenta, primećeno je značajno povećanje u preciznosti (87% u odnosu na 68%) i opozivu (81% u odnosu na 54%) kod generacije studenata koja je slušala vežbe konstruisane uz pomoć radnog okvira, u odnosu na studente koji su slušali tradicionalne vežbe. Ovi rezultati su statistički značajni, što se proverava uz pomoć *Mann-Whitney* testa. Dodatno, Scandariato i drugi (2015) ističu da rezultati preko 80% u preciznosti i opozivu predstavljaju povoljan rezultat i uspešnu analizu.

Povodom evaluacije proširenja, predavači su primetili povećan stepen angažmana studenata. Deo zasluge se pripisuje realističnosti ranjive aplikacije koja izgleda kao prosečna veb-prodavnica koja se može pronaći na internetu. Drugi razlog je familijarnost koju studenti imaju sa ovakvom aplikacijom, koja je izgrađena njima poznatim tehnologijama i principima. Najzad, sistem izazova koji *OWASP Juice Shop* pruža je stimulisao atmosferu takmičenja što je dodatno podiglo nivo zabave i angažmana. Anketa koja je usledila je potvrdila opservacije gde su vežbe koje su koristile ranjivi softverski paket bile najbolje ocenjene od svih ostalih.

U sklopu ove sekcije razmatraju se i ograničenja studije i samog radnog okvira. Deo unapređenja kvaliteta modela pretnji generacije koja je slušala vežbe konstruisane putem radnog okvira dolazi i iz unapređenja kvaliteta samih predavača koji su imali jednu godinu više da unaprede svoje znanje. Schoenfield (2015) ističe da je za bezbednosnu analizu dizajna iskustvo bitan faktor, te je ovo sigurno imalo efekta na kvalitet vežbi. Dodatno, nije moguće isključiti unapređenje kvaliteta kurseva koje su studenti ranije slušali tokom svog školovanja. Iako značajnijih promena (npr. u vidu izmena studijskog programa) nije bilo, nije ispraćeno da li postoje sitnije izmene u okviru predmeta koje bi mogle da utiču na ove rezultate. Što se tiče radnog okvira, bitno ograničenje u njegovoj upotrebi je kompleksnost. Kreator laboratorijskih vežbi ili radionica treba da formira pripremne materijale, konstruiše studiju slučaja i formuliše tok vežbi koji sve to sinhronizuje. Ranjivi softverski paketi mogu u značajnoj meri da redukuju ovaj problem, no ograničenje koje postoji jeste relativno mala količina kvalitetnih alata.

Bitno je istaći da radni okvir za kreiranje materijala za radionice je moguće primeniti u korporativnom okruženju gde će proizvođači softvera moći da konstruišu radionice za učenje bezbednosne analize dizajna koristeći tehnologije i pretnje koje su relevantne za kontekst softvera koji proizvode. Studije slučaja u ovom kontekstu treba da budu softverski proizvodi koje organizacija razvija. Isto tako, univerziteti i fakulteti koji nude kurseve na temu razvoja bezbednog softvera mogu da iskoriste radni okvir da konstruišu sadržaj laboratorijskih vežbi za potrebe učenja bezbednosne analize dizajna. Iako je radni okvir evaluiran na studentima završne godine osnovnih akademskih studija, ne postoje prepreke da se na sličan način radni okvir primeni i u softverskim kompanijama za obuku inženjera. Prema tome, radni okvir se može smatrati pogodnim načinom da se inženjeri softvera efikasno obučavaju kako da sprovode bezbednosnu analizu dizajna.

Poglavlje 4 definiše SATMUS (engl. *Security Analysis and Threat Modeling of User Stories*) proces za bezbednosnu analizu dizajna koji je prikladan agilnom načinu razvoja softvera, pruža opipljive i merljive rezultate, sadrži dovoljno instrukcija za njegovu upotrebu i adaptaciju i proizvodi praktično primenljiv model pretnji koji prati inkrementalan razvoj softvera. Kroz sekciju 4.1 je opisan SATMUS proces, što uključuje aktivnosti od kojih je sastavljen, kao i njegove ulaze i izlaze. U sklopu te sekcije su istaknuti detalji aktivnosti procesa i smernice za njihovo izvršavanje. Cilj SATMUS procesa jeste da otkrije ranjivosti i slabosti u dizajnu razvijenog softvera kako bi definisao korektivne mere.

Da bi podržao inkrementalan razvoj, SATMUS se primenjuje sa svakom korisničkom pričom (engl. *User story*), kratkom izjavom koja definiše šta je to što softver treba da radi i podrži. Kroz korisničke priče, *Scrum* metodologija inkrementalno razrađuje i proširuje softver (Cohn, 2004). U sklopu SATMUS analize, korisnička priča se ispituje kako bi se procenilo kako utiče na bezbednost softvera i da li je potrebno uložiti dodatne resurse kako bi se proizvod sa novim programskim kodom adekvatno zaštitio. Rezultat izvršavanja SATMUS procesa obuhvata:

- Nove korisničke priče koje opisuju dodatan razvoj koji je potreban,
- Dodatne bezbednosne kriterijume prihvatanja ispunjenosti analizirane korisničke priče (Leffingwell, 2010),

- Zahtev za dodatno istraživanje (engl. *Research spike*), kada se otkrije potencijalan bezbednosni problem, ali ne postoji dovoljno znanja kako da se razreši (Knaster i Leffingwell, 2018).

Prvi skup aktivnosti SATMUS procesa se tiče proračuna kritičnosti analizirane korisničke priče sa aspekta bezbednosti. Ovde se određuje da li će razvoj potreban za ispunjenje korisničke priče raditi sa osetljivim resursima i kontekst u kom će se novi programski kod dodati (npr. da li komunicira sa čovekom, da li je blizu izložene površine za napad), na osnovu čega se sprovodi formula za računanje kritičnosti. Spram ovih faktora se definiše nivo kritičnosti korisničke priče sa aspekta bezbednosti.

U slučaju da korisnička priča nije interesantna sa aspekta bezbednosti, odnosno ima nizak nivo kritičnosti, proces se završava uz eventualnu napomenu da je analiza sprovedena.

Za korisničke priče srednje kritičnosti, analizira se model pretnji kroz skup aktivnosti koje predstavljaju bezbednosnu analizu dizajna. Modul se analizira kako bi se proverilo da li postoje značajne promene, poput novih komponenti ili tokova podataka i da li se novi resursi uključuju u postojeći modul. Ukoliko se značajna promena identifikuje, sledeći korak predstavlja analiza pretnji za datu novinu, gde se ističu ranjivosti i scenariji napada koji bi mogli da eksploatišu novi kod. Najzad, definišu se protivmere i zadaci koji treba da reše identifikovani problem i koji se dodaju u spisak zahteva za softver. Rezultati analize se dokumentuju uz pomoć odgovarajućeg alata (npr. *Microsoft Threat Modeling Tool* (Microsoft, 2019)).

Najzad, za korisničke priče visoke kritičnosti, razvojni tim zahteva dodatnu podršku u bezbednosnoj analizi. Ova podrška može doći van organizacije, dovođenjem konsultanta, a može doći i interno, angažovanjem iskusnih inženjera ili domenskih eksperata.

Sekcija 4.2 razmatra šest segmenata SATMUS procesa koji su skloni modifikaciji i adaptaciji za različite organizacije i nivoe bezbednosti koji su zahtevani od softvera koji se proizvodi. Organizacija treba da donese odluke na koji način će konfigurisati svaki od navedenih segmenata prilikom usvajanja SATMUS procesa. Prvi segment se tiče inventara i načina reprezentacije resursa sa kojim razvojni timovi rade i njihovog mapiranja na funkcionalne i bezbednosne zahteve koji diktiraju prioritet zaštite samih resursa. Drugi segment se tiče same formule za računanje kritičnosti korisničke priče koja treba da uvaži resurse koji novi razvoj dotiče, okruženje u kom se nalazi novi kod, kritičnost same funkcionalnosti koju programski kod pruža i druge potencijalne faktore. Pošto formula diktira učestalost bezbednosne analize dizajna, organizacija definiše prioritet ove analize samom konfiguracijom formule. Putem formule, moguće je konfigurisati da timovi koji rade na osetljivim komponentama moraju za svaki razvoj da analiziraju model pretnji, dok će ostali timovi to raditi dosta ređe. Treći segment predstavlja odluku o angažovanju internog formalnog ili neformalnog tima za ekspertizu iz softverske bezbednosti, koji može da pomogne sa analizom korisničkih priča visoke kritičnosti. Povodom bezbednosne analize dizajna, potrebno je da se odabere pogodni metod, kao i da se razmotri upotreba alata i potreba za obukom razvojnih timova kako bi se metod bezbednosne analize dizajna efikasno sprovodio. Ovo predstavlja četvrti i peti segment procesa. Najzad, u sklopu šestog segmenta je neophodno definisati strukture rezultata analize, u vidu šablona za modele pretnji i njihove rezultate, kako bi se ispunili zahtevi koje organizacija ima po ovom pitanju (npr. definisane od strane *IEC 62443-4-1* standarda).

U istoj sekciji se prikazuju i dve relevantne studije slučaja implementacije i adaptacije SATMUS procesa u okviru dva različita okruženja kako bi se istakle dodatne smernice za upotrebu procesa i njegovo prilagođavanje. Prva studija slučaja predstavlja implementaciju procesa u velikoj organizaciji koja razvija softver za industrijski sistem kontrole, gde postoji visok stepen bezbednosnih zahteva. Takođe, istaknuta je implementacija SATMUS procesa za organizaciju koja razvija softver sa relativno niskim bezbednosnim zahtevima. Ova organizacija razvija informacione sisteme za podršku rukovodstvu koji sadrže nešto manji broj osetljivih resursa. Kroz ove studije slučajeva se ističu načini na koje SATMUS proces može da se izmeni i adaptira.

Za evaluaciju SATMUS procesa se koriste dve najčešće tehnike za evaluaciju metoda za bezbednosnu analizu – komparativna analiza i analiza studija slučaja. Sekcija 4.3 razmatra kako SATMUS odgovara na probleme kompleksnosti, nedefinisanog posla i nedostatka smernica u odnosu na druge razmotrene metode bezbednosne analize dizajna. Nakon toga, prikazuje se nekoliko ilustrativnih primera izvršavanja SATMUS procesa, na korisničkim pričama koje dolaze iz konteksta studija slučajeva implementacije SATMUS procesa koje su opisane u sekciji 4.2.

Zahvaljujući svojoj fleksibilnosti, kako u odabiru samog metoda bezbednosne analize dizajna, tako i u konfiguraciji formule za računanje kritičnosti korisničke priče, SATMUS je moguće prilagoditi različitim organizacionim kontekstima. Ova fleksibilnost omogućuje da se definiše tačno onoliko posla oko bezbednosne analize dizajna koliko je potrebno da organizacija ispuni svoje poslovne zahteve i ciljeve. Time što se vremenski ograniči analiza pretnji sav posao je jasno merljiv i opipljiv, te nema poteškoća zbog nedefinisanog posla. Smernice za izvršavanje SATMUS procesa su definisane kroz sekcije 4.1, 4.3.2 i 4.3.3, dok su uputstva za njegovo prilagođavanje istaknuta kroz sekciju 4.2. Deo ograničenja

SATMUS procesa predstavlja njegova delimična složenost, gde je neophodno obučiti razvojne timove, angažovati domenske eksperte, ili iskoristiti alate. Sa druge strane, rezultujuća dokumentacija je onoliko kompleksna koliko se zahteva zbog čega je problem kompleksnosti parcijalno ispunjen.

Najzad, SATMUS se evaluira sprovođenjem analize na pet odabranih korisničkih priča, gde dve dolaze iz konteksta prve organizacije koja razvija industrijski sistem, a tri iz konteksta druge organizacije koja razvija informacione sisteme za podršku rukovodstvu. Kroz ovu evaluaciju se ističe način izvršavanja procesa, gde se za svaku aktivnost opisuje njeno sprovođenje i rezultati.

Villamizar i drugi (2018) u sklopu sistematične studije mapiranja tehnika za inženjering bezbednosnih zahteva u agilnom razvoju ističu nekoliko segmenata ograničenja za ove tehnike. U sklopu sekcije 4.4 se razmatra na koji način SATMUS odgovara na ova ograničenja.

Poglavlje 5 ove disertacije kombinuje radni okvir za konstrukciju radionica sa SATMUS procesom i ističe način na koji ove metode zajedno mogu da omoguće integraciju bezbednosne analize dizajna softvera u proces agilnog razvoja.

U sklopu sekcije 5.1 se opisuju koraci koje organizacija treba da sprovede da integriše tehnike izložene kroz ovu disertaciju u svoje poslovanje i uspostavi njihovo kontinualno unapređenje. Na početku se određuju eksperti koji će uspostaviti ovu novinu, a koji mogu biti eksterno angažovani ili mogu doći i iz organizacije u slučaju kada postoje zaposleni sa adekvatnim poznavanjem softverske bezbednosti. Ovi eksperti biraju studije slučajeva među softverskim projektima organizacije, kako bi našli realistične i relevantne studije slučajeva za obuku zaposlenih. Eksperti potom biraju metod za bezbednosnu analizu dizajna koji se ugrađuje u radni okvir i u SATMUS proces, i vrše dodatne adaptacije SATMUS procesa spram želja vlasnika organizacije i proizvoda. U sklopu radnog okvira za konstrukciju radionica, ekspert formira pripremne materijale i koristeći ranije odabranu studiju slučaja formuliše radionice i njihove tokove.

Nakon konstrukcije radionica neophodno je sprovesti ih, gde će članovi razvojnih timova doći da nauče kako da sprovode bezbednosnu analizu dizajna i SATMUS proces. Kroz ovu aktivnost, eksperti treba da usmere razvojne timove da formiraju modele pretnji za svoje komponente, koje potom mogu koristiti kao polaznu tačku za kasnije izvršavanje SATMUS procesa u svom standardnom razvoju.

Sa ovakvom osnovom, ekspert može da radi na kontinualnom razvoju time što će unapređivati radionice kako bude njihova ekspertiza iz bezbednosti softvera rasla i kako se nove pretnje budu otkrivale. U sklopu ove sekcije su navedene različite strategije za kontinualno unapređenje upotrebe SATMUS procesa i radnog okvira za konstrukciju radionica.

Sekcija 5.2 ističe način na koji SATMUS proces i radni okvir za konstrukciju radionica direktno odgovaraju na zahteve za proces razvoja bezbednog softvera, kao i način na koji su povezani sa ostalim praksama ovog procesa, što uključuje upravljanje, programiranje i testiranje bezbednosti u razvijenom softveru.

Poglavlje 6 zaključuje istraživanje koje je sprovedeno kroz ovu disertaciju. Navedeni su glavni doprinosi koji su proizišli iz disertacije:

- Definisanje radnog okvira za konstrukciju radionica sa ciljem obučavanja inženjera softvera kako da sprovode bezbednosnu analizu dizajna. Ovaj doprinos podrazumeva i same smernice upotrebe radnog okvira, uz primere njegovog izvršavanja.
- Definisanje SATMUS procesa koji proširuje bezbednosnu analizu dizajna da je učini kompatabilnom sa pristupom agilnog razvoja softvera. U sklopu ovog doprinosa ulaze i smernice za izvršavanje procesa i njegovo prilagođavanje različitim kontekstima.
- Definisanje procesa za integraciju bezbednosne analize dizajna u proces agilnog razvoja kroz upotrebu radnog okvira za konstrukciju radionica i SATMUS procesa. Ovo uključuje smernice za inicijalnu postavku bezbednosne analize dizajna i uputstva za kontinualno unapređenje.

Ključne reči: bezbednosna analiza dizajna, modelovanje pretnji, razvoj bezbednog softvera, životni ciklus razvoja bezbednosti, bezbednosna ekspertiza, bezbednost softvera

Dedication

This thesis is dedicated to

My loving partner, without whom life would be a lot less fun and a lot more difficult,

My supervisor, without whose continued support and lessons I would not be where I am today,

My family, for their unconditional love and understanding,

My cat Beljko, for being Beljko.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

In an age when data is more valuable than oil and everything from the physical world has found its digital counterpart, software systems have become the new battlefield. Digital warfare and crime are on the rise, harming individuals, business, and nations. To secure ourselves, we must secure our software, and that is where we come in.

This thesis examines secure software engineering practices and focus on security design analysis, a practice that discovers and mitigates vulnerabilities early in the software's development. While cost-efficient and praised by industry leaders and researchers alike, there exist several issues that limit the adoption of this practice in contemporary agile software development. Throughout this work, we identify and address these issues. We start by describing the general problem area that our work addresses in Section 1.1. Sections 1.2 and 1.3 describe the theoretical background behind the problem, where we examine agile software development methodologies and secure software engineering practices, focusing on security design analysis. In Section 1.4, we specify the exact problem that our work addresses and describe our hypothesis and research goals.

1.1 Problem Area

With the emergence of the Digital Age, malicious groups and individuals have focused their efforts on attacking the cyberspace to further their agenda. These threat agents use sophisticated tooling and know-how to hack computer systems of individuals, organizations, and governments (Gandhi et al., 2011). The impact of cyberattacks has not been negligible. A decade ago, Kshetri (2009) examined the economic impact of cybercrime and concluded that cybercrime was costing businesses and individuals billions of dollars a year. Notable cyberattacks were conducted as part of cyberwarfare, destroying industrial control system equipment (Langner, 2011) and causing power grid blackouts for hundreds of thousands (Case, 2016). More common attacks are conducted by cybercriminals trying to steal personally identifiable information, where major websites such as LinkedIn, Facebook, Twitter, and other have suffered from data breaches (Parwani et al., 2013).

Cyberattackers realize their agendas by issuing attacks that exploit vulnerabilities that a system might have. Organizations have started securing their operations by adopting cybersecurity standards (e.g., the ISO/IEC 27k series (Disterer, 2013), the NIST Cybersecurity Framework (Sedgewick, 2014)) to protect their business and customers from cyberattacks. As computer systems present a broad and often-targeted attack surface through the software they run (ENISA, 2019; Positive Technologies, 2019), organizations are requiring vendors of software used by their system to be constructed with security built-in to avoid the introduction of vulnerabilities to the organization through those products (Salini and Kanmani, 2012).

Software security has traditionally been considered through non-functional or quality requirements, less valuable than functional requirements (Salini and Kanmani, 2012; Türpe and Poller, 2017). However, recent years have seen a rise in security requirements, mandated by security standards and regulations, as well as directly by the customer. These sources often specify security requirements at a high level of abstraction (e.g., "protect sensitive data throughout its lifecycle"), requiring software engineers to derive lower level requirements through security requirements engineering techniques such as threat modeling and misuse cases (Lamsweerde, 2004; Myagmar et al., 2005; Sindre and Opdahl, 2005).

Security requirements engineering is one of several categories of security practices that produce more secure software. Many software vendors have modified their software development lifecycle during the past decade by integrating various security-related practices into the development process. The result is the security development lifecycle (SDL), the most famous of which was published by Microsoft (Howard and Lipner, 2006) at the start of the millennia. It represents a development process that creates demonstrably more secure software. Since then, many software vendors have adopted and adapted the SDL to fit their workflow and organizational needs, as well as the needs of their customers (Geer, 2010; Baca and Carlsson, 2011; Mohamed et al., 2016; Oyetoyan et al., 2016; Weir et al., 2017; Morrison et al., 2017). Furthermore, in the past few years, several standards were issued, that specify process requirements for the secure development of hardware and software products (Ross et al., 2016; IEC, 2018a).

Despite a significant number of SDL initiatives found in literature, as well as the explicit requirement for secure products, adoption rates of the security development lifecycle are still low. A survey conducted by Errata Security in 2010 shows that, out of 46 organizations, only 30% use a formal SDL methodology in their workflow, while 43% do not use any SDL methodology, formal or otherwise (Geer, 2010). More recently, a study was conducted in Malaysia that interviewed software development practitioners about security (Mohamed et al., 2016). The study concluded that, while practitioners are becoming more aware of the importance of security, the SDL practices are immature, and a notable percent of practitioners (19.4%) never receive any security-related training. As software presents a significant and often targeted attack surface, the lack of adoption of the SDL, and the low maturity of existing SDLs is troubling.

The lack of SDL adoption can be explained, in part, due to the nature of state-of-the-art software development practices. Specifically, agile software development (Cockburn, 2002) defines the development workflow of many software vendors today. This set of development methodologies has been criticized for neglecting security requirements (Beznosov and Kruchten, 2004; Ramesh et al., 2010). The first instances of the SDL (Howard and Lipner, 2006; Ayalew et al., 2013) were initially designed for the waterfall development model, though since then several adaptations of this process have been created for agile development methodologies (Baca and Carlsson, 2011; Oyetoyan et al., 2016; Poller et al., 2017). However, the traditional practices that compose the SDL are challenging to integrate into agile development and are often inefficiently practiced (Baca and Carlsson, 2011; Oyetoyan et al., 2016; Türpe and Poller, 2017).

1.2 Agile Software Development

Since its inception, the practice of developing software has proven to be unpredictable. Software developers, while experts in the technology used to build computer systems, often do not correctly understand customer requirements. On the other hand, parties interested in procuring software often have a vague, inaccurate idea of what exactly is the problem that they wish to solve (Wysocki, 2011). This lack of foresight, coupled with miscommunication between the vendor and the buyer, has resulted in a significant percentile of projects developed using traditional software methods to fail to meet customer requirements (Othmane, 2014a).

A new class of development methodologies has been defined during the past two decades to deal with the identified issues. Agile software development has emerged as a practice that emphasizes adaption to changing software requirements. Instead of relying on long-term plans constructed during project inception, developers produce frequent increments of running, tested software features and continuously collaborate with the customer, to make sure the correct requirements are met (Leffingwell, 2010).

Out of all agile development approaches, the Scrum framework is by far the most widespread. According to the 2019 State of Agile survey (CollabNet VersionOne, 2019), pure Scrum is applied by 54% of software vendors that practice agile development. When considering hybrid methods that rely on Scrum (e.g., Scrum with Kanban), this number goes up to 72%.

Scrum is a lightweight framework for team collaboration on complex product development. It introduces roles, events, artifacts, and rules that bind them together while leaving plenty of room for self-organizing to deal with unpredictable and challenging problems (Sutherland and Schwaber, 2011).

We examine the different scrum roles and their responsibilities in Section 1.2.1. We then describe how these roles interact and what the general Scrum workflow looks like in Section 1.2.2.

1.2.1 Scrum Roles

The Scrum team consists of several roles, including the product owner, the Scrum master, and members of the development team. These roles interface with several roles external to the Scrum team, which include the business owner, the stakeholders that benefit from the developed product, and the subject matter experts that assist the Scrum team. Figure 1 illustrates the Scrum team, outlining the core roles of the team and their interaction with roles external to the team.

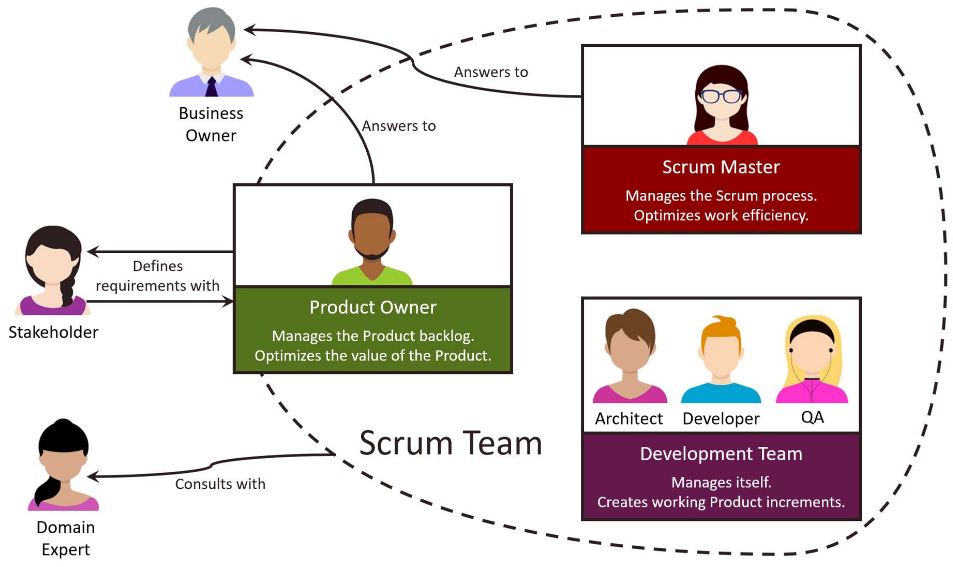
Business Owner
Answers to
Answers to
Scrum Master
Manages the Scrum process.
Optimizes work efficiency.
Defines requirements with
Stakeholder
Product Owner
Manages the Product backlog.
Optimizes the value of the Product.
Architect
Developer
QA
Scrum Team
Development Team
Manages itself.
Creates working Product increments.
Consults with
Domain Expert

Figure 1 Roles in Scrum development

The goal of the product owner is to optimize the value that the developed product brings to the stakeholders. They are responsible for determining and prioritizing user requirements, by communicating and defining them with the relevant stakeholders (e.g., end-users of the software, IT administrators responsible for maintaining and configuring the software). The product owner documents and prioritizes requirements in the product backlog. While not mandatory, the most common way to express these requirements is through user stories (Cohn, 2004). A user story is a brief statement of intent that describes what the software needs to do for the user. The product owner monitors the development team's progress to ensure that the correct vision of the product is being implemented and that the requirements are understood correctly. Multiple product owners can coordinate to specify and deliver a sophisticated product, where each product owner is part of one or more Scrum teams (Knaster and Leffingwell, 2018).

The Scrum master assists the team in practicing Scrum by facilitating a dynamic that optimizes the performance of the team. They are responsible for organizing the Scrum workflow, helping with user story analysis and workload management. For sophisticated products that are built by multiple Scrum teams, the Scrum masters meet periodically, to discuss the progress, issues, and any cross-team coordination that is required to successfully build the product. This meeting is known as Scrum of Scrums (Knaster and Leffingwell, 2018).

The development team includes architects, coders, and testers that design, implement, and test the software described through the user stories. Scrum recognizes no titles for the development team members, regardless of the work being performed. While the size of the team varies, it usually spans from four to seven members. Optimal team size is small enough to remain nimble and large enough to complete significant work (Sutherland and Schwaber, 2011).

The business owner monitors the work results of the Scrum team and provides resources and assistance when necessary. The product owner works with the business owner to determine the priority of up-coming backlog items, process conflicting needs of stakeholders, determine release schedules, and provide resources for the team. Likewise, the business owner consolidates with the Scrum master, to resolve organizational constraints and difficulties, so that the team can work more efficiently (Knaster and Leffingwell, 2018).

The stakeholders present a group of people that have a legitimate interest in the developed product and are the primary driver for developing the product. The goal of the product is to satisfy the needs and desires of the stakeholders (Knaster and Leffingwell, 2018).

Domain experts or subject matter experts contain specialized knowledge or skills that the Scrum team needs to develop their product. Domain experts can have knowledge related to the business, offering insight into the needs of the stakeholders. Furthermore, domain experts can have technical knowledge (e.g., databases, cloud technologies, cybersecurity) that the team requires to fulfill the stakeholder's desires efficiently (Rawsthorne and Shimp, 2011).

1.2.2 Workflow

The Scrum team constructs software iteratively, by producing sets of functionalities in short intervals (usually two to four weeks), called sprints. At the beginning of the sprint is the sprint planning event, where the team selects the user stories that they will implement from the product backlog during this sprint. The result of this activity is the construction of the sprint backlog that defines the goal for this sprint (Sutherland and Schwaber, 2011).

From the sprint backlog, members of the development team complete user stories until there are no more stories or the sprint ends. During this time, user stories are decomposed, understood, implemented, and tested. Throughout the sprint, the team refines the acceptance criteria for each user story, which determine when a user story is finished (Leffingwell, 2010). While the acceptance criteria are story-specific, the definition of done is an artifact that lists additional acceptance criteria for all user stories. This artifact lists tasks that need to be completed to maintain the quality of the product (e.g., regression testing completed, user documentation updated to reflect the new story, and the performance testing benchmark achieved). The definition of done contains business-facing tests that provide quality control (Rawsthorne and Trainer, 2010).

Through daily Scrum meetings, the development team discusses their progress, highlighting work that they completed since the last meeting, their plans until the next meeting, and any potential blockers or issues that are hampering their progress. The goal of these meetings is to increase the productivity of the development team, bring to attention any issues as soon as they appear and continuously monitoring the remaining work to reorganize and plan accordingly (Rawsthorne and Shimp, 2011).

At the end of the sprint is the review. At this stage, the Scrum team and stakeholders discuss the results of the sprint. The development team holds a demo, presenting the implementation of each story. The product owner accepts stories they deem finished while returning incomplete ones to the backlog (Knaster and Leffingwell, 2018).

The last activity is the retrospective, where the team takes the time to reflect on and assess the results of the sprint. During this time, the team determines what went well in the sprint, what could be improved, and what they will improve in the next sprint.

Figure 2 illustrates the general flow of the Scrum framework (Sutherland and Schwaber, 2012).

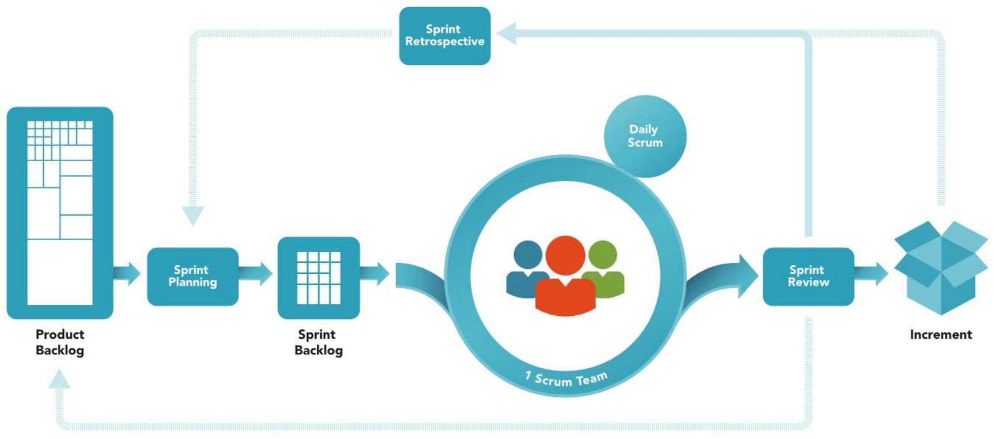

Figure 2 Scrum framework workflow

1.3 Secure Software Engineering

Secure software engineering is the broad domain that covers various practices that serve to enhance the security of the developed software (Devanbu and Stubblebine, 2000). We first examine the modern view of secure software engineering, called the security development lifecycle, in Section 1.3.1. Next, in Section 1.3.2, we place our focus on threat modeling, a high-value practice that is difficult to integrate into agile development. We then explore a particular type of threat modeling, called security design analysis, in Section 1.3.3.

1.3.1 Security Development Lifecycle

The security development lifecycle (SDL) is a business process that defines a set of practices that augment the software development lifecycle (SDLC) to produce demonstrably more secure software. It permeates the whole SDLC, expanding each phase of software development with one or more practices. Microsoft defined an often-cited catalog of practices that make up their SDL (Howard and Lipner, 2006). We summarize the current Microsoft's SDL practices in Table 1. Notably, Microsoft's SDL practices span a wide range of SDLC phases, from requirements engineering, through design and implementation, down to the testing and deployment activities.

