ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

MEASURING AND PREDICTING SOFTWARE REQUIREMENTS VOLATILITY FOR LARGE-SCALE SAFETY-CRITICAL AVIONICS PROJECTS

M.Sc. THESIS

Anıl HOLAT

Department of Computer Engineering

Computer Engineering Programme

FEBRUARY 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

MEASURING AND PREDICTING SOFTWARE REQUIREMENTS VOLATILITY FOR LARGE-SCALE SAFETY-CRITICAL AVIONICS PROJECTS

M.Sc. THESIS

Anıl HOLAT
(504171560)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Asst. Prof. Dr. Ayşe TOSUN KÜHN

FEBRUARY 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BÜYÜK ÖLÇEKLİ EMNİYET KRİTİK HAVACILIK ELEKTRONİĞİ PROJESİNDE YAZILIM GEREKSİNİMİ DEĞİŞKENLİĞİ ÖLÇÜMÜ VE TAHMİNİ

YÜKSEK LİSANS TEZİ

Anıl HOLAT
(504171560)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Dr. Öğr. Üyesi Ayşe TOSUN KÜHN

ŞUBAT 2022

Anıl HOLAT, a M.Sc. student of ITU Graduate School student ID 504171560, successfully defended the thesis entitled "MEASURING AND PREDICTING SOFTWARE REQUIREMENTS VOLATILITY FOR LARGE-SCALE SAFETY-CRITICAL AVIONICS PROJECTS", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Asst. Prof. Dr. Ayşe Tosun Kühn** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Feza Buzluca** Istanbul Technical University	.....
	Assoc. Prof. Dr. Gülfem Işıklar Alptekin Galatasaray University	.....

Date of Submission : 12 January 2022
Date of Defense : 1 February 2022

To my family,

FOREWORD

Firstly, I would like to thank to my supervisor Asst. Prof. Dr. Ayşe Tosun for her valuable guidance, effort, patience and encouragement throughout this research.

I would also like to thank my wife Elif Alavanda Holat and my family for support and motivation they have provided in my life.

Additionally, I want to thank ASELSAN for providing industrial information to create the dataset used in this study.

February 2022 Anıl HOLAT

TABLE OF CONTENTS

Page

FOREWORD ..... **ix**
TABLE OF CONTENTS ..... **xi**
ABBREVIATIONS ..... **xiii**
LIST OF TABLES ..... **xv**
LIST OF FIGURES ..... **xvii**
SUMMARY ..... **xix**
ÖZET ..... **xxi**
1. INTRODUCTION **1**
2. LITERATURE REVIEW **5**
3. STUDY DESIGN **9**
3.1 Research Questions 9
3.2 Analyzed Project 10
3.3 Input Metrics 11
3.3.1 Textual metrics 12
3.3.2 Project specific metrics 16
3.3.3 Network metrics 17
3.4 Model Output 19
3.5 Tools 19
3.6 Machine Learning Techniques 20
3.7 Performance Evaluation 22
4. RESULTS AND DISCUSSION **25**
4.1 Research Question 1 25
4.2 Research Question 2 27
5. THREATS TO VALIDITY **33**
6. CONCLUSIONS AND RECOMMENDATIONS **35**
REFERENCES **37**
CURRICULUM VITAE **41**

ABBREVIATIONS

ARM	**:** Automated Requirements Measurement tool
CR	**:** Change Request
KNN	**:** K-Nearest Neighbor Regression
LR	**:** Linear Regression
MdMRE	**:** Median Magnitude of Relative Error
MDP	**:** Metrics Data Program
ML	**:** Machine Learning
MMRE	**:** Mean Magnitude of Relative Error
N	**:** Network Metrics
P	**:** Project Specific Metrics
QFD	**:** Quality Function Deployment
REL	**:** Software Release
REQ	**:** Requirement
RF	**:** Random Forest Regression
RQ	**:** Research Question
RQA	**:** Requirement Quality Analyzer tool
SVR	**:** Support Vector Regression
SRS	**:** Software Requirement Specification
T	**:** Textual Metrics

LIST OF TABLES

Page

Table 3.1 : Statistical information of the safety-critical avionics software project. **11**
Table 3.2 : Requirement textual metrics. **12**
Table 3.3 : Project specific metrics. **17**
Table 3.4 : Network metrics. **18**
Table 3.5 : Requirements volatility sorting assessment for Research Question 2. **24**
Table 4.1 : Performance evaluation results for Research Question 1. ..... **26**
Table 4.2 : Comparison of our performance (RQ 1) against [1]. **27**
Table 4.3 : Change request coverage statistics for project under study. **28**
Table 4.4 : Performance results of predictive models for RQ 2 with 80% CR coverage. **29**
Table 4.5 : Performance results of predictive models for RQ 2 with 90% CR coverage. **30**
Table 4.6 : Performance results of predictive models for RQ 2 with 70% CR coverage. **31**
Table 4.7 : Performance results of predictive models for RQ 2 with 60% CR coverage. **32**

LIST OF FIGURES

Page

Figure 3.1 : A network graph illustration for electronic devices [2]. **18**
Figure 3.2 : A linear regression illustration. **20**
Figure 3.3 : Random forest regression structure [3]. **21**
Figure 3.4 : K-nearest neighbour and support vector regression visualisation [4]. **21**

MEASURING AND PREDICTING SOFTWARE REQUIREMENTS VOLATILITY FOR LARGE-SCALE SAFETY-CRITICAL AVIONICS PROJECTS

SUMMARY

During the software development life cycle, software requirements are subjected to many changes despite the recent developments in software engineering. These modifications, additions, or removals are referred to as requirements volatility. Constantly changing requirements affect cost of the project, the project schedule and the quality of the product. In the worst case projects fail or partially completed due to requirements volatility. Various requirement volatility measures have been used in previous requirement volatility prediction studies and industrial volatility measurement practices. A very big safety-critical avionics software project with thousands of software requirements from ASELSAN company is employed to forecast the number of changes for each software requirement as requirements volatility in this thesis.

To explain requirements volatility, we use a complete collection of the following metrics: requirement textual metrics, project-specific characteristics, and interdependencies between software requirements. Requirement textual metrics in this thesis are chosen from two requirements quality analyzer tools that are used in the literature. Project-specific metrics are created by focusing on safety-critical avionics project features one by one and including the ones that would give information on requirements volatility. Traceability links between system and software requirements are used to create a network graph, and network centrality metrics are created for software requirements with regard to this graph. Requirement volatility prediction is done by employing several machine learning techniques which are utilized by base studies: k-nearest neighbor regression algorithm, linear regression, random forest regression and support vector regression.

Combining input metric groups with machine learning algorithms, 28 predictive models are created in this study. This research evaluates the performance of proposed models in predicting software requirement change proneness, outperforming input metric combinations, outperforming machine learning techniques, and the success of proposed models in labeling highly volatile software requirements.

The model that combines requirement textual measurements, avionics project features, and network centrality metrics with a k-nearest neighbor machine learner produces the best prediction results (MMRE=0.366). Furthermore, the best predictive model properly labels 63.2 percent of highly volatile software requirements that are subject to 80 percent of total software requirement changes. The findings of our research are positive in terms of developing automated requirement change analyzer tools to minimize requirement volatility concerns in early development phases.

BÜYÜK ÖLÇEKLİ EMNİYET KRİTİK HAVACILIK ELEKTRONİĞİ PROJESİNDE YAZILIM GEREKSİNİMİ DEĞİŞKENLİĞİ ÖLÇÜMÜ VE TAHMİNİ

ÖZET

Yakın tarihlerde yazılım mühendisliği alanında önemli ilerlemeler gerçekleşmiş olmasına rağmen, büyük yazılım projelerindeki yazılım gereksinimleri yazılım geliştirme yaşam döngüsü boyunca birçok defa değişime maruz kalmaktadır. Projeye yeni bir gereksinim eklenmesi, mevcut bir gereksinimin değiştirilmesi ya da mevcut bir gereksinimin yazılım gereksinim dökümanından silinmesi gereksinim değişkenliği olarak tanımlanır. Devamlı değişen gereksinimler, proje geliştirme süreçleri açısından birçok kritik risk faktörü oluşmasına sebep olur. Gereksinim değişkenliği sebebiyle oluşan ekstra yazılım geliştirme, test ve doğrulama faaliyetleri projelerin tahmin edilen bütçeleri aşmasına, proje takviminin gecikmesine veya proje sonunda ortaya çıkacak ürünün kalitesinin düşmesine neden olabilir. En kötü durumda ise yazılım projeleri tamamlanmadan yarım bırakılabilir.

Literatürde gereksinim değişkenliği tahmini yapan çalışmalar ve endüstride gereksinim değişkenliğini ölçmeye çalışan proje liderleri farklı gereksinim değişkenliği ölçütlerini kullanmışlardır. Literatürde gereksinim değişim sayısı, projenin gereksinim istikrar endeksi, değişen gereksinimlerin diğer gereksinimlerin güncellenmesine etkisi, gereksinim değişikliğinin proje bütçesine olan etkileri, gereksinim değişikliğinin proje takvimine etkisi ölçüt olarak gereksinim değişkenliğini ölçmek amacıyla kullanılmıştır. Proje liderleri ise bu ölçütlerin yanında kullanım durumu değişme sayısı, bir değişim isteği kapsamında güncellenmiş toplam gereksinim sayısı, gerçeklenen gereksinim sayısının toplam gereksinim sayısına oranı ve gereksinim değişikliği sebebiyle gereken ekstra proje bütçesi ölçütlerini de gereksinim değişkenliğini ölçmek amacıyla kullanmışlardır.

Bu tez çalışmasının amacı ASELSAN'a ait bir emniyet-kritik havacılık elektroniği projesinde metinsel ölçütleri, proje karakteristiklerini ve gereksinimler arası ilişkileri kullanarak her bir yazılım gereksinimi için oluşturulmuş değişim isteği sayısını tahmin etmektir. Toplamda üç yazılım sürümünden 20,000'in üzerinde yazılım gereksinimi veri kümesine dahil edilmiştir. Girdi ölçütleri ve makine öğrenmesi metotları kombine edilerek 28 öngörücü model oluşturulmuştur. Birinci araştırma sorusu bu öngörücü modellerin gereksinim değişkenliğini hangi ölçüde tahmin edebildiği, en başarılı girdi ölçütü kombinasyonun hangisi olduğu, en başarılı makine öğrenmesi metodunun hangisi olduğudur. İkinci araştırma sorusu ise öngörücü modellerin çok değişken gereksinimleri tahmin etmede başarılı olup olmadığıdır. Bu öngörücü modellerin pratikteki amacı çok değişken gereksinimleri yazılım geliştirme süreçlerinin erken aşamalarında tahmin edip gerekli önlemleri alarak proje risklerini azaltmaktır. Örneğin

yazılım gereksinim dökümanı tamamlanmadan önce bu modelleri kullanarak olası çok değişken gereksinimler bulunup, bu gereksinimlerin tecrübeli kişiler tarafından gözden geçirilmesi sağlanabilir.

Bu çalışmada gereksinim değişkenliği ölçütü her bir yazılım gereksinimi için oluşturulmuş değişim isteği sayısıdır. Yazılım gereksinim dökümanı gözden geçirilip tamamlandıktan sonra ihtiyaç olan her bir değişiklik için değişiklik isteği oluşturulur ve yazılım geliştirme ekibi bu değişiklik isteği kapsamında yazılım gereksinimlerini ekler ya da günceller. Bu sebeple her bir yazılım gereksinimi için oluşturulmuş değişim isteği sayısı aynı zamanda her bir yazılım gereksinimin yazılım gereksinim dökümanı tamamlandıktan sonra kaç defa değiştiğini ifade eder.

Öngörücü modeller için girdi olarak kullanılan metinsel ölçütler literatürde kullanılan iki farklı programdan derlenerek üretilmiştir. İlk olarak NASA'nın ARM gereksinim çözümleyici programının kullandığı bazı ölçütler bu tezde kullanılmıştır. Bu programın kullandığı ölçütler literatürde daha önceden hatalı modülleri bulmak amacıyla defalarca kullanıldığından dolayı tercih edilmiştir. İkinci olarak ise REUSE firması tarafından üretilen RQA programının kalite ölçütleri kullanılmıştır. Bu programın gereksinimlerin kalitesini ölçme konusunda güncel programlara göre daha başarılı olduğunun iddia edilmesi bu ölçütleri seçmemizde etkili olmuştur. Bir diğer kullanılan ölçüt grubu ise projeye özel ölçütlerdir. Bu tez kapsamında emniyet kritik havacılık elektroniği projelerine ait özellikleri de ölçüt olarak kullanmak istedik, bu sebeple proje özelliklerine tek tek yoğunlaşarak gereksinim değişkenliğini etkileyebileceğini düşündüklerimizi bu ölçüt grubuna ekledik. Son olarak ağ merkezilik ölçütleri kullanılmıştır. Benzer bir çalışmada bu ölçütler kullanıldığı için biz de gereksinim değişkenliği tahmini için bu ölçütleri kullanmak istedik. Bu ölçütleri kullanan çalışma gereksinim ağını dilsel benzerliklere göre oluşturuyordu, fakat biz yazılım gereksinimlerinin sistem gereksinimleriyle olan ilişkilerine göre bu ağı oluşturduk. İlk gözlemlerimizde, değişiklik istekleri notlarını inceledik ve bir değişiklik isteği kapsamında güncellenen gereksinimlerin aynı sistem gereksinimlerinden türetilmiş olmaya yatkın olduğunu fark ettik. Bu sebeple yazılım ve sistem gereksinimi ilişkilerine göre ağ merkezilik ölçütlerini oluşturduk ve bunları gereksinim değişkenliği tahmininde kullandık.

Bu tez çalışmasında, baz alınan çalışmalarda kullanılan dört farklı makine öğrenmesi kullanıldı: en yakın k-komşu regresyon algoritması, doğrusal bağıntı, rastgele orman regresyon methodu ve destek vektör regresyonu. Eğitim setini ve test setini ayırmak için 10 kat çapraz geçerleme methodu kullanıldı. Bu tez çalışmasında makine öğrenmesi yöntemlerini eğitmek amacıyla WEKA programı kullanıldı. Öngörücü model tahminleri MATLAB programı ile işlenerek araştırma soruları için başarım ölçütleri hesaplandı.

Birinci araştırma sorusu için, öngörücü modellerin gereksinim değişkenliğini hangi ölçüde tahmin edebildiğini ölçmek amacıyla çeşitli performans ölçütleri kullanıldı: bağıl ortalama hata (MMRE), bağıl ortanca hata (MdMRE), Pred(0.5) ve Pred(0.25). Bu ölçütler benzer bir çalışmada da gereksinim değişkenliği tahmin modellerinin performansını ölçmek için kullanılmıştır. İkinci araştırma sorusu öngörücü modellerin çok değişken gereksinimleri tahmin etme başarısı ile ilgilidir. Öngörücü modellerin bu açıdan değerlendirebilmesi için öncelikle çok değişken gereksinim tanımının nasıl yapılacağı ve model başarısının nasıl ölçüleceği belirlenmelidir. Bu konuda literatürde daha önceden kullanılmış bir yöntemden ilham alarak bu çalışmaya uyarladık. İkinci

araştırma sorusu için duyarlılık, doğruluk, yanlış alarm oranı performans ölçütleri kullanıldı. Duyarlılık ölçütü öngörücü modellerin çok değişken gereksinimleri bulmadaki başarısını gösterir, bu sebeple ikinci araştırma sorusu için en önemli ölçüt budur. Doğruluk ölçütü bütün gereksinimler üzerindeki tahmin başarısını gösterir. Yanlış alarm oranı ise çok değişken olmayıp çok değişken olarak işaretlenen gereksinimler yüzünden boşa harcanan emeğin ölçütüdür.

Bu tez çalışmasında farklı girdi ölçütleri ve makine öğrenmesi metotları kombinasyonuyla oluşturulan 28 farklı öngörücü model için sonuçlar sunuldu. Birinci araştırma sorusu sonuçları incelendiğinde en iyi tahmin sonuçlarının gereksinim metinsel ölçütleri, proje özel ölçütleri ve ağ merkezilik ölçütleri beraber kullanıldığında ve en yakın k-komşu regresyon algoritması ile model eğitildiğinde elde edildiği görüldü (MMRE=0.366, MdMRE=0, Pred(0.25)=0.57, Pred(0.5)=0.681). Ayrıca sonuçlar daha önceden yine gereksinim değişim sayısını tahmin eden bir çalışmayla kıyaslandı. Diğer çalışma sadece doğrusal bağıntı makine öğrenmesini kullandığı için sadece bu kulvarda kıyaslama yapıldığında bizim sonuçlarımızın diğer çalışmaya yakın sonuçlar elde ettiği görüldü. Fakat diğer makine öğrenmesi kullanan öngörücü model sonuçları da dahil edildiğinde bizim modellerimizin tahmin sonuçlarının çok daha iyi olduğu gözlemlendi. İkinci araştırma sorusu için öncelikle çok değişken gereksinimlerin nasıl işaretleneceği konusunda çalışıldı. Bu proje kapsamında yazılım gereksinim dökümanını gözden geçiren kişilerin yaklaşık yüzde 40'nın tecrübeli olduğu görüldü. Yine bu proje kapsamında tüm değişim isteklerinin yüzde 80'inin tüm gereksinimlerin yüzde 38.6'sı için oluşturulduğu görüldü. Bu sebeple tüm değişim isteklerinin yüzde 80'ine sahip olan gereksinimler çok değişken olarak tanımlandı. İkinci araştırma sorusu kapsamında da tüm girdi ölçütleri beraber kullanıldığında ve en yakın k-komşu regresyon algoritması ile eğitim seti eğitildiğinde en iyi sonuçların elde edildiği görüldü. Toplamda değişiklik isteklerinin yüzde 80'ine maruz kalmış yazılım gereksinimlerinin yüzde 63.2 duyarlılık başarısıyla doğru tahmin edildiği görüldü. Ayrıca doğruluk skoru 0.716 ve yanlış alarm oranı 0.232 olarak ölçüldü.

Sonuç olarak bu tez çalışmasında gereksinim metinsel ölçütlerini, proje karakteristiklerini ve gereksinimler arası ilişkileri kullanarak ASELSAN'a ait bir emniyet kritik havacılık elektroniği projesinde her bir yazılım gereksinimi için oluşturulmuş değişim isteği sayısı tahmini yapıldı. Üç yazılım sürümünden toplanan 22,771 yazılım gereksinimi ve 28 farklı öngörücü model kullanılarak değişim isteği sayısı tahminini yapmada en başarılı girdi ölçütü kombinasyonları ve makine öğrenmesi metotları bulundu. En başarılı sonuçlar gereksinim metinsel ölçütleri, proje özel ölçütleri ve ağ merkezilik ölçütleri beraber kullanıldığında ve en yakın k-komşu regresyon algoritması ile model eğitildiğinde elde edildi. Bu öngörücü model ile yüzde 36.6 bağıl ortalama hata (MMRE) ile herbir yazılım gereksinimi için oluşturulmuş değişim sayısının tahmin edildiği gözlemlendi. Ayrıca aynı model, toplamda değişiklik isteklerinin yüzde 80'ine maruz kalmış yazılım gereksinimlerini yüzde 63.2 başarıyla doğru tahmin edebiliyor. Bu sonuçlar ilerleyen zamanlarda gereksinim yönetim araçlarında güncellemeler yaparak gereksinim değişkenliğinin projenin erken aşamalarında tahmin edilebilmesi ve buna bağlı proje risklerinin azaltılması konusunda umut verici. Literatürde bu alanda deneysel çalışma yapan çok fazla yayın olmaması sebebiyle en başarılı öngörücü modellerin belirlenmesi için bu alanda deneysel yeni çalışmaların yapılması gerektiğini düşünmekteyiz.

1. INTRODUCTION

Over the last few decades important advancements occurred in software engineering, however most of the major software projects still encounter considerable amount of requirement changes during the software development process because of the unpredictable nature of software development [5]. Adding a new requirement, deleting an existing requirement or modifying an existing requirement are defined as requirements volatility [6]. Changing requirements persistently during software development life-cycle may lead to several project risks. Additional design, verification and review activities due to requirement changes increase the budget required to complete the project, delay the schedule and worsen the quality of the final product. Most importantly, many software projects either cannot be completed or partially completed due to highly volatile requirements [6].

Measures of requirements volatility are defined differently by previous studies and industry practitioners. Numerous metrics are utilized in previous studies to measure requirements volatility, e.g. change size [7], number of requirement changes [1], timing of requirement change occurrence [8], tendency of change based on project characteristics [9], the number of changes within a particular duration [10], and requirement stability index of overall project [11]. Regarding a survey carried out by Thakurta [12], various other requirement volatility measures are employed by project managers in industry: the total number of changes to the use cases, the total changed requirements in a change request, implemented requirements over the total number of requirements, and additional cost for changing requirements.