Table 1 Summary of Microsoft's SDL practices

Name	Description
Provide Training	Ensure that everyone has basic security awareness and is trained to perform the security activities related to their job role.
Define Security Requirements	Update security requirements to align with the changes in the developed software, regulatory compliance, and the threat landscape.

Define Metrics and Compliance Reporting	Define the acceptable levels of security quality (e.g., through bug bars) and hold development teams accountable to meeting that criteria.
Perform Threat Modeling	Discuss the security of designs in the context of their planned operational environment to more effectively identify security vulnerabilities and prioritize appropriate mitigations.
Establish Design Requirements	Define standard security features that all development teams should use to avoid design flaws.
Define and Use Cryptographic Standards	Ensure vetted and properly configured cryptography is used to protect data.
Manage the Security Risk of Using Third-Party Components	Manage the inventory of third-party components used by the software, monitor disclosures of their vulnerabilities, and create a plan to evaluate and mitigate the reported vulnerabilities.
Use Approved Tools	Maintain a list of approved tools that developers can use.
Perform Static Security Testing	Analyze source code before compiling to validate adherence to the secure coding standard.
Perform Dynamic Security Testing	Perform run-time security verification and validation of the compiled software to test the security of the fully integrated and running code.
Perform Penetration Testing	Uncover vulnerabilities resulting from coding errors, system configuration faults, or other deployment weaknesses.
Establish a Standard Incident Response Process	Prepare to address new threats and vulnerabilities in the deployed software that emerge over time.

The traditional SDLs followed the waterfall development model. Since then, software vendors have, for the most part, reorganized their development business processes to follow various agile development methodologies (Kapitsaki and Christou, 2014). As the SDL is a set of security practices, applying the waterfall-based SDLs to the agile development environment calls for the redistribution of security practices, with or without modification, to the agile workflow. Microsoft has adapted its original SDL for the agile context (Microsoft, 2018). Their agile SDL contains the same security practices as in the waterfall SDL model, distributed in three categories:

- One-Time practices – security practices conducted at the start of a new project;
- Every-Sprint practices – security practices performed in every sprint;
- Bucket practices – security practices that are completed periodically, spread across multiple sprints during the project lifetime.

While Microsoft's SDL is comprehensive, it does not contain all the notable secure development practices found in the literature. Several papers explored and compared different processes for constructing secure software. De Win et al. (2009) compared the OWASP CLASP (which has since evolved into the OWASP Software Assurance Maturity Model Project (OWASP, 2019)), Cigital's Touchpoints (which has since evolved into the Building Security-In Maturity Model (McGraw et al., 2018)), and Microsoft's SDL. They break down the three major secure development processes of the time into practices, providing an extensive catalog of 45 security practices. A similar study was conducted by Baca and Carlsson (2011), who cataloged and compared practices found in Microsoft's SDL, Cigital's Touchpoints, and the Common Criteria (Keblawi and Sullivan, 2006) security engineering processes. After applying the practices to a real-world agile development process, they conducted interviews to determine which practices were cost-efficient. Informed by the discussions, they constructed a hybrid SDL process that contained all the practices from the different source that were deemed to be cost-efficient.

The SDL has become the topic of industry standards, where organizations such as the National Institute of Standards and Technology (NIST) and the International Electrotechnical Commission (IEC) have issued standards that define requirements for a secure product development lifecycle (Ross et al., 2016; IEC, 2018a). Table 2 outlines the different security practices defined in the IEC 62443-4-1:2018 standard (IEC, 2018a). Compared to Microsoft's SDL, the standard puts great emphasis on security management activities and includes the construction of security guidelines for the end-users of the developed product.

Table 2 Summary of the IEC 62443-4-1:2018 secure product development practices

Name	Description
Security management	Build a secure development environment where roles and responsibilities are clearly defined, and security work is appropriately managed.
Specification of security requirements	Prioritized needs that bring value and are focused through the lens of risks, considering regulatory compliance, the threat model, and best practices.
Secure by design	Design layers of reliable defenses, protecting a minimized attack surface of a high-quality product.
Secure implementation	Write a resilient code design with secure constructs that adhere to the secure coding standard.
Security verification and validation testing	Verify adherence to security requirements and validate the quality of the layered protective controls.
Management of security-related issues	Detect, assess, prioritize, and address security-related issues in running products.

Security update management	Construct tested and proven mitigations for new vulnerabilities and deliver them promptly.
Security guidelines	Aid all parties that interact with the product in staying secure.

Regardless of the source, all security development lifecycles span a significant portion of the SDLC, if not the complete process. They entail many significantly different activities. For example, while security management is concerned with monitoring other activities and providing proper education to employees, security testing activities entail the use of specialized tools. Notably, the modern SDL approaches require that all personnel involved in software development perform some security activities, based on their role in the process.

1.3.2 Threat Modeling

A significant pillar of the SDL is threat modeling, which represents the systematic assessment of a software or system design regarding its ability to withstand attacks from adversaries (Shostack, 2014b; Synopsys, 2017). At a high level, threat modeling takes as input the design of a module[1], considers its functional and security requirements, and outputs the necessary changes to the design and additional requirements (e.g., new development effort, requirements for third-party tool integration). Therefore, it is considered both a technique for security requirements engineering and secure design construction (Türpe, 2017). On the one hand, the result of threat modeling is a more secure design, while on the other, this analysis translates one set of requirements (i.e., high-level business requirements and security requirements) into low-level security requirements that developers can understand and fulfill.

We use the term **explicit** security requirement to denote concrete security-related requests for the software, that come from industry standards, regulations, and best practices. Examples of explicit requirements include securing the confidentiality of payment card information, as defined by PCI DSS (PCI, 2018), and the privacy of personal data defined under GDPR (GDPR, 2016). The other type of security requirements we call **quality** requirements. These requirements come from stakeholders and are often more ambiguous. For example, a stakeholder might define that he wants to maintain the integrity of their public website. Threat modeling can then be

[1] We use the term *module* for the target of threat modeling or security design analysis. A module represents a software segment of any size. It can be an enterprise system, a single application, a single component in an application, or a set of functions provided by the software.

employed to decompose this request into actionable work items, such as building controls for anti-Injection and ensuring proper access control for the website files and the application-level functions (Stock et al., 2017). Finally, development teams can identify quality security requirements for technical assets, even when there are no explicit security requirements. Examples of this include the confidentiality of user credentials and cryptographic keys and the availability of infrastructure components and functions. Explicit security requirements are often easily mapped and threat modeling aids in identifying the areas of the software that a requirement addresses. On the other hand, quality security requirements must be threat modeled to determine actionable work items (Türpe and Poller, 2017).

This practice finds design flaws and security issues before they enter the code, making them uncostly to fix (Security Innovation Europe, 2016). An interview of fourteen secure software engineering experts aimed to identify security practices that bring the most value to the organization, where threat modeling was among the top five practices (Weir et al., 2017).

There is some confusion regarding what constitutes threat modeling in software development, that stems from the two often-used interpretations of the term threat (Synopsys, 2017).

The first interpretation is that the threat is an individual or organization from which an attack originates. Threat modeling of these threats entails the analysis of these threat agents to determine their motivation and general ways in which they can accomplish their goals. This analysis includes assessing the attacker's risk tolerance, their capabilities and opportunities, and the resources they might invest in targeting the module under analysis. The analyst can then identify high-risk areas of the developed module with a general idea of where to invest in security and to what extent.

The second interpretation sees threats as potential events with unwelcome consequences. While such events can be broad in scope (e.g., destruction of the system, disclosure of information stored in the datacenter), they support lower levels of detail, where examples of specific threats include:

- Pretending to be a different human when communicating with an application (and obtaining that human's data and privileges);
- Changing the data as it flows from one software service to another;
- Performing a sensitive action and refuting doing so.

By looking at threats as events in the module, it is possible to perform a fine-grain analysis that finds security design flaws and sensitive areas of the module that require more attention. In turn, this form of threat modeling enables the construction of a layered defense in depth for the developed software.

Türpe (2017) examines three dimensions of security requirements and the security requirements engineering methods that combine them. The three dimensions are:

- The security goal, an asset-centric view of security requirements concerned with protecting security properties (i.e., confidentiality, integrity, availability) of assets in the module. This view is common for regulatory compliance, as standards and regulations usually define security requirements through assets, as well as stakeholders, who might not understand security but do know which assets are important to them.
- The threat, an attacker-centric view that defines security requirements as a means of preventing an individual or organization from attacking the module and accomplishing their malicious goals. Stakeholders usually do not understand threats, and the effectiveness of the design against threats is difficult to assess. However, threats can guide and prioritize secure design construction and can be a useful tool for verification tasks.
- The security design, a software-centric view of security requirements, which defines technical security controls that can be traced to business-level requirements. This view of requirements is suitable for developers that are familiar with technical concepts, although eliciting such requirements from stakeholders can be challenging.

Türpe (2017) notes that examining a single dimension is insufficient to define adequate security requirements. However, examining all dimensions is a three-variable problem, where any change in one dimension may entail changes or new questions in the remaining two. The author suggests temporarily fixating a single dimension and focusing on the other two, noting that this is the approach used by security requirements engineering techniques to reduce the complexity. The techniques that arise by combining two dimensions are:

- Risk analysis (security goal and threat), where goal-threat combinations can be ranked by goal importance, expected damage, and the likelihood of an incident to determine the risk and prioritize further investments.
- Design process (security goal and security design), where the security goals are translated into design decisions, by selecting, placing, and configuring security controls around assets.

- Security design analysis (threat and security design), where the threat's behavior in the presence of the module is examined, to identify events which attackers might provoke to bypass existing controls and achieve their goals.

1.3.3 Security Design Analysis

Security design analysis, as defined by Türpe (2017), aligns with the second interpretation of the threat, as a potential event with unwelcome consequences. We base our work around security design analysis (SDA), the second interpretation for threat modeling.

Throughout the rest of this thesis, SDA and threat modeling are used interchangeably to mean the analysis of a module's design to identify events with unwelcome consequences (i.e., threats), their decomposition into attack scenarios that realize them, and the definition of mitigations that resolve them.

Several prominent SDA methodologies are often cited in both scientific and grey literature. These include the STRIDE methodology (Hernan et al., 2006), LINDDUN (Wuyts and Joosen, 2015), PASTA (UcedaVelez and Morana, 2015), and Trike (Saitta et al., 2005). Additionally, both the scientific and grey literature boasts a plethora of other threat modeling techniques, and several surveys were conducted on the topic (Hussain et al., 2014; Ramesh and Reddy, 2016; Tuma et al., 2018). Furthermore, several books were written about threat modeling, offering a practitioner's insight into the intricacies of applying different threat modeling techniques to the real world (Ransome and Misra, 2013; Shostack, 2014b; Schoenfield, 2015). Regardless of the exact methodology, all SDA methods share a common set of high-level steps, as Figure 3 illustrates.

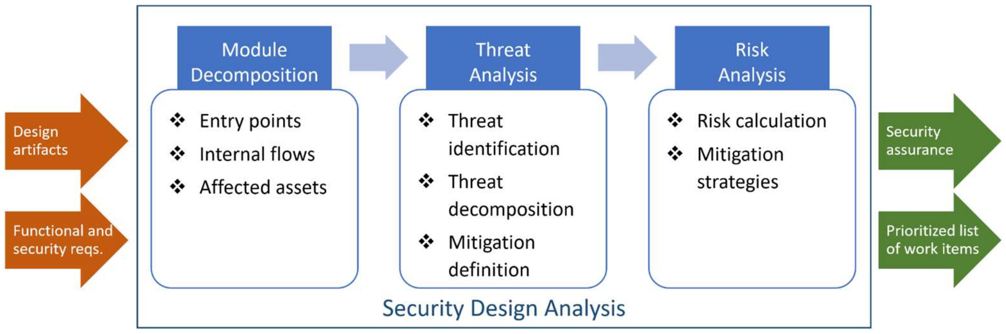

Figure 3 Security design analysis steps

The primary inputs for SDA include a module's design artifacts, which are analyzed against the set of relevant functional and security requirements to assess the security posture of the current design.

The goal of *module decomposition* is to define the scope and understand the target of analysis (Synopsys, 2017). This part of SDA examines the module's entry points, from which an attacker might deploy an attack, as well as the internal data and control flows to understand how attackers might progress through the module to achieve their goal. With an understanding of the module's flows, assets can be mapped to determine the location of the attacker's target in the module.

The next step is *threat analysis*, whose purpose is to identify threats or events that an attacker might cause to work towards their goal. Identified threats are decomposed into specific attacks that can realize the threat. Furthermore, threat analysis entails defining mitigations (either existing or ones that should be introduced) that prevent the attacks and, consequently, the related threats from occurring.

Finally, *risk analysis* examines the identified threats and existing mitigation to determine the risk. Based on the risk and the proposed mitigations, a suitable risk mitigation strategy is selected, which can require design changes, additional development work, or other forms of effort.

The outputs of SDA include a prioritized list of work items that aid product management and development teams to secure the examined module. Additionally, security assurance is produced by performing and documenting SDA, providing evidence that security was addressed during the design of the module.

Different SDA methods vary in how they address each of the three steps described above. For module decomposition, a common approach is to use data flow diagrams to represent the system (Shostack, 2014b; Wuyts and Joosen, 2015). Other methods use UML diagrams to examine potential security issues (Jürjens, 2002; Lodderstedt et al., 2002). The highest variability between methods is found in the threat analysis method. Each approach offers a unique take on how to identify and decompose threats, relying on taxonomies like STRIDE (Hernan et al., 2006) and LINDDUN (Wuyts and Joosen, 2015), utilizing misuse cases (Alexander, 2003; Sindre and Opdahl, 2005), attack trees (Schneier, 1999) or attack pattern libraries (Martin, 2007; Barnum, 2008). Risk analysis can be supported by external risk management methodologies (Ross, 2011) or more straightforward calculations that do not require significant overhead (Shostack, 2014b). Importantly, the core distinction of an SDA method is in the way threat analysis is conducted, while both module decomposition and risk analysis can be swapped between alternatives.

While the work presented in this thesis can support and utilize most SDA methods, we focus our examples and evaluations around the STRIDE SDA method, for several reasons. First, STRIDE is extensively described through a series of articles published by Microsoft (Hernan et al., 2006; Shostack, 2008), as well as a complete book on threat modeling (Shostack, 2014b). Next, the scientific literature is full of papers related to the application of STRIDE to different technologies (e.g., IPv6 (Georgescu et al., 2016), web browser APIs (Dev and Jevitha, 2017), cyber-physical systems (Khan et al., 2017)) and domains (e.g., online banking (Xin and Xiaofang, 2014), telehealth (Abomhara et al., 2015), smart grid (Jelacic et al., 2017)). Finally, the method has been studied and tested for usability and effectiveness (Scandariato et al., 2015), where the authors experimented with 57 students and concluded that STRIDE was not challenging to learn or execute, but it did produce many false negatives and was time-consuming. We describe STRIDE-based threat analysis in more detail to provide the reader with the necessary background for understanding our work.

STRIDE is a tool for threat identification, a mnemonic where each letter represents a threat, specifically:

- Spoofing – The impersonification of identity and subversion of authentication. In the context of software systems, anything from a user to a service, message, file, or machine can be spoofed;
- Tampering – Unauthorized modification of data and loss of integrity. In our context, data can be tampered with at rest (e.g., in a file or database), in transit (e.g., while traveling over a network channel), or during use (e.g., while in active memory or during processing);
- Repudiation – Lack of action accountability and subversion of non-repudiation controls, allowing for denial and hiding of performed actions;
- Information Disclosure – Unauthorized access to data and loss of confidentiality. As with tampering, data can be accessed at rest, in transit, or during use;
- Denial of Service – Partial or complete denial of authorized access to data or systems and loss of availability;
- Elevation of Privilege – The gain of privileged access and subversion of authorization, to gain unauthorized access to information or compromise the system.

The astute reader might notice that spoofing and elevation of privilege lead to threats to confidentiality (information disclosure), integrity (tampering), and possibly availability (denial of service). The reason for this separation lies in non-linear attacks (Gantenbein, 2016), where the attacker performs spoofing and elevation of privilege attacks to "open the door" and gain access to the system components for further attacks.

By expanding the threat view, STRIDE allows for a software-centric approach to threat modeling, where the focus is placed on the software components and their interaction. This approach is more in line with the practice of software development, avoids some of the problems of asset-centric threat modeling, like the ambiguity of assets, and allows for the identification of actionable steps to mitigate threats (Shostack, 2017).

The idea is to map STRIDE threats to elements of a data flow diagram (that were produced as part of module decomposition) following the applicability matrix presented in Table 3.

Table 3 Applicability of STRIDE threats to different data flow diagram elements

Threat / *DFD el.*	*Spoofing*	*Tampering*	*Repudiation*	*Information Disclosure*	*Denial of Service*	*Elevation of Privilege*
Process	X	X	X	X	X	X
Data store		X		X	X	
Data flow		X		X	X	
External Entity	X		X			

For each identified threat, the appropriate decomposition is required, which includes the identification of attacks that can realize the threat, vulnerabilities that enable them, and security controls that mitigate them. While threat identification can be completed without any expert knowledge, threat decomposition requires an interdisciplinary skillset. For this, the threat modeling team needs to be aware of both general and technology-specific security concepts (i.e., attacks, vulnerabilities, controls) to complete this exercise effectively. Finally, the identified threats and mitigations serve as input for risk analysis, which can prioritize further work items, as described above.

1.4 Problem Statement and Hypothesis

Baca et al. (2011) applied practices from Microsoft's SDL to an agile development process practiced in Ericsson AB. They conducted interviews with personnel involved in software development to assess the applicability and effectiveness of the different security practices. The discussions concluded that Microsoft's SDL had a significant adverse effect on the agile development process, criticizing practices such as threat modeling for being too costly to conduct. Similar conclusions were presented by Oyetoyan et al. (2016), where Microsoft's Agile SDL was criticized for being too unwieldy for the agile context. Traditional SDA is conducted during the early design phase. Incremental development approaches, such as Scrum, can drastically change the software's design throughout its development, making the results of the initial security design analysis obsolete. Therefore, some form of incremental threat modeling is required to compensate for this, but this is not possible if threat modeling is too costly to conduct.

Security design analysis is challenging to integrate into contemporary software development methodologies. One issue arises from the inherent complexity of SDA. As it is both difficult to learn and teach (Shostack, 2014b; Schoenfield, 2015), Scrum teams often do not possess enough security knowledge to perform it effectively (Morrison et al., 2017). Schoenfield (2015) points out that apprenticeship programs, where an experienced security analyst teaches interns the art of SDA, are a viable way to train someone in performing quality SDA. However, this is a slow process that is hard to scale, which is why a more efficient solution is required for training Scrum teams on how to perform SDA.

Poller et al. (2017) identified a different set of problems with security engineering processes such as Microsoft's SDL and Cigital's Touchpoints, concerning agile development. The authors conducted a long-term empirical study in which they examined the organizational impact that a security consultation had on a large, multinational software vendors' development process. While the security assessment and workshops conducted by the consultants resulted in an initial rise of awareness and enthusiasm for developing secure software, after two months most of the initiative has faded, and no lasting consequences of the security consultation were observed. The authors identified several key factors that influenced the outcome. A major issue is unaccountability of work, where security work is intangible and not part of the binding agreement between the managers and development teams. Finally, security engineering processes, such as Microsoft's SDL and Citigtal's Touchpoints, do not offer appropriate guidance for organizations and developers on how to adapt the different practices to their workflow.

As presented in (Luburić et al., 2018a), there exist several applicability issues that hamper the efficient practice of security requirements engineering, including SDA, in Scrum development:

- Complexity – If the security analysis technique introduces new types of documentation, additional job roles, or requires much training to practice effectively, it works against agility and therefore cannot complement agile development;

- Unaccountability – If security is intangible and not part of the binding agreement between the managers and development teams, it will not be done as it cannot be accounted for;
- Lack of Guidance – If the security analysis technique does not offer appropriate guidance, which includes examples of usage and tailoring considerations, it will require significant effort to introduce to an organization.

Based on this, we define the problem through the following research questions:

RQ 1. *How to efficiently train Scrum teams to perform security design analysis?*

RQ 2. *How to efficiently integrate security design analysis into Scrum development, providing the appropriate security assurance and visibility of security work?*

Based on these questions, we define the hypothesis around which the thesis is based. It can be summarized as follows:

- *It is possible for a Scrum team to practice security design analysis throughout a software's development lifecycle, assuring that sufficient security is built into the software solution, provided that:*
    - *Adequate training is provided to the Scrum team to perform security design analysis efficiently.*
    - *The security design analysis is compatible with the Scrum development process, does not require the introduction of new roles to the team, and does not mandate the construction of heavyweight documentation.*
    - *Security work is tangible and can be planned and prioritized like any other work item.*
    - *Enough guidance and knowledge exist to adopt, use, and adapt the method to a specific organization's context.*

From this hypothesis, we derive the primary goals of the proposed research, where the expected results include:

1) The construction of a framework for developing training laboratories for software engineers to effectively learn SDA. The SDA Training Framework addresses the first research question and is the topic of Chapter 3.
2) The definition of a process around SDA that sufficiently addresses the applicability issues defined in (Luburić et al., 2018a) and provides assurance that enough security is built into the developed software.

This method addresses the second research question and is the topic of Chapter 4.

3) The utilization of the SDA Training Framework to integrate, support, and continuously improve the SDA process, defined under the second goal. When combined, these two methods support agile software development and comply with the requirements for SDA established by standard-defined security development lifecycles.

To fulfill the Guidance requirement, the results of the proposed research will include:

- Demonstrations of the SDA Training Framework execution, to develop labs for training different aspects of SDA and instructions on how best to utilize the framework.
- Guidelines for tailoring the SDA method to organizations with varying security requirements for their software.
- Instructions on how to utilize the SDA Training Framework to first introduce the SDA method to an organization and then to continuously improve its execution.
- Explanations about how both the SDA method and training framework integrate with other SDL practices, and how they answer the requirements imposed by SDL industry standards, such as IEC 62443-4-1: Secure product development lifecycle (IEC, 2018a).

1.5 Thesis Structure

Throughout this introductory Chapter, we defined the problems that this thesis addresses and presented the necessary background to anchor our work. Here we outline the rest of the thesis.

Chapter 2 presents the literature review, where we examine methods for integrating SDA into agile development. We analyze teaching methods suitable for training software engineers SDA to address our first research question. We further examine existing SDA techniques and requirements for such techniques coming from both industry standards and the agile workflow to address the second research question.

Chapter 3 details our SDA Training Framework, used for teaching software engineers on how to conduct SDA. The framework constructs educational materials and assignments for training workshops. Here we also demonstrate the use of the framework and its evaluation in an undergraduate university course setting.

Chapter 4 describes our Security Analysis and Threat Modeling of User Stories (SATMUS) process. SATMUS is a method for SDA that is compatible with agile software development practices and provides accountability of work. Here we provide guidance for method execution and tailoring and present two case study implementations of SATMUS.

Chapter 5 combines the work presented in Chapters 3 and 4 and demonstrates how to integrate SDA into agile development by using the SDA Training Framework and the SATMUS process. Here we also examine how our work interacts with practices from well-established security development lifecycle approaches.

Chapter 6 concludes our work and presents opportunities for further research and development.

2 Research Review

In this Chapter, we present the results of our research reviews aimed at understanding how to effectively train software engineers the practice of SDA, as well as how to adapt the SDA to make it suitable for the agile development process, addressing the applicability issues discussed earlier.

To answer the first part of the hypothesis defined in Section 1.4 and the research question related to defining adequate training for SDA, we surveyed the literature for methods used to teach secure software engineering in general, and SDA in particular. Section 2.1 presents the results of our literature review. To address the remainder of the hypothesis and the second research question, we examined the literature for security design analysis methods that were specifically designed for agile development methodologies, as well as a few industry standards which address this topic. The results of this research review are presented in Section 2.2.

2.1 Training Approaches for Secure Software Engineering

As the traditional classroom is considered to train engineers the practice of security design analysis inefficiently, we examine the literature to find alternative teaching methods that could be used to achieve this learning objective. This research was initially published in (Luburić et al., 2019a) and is expanded here.

Section 2.1.1 describes case study analysis, a teaching method that fosters critical thinking and is suitable for inductive reasoning. Next, we examine the use of games in education in Section 2.1.2, where several publications address security design analysis through gamification. In Section 2.1.3, we explore the hybrid flipped classroom, a combination of the traditional and flipped classroom that promises to develop the security skills and expertise required for SDA. Section 2.1.4 presents teaching approaches that benefit from e-learning. Here we select parts of the literature reviews published in our previous work (Luburić et al., 2016, 2019b) and discuss the use of specialized tools for teaching different secure software engineering concepts. Finally, in Section 2.1.5, we summarize the results of our literature review and highlight the primary influences for the work presented in Chapter 3.

2.1.1 Case Study Analysis

Research was conducted under the National Science Foundation project "*Developing case studies for information assurance education*" (NSF, 2008). As a result of the project, 12 case studies were created to be used in information assurance and risk management courses. Several publications were released that present the use of these case studies in different courses, where the impact on student learning was assessed. In general, the results showed that the use of the case study method was effective and that it enhanced learning.

Varma and Garg (2005) discuss how different conventional and non-conventional teaching methods do not achieve quality learning outcomes in the field of software engineering. They declare case study analysis as an effective teaching technique for software engineering, where complexity and intricacies related to the field of software engineering can only fully be experienced by examining a realistic case study.

Meneely and Lucidi (2013) examine real-world vulnerabilities as case studies. They introduce the Vulnerability of the Day, where during the first 10 minutes of each class, the teaching staff demonstrates a vulnerability through live code examples. The class discusses implications of the vulnerability, how to exploit it, the negative impact it can have on the system, and what are the appropriate mitigations. While no formal evaluation of the effectiveness of the teaching method was performed, the authors note the overwhelming interest of the students for this activity, measured through questionnaires.

Case study analysis immerses students in realistic situations, where they must deal with incomplete information, conflicting goals, and time (Andersen and Schiano, 2014). The class discussion that emerges during case study analysis stimulates the development of students' critical thinking skill. As many students are more inductive than deductive reasoners, they learn better from examples than from logical assertions that build upon basic principles (Dunne and Brooks, 2004). The use of case studies can, therefore, be a very effective classroom technique. According to (Dunne and Brooks, 2004), the case study should fulfill the following requirements:

- It represents a general issue beyond the case itself, tells an engaging story, and focuses on an interest-arousing, controversial issue, where it poses a problem that has no obvious right answer;
- It requires the reader to use both the presented information in the case study, as well as critical and analytical thinking, to address the problem, and
- It has just enough information for proper analysis and is relevant to the students.

2.1.2 Gamification

Gamification has gained some traction when it comes to teaching cybersecurity. Over the years, several card games were produced to facilitate cybersecurity education, with many focusing on SDA. A group of researchers constructed a card game called Control-Alt-Hack, which acts as a light-weight learning tool that raises awareness about cybersecurity and offers teaching staff a light-hearted way to talk about threats, attacks, and countermeasures (Denning et al., 2013). They evaluate their approach by distributing copies of the game to a dozen information security courses, along with a survey to be filled by the teaching staff. The results of the questionnaire show that the game is well received and accomplishes the goal of facilitating interactive cybersecurity education.

Williams et al. (2010) invented a security risk assessment game for agile development called Protection Poker. The agile team plays Protection Poker for every product development iteration, as part of a dedicated planning meeting. They identify and rank the security risks of each feature that is planned for development in the upcoming iteration. Through this activity, it is possible to identify additional security mechanisms that must be implemented to maintain an acceptable risk level across the product. The authors conducted a feasibility study with undergraduate students to determine the method's practicality. Tøndel et al. (2018) observe the use of Protection Poker in Norwegian companies and determine the benefits and challenges of adopting it. They praise Protection Poker for raising security awareness and facilitating a discussion around the subject. However, the authors note that the discussion must be guided and that the time it takes to play can be significant. Furthermore, the game does not provide a mechanism for validating the quality of the output.

Taking a similar approach, Shostack produced the Elevation of Privilege card game intended to teach developers at Microsoft the craft of SDA (Shostack, 2014a). Through his research, Shostack identified that the primary challenge to efficient SDA is the lack of intuition when it comes to determining threats, attack vectors, and security controls on a real system. Shostack notes that implementing security features is usually only slightly more challenging than implementing any software feature. However, understanding where an attacker might strike or how an asset might be compromised is something that alludes many software developers. Elevation of Privilege was explicitly designed to teach cybersecurity in an enticing, supportive, and non-threatening way. No experimental evaluation is performed to test the efficiency of Elevation of Privilege.

One novel approach to developing the intuition that Shostack mentions was proposed by Kohno and Johnson (2011). In their work, the authors concur that the students need to attain a mindset focused on the broader societal and contextual issues surrounding information security. They use science fiction prototyping to stimulate such thinking, where students are asked to research about cutting-edge technologies, extrapolate their development to the near future, and imagine threats, vulnerabilities, attacks, and controls related to these future systems. While the authors note the usefulness of science fiction prototyping, no experiment nor evaluation is presented.

Krutz et al. (2015) utilize a role-playing game in the classroom. Students have to design a secure system, where one student acts as a malicious insider whose goal is to produce a flawed design, which can allow an attacker to infiltrate the system once it is created. This activity pins the students against each other without knowing who the insider is, stimulating a fun environment in which the students learn about threat analysis by identifying attacks and countermeasures. No experiment was conducted to test the effectiveness of this method, but student satisfaction was rated using a questionnaire based on a Likert scale.

2.1.3 Hybrid Flipped Classroom

Recently, Carranza and DeCusatis (2015) critiqued the conventional approaches employed in cybersecurity education, both at universities and in industry-certified programs. The authors recognized a tendency to emphasize memorization of facts over a more in-depth cognitive understanding of the subject. They propose the use of the flipped classroom model to teach cybersecurity, where students are expected to complete weekly reading assignments, after which they discuss the subject matter with the teaching staff through consultations. Furthermore, the authors examine a variant of the flipped classroom, called the hybrid flipped classroom. Here, students additionally attend group lectures to gain a different view from the textbook on complex topics like encryption and public key infrastructure. Once again, no formal evaluation was conducted to test the efficiency of the hybrid flipped classroom.

Researchers from the Anderson School of Management examined cybersecurity training initiatives and awareness campaigns held in corporations (Kassicieh et al., 2015). The article examines different types of cybersecurity training and awareness methodologies and tools. The authors conclude that the current cybersecurity training and awareness programs are limited in their efficacy and list several ways in which this can be improved, including the use of the flipped classroom.

2.1.4 E-Learning

E-learning is concerned with effective multimedia learning using information and communication technology (Mayer and Moreno, 1998). At its purest form, e-learning can be supported with the use of a PowerPoint presentation or a video describing a specific subject matter.

More elaborate uses of e-learning entail the use of software created for the sole purpose of teaching a specific subject matter. The papers (Dios et al., 2008; Lovejoy and Wickert, 2015) point out the importance of using software tailored for a course in laboratory exercises. By using the Moodle learning management system and Maple software, which is a heavyweight toolset for supporting mathematical and technical courses, a cryptography course held in the Salamanca University, Spain, was presented by Dios et al. (2008). Lovejoy and Wickert (2015) show how a signals and systems course greatly benefited from using Python and, specifically, the IPython notebook, which is a computational notebook that combines images, formulas, text and interactive code snippets. Both papers noted improvements in the educational experience of their students, owing to the use of specialized software.

A specialized set of e-learning tools, called vulnerable software packages (VSP), are used for learning about attacks, vulnerabilities, and defenses related to software systems. Pohl et al., (2015) present BREW, a VSP designed to teach students how to find and exploit vulnerabilities as an attacker and to subsequently identify the issues in the code and resolve them as a defender. The authors define different educational usage scenarios, offering guidance on how to integrate BREW into different classrooms. Furthermore, they list settings where BREW has been successfully integrated into lectures and lab exercise.