A literature review on studies predicting requirements volatility is reported by Alsalemi et al. [13]. As reported by, previous ten studies have used machine learners to predict volatility of requirements until 2017. Those studies employ various requirement volatility measures e.g. number of requirement changes [1], requirement stability index of overall project [11], requirement changes in next iteration [14], influence of requirement change on other requirements [15], the effect of changing requirements on

project distribution and budget of the project [16], schedule of the project [17]. Those publications also use following input metrics to predict their own interpretation of requirement volatility measure: complexity metrics [11], requirement relation metrics [15], size metrics [1] and change history metrics [14].

The purpose of thesis is to predict number of change requests for each software requirement by utilizing textual features of software requirements, characteristics of the selected project and network centrality metrics. Software requirements convey expectations for a software product, they define capabilities and constraints on the system. A software requirement is written as a combination of conditions which specifies in which circumstances this software requirement should be applied and which actions to be taken when these conditions are satisfied. Software requirements used in this thesis vary in size such that some requirements consist of single sentence whereas others consist of a dozen of sentences. An illustrative software requirement can be formed as following: "If barometric pressure is greater than X and if temperature is smaller than Y, label Z visibility shall be set to invisible". A change request for this project could be created to either add a new software requirement or update an existing software requirement from a Software Requirement Specification document. Following the review and completion of Software Requirements Specification document by development teams, every new change is processed by creating change requests. Therefore in our context, the number of change requests per software requirement are equal to the number of changes per software requirement after an SRS is issued.

There are various reasons to create change requests. Customers may want to add additional features or change existing features of a product in later phases of development which leads to system requirement and software requirement changes. If software project schedule is too tight, developers may rush to complete Software Requirement Specification document on time and this could result in immature software requirements. Software developers can make logical mistakes while creating software requirements from system requirements and that would trigger future software requirement changes. Typos and ambiguities in requirements could be exposed by software testers and this could lead to change in software requirements

as well. For those reasons some software requirements could change many times and result in project risks.

In ASELSAN company, a safety-critical avionics software project with several software releases and a great number of software requirements is selected for this research. In general avionics software projects are large-scale projects, because avionics systems are complex and they demand to implement many critical functions such as communication, navigation, flight control and data handling. Also safety critical requirements leads to a more strict design, in which case each condition must be defined by requirements in detail to prevent potential failures due to undefined cases. Thus normally avionics software projects contain many software requirements. It's worth noting that change requests for this project occurred in any software development phase after Software Requirements Specification document is completed. Therefore we measure post-SRS software requirements change proneness for the safety-critical avionics software project in this thesis.

A prior empirical study was carried out to predict the number of changes per requirement by employing a limited number of size metrics on industrial software projects that have fewer than 50 requirements [1]. Our research complements the previous study by utilizing a bigger dataset created from more than 20,000 software requirements and a larger input metric set in consideration of textual and interrelation features of software requirements in addition to safety-critical avionics project characteristics.

2. LITERATURE REVIEW

In this literature review we report related studies that focus on predicting requirements volatility. The focus of this chapter is input metric groups that are used by previous studies to build prediction models. Rationales for selection or exclusion of previously employed input metric groups will be given. Details of some previous researches [1, 11, 14–16, 18] listed in a literature review carried out by Alsalemi et al. [13] will be presented. Approaches of recently published other related studies [19–21] are discussed as well.

Anang et al. [21] proposed a method to calculate customer requirement's volatility using Quality Function Deployment(QFD) method. QFD method is described as a method that helps to derive engineering design elements from customer requirements. Using Quality Function Deployment, authors derived software functions from customer requirements and architectural design elements from software functions. Weights are assigned while deriving software function and architectural design elements considering strength of relations. Then different volatility values are assigned to each architectural design element for layers they belong to. Finally using weights and static volatility values requirement volatility is calculated for each customer requirement. To sum up authors used static volatility analysis based on software architecture. Software architecture could be used to predict requirements volatility, however in our thesis we aim to predict software requirements volatility in early phases even before software is completed.

Nakatani et al. [18] present a way for predicting requirements volatility based on social relations between cooperative organizations, executives, natural environment and the competitors. By using those relations, authors aim to predict requirement changes in next iteration. Authors employed Strategic Dependency model and class diagrams in UML for the sake of volatility analysis. Those social relation measurements are simple to apply to client requirements, however applying them to software requirements require some effort.

Christopher et al. [11] employ complexity metrics and fuzzy approach to predict volatility. Complexity metrics of functional requirements, input-output, non-functional requirements, interfaces and files are utilized to find out project overall requirement stability index. As an initial step complexity weights are assigned to requirement complexity attributes. Following this step software complexity point is calculated. Finally, stability index of whole project is predicted with respect to the change of complexity point of developing project. Requirements stability index by the definition is inversely proportional with requirements volatility so both terms have close relation. However authors predict requirements stability index for whole project whereas we aim to forecast requirements volatility per software requirement.

Shi et al. [14] show a prediction model that uses history metrics of requirement changes to forecast future requirement changes. Authors use six requirement change history metrics that give information on the topic's volatility, change frequency and time between requirement changes. History metrics could be employed to guess requirements which will be altered in the following iteration, but they aren't very useful in predicting the volatility of requirements in early development phases for new projects.

Pedrycz et al. [20] use the following requirement change logs in their work to predict volatility: number of updates, created version of the requirement, last developer, and lifetime of the requirement. Authors use logistic regression to predict requirements which are deleted throughout software development life cycle. Requirement volatility is defined as addition, modification and deletion of requirements, but this study only focused on predicting deleted ones. Research is conducted in Italian Aeronautical Company using 6,608 requirements and software logs. Similar to history metrics change logs are also created in the later phases of software development. Therefore this metric group is not effective in predicting requirements volatility in early development phases of software projects.

Morkos et al. [16] used distinct approaches to create relationships between requirements and used those relations to predict requirement change propagation. In this study first method is manually creating relations between requirements, second method is syntactic and lingual parsing and grammatically relating requirements, and the last one is using artificial neural networks to create relations. We also benefit from

requirement relations to explain requirements volatility in our study. However, we did not use manual analysis, linguistic relations or artificial neural networks to create software requirement relations.

Hein et al. [19] and Goknil et al. [15] utilize interconnection of requirements for predicting requirements volatility. Hein et al. [19] utilize network metrics derived from linguistic data, whereas Goknil et al. [15] use formal requirement relations as metric group to predict volatility for requirements. In our study we merged both input metric group and used network metrics derived from traceability links between software and system requirements rather than deriving from linguistic relationships. Hein et al. [19] employed 40 different network metrics in their study whereas we only used following network centrality metrics degree, eigenvector, closeness and betweenness.

Respecting the literature review findings, only research carried out by Loconsole et al. [1] report an empirical work to guess total changes for each requirement, thus this research is the most related study regarding volatility definition. Authors have used following requirement size measures: the number of revisions in each file, the amount of people who interact with use cases, the number of words used in a requirement file, and the number of lines in a requirement file. In this thesis size metrics are used in requirement textual metrics but also metric set is enriched with requirement interrelations and project characteristics. Authors used only linear regression in their work to predict requirement changes, whereas we used linear regression and additional machine learners. It's worth noting that when defining requirements volatility, we don't take into consideration deleted requirements or deletion requests since in our industrial setting deletion of safety-critical avionics software requirements are uncommon. Lastly, predictive models in this study are applied on vast number of software requirements to see if results of this thesis on predicting software requirement volatility by employing various metrics sets are generalizable.

3. STUDY DESIGN

Within study design chapter, empirical volatility prediction study for the avionics software project is defined in depth. Research questions are described in Section 3.1. The employed software project for which predictive model are proposed is reported in Section 3.2. Utilized input metric groups for volatility prediction are defined in Section 3.3. Predictive model output is stated in Section 3.4. The employed tools for creating the dataset, training machine learners and measuring the performance are reported in Section 3.5. Used Machine learning methods for predictive model are expressed in Section 3.6. As last Section in 3.7, the ways of measuring performance with regard to research questions are defined in detail.

3.1 Research Questions

Main objective of this thesis is to predict volatility of requirements at an earlier stage in the software development life cycle, and respecting this goal two main research questions have been determined.

Research Question(RQ) 1: To what extent do input metrics created from requirement textual features, safety-critical avionics project characteristics and graph based centrality scores forecast the volatility of a software requirement?

Prior volatility prediction researches utilized different metric groups as input metrics. In this thesis aim is to employ wide range of input metric groups, and inspect how they affect volatility of software requirements individually. Volatility of requirements is defined as number of change (modification and addition) requests for a software requirement. In this study we create requirement textual metrics and network centrality metrics being inspired by the metric groups utilized in the literature. Furthermore, project feature metrics for this avionics software project are determined and used in this research. Each input metric set's performance, as well as the success of combination

of them, are investigated during the model evaluation. Additionally, RQ 1's detailed sub-questions are reported here.

RQ 1.1: Among the employed input metric groups which one outperform in terms of predicting the number of software requirement changes?

RQ 1.2: Among the employed machine learners which one outperform in terms of predicting the number of software requirement changes?

RQ 2: In terms of anticipating highly volatile software requirements, how prospering are the predictive models?

Throughout the software development life cycle, software requirements have undergone a lot of changes. Some requirements do not change at all, whereas others are subject to considerable amount of revisions and create significant threats to a software project. In practice, the predictive model must predict highly volatile requirements, hence those highly volatile requirements will be thoroughly reviewed by experienced reviewers. We evaluate the performance of our models for RQ 2 inspired from a technique in [22].

3.2 Analyzed Project

To conduct our research, a safety-critical avionics project from ASELSAN is selected. Safety-critical systems are ones that could result in a loss of life, major property damage, or environmental damage if they fail [23]. The project under this study will be referred as AVPRJ throughout this thesis. AVPRJ has a large number of releases from which three have been chosen. Because they are all part of the same project, the software requirements for those releases are related; nevertheless, those requirements are somewhat divergent because each software release comprises of the implementation of separate software units developed by numerous developers.

There are 22,771 software requirements in total for AVPRJ. Table 3.1 reports some descriptive statistics for AVPRJ based on release. CR is used as an acronym for change request, REL is used as an acronym for a software release, REQ is used as an acronym for a software requirement. The majority of the requirements used are from the second release, which has the highest mean change request amount for each software requirement. The third release has less software requirements, and this

release's requirements are added or modified less frequently. For this project, more than half of the requirements are changed at least once; 9,848 out of 22,771 total software requirements are not modified which complies to the results of the Standish Group's survey of over 8,000 software projects [24].

Table 3.1 : Statistical information of the safety-critical avionics software project.