Walden (2008) utilizes the OWASP WebGoat VSP to teach web security attacks and vulnerabilities, focusing on SQL injection attacks. He created several labs, where they attacked WebGoat manually and with the aid of security testing tools. The OWASP WebGoat is a maintained VSP, currently standing as a medium-level project in the OWASP organization. It provides an impressive array of challenges, presented as a collection of exercises.

2.1.5 Literature Review Results

During our experience with teaching students the security design analysis, we found that even students who did not attend laboratory exercises had little trouble learning how to use and implement security controls once they knew which specific controls to implement. However, the question of when a security control is needed, when to use a specific security control, and how it could be bypassed had alluded many students, even the ones that attended every class, signifying the lack of critical thinking and the in-depth understanding of the subject. These findings fall in line with the conclusions made by Carranza and DeCusatis (2015), as well as Shostack (2014a). Based on this, we decided that the hybrid flipped classroom, as described in Section 2.1.3, would be suitable for our context. We gave reading materials for students to learn on their own what specific security controls exist and how to use them while using the laboratory exercise to put more emphasis on recognizing when to use them. To facilitate the development of critical thinking, we utilized the case study analysis technique, as described in Section 2.1.1.

The most notable research that inspired our new course design and initial version of the training framework includes (Carranza and DeCusatis, 2015) and (Kassicieh et al., 2015) for the hybrid flipped classroom, as well as the articles related to the use of case studies in the classroom, especially those developed under the NSF project (NSF, 2008; Andersen i Schiano, 2014). Other examined literature did affect our overall design, and we have expanded our framework to utilize both gamification, as described in Section 2.1.2, and an e-learning platform in the form of a vulnerable software package, as presented in Section 2.1.4. By utilizing these methods, we created an SDA Training Framework, which is described in Chapter 3.

2.2 Agile-based Security Design Analysis Methods

In this Section, we examine the scientific literature of the past decade for security analysis techniques designed to identify security requirements and increase the security posture of software products developed following the agile development methodologies.

First, we present a review of security requirements engineering (SRE) methods based on security design analysis. We examine how the proposed methods address the applicability issues of Complexity, Accountability, and Lack of Guidance, as defined in the problem statement in Section 1.4. The results, initially published in (Luburić et al., 2018a), are presented in Section 2.2.1. Next, we examine several papers that, through literature review or case study analysis, identify additional issues when introducing SDA to agile development. We present these conclusions in Section 2.2.2. In Section 2.2.3, we analyze several industry standards that define requirements for the secure software development lifecycle, paying particular attention to the requirements that directly address security design analysis. Finally, in Section 2.2.4, we compile the results of our literature review and draw conclusions which guide the construction of our SDA method.

2.2.1 Applicability Issues in Agile Security Requirements Engineering

We analyze a recent systematic mapping study on the topic of security in agile requirements engineering (Villamizar et al., 2018), from which we extract methods applicable to the Scrum framework, excluding other agile development techniques. We further examine the literature for any additional relevant methodologies.

We analyze each method to determine the extent to which it has issues of Complexity, Unaccountability, and Lack of Guidance. We do this by rating each technique on how well it fulfills the following requirements:

1. Simplicity – The security analysis technique should not mandate the introduction of additional types of documentation or job roles. Additionally, the method should require as little training as possible to practice effectively. This requirement answers the Complexity issue.
2. Accountability – The security analysis technique should be integrated into the standard agile development workflow and produce visible and quantifiable action items. This requirement answers the Unaccountability issue.
3. Guidance – The security analysis technique should be fully documented, offering illustrative examples of its use, as well as advice for integration into different real-world contexts. This requirement addresses the Lack of Guidance issue and partially Unaccountability, as the execution of ambiguous methods cannot be sufficiently accounted for.

Peeters (2005) introduces the concept of abuser stories – a user story that describes how a threat agent can achieve a goal that compromises the system or its assets. Abuser stories represent a skeleton for the threat model, but lack adequate requirement traceability to provide security assurance. Regarding Simplicity, abuser stories introduce a simple and intuitive artifact, while not requiring any new roles. Training needs are not discussed. Accountability is not addressed, as it is both unclear when sufficient abuser stories have been defined and what development work needs to be done to resolve the abuser story. While the original paper is concise and does not offer much regarding Guidance, abuser stories have been around for over a decade, and several papers have utilized this concept and offered some illustrative examples (Tondel et al., 2008; Mellado et al., 2010).

Baca et al. (2015) present an extended Security-Enhanced Agile Software Development Process (SEAP), which was practiced in Ericsson AB. The method introduces four roles in the agile development process, including the security manager, security architect, security master, and penetration tester. Activities conducted by the penetration tester are out of the scope of our context. Regarding Simplicity, SEAP mandates the introduction of several new roles to the organization, most prominently the security master who expands the Scrum team. As a result, the development team does not require any training, as the security master performs the security assessment. The paper does not discuss the documentation resulting from the analysis. Regarding Accountability, the process outlines the activities that need to be conducted by the development team. Finally, regarding Guidance, the only resources provided by the authors is the paper that documents the SEAP flow, practiced in Ericsson AB, which lists roles and their responsibilities in the process. No examples of use or tailoring guidelines are presented.

Pohl and Hof (2015) describe Secure Scrum, where they expand user stories with S-Tags – descriptions of security concerns related to one or more backlog items. S-Tags represent security-related effort (e.g., in the form of new user stories, specialized testing, research). During the implementation of a user story, all related S-Tags need to be present in the sprint backlog, where the definition of done states that verification needs to make sure that all present S-Tags are resolved. Regarding Simplicity, Secure Scrum defines an additional, although simple, documentation type and no new roles. Training requirements are not discussed. Defining security-related effort in the backlog in the form of an S-Tag helps achieve Accountability. Regarding Guidance, it is unclear how to implement Secure Scrum in an organization efficiently. For example, the generalized nature of S-Tags as "anything security-related" is susceptible to threat explosion (Tuma et al., 2017), where developers can virtually indefinitely populate the backlog with new S-Tags. This raises an issue with Accountability, as it is unclear how to identify S-Tags, and therefore determine when this activity is complete. Finally, there is no guidance on how to tailor the process for different contexts, as well as no illustrative examples to show the process in action.

Azham et al. (2011) introduce a security analysis process for agile development based around a new document, the security backlog. The security backlog is managed by a new role, the security master, who processes the product backlog to identify security concerns with user stories, which are then entered into the security backlog. Regarding Simplicity, the security backlog approach introduces a new document structure, a new role, as well as a process which is disjoint from the development process. As the security master manages the security backlog, the development team does not require additional training. Accountability is achieved as the security master processes security-related concerns into user stories, where the developers are required to perform only development tasks. Regarding Guidance, while the general flow of the security backlog management process is present, the details are not documented. In a later paper (Ghani et al., 2014), the authors showcase an illustrative example of the method used in an industry-based case study. There is no process tailoring guidance.

Mougouei et al. (2013) present S-Scrum, an expansion to the Scrum framework that introduces three types of security-related research spikes. The first type of the research spike is conducted after release planning, where a security analysis is performed on the backlog, potentially introducing new items into the backlog. Then, the next spike is issued for security modeling, to incorporate the results of the security analysis into the software design. The final type of research spike entails a detailed security analysis for each sprint. Regarding Simplicity, the method does not introduce new documents or roles, while training needs are not discussed. However, S-Scrum offers little regarding Accountability, as it is unclear when the proposed research spikes are considered done. Furthermore, there is very little Guidance provided by the paper, as the research spikes are vague in their purpose and offer no detailed description of how to efficiently conduct them. Additionally, the article provides no illustrative examples and no tailoring guidance.

Singhal (2011) describes the Agile Security Framework. This framework is presented as an end-to-end security development lifecycle for agile development, covering SRE and other activities (e.g., penetration testing). We focus on the SRE activities of the framework, where abuser stories, attack trees, and threat modeling is utilized to discover threats and attacks, and form requirements which mitigate them. Regarding Simplicity, the method introduces several artifacts (e.g., abuser stories, attack trees) and mentions a new role (i.e., security expert). The author recommends dedicated security training. Accountability is achieved through a method that decomposes abuser stories to make sure that relevant threats are identified, and appropriate mitigations are planned, which results in the construction of acceptance tests. Guidance is partially achieved, as the paper presents an extensive description of the method, coupled with illustrative examples of use, but no guidance is provided for tailoring the technique to different contexts.

In (Othmane et al., 2014a, 2014b), the authors examine how to construct secure software incrementally. They introduce security assurance cases and present an expansion to the agile development process that entails the construction of these documents. Security assurance cases are structures that provide evidence that the developed software is acceptably secure and constructing such a body of evidence requires appropriate threat modeling and security requirement analysis. It should be noted that the papers place greater emphasis on the end product of SRE (i.e., the security assurance case) and not the SRE method itself. Regarding Simplicity, the paper introduces a new document type that needs to be constructed, reviewed, and maintained. Furthermore, the article mentions the security expert role during the process of creating the security assurance case and notes that security training is required for developers. Concerning Accountability, the security assurance case is a well-structured body of evidence that can be directly mapped to work that needs to be realized. Finally, enough Guidance is given for executing all the tasks related to security assurance case construction, and two illustrative examples are presented in the papers. No tailoring guidance is offered.

Rindell et al. (2015) present an expanded Scrum framework (called VAHTI-Scrum) that is meant to be compliant with the Finnish security standard collection (VAHTI). As such, it is concerned with security throughout the whole software development lifecycle, where we once again focus on the SRE activities. Regarding Simplicity, VAHTI-Scrum introduces a new role to each Scrum team, the security developer, responsible for security reviews, security test cases, and other security-relevant work. The Scrum master is required to have substantial security knowledge. The method introduces threat modeling and application risk analysis, although it is unclear when these activities should be conducted. The process is further expanded by adding “sprint zero” for security analysis and periodic hardening sprints to enhance the security of the software. The method addresses Accountability by relying on the expertise of the security developer when completing well-known tasks such as threat modeling. While the paper explains the roles and responsibilities, it offers only minor tailoring Guidance and does not present any illustrative examples.

Table 4 summarizes the extent to which each SRE method has achieved the goals of Simplicity, Accountability, and Guidance.

Table 4 Agile SRE methods in relation to Simplicity, Accountability, and Guidance

Method	Simplicity	Accountability	Guidance
Abuser Stories (Peeters, 2005)	Full	Partial	Partial
SEAP (Baca et al., 2015)	Partial	Full	Insufficient
Secure Scrum (Pohl and Hof, 2015)	Full	Partial	Insufficient
Security Backlog (Azham et al., 2011)	Partial	Full	Partial
S-Scrum (Mougouei et al., 2013)	Full	Insufficient	Insufficient

Agile Sec. Framework (Singhal, 2011)	Insufficient	Full	Partial
Sec. assurance case (Othmane et al., 2014a)	Insufficient	Full	Partial
VAHTI-Scrum (Rindell et al., 2015)	Insufficient	Full	Partial

From these results, we draw several conclusions:

1. Guidance is insufficiently addressed. Three out of eight methods are presented in a single paper and offer little more than basic illustrative examples. No paper provides sufficient guidelines for tailoring the technique to different organizational contexts.
2. Full Accountability is only achieved when a dedicated security expert exists in the process. However, most such methods are too complicated and fail to address the Simplicity requirement.
3. Methods that fulfill the Simplicity requirement do not explain when the security analysis is done, which is why they can only partially satisfy the Accountability requirement. Furthermore, simple methods often lack proper Guidance, hinting at their lack of maturity.

2.2.2 Additional Issues with Agile Security Design Analysis

In (Tuma et al., 2018), the authors conducted a systematic literature review for security design analysis techniques, where they examined 26 methodologies to determine their inputs, outputs, and internal workings. Furthermore, they analyze the current state of their adoption in contemporary software engineering trends, such as agile and DevOps, and discuss obstacles for their adoption. The authors present several applicability issues that impede the adoption of SDA in agile development and list recommendations to solve them, which include:

1. Insufficient use of automatization
    a. There is little support for traceability between the discovered threats and the implemented code that can ensure that any change to the code that introduces a vulnerability does not go unnoticed.
    b. There is little support for composition of SDA results, where in practice the software systems are too large to be analyzed all at once, which is why SDA is performed on subsystems or components of the complete software.
    c. Time-consuming steps of the SDA activity, such as impact analysis, are not offloaded to tools.

2. Lack of Definition of Done
    a. It is up to the analyst to determine which part of the analyzed software should be decomposed to discover critical threats.
    b. It is up to the analyst to determine which threats are relevant and when all the critical threats have been discovered.
    c. It is up to the analyst to determine the abstraction level and what constitutes a unit of analysis.
3. Lack of guidance
    a. Most of the methods do not precisely define all the steps of execution.
    b. Most of the tools that support an SDA method do not have a proper manual.

Galvez and Gurses (2018) explore the challenges and opportunities of introducing privacy threat modeling to agile development of service-oriented architectures. Privacy threat modeling entails discovering, addressing, and validating threats that affect the realization of privacy goals. The authors conclude that agile development enables effective, iterative analysis and resolution of complicated problems, at the expense of comprehensive end-to-end design analysis. They identify 21 challenges, some of which arise from agile development and others from service-oriented architectures. The challenges related to agile development can roughly be grouped into the following categories:

1. Changing environment
    a. Maintaining an up-to-date threat model is difficult in an environment that emphasizes working software over documentation.
    b. High-level threat models lack important details and can lead to vague privacy threats and requirements, while low-level threat models focus on a single component and can miss critical privacy threats and requirements that affect multiple components.
    c. Frequent changes in the software can introduce, change, or remove threats and keeping the relevant threat list up-to-date is difficult in an environment where both threats and software evolve.
2. Lack of expertise and guidance
    a. Deriving threats and requirements from threat modeling and prioritizing them can be time-consuming.

    b. Customers may not possess enough domain knowledge to elaborate on the impact of low-level privacy requirements.
    c. Threat catalogs are limited, and developers may focus on unrealistic threats and attack scenarios, where analyzing realistic attack scenarios requires much creativity.

Finally, Cruzes et al. (2018) elaborate on the challenges and experiences of performing security design analysis in the agile development process of a specific software vendor. They perform action research (Davison et al., 2004) by facilitating the adoption of STRIDE-based security design analysis (Shostack, 2014b) to the software vendor's development process. The authors observe SDA sessions conducted by the development teams and conducting interviews with members of the teams to discover the main challenges and ways in which the SDA method can be adapted to suit the agile workflow better. The result is a list of 21 challenges, from which we derive the following conclusions and list the challenges that support them:

1. Development teams shy away from producing SDA documentation, especially when they do not perceive it as valuable.
    a. Teams did not document the list of assets relevant to the components they were developing as they did not see the value in this activity.
    b. It was challenging to motivate the teams to document the data flow diagrams required for threat analysis, as this was perceived as a time-consuming activity.
    c. Teams did not want to update the data flow diagrams frequently, as they could not determine a suitable moment for this activity.
    d. Teams did not want to follow up on threats and document them in more detail after the initial threat modeling meeting.
2. Development teams lack the expertise and guidance necessary for efficient SDA.
    a. Many discussions around threats and mitigation strategies got lost, signaling a lack of structure and guidance.
    b. It was hard to decide the right level of abstraction for the DFDs, as high-level diagrams lacked essential details, while low-level diagrams lead to less effective meetings.
    c. It was hard to determine when enough analysis was done, and when all the critical threats were identified and decomposed.

d. The threat modeling meetings were not productive, causing frustration among team members.

e. There was a need for a security expert to run the meeting, which was a resource that most teams did not have.

f. The outputs of the threat modeling sessions were a list of concerns and threats which were not actual work items.

2.2.3 Relevant Industry Standards

We explore the "*IEC 62443, Security for industrial automation and control systems - Part 4-1: Secure product development lifecycle requirements*" standard (IEC, 2018a). As SDA is concerned with security requirements engineering and constructing a secure design, we examine the parts of the standard that address security requirements and secure design.

The purpose of the security requirements specification practice, as defined by IEC (2018a), is to document the security capabilities that are required for a product along with the security capabilities expected of its production environment. The first part of this practice entails the definition of the product security context, a set of security-related assumptions about the environment in which the product will operate, which guide the definition and prioritization of threats and security requirements for the product. The second part of this practice requires the construction and maintenance of a threat model of the product, detailing its data flows, stores, and processes, as well as threats and mitigations. The rest of the practice is concerned with specifying and documenting explicit security requirements, which can affect SDA but are not an integral part of it.

The secure by design practice (IEC, 2018a) entails the construction of a secure software design with multiple layers of defenses following the defense in depth principle. This applies to both conceptual and detailed design for the developed product. This practice is directly concerned with security design analysis, where the goal is to develop and document a secure design that identifies and protects all external and internal interfaces that cross a trust boundary. The SDA is further guided by the defense in depth principle, as well as secure design best practices like attack surface reduction, least privilege, and economy of mechanisms.

Next, we examine the "*NIST SP 800-160 vol. 1, Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems*" (Ross et al., 2016). This publication provides a basis for secure system engineering, describing the various security activities, principles, and concepts that integrate into the system development lifecycle. Here we are concerned with the technical processes that address the construction of a secure architecture and its translation into a secure design.

The architecture definition process described by Ross et al. (2016) defines a set of security viewpoints of the system's architecture based on the stakeholders' security concerns. Through threat modeling, candidate architectures are constructed, each addressing the security concerns in different ways to inform risk assessment and management, where the goal is to select a candidate architecture that optimizes security against other requirements. The architecture is then mapped to the design definition process, where security design analysis is employed at the subsystem and component level to ensure the construction and maintenance of a secure design.

2.2.4 Literature Review Results

From the conclusions listed in Sections 2.2.1 and 2.2.2, we derive several guiding principles for developing an SDA method that is compatible with agile development:

- High complexity contradicts agile development principles. Organizations that favor agility will not adopt complicated SDA methods.
- Some security expertise is required to achieve Accountability of security analysis work. Dedicated personnel or training for the Scrum team is mandated. Furthermore, security analysis must be time-slotted to provide a stopping condition when the analysis is considered done.
- Organizations need Guidance to adopt an SDA method, which includes a detailed explanation of the technique, illustrative examples of its use, and tailoring guidance for adapting the SDA to their context.

We follow these principles to define a security design analysis method. Furthermore, we consider requirements for security requirements engineering and secure design construction practices, as presented in Section 2.2.3, to construct a method that can be used to address this regulatory compliance. The resulting SDA method is presented in Chapter 4.

3 SDA Training Framework

This Chapter details our solution for the first research question and the first segment of the hypothesis listed in Section 1.4, related to providing adequate training to software engineers on the topic of security design analysis. The proposed solution is the SDA Training Framework, which guides the construction of educational materials and their composition into laboratory exercises. The framework's core revolves around two teaching methods:

(1) the hybrid flipped classroom (Carranza and DeCusatis, 2015), and

(2) case study analysis (NSF, 2008).

The resulting lab exercises address some of the issues described in Section 2.1. The framework and its evaluation are published in (Luburić et al., 2019a) and expanded through two additional papers (Luburić et al., 2019b, 2019c).

Section 3.1 outlines the main components of the framework, highlighting its inputs, outputs, and internal workings. Here we describe the process of using the framework to compose the laboratory exercises. Section 3.2 demonstrates the application of the framework, where we formulate the laboratory exercises using the framework, to provide guidance on how to use it. Here we also illustrate how different tools, such as industry standards and vulnerable software packages, can be utilized with the framework. Section 3.3 explains the comparative analysis which we have conducted, to evaluate if the exercises conceptualized using the framework achieve better learning outcomes than lab exercises which utilize the traditional teaching method. Finally, Section 3.4 discusses the results of our analysis, as well as the limitations and implications of the work presented in this Chapter.

3.1 Framework Structure

This Section starts with an overview of the training framework, presenting its main components in Section 3.1.1. Next, in Section 3.1.2, we discuss the framework usage process to describe its internal workings that, when executed, formulate laboratory exercises for teaching SDA.

3.1.1 Framework Overview

As described in (Luburić et al., 2019a), the proposed framework consists of four parts:

(1) the applied security design analysis method,

(2) the preparatory materials containing enough information, as determined by the lab constructor, for the lab participants to actively participate in the analysis,
(3) one or more case studies which are targets of SDA, and
(4) the laboratory exercises constructed by using the framework.

The input for the framework is the *SDA method*, which is the primary learning goal. The selected SDA method guides the design of the laboratory exercises. A single workshop might cover a specific SDA method, while methods targeting a broad domain may span several courses. Sophisticated methods need to be decomposed into subactivities to fit the format of a workshop or lab exercise.

The *preparatory materials* are the materials that lab participants need to examine before the lab exercise, the core pillar of the hybrid flipped classroom technique (Carranza and DeCusatis, 2015). This can be anything from reading materials (e.g., book chapters, scientific articles, blog posts) to videos (e.g., conference presentations, online course segments). In the context of security design analysis, these materials should detail vulnerabilities, attacks, and security controls relevant for the examined SDA or its subactivity.

Following the case study analysis technique (NSF, 2008), the *case study* is the target of the SDA and represents a module[2]. The size and complexity of the case study dictate the effort required for a complete analysis. A single case study might be enough to cover all subactivities of an SDA, while multiple case studies may be examined to cover the SDA in-depth or reinforce the learning goals. The selection and presentation of the case study must be carefully considered to maximize trainee engagement.

Finally, *laboratory exercises* are the main output of the framework. Each lab exercise contains:

(1) a set of preparatory materials, to facilitate the hybrid flipped classroom,
(2) a description of one or more case studies (which might be reused in multiple lab exercises), to support case study analysis,
(3) an outline of the expected flow for that training session, and

[2] As mentioned earlier, the term *module* is a generic term which can represent software of any size. A module can be an enterprise system, a single application, a single component in an application, or a set of functions provided by the software.

(4) guidelines for the trainers on how to apply the SDA on the case study, using the information described in the preparatory materials, to support them in their job.

Following the method of the hybrid flipped classroom, trainees go over the preparatory materials before the lab exercise. At the start of the exercise, the trainer facilitates a discussion around the preparatory materials, to ensure the group has understood its content. Next, the trainer presents the case study, providing sufficient information for the group to perform SDA. During the exercise, the trainer guides the trainees in performing SDA on the relevant segments of the case study by utilizing what they have learned from the preparatory materials and expanding upon that base. The session concludes when the trainer summarizes the performed work and highlights the most significant learning outcomes of the exercise.

3.1.2 Framework Usage

The usage of the framework is a process that takes as input the components listed in the previous Section, as well as the constraints regarding the number of slots for laboratory exercises and the length of each slot and outputs a set of laboratory exercises. Figure 4 illustrates the process.

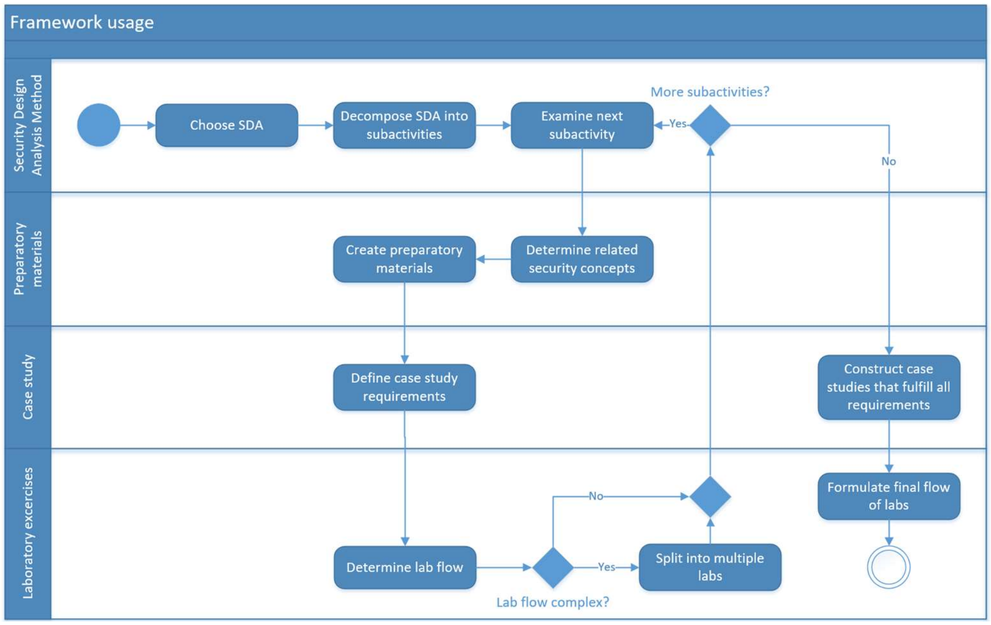
Framework usage
Security Design Analysis Method
Preparatory materials
Case study
Laboratory excercises
Choose SDA
Decompose SDA into subactivities
Examine next subactivity
More subactivities?
Yes
No
Determine related security concepts
Create preparatory materials
Define case study requirements
Construct case studies that fulfill all requirements
Formulate final flow of labs
Determine lab flow
Lab flow complex?
Yes
No
Split into multiple labs

Figure 4 Usage of the training framework to construct laboratory exercises

The first step of this process is to *choose the SDA* method and set it as the learning goal of the lab exercises. Examples of SDA methods are discussed in Section 1.3.3 and include STRIDE (Shostack, 2014b), LINDDUN (Wuyts and Joosen, 2015), PASTA (UcedaVelez and Morana, 2015), Trike (Saitta et al., 2005). The next step entails *decomposing the SDA activity into subactivities*. The decomposition of the SDA method should consider the complexity of the method, and the time limitations (i.e., the number of available lab exercises, the duration of each exercise). As described in Section 1.3.3, most SDA methods have three general steps – module decomposition, threat analysis, and risk analysis, where the primary focus of the method is on the threat analysis step. Therefore, most subactivities should be related to threat analysis (where each subactivity is concerned with a subset of relevant threats, attacks, and mitigations), while module decomposition and risk analysis might entail a single subactivity each. While the resulting set of the SDA subactivities does not require a one-to-one mapping on the final set of constructed lab exercises, the trainer should strive towards this goal at the start, to simplify the following framework activities.

Each subactivity is analyzed (*Examine next subactivity*) to *determine the related security concepts* (i.e., attacks, vulnerabilities, security controls). The trainer *creates the set of preparatory materials* which cover the identified security concepts. These should contain information that is easy to consume while being relevant for the assignments and discussions of the lab exercise. The construction of preparatory materials can be a vast undertaking if the subject matter is complicated and if there are no readily available materials (e.g., publicly available lectures, articles, tools).

Next, the trainer elicits requirements which the case study must fulfill to be relevant for the SDA subactivity and identified security concepts (*Define case study requirements*). For example, if a subactivity of an SDA examines threats of loss of confidential data, then the case study needs to work with sensitive data. Requirements for such a case study might be that:

(1) it processes sensitive data,
(2) it transports sensitive data over internal networks,
(3) it provides (a subset of) sensitive data through Internet-facing services, and
(4) it stores sensitive data in several different types of data stores.

The breadth of the case study requirements is limited only by the time limit of the lab exercise. At this point, the case study is not conceptualized, as requirements from multiple SDA subactivities (and lab exercises which address them) might be grouped to form a single case study when all case study requirements are known.

At this stage, the trainer formulates the general flow of the lab exercise (*Determine lab flow*). By focusing on the SDA subactivity, while keeping the known aspects of the case study and preparatory materials in mind, the trainer decomposes the learning goal into assignments and tasks which need to be completed during the lab exercise. Tasks might include presenting the case study (conducted by the trainer), performing security design analysis on the case study (conducted by the trainees) or discussing the content of the preparatory materials (conducted by both the trainer and the trainees). By mapping these tasks to a timeline, the trainer constructs the general flow of the lab exercise. At this point, the lab exercise might be too complicated for the given time frame, in which case it could be *split into multiple labs* (if there is room to accommodate this).

This loop repeats until the trainer creates preparatory materials, case study requirements, and general lab exercise flows for each subactivity of the SDA method. At this stage, the trainer compiles the list of requirements for the case studies to define the actual case studies for the lab exercises (*Construct case studies that fulfill all requirements*). The number of case studies can range from one to many, where one case study can be examined during one or multiple lab exercises, while one or multiple case studies can be examined during a single lab exercise. Apart from the elicited requirements, each case study should fulfill several global requirements, including:

(1) being a believable representation of a real system, as this increases trainee engagement (Luburić et al., 2019b), and
(2) being understandable to the trainees, so that they can quickly grasp the scope of the case study and focus their mental effort on the learning objective.

Furthermore, each case study needs to be documented so that lab participants can familiarize themselves with it, either through preparatory materials or during the lab exercise. This documentation can take many forms (e.g., video, white paper, presentation) and the trainer should select the medium which maximizes usability and trainee engagement.

Finally, once the case study set is defined, the trainer *formulates the final flow for all lab exercises*. During this step, lab exercises can be merged, further divided, or omitted based on the constraints of the working environment (e.g., equipment, time).

3.2 Framework Application

We utilized the framework to formulate laboratory exercises for a university course concerned with secure software engineering. We construct six laboratory exercises (Luburić et al., 2019a) and integrated them into our course, some of which were reworked and expanded during the next two school years (Luburić et al., 2019b, 2019c).

This Section illustrates the application of the training framework, where we create a set of laboratory exercises to provide guidance for the framework's use. First, in Section 3.2.1, we present the application published in (Luburić et al., 2019a), where we select an SDA method, decompose it into subactivities, present the case study and an outline of the content of the preparatory materials. Next, in Section 3.2.2, we illustrate a single lab exercise in detail and demonstrate how an industry standard can be used as preparatory material, as published in (Luburić et al., 2019c), to offer further guidance for the framework's use. Finally, in Section 3.2.3, we describe how the framework can utilize vulnerable software packages to quickly construct lab exercises, as published in (Luburić et al., 2019b), both reducing the complexity of the lab construction, and increasing the engagement of the trainees.

3.2.1 Generation of Multiple Lab Exercises

Following the process depicted in Figure 4, we first *chose the SDA method*[3]. From the methods examined in Section 1.3.3, we selected the traditional STRIDE-based SDA method (Shostack, 2014b), as it is popularly used and is suitable for web-based enterprise information systems with which our trainees (i.e., students) are already familiar.

The next step entails the *decomposition of the SDA into subactivities*. We initially divided the selected SDA into six subactivities, which are listed in Table 5. Here we also *identified some of the relevant security concepts* which served as input for the construction of preparatory materials. Initially, we decomposed this SDA method into eight subactivities (module decomposition, one subactivity for each letter of STRIDE, and risk analysis). As module decomposition is a software engineering activity (and does not require security knowledge) we concluded that a single lab was enough to introduce concepts such as data flow diagrams, so we did not further decompose this subactivity. When examining the threat analysis subactivities, we decided to focus on cryptography as a means of mitigating Information disclosure and Tampering, so we merged these subactivities. Likewise, we assessed that Spoofing and Repudiation could be grouped due to their relation to authentication. Finally, while risk analysis can be sophisticated, making it suitable for further decomposition, we focused on the basic variant of this activity recommended by the STRIDE SDA method (Shostack, 2014b).

[3] The use of italic in this Section signals that this is an activity illustrated in Figure 4.

Table 5 Defined Subactivities of the Chosen SDA Method

Subactivity	Description
Decomposing the module	The case study is introduced. Subsystems, actors, assets, entry points, and data flows are identified. The goal is to define relevant threat agents and construct data flow diagrams of the module under analysis.
Threat analysis: ***Information disclosure, Tampering***	Threats related to information disclosure and tampering are identified and decomposed for the target module. Focus is placed on attacks and vulnerabilities mitigated by cryptographic controls.
Threat analysis: ***Denial of service***	Threats related to denial of service are identified and decomposed for the target module. Focus is placed on attacks and vulnerabilities mitigated by network segmentation, high availability design, and DDoS protection controls.
Threat analysis: ***Spoofing, Repudiation***	Threats related to spoofing and repudiation are identified and decomposed for the target module. Focus is placed on attacks and vulnerabilities mitigated by authentication and logging controls.
Threat analysis: ***Elevation of privilege***	Threats related to the elevation of privilege are identified and decomposed for the target module. Focus is placed on attacks and vulnerabilities mitigated by access control and input validation controls.
Risk analysis	Security requirements for the selected case study are examined to determine the impact of each threat. Basic risk calculation is performed to determine a prioritized list of security controls.

We *created preparatory materials* which cover cybersecurity concepts (i.e., attacks, vulnerabilities, countermeasures) concerning web-based information systems. The concepts covered in the materials are grouped based on the related subactivity, as demonstrated in Table 6.

Table 6 Preparatory Materials Related to SDA Subactivities Listed in Table 5

Subactivity	Description
Decomposing the module	Presentation describing the case study and white paper detailing data flow diagrams and their graphical elements.
Threat analysis: ***Information disclosure, Tampering***	White paper and lecture covering applied cryptography - symmetric ciphers, asymmetric ciphers, hash functions PKI, digital signatures, TLS, and select attacks against these mechanisms.
Threat analysis: ***Denial of service***	White paper covering high availability design, performance counters, denial of service attacks and mitigations.
Threat analysis: ***Spoofing, Repudiation***	White paper covering authentication controls, session management, logging, and spoofing attacks.
Threat analysis: ***Elevation of privilege***	White paper covering role-based access control, access control lists, input validation, and injection attacks.
Risk analysis	Summary of risk management from (Ross, 2011).

Based on the subactivities listed in Table 5, and the security concepts identified in Table 6, we *constructed a set of requirements for the case study*, which are presented in Table 7.

Table 7 Case Study Requirements Related to SDA Subactivities Listed in Table 5

Subactivity	Description
Decomposing the module	The system needs to be web-based, consist of multiple applications which (to some extent) interact, contain several data stores (e.g., file system, databases) and service several different user categories (e.g., insiders, outsiders).
Threat analysis: ***Information disclosure, Tampering***	The system needs to process, transmit, and store sensitive data, such as PII and user passwords.
Threat analysis: ***Denial of service***	Some part of the system should require high availability. The system should communicate over the Internet.
Threat analysis: ***Spoofing, Repudiation***	The system should require multi-factor authentication for some group of users. The system should require service-to-service authentication. The system should offer sensitive functions that demand accountability.
Threat analysis: ***Elevation of privilege***	The system needs to have some form of shared user interface to demonstrate access control. The system needs to have OS-level assets (i.e. files) that require access control. The system needs to have some forms of data parsers or interpreters (e.g., SQL database, XML parser).
Risk analysis	The system needs to have some form of security requirements derived from standards and regulations.

At this stage, we conceptualized a set of laboratory exercises and *determined their flow*. During this, we *divide the second subactivity* (Threat analysis: Information disclosure, Tampering) into two lab exercises, as we assessed that the topics covered by this subactivity (cryptographic primitives and applied cryptography) required more time to go through based on our prior experience with the traditional classroom. Furthermore, we did not construct a separate lab exercise for the third subactivity (Threat analysis: Denial of service), as we considered it, for the most part, out of the scope of our lecture plan.

With most of the work done, we started *constructing the case study*. We formed a description of a hospital information system (HIS) by examining scientific articles and grey literature related to applications of technology to the healthcare domain. The HIS was chosen as a suitable and highly relevant case study for the security assessment. First, it can be quite sophisticated, servicing a wide array of different actors, which makes choosing a representative subset of functionality that much easier. There have been several papers and articles that call for, examine, and present technological innovation in the field of healthcare (Hamine et al., 2015; Spanakis et al., 2016). As people live longer, more attention is directed at Smart Health systems that optimize the healthcare industry and reduce costs. The second reason the security analysis of such a system is suitable for our needs is the fact that hospitals deal with sensitive data. Health records have been a major target of cybercriminals, but there is also a wide array of sensitive data common to most business systems, like personally identifiable information, financial records, and system user credentials. There are a few articles in the literature calling for and proposing different security measures to protect these systems (Appari and Johnson, 2010).

Our HIS case study is imagined to be deployed on a set of machines inside the hospital's data center. The system interacts with several different actors, each chosen to present a set of attacks. This includes:

- Hospital management, which consists of many staff members who handle human resources, finances, hospital equipment, and operating room schedules. This part of the system falls in the domain of business informatics. Managers interact with the system through a web application, from their workstations, which are located inside the hospital.
- Medical staff, including physicians, technicians, and other relevant subjects concerned with patient management. Physicians use the system to examine their schedule, follow their patient's treatments and health records, communicate with their patients as well as with the management, and so on. Like hospital management, the medical staff interacts with the system inside the corporate network, using a web application;
- Patients, who use the system to follow their hospital appointments, recommended diet and therapy, and treatment history. They interact with the system over the internet, using a mobile or web application. Following the current trends in technological development, a patient can have several wearables or implanted devices, which monitor the patient's physiological parameters and send them to the HIS;
- The government, which periodically contacts the hospital to get statistical data. The government service sends a request over the internet and uses the retrieved data to monitor the health of its citizens, detect early signs of epidemics or high volumes of a particular type of illness.

Finally, we *constructed the final lab flow*, the outline of which is as follows:

(1) Decomposing the module: The HIS is introduced and examined as a real-world system that supports healthcare institutions. The system's purpose is discussed, as well as its actors, functions, data flows,

subsystems, and entry points. Initial security requirements are discussed, and attackers are determined.

(2) Threat analysis (Information disclosure, Tampering) 1: Threats related to the loss of confidentiality and integrity of the data processed by the HIS are examined. Sensitive data assets, such as financial information, patient information, and user credentials, are discovered, and protective mechanisms are proposed using cryptographic primitives. Cryptographic keys are identified as sensitive assets that require further protection.

(3) Threat analysis (Information disclosure, Tampering) 2: Controls proposed in the previous lab exercise are enhanced using mechanisms such as PKI and TLS. Certificate verification is examined, and attacks related to these technologies, such as certificate pinning and attacks against TLS, are discussed.

(4) Threat analysis (Spoofing, Repudiation): Authentication and logging controls and attacks are examined in the context of the HIS. User interfaces and services that require authentication are determined, and appropriate controls are selected. Sensitive actions of the HIS are determined, and logging controls that achieve non-repudiation are designed.

(5) Threat analysis (Elevation of privilege): Access control and injection attacks are examined in the context of the HIS. Entry points that accept external input are identified, and the optimal input validation controls are selected. An access control matrix that defines all user groups, roles, and permissions for the HIS is constructed.

(6) Risk analysis: The final lab is spent discussing security requirements and risk analysis. Regulations such as HIPAA (United States, 2004) are examined in the context of the HIS, and a basic risk analysis method is presented and performed to determine the critical vulnerabilities that need to be patched and the security controls that need to be applied to the HIS.

We utilized these laboratory exercises during the 2016/2017 iteration of the secure software engineering course.

3.2.2 Detailed Lab Construction

To provide further guidance and resolve remaining ambiguity, this Section analyzes one of the labs in detail and present all the artifacts selected and created through the training framework. The lab in question is an evolution of the fifth lab (Threat analysis: Spoofing, Repudiation), which we split into two labs for the subsequent year. This Section focuses on the SDA subactivity that addresses repudiation threats, as described in (Luburić et al., 2019c).

The main vulnerability that enables repudiation threats in software systems is missing or poorly designed logging mechanisms (Shostack, 2014b). Log files contain entries that track the events of a system. On the one hand, they offer insight into a system's (mis)behavior, aiding software engineers in debugging issues. On the other hand, they offer non-repudiation by recording user actions. While the concept of an event logger is simple, correctly implementing logging controls throughout the system to achieve non-repudiation can be challenging (IEC, 2018b).

Preparatory materials

Recently, the IEC has released a standard describing technical security requirements for industrial automation and control systems (IEC, 2018b), detailing many security controls and component requirements (CR), including a logging mechanism for non-repudiation. We use this document as a basis for our preparatory materials and from it derive the following requirements for the logging mechanism (the related requirements from the standard are noted in the braces):

(1) Completeness – Each log entry needs to contain enough data to prove non-repudiation of an action (CR 2.8) and each event for which non-repudiation is required needs to be logged (CR 2.12).

(2) Reliability – Logging needs to be reliable, which is achieved by ensuring the availability of the mechanism (CR 2.9, CR 2.10) and the integrity of the log files (CR 3.9, CR 6.1).

(3) Accuracy – Log entries across the system need to state their creation time precisely (CR 2.11).

Apart from the requirements derived directly from the standard, we add two requirements that improve the efficiency of the logging mechanism:

(4) Usability – The logging mechanism needs to be designed so that security-relevant events (e.g., those that provide non-repudiation) can be easily extracted from the log files.

(5) Minimalism – The logging mechanism should create a minimal amount of log entries needed to serve its purpose, to avoid cluttering the log files.

As log files contain system events that are used primarily for debugging, we need to make sure that security events are not buried and lost due to a large amount of non-security events. Based on these requirements, we construct a three-page white paper to serve as preparatory materials for the lab exercise.

The document explains the danger of repudiation, illustrates it through real-world examples, and describes the motivation behind it. The paper concludes by explaining event logging and details the requirements for an efficient and secure logging mechanism.

Case Study

Audit records need to be generated for access control, request errors, critical system events, backup and restore events, configuration changes, and audit log events, as noted in IEC (2018b), CR 2.8. Furthermore, CR 2.10 defines additional activities that require logging, including performing system actions, creating or changing information, and sending messages.

Based on this list, we conclude that any software system that interacts with human users and has some sensitive assets can be used as a case study. As the SDA subactivity and relevant security concepts do not impose significant limitations for our case study selection, we can utilize the hospital information system (HIS) described earlier. The HIS contains a wide array of sensitive assets, including health data and PII.

Lab Flow

Figure 5 illustrates the flow of the lab exercise, where the arrows originating from the trainees and trainer signify who drives the activity.

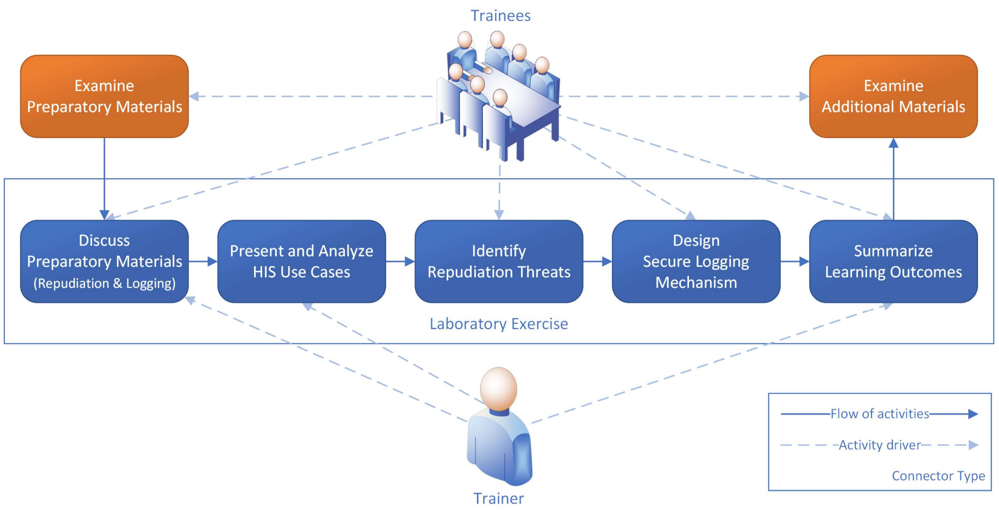
Trainees
Examine Preparatory Materials
Examine Additional Materials
Discuss Preparatory Materials (Repudiation & Logging)
Present and Analyze HIS Use Cases
Identify Repudiation Threats
Design Secure Logging Mechanism
Summarize Learning Outcomes
Laboratory Exercise
Trainer
Flow of activities
Activity driver
Connector Type

Figure 5 Flow of laboratory exercise dedicated to SDA for repudiation threats

The trainees are required to go over the white paper describing repudiation and logging before attending the lab exercise. At the start of the lab exercise, the trainer conducts a brief discussion with the trainees to summarize the main points of the preparatory materials.

After the initial recap, the trainer presents use case diagrams for the HIS case study, focusing on the user's interaction with the system. The trainer takes care to introduce the main points of the system that need to be protected, without making them obvious. This information is masked with irrelevant information and low priority assets. However, care is taken not to bloat the presentation too much, to avoid loss of interest from the trainees. The presentation concludes with data flow and deployment diagrams of the HIS, which were developed during earlier labs.

Next, the trainees perform SDA to discover repudiation threats and try to find actions that a user might have reason to rebut. They identify interfaces between the human users and the software and discuss where and how the actions need to be logged. The goal of this discussion is to fulfill the *Completeness* requirement of the logging mechanisms, as well as obtain an understanding that logs can be generated at different levels of the software system (e.g., operating system, web server, application software).

Once most of the system events requiring non-repudiation have been mapped, the trainees expand the data flow diagrams with log data stores. At this point, the trainer directs the discussion towards the *Reliability* requirement, examining how the logging mechanism can be protected from tampering and denial of service. Scenarios that detail attacks are discussed, and the trainees determine appropriate security controls and design changes to protect the logging mechanisms.

The trainer addresses the final security requirement, *Accuracy*, by explaining how the network time protocol and GPS time synchronization protocols (Mills, 2016) can be used to create system-wide time synchronization. The design of ACME's system is expanded with these controls, and their security is discussed.

Finally, the software engineering requirements of *Usability* and *Minimalism* are addressed. The trainer divides the trainees into teams and asks them to design an application logging mechanism that can answer the following user stories:

> "*As a data protection officer, I want to quickly examine all access requests to GDPR (GDPR, 2016) related data, so that I can examine if there is an anomaly in the system's behavior.*"

> "*As a reliability engineer, I want to quickly examine all mission-critical function calls, so that I can monitor performance to prevent a denial of service.*"

> "*As a software engineer, I want to examine log entries when an error occurs in a system, so that I can triage the bug and resolve the issue.*"

At the end of this exercise, each team presents their design and argues how it can fulfill the listed user stories. All trainees take this opportunity to discuss the pros and cons of each approach. At the end of the lab, the trainer summarizes the main learning points of the lab and offers additional exercises and reading materials.

3.2.3 Introducing Vulnerable Software to the SDA Training Framework

A significant limitation of the training framework is its complexity. The trainer prepares the labs by constructing preparatory materials, the case study, assignments for the trainees, and then putting it all together. To address this issue, we created an enhancement to the training framework, published in (Luburić et al., 2019b).

We utilize publicly available vulnerable software packages (VSPs) to offload some of the work required from the lab constructor. Vulnerable software packages offer software engineers, security auditors, and penetration testers a playground to practice software security skills, both from the attacker's and defender's perspective and for this reason, are often used in training programs (Yuan et al., 2016; Siles and Bennets, 2019). Additionally, these packages are used to test the efficiency of hacking tools, such as vulnerability scanners (Esposito et al., 2018) as well as security controls which mitigate the vulnerabilities of the software package (Pupo et al., 2018).

We propose that the vulnerable software package can be used as a case study and the target of security design analysis. Once requirements for the case study are gathered, the lab constructor searches for a suitable VSP instead of manually constructing a case study. Furthermore, when determining the lab flow, assignments for the lab trainees can be derived directly from the features of the VSP.

To illustrate the enhancement to the framework and provide low-level guidance, we demonstrate its usage by constructing a lab with the learning objective of examining, understanding the impact of, and mitigating elevation of privilege threats through injection attacks and vulnerabilities (Stock et al., 2017).

Preparatory materials

The preparatory materials for this lab are created by utilizing the latest OWASP Top Ten list (Stock et al., 2017). This list is an authoritative document that presents a broad consensus regarding the most common and most critical security risks for web applications. It is updated and published every several years to keep up with the shifting threat landscape.

The OWASP Top Ten list provides a thorough overview of each class of security risks, offering insight into the risks impact, examples of attacks, pointers for vulnerability discovery, and mitigation planning. For the lab in question, trainees examine the entries which cover elevation of privilege threats realized through some form of injection attacks. They examine the following categories before attending the lab:

- A1 – Injection, as the primary subject matter of the lab.
- A4 – XML External Entity, due to its similarities with XML injection attacks from A1.
- A7 – Cross-Site Scripting, which can be seen as an injection attack aimed at the browser's command interpreter.

Furthermore, trainees are required to find and examine at least one example of each of the listed attacks (e.g., SQL injection, LDAP injection, Stored XSS) to get a sense of what the attack vectors look like and how they might be deployed.

Case Study

Guided by the learning objective and preparatory materials, we go over the list of different attacks and vulnerabilities and define the following requirements for a suitable case study:

- It should provide a web user interface to demonstrate cross-site scripting issues.
- It should have an SQL database where at least one command sent from the application is dependent on user-supplied input, to explain SQL injection issues.
- It should process XML documents supplied by external entities, to demonstrate XML injection issues.
- (Optional) It should provide functionality suitable for showing additional injection issues (e.g., OS command injection, LDAP query injection).
- (Optional) It should be built using modern technologies and ideally those directly utilized by the trainees, to increase the perceived relevance and their engagement.

We selected the OWASP Juice Shop (Kimminich, 2019) VSP as a suitable case study that fulfills both the mandatory and optional requirements listed above. The OWASP Juice Shop has several benefits when compared to other VSPs:

- It is mature, categorized as a Flagship project by the OWASP organization, signifying its value to the field of application security. Furthermore, it has previously been utilized in the classroom (Yuan et al., 2016; Kimminich, 2019), as a tool for testing the efficiency of a hacking tool (Esposito et al., 2018) and a security defense (Pupo et al., 2018).
- It is rich with content, covering a wide array of attacks and defenses and containing 74 challenges as of version 8.3, released in January 2019.
- It is easy to use, offering detailed documentation, presentation, and video material to aid with its use and a companion guide which details each challenge and its solution (Kimminich, 2019).
- It is built on a technological stack (Angular, Node.js, SQLite) familiar to our trainees.
- It is a fully functioning, realistic web shop, offering browsing and shopping functionalities similar to applications which the trainees built on earlier courses.

Lab Flow

The lab constructor utilizes the OWASP Juice Shop companion guide to select challenges related to the learning objective and marks them as assignments for the lab. With these assignments, the final lab flow can be created, as Figure 6 illustrates.

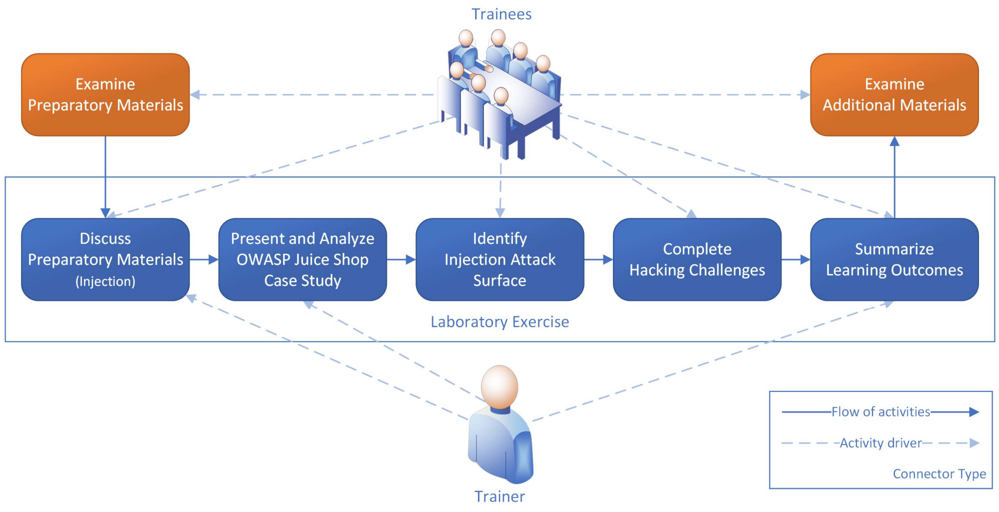
Trainees
Examine Preparatory Materials
Examine Additional Materials
Discuss Preparatory Materials (Injection)
Present and Analyze OWASP Juice Shop Case Study
Identify Injection Attack Surface
Complete Hacking Challenges
Summarize Learning Outcomes
Laboratory Exercise
Trainer
Flow of activities
Activity driver
Connector Type

Figure 6 Flow of laboratory exercise examining injection attacks with the aid of a VSP

At the start of the lab, the trainer goes over the different injection attacks and answers any questions which the trainees might have.

Next, the trainer introduces the OWASP Juice Shop and demonstrates the basic functionality of the application, after which the architecture and data flow diagrams of the module are presented (available at (Kimminich, 2019)).

The first assignment for the trainees is to identify the attack surface where injection attacks might be deployed, as part of a group discussion.

Once most of the attack surface is discovered, the trainees are given a list of challenges from the companion guide to complete on the laboratory computers.

At the end of the lab, the trainer highlights the learning objective, summarizes the activities conducted during the lab, and notes the important takeaways. During this discussion, the participants:

- Define the impact of the attacks they performed;
- Determine which vulnerabilities exist in the software to allow those attacks to succeed;
- Specify mitigations which resolve the vulnerabilities and discuss ways in which the mitigations can be circumvented.

3.3 Framework Evaluation

In this Section, we describe the evaluation of the proposed framework. Section 3.3.1 describes our course, the context in which it resides, and the knowledge trainees gain before attending our course. Next, in Section 3.3.2, we described the structure of the two instances of the course, which we compare as part of the evaluation. We then detail the design of the controlled experiment and comparative analysis used to evaluate the framework in Section 3.3.3. The evaluation described here concerns only the basic structure of the framework (described in Section 3.1 and illustrated in Section 3.2.1) and is not concerned with the enhancements described in Section 3.2.3. Finally, in Section 3.3.4, we detail the empirical evaluation, conducted through interviews and questionnaires, which assess the quality of the enhancements described in Section 3.2.3.

3.3.1 Course Context

Our course on secure software engineering is an elective course for fourth-year undergraduate students of computer science studies. Around 50 students enroll in this course each year. Before attending the course, the students complete several mandatory courses covering the topics of data modeling, software engineering, network-based systems, distributed software systems, and information systems. They have no prior knowledge related to the domain of information security or any of its subdomains.

Based on the student's prior knowledge, we focus the course on topics related to the application and system security. As most of the students find employment as software engineers, we focus on topics that offer the most value to future software designers and developers. Topics include applied cryptography, data security (protecting data in transit, storage, and in use), authentication and access control mechanisms, and secure software development, with a focus on security design analysis.

3.3.2 Course Structure

Two instances of the course are relevant for this research. The 2015/2016 instance (referred to as the *traditional* course) uses the traditional classroom approach, where the professor holds lectures every week for all trainees (i.e., students), while the trainer (i.e., the teaching assistant) runs laboratory exercises for groups of 10 to 16 trainees. The laboratory exercises focus on the topic from the previous week's lecture, where trainees complete assignments that require them to implement or use a specific security control or stop a common attack.

The 2016/2017 instance (referred to as the *framework* course) contains one significant difference, where the laboratory exercises are constructed using the training framework. The lecture materials used in both instances of the course are mostly identical. However, to accommodate the hybrid flipped classroom, the materials for the laboratory exercises are restructured for the new course. In both instances of the course, there are six laboratory exercises, each lasting 2 hours, as described in Section 4.2.1.

The main difference between the old and new course design, relevant to our experiment, is the structure of the teaching materials. In the new course, the lab time is dedicated to the discussion and learning the intricate craft of security design analysis. In our experience, security concepts, such as common controls, attacks, and vulnerabilities, when presented in a vacuum, is something that the trainees can learn on their own, which is why we leave this to the trainees, as part of their preparation for the lab.

3.3.3 Experiment Design

To complete the traditional instance of the course, trainees had to secure a part of a banking information system implemented as part of another course. They did this by producing a security design analysis of the system before implementing it, after which they implemented the identified security controls in their code and configurations.

To complete the framework instance of the course, trainees had to complete the same project as their predecessors, the traditional group. They performed a security design analysis of the banking information system and then implemented the identified security controls during the system development. It should be noted that a full banking information system can have hundreds of critical assets, each of which can have several accompanying threats. The trainees examine a subset of the system to reduce the workload.

Another thing to note is that during project development of both instances of the course, teams of trainees had a series of checkpoints for which they had to produce specific deliverables. We did this to track the progress of trainees more efficiently.

By providing both groups of trainees with an identical project and roughly the same amount of time to complete it, we were able to do a comparative study of the effectiveness of the training framework, as opposed to the conventional teaching approach.

The trainees were divided into teams of three or four members. By comparing the quality of threat models produced by teams from both groups of trainees, we evaluated the new design. We have adopted the approach from (Scandariato et al., 2015) to define the quality of a threat model as a set of metrics, which include the following:

- The quality of the produced data flow diagrams (DFDs);
- The quality of the threat identification step.

The quality of the DFDs is measured by examining:

- The number of produced DFDs;
- The average number of elements in each diagram.

We compare DFD sets produced by teams from both groups to a baseline DFD set produced by the professor and the trainer. The quality of the threat identification step is evaluated by looking at the following metrics:

- The number of correctly identified threats (*TP*, true positives);
- The number of incorrectly identified threats (*FP*, false positives);
- The number of unidentified relevant threats (*FN*, false negatives).

The maximum number of relevant threats a team of trainees can identify ($T_{rel} = TP + FN$) is closely tied to assets, as relevant threats are defined as losses of security objectives. We define relevant assets as assets for which true positive threats have been identified. This excludes abstract assets such as company reputation and user satisfaction. We also aggregate similar assets into a single asset. For example, all log files located on the corporate bank network are considered a single asset. The list of relevant assets includes but is not limited to:

- Data assets, including user credentials, credit card information, invoices, reports;
- System assets, including important subsystems and services, log and configuration files, the website;
- Infrastructure assets, such as workstations, the network, ATMs.

The maximum number of relevant threats that each team can identify is directly linked to the relevant assets identified by each team N_{assets}. For each team, the professor and trainer have analyzed the assets obtained by that team to determine their T_{rel}.

We summarize the obtained results by calculating the average correctness (precision) and average completeness (recall) of the individual teams of both class instances. The correctness of an individual team is calculated by dividing the number of correctly identified threats to the total number of threats identified by a particular team:

$$correctness = \frac{TP}{TP + FP} \tag{1}$$

The completeness of an individual team is calculated by dividing the number of correctly identified threats with the estimated maximum number of threats (for that individual team):

$$completeness = \frac{TP}{T_{rel}} \tag{2}$$

We aimed to determine whether the teams of the framework instance had on average achieved significantly higher correctness and completeness compared to the traditional instance teams. Thus, we analyze the obtained results by applying an unpaired test as we had different subjects in the two test groups. We used the Mann–Whitney test, a non-parametric analog of the unpaired t-test that does not require the assumption of normal distributions. We validated the null hypothesis H_0: "There is no difference between teams of the traditional instance and teams of the framework instance in terms of achieved correctness/completeness." For statistical tests, we set the significance level of $\alpha = 0.05$.

We did not measure the quality of the threat decomposition step, nor the risk analysis step. Regarding threat decomposition, we could not find a suitable way to measure the quality of this step, as no evaluation metrics are proposed in the literature. We decided to avoid comparing the quality of the risk analysis, as the framework trainee group had one whole laboratory dedicated to this activity, while the traditional group only briefly examined this step.

To measure the effectiveness of our new approach, we only considered threat model documents produced by teams where the document owner was present during the six laboratory exercises. Twenty-eight trainees were present during the six lab sessions of the traditional instance, while thirty-six trainees attended the six lab sessions during the framework instance. Not all of these trainees were threat model document owners as some of them belonged to the same team. Overall, fourteen threat models were examined from the traditional instance of the course, while seventeen threat models were looked at from the framework instance.

The results of this evaluation are described in Section 3.4.1.

3.3.4 Empirical Evaluation of Enhancement

We conducted an informal evaluation of the enhancement described in Section 3.2.3 by analyzing observations of the trainer and supplying a questionnaire to the trainees (i.e., the students).

Through participant observation (Taylor and Bogdan, 1984), the trainers observed how the trainees interacted with the VSP during the exercise. The trainers noted and payed special attention to:

- The levels of trainee engagement;
- The nature of the interaction with the VSP;
- The way the VSP influenced the lab;
- Any difficulties that occurred.

At the end of the semester, the trainers conducted a survey, following the approach described by Punter et al. (2003). Trainees were provided with a questionnaire for evaluating the quality of each of the labs. The questionnaire had the following structure:

- A brief description of the lab and its goal;
- A set of statements to be graded through a 5-point Likert scale (Albaum, 1997) (1 – strongly disagree, 3 – undecided, 5 – strongly agree), including:
    - *The goal of the exercise is clear and meaningful*;
    - *During the lab we have achieved the specified goal*;
    - *The lab was interesting.*
- The question *What was the best part of the lab*, to be answered using a free text field;

- The question *What would you improve*, to be answered using a free text field.

The results of this evaluation are described in Section 3.4.2.

3.4 Results and Analysis

In this Section, we present the results of our evaluations and discuss their meaning. In Section 3.4.1, we present the results of the experiment described in Section 3.3.3, where we show that our SDA Training Framework achieves better learning outcomes when compared to the traditional teaching approach. Next, in Section 3.4.2, we examine the findings of our observations and surveys described in Section 3.3.4, regarding the quality of the enchantment to our framework. In Section 3.4.3, we discuss the limitations of our experimental evaluation and the training framework, including its enhancement. Finally, in Section 3.4.4, we discuss the implications of all the work presented in Chapter 3 and how the training framework can be applied in different contexts.

3.4.1 Results of Comparative Analysis Experiment

We first analyze the quality of the DFDs produced by the two groups of trainees and compare them with the quality of the DFDs produced by the teaching staff. Each group produced 1 level 1 DFD and multiple lower-level DFDs. We measure the average number of elements on the level 1 DFD, the number of level 2 DFDs, the average number of elements on level 2 DFDs, and the number of level 3 DFDs. When counting the number of elements, we take into account data stores, process nodes, and external entities. The results are presented in Table 8.

Table 8 Quality of the DFDs

Group	Traditional trainees	Framework trainees	Teaching staff
Avg. # of level 1 DFD elements	9.64	10.71	10
# of level 2 DFDs	2.29	2.24	3
Avg. # of level 2 DFD elements per diagram	6.09	8.56	9.33
# of level 3 DFDs	6.36	0.24	0

While both groups showed similar results when analyzing the high-level view of the system through a level 1 DFD, the framework trainee group showed significant improvement when it came to more detailed analysis. In general, the framework trainees showed a better understanding of their system from the perspective of data flows.

The main improvement was related to the significantly lower amount of level 3 DFDs. In general, level 3 DFDs signal a too-detailed model, which rarely adds value to the threat modeling activity. Indeed, of all level 3 diagrams produced by the traditional trainee group, not one was produced that introduced a new threat to the system, making the diagram useless.

We believe that this improvement is a direct consequence of previous experience, where we warned the trainees about excessive level 3 diagrams. Therefore, the new teaching approach had little impact on the positive results.