REL	Number of REQs	Mean CR per REQ	Median CR per REQ
REL-1	8,640	0.75	1
REL-2	11,401	1.12	1
REL-3	2,730	0.53	0
Total	22,771	0.91	1

3.3 Input Metrics

In AVPRJ, we used various metric groups to predict the volatility of each software requirement. Three aspects are represented by the metrics: requirements textual features, project characteristics and requirement interdependencies. Textual metrics of requirements created by NASA Automated Requirements Measurement(ARM) tool have already been employed to predict fault proneness of software modules [25]. At the beginning of research, we intuitively thought the way requirements are written will have an impact on requirements volatility as well as module fault proneness. In a related study [1] some textual size metrics are already used to predict volatility of requirements, thus we determined to add a set of requirement textual metrics in our research. In a recent work [19], network metrics were used to predict requirement change volatility. This research sparked the idea to use network centrality metrics in predicting volatility of software requirements. Initial observation of numerous change request notes revealed that software requirements that are added or updated inside a change request are frequently tied to similar system requirements. As a result, we used network measurements derived from traceability data. With the intention of enriching input metric group with additional metric group we concentrated on safety-critical avionics software project characteristics under this thesis. Avionics project features are analyzed individually, and those that give information on requirement volatility are chosen as project-specific metrics. The reasons for selecting each avionics project

feature as input metric is given in subsection 3.3.2. The following subsections provide in-depth descriptions for each group.

3.3.1 Textual metrics

We were motivated by two studies while creating requirement textual metrics to predict software requirement volatility. The first study [25] proposes requirement metrics for predicting software defects in the context of NASA's Metrics Data Program (MDP). These requirement textual measures are derived by an automated tool that scan requirement documents for complex, vague, long, and ambiguous requirements. Authors used those metrics to predict fault prone modules but we adopted some of those requirement metrics to predict requirement volatility. The second study [26] compares requirement quality analyzer tools with respect to defined criteria and finds out the best performing one is Requirements Quality Analyzer tool by REUSE company. We enriched our textual metric group by including some textual metrics defined by RQA. Requirement textual metrics of those two tools are employed since textual metrics of ARM tool widely used in the literature previously and RQA tool is claimed to be more successful than other analyzer tools according to [26].

We propose 20 requirement textual metrics in Table 3.2 by combining the lists from both research and tailoring them to the avionics project requirement documents in our context. All of these metrics, such as the number of negative phrases in a requirement or the number of connectors in a requirement, take numeric values.

Table 3.2 : Requirement textual metrics.

Acronyms	The number of abbreviations utilized within a software requirement specification.
Actions	Software requirement specifications explains what should be done if particular circumstances exist. Actions define what should be done and this metric is the numeric value of total defined actions within a software requirement.
Ambiguity	The total count of the indefinite phrases within a software requirement specification. Software requirements should be specified correctly, otherwise implementation of requirements could be incorrect.
Chars between punctuation	The average character count between punctuation marks within a software requirement specification. Readability of a requirement is decreased by long sentences without punctuation marks.

Table 3.2 (continued) : Requirement textual metrics.

Conditions	Software requirement specifications explains what should be done if particular circumstances exist. Conditions define particular circumstances and this metric is the numeric value of total defined conditions within a software requirement.
Conditional	The total count of expressions that give the software developers freedom to implement actions specified by a software requirement. Software requirements should be written precisely, there should not be optional statements.
Connectors	The total count of connectors that are utilized to associate a handful of sentences or word groups.
Directives	The total count of phrases to refer a table, an example, a note, or a figure.
Flow sentences	The total count of phrases that semantically bond a sentence to another one.
Imperatives	The total count of expressions that mandate to carry out certain tasks within a software requirement specification.
Implicitness	The total count of pronouns that creates implicit meaning within a software requirement specification and confuse developers on implementing the requirement. Requirements should be defined explicitly, otherwise implementation of requirements could be incorrect.
Incompleteness	The total count of phrases that show a software requirement is not completed yet. After Software Requirement Specification document is reviewed and signed off there should not be any incomplete software requirements.
In links	The total count of incoming links for a software requirement from other project documents.
Negative Sentences	The total count of expressions that negate the meaning of software requirement specification.
Nested levels	Nested level metric value is the greatest level in hierarchical nesting structure of a software requirement.
Out links	The total count of out links from a software requirement to the other project documents.
Rationale	The total count of phrases that give rationalization within a software requirement specification.
Speculative Sentences	The total count of speculative expressions which result in feeling suspicious about needfulness of a software requirement.
Subjectivity	The total count of subjective expressions expressing individual sentiment instead of objectivity.
Text length	The total count of characters within a software requirement specification.

Numeric values of requirement textual metrics under this thesis are created by evaluating specification statements along the following dimensions:

- *Acronyms:* For the safety-critical avionics software project used in this study a document which lists the allowed acronyms exist. The list from this particular document is employed to get Acronyms metric data. Those acronyms are shorter names for project specific partitions, devices or terms. Due to confidentiality of the project this list cannot be provided in this thesis.

- *Actions:* Software requirement specifications of AVPRJ are created with respect to software requirement design document of the project. In this document how actions will be represented is defined strictly. We parsed requirement specifications in that manner and got the numeric value of this metric.

- *Ambiguity:* Count the phrases "adequate", "approximate", "approximately", "maximum", "minimum", "minimal", "appropriate", "as required", "bad", "be able to", "best practices", "better", "capability of", "capability to", "close", "quickly", "easy", "effective", "efficient", "fast", "flexible", "good", "improved", "maximize", "minimize", "medium sized", "medium-sized", "optimize", "optimal", "optimum", "nominal", "normal", "typical", "typically", "useable", "suitable", "not limited to", "provide for", "prompt", "quick", "rapid", "reliable", "routine", "safe", "slow", "sufficient", "sufficiently", "timely", "too", "user friendly", "user-friendly", "versatile", "worst", "at least", "enough", "clearly", "based on", "some", "several", "many", "a lot of", "few", "about", "very nearly", "manage", "easily", "small", "significant", "vague", "ancillary", "relevant", "common", "generic", "customary", "necessary", "both", "acceptable", "accurate", "adjustable", "affordable", "applicable", "average", "careful", "deep", "dependable", "desirable", "economical", "essential", "excessive", "high", "quality", "immediately", "improper", "instant", "insufficient", "known", "less", "low", "major", "neat", "other", "periodically", "pleasing", "possible", "practicable", "practical", "proper", "reasonable", "recognized", "reputable", "secure", "similar", "smooth", "stable", "temporary", "variable", "various", "worse".

- *Chars between punctuation:* This metric is calculated by dividing total number of characters in a software requirement specification to total number of punctuation marks.

- *Conditions:* Software requirement specifications of AVPRJ are created with respect to software requirement design document of the project. In this document how conditions will be reported is defined strictly. We parsed requirement specifications in that manner and got the numeric value of this metric.

- *Conditional:* Count the phrases "may be", "maybe", "can", "can't", "cannot", "could", "couldn't", "could not", "should", "shouldn't", "should not", "ought to", "oughtn't", "ought not", "would", "wouldn't", "would not".

- *Connectors:* Count the phrases "and", "or", "as well as", "but also", "however", "whether", "meanwhile", "whereas", "on the other hand", "otherwise".

- *Directives:* Count the words "for example", "note", "figure", "table".

- *Flow sentences:* Count the phrases "although", "as well as", "but", "else", "except", "if", "then", "unless", "when", "while".

- *Imperatives:* Count the phrases "shall", "will", "must", "shan't", "won't", "mustn't".

- *Implicitness:* Count the phrases "he", "her", "him", "his", "I", "it", "my", "our", "she", "their", "them", "they", "us", "we", "you", "your", "this", "that".

- *Incompleteness:* Count the phrases "among others", "and so on", "as a minimum", "etc", "etcetera", "further", "not defined", "not determined", "not limited to", "tbd", "tbc", "tbs", "to be determined", "e.g", "eg", "example", "such as".

- *In links:* Because the test cases of the safety-critical avionics software project are linked to software requirements, the total count of in links refers to the total count of test cases that are linked.

- *Negative Sentences:* Count the phrases "cannot", "doesn't", "never", "no", "nobody", "non", "none", "nor", "not", "nothing", "won't", "shan't", "mustn't", "couldn't", "shouldn't", "oughtn't", "can't".

- *Nested levels:* For AVPRJ software requirement specifications are written with respect to software requirement design document of the project. In this document hierarchical nesting structure of software requirements is defined, thus we parsed requirements text regarding this document and created greatest nested level inside a software requirement.

- *Out links:* The software requirements of AVPRJ are tied to the system requirements. As a result, the total count of linked system requirements by a software requirement equals the total count of out links.

- *Rationale:* Count the phrases "in order to", "so that", "thus", "allowing".

- *Speculative Sentences:* Count the phrases "commonly", "frequently", "generally", "normally", "often", "optionally", "perhaps", "probably", "rarely", "typically", "usually", "eventually", "at last", "almost", "always".

- *Subjectivity:* Count the phrases "I think", "imho", "imo", "in my humble opinion", "in my opinion".

- *Text length:* This metric is calculated by counting the number of characters within a software requirement specification.

We had to delete three metrics from our study during the preprocessing step because they provided almost no information for safety-critical avionics software project: Subjective, Rationale, Conditional. There are only conditional phrases in one software requirement, three rationale phrases in three software requirements, and no subjective expressions in any of the software requirements. As a result, we came up with 17 metrics to reflect the textual features of requirements in order to predict change proneness.

3.3.2 Project specific metrics

Project characteristics could differ with respect to scope of a software project. However, we chose metrics that aren't tied to the programming language, development environment, or domain in which the software is created. We think that project feature metrics could provide insight into an organization's development characteristics, and hence the elements driving requirement change proneness. This study's

project-specific metrics are reported in the Table 3.3. If the project follows a requirement inspection activity, ambiguities and inconsistencies in the requirements have a tendency to be uncovered by the development team. User acceptance tests cannot be used to validate derived requirements because they are not part of the customer's demands. If a requirement is safety-critical, more extensive software tests will be run, increasing the likelihood of a potential modification being discovered. Because the number of linked software units is an indicator of a software requirement's impact on the whole product, the development team will give more feedback to requirements that affect many components. Each software release has its own set of characteristics that influence requirements maturity, such as the release timeline, developer experience, and system complexity. To illustrate, if the schedule for completing the Software Requirement Specification document is too tight for a release, software requirements may be immature, and further requirements modifications may be required later. All of the project specific metrics for AVPRJ are already recorded as property of a software requirement in requirement management tool, thus values of project specific metrics are retrieved directly from the tool.

Table 3.3 : Project specific metrics.

Inspection	The value of Inspection metric is true when a requirement is reviewed by an inspection activity. This method could be used to complement functional tests.
Derived	Derived software requirements are developed from design decisions rather than being explicitly defined in system requirements [27].
Safety	Indicates that a software requirement failure could endanger a human life or damage a property.
No. of Related Components	Number of distinct software components that a software requirement is associated.
Release Number	The release number of the software requirement.

3.3.3 Network metrics

In a variety of environments, such as technical and transportation infrastructures, social phenomena, and biological systems, networked structures emerge [28]. In graph theory and network analysis those network structures are used to create network metrics and analyze topology in detail. Figure 3.1 visually illustrates a network graph for usb and

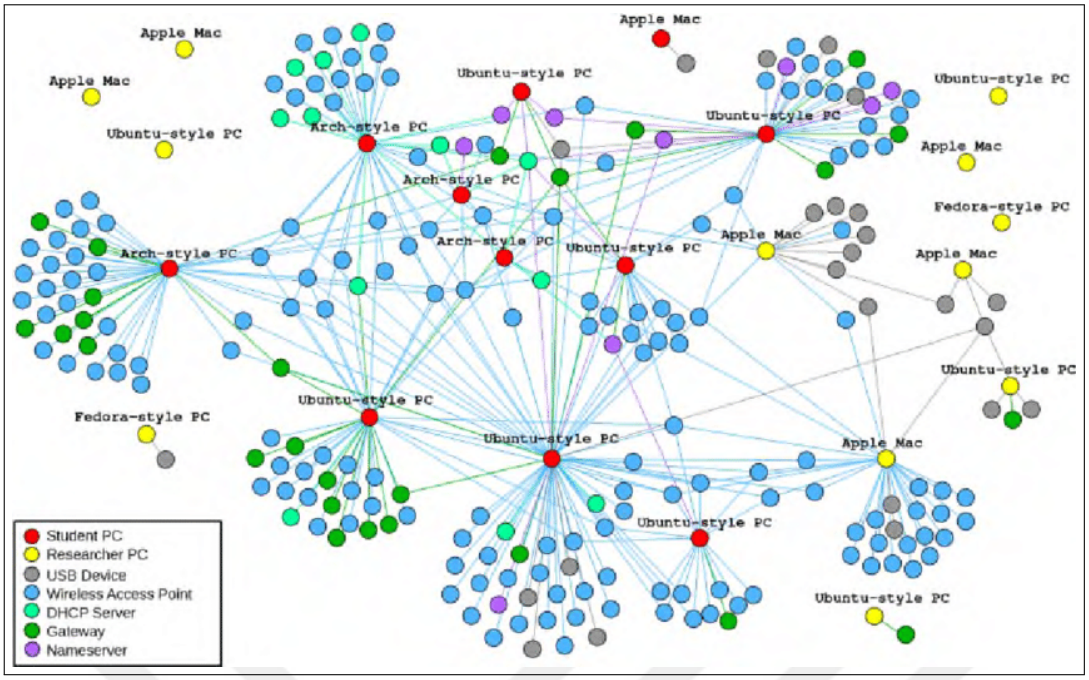

Figure 3.1 : A network graph illustration for electronic devices [2].

bluetooth devices, Wi-Fi access points, and gateways where edges are created based on connections.

Hein et al. [19] used 40 network metrics to predict the volatility of requirements change in a recent study. On the other hand, Valente et al. [29] present correlations between closeness, degree, eigenvector, closeness centrality metrics, and demonstrate that these metrics are separate but notionally related. Instead of using 40 measures, we used the metrics in [29] to predict AVPRJ requirements volatility in this study. These centrality metrics assign a value to each software requirement based on its position in the network. Table 3.4 provides explanations of the network metrics used.

Table 3.4 : Network metrics.

Degree centrality	Degree centrality score is a simple measure which is correlated with number of connections of a software requirement in a graph.
Betweenness centrality	Betweenness centrality score is large for software requirements that lay on the shortest path many times.
Closeness centrality	This measure is an indicator of closeness of a node to others regarding network connections.
Eigenvector centrality	A greater eigenvector centrality score means a software requirement have a strong influence on other software requirements in graph via connections.

Hein et al. [19] created a network for requirements using linguistic data. Instead, we used formal requirement links to develop a network graph for software requirements in our thesis. This is accomplished through the usage of traceability links from software requirements to system requirements. Weights are allocated between software requirements considering mutual system requirement links. In our methodology, software requirements that are created through similar system requirements have a tendency to be closer in graph. Weight formula for requirement connections is given in equation 3.1. W symbol represents assigned weights for software requirements connections, NCLINK value is the total count of common system requirement links between two software requirements and NTOTLINK value is the total count of system requirements linked from those two software requirements. A symmetrical $n \times n$ matrix is formed after weight assignment, where n specifies the number of software requirements. The network centrality metrics are then calculated using this matrix.

$$W_{ij} = \frac{NCLINK_{i,j}}{NTOTLINK_{i,j}} \tag{3.1}$$

3.4 Model Output

The number of change requests for each software requirement is the output of our suggested model in this thesis. After reviewing and completing the Software Requirement Specification document for the safety-critical avionics software project, change requests relating to each requirement are recorded, and the necessary updates are done for SRS document by the development team. As a result, we define requirements volatility in our environmental setting as the number of change requests to add or update a software requirement. Worth bearing in mind that our model generates decimal values, but in fact, the number of modification requests for each software requirement can only be integer. As a result, fractional parts are rounded to the nearest integer.

3.5 Tools

As requirement management tool IBM Rational DOORS used in AVPRJ. Dxl scripts are written to get relevant textual and project feature metrics from requirement documents. Raw input data is processed in Microsoft Visual Studio to retrieve

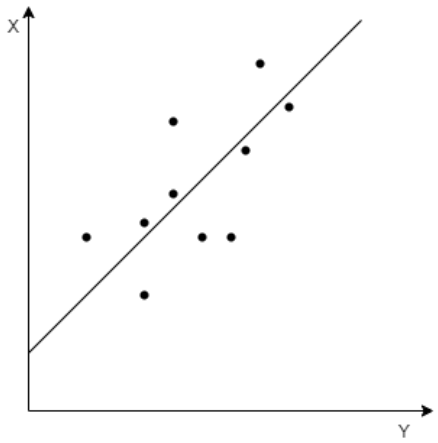

Figure 3.2 : A linear regression illustration.

textual metrics. The UCINET [30] is then used to generate network centrality metrics from the matrix we created regarding software and system requirement interrelations. WEKA tool [31] is utilized to train regression models with different machine learning techniques. The results of the prediction are then post-processed in MATLAB to generate performance measurements for all research questions.

3.6 Machine Learning Techniques

Random forest regression, linear regression, support vector regression and k-nearest neighbor regression methods are employed to train predictive models in this thesis. Linear regression was used in [1], while classifiers of the rest of the methods were utilized in [19]. Linear regression calculates coefficients for the best-fitting line or hyperplane to the training data. A visual illustration is provided in Figure 3.2. Random forest regression algorithm is based on creating many decision trees during training data and averaging prediction results of trees to make prediction for test instance. The diagram for the random forest regression algorithm flow is given in Figure 3.3. K-nearest neighbour regression technique makes predictions for test set regarding k closest training samples. Aim for support vector regression algorithm is to determine a line of best fit that minimizes a cost function's inaccuracy. This is accomplished by an optimization method that only takes into account the data instances in the training dataset that are closest to the line with the lowest cost [32]. Both instances and technique are named as support vectors. A visualisation of k-nearest neighbour and support vector regression methods is shown in Figure 3.4.

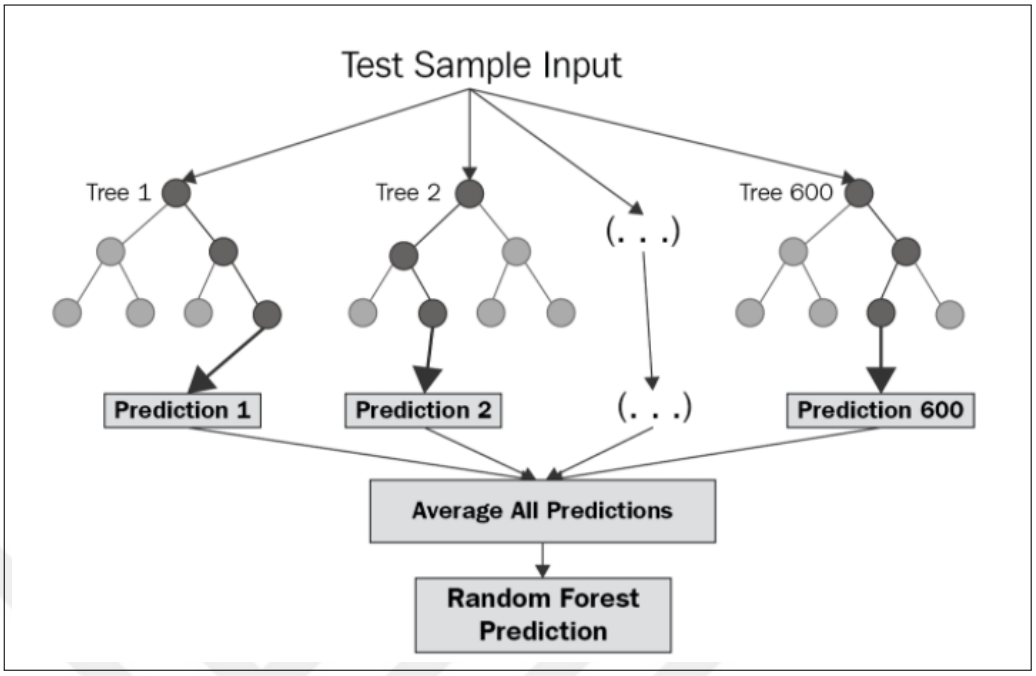

Figure 3.3 : Random forest regression structure [3].