Next, we analyze the number of identified relevant assets N_{assets}, which determines the maximum number of threats that each team can identify. The average number of identified relevant assets for each course instance is presented in Table 9. These results show that the framework trainee group identified more assets. We attribute this to our own increased experience, as we recognized that the traditional trainees completely missed infrastructure assets, and we put more emphasis on this topic in the next instance of the course.

Table 9 Number of identified relevant assets

Group	Traditional trainees	Framework trainees	Teaching staff
# of identified assets	9.57	12.18	16

Finally, the results of the threat identification step are presented in Table 10.

Table 10 Number of identified threats

Group	Traditional trainees	Framework trainees
Avg. # of correctly identified threats (*TP*)	11.36	22.82
Avg. # of incorrectly identified threats (*TN*)	5.36	3.18
Avg. # of missed threats (*FN*)	9.64	4.82
Correctness (precision)	68%	87%
Completeness (recall)	54%	81%

These results show a significant increase in the quality of the threat identification step between the two class instances. The differences in correctness/completeness between the traditional and framework instances were found to be statistically significant (the null hypothesis H_0 was rejected for completeness with the p-value of $1.78 \times 10^{-5} < 0.05$; for correctness, the null hypothesis was rejected with the p-value of $2.76 \times 10^{-5} < 0.05$). Thus, we may conclude that the teams of the framework instance have achieved significantly better correctness/completeness compared to the teams of the traditional instance.

Moreover, the Scandariato et al. (2015) measure the correctness and completeness of the threat identification performed by their students when compared to their own threat identification. They choose the 80% threshold for both precision and recall as a good reference point for student success. According to this, we conclude that the framework trainee group has achieved good precision and recall when it comes to threat identification.

It is our opinion that these results are a direct consequence of our SDA Training Framework. By supplying trainees with attacks and security controls beforehand, trainees were able to discuss and reason about threats and focus on the cognitive aspect of security design analysis during the lab exercises.

3.4.2 Results of Empirical Evaluation of Enhancement

During the execution of the lab exercise created using our framework and a vulnerable software package, the trainers noted a higher than expected level of engagement from all trainees. Concretely, the VSP with its challenge system facilitated a competitive atmosphere, where trainees were rushing to complete the next challenge before their colleagues. For more difficult challenges, the trainees gathered in smaller groups to brainstorm solutions, maintaining the competitive spirit by contesting other groups.

Problems would occur when trainees got stuck with a challenge, causing frustration. Additionally, more successful trainees or groups gave answers to other groups when left unchecked, subverting the learning goal for the other groups. Both cases were resolved by the trainer intervening, to give advice to stuck trainees and help them solve the challenge, as well as to foster a fair playground, where more successful groups would not ruin the game for the rest.

Near the end of the lab, the participants summarized the learning outcomes, where the trainer noted that the trainees clearly understood the impact injection attacks could have on the software, why they occur and how to conduct them. Finally, the trainer facilitated a discussion around the quality of the lab and the use of the VSP. The consensus was that the lab was fun and engaging and that the case study was both realistic and relatable. The trainees made requests for similar labs in the future.

Fifteen trainees filled the optional questionnaire at the end of the semester. Table 11 presents the average grade for each statement, given by the trainees using a 5-point Likert scale.

Table 11 Results of the survey grading the quality of the VSP Injection lab

Statement	Grade
The goal of the exercise is clear and meaningful	4.67
During the lab we have achieved the specified goal	4.73
The lab was interesting	4.6

Notably, not a single trainee disagreed (strongly or otherwise), with any of the statements. The first statement had two undecided votes, while the second and third had one undecided vote each. Most of the trainees strongly agreed with the statements. Importantly, this lab had the highest grade for all three statements, when compared to all our other labs.

Six responders answered the free-form question "*What was the best part of the lab*". All six answers praised the use of the vulnerable software package and the opportunity to conduct real attacks.

Two responders answered the free-form question "*What would you improve*". The first responder asked for more guidance when performing the attacks, while the second responder proposed that different groups of trainees tackle different sets of challenges provided by the VSP and present the results to each other near the end of the lab. The first answer is aligned with the observations made by the trainers, where some trainees got frustrated when not being able to solve a challenge. To address this, we developed *Hint* cards which we can distribute to trainees that need more assistance with the challenges. The second answer gave us an idea to bundle the challenges of different levels of difficulty (e.g., 3 level 1 challenges, 2 level 2 challenges, and a single level 3 challenge) and to distribute a bundle to each trainee group and have them compete for points against the time limit of the lab. This setup is typical in capture the flag events and different gamification approaches discussed in Section 2.1, so we feel confident that this approach can further enhance the quality of the learning outcomes.

3.4.3 Limitations

Several limitations influence the results of the study. First and foremost, both the professor and the trainer (i.e., the teaching assistant) have gained experience in both teaching and the domain of information security. We cannot accurately determine the quality of the gained knowledge and skill between the old and new course. However, we can safely say that our understanding of both information security and teaching information security has increased in general. As Schoenfield (2015) points out, threat modeling is an art form, where experience plays a vital role in the quality of the produced models. If we were to apply our old class design to the next generation, then we are confident that we would get better overall results compared to the 2015/2016 instance of the course, if only marginally better.

The next limitation to consider is the fact that we have not adequately measured the change in the quality of courses the students attended before our course. This is especially important for the software engineering and network-based systems courses, as we rely on the knowledge students receive here to reason about security. According to the official study program document, no significant changes have occurred in the curriculum of these courses. To the best of our knowledge, no minor changes in the teaching technique or the subject matter of the relevant courses have taken place. However, it is difficult to assess if this is the case.

The final limitation related to the study is the lack of a larger dataset to analyze. Fourteen threat models were examined from the 2015/2016 instance of the course, and seventeen from the 2016/2017 instance, which is a relatively small scale.

A limitation of our framework is its inherent complexity, which requires additional effort to prepare the course. For the traditional classroom, the teacher prepares course materials that can then be used in the classroom. Our approach involves the construction of preparatory materials, in addition to the case studies that are examined in the classroom. Furthermore, an effort is required to align the preparatory materials, the case studies, and the security analysis method. As with regular course materials, the preparatory materials need to be periodically updated to stay relevant.

To address the complexity limitation, we have utilized vulnerable software packages (VSPs), as described in Section 3.2.3. By enhancing our framework and integrating a VSP into the formulated labs, we drastically reduced the amount of time it takes to prepare a lab using our framework. Initially, before enhancing our framework, the lab constructor needed to invest significant time (approximately a week) to build the case study, the assignments and related code samples for the lab covering the topics of injection attacks. With the Juice Shop application, the build time was reduced to a single day, which included the time it took to find and explore the VSP. Additionally, the preparatory materials were constructed by combining resources from the Juice Shop website and the OWASP Top 10 list (Stock et al., 2017).

A significant limitation of this enhancement is the relative shortage of VSPs. While web technologies and common web security issues (Stock et al., 2017) are covered by several high-quality VSPs (Siles and Bennetts, 2019), domains such as mobile or embedded application are scarcely covered. While elementary code samples and toy projects can be found, sophisticated solutions reaching the quality of the OWASP Juice Shop project are few and far between.

3.4.4 Implications

The goal of our SDA framework is to contribute to the development of a security-aware workforce of software engineers. As part of our secure software engineering course, we designed a novel framework based on the hybrid flipped classroom and case study analysis to teach students the practice of security design analysis. We evaluated our new approach by comparing the quality of the threat model documents produced by the 2015/2016 student group (who attended traditional labs) with the threat model documents produced by the 2016/2017 student group (who attended labs created using our proposed framework). Our results show that the student teams of the framework instance of the course have achieved better overall correctness and completeness on the threat identification task compared to the student teams of the traditional instance of the course. The applied statistical test shows that the obtained differences are statistically significant. Therefore, labs created through the SDA Training Framework are the preferred alternative, as opposed to the traditional classroom, when it comes to teaching SDA.

A significant limitation of the framework is its complexity, which we tried to address by enhancing the framework using vulnerable software packages. Vulnerable software packages enable trainers to demonstrate security issues in a real-world context and allow trainees to apply their security knowledge to both perform attacks on and build defenses in a real-world software system. Thus, vulnerable software packages serve the first principles of instruction (Merrill, 2002) to increase trainee learning. By utilizing these tools, we have reduced the effort it takes to develop a laboratory exercise through our framework. Additionally, we have increased the overall quality of the lab by replacing a case study description and vulnerable code samples with a fully functioning application that contains vulnerable code.

The framework proved useful for the fourth-year undergraduate students of a software engineering university program. However, we argue that the framework is also useful for employed software engineers, as most of the fourth-year students find employment right after finishing our course on software security. Therefore, the framework presented in this Section can be used to formulate a one-day workshop for a conference, a set of training exercises for software developers, as part of a corporate training program, or, as we have demonstrated, a set of labs for a university course. While the framework is designed for teaching security design analysis, the target of the SDA can be anything from a web-based system to a hardware chipset.

4 SATMUS Process

This Chapter details our solution for the second research question and the second segment of the hypothesis listed in Section 1.4. Our goal is to define a security design analysis method which:

- is compatible with agile software development practices,
- provides accountability of work,
- is accompanied with adequate guidance for method execution and adaptation to different contexts, and
- offers security assurance for the developed software and an incrementally updated threat model.

We conceptualize an SDA method called SATMUS, which stands for the Security Analysis and Threat Modeling of User Stories. SATMUS represents an adaptation of the traditional threat modeling method (Shostack, 2014b) to the context of software development following the Scrum framework (Rawsthorne and Shimp, 2011). SATMUS examines each user story to derive actionable low-level design decisions and implementation tasks from ambiguous and generalized high-level security requirements. Furthermore, SATMUS aims to address the applicability issues which hamper the adoption of SDA in agile development, identified in (Luburić et al., 2018a).

Section 4.1 defines the SATMUS process, where we examine the methods inputs, outputs, and composing activities. Here we explain how the process integrates with Scrum development and what part the different organizational roles play in SATMUS. Section 4.2 discusses how to tailor SATMUS to contexts with different levels of security requirements for the developed software. Here we illustrate two case studies of SATMUS adaptation to the workflow of two different software vendors. In Section 4.3, we demonstrate the execution of the tailored SATMUS processes on real-world user stories supplied by the two case study vendors, as well as the comparative analysis which examines how SATMUS addresses the applicability issues in (Luburić et al., 2018a), compared to similar methods proposed in the literature. Finally, Section 4.4 discusses the outcome, as well as the limitations and implications of the presented work.

4.1 Process Structure

This Section starts with an overview of the SATMUS process, presenting its purpose, inputs, and outputs in Section 4.1.1. Next, in Section 4.1.2, we describe the process in detail, analyzing each of its activities.

4.1.1 Overview

The goal of the SATMUS is to identify weaknesses and vulnerabilities in the developed software system that an attacker might exploit. SATMUS analyzes user stories to identify security-related issues concerning either the feature represented by the examined user story or an asset with which the feature interacts. Consequently, SATMUS translates high-level security requirements (e.g., protecting the confidentiality of personal data and user credentials) into actionable, low-level design decisions and tasks.

To integrate SATMUS into the Scrum development process, we utilize the Definition of Ready (Power, 2014). The Definition of Ready is a set of rules that help agile teams remember all the things that need to be done before the development team can start implementing a user story. We expand the Definition of Ready with a single task that states that SATMUS analysis must be conducted for each user story. Therefore, SATMUS executes for each user story that is in the product backlog, before adding the user story to a sprint backlog and before any code is written. To minimize wasted work, SATMUS should be among the final tasks conducted as part of user story preparation, to analyze a mature user story that is unlikely to change significantly. We also utilize the Definition of Done, which contains a list of tasks that must be completed for each user story before it is considered finished (Rawsthorne, 2010). As it is possible for a user story to be misinterpreted before developing the code, we add a task to the Definition of Done that requires each story to be reevaluated for security considerations, to ensure that the implemented code is aligned with the initial security assessment.

Figure 7 presents the SATMUS process overview, noting the inputs and outputs of the process.

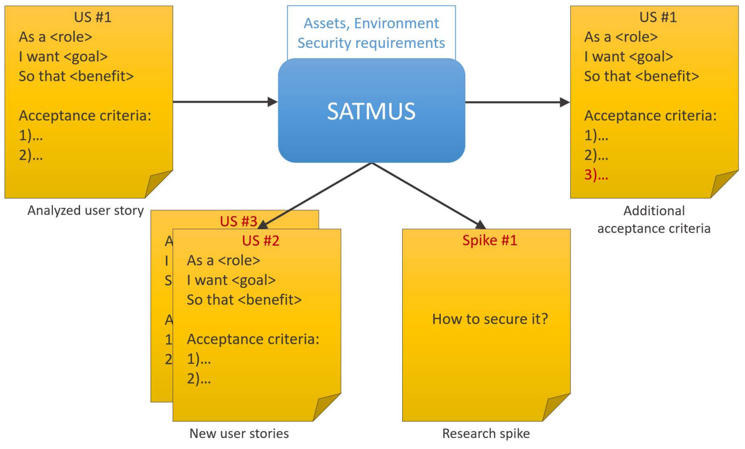

Figure 7 SATMUS process overview

Inputs

The primary input to the SATMUS process is the user story. User stories are a mechanism for describing features provided by the software system, which bring value to the user of the software and the business running it (Cohn, 2004). The elements of a user story are:

- The software user role, for which the feature is being developed;
- The goal, which describes what the user wants to achieve;
- The value, which explains why the user wants to achieve the goal;
- The acceptance criteria, a list of conditions and requirements under which the user story is considered complete.

Based on these elements, the user story is often described through the template which has the following structure: *As a <role> I want <goal> so that <benefit>*, where the acceptance criteria follow as a list of items that need to be met. An example user story might be:

"*As a system administrator, I can block user accounts from accessing the system, so that I can stop suspicious activity.*

Acceptance criteria:

1) *I can block any user with lower privilege than me;*
2) *I can ban multiple users at once;*
3) *Blocking users should be quickly achieved (e.g., with one click).*"

In the previous example, the acceptance criteria contain a mixture of functional requirements (the first and the second acceptance criteria) and non-functional requirements, such as usability (the third acceptance criteria). In general, acceptance criteria can consist of anything the product owner and development team deem necessary to specify explicitly. Apart from the distinct functional requirements, this can include the need for specialized analysis, testing, or documentation. Standard unit and integration testing might be part of the Definition of Done (Rawsthorne, 2010). However, performance testing or security testing might be reserved for high priority features and listed in the acceptance criteria for the appropriate user stories (Leffingwell, 2010).

It should be noted that *assets* that the new code will manipulate, and its *environment* are important for SATMUS, as from these we derive *security requirements* for the examined user story. They are not presented as an input here, as they are an integral part of the SATMUS process, determined during process tailoring, as discussed in Section 4.2.

Outputs

One possible outcome of applying SATMUS to a user story is the introduction of additional, security-relevant, acceptance criteria. These might include requirements for the software (e.g., Arriving XML documents must be validated against an XML Schema; The maximum image upload size can be 20 megabytes) or for the organization developing the software (e.g., The feature code needs to undergo a team-wide security code review; The feature endpoints need to be tested against fuzz testing tools).

Another output of SATMUS can be a spike (Knaster and Leffingwell, 2018). A spike represents activities such as research, design, and prototyping. The goal is to gain the necessary knowledge to sufficiently understand a requirement, increase the reliability of a story estimate, or understand the risk of a technical approach. As a result of SATMUS, the Scrum team might realize that a vulnerability exists, but may not know how to mitigate it. Likewise, the team might understand that a specific security control needs to be integrated into their solution but might not possess the knowledge about which provider of the security control is tried and tested, and which configuration offers the most security. For these situations, a research spike is issued. For example, when a need for an anonymization function for personal data is discovered, the team conducts a research spike to find out how to develop and use such a feature.

Finally, during the security analysis, the team might discover that additional work needs to be done to introduce or modify a security control, change the infrastructure, or perform any significant development to increase the security posture of their solution. Therefore, SATMUS can output new user stories directly focused on improving the security of the system. An example of this might be the development of an infrastructure component for input validation of data specific to the application domain.

4.1.2 Internals

The SATMUS process flow, including its composing activities, is illustrated in Figure 8. The first set of activities, *Estimate Impact*[4], is concerned with calculating the security impact of the user story (which can be high, medium, or low), to justify the investment of effort for further security analysis. Low impact user stories are not analyzed further, as they do not introduce a significant security risk. User stories of medium and high impact undergo the *Threat Model* set of activities. Furthermore, external experts assist the Scrum team for high impact stories, as these usually map to critical security

[4] The use of italic in this Section signals that this is an activity illustrated in Figure 8.

requirements. The goal of this set of activities is to identify any potential security issues concerning the user story, incrementally update the threat model for the component which the user story expands, and define requirements for mitigations. Finally, the *evidence is documented* to:

- Ensure that the SATMUS process was conducted;
- Fulfill any potential regulatory compliance;
- Issue new tasks (e.g., user stories, research spikes) to resolve any identified security issues.

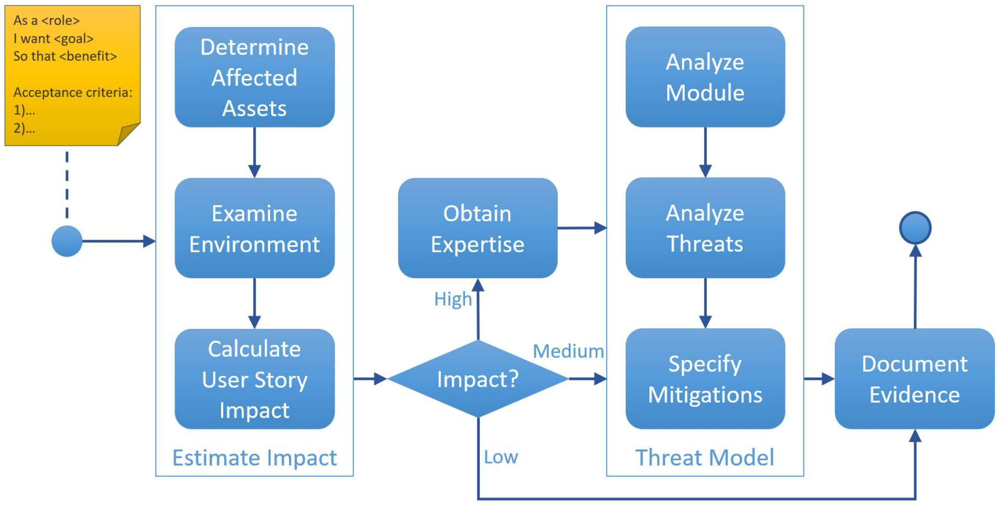

Figure 8 SATMUS process internals

Determine Affected Assets

The process starts by determining assets that will be affected by the code that will implement the user story. The term asset is often ambiguous and generic, most often defined following the ISO definition as “anything that has value to an organization” (Disterer, 2013). In the context of our method, we describe an asset as follows:

- An asset is a software object (e.g., data, function, UI component, service, an application server) manipulated by the software system under development, which has a security requirement.

This definition, which is a subset of the ISO definition, is suitable for our context because:

- It is only concerned with concepts related to software engineering and the system under development, which makes it more understandable to the people practicing the process;

- It focuses only on objects that have security requirements, enabling reasoning about security throughout the rest of the process.

Organizations that develop software that has regulatory compliance requirements should consider constructing an asset inventory that notes the security requirements for each asset. In (Luburić et al., 2018b), we have proposed a conceptual model for an asset inventory, that maps assets to security goals (i.e., confidentiality, integrity, availability). When an asset has a security goal, the owner of the asset needs to determine the security goal value (on a scale of 1 to 3) and provide the reasoning behind this value (e.g., references to standards or best practice documentation).

With the aid of such resources, software engineers can perform checklist verification to make sure that vital assets were not forgotten during this step. Organizations that have little or no explicit security requirements can rely on the expertise and agility of their Scrum team and their stakeholders to understand and identify which assets have quality security requirements.

Examine Environment

Another significant aspect to consider is the environment in which the developed feature will reside.

For example, a piece of code that realizes a user story might manipulate user passwords, cryptographic keys, or personal data, making it highly sensitive code. However, if this code is accessible only to internal services, sheltered by layers of security controls where no user input arrives unvalidated, then the user story might have fewer security implications than first assumed. Likewise, while code accessing a database might not work with sensitive data, it can raise a significant security issue if, for example, there exists an SQL injection vulnerability. Therefore, it is essential to examine the context in which the code that realizes a user story will operate.

The primary goal of this activity is to determine if the data supplied to or resulting from the new code crosses a trust boundary and what the nature of that boundary is (Myagmar et al., 2005). For example, publicly available APIs that accept user input have more significant security implications than code that interacts with a service running under the same privilege on the same machine. Likewise, code that accesses a database or offers a critical system function has higher security requirements than code that does not handle sensitive data and provides a low priority service.

Finally, it is vital to examine how the new user story changes the environment, as the introduction of new data flows or entry points can increase the attack surface and produce new threats to the environment.

Calculate User Story Impact

The purpose of the following step is to funnel the security-related development effort to the most sensitive parts of the system. Threat modeling requires effort and thoroughly performing it might entail more effort and expertise than the Scrum team possesses. Therefore, we propose an activity that determines whether a user story requires such an investment.

Each organization must define the calculation algorithm for its context, based on their internal resources and the security requirements for their product. This calculation should consider:

- The assets affected by the user story;
- The environment in which the new code will run.

An example of such a calculation formula utilizes the asset inventory presented in (Luburić et al., 2018b). The Scrum team determines the highest security goal value from the list of *affected assets* (on a scale of 1 to 3). Next, the team *examines the environment* and grades it on a scale of 1 to 3, based on the following criteria:

1) When the new code does not interact with a human user and it does not introduce new data flows.
2) When the new code interacts with an internal human user.
3) When the new code represents a security feature (e.g., cryptographic module, access control module), or interacts with an entity external to the software solution (e.g., Internet users, integrated external systems).

For this example, the user story impact is calculated based on the formula presented in Table 12.

Table 12 Sample user story impact calculation formula

Asset security goal / Environment	1	2	3
1	Low	Low	Medium
2	Low	Medium	High
3	Medium	High	High

Regardless of the exact calculation algorithm, the output of this step determines the amount of effort invested for further security analysis, where the three general outcomes are:

- A user story has a **low** security impact, where no additional security analysis is required;
- A user story has a **medium** security impact, requiring further security analysis;
- A user story has a **high** security impact, requiring a detailed security analysis, with a possible investment of resources from other parts of the organization or experts from a different organization.

It should be noted that the *Estimate Impact* set of activities (*Determine Affected Assets*, *Examine Environment*, and *Calculate User Story Impact*) should introduce only a minor overhead to the development process. A glance at the asset inventory and a brief discussion of the environment can quickly determine what the impact of the user story is. As this is the part of SATMUS that must be conducted for each user story, it must be quickly finished.

Obtain Expertise

When a user story has high impact, additional resources need to be invested to ensure that no significant security issues get introduced with the upcoming code implementation. Ideally, this activity should be triggered rarely, as security expertise in software development is hard to come by and is expensive (Whyte and Harrison, 2010; Assante and Tobey, 2011).

Organizations building software with plenty of explicit security requirements should consider obtaining a dedicated security team to support software developers for high impact user story assessment (McGraw et al., 2018). A less expensive alternative might be to hire security consultants (Poller et al., 2017) to perform the threat modeling and train the Scrum team on how to mitigate any issues that might arise from it. One caveat regarding this approach is that external consultants might understand the security of a technological stack or problem domain, but they cannot fully grasp the inner workings and intricacies of the specific software developed by the organization. Alternatively, the Scrum team should contact senior developers and architects within the organization to aid with the threat modeling.

As part of this activity and before expending significant resources, a research spike might be called for to gain a better understanding of the problem and the best solutions for it.

Analyze Module

When a user story becomes the Scrum team's focus, software code and design level details are discussed so that the whole team can get an understanding on what tasks need to be executed to fulfill the user story. During this discussion, software components, functions, and control flows are examined in an informal and unstructured manner. When a user story has a high or medium impact, our approach expands this discussion by analyzing the module from a security perspective. The purpose of this step is to define the entire attack surface and ensure that no threats go unnoticed and that a suitable defense in depth can be planned.

Analyzing the module usually entails examining design artefacts such as UML activity and sequence diagrams or data flow diagrams. Depending on which SDA method is used, a suitable representation can be selected to support the next step, which is threat analysis. For STRIDE-based SDA, a common approach is to utilize data flow diagrams (Shostack, 2014b). Here, the team needs to identify all data stores, flows, and processing nodes affected by the proposed design and code changes. Flows which interact with entities external to the jurisdiction of the Scrum team (e.g., outside users, third-party services, components developed by other Scrum teams) are especially important, as they cross significant trust boundaries.

An issue arises during this step if a data flow (especially one which contains assets of high security goal values) leaves the components of the system with which the Scrum team is familiar (Cruzes et al., 2018; Tuma et al., 2018). If this is the case, a Scrum team member (e.g., Scrum master) or a security team member, if available, need to consult with other teams and expand the data flow diagram to include their components. This is especially important when new assets are introduced to the system, as this can affect the threat models of the other teams. As Scrum favors face-to-face communication, the organization should encourage and facilitate such knowledge transfer.

The team builds the initial threat model at the start of development and incrementally updates it with each impactful user story. Therefore, data flow analysis for a specific story might entail a small tweak to the existing data flow diagrams (DFDs), or it might not even require modification to these artifacts (when the new story does not change the current flows). Importantly, data flow diagrams need to be concise and accurate, focusing on trust boundaries (e.g., between the application and the OS, between internal and external services, between human users and the software). Data flow minimization, as described by Tuma et al. (2017), aids developers in constructing more concise diagrams that are less prone to the threat explosion problem. For example, two communicating processes where there is no trust boundary between them can be merged into a single process. Likewise, when two nodes have multiple data flows between them, they can be merged into a single flow, which gains the criticality of the most critical initial flow.

Analyze Threats

Threat analysis, as the core activity of SDA, largely depends on the selected SDA method. For the example above, we utilize STRIDE threat discovery on the constructed DFDs. We choose STRIDE as the basis for this step due to the method's relative maturity and prevalence, as described in Section 1.3.3. Once applicable threats are determined, threat decomposition takes place. The goal of threat decomposition is to identify attacks that realize a threat and the vulnerabilities that enable attacks so that mitigations can be planned during the next step.

Threat analysis, and especially threat decomposition, is the most challenging activity for Scrum teams, as it requires security expertise, which developers usually lack (Dhillon, 2011; Morrison et al., 2017). Looking at SATMUS, most of the activities discussed so far can be performed with no or simple training, as *Estimate Impact* activities and *Analyze Module* requires expertise in software engineering, while threat discovery can be achieved following a simple algorithm. While threat decomposition can be performed by identifying misuse cases (Alexander, 2003; Sindre and Opdahl, 2005) and creating control flows to stop them, this can be practiced successfully only when aided by a security mindset and software security knowledge (Shostack, 2014a; Schoenfield, 2015). For example, a software developer lacking security expertise might realize that an authentication mechanism can be bypassed because of poorly-designed control flow in the existing code but might not understand what cross-site scripting is.

To combat this issue, organizations need to invest in dedicated training. While this is in contrast with our Simplicity requirement, some security training is unavoidable. As pointed out by Poller et al. (2017), organizations rely on developers to produce quality software, fulfilling quality security requirements alongside other "ilities" (e.g., maintainability, usability) without dedicated training, leading ultimately to the production of insecure software. Therefore, organizations should invest research spikes conducted by their senior developers (with the aid of a security team if one is available) or hire external security experts to produce and administer dedicated software security training. The training needs to be related to the product's context, considering the technologies used to create the product and any explicit security requirements (e.g., derived from standards and regulations).

Here the SDA Training Framework described in Chapter 3 can be utilized to optimize the training program and achieve better learning outcomes.

An alternative or supplemental approach is to employ attack pattern catalogs, which aids threat decomposition by providing example attacks and vulnerabilities that can be mapped to the context of the developed product. Several public catalogs exist, such as the Common Attack Pattern Enumeration and Classification – CAPEC (Barnum, 2008), the Common Weakness Enumeration – CWE (Martin, 2007), and resources provided by OWASP (Stock et al., 2017). However, such catalogs can be unwieldy as they contain many entries that might not be relevant to the Scrum team's context.

For the construction and maintenance of the threat models, the Scrum team might benefit from a tool like the freely available Microsoft Threat Modeling Tool (Microsoft, 2019). Apart from the diagramming functionality, the tool offers a knowledge base that generates threats for the given diagrams. However, the tool is susceptible to the threat explosion problem, where dozens of (possibly vague) threats can be generated for even a basic diagram, especially when relying on the knowledge base offered by the default templates (Sion et al., 2018). Organizations should consider constructing their own knowledge base for the given technological stack, even if it is less comprehensive, as it is better to generate a few relevant threats than many vague threats (Cruzes et al., 2018).

When not aided by an attack pattern catalog or tool, the threat decomposition step is usually conducted during one or more brainstorming sessions (Ransome and Misra, 2013; Shostack, 2014b; Schoenfield, 2015). Given the exploratory nature of the problem, threat decomposition can be time-consuming, where unlikely attacks can be discovered indefinitely, and there is no clear way to determine when all the critical threats have been discovered. Therefore, organizations need to determine the amount of effort and time dedicated to threat decomposition, based on the user story impact, to introduce a stopping condition (Schoenfield, 2015; Luburić et al., 2018a).

Specify Mitigations

Once threats are decomposed, and attacks are identified, mitigations can be analyzed to resolve potential security issues. At this point, new user stories can be created, a research spike can be planned, or additional acceptance criteria can be issued for the examined user story. For each identified attack there should be explicit mitigation (either already in place, for example, an infrastructure component, or planned). If it is not possible to identify mitigation for an attack, a research spike is expected.

If misuse cases were utilized, additional acceptance criteria could be defined to test if a misuse scenario was prevented. Likewise, acceptance criteria can be expanded to include specialized testing and code review (manual or aided by security tooling).

New user stories arise when a need for a new component (usually in the infrastructure of the system) is discovered. The implementation of a new access control model, the use of a cryptographic module, or a need for an enhanced event logger are all examples of features that can be described by a new user story.

The goal of research spikes can be to obtain a better understanding of threats and attacks or a prerequisite for the successful implementation and testing of a new user story or acceptance criteria.

It should be noted that mitigation analysis does not entail risk analysis, so the mitigations are not prioritized. While the Scrum team takes part in the whole SATMUS process, prioritization of user stories and tasks, in general, is left to the product owner and the management of the organization, rather than the developers.

Document Evidence

The final step of the SATMUS process is the documentation of evidence. The purpose of this step is to provide evidence that SATMUS took place, as well as list any results of the analysis relevant for the organization.

Depending on the agility, relevant security requirements, the impact of the user story, and general context of the organization, this step can have significantly different outcomes. For example, an organization aiming to comply with the IEC 62443-4-1:2018 standard (IEC, 2018a) can construct templates for documenting the execution of the SATMUS process. Another organization with less stringent security requirements can tag medium or high impact user stories with the “security” tag to note which stories required threat modeling.

4.2 Process Tailoring

The SATMUS process can have significantly different implementations for different software vendors, for different software products constructed by the same software vendor, or even for different sets of teams working on the same software product. Software products require different levels of security, determined by the purpose of the software, the functions and assets it works with, and the context in which it operates (Schoenfield, 2015). While one vendor might produce software with strict security requirements and a strong requirement for security assurance, another vendor might have basic security requirements for the developed software and no need for documented evidence of security assurance.

In Section 4.2.1, we analyze the activities of the SATMUS process that are most prone to change based on the security context of the developed software. We then present two software vendors with significantly different case study implementations of SATMUS and describe how the process is adapted to their context. The differences between the software vendors include organizational structure, the level of security requirements for their software, and the domain in which their software operates. In Section 4.2.2, we describe SATMUS tailored for a software vendor ("Vendor A") that produces software for industrial control systems for utility companies around the world and has strict security requirements (both explicit and quality). In Section 4.2.3, we present a SATMUS implementation used by a software vendor ("Vendor B") that produces management information systems for businesses and has few security requirements that need to be fulfilled. For each case study, we describe the context of the organization using SATMUS, providing a brief description of the organization's capabilities, their business model, the goals of the software they produce and the security requirements for that software. Next, we examine the specific SATMUS implementation, where we describe how SATMUS is tailored for the given organization.

4.2.1 Tailoring Areas

When tailoring SATMUS to a specific organization's context, there are several areas of the process that can vary based on the required level of security for the developed software, as illustrated in Figure 9.

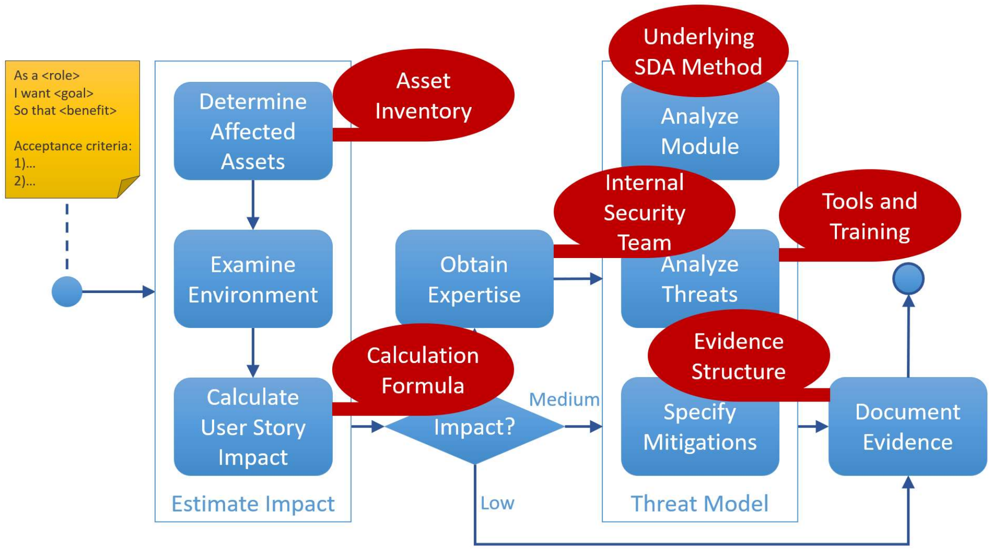

Figure 9 Tailoring areas in the SATMUS process

Asset Inventory

Explicit security requirements are arguably more important than quality security requirements, as they are required for regulatory compliance. Furthermore, explicit security requirements are often concerned with easily identifiable assets (e.g., user passwords, cryptographic keys, personal data). Therefore, an organization with regulatory requirements for their software can benefit from an asset inventory that Scrum teams can reference during the *Determine Affected Assets*[5] step of SATMUS. However, this inventory can contain lightweight entries (e.g., the name of asset and asset owner) for organizations that do not wish to complicate their implementation of SATMUS and can even be a lightweight list of items (e.g., on a sticky note) that each team defines for themselves in organizations that value flexibility and speed of development over security.

While assets with explicit security requirements are easy to map to an asset inventory, the asset inventory can and should contain assets with quality security requirements, deemed of high value to the organization. Essential services and business functions, trade secrets, and confidential documents can all be listed in the inventory.

Finally, it should be noted that a sophisticated asset inventory needs an inventory management process to maintain its contents and would benefit from a tool dedicated for asset management.

Calculation Formula

The calculation formula is arguably the most important decision an organization needs to make when implementing SATMUS, as the calculation algorithm directly determines the amount of security-related effort that will be invested during software development. Organizations that want to spend less on security will aim to classify most user stories as low impact, while organizations that wish to invest in security will classify most stories as medium or high impact. Conducting threat modeling for each user story is inefficient and does not bring value to the organization (Türpe and Poller, 2017).

The calculation formula can consider the importance of adequately implementing the user story itself (e.g., important business function, essential infrastructure component). This can be achieved by estimating the loss value of a user story, as suggested by Pohl and Hof (2015). Loss value

[5] The use of italic in this Section signals that this is an activity illustrated in Figure 8 and described in Section 4.1.2.

represents the cost to the organization, should the user story functions or the data involved in the user story get attacked.

An essential requirement for the calculation formula is simplicity, as Scrum teams need to quickly and effectively measure the impact of each user story. Ideally, the whole *Estimate Impact* set of activities should introduce a negligible overhead to the development process, once teams are familiar with the algorithm.

One example of the formula is presented in Section 4.1.2. An example of a lightweight formula, suitable for a development team only concerned with GDPR (GDPR, 2016) would be to rate user stories as having a medium impact only if the new code manipulates PII or expands the attack surface towards the Internet.

Internal Security Team

Organizations, especially those with regulatory requirements for their software, need to consider investing in an internal security team. The internal security team, in this context, is a team dedicated to the practice of secure software engineering, examining arising threats, attacks, vulnerabilities, and mitigations applicable to the software being developed. Importantly, this is different from the security team that organizations usually have, which is focused on corporate security measures (e.g., IT security of the internal network, physical security) and general security management. As pointed out by Poller et al. (2017), a security management team that is not familiar with agile development practices can hurt security practices in development by imposing inappropriate activities and documentation.

A team dedicated to secure software engineering practices can assist Scrum teams for high impact user stories and can also take part in training construction and administration, conduct or aid research spikes, and improve the overall maturity of SATMUS, making it more valuable, usable, and efficient.

Importantly, security experts are inherently less connected to the low-level software components and their everyday evolution, when compared to development teams that are designing and implementing them. Therefore, an internal security team cannot continually manage and conduct SATMUS, especially for large software products, which is why this duty must fall onto the development teams that can do this efficiently and comprehensively.

Finally, organizations should consider constructing an informal security advisory team. This team consists of developers, architects, and any employees interested in software security. Members dedicate a portion of their time (e.g., 6 hours a week) to learn about the latest security trends and assist in the threat modeling of high impact user stories.

Underlying SDA Method

The security design analysis method defines the views of the system that should be constructed and maintained as part of the *Analyze Module* step, as well as the practices conducted on those artifacts during the *Analyze Threats* step. Section 1.3.3 lists some of the various SDA methods that can be integrated as the underlying SDA method for SATMUS.

Tools and Training

Efficient conducting of any process, SATMUS included, requires some training to understand the steps and expected outputs of each activity. However, dedicated training is necessary for the threat analysis activity, especially the threat decomposition step. This type of training requires security expertise to construct and conduct and needs to be updated regularly with new attacks and mitigations as threats arise. Furthermore, it is not enough to obtain general training on secure software engineering, as it brings less value for developers than training relevant to their context (Bartsch et al., 2011; Oyetoyan et al., 2016; Poller et al., 2017). Therefore, the training should focus on the technology used by the Scrum teams, as well as the context into which the developed software is delivered.

Dedicated training is crucial for the success of SATMUS, both for the whole process and especially for security analysis. Oyetoyan et al. (2016) analyzed two software vendors on the use of security-related activities in their agile workflow. By interviewing different roles in the organization, they concluded that proper training is essential for any security-related activity, as skill determines whether it will be conducted. While security bulletins, blogs and e-learning can be a cost-efficient way to distribute security knowledge, especially to introduce new threats, more involved training is required to achieve a mastery of the basics (Kassicieh et al., 2015). Dedicated training should take the form of interactive workshops, especially when starting with SATMUS. As examined in our previous work, published in (Luburić et al. 2019a) and described in Chapter 3, the hybrid flipped classroom and case study analysis produces good results for this type of training.

If the organization chooses to utilize attack pattern catalogs dedicated training can be replaced in part with a process for constructing and maintaining the knowledge base that will aid threat decomposition. Tools for threat modeling and security design analysis, such as (Goodwin, 2018; Microsoft, 2019), can support the use and maintenance of this knowledge base. The use of threat libraries and attack pattern catalogues can increase the value of the generated threat models (Dhillon, 2011).

Evidence Structure

The final consideration regarding the adaptation of SATMUS is related to the structure of the evidence produced by the security analysis.

Evidence for a user story of low impact might be in the form of an oral statement from the product owner or the most senior developer, stating that the *Estimate Impact* activities were conducted and that the impact was deemed as low. Organizations with explicit security requirements might supplement this statement with a list of identified assets and notes regarding the environment of the user story.

Evidence for user stories of high impact in organizations with lax security requirements might contain a list of planned mitigations, while organizations with regulatory compliance needs might document every part of the threat model in detail, to be used as a form of security assurance, as examined in our previous work (Luburić et al., 2018b).

The bottom line is that the evidence documented at the end of SATMUS should be synced with the needs of the organization, be it to fulfill regulatory compliance, or the needs of its customers. Furthermore, it should facilitate accountability of work so that security work is visible both for planning and auditing needs.

4.2.2 Case Study – Industrial Control System Software Development

With the rise of Industry 4.0, industrial control systems are increasingly benefiting from software solutions that automate their operations. As these solutions often play a critical role in the functioning of factories and parts of a society's critical infrastructure, they have become a target of the most sophisticated cyberattacks that threaten businesses, governments, and human lives (Kobara, 2016).

Here we describe SATMUS tailored for Vendor A, a company that produces software for industrial control system for utility companies around the world.

Organization Context

Vendor A has a workforce of about 500 software engineers (including architects, coders, and testers), organized into 50 Scrum teams. Additionally, Vendor A has a dedicated internal security team that covers a broad spectrum of security-related activities. The vendor's customers are spread across the globe.

Vendor A produces a set of software products for the utility companies. Starting from the baseline product, the software is customized for each customer of Vendor A and is deployed on the premises of the utility company, where the customer's personnel operates it. The software is not directly accessible from the Internet. It is configured and maintained by system administrators, used by utility controllers, and integrated with several other systems (e.g., remote terminal units in the field, customer's internal information systems, geographic information systems).

Utility companies are a priority target for sophisticated attackers like cyberterrorists. To combat this, organizations for standardizations and governments have issued various standards and regulations with which the utility company must be compliant with, some of which directly address the software used by the utility company, such as NIST SP 800-53 (Force and Initiative, 2013), NERC CIP (NERC, 2019), IEC 62443-4-2 (IEC, 2018b). Therefore, the software produced by Vendor A must adhere to a broad set of explicit and quality security requirements. Furthermore, the fulfillment of these requirements needs to be adequately documented, to enable the customer to prove their regulatory compliance concerning the purchased software.

Tailoring Decisions

Asset Inventory: Vendor A utilizes an asset inventory which details the name and type of the asset, as well as its security goals (i.e., confidentiality, integrity, availability) and the priority of each security goal (on a scale of 1 to 3), as presented in (Luburić et al., 2018b). Vendor A expands the model by including references toward the sources of these security requirements (e.g., entries in an industry standard, regulation, or best practice document). Furthermore, the asset inventory contains entries related to critical functions and services, whose protection is paramount to ensure the reliability of the core system.

The entries in the asset inventory are split into several groups, where each Scrum team is made aware of which parts of the inventory are relevant for them. The product owner manages the inventory for each team and immediately communicates any updates of the inventory to the development teams.

Calculation Formula: Once affected assets are determined, and the environment is examined, the calculation formula takes as input two numbers, following the example given in Section 4.1.2, *Calculate User Story Impact*. The first number is the highest security goal value in the list of affected assets (on a scale of 1 to 3). The second number is related to the environment and can have the following values:

1) When the new code does not interact with a human user, does not introduce new data flows, and the user story does not represent an essential feature (as determined by the product owner).
2) When the new code interacts with an internal human user or the user story represents an essential feature.
3) When the new code represents a security feature (e.g., cryptographic module, access control module), or interacts with an entity external to the software solution (e.g., remote terminal units, remote users accessing the system, integrated external systems).

Based on these two numbers, the impact of the user story is calculated by using Table 12.

Internal Security Team: Vendor A maintains an internal security team, with members dedicated to assisting Scrum teams with high impact user stories. Additionally, the security team aids product owners in managing their asset inventories.

Furthermore, the internal security team constructs and administers dedicated security training for threat analysis utilizing the SDA Training Framework presented in Chapter 3, as well as the whole SATMUS process, driving its continuous improvement.

Finally, the security team organizes the security advisory forum, a quarterly event where representatives from every team join in on the discussion about software security. The security team presents new threats, attacks, and mitigations, and highlights security problems and solutions identified and implemented by the different teams in the organization, facilitating the knowledge transfer between participants.

Underlying SDA Method: Vendor A relies on STRIDE-based SDA that examines data flow diagrams, as described in Section 4.1.2.

Tools and Training: Dedicated training is constructed and updated by the internal security team using the SDA Training Framework from Chapter 3. The primary security design analysis methods include STRIDE-based threat analysis (Shostack, 2014b) and misuse case identification and analysis (Sindre and Opdahl, 2005). The developed software acts as the primary case study for analysis, to present relevant threats, attacks, and mitigations. The internal security team constructs and distributes preparatory materials through an e-learning platform and holds periodic workshops to apply the knowledge from the preparatory materials to the case study. Training is readministered if the Scrum team detects a need, vulnerabilities are discovered in production, or if there is a significant update in the training materials. The security team utilizes attack and vulnerability catalogs, such as CAPEC (Barnum, 2008) and CWE (Martin, 2007), to hand pick relevant entries and construct easy-to-consume training materials from them. They integrate this knowledge into a template for the Microsoft Threat Modeling Tool (Microsoft, 2019), which the development teams utilize to construct their threat models and receive feedback about the most critical threats.

Evidence Structure: Finally, templates are provided for documenting the execution of the SATMUS process, which is done by the product owner. The templates are constructed to fulfill the requirement of the IEC 62443-4-1:2018 standard, namely SR-2: Threat modeling and SD-1: Secure design principles. For low impact user stories this entails a note that SATMUS was conducted, while medium and high impact stories include data flow diagrams and lists of decomposed threats and mapped mitigations, as well as references to SATMUS outputs.

4.2.3 Case Study – Management Information System Software Development

Management information systems aid organizations in coordinating, controlling, and analyzing information in an organization, to provide support for decision-making. Their use is wide-spread, and they present a core pillar for all modern organizations (O'Brien and Marakas, 2006). From the security perspective, they are less critical than industrial control systems, but due to the sensitivity of the data they handle (e.g., trade secrets, PII) they can be a viable target for cybercriminals.

Here we present the SATMUS implementation used by Vendor B, a company that produces management information systems for businesses.

Organization Context

Vendor B has a workforce of about 50 software engineers (including architects, coders, and testers), organized into 8 Scrum teams. The vendor's customers are always from the European Union.

Vendor B produces software solutions to support management information systems. Each solution is built from the ground up and is deployed on the premises of the customer. The solutions are usually developed by three to five Scrum teams at a time, where teams need to be agile and move from project to project quickly. Most solutions have at least one set of endpoints that are accessible from the Internet, while most of the code is dedicated to internal data and process support.

Customers of Vendor B are small and medium business with medium-scale databases. They are low to medium priority targets for cyberattackers, as often the only target worth their engagement is PII. The primary regulation that affects these businesses is GDPR. Therefore, the software produced by Vendor B must explicitly protect personal information. Furthermore, as the Internet presents an attack surface with practically infinite attackers, some quality security requirements are needed to protect the customer from chaotic attackers, as well as corporate sabotage.

Tailoring Decisions

Asset Inventory: For each product, Vendor B maintains a simple list of GDPR-relevant data assets, including a name and a brief description of each asset. The list is printed next to each whiteboard where sprint planning occurs.

Calculation Formula: The impact of the user story is calculated based on the following three questions:

1) Will the new code manipulate assets from the inventory?
2) Will the new code directly interact with users?
3) Is the new code a security control?

If the answer to 1 and 2 is yes, the user story impact is Medium. If the answer to 3 is yes, the user story impact is also Medium. In all other cases, the user story impact is Low. High impact user stories do not exist in the current context of Vendor B. The product owner maintains the asset inventory.

Internal Security Team: Vendor B does not maintain an internal security team. The most senior developers are tasked with identifying and prescribing security practices for design and development.

Underlying SDA Method: Like Vendor A, Vendor B utilizes STRIDE-based SDA that examines data flow diagrams, as described in Section 4.1.2.

Tools and Training: Vendor B does not maintain an internal security team and instead annually sends members of the development teams to application security-related workshops and conferences. Employees who attended are then tasked to construct training materials that present the most relevant knowledge from the conference for the context of the organization. Time is set aside every sprint for team members to spend on improving their software engineering skill, which must be focused on security at least once per quarter.

Evidence Structure: User stories with medium impact are tagged as “security”. After SATMUS completes for these stories the outputs of SATMUS are referenced in a note tied to the examined story (if the output is a research spike or a new user story).

4.3 Process Evaluation

In this section, we present our evaluation of the SATMUS process. Following the results of the systematic literature review on security threat analysis methods (Tuma et al., 2018), we utilize the two most common evaluation methods for security analysis techniques - comparative analysis and case study analysis.

In Section 4.3.1, we refer to our previous work (Luburić et al., 2018a), and we examine how SATMUS compares to the Scrum security requirements engineering techniques found in literature, concerning the Simplicity, Accountability, and Guidance requirements (examined in Section 2.2). In Section 4.3.2, we present a set of user stories from Vendor A (described in Section 4.2.2) and detail how their version of SATMUS executes and what are the outcomes of this analysis. We perform a similar exercise in Section 4.3.3, where we detail the process of applying SATMUS to a set of user stories from Vendor B. These case study analyses both evaluate our process as well as offer guidance for the utilization of SATMUS.

4.3.1 Comparative Analysis

In (Luburić, 2018a), we analyzed the literature to extract requirements for our process to maximize its real-world applicability. To reiterate, the requirements are:

1) Simplicity – The security analysis technique should not mandate the introduction of additional types of documentation or job roles. Additionally, the method should require as little training as possible to practice effectively;
2) Accountability – The security analysis technique should be integrated into the standard agile development workflow and should produce visible and quantifiable action items;
3) Guidance – The security analysis technique should be fully documented, offering illustrative examples of its use, as well as advice for integration into different real-world contexts.

Furthermore, we examined how different Scrum SDA methods fulfilled these requirements, as described in Section 2.2. Table 13 presents our previous results and adds SATMUS to provide a comparative analysis. Here we summarize the extent to which each method has achieved the goals of Simplicity, Accountability, and Guidance.

Table 13 SATMUS compared to different Scrum SDA techniques in relation to Simplicity, Accountability, and Guidance

Method	Simplicity	Accountability	Guidance
SATMUS	Partial	Full	Full
Abuser Stories (Peeters, 2005)	Full	Partial	Partial
SEAP (Baca et al., 2015)	Partial	Full	Insufficient
Secure Scrum (Pohl and Hof, 2015)	Full	Partial	Insufficient
Security Backlog (Azham et al., 2011)	Partial	Full	Partial
S-Scrum (Mougouei et al., 2013)	Full	Insufficient	Insufficient
Agile Sec. Framework (Singhal, 2011)	Insufficient	Full	Partial
Sec. assurance case (Othmane et al., 2014a)	Insufficient	Full	Partial
VAHTI-Scrum (Rindell et al., 2015)	Insufficient	Full	Partial

Regarding Simplicity, we believe we have partially fulfilled this requirement. While our method is flexible regarding new roles (the internal security team is optional) and documentation structure (evidence can be lightweight, as well as the asset inventory), we cannot avoid the need for some dedicated security training or the integration of tools to supply the security knowledge. A checklist approach that replaces the need for the training is feasible if attack pattern catalogs are selected and prepared. However, this introduces a new document and a process for maintaining and updating it. Therefore, our recommendation is to invest in dedicated security training, especially that which is constructed through the SDA Framework described in Section 3.

Regarding Accountability, we believe we have fulfilled this requirement entirely. Our process has actionable outputs and clear steps that are conducted by the Scrum team. The only caveat is the threat decomposition step of the threat analysis activity, which lacks a concrete stopping condition. Therefore, it is our recommendation to timeslot this activity, where time is invested into threat decomposition depending on the impact of the examined user story.

Regarding Guidance, we believe we have fulfilled this requirement entirely. Section 4.1 documents our process, offering descriptions of the inputs and outputs to the process, as well as details regarding each activity. Furthermore, we describe how to tailor the process to different real-world contexts, and two case study implementations in Section 4.2, noting significant decisions that need to be made when implementing SATMUS in the organization. Finally, in Section 4.3.2 and 4.3.3, we present examples of SATMUS execution on several real-world user stories.

Comparing SATMUS to other SDA techniques, our method stands out concerning the Guidance requirement. Furthermore, SATMUS is the only technique apart from SEAP that achieves full Accountability while being able to address the Simplicity requirement partially.

4.3.2 Vendor A Case Study Analysis

This Section illustrates the execution of SATMUS, as defined in Section 4.1 and augmented in Section 4.2.2. Here we execute SATMUS for the user stories *Introduction of Personally Identifiable Information* and *Service-to-Service Password-Based Authentication*.

Introduction of Personally Identifiable Information

The examined user story has the following text:

"*As a utility controller, I want to manage maintenance teams so that I can track and assign available teams to handle and fix utility outage incidents.*

Acceptance criteria:

- *CRUD operations for maintenance teams and team members, following the agreed upon data model;*
- *Ability to examine and search for all available maintenance teams;*
- *Ability to assign available teams to utility outage incidents.*"

Determine Affected Assets: The Scrum team goes over the part of the asset inventory relevant for them and notes that maintenance team member data, which is personal data and therefore protected under GDPR (GDPR, 2016), has the highest security goal value (a 3 for the security goal of confidentiality). The Scrum team notes other assets affected by the user story, as they will be mapped to the data flow diagrams.

Examine Environment: The new code will communicate with a database over an object-relational mapper, where the code will be accessible to internal non-administrative users (the utility controller). Based on this information, and the calculation formula used by the organization, the Scrum team grades the sensitivity of the environment as 2.

Calculate User Story Impact: The user story has a High impact, based on the formula described in Table 12.

Obtain Expertise: The product owner contacts the internal security team to confirm the impact of the user story. The security team confirms the result of the calculation and representatives from both teams schedule a meeting where most of the threat modeling shall be conducted.

Analyze Module: During the meeting, the team draws data flows on a whiteboard, as presented in Figure 10. They discuss the use cases and map the affected assets to the elements of the diagram.

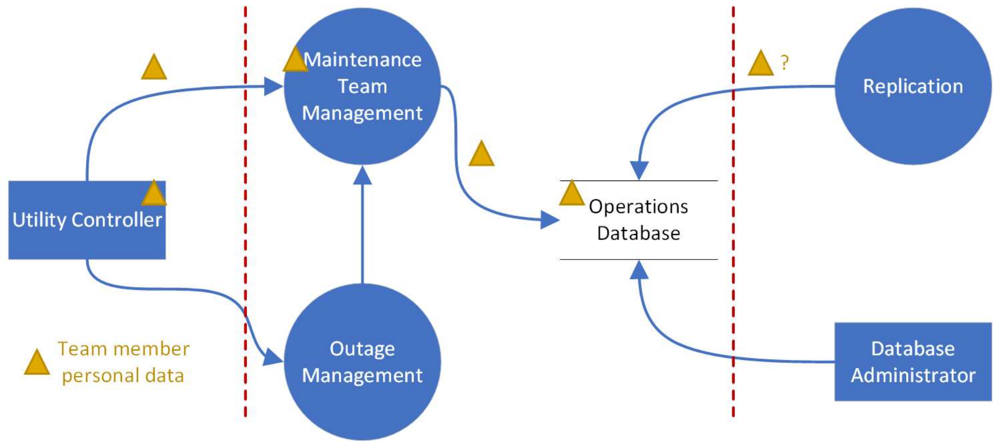

Figure 10 Data flows of maintenance team member PII

As the product the Scrum team is working on is complex and developed by dozens of other Scrum teams, system components and data flows exist with which the analysis team is unfamiliar. In this case, the Replication Service copies some of the content of the Operations Database to other parts of the system. The problem is that the analysis team is unsure if it will copy the new team member personal data. Therefore, the first output of this SATMUS execution is defined, even before threat modeling completes – a research spike which states that the data flow needs to be completed by consulting with colleagues from other teams, after which a threat and mitigation analysis needs to be conducted for any potential additions to the original DFD.

Analyze Threats: The first step of threat analysis, threat identification, is conducted following STRIDE (Shostack, 2014b). The team places focus on the trust boundary between the system and the users (in this case, the utility controller and database administrator) and the boundary between components under the Scrum team's jurisdiction and components which fall under the jurisdiction of other teams (in this case, the replication service). Each identified threat is decomposed through misuse cases and aided by the security knowledge of both the Scrum team (obtained through dedicated training) and the security expert. For each identified attack, the team notes mitigations which exist in the system. Each unmitigated attack vector is listed for mitigation analysis. In the context of the analyzed user story, two threats are identified with unmitigated attacks. These include:

- Elevation of privilege, where a non-controller user that uses the same UI as the utility controller can read team member data;
- Repudiation, where a database administrator can read the content of the database without the system logging this action.

Specify Mitigations: As a result of the two discovered vulnerabilities, the team plans mitigations. They issue two more outputs of the SATMUS analysis – an additional acceptance criterion "CRUD operations of team member data require access control checks for permissions tied to the utility controller role", as well as a new user story, which requires that logging of database administrator actions, especially database reads of team member data, needs to be carefully tracked.

Document Evidence: The team selects a member to fill the regulatory compliance templates. The member notes the impact of the user story, draws the examined data flow diagrams, lists identified threats and their mitigations, and provides references to each output of SATMUS (the research spike defined during data flow analysis and the new user story and acceptance criteria outlined during mitigation analysis). Finally, the member adds a note to the initially examined user story that references the new document.

Service-to-Service Password-Based Authentication

The examined user story has the following text:

"*As a utility company, I want to authenticate the utility management software to relevant internal systems, so that I can protect my systems from rogue services.*

Acceptance criteria:

- *The utility management system provides a username and password to authenticate to the customer's internal systems before every request;*
- *The password needs to be protected.*"

Determine Affected Assets: The Scrum team goes over the part of the asset inventory relevant for them and notes that passwords, a sensitive asset in any system, has the highest security goal value (a 3 for the security goal of confidentiality).

Examine Environment: The new code is a security control (i.e., authentication) and communicates with entities external to the software. The Scrum team grades the sensitivity of the environment as 3.

<u>Calculate User Story Impact</u>: The user story has a High impact, based on the formula described in Table 12.

<u>Obtain Expertise</u>: The product owner contacts the internal security team to confirm the impact of the user story. The security team confirms the result of the calculation and representatives from both teams schedule a meeting where most of the threat modeling shall be conducted.

<u>Analyze Module</u>: During the meeting, the team draws data flows on a whiteboard, as presented in Figure 11. They discuss the control flow and map the affected assets to the elements of the diagram.

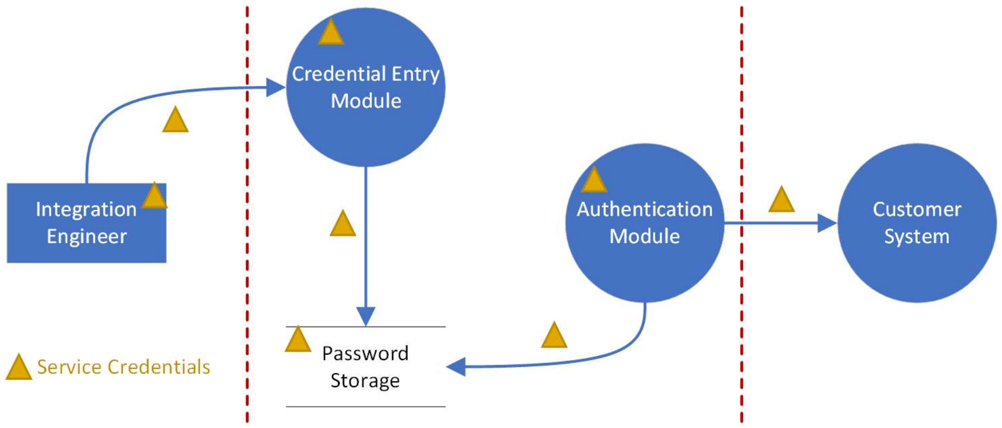

Figure 11 Data flows of service passwords

The main discussion is about protecting the passwords throughout their lifecycle. The data flows need to be precisely mapped and understood to identify all threats.

<u>Analyze Threats</u>: The team conducts threat identification following the STRIDE method. They place focus on the trust boundary between the system and the users (in this case, the integration administrator) and the boundary between the developed software and other systems (to which the solution authenticates). The Scrum team and the security expert perform threat decomposition. For each identified attack, mitigations which exist in the system are noted. Each unmitigated attack vector is listed for mitigation analysis. In the context of the analyzed user story, two threats are identified with unmitigated attacks. These include:

– Information disclosure, where any user with access to the machine can read the password storage (passwords cannot be hashed, as they need to be read by the integration service and supplied to the customer's systems);

- Repudiation, where all actions related to password manipulation need to be logged for security monitoring purposes.

<u>Specify Mitigations</u>: The team issues two sets of outputs of the SATMUS analysis. The first is two additional acceptance criteria ("Log all access and changes to the password storage, including the identity of the action executor" and "Establish access control on the password storage"). The second is a research spike to determine how to best utilize cryptography for protecting passwords at rest (without the option of using hash functions).

<u>Document Evidence</u>: The team selects a member to fill the regulatory compliance templates. The member notes the impact of the user story, draws the examined data flow diagrams, lists identified threats and their mitigations, and provides references to each output of SATMUS (the additional acceptance criteria and the research spike). Finally, the member adds a note to the initially examined user story that references the new document.

4.3.3 Vendor B Case Study Analysis

This Section illustrates the execution of SATMUS, as defined in Section 4.1 and augmented in Section 4.2.3. Here we execute SATMUS for the user stories *Calendar National and Religious Holidays*, *Employee Vacation Monthly Report*, and *Support for Electronic Payment*.

Calendar National and Religious Holidays

The examined user story has the following text:

"*As an HR worker, I want to examine and enter national and religious holidays in the corporate work calendar so that I can manage employee vacation days.*

Acceptance criteria:

- *Introduce two types of events to the corporate work calendar (national holiday and religious holidays);*
- *Expand event creation forms to include these types and automatically set them to repeat each year.*"

<u>Determine Affected Assets</u>: The Scrum team determines that no assets are affected by the user story.

<u>Examine Environment</u>: The new code interacts with HR workers and is contained in the calendar app. It does not generate flows which cross a significant trust boundary.

Calculate User Story Impact: The user story does not introduce a security control and, based on the previous two activities, it is deemed as having Low impact. No further security analysis is conducted.

Document Evidence: The product owner confirms orally that the user story does not require additional security considerations.

Employee Vacation Monthly Report

The examined user story has the following text:

"*As an HR worker, I want to generate monthly reports that show which employees have taken vacation days so that I can examine trends and aid planning management.*

Acceptance criteria:

- *A new report type can be selected from the generate reports menu;*
- *The system creates a report according to the agreed-upon template.*"

Determine Affected Assets: The Scrum team notes that the new code will manipulate personal information of employees.

Examine Environment: The new code will be called by an HR worker where the result of the code execution is a file which will be placed in the file system.

Calculate User Story Impact: The Scrum team grades the user story as having Medium impact, as it works with user-supplied input and handles personal data.

Analyze Module: The team draws data flows on a whiteboard, as presented in Figure 12. They discuss the control flow and map the personal data to the elements of the diagram.