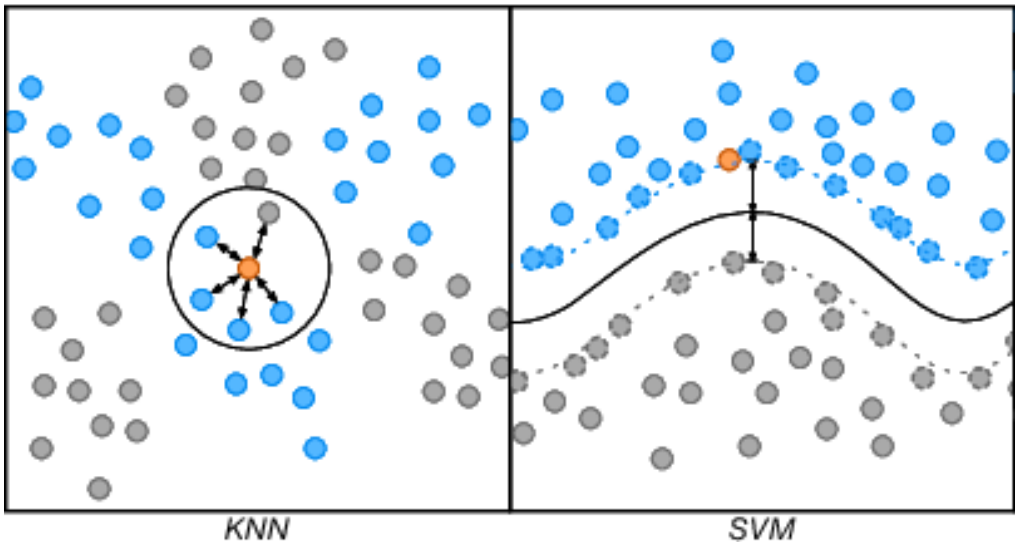

Figure 3.4 : K-nearest neighbour and support vector regression visualisation [4].

The dataset extracted from the safety-critical avionics project under study is very large compared to datasets used by previous studies. A larger dataset is usually desired for generalizability and credibility of results. Besides, a large dataset would affect k parameter value for k-nearest neighbor regression. A smaller value for k parameter could lead to over-fitting for a large training set. Thus, the k parameter is set to a comparatively large value such as 181, since approximately 20,000 samples are used in training set. 181 closest training samples, still less than one percent in training set, are used to decide on predicted change requests for a test instance.

WEKA allows to tune machine learning parameters for the employed techniques. The inversely proportional weighting option is chosen for k-nearest neighbor regression. Closer training samples are given higher weights, resulting in better prediction outcomes for our predictive model. The radial basis function kernel is chosen for support vector regression which is frequently preferred. Over-fitting can occur if the gamma parameter is increased too high [33] and for large gamma we similarly experienced a high computational cost with almost no prediction success gain. As a result, the C and gamma parameters are set to 1.

The 10-fold cross validation method is employed to divide the dataset into training and test sets in this research. To begin, the entire dataset of software requirements is mixed up randomly and divided into ten groups of roughly equal size. One group is tagged as a test set, while the other groups are utilized to train machine learning regression models. This technique is performed ten times until each distinct group has been utilized as a test group once.

3.7 Performance Evaluation

The following performance measures are used to evaluate RQ 1: Mean Magnitude of Relative Error (MMRE), Median Magnitude of Relative Error (MdMRE), Pred(0.5) and Pred(0.25) [34]. Those performance measurements are also used in the prior study by Loconsole et al. [1]. Relative error is calculated according to equation 3.2. $Err_{relative}$ is relative error, Val_{act} is the actual value, whereas Val_{pred} is the predicted value.

$$Err_{relative} = \frac{|Val_{act} - Val_{pred}|}{|Val_{act}|} \tag{3.2}$$

There are unchanged requirements which means zero change requests are attached. Therefore when calculating relative error for these unchanged requirements division by zero problem occurs. For those cases, we made an assumption to calculate relative error for software requirements with zero change requests as shown in equation 3.3.

$$\text{If } Val_{act} = 0 \qquad Err_{relative} = \frac{\left|Val_{act} - Val_{pred}\right|}{1} \tag{3.3}$$

The variability of the error can be measured by Pred(k). This performance measure relies on relative error, and it displays the percentage of predictions with errors smaller than or equal to k.

Regarding Research Question 2, our purpose is to predict highly volatile software requirements, for that reason we use a method to identify highly volatile requirements:

- Step 1: Sort software requirements within the dataset with respect to their actual number of change requests in descending order. Report their rank in this sorting as R_{actual}.
- Step 2: By using predictive model gather regression prediction results of each sample in the dataset.
- Step 3: Following obtainment of prediction results, sort software requirements regarding their predicted number of change requests in descending order this time. Report their rank in this sorting as $R_{predicted}$.
- Step 4: Assess ranking values regarding the listing in Table 3.5. P represents the percentage of software requirements that are regarded as highly volatile, and N_{req} refers to the total count of requirements in validation set.
- Step 5: Calculate performance measures for second research question: accuracy, recall, false alarm rate.

Table 3.5 can be read as follows: True Positive evaluations are those that are genuinely highly volatile, where also the predictive model classifies them as such. A requirements is actually less volatile in the case of True Negatives, as is its prediction. False Negatives emerge when the predictor considers highly volatile requirements to

Table 3.5 : Requirements volatility sorting assessment for Research Question 2.

Case	Assessment
R_{actual},$R_{predicted} \leq N_{req} \times P$	True Positive
R_{actual},$R_{predicted} > N_{req} \times P$	True Negative
$R_{actual} \leq N_{req} \times P$, $R_{predicted} > N_{req} \times P$	False Negative
$R_{actual} > N_{req} \times P$, $R_{predicted} \leq N_{req} \times P$	False Positive

be less volatile. Lastly, False Positive requirements are less volatile, but are projected to be highly volatile.

Recall, accuracy, and false alarm rate are calculated to answer RQ2. Recall scores demonstrates how well the model predicts highly volatile requirements. This, in our opinion, is the most crucial RQ2 performance indicator. Accuracy measurement reports the prediction success for both less-volatile and highly volatile software requirements. False alarm rate shows the amount of work has gone to waste due to miscalculation of less volatile requirements.

4. RESULTS AND DISCUSSION

In this chapter, we report and discuss the models' success in relation to two research questions. Additionally, performance results of the predictive models proposed in this thesis are compared with the previous study conducted by Loconsole et al. [1].

4.1 Research Question 1

After gathering the processed dataset, four regression algorithms are used to address the question of whether textual metrics, network centrality metrics, and project specific metrics could be utilized to estimate the number of software requirement changes using machine learning methods. The results of predictive model performance are obtained independently for each input metric - machine learning method combination. In this thesis four machine learners and seven input metric combinations are used, and thus, there are 28 predictive models in total.

Table 4.1 shows the RQ 1 results. The acronyms used in the table are as follows: KNN stands for k-nearest neighbor regression, LR stands for linear regression, RF stands for random forest regression, and SVR stands for support vector regression, ML stands for machine learning, T for requirement textual metrics, P for project specific metrics, N for network centrality metrics.

The best MMRE results are achieved with the following input metric combinations: T&P&N(0.366), T&N(0.381) and T&P(0.402). We can claim that textual metrics (T) are effective at predicting the number of change requests for each software requirement, and that combining them with other metrics yields even better results. Among the machine learning algorithms, when the k-nearest neighbor regression algorithm is used, the three top performing metric combinations deliver the best prediction success.

For the following input metric and machine learning algorithm combinations, MdMRE is zero: T&N+SVR, T&N+RF, T&N+KNN, P&N+KNN, T&P+KNN, T&P&N+SVR,

Table 4.1 : Performance evaluation results for Research Question 1.

Metrics+ML method	MMRE	MdMRE	Pred(0.5)	Pred(0.25)
T&P&N+KNN	0.366	0	0.681	0.57
T&P&N+LR	0.53	0.5	0.524	0.411
T&P&N+RF	0.392	0	0.663	0.545
T&P&N+SVR	0.392	0	0.662	0.554
T&P+KNN	0.402	0	0.641	0.529
T&P+LR	0.513	0.5	0.541	0.428
T&P+RF	0.45	0.333	0.6	0.486
T&P+SVR	0.459	0.5	0.584	0.479
P&N+KNN	0.422	0	0.632	0.52
P&N+LR	0.55	0.667	0.484	0.372
P&N+RF	0.454	0.5	0.595	0.48
P&N+SVR	0.469	0.5	0.561	0.475
T&N+KNN	0.381	0	0.665	0.553
T&N+LR	0.534	0.5	0.52	0.407
T&N+RF	0.394	0	0.662	0.545
T&N+SVR	0.426	0	0.621	0.515
T+KNN	0.443	0.333	0.598	0.488
T+LR	0.512	0.5	0.542	0.43
T+RF	0.455	0.5	0.594	0.483
T+SVR	0.483	0.5	0.556	0.446
P+KNN	0.555	0.667	0.483	0.37
P+LR	0.548	0.667	0.485	0.373
P+RF	0.556	0.667	0.483	0.371
P+SVR	0.516	0.5	0.512	0.417
N+KNN	0.448	0.5	0.596	0.482
N+LR	0.549	0.667	0.484	0.372
N+RF	0.485	0.5	0.561	0.446
N+SVR	0.53	0.5	0.5	0.392

T&P&N+RF and T&P&N+KNN. These predictive models successfully forecast the number of change requests for more than half of the software requirements. The most successful models are not sorted in terms of MdMRE because numerous prediction models produce the same best performance result.

Best performing models regarding pred(0.25) measure are T&P&N+KNN (0.57), T&N+KNN (0.553), T&P+KNN (0.529) gives third best result. Similar to other performance evaluation results, textual assessment metrics and k-nearest neighbor algorithm are successful for this measure as well.

When we change our focus to pred(0.50) results best performing model is T&P&N+KNN (0.681). T&N+KNN (0.665) gives second and T&P+KNN (0.641)

gives third leading result. Requirement textual metrics and k-nearest neighbor algorithm are employed by best performing three models for this measure as well.

To summarize, the greatest performance scores are obtained by combining the requirement textual measurements, project specific metrics, and network metrics. With respect to the results of RQ1, K-nearest neighbor regression gives the best results for all performance evaluation measures. Textual measurements are used in all of the best-performing models, either in conjunction with project or network metrics, or as a combination of all input metrics. The way software requirement specifications are reported appears to have a significant impact on requirement volatility rates.

We compared our RQ1 findings with the research carried out by Loconsole et al. [1]. The success of the linear regression model with the best metric specified in our study, as well as our most successful model and best performing model of Loconsole et al. [1], is reported in the Table 4.2. When comparing the data on linear regression, we see that utilizing the number of lines metric predicts requirement volatility better in their context, whereas using requirement textual indicators alone does not deliver the best result in our commercial setting. Other algorithms, such as KNN, when used in conjunction with all metrics considerably increase prediction success by lowering MMRE to 0.36 and MdMRE to 0, and increasing Pred(0.25) to 57%.

Table 4.2 : Comparison of our performance (RQ 1) against [1].

	MMRE	**MdMRE**	**Pred(0.25)**	**Pred(0.5)**
T+LR	0.51	0.5	0.43	0.54
Best model	0.36	0	0.57	0.68
NLines+LR [1]	0.58	0.27	0.5	0.63

4.2 Research Question 2

Research Question 2 intends to assess our model's ability to predict highly volatile requirements. In Section 3.7, we describe our method for identifying these requirements. To label highly volatile software requirements, we must first determine change request coverage, and then compute recall, accuracy, and false alarm rates.