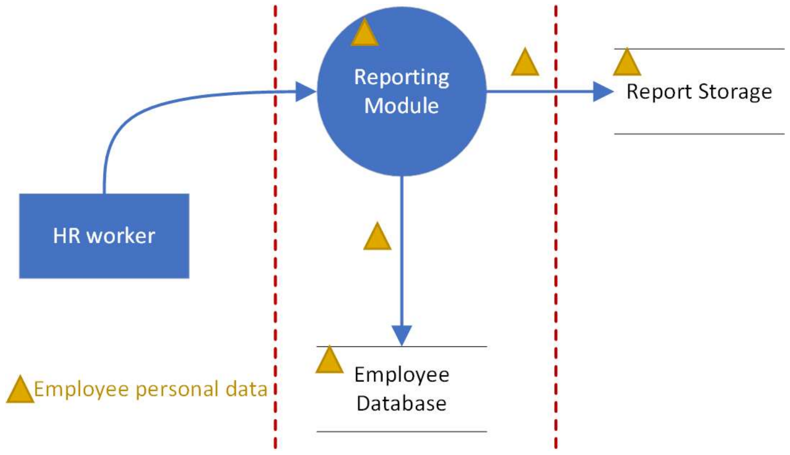

Figure 12 Data flows regarding employee reporting

Analyze Threats: The team conducts threat identification following the STRIDE-per-interaction method. They place particular focus on the trust boundary between the system and the users (in this case, the HR worker) and the boundary between the software and the file system. In the context of the analyzed user story, two threats are identified with unmitigated attacks. These include:

- Information disclosure, where any user with access to the file system can read the reports;
- Repudiation, where the system does not log report generation events.

Specify Mitigations: The team issues a single output of the SATMUS analysis. Two additional acceptance criteria ("Log report generation events, including the identity of the action executor" and "Establish access control lists on the report storage system").

Document Evidence: The user story is tagged as *security*.

Support for Electronic Payment

The following user story can be considered a feature, due to the scale of the work which goes beyond the traditional user story scope. It has the following text:

"*As a buyer, I want to pay for my shipped goods through the mobile shipping app, so that I can quickly pay for the goods upon successful shipping.*

Acceptance criteria:

- *The in-app wallet should support Master Card and Visa wallet top-ups;*
- *The app scans QR code from shipped container to verify it is the correct item.*
- *Upon success, the app automatically subtracts the appropriate amount from the wallet.*"

Determine Affected Assets: The Scrum team notes that the new code will manipulate payment card information and cardholder data. Although this is not an asset in their list, the product owner is aware that this data is sensitive and quickly learns about the PCI DSS (PCI, 2018).

Examine Environment: Due to the sensitive nature of the financial functions, the new code will require some form of security controls, notably cryptography, careful auditing, and excellent access control.

Calculate User Story Impact: The Scrum team realizes that their standard calculation formula does not hold for this unique feature. Due to the security implications of the feature, the team sees the new development as having a High security impact.

Obtain Expertise: The product owner consults with product management to determine the best strategy for securing the new feature. Due to the risk introduced by the new feature and insufficient expertise in the organization, the product management decides to outsource development to a third-party organization specialized in PCI DSS-compliant development.

Threat Model: The team requires an up-to-date threat model of the externally-developed component as one of the deliverables at the start and end of the project. Furthermore, they request guidance for securing parts of their component, which interface with the externally-developed component. With the aid of the third-party vendor, they define new user stories.

Document Evidence: The team archives the threat model from the third-party vendor and tags the user stories derived from the threat model as *security*.

4.4 Process Analysis

In this Section, we discuss the results of our work, examining the limitations of the SATMUS process and the implications of all the work presented in Chapter 4. In Section 4.4.1, we discuss the limitations of our method by consulting with the list of limitations that affect agile security requirements engineering methods as defined by Villamizar et al. (2018). In Section 4.4.2, we discuss the implications of the SATMUS process and how it addresses agile secure software engineering.

4.4.1 Limitations

To assess the limitations of our process, we examine the list of constraints determined in the systematic mapping study of agile security requirement engineering methods (Villamizar et al., 2018). The authors identified several areas of limitations, including the environment, people, effort, and resource investment. To this list, we add completeness, as a metric that assesses the degree to which a method achieves the security of a product.

Environment limitations arise from the Scrum framework, as release cycles are short, making addressing of all security requirements difficult. Indeed, SATMUS can produce new user stories and research spikes, tasks which might not be included in the current sprint or even the current release cycle. However, while having a completely secure product might be something to strive towards, security is one of many sources of requirements which bring value to an organization. The bottom line is that if security is given sufficient attention, it will get implemented, but this is entirely up to the product management and their risk assessment. Therefore, we believe that the Scrum environment does not impose limitations on security, but instead prioritizes it accordingly, so long as the product management is aware of the need for security. For this assumption to hold, it is the responsibility of every Scrum team member to champion security awareness and fight for higher-quality development.

People limitations are related to the lack of security-related skill in the Scrum team. As we pointed out earlier, dedicated training is required to practice SATMUS efficiently. This is a limitation of our approach (because of which SATMUS only partially fulfills the Simplicity requirement), which can somewhat be mitigated by a knowledge base and tool support.

Effort and resource limitations are concerned with the introduction of new roles and the overhead introduced by security requirement engineering methods. Furthermore, lack of guidance is listed under this category. The nature of SATMUS allows it to be tailored to organizations of varying degrees of security and agility requirements. A SATMUS implementation can be a lightweight process that rarely triggers significant investment of effort, and it can be a rigorous process that ensures that a product is secure by design. For example, an organization with little security requirements can use the calculation formula presented in Section 4.1.2, while reconfiguring Table 12 to produce a Medium impact output only when both axes have a value of 3 and a Low impact output in all other cases. Therefore, we believe that SATMUS introduces just enough overhead, according to the organization's needs. Furthermore, no new roles are mandated by SATMUS. Finally, we offer significant guidance to tailor and execute SATMUS through this document.

Regarding completeness, the question is can SATMUS be used as standalone to ensure that a product is secure. The answer is no, and this can be considered a limitation of our approach. SATMUS is concerned with translating high-level security requirements to low-level design decisions and implementation tasks revolving around user stories. Importantly, the quality of the *Analyze Threats* activity can significantly vary based on the expertise of the analysts and, depending on the investment in education, and some time may pass before the team can adequately identify and decompose threats. Importantly, SATMUS achieves a reasonably secure design only if it is applied to every story, through both the Definition of Ready and the Definition of Done, as described in Section 4.1.1. As discussed by Othmane and Ali (2016), frequent changes to the code and the initial design found in Scrum can quickly make the initial threat model obsolete. Therefore, it is necessary to frequently reevaluate the threat model and ensure that new threats are discovered and addressed.

Additionally, SATMUS fails to address code-level vulnerabilities, and it is not concerned with verifying that enough security is present in the product (e.g., through security and penetration testing). SATMUS is applied between determining high-level security requirements and implementing the code. To combat this limitation, we recommend that organizations utilize tools that aid in code-level vulnerability discovery, such as static code analysis and fuzz testing. Following recent trends, DevSecOps (Myrbakken and Colomo-Palacios, 2017) should be explored to complement SATMUS. Tools integrated into the DevSecOps cycle help prevent code-level vulnerabilities and can enhance the maturity of SATMUS by supplying it with information on what categories of threats and attacks are not sufficiently addressed by the Scrum teams. Finally, penetration testing can be conducted to ensure that a product is resistant to attacks, though this entails hiring third-party auditors.

4.4.2 Implications

Throughout Chapter 4, we presented the SATMUS process, starting from the high-level overview and discussion about its internal activities, and moving on to specific adaptations of SATMUS and examples of its execution. By focusing SATMUS around existing development artifacts and activities (e.g., user stories, research spikes), we were able to design a method that is harmonized with Scrum development practices. Firstly, the outputs of SATMUS are requirements defined just like any other requirement in Scrum development, which can, therefore, be prioritized, verified, and validated following regular development practices. Secondly, the requirement for new knowledge and expertise in the Scrum team is limited to the *Analyze Threats* activity. However, this is not an insignificant requirement, as the quality of *Analyze Threats* activity significantly affects the security of the product. The investment in security education and tools directly correlates to the efficiency and value of threat identification and decomposition, as well as appropriate mitigation planning. Based on the experience with Vendor A and B (described in Sections 4.3.2 and 4.3.3), a good strategy is to start with the basic threat analysis (e.g., STRIDE threat discovery with no additional decomposition) and build on that over time.

Another benefit of tying the SATMUS process to the user story's definition of ready is that it guarantees that the threat model will be incrementally updated, reflecting the design changes that can be frequent in agile development. On the one hand, this avoids the common problem with design documentation in agile development, where the initial design is created and then quickly becomes deprecated and forgotten (Prause and Durdik, 2012). As teams define the structure of their threat model (e.g., a series of photos of the whiteboard, a sophisticated model using a tool), they can maintain agility while incrementally updating the security view of their components. On the other hand, this allows the team to quickly produce a body of evidence for security assurance, requiring negligible effort to compile the report when the client or third-party requires it.

The SATMUS process enables the security analysis and threat modeling of the developed software in the Scrum development workflow. It requires some investment to understand, integrate into the environment, adapt it to maximize efficiency, usability, and value, and finally roll out on the whole organization, providing training and aid where necessary. For this initial investment, SATMUS requires a champion which will drive its adoption and continuously improve its quality. Once installed, SATMUS produces just enough overhead to the development teams to make sure they adequately address the security of the product, while maintaining both their agility and efficiency.

5 Discussion

In this Chapter, we combine the work presented in Chapters 3 and 4 and discuss how to integrate it into the agile development workflow to build teams that can efficiently perform security design analysis. Supported by the SDA Training Framework and the SATMUS process, development teams incrementally construct a secure software design, from the high-level system architecture to the low-level code design. While the SATMUS process provides the basic structure for the analysis, the SDA Training Framework enhances its quality and supports its continuous improvement.

In Section 5.1, we examine the integration and use of both the SDA Training Framework and the SATMUS process into the agile development process. Here we explain how organizations can adopt these two methods to introduce efficient SDA into their workflow. We examine how our work expands the responsibilities of the traditional Scrum roles, how it interacts with the key Scrum events, and how the application of our methods in individual Scrum teams can achieve the security of the product developed across multiple teams. Section 5.2 demonstrates how our work compares to well-established security development lifecycle approaches. Here we map our methods to the different phases and practices of these SDLs and discuss how our methods, that focus on security requirements and design, interact with the rest of the SDL.

5.1 SDA Training Framework and SATMUS in Agile

The SATMUS process enables incremental security design analysis by examining new user stories and building upon the existing threat model of the developed product. When the Scrum team is adept in SATMUS and is starting a new project, the continuous application of SATMUS builds a product that is secure by design. The issue is introducing SATMUS to a Scrum team that lacks knowledge in SDA and is in the middle of developing a product. Introducing SATMUS to such a context requires both training the Scrum team members how to perform SDA and constructing a baseline threat model for the existing product, which SATMUS can then build upon.

The SDA Training Framework is used to accomplish both goals and to continuously improve the SATMUS process by enhancing the SDA knowledge of the Scrum team. In Section 5.1.1, we describe the process of the initial adoption of the SDA Training Framework and SATMUS process, noting how the different Scrum roles participate in the adoption process. In Section 5.1.2, we describe areas for continuous improvement. Here we examine how the SDA Training Framework aids in the continual development of knowledge and expertise, how the SATMUS process can be optimized in individual teams, and how the SATMUS process should be supplemented in larger organizations, to achieve adequate security across the developed product.

5.1.1 Initial Adoption

Organizations can integrate efficient SDA into their workflow by combining the SDA Training Framework with the SATMUS process. Figure 13 illustrates the process of integrating SDA to an organization's agile development workflow by using our methods. For each activity, we note the role that drives the activity (on the left-hand side of the activity), as well as any other roles that require significant involvement (on the right-hand side of the activity).

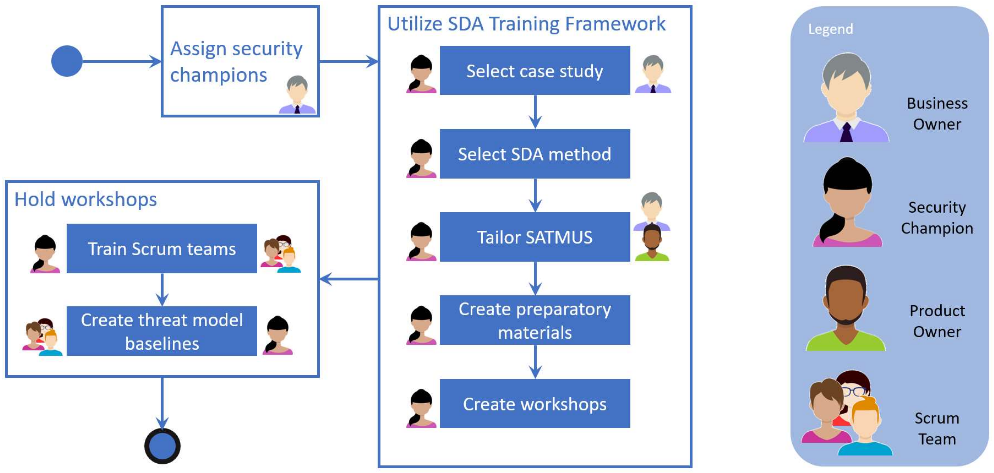

Figure 13 Adoption of the SATMUS process and SDA Training Framework

To start the process, the business owner must assign security champions to drive the adoption and development of the SDA Training Framework and SATMUS process. The security champion is an informal role that must possess a certain level of security knowledge that they can distribute onto the organization through the SDA Training Framework. They can be internally built, for example by using senior developers that have performed security work during their career and are knowledgeable about the subject, or they can be externally hired, as domain experts.

The security champions utilize the SDA Training Framework described in Chapter 3 with one crucial difference – they do not build the case studies and instead select them from the products in development. With the aid of the business owner, they select a suitable product for the case study and note the

Scrum teams involved in its development. The security champions select the SDA method that they wish to use as part of the SATMUS process, based on both the selected case study and the needs of the organization. While the SDA method is the most complicated aspect of the SATMUS process, champions need to consider the whole process when constructing the workshops. Importantly, SATMUS tailoring must be completed to include the final process in the training workshops. Champions should consult with the product owners of the selected Scrum teams and the business owners to determine:

- Inventories of affected assets for each Scrum team, based on the security requirements for the product;
- The rigor of the calculation formula, based on the risk tolerance of the organization;
- The suitable evidence structure, based on the security assurance requirements of the organization;
- Investments for tools to support SDA and strategies related to obtaining expertise for high impact development.

Next, the security champions determine the relevant security concepts and create preparatory materials in the form suitable for the organization. For example, if an organization uses an internal e-learning platform, the champions can record lectures detailing the security concepts for easy distribution and reuse.

With a defined SATMUS process, the constructed preparatory materials, and a realistic and relevant case study, the security champions can construct the training workshops. At this stage, champions should consider the different roles in the Scrum team, as each role has a different part to play in SATMUS. The product owner must be aware of the purpose of the process, how to manage the related asset inventory, and how to handle and prioritize the outputs of SATMUS. The Scrum master must be aware of the general flow of the process. Their goal is to facilitate the efficient execution of SATMUS by recognizing when development team members require better SDA training and tools and communicating these issues with the business owner. Furthermore, Scrum masters can utilize the Scrum of Scrums to coordinate SDA across multiple teams when the need arises, as discussed in Section 4.1.2, *Analyze Module*. Finally, the development team requires a deep understanding of SATMUS and the underlying SDA method. On the one hand, it enables them to define a secure design, and on the other, it guides their security code review and testing activities, as described in Section 5.2.3. Therefore, most of the workshops should be directed towards the development team.

Additionally, the security champions should structure the workshops for specific Scrum teams, focusing on the part of the case study relevant for that team. Depending on the product, one team might be focused on web application development, while another might produce code that solely works with databases and data stores, which is why effort is required to increase relevance for the Scrum team members.

Importantly, the workshops should be designed with two goals in mind. The first goal is to train the Scrum teams on how to conduct SATMUS, focusing on the SDA activities conducted by the development team. The second goal is to kickstart the creation of the baseline threat model that the team can use and expand as the product expands. Once the workshops are prepared, the champions can administer them to the Scrum teams. Through these workshops, the baseline threat model for the Scrum team can be started or even completed. While it might contain only the first elementary set of threats (e.g., high-level STRIDE threats), constructing the baseline is a necessary first step (Tarandach, 2019) which can then be expanded as the Scrum team becomes more knowledgeable about the subject.

As with most security-related initiatives, several crucial prerequisites that must be addressed to ensure the successful integration of SDA into the development workflow. First, the top-level management needs to clearly state their support for SDA and the implementation project surrounding it. Secondly, everyone taking part in SDA, from the product owner to the Scrum master, must be aware of their responsibilities. Finally, care and patience must be practiced, and steps need to be carefully considered to gain buy-in from the development teams and minimize the risk of their aversion to SDA.

5.1.2 Continuous Improvement

Once the Scrum teams are familiar with the SATMUS process, they can look for areas where they can optimize the process and adapt it to better suit their workflow and bring higher value to the organization. The primary candidates for continuous improvement include:

- The SDA knowledge and expertise of the development teams;
- The details of the SATMUS process;
- The security of the developed product.

The quality of the SDA, and in turn, the security of the product, rely heavily on the expertise of the analysts performing the security design analysis. Therefore, an ongoing cost for the organization practicing SDA is the construction of new workshops through the SDA Training Framework. Introducing new workshops can be the responsibility of the internal security team (if one is available) or the informal team of security champions. They need to research the changing threat landscape and distribute the new relevant knowledge to the development teams.

Following the example of the case study organization presented in Section 4.3.3, team members can attend external security workshops or conferences and use the knowledge they obtained to construct internal workshops through the SDA Training Framework. Likewise, Scrum teams might obtain security knowledge through their regular work, where the organization can benefit if this knowledge is distributed. The Scrum masters need to communicate and share this information with other Scrum masters and collaborate with the business owner to facilitate the investment in new workshops.

Finally, the organization can hire external domain experts to perform security audits, where the vulnerabilities and design flaws discovered during the audit serve as valuable input for new educational workshops. Furthermore, the domain experts can help prepare the workshops by providing knowledge, preparatory materials, and advice.

Regardless of the source of new knowledge and workshops, Scrum teams need to reevaluate their threat models every time they obtain new knowledge through the workshop. When aided by threat analysis tools, Scrum teams should examine their threat models with each major update of the attack pattern library, to ensure that the model is in line with the actual threat landscape.

The next area for improvement is related to the details of the SATMUS process. The teams might wish to modify SATMUS and tailor it to the way they practice Scrum development. Importantly, any team-level changes to the SATMUS process should not harm the general strategy of the organization regarding SATMUS, and reasonable justification must be provided for each such change. The internal security team or the security champions that introduced SATMUS can assess the justification provided by the team for the change and consult the business owner. While teams might wish to modify the SATMUS process itself, they also might wish to modify how it interacts with their Scrum development. For example, the team might collect all user stories with medium or high impact and collectively analyze them when grooming the sprint backlog.

Notably, when teams develop a more efficient version of SATMUS, they should push for changes across the organization through the business owner or Scrum of Scrums event. Additionally, the organization should consider developing internal tools to support their version of SATMUS, automating whatever is possible and integrating it into their build cycles where appropriate.

The final area for improvement is tied to the goal of SDA, which is the construction of a secure software product. While a single Scrum team that is trained using the SDA Training Framework and following the SATMUS process might hope to secure their part of the product, additional effort is required to coordinate the security of the complete product. This responsibility might be delegated to the internal security team, but for organizations that lack such resources, the Scrum teams need to coordinate to implement security efficiently and achieve product-wide security assurance.

Baca et al. (2015) point out that the underlying issue with integrating agile methods and security is that individual teams are not familiar with the details of the complete product. As security is often a cross-cutting concern, the efficient way to introduce security controls into the software is to build them into its infrastructure components. The less efficient alternative is to introduce the same security controls to multiple higher-level components, increasing maintenance cost and the possibility of developer error. However, for individual Scrum teams, it might not be obvious when integrating a security control is their responsibility, and when it might be offloaded to other teams. To remedy this, the Scrum master should communicate these issues during the Scrum of Scrums event. Another issue is that a security control might not fit into the infrastructure initially, but over time as the system evolves, this might become the preferred option. Rindell and Holvitie (2019) mark this type of issue as technical debt, where the software needs to be redesigned, and the code refactored to increase its quality.

5.2 SDA Training Framework and SATMUS in the SDL

The security of a software product requires the application of various security activities throughout the product's development lifecycle. Secure software engineering entails far more than SDA, from secure coding and testing to security management of both regular development and issues found in production.

While the SDA Training Framework and SATMUS aid with security requirements engineering and secure design construction, they also interact with other practices of the security development lifecycle. Here we examine how our work integrates into the SDL defined by Microsoft (Howard and Lipner, 2006) and the IEC organization (IEC, 2018a). Notably, the SATMUS process directly maps to segments of both SDLs concerned with threat modeling and security design analysis, as discussed in Sections 1.3.1 and 2.2.3. Likewise, the SDA Training Framework maps to the *provide training* practice, described by Microsoft, and the *security expertise* required by the

IEC 62443-4-1:2018 standard. Both practices require that personnel receive adequate security-related training for their job role, which aligns with the goal of the SDA Training Framework.

We further explore these SDLs to determine practices with which our work interacts and discuss the nature of that interaction. In Section 5.2.1, we examine how security requirements serve as valuable input for the SATMUS process and the SDA Training Framework. In Section 5.2.2, we discuss how secure design concepts, such as security design patterns and secure design principles, can be built into our work. Section 5.2.3 describes how our work guides secure implementation and testing activities. Finally, in Section 5.2.4, we detail how our work answers the requirements related to security management.

5.2.1 Security Requirements

Microsoft's *define security requirements* practice entails the specification of both explicit and quality security requirements (as defined in Section 1.3.2) that arise from regulatory compliance, internal quality standards, reviews of previous incidents, and known threats. The IEC 62443-4-1:2018 (IEC, 2018a) specifies a similar set of security requirements sources in its *specification of security requirements* practice. In addition to these sources, we add stakeholders (possibly aided by security consultants), internal Scrum teams, and threat intelligence services as a source for quality security requirements.

Security requirements are a crucial input for the SATMUS process. One goal of SDA is to map explicit security requirements onto the developed software, and another is to decompose quality security requirements into actionable work items. For SATMUS, both types of security requirements are mapped to assets, either explicitly or implicitly. Depending on the tailoring decisions, the asset inventory can include traceability between the assets and the security requirements (Luburić et al., 2018b).

Security requirements are likewise an essential input for the SDA Training Framework. When a software vendor acquires a project that entails compliance with a new security standard, the workshop constructors must examine the standard to determine the relevant security concepts and the appropriate SDA method. From this information, they can construct preparatory materials and the set of workshops to train the development teams.

5.2.2 Secure Design

Regarding secure design, Microsoft proposes a practice to *establish design requirements*, where the goal is to determine organization-wide implementations of security design patterns (Yoshioka et al., 2008; Uzunov et al., 2012). Developers can then use these implementations instead of building their security controls.

Security design patterns present mature solutions for a class of problems, much like software design patterns. Security design patterns, like authorization and secure communication, mitigate a class of vulnerabilities and attacks. A security design pattern can be more abstract (e.g., authorization) or more concrete (e.g., role-based access control), forming a hierarchy, where an abstract pattern can have multiple children in the form of concrete designs that can be translated into code (Uzunov et al., 2012). For example, input validation is a pattern that has specific implementations for preventing SQL Injection (i.e., prepared statements), XSS (i.e., input validation paired with output encoding), and buffer overflows.

The IEC 62443-4-1:2018 standard requires that all development teams conduct design activities following security design principles (Saltzer and Schroeder, 1975; Ross et al., 2016). While security design patterns present solutions for a class of problems, security design principles do not address a specific set of problems and are instead applicable for most problems and solutions. They serve as guidelines for all design activities. For example, the principle of least privilege states that each entity should have privileges to accomplish its specified functions and no more. While conceptually simple, adherence to this principle can require significant additional development. The apparent restriction is to reduce active permissions of user roles to the functions that are required by such roles, but additional consideration can reveal that the time-frame when a function should be available to the user can also be defined. When paired with the principle of complete mediation, this type of access restriction should be applied across the whole system, including application-level access, OS-level access, and potentially physical access.

Both security design patterns and principles serve as valuable input for the SDA Training Framework and the SATMUS process. The patterns and principles are essential security concepts that must be addressed by the workshops created using the SDA Training Framework. They can be covered indirectly, as part of a workshop that examines some aspect of SDA, or even directly, where the workshop's focus is on a set of patterns or principles. By examining least privilege, a workshop can be constructed to explore the different areas of the case study and define how privileges can be restricted to enable the standard business functionality and nothing else. SATMUS directly employs both patterns and principles, as the goal of SDA is to produce a secure design that fulfills the security requirements, which can only be accomplished by utilizing security design patterns and adhering to security design principles (Ross et al., 2016).

5.2.3 Secure Implementation and Security Verification

Secure implementation practices, as defined by the examined SDLs, include the application of static code analysis tools and security-focused code reviews to ensure compliance with the secure coding standard and that no code-level vulnerabilities are introduced to the developed software. These practices look for flawed code design and use of insecure constructs that an attacker might exploit.

While code-level security is a requirement for secure software in general, it is especially important in the sensitive areas of the code, which includes code exposed to a large attack surface and code that manipulates critical assets. As SATMUS examines both the assets and the environment for all new development, it identifies sensitive code as medium or high impact user stories. Therefore, SATMUS can guide security code reviews and define areas of the code that require more effort to examine. For example, SATMUS can generate an acceptance criterion, where a team-wide code review must be conducted for the code implementing a particular user story. Regarding the SDA Training Framework, vulnerable code constructs, not aligned with the secure coding standard, should be included in the preparatory materials and utilized during the workshop assignments, to enrich the learning experience and include both secure design and secure implementation learning objectives.

Security testing is a broad area that includes practices related to the functional testing of security controls, the security testing of functional controls, and penetration testing. While the SDA Training Framework has no significant interaction with the security testing practices, the SATMUS process can guide security testing activities, in a similar way to the security code reviews. For example, SATMUS can define an acceptance criterion for a user story which states that fuzz testing should be conducted on a specific set of endpoints and that all significant vulnerabilities are resolved.

5.2.4 Security Management

Security management is concerned with, among other things, ensuring that security is sufficiently addressed across the product development lifecycle. As the goal of the SDL is to produce demonstrably more secure software, security management is concerned with achieving and maintaining this security assurance. As pointed out by Vivas et al. (2011), the security assurance case is built by using mature SDL procedures and extracting from their execution the evidence and argumentation needed to support the assurance case and to reasonably prove the software is secure. Importantly, if any part of the product proves insecure even with a well-developed case, it is crucial to understand why this happened and how these issues can be avoided in the future (Goodenough et al., 2007). The IEC 62443-4-1:2018 standard (IEC, 2018a) requires continuous improvement as part of the security management practice to address these issues.

The SATMUS process is flexible enough to support rigorous security analysis, depending on the context. By using the user story as the input for SATMUS, we ensure that all new development undergoes security examination. A vital prerequisite for satisfactory SDA is adequate training on the subject, and the SDA Training Framework is built to support that. The SDA Training Framework and the SATMUS process interact with each other to enhance the security work and provide continuous improvement. As the threat landscape changes, new workshops can be constructed and any significant changes to this landscape should trigger a reevaluation of the existing threat models.

6 Conclusion

Vulnerabilities in wide-spread hardware, operating systems, and applications, are making headlines daily and news about massive data breaches and successful hacks are only slightly less frequent. Software security is becoming a leading concern in the developed world, and much effort is put into building security into software during its development.

Throughout this work, we have focused on security design analysis, a cost-efficient practice concerned with security requirements engineering and secure design construction for the developed software. SDA examines a software's design and contrasts it with its functional and security requirements to determine enhancements required to make a more secure software product. This analysis is applied from the macro-level of the complete software architecture to the micro-level of software features and code design to achieve defense in depth of the developed product.

Specifically, we examined how the leading SDA methods interacted with modern agile software development practices and discovered issues that hamper the adoption of SDA to the agile workflow. Notably, we discovered that SDA is both difficult to teach and learn, resulting in its inefficient practice and abandonment by development teams. Furthermore, we found incompatibility issues between traditional SDA methods and the contemporary software development processes, specifically those following the Scrum framework. Based on this research, we defined the following research questions:

RQ 1. *How to efficiently train Scrum teams to perform security design analysis?*

RQ 2. *How to efficiently integrate security design analysis into Scrum development, providing the appropriate security assurance and visibility of security work?*

From these, we formulated a hypothesis that guided our work:

- *It is possible for a Scrum team to practice security design analysis throughout a software's development lifecycle, assuring that sufficient security is built into the software solution, provided that:*
    - *Adequate training is provided to the Scrum team to perform security design analysis efficiently.*
    - *The security design analysis is compatible with the Scrum development process, does not require the introduction of new roles to the team, and does not mandate the construction of heavyweight documentation.*

- *Security work is tangible and can be planned and prioritized like any other work item.*
- *Enough guidance and knowledge exist to adopt, use, and adapt the method to a specific organization's context.*

Section 6.1 highlights the contributions of this thesis, which can be grouped around the SDA Training Framework and the SATMUS process, as well as their interconnection. In Section 6.2, we examine further research and development opportunities.

6.1 Contributions of the Thesis

To address the first question, we inspected various approaches that have found success in teaching secure software engineering. While our primary goal was not to make a comprehensive catalog, Section 2.1 does provide a useful map of different teaching approaches for this domain, grouped around gamification, case study analysis, e-learning, and the hybrid flipped classroom.

From this analysis, we constructed the SDA Training Framework, as a structure for constructing educational workshops that have the learning goal of training developers on how to perform SDA. We combined the case study analysis and hybrid flipped classroom teaching methods and enhanced them with gamification and e-learning techniques, to construct labs that produce better learning outcomes when compared to the traditional classroom. Chapter 3 described the framework in detail, from its structure to the process of its use. To guide the use of the framework, we demonstrated the generation of six labs at a medium level of detail and examined the construction of a single lab in great detail. A minor contribution is the resulting set of labs, grouped around the STRIDE SDA method and hospital information system case study, described in the same Chapter. Near the end of the Chapter, we presented the controlled experiment and observational evaluations that proved that the labs formulated through the SDA Training Framework achieve better learning outcomes than traditional labs.

Throughout Section 2.2, we examined SDA methods used in the industry, as well as those proposed by the scientific community and contrasted these with process requirements issued by contemporary standards for security development lifecycles. Albeit incomplete, this knowledge base can help practitioners examine other SDA approaches to find a suitable candidate for their organization as well as highlight issues that might arise with these methods and SDA in general.

We defined the SATMUS process to address the second question. By examining applicability issues with proposed SDA approaches and SDA process requirements imposed by agile development and well-established SDLs, we have conceptualized a process that offers the security assurance provided by SDA, while being compatible with agile development practices. Throughout Chapter 4, we demonstrated the internals of our process, provided guidance for its use and tailoring. We evaluated SATMUS on two case study implementations, describing the tailoring decisions for both contexts, and illustrating the execution of the two instances of SATMUS on several user stories, offering further guidance.

With both research questions answered, we described how to combine our methods to integrate software security design analysis into the agile development process throughout Chapter 5. Here we also discussed how to integrate our methods into the well-established SDL processes and how our methods help answer some of the requirements imposed by SDL standards.

To summarize, the primary contributions of this thesis include:

- The definition of the SDA Training Framework and its enhancement, including guidance for its execution. The SDA Training Framework enables the generation of laboratory exercises for teaching SDA that achieve better learning outcomes than traditional labs.
- The definition of the SATMUS process, including guidance for its execution and tailoring. SATMUS enables the incremental development of threat models as the software changes, where organizations can tailor the process according to their needs, to define and prioritize security work accordingly.
- Instructions for integrating both the SDA Training Framework and SATMUS process to introduce SDA into the agile development workflow, construct the baseline threat models, and set up the foundation for the continuous improvement of SDA.

Minor contributions, which arose as a byproduct of the research, include:

- A catalog of approaches for teaching secure software engineering practices.
- A knowledge base of SDA techniques, requirements for these techniques, and their related issues.
- A set of laboratory exercises with the accompanying case study that can be used as part of a university course or a corporate training workshop.

Considering these contributions, we confirm the central hypothesis and meet all the introduced goals and expected results of this research.

6.2 Future Work

Importantly, SDA is not a standalone method that achieves complete software security. It plays a significant role in an array of security practices and interacts with other security requirements engineering techniques and secure coding practices. The threat models constructed and addressed by SDA need to undergo security verification and validation to ensure the implementation adheres to the model, and all these activities require management and continuous improvement. For genuinely comprehensive defense in depth, many more practices need to be implemented and integrated to produce an agile security development lifecycle.

With this integration in mind, a natural expansion for the work presented here is the development of a supporting tool which can automate parts of the SDA Training Framework and SATMUS process. Such a tool must offer forms and graphical editors for assisting development teams in performing SDA. For full integration and traceability, the tool needs to integrate with commonly used software development management tools, such as requirements and user story repositories, code review assistants, and bug tracking tools.

Traceability between different practices of secure software engineering remains an open issue, and this presents several avenues for further research. On the one hand, research and development can be aimed at defining methods and tools that examine the threat model and ensure that the code implements the mitigations defined by it. On the other hand, mapping the security requirements from standards and regulations to threat models can aid with formal security assurance. The issue here arises from the distributed nature of threat models produced through SATMUS. A structure that integrates threat models produced by the development teams into a product-level threat model is required to demonstrate security assurance formally.

Another related open issue is assessing the quality of threat models, where contemporary methods do not provide a comprehensive way to determine if all the critical threats were identified, let alone decomposed sufficiently. Methods for risk analysis that consider the completeness of the model, along with the likelihood and impact of threats can be examined to address these issues.

Security design analysis is a cost-efficient method for discovering vulnerabilities and design flaws early in the software's development and before they are introduced to the code. It is one of the core pillars of secure software engineering, which is why organizations must integrate it correctly into their workflow. This integration is not easy and requires dedicated security champions to build the security awareness and starting expertise required to perform it effectively. Unlike secure coding, that can be significantly aided by mature static code analysis tools, SDA is a more sophisticated practice that lacks adequate tool support. This thesis aids practitioners and security champions in introducing SDA into their organization and offers researchers a baseline on which they can build tools and methods for enhancing secure software engineering practices.

Literature

Abomhara, M., Gerdes, M. and Køien, G.M., 2015. A stride-based threat model for telehealth systems. Norsk informasjonssikkerhetskonferanse (NISK), 8(1), pp.82-96.

Albaum, G., 1997. The Likert scale revisited. Market Research Society. Journal., 39(2), pp.1-21.

Alexander, I., 2003. Misuse cases: Use cases with hostile intent. IEEE software, 20(1), pp.58-66.

Andersen, E., and Schiano, B., 2014. Teaching with Cases: A Practical Guide. Harvard Business Review Press.

Appari, A. and Johnson, M.E., 2010. Information security and privacy in healthcare: current state of research. International journal of Internet and enterprise management, 6(4), pp.279-314.

Assante, M.J. and Tobey, D.H., 2011. Enhancing the cybersecurity workforce. IT professional, 13(1), pp.12-15.

Ayalew, T., Kidane, T. and Carlsson, B., 2013, October. Identification and evaluation of security activities in agile projects. In Nordic Conference on Secure IT Systems (pp. 139-153). Springer, Berlin, Heidelberg.

Azham, Z., Ghani, I. and Ithnin, N., 2011, December. Security backlog in Scrum security practices. In Software Engineering (MySEC), 2011 5th Malaysian Conference in (pp. 414-417). IEEE.

Baca, D. and Carlsson, B., 2011, May. Agile development with security engineering activities. In Proceedings of the 2011 International Conference on Software and Systems Process (pp. 149-158). ACM.

Baca, D., Boldt, M., Carlsson, B. and Jacobsson, A., 2015, August. A novel security-enhanced agile software development process applied in an industrial setting. In Availability, Reliability and Security (ARES), 2015 10th International Conference on (pp. 11-19). IEEE.

Barnum, S., 2008. Common attack pattern enumeration and classification (capec) schema description. Cigital Inc.

Bartsch, S., 2011, August. Practitioners' perspectives on security in agile development. In Availability, Reliability and Security (ARES), 2011 Sixth International Conference on (pp. 479-484). IEEE.

Beznosov, K. and Kruchten, P., 2004, September. Towards agile security assurance. In Proceedings of the 2004 workshop on New security paradigms (pp. 47-54). ACM.

Carranza, A. and DeCusatis, C., 2015. Hybrid implementation of flipped classroom approach to cybersecurity education. NATIONAL CYBERSECURITY INSTITUTE JOURNAL, p.45.

Case, D.U., 2016. Analysis of the cyber attack on the Ukrainian power grid. Electricity Information Sharing and Analysis Center (E-ISAC).

Cockburn, A., 2002. Agile software development (Vol. 177). Boston: Addison-Wesley.

Cohn, M., 2004. User stories applied: For agile software development. Addison-Wesley Professional.

CollabNet VersionOne, 2019. 13th annual state of agile survey. In Technical Report.

Cruzes, D.S., Jaatun, M.G., Bernsmed, K. and Tøndel, I.A., 2018, November. Challenges and experiences with applying Microsoft threat modeling in agile development projects. In 2018 25th Australasian Software Engineering Conference (ASWEC) (pp. 111-120). IEEE.

Davison, R.M., Martinsons, M.G., and Kock, N. (2004) Principles of canonical action research, Information Systems Journal, 14(1), 65–86.

De Win, B., Scandariato, R., Buyens, K., Grégoire, J. and Joosen, W., 2009. On the secure software development process: CLASP, SDL and Touchpoints compared. Information and software technology, 51(7), pp.1152-1171.

Denning, T., Lerner, A., Shostack, A. and Kohno, T., 2013, November. Control-Alt-Hack: the design and evaluation of a card game for computer security awareness and education. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 915-928). ACM.

Dev, P.A. and Jevitha, K.P., 2017. STRIDE based analysis of the chrome browser extensions API. In Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications (pp. 169-178). Springer, Singapore.

Devanbu, P.T. and Stubblebine, S., 2000, May. Software engineering for security: a roadmap. In Proceedings of the Conference on the Future of Software Engineering (pp. 227-239). ACM.

Dhillon, D., 2011. Developer-driven threat modeling: Lessons learned in the trenches. IEEE Security & Privacy, 9(4), pp.41-47.

Disterer, G., 2013. ISO/IEC 27000, 27001 and 27002 for information security management. Journal of Information Security, 4(02), p.92.

Dios, A.Q., Encinas, L.H. and Queiruga, D., 2008. Cryptography adapted to the new european area of higher education. In Computational Science–ICCS 2008 (pp. 706-714). Springer Berlin Heidelberg.

Dunne, D., and Brooks, K. (2004) Teaching with cases. Halifax, NS: Society for Teaching and Learning in Higher Education.

ENISA, European Union Agency for Network and Information Security, 2019. ENISA Threat Landscape Report 2018. www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018, retrieved: 12.8.2019.

Esposito, D., Rennhard, M., Ruf, L. and Wagner, A., 2018. Exploiting the potential of web application vulnerability scanning. In ICIMP 2018, Spain, July 22-26, 2018 (pp. 22-29). IARIA.

Force, J.T. and Initiative, T., 2013. Security and privacy controls for federal information systems and organizations. NIST Special Publication, 800(53), pp.8-13.

Galvez, R. and Gurses, S., 2018, April. The Odyssey: modeling privacy threats in a brave new world. In 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) (pp. 87-94). IEEE.

Gandhi, R., Sharma, A., Mahoney, W., Sousan, W., Zhu, Q. and Laplante, P., 2011. Dimensions of cyber-attacks: Cultural, social, economic, and political. IEEE Technology and Society Magazine, 30(1), pp.28-38.

Gantenbein, H., 2016., STRIDE, CIA and the Modern Adversary, blogs.msdn.microsoft.com/heinrichg/2016/06/07/stride-cia-and-the-modern-adversary/, retrieved: 19.6.2019.

GDPR, General Data Protection Regulation, 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46. Official Journal of the European Union (OJ), 59(1-88), p.294.

Geer, D., 2010. Are companies actually using secure development life cycles?. Computer, 43(6), pp.12-16.

Georgescu, M., Hazeyama, H., Okuda, T., Kadobayashi, Y. and Yamaguchi, S., 2016, February. The STRIDE Towards IPv6: A Comprehensive Threat Model for IPv6 Transition Technologies. In ICISSP (pp. 243-254).

Ghani, I., Azham, Z. and Jeong, S.R., 2014. Integrating Software Security into Agile-Scrum Method. KSII Transactions on Internet & Information Systems, 8(2).

Goodenough, J., Lipson, H. and Weinstock, C., 2007. Arguing security-creating security assurance cases. rapport en ligne (initiative build security-in du US CERT), Université Carnegie Mellon.

Goodwin, M., 2018. OWASP Threat Dragon Project, github.com/mike-goodwin/owasp-threat-dragon, retrieved: 20.6.2019.

Hamine, S., Gerth-Guyette, E., Faulx, D., Green, B.B. and Ginsburg, A.S., 2015. Impact of mHealth chronic disease management on treatment adherence and patient outcomes: a systematic review. Journal of medical Internet research, 17(2), p.e52.

Hernan, S., Lambert, S., Ostwald, T. and Shostack, A., 2006. Threat modeling-uncover security design flaws using the stride approach. MSDN Magazine-Louisville, pp.68-75.

Howard, M. and Lipner, S., 2006. The security development lifecycle (Vol. 8). Redmond: Microsoft Press.

Hussain, S., Kamal, A., Ahmad, S., Rasool, G. and Iqbal, S., 2014. Threat modelling methodologies: a survey. Sci. Int.(Lahore), 26(4), pp.1607-1609.

IEC, International Electrotechnical Commission, 2018. 62443-4-1: Security for industrial automation and control systems, part 4-1: Product security development life-cycle requirements. USA.

IEC, International Electrotechnical Commission, 2018. 62443-4-2: Security for industrial automation and control systems, part 4-2: Technical security requirements for IACS components. USA.

Tøndel, I.A., Jaatun, M.G., Cruzes, D. and Oyetoyan, T.D., 2018. Understanding challenges to adoption of the Protection Poker software security game. In Computer Security (pp. 153-172). Springer, Cham.

Jelacic, B., Rosic, D., Lendak, I., Stanojevic, M. and Stoja, S., 2017. STRIDE to a Secure Smart Grid in a Hybrid Cloud. In Computer Security (pp. 77-90). Springer, Cham.

Jürjens, J., 2002, September. UMLsec: Extending UML for secure systems development. In International Conference on The Unified Modeling Language (pp. 412-425). Springer, Berlin, Heidelberg.

Kapitsaki, G.M. and Christou, M., 2014, April. Learning from the Current Status of Agile Adoption. In International Conference on Evaluation of Novel Approaches to Software Engineering (pp. 18-32). Springer, Cham.

Kassicieh, S., Lipinski, V. and Seazzu, A.F., 2015, August. Human centric cyber security: What are the new trends in data protection?. In Management of Engineering and Technology (PICMET), 2015 Portland International Conference on (pp. 1321-1338). IEEE.

Keblawi, F. and Sullivan, D., 2006. Applying the common criteria in systems engineering. IEEE security & privacy, 4(2), pp.50-55.

Khan, R., McLaughlin, K., Laverty, D. and Sezer, S., 2017, September. STRIDE-based threat modeling for cyber-physical systems. In Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), 2017 IEEE PES (pp. 1-6). IEEE.

Kimminich, B., OWASP Juice Shop Project, www.owasp.org/index.php/OWASP_Juice_Shop_Project, retrieved: 19.1.2019.

Knaster, R. and Leffingwell, D., 2018. SAFe 4.5 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering. Addison-Wesley Professional.

Kobara, K., 2016. Cyber physical security for industrial control systems and iot. IEICE TRANSACTIONS on Information and Systems, 99(4), pp.787-795.

Kohno, T. and Johnson, B.D., 2011, March. Science fiction prototyping and security education: cultivating contextual and societal thinking in computer security education and beyond. In Proceedings of the 42nd ACM technical symposium on Computer science education (pp. 9-14). ACM.

Krutz, D.E., Meneely, A. and Malachowsky, S.A., 2015, October. An insider threat activity in a software security course. In 2015 IEEE Frontiers in Education Conference (FIE) (pp. 1-6). IEEE.

Kshetri, N., 2009. Positive externality, increasing returns, and the rise in cybercrimes. Communications of the ACM, 52(12), pp.141-144.

Lamsweerde, A.V., 2004, May. Elaborating security requirements by construction of intentional anti-models. In Proceedings of the 26th International Conference on Software Engineering (pp. 148-157). IEEE Computer Society.

Langner, R., 2011. Stuxnet: Dissecting a cyberwarfare weapon. IEEE Security & Privacy, 9(3), pp.49-51.

Leffingwell, D., 2010. Agile software requirements: lean requirements practices for teams, programs, and the enterprise. Addison-Wesley Professional.

Lodderstedt, T., Basin, D. and Doser, J., 2002, September. SecureUML: A UML-based modeling language for model-driven security. In International Conference on the Unified Modeling Language (pp. 426-441). Springer, Berlin, Heidelberg.

Lovejoy, M.R. and Wickert, M.A., 2015, August. Using the IPython notebook as the computing platform for signals and systems courses. In Signal Processing and Signal Processing Education Workshop (SP/SPE), 2015 IEEE (pp. 289-294). IEEE.

Luburić, N., Stojkov, M., Savić, G., Sladić, G. and Milosavljević, B., 2016, August. Crypto-Tutor: An educational tool for learning modern cryptography. In 2016 IEEE 14th International Symposium on Intelligent Systems and Informatics (SISY) (pp. 205-210). IEEE.

Luburić, N., Sladić, G. and Milosavljević, B., 2018, October. Applicability Issues in Security Requirements Engineering for Agile Development. In Proceedings/8 th International conference on applied internet and information technologies (Vol. 8, No. 1, pp. II-VII). "St Kliment Ohridski" University-Bitola, Faculty of Information and Communication Technologies-Bitola, Republic of Macedonia.

Luburić, N., Sladić, G., Milosavljević, B., and Kaplar, A., 2018. Demonstrating Enterprise System Security Using an Asset-Centric Security Assurance Framework. In: Trajanović, M., Zdravković, M., Konjović, Z. (Eds.) ICIST 2018 Proceedings Vol.1, pp.16-20, 2018.

Luburić, N., Sladić, G., Slivka, J., and Milosavljević, B., 2019. A Framework for Teaching Security Design Analysis Using Case Studies and the Hybrid Flipped Classroom. ACM Transactions on Computing Education (TOCE), 19(3), p.21.

Luburić, N., Sladić, G., and Milosavljević, B., 2019. Utilizing a Vulnerable Software Package to Teach Software Security Design Analysis, Proceedings of the 42'th International ICT Convention on Information and Communication Technology, MIPRO, Opatia, Croatia, 21. - 25. may, 2019.

Luburić, N., Sladić, G., and Milosavljević, B., 2019. Examining Repudiation Threats Using a Framework for Teaching Security Design Analysis. 9th International Conference on Information Society and Technology ICIST 2019, Society for Information Systems and Computer Networks, Kopaonik, Serbia.

Martin, R.A., 2007. Common weakness enumeration. Mitre Corporation.

Mayer, R.E. and Moreno, R., 1998. A cognitive theory of multimedia learning: Implications for design principles. Journal of Educational Psychology, 91(2), pp.358-368.

McGraw, G., Migues, S. and West, J., 2018. Building Security In Maturity Model (BSIMM 8).

Mellado, D., Blanco, C., Sánchez, L.E. and Fernández-Medina, E., 2010. A systematic review of security requirements engineering. Computer Standards & Interfaces, 32(4), pp.153-165.

Meneely, A. and Lucidi, S., 2013, May. Vulnerability of the day: Concrete demonstrations for software engineering undergraduates. In Proceedings of the 2013 international conference on software engineering (pp. 1154-1157). IEEE Press.

Merrill, M.D., 2002. First principles of instruction. Educational technology research and development, 50(3), pp.43-59.

Microsoft, 2018. Agile Development Using Microsoft Security Development Lifecycle, www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx, retrieved: 7.12.2018.

Microsoft, 2019. Microsoft Threat Modeling Tool, docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool, retrieved: 20.6.2019.

Mills, D.L., 2016. Computer network time synchronization: the network time protocol on earth and in space. CRC Press.

Mohamed, S.F.P., Baharom, F., Deraman, A., Yahya, J. and Mohd, H., 2016. An Exploratory Study on Secure Software Practices Among Software Practitioners in Malaysia. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 8(8), pp.39-45.

Morrison, P., Smith, B.H. and Williams, L., 2017, May. Measuring security practice use: a case study at IBM. In Proceedings of the 5th International Workshop on Conducting Empirical Studies in Industry (pp. 16-22). IEEE Press.

Mougouei, D., Sani, N.F.M. and Almasi, M.M., 2013. S-scrum: a secure methodology for agile development of web services. World of Computer Science and Information Technology Journal, 3(1), pp.15-19.

Myagmar, S., Lee, A.J. and Yurcik, W., 2005, August. Threat modeling as a basis for security requirements. In Symposium on requirements engineering for information security (SREIS) (Vol. 2005, pp. 1-8).

Myrbakken, H. and Colomo-Palacios, R., 2017, October. DevSecOps: A Multivocal Literature Review. In International Conference on Software Process Improvement and Capability Determination (pp. 17-29). Springer, Cham.

NERC, North American Electric Reliability Corporation, 2019. Critical Infrastructure Protection Standards, www.nerc.com/pa/Stand/pages/cipstandards.aspx, retrieved: 21.6.2019.

NSF, National Science Foundation. 2008. Developing case studies for information security education, www.nsf.gov/awardsearch/showAward?AWD_ID=0737304, retrieved: 14.1.2019.

Othmane, L.B., Angin, P., Weffers, H. and Bhargava, B., 2014. Extending the agile development process to develop acceptably secure software. IEEE Transactions on dependable and secure computing, 11(6), pp.497-509.

Othmane, L.B., Angin, P. and Bhargava, B., 2014, September. Using assurance cases to develop iteratively security features using scrum. In Availability, Reliability and Security (ARES), 2014 Ninth International Conference on (pp. 490-497). IEEE.

Othmane, L.B. and Ali, A., 2016, August. Towards effective security assurance for incremental software development the case of zen cart application. In 2016 11th International Conference on Availability, Reliability and Security (ARES) (pp. 564-571). IEEE.

O'Brien, J.A. and Marakas, G.M., 2006. Management information systems (Vol. 6). McGraw-Hill Irwin.

Oyetoyan, T.D., Cruzes, D.S. and Jaatun, M.G., 2016, August. An empirical study on the relationship between software security skills, usage and training needs in agile settings. In Availability, Reliability and Security (ARES), 2016 11th International Conference on (pp. 548-555). IEEE.

OWASP, Open Web Application Security Project, 2019. Software Assurance Maturity Model (SAMM) Project, www.owasp.org/index.php/OWASP_SAMM_Project, retrieved: 8.8.2019.

Parwani, T., Kholoussi, R. and Karras, P., 2013, May. How to Hack into Facebook without being a Hacker. In Proceedings of the 22nd International Conference on World Wide Web (pp. 751-754). ACM.

PCI, Payment Card Industry, 2018. Data Security Standard, www.pcisecuritystandards.org/document_library, retrieved: 21.6.2019.

Peeters, J., 2005, August. Agile security requirements engineering. In Symposium on Requirements Engineering for Information Security.

Pohl, C. and Hof, H.J., 2015. Secure scrum: Development of secure software with scrum. arXiv preprint arXiv:1507.02992.

Pohl, C., Schlierkamp, K. and Hof, H.J., 2015. BREW: A Breakable Web Application for IT-Security Classroom Use. arXiv preprint arXiv:1506.03325.

Poller, A., Kocksch, L., Türpe, S., Epp, F.A. and Kinder-Kurlanda, K., 2017, February. Can Security Become a Routine?: A Study of Organizational Change in an Agile Software Development Group. In CSCW (pp. 2489-2503).

Positive Technologies, 2019. Web Application Attack Trends 2018. www.ptsecurity.com/ww-en/analytics/web-application-attacks-2019/, retrieved: 12.8.2019.

Power, K., 2014, May. Definition of ready: An experience report from teams at cisco. In International Conference on Agile Software Development (pp. 312-319). Springer, Cham.

Prause, C.R. and Durdik, Z., 2012, June. Architectural design and documentation: Waste in agile development?. In 2012 International Conference on Software and System Process (ICSSP) (pp. 130-134). IEEE.

Punter, T., Ciolkowski, M., Freimut, B. and John, I., 2003, September. Conducting on-line surveys in software engineering. In 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings. (pp. 80-88). IEEE.

Pupo, A.L.S., Nicolay, J. and Boix, E.G., 2018, September. GUARDIA: specification and enforcement of javascript security policies without VM modifications. In Proceedings of the 15th International Conference on Managed Languages & Runtimes (p. 17). ACM.

Ramesh, B., Cao, L. and Baskerville, R., 2010. Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal, 20(5), pp.449-480.

Ramesh, M.R. and Reddy, C.S., 2016. A survey on security requirement elicitation methods: classification, merits and demerits. Int. J. Appl. Eng. Res, 11(1), pp.64-70.

Ransome, J., Misra, A., 2013., Core Software Security, CRC Press.

Rawsthorne, D. and Trainer, C.S., 2010. Definition of Done. CollabNet, www.open.collab.net/media/pdfs/SBU_DRDefinitionOfDone.pdf, retrieved:1.7.2019.

Rawsthorne, D. and Shimp, D., 2011. Exploring Scrum: The Fundamentals. CreateSpace.

Rindell, K., Hyrynsalmi, S. and Leppänen, V., 2015. Securing Scrum for VAHTI. In SPLST (pp. 236-250).

Ross, R.S., 2011. Managing Information Security Risk: Organization, Mission, and Information System View| NIST (No. Special Publication (NIST SP)-800-39).

Ross, R., McEvilley, M. and Oren, J., 2016. Nist special publication 800-160: Systems security engineering considerations for a multidisciplinary approach in the engineering of trustworthy secure systems. Gaithersburg: National Institute of Standards and Technology.

Saitta, P., Larcom, B. and Eddington, M., 2005. Trike v. 1 methodology document dymaxion.org/trike/Trike_v1_Methodology_Documentdraft.pdf, retrieved: 9.8.2019.

Salini, P. and Kanmani, S., 2012. Survey and analysis on security requirements engineering. Computers & Electrical Engineering, 38(6), pp.1785-1797.

Saltzer, J.H. and Schroeder, M.D., 1975. The protection of information in computer systems. Proceedings of the IEEE, 63(9), pp.1278-1308.

Scandariato, R., Wuyts, K. and Joosen, W., 2015. A descriptive study of Microsoft's threat modeling technique. Requirements Engineering, 20(2), pp.163-180.

Schneier, B., 1999. Attack trees. Dr. Dobb's journal, 24(12), pp.21-29.

Schoenfield, B.S., 2015. Securing systems: Applied security architecture and threat models. CRC Press.

Sedgewick, A., 2014. Framework for improving critical infrastructure cybersecurity, version 1.0 (No. NIST-Cybersecurity Framework).

Shostack, A., 2008, September. Experiences threat modeling at microsoft. In Modeling Security Workshop. Dept. of Computing, Lancaster University, UK.

Shostack, A., 2014. Elevation of privilege: Drawing developers into threat modeling. In 2014 {USENIX} Summit on Gaming, Games, and Gamification in Security Education (3GSE 14).

Shostack, A., 2014. Threat modeling: Designing for security. John Wiley & Sons.

Shostack, A., 2017., Answer to "Granularity for data assets when determining risk during software development", security.stackexchange.com/questions/176099/granularity-for-data-assets-when-determining-risk-during-software-development, retrieved: 8.6.2018.

Security Innovation Europe, 2016. The Business Case for Security in the Software Development Lifecycle, cdn2.hubspot.net/hub/355303/file-559719186-pdf/whitepapers/business-case-appsec.pdf, retrieved: 9.8.2019.

Siles, R., Bennetts, S., OWASP Vulnerable Web Application Directory Project, www.owasp.org/index.php/OWASP_Vulnerable_Web_Applications_Directory_Project, retrieved: 24.1.2019.

Sindre, G. and Opdahl, A.L., 2005. Eliciting security requirements with misuse cases. Requirements engineering, 10(1), pp.34-44.

Singhal, A., 2011. Development of agile security framework using a hybrid technique for requirements elicitation. In Advances in Computing, Communication and Control (pp. 178-188). Springer, Berlin, Heidelberg.

Sion, L., Yskout, K., Van Landuyt, D. and Joosen, W., 2018, April. Solution-aware data flow diagrams for security threat modeling. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing (pp. 1425-1432). ACM.

Spanakis, E.G., Santana, S., Tsiknakis, M., Marias, K., Sakkalis, V., Teixeira, A., Janssen, J.H., de Jong, H. and Tziraki, C., 2016. Technology-based innovations to foster personalized healthy lifestyles and well-being: a targeted review. Journal of medical Internet research, 18(6), p.e128.

Stock, A.V.D., Glas, B., Smithline, N. and Gigler, T., 2017. OWASP Top 10 2017. The Ten Most Critical Web Application Security Risks.

Sutherland, J. and Schwaber, K., 2011. The scrum guide: the definitive guide to scrum.

Sutherland, J. and Schwaber, K., 2012. Software in 30 days - How Agile managers beat the odds, delight their customers, and leave competitors in the dust. Wiley, Hoboken (NJ).

Synopsys, 2017. Overcoming the 6 Most Common Threat Modeling Misconceptions. www.synopsys.com/software-integrity/resources/ebooks/security-threat-modeling-misconceptions.html, retrieved: 9.8.2019.

Tarandach, I., 2019. Threat Model Every Story: Practical Continuous Threat Modeling Work for Your Team. OWASP AppSec California, 2019. www.youtube.com/watch?v=W83hwtcEOjk, retrieved: 8.3.2019.

Taylor, S.J. and Bogdan, R., 1984. Introduction to qualitative research methods: The search for meaning.

Tondel, I.A., Jaatun, M.G. and Meland, P.H., 2008. Security requirements for the rest of us: A survey. IEEE software, 25(1).

Tuma, K., Scandariato, R., Widman, M. and Sandberg, C., 2017. Towards security threats that matter. In Computer Security (pp. 47-62). Springer, Cham.

Tuma, K., Calikli, G. and Scandariato, R., 2018. Threat Analysis of Software Systems: A Systematic Literature Review. Journal of Systems and Software.

Türpe, S. and Poller, A., 2017. Managing Security Work in Scrum: Tensions and Challenges. In SecSE@ ESORICS (pp. 34-49).

Türpe, S., 2017, September. The trouble with security requirements. In Requirements Engineering Conference (RE), 2017 IEEE 25th International (pp. 122-133). IEEE.

UcedaVelez, T. and Morana, M.M., 2015. Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis. John Wiley & Sons.

United States, 2004. The Health Insurance Portability and Accountability Act (HIPAA). [Washington, D.C.], U.S. Dept. of Labor, Employee Benefits Security Administration. purl.fdlp.gov/GPO/gpo10291, retrieved: 16.6.2019.

Uzunov, A.V., Fernandez, E.B. and Falkner, K., 2012. Securing distributed systems using patterns: A survey. Computers & Security, 31(5), pp.681-703.

Varma, V. and Garg, K., 2005, September. Case studies: the potential teaching instruments for software engineering education. In Fifth International Conference on Quality Software (QSIC'05) (pp. 279-284). IEEE.

Villamizar, H., Kalinowski, M., Viana, M. and Fernández, D.M., 2018, August. A Systematic Mapping Study on Security in Agile Requirements Engineering. In 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 454-461). IEEE.

Vivas, J.L., Agudo, I. and López, J., 2011. A methodology for security assurance-driven system development. Requirements Engineering, 16(1), pp.55-73.

Walden, J., 2008, October. Integrating web application security into the IT curriculum. In Proceedings of the 9th ACM SIGITE conference on Information technology education (pp. 187-192). ACM.

Weir, C., Rashid, A. and Noble, J., 2017. Developer Essentials: Top Five Interventions to Support Secure Software Development.

Whyte, B. and Harrison, J., 2010. State of practice in secure software: experts' views on best ways ahead. Software Engineering for Secure Systems: Industrial and Research Perspectives: Industrial and Research Perspectives, p.1.

Williams, L., Meneely, A. and Shipley, G., 2010. Protection poker: The new software security" game". IEEE Security & Privacy, 8(3), pp.14-20.

Wuyts, K. and Joosen, W., 2015. LINDDUN privacy threat modeling: a tutorial. CW Reports.

Wysocki, R.K., 2011. Effective project management: traditional, agile, extreme. John Wiley & Sons.

Xin, T. and Xiaofang, B., 2014. Online Banking Security Analysis based on STRIDE Threat Model. International Journal of Security and Its Applications, 8(2), pp.271-282.

Yoshioka, N., Washizaki, H. and Maruyama, K., 2008. A survey on security patterns. Progress in informatics, 5(5), pp.35-47.

Yuan, X., Yang, L., Jones, B., Yu, H. and Chu, B.T., 2016. Secure software engineering education: Knowledge area, curriculum and resources. Journal of Cybersecurity Education, Research and Practice, 2016(1), p.3.

Biography

My work is the synthesis of a three-pronged background, which includes:

1) The experience I've acquired teaching a university course on secure software engineering,
2) The research I've conducted as part of my Ph.D. studies, covering the security development lifecycle,
3) The work I've done as a security advisor for a prominent software vendor.

I started my career as a teaching assistant at the Faculty of Technical Sciences in Novi Sad in 2014. From the start, I held to the principle that how something is taught is equally important as what is taught. Over the years, I have experimented with different teaching approaches, examining gamification, e-learning, case study analysis, and the hybrid flipped classroom. My primary course covers secure software engineering, where I have developed a set of mature and relevant learning objectives as a result of my experience in the industry and as a scientific researcher.

As part of my Ph.D., I have studied the different secure software engineering methodologies and practices, covering both standard-defined processes and industry-proven methods. My narrow research focus includes security requirements engineering, particularly threat modeling and security design analysis. Up to this point, I have published six papers in this field, most notably a methodology for training software engineers the practice of security design analysis.

Through my work at Schneider Electric, I have performed threat modeling and security design analysis on several modules of a complex software system for energy management and have taken part in dozens of security analysis activities, examining tools, APIs, and 3rd-party components. My primary focus is on introducing the security development lifecycle, as defined by the IEC 62443-4-1:2018, to the organization.

By combining the different skillsets developed through my background, I have focused my expertise towards performing and teaching others to conduct various software security practices, dedicated to enhancing the security posture of a software system efficiently and measurably.