Table 4.3 shows the rates for different change request coverage by the most volatile software requirements. When change request coverage gets larger more software

requirements are marked as highly volatile. In this thesis different change request performance results are reported. However results belonging to 80% change request coverage is the most valuable in this study, since roughly 40% percent of reviewers are regarded as experienced in the safety-critical avionics software project. As a result of using this strategy, we were able to assign 38.6% of total requirements, which are potentially highly volatile, to experienced developers in the earlier stages of software development. Other coverage results are also presented to show how would model performance evaluation change if experienced reviewer ratio were different than 40%.

Table 4.3 : Change request coverage statistics for project under study.

Change Request Coverage	**Requirement Coverage**
60%	20.5%
70%	29.6%
80%	38.6%
90%	47.7%

Firstly, 80% change request coverage results will be reported since in this project we have 40% of experienced reviewers. The leading recall scores are T&P&N+KNN(0.632), T&N+RF(0.616) and T&P+KNN(0.604) as presented in Table 4.4. Similar to RQ 1 findings the most successful predictive models have textual metrics, where the best performance results are obtained by using input metrics altogether. The best accuracy scores are T&P&N+KNN(0.716), T&N+RF(0.703) and T&P+KNN(0.694) whereas the best and lowest false alarm rate scores are T&P&N+KNN(0.232), T&N+RF(0.242) and T&P+KNN(0.249). K-nearest neighbor regression and random forest regression machine learning techniques are good at detecting highly volatile software requirements for 80% change request coverage.

Because the goal of second research question is to assess performance in predicting highly volatile requirements, recall is the most relevant metric. We properly identify 63.2 percent of highly volatile requirements that are subject to 80 percent of all requirement changes.

Other performance results for 90%, 70% and 60% change request coverage are reported in Table 4.5, Table 4.6, and Table 4.7 to observe if research question 2 findings would differ if AVPRJ had a different number of experienced reviewers. The best performance result for 90% change request coverage is T&P&N+KNN(Recall:

Table 4.4 : Performance results of predictive models for RQ 2 with 80% CR coverage.

Metrics+ML method	Recall	Accuracy	False Alarm Rate
T&P&N+KNN	0.632	0.716	0.232
T&P&N+LR	0.531	0.638	0.295
T&P&N+RF	0.624	0.71	0.237
T&P&N+SVR	0.591	0.684	0.258
T&P+KNN	0.604	0.694	0.249
T&P+LR	0.508	0.62	0.31
T&P+RF	0.602	0.692	0.251
T&P+SVR	0.558	0.658	0.278
P&N+KNN	0.574	0.671	0.268
P&N+LR	0.418	0.55	0.366
P&N+RF	0.557	0.658	0.279
P&N+SVR	0.496	0.61	0.318
T&N+KNN	0.614	0.702	0.243
T&N+LR	0.53	0.637	0.296
T&N+RF	0.616	0.703	0.242
T&N+SVR	0.569	0.667	0.271
T+KNN	0.586	0.68	0.261
T+LR	0.508	0.62	0.31
T+RF	0.582	0.677	0.263
T+SVR	0.529	0.636	0.296
P+KNN	0.503	0.616	0.313
P+LR	0.423	0.554	0.363
P+RF	0.496	0.611	0.317
P+SVR	0.462	0.584	0.339
N+KNN	0.557	0.657	0.279
N+LR	0.404	0.539	0.376
N+RF	0.565	0.664	0.274
N+SVR	0.498	0.612	0.316

0.707, Accuracy:0.721, False Alarm Rate:0.267), for 70% change request coverage is T&P&N+RF(Recall: 0.581, Accuracy:0.752, False Alarm Rate:0.176), for 60% change request coverage is T&P&N+RF(Recall: 0.582, Accuracy:0.828, False Alarm Rate:0.108). For other change request coverage cases, again the most desired results are gathered by utilizing textual, project specific and network metrics altogether. For 90% change request coverage K-nearest neighbor regression, whereas for 70% and 60% change request coverage random forest regression are used by the best performing predictive models. However, the difference between performance results of T&P&N+KNN and T&P&N+RF is very small for 70% and 60% change request

coverage cases. Therefore, we can claim K-nearest neighbor regression machine learner is still powerful but in some cases random forest regression could give slightly better results for predicting highly volatile software requirements.

Table 4.5 : Performance results of predictive models for RQ 2 with 90% CR coverage.

Metrics+ML method	**Recall**	**Accuracy**	**False Alarm Rate**
T&P&N+KNN	0.707	0.721	0.267
T&P&N+LR	0.589	0.608	0.375
T&P&N+RF	0.69	0.705	0.282
T&P&N+SVR	0.667	0.682	0.304
T&P+KNN	0.671	0.686	0.3
T&P+LR	0.58	0.599	0.383
T&P+RF	0.662	0.677	0.308
T&P+SVR	0.626	0.643	0.341
P&N+KNN	0.661	0.677	0.309
P&N+LR	0.468	0.492	0.485
P&N+RF	0.641	0.658	0.327
P&N+SVR	0.59	0.609	0.374
T&N+KNN	0.687	0.702	0.285
T&N+LR	0.591	0.61	0.373
T&N+RF	0.682	0.697	0.29
T&N+SVR	0.643	0.66	0.325
T+KNN	0.652	0.668	0.317
T+LR	0.578	0.597	0.385
T+RF	0.641	0.658	0.327
T+SVR	0.603	0.621	0.362
P+KNN	0.555	0.576	0.406
P+LR	0.502	0.525	0.454
P+RF	0.552	0.573	0.408
P+SVR	0.534	0.555	0.425
N+KNN	0.64	0.657	0.328
N+LR	0.475	0.499	0.479
N+RF	0.629	0.646	0.338
N+SVR	0.555	0.575	0.406

Table 4.6 : Performance results of predictive models for RQ 2 with 70% CR coverage.

Metrics+ML method	**Recall**	**Accuracy**	**False Alarm Rate**
T&P&N+KNN	0.578	0.75	0.177
T&P&N+LR	0.486	0.696	0.216
T&P&N+RF	0.581	0.752	0.176
T&P&N+SVR	0.531	0.722	0.197
T&P+KNN	0.549	0.733	0.19
T&P+LR	0.461	0.681	0.227
T&P+RF	0.551	0.734	0.189
T&P+SVR	0.503	0.706	0.209
P&N+KNN	0.509	0.71	0.206
P&N+LR	0.375	0.63	0.263
P&N+RF	0.504	0.706	0.209
P&N+SVR	0.421	0.657	0.244
T&N+KNN	0.56	0.739	0.185
T&N+LR	0.482	0.694	0.218
T&N+RF	0.576	0.749	0.178
T&N+SVR	0.502	0.705	0.209
T+KNN	0.537	0.726	0.195
T+LR	0.456	0.678	0.229
T+RF	0.543	0.729	0.192
T+SVR	0.472	0.687	0.222
P+KNN	0.445	0.672	0.233
P+LR	0.337	0.608	0.279
P+RF	0.432	0.664	0.239
P+SVR	0.264	0.564	0.31
N+KNN	0.488	0.697	0.215
N+LR	0.339	0.608	0.278
N+RF	0.492	0.699	0.214
N+SVR	0.345	0.612	0.276

Table 4.7 : Performance results of predictive models for RQ 2 with 60% CR coverage.

Metrics+ML method	**Recall**	**Accuracy**	**False Alarm Rate**
T&P&N+KNN	0.559	0.819	0.114
T&P&N+LR	0.447	0.773	0.143
T&P&N+RF	0.582	0.828	0.108
T&P&N+SVR	0.518	0.802	0.125
T&P+KNN	0.52	0.803	0.124
T&P+LR	0.415	0.759	0.151
T&P+RF	0.538	0.81	0.12
T&P+SVR	0.479	0.786	0.135
P&N+KNN	0.484	0.788	0.133
P&N+LR	0.322	0.721	0.175
P&N+RF	0.471	0.783	0.137
P&N+SVR	0.354	0.735	0.167
T&N+KNN	0.534	0.808	0.121
T&N+LR	0.433	0.767	0.147
T&N+RF	0.566	0.821	0.112
T&N+SVR	0.474	0.784	0.136
T+KNN	0.508	0.798	0.127
T+LR	0.4	0.753	0.155
T+RF	0.514	0.8	0.126
T+SVR	0.434	0.767	0.146
P+KNN	0.29	0.708	0.184
P+LR	0.205	0.673	0.206
P+RF	0.3	0.712	0.181
P+SVR	0.23	0.683	0.199
N+KNN	0.442	0.771	0.144
N+LR	0.279	0.704	0.186
N+RF	0.44	0.77	0.145
N+SVR	0.251	0.692	0.194

5. THREATS TO VALIDITY

In this thesis we reported an experiment-based empirical study. Therefore, in this chapter we investigate threats to internal validity, external validity, construct validity and conclusion validity.

Internal validity: In this paper, we show how the volatility of software requirements in a project can be anticipated to some extent using a combination of requirement textual measures, project-specific characteristics, and network metrics. However, findings of our study does not allege a causal relation between input metrics we employed and predictive model output. Because, in this work we did not conduct a controlled experiment, so there is no evidence for causal relationship.

External validity: The case study is carried out on one project in ASELSAN. The dataset, however, is rather enormous, containing over 20,000 requirements from three different releases created by a variety of software developers. However, in terms of generalization of outcomes, using the prediction models on upcoming new projects hereafter would be beneficial. Additionally, researchers from different industrial organizations may apply these predictive models in their commercial settings to verify success of models on predicting volatility of requirements.

Construct validity: Software specifications were not written in the developers' native tongue. As a result, there may be occasional linguistic errors that change the textual requirement textual metric scores. Developers may also utilize certain phrases in software requirement specifications, e.g. speculative expressions that should have been taken into account when developing textual measurements but we failed to do so. We were not able to manually analyze these types of typos and grammatical problems due to the size of the dataset, but we do know that the team that carry out review activity is responsible for rectifying errors. We build network diagrams relying on the SRS's traceability relationships between software and system requirements. As in prior research [19], we may have used lingual relations to create a graph for software

requirements, which may better portray the relationship between requirements. Our intention is to accomplish this on new software projects.

Conclusion validity: For the second research question the results of 80% change request coverage are reported as success of predictive models on predicting highly volatile requirements. However, labeling requirements as highly volatile is dependent on development environment and it is subjective. In this study 38.6% of software requirements are labeled as highly volatile, since approximately 40% of reviewers are regarded as well experienced. However, in other software projects it could be costly assigning that amount of valuable labor force to reduce requirements volatility related risks. Thus project managers may want to label less requirements as highly volatile with covering moderate amount of change requests. For this particular safety-critical avionics software project approximately 5% of most volatile requirements cover 25% of total change requests. Accordingly, even with assigning small number of experienced labor such as 5%, it could be possible to cover considerable amount of possible highly volatile requirements with proposed predictive models. To sum up, highly volatile requirement labeling is subjective and success of predictive models on detecting highly volatile requirements would have been different if another change request coverage is chosen to mark highly volatile requirements.

6. CONCLUSIONS AND RECOMMENDATIONS

In this thesis, we used requirement textual metrics, project characteristics, and requirement inter-dependencies to conduct an empirical analysis to predict the amount of changes for each software requirement. A safety-critical avionics software project from ASELSAN consisting of 22,771 software requirements is used to train 28 prediction models. We evaluate the outperforming combination of metric and machine learning algorithm. To conclude, by analyzing metrics of similar requirements through K-nearest neighbor regression, we can predict software requirement volatility with MMRE of 36%. We've also noticed that measuring software requirements from several perspectives, such as textual, project, and network graph dependencies, yields significantly better results. Also the best performing model predicts highly volatile software requirements, which are exposed to 80% of total software requirement changes, with 63.2% success rate.

The usage of this predictive model is defined as predicting highly volatile requirements before Software Requirement Specification document completion to assign review task of probable highly volatile requirements to experienced reviewers. Therefore it is expected to find inconsistencies and ambiguities among software requirements in early phases to prevent requirement volatility project risks. All input metrics could be extracted before completing Software Requirement Specification document in our industrial context. Project specific metrics are recorded as distinct features of software requirements in requirement management tool and expected to be completed before SRS review activity. Traceability links between software and system requirements are needed to extract network metrics again should be completed before signing off Software Requirement Specification document. Textual metrics could also be extracted in that phase. However, extracting all input metrics before completing SRS is not prerequisite to use this predictive model in requirements volatility prediction and different organizations can use this predictive model in different ways. For example this predictive model can be used in agile software development as well. The predictive

model can be applied to a software project in any phase when input metric extraction is possible and probable highly volatile requirements can be labeled. Later on when an issue is created and story point estimation is needed to calculate effort to be spent resolving the issue this predictive model can be used. If an issue requires changes in probable highly volatile requirements higher story points can be assigned and maybe more experienced testers or reviewers can be assigned to resolve this issue. Thus organizations could use this predictive model in different ways and in any phase.

We want to incorporate such a predictor model into requirement management tools e.g. DOORS, which will be employed before the Software Requirement Specification review activity, in order to identify highly volatile requirements automatically and reliably. As a result, software leads would take steps ahead of time to mitigate the risks associated with requirements change proneness. Because there aren't enough empirical studies in this field, more empirical research is needed to validate the best-performing models.

REFERENCES

[1] **Loconsole, A. and Börstler, J.** (2007). Construction and Validation of Prediction Models for Number of Changes to Requirements, **Umeå University Technical Report UMINF-07.03**, Umeå University Department of Computing Science, UMEÅ, SWEDEN.

[2] **Nurse, J.R.** (2015). Exploring the risks to identity security and privacy in cyberspace, *XRDS: Crossroads, The ACM Magazine for Students*, *21*(3), 42–47.

[3] Random Forest Regression, `https://levelup.gitconnected.com/random-forest-regression-209c0f354c84`, accessed: 2021-12-29.

[4] **Page, A.**, **Turner, J.**, **Mohsenin, T. and Oates, T.** (2014). Comparing raw data and feature extraction for seizure detection with deep learning methods, *The Twenty-Seventh International Flairs Conference*.

[5] **Nurmuliani, N.**, **Zowghi, D. and Powell, S.** (2004). Analysis of requirements volatility during software development life cycle, *2004 Australian Software Engineering Conference*, ASWEC '04, IEEE, Melbourne, VIC, Australia, pp.28–37.

[6] **Swathi, G.**, **Jagan, A. and Prasad, C.** (2011). Writing Software Requirements Specification Quality Requirements: An Approach to Manage Requirements Volatility, *Int. J. Comp. Tech. Appl.*, *2*(3), 631–638.

[7] **Ambriola, V. and Gervasi, V.** (2000). Process Metrics for Requirement Analysis, *7th European Workshop on Software Process Technology*, Springer, Kaprun, Austria, pp.90–95.

[8] **Javed, T.**, **Maqsood, M.e. and Durrani, Q.S.** (2004). A Study to Investigate the Impact of Requirements Instability on Software Defects, *ACM SIGSOFT Software Engineering Notes*, *29*(3), 1–7.

[9] **Costello, R.J. and Liu, D.B.** (1995). Metrics for requirements engineering, *Journal of Systems and Software*, *29*(1), 39–63.

[10] **Ferreira, S.**, **Collofello, J.**, **Shunk, D. and Mackulak, G.** (2009). Understanding the effects of requirements volatility in software engineering by using analytical modeling and software process simulation, *Journal of Systems and Software*, *82*(10), 1568–1577.

[11] **Christopher, D.F.X. and Chandra, E.** (2012). Prediction of Software Requirements Stability Based On Complexity Point Measurement Using

Multi-Criteria Fuzzy Approach, *International Journal of Software Engineering & Applications*, *3*(6), 101–115.

[12] **Thakurta, R.** (2011). A Mixed Mode Analysis of the Impact of Requirement Volatility on Software Project Success, *Journal of International Technology and Information Management*, *20*(1).

[13] **Alsalemi, A.M. and Yeoh, E.T.** (2017). A Systematic Literature Review of Requirements Volatility Prediction, *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication*, ICCTCEEC-2017, IEEE, Mysore, India, pp.55–64.

[14] **Shi, L.**, **Wang, Q. and Li, M.** (2013). Learning from evolution history to predict future requirement changes, *2013 21st IEEE International Requirements Engineering Conference*, RE-2013, IEEE, Rio de Janeiro, Brazil, pp.135–144.

[15] **Goknil, A.**, **van Domburg, R.**, **Kurtev, I.**, **van den Berg, K. and Wijnhoven, F.** (2014). Experimental evaluation of a tool for change impact prediction in requirements models: Design, results, and lessons learned, *2014 IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE)*, MoDRE 2014, IEEE, Karlskrona, Sweden, pp.57–66.

[16] **Morkos, B.**, **Mathieson, J. and Summers, J.D.** (2014). Comparative analysis of requirements change prediction models: manual, linguistic, and neural network, *Research in Engineering Design*, *25*(2), 139–156.

[17] **Wang, X.**, **Wu, C. and Ma, L.** (2010). Software project schedule variance prediction using Bayesian Network, *2010 IEEE International Conference on Advanced Management Science*, volume 2 of *ICAMS 2010*, IEEE, Chengdu, China, pp.26–30.

[18] **Nakatani, T. and Tsumaki, T.** (2014). Predicting requirements changes by focusing on the social relations, *Proceedings of the Tenth Asia-Pacific Conference on Conceptual Modelling*, volume154 of *APCCM 2014*, Auckland, New Zealand, pp.65–70.

[19] **Hein, P.H.**, **Kames, E.**, **Chen, C. and Morkos, B.** (2021). Employing machine learning techniques to assess requirement change volatility, *Research in Engineering Design*, *32*(2), 245–269.

[20] **Pedrycz, W.**, **Iljazi, J.**, **Sillitti, A. and Succi, G.**, (2016). Prediction of the Successful Completion of Requirements in Software Development—An Initial Study, Agent and Multi-Agent Systems: Technology and Applications, Springer, pp.261–269.

[21] **Anang, Y.**, **Takahashi, M. and Watanabe, Y.** (2016). A Method for Software Requirement Volatility Analysis Using QFD, *Complex Systems Informatics and Modeling Quarterly*, (8), 1–14.

[22] **Ostrand, T.J. and Weyuker, E.J.** (2007). How to measure success of fault prediction models, *Fourth international workshop on Software quality*

assurance: in conjunction with the 6th ESEC/FSE joint meeting, SOQUA'07, Dubrovnik, Croatia, pp.25–30.

[23] **Knight, J.C.** (2002). Safety critical systems: challenges and directions, *Proceedings of the 24th international conference on software engineering*, pp.547–550.

[24] **Clancy, T.** (1995). The standish group report, *Chaos report*.

[25] **Jiang, Y.**, **Cukic, B. and Menzies, T.** (2007). Fault prediction using early lifecycle data, *The 18th IEEE International Symposium on Software Reliability*, ISSRE'07, IEEE, pp.237–246.

[26] **Génova, G.**, **Fuentes, J.M.**, **Llorens, J.**, **Hurtado, O. and Moreno, V.** (2013). A framework to measure and improve the quality of textual requirements, *Requirements engineering*, *18*(1), 25–41.

[27] **Faisandier, A.** (2013). *Systems architecture and design*, Sinergy'Com Belberaud, France.

[28] **Barrat, A.**, **Barthelemy, M.**, **Pastor-Satorras, R. and Vespignani, A.** (2004). The architecture of complex weighted networks, *Proceedings of the national academy of sciences*, *101*(11), 3747–3752.

[29] **Valente, T.W.**, **Coronges, K.**, **Lakon, C. and Costenbader, E.** (2008). How correlated are network centrality measures?, *Connections*, *28*(1), 16–26.

[30] **Borgatti, S.P.**, **Everett, M.G. and Freeman, L.C.** (2002). Ucinet for Windows: Software for social network analysis, *Harvard, MA: analytic technologies*, *6*.

[31] **Eibe, F.**, **Hall, M.A. and Witten, I.H.**, (2016). The WEKA workbench. Online appendix for data mining: practical machine learning tools and techniques, Morgan Kaufmann.

[32] Random Forest Regression, `https://levelup.gitconnected.com/random-forest-regression-209c0f354c84`, accessed: 2021-12-29.

[33] **Ben-Hur, A. and Weston, J.**, (2010). A user's guide to support vector machines, Data mining techniques for the life sciences, Springer, pp.223–239.

[34] **Zhang, D. and Tsai, J.J.** (2005). *Machine learning applications in software engineering*, volume 16, World Scientific.

CURRICULUM VITAE

Name Surname : Anıl Holat

EDUCATION :

- **B.Sc.** : 2014, Bilkent University, Faculty of Engineering, Department of Electrical and Electronics Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2015-Present, Software Engineer/ASELSAN
- 2014-2015, Software Design Engineer/VESTEL

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Holat, A. and Tosun,A** 2021. Predicting Requirements Volatility: An Industry Case Study. *9th International Workshop on Quantitative Approaches to Software Quality*, December 6, 2021 Taipei, Taiwan.