AZƏRBAYCAN RESPUBLİKASI ELM VƏ TƏHSİL NAZİRLİYİ

AZƏRBAYCAN TEXNİKİ UNİVERSİTETİ

Əlyazması hüququnda

Qarayev İbrahim Məhəmməd oğlu

Qüdrətov Orxan Habil oğlu

İsrafilli İsmayıl Saleh oğlu

PROQRAM KODLARININ MÜRƏKKƏBLİYİNİN ÖLÇÜLMƏSİ ÜSULLARININ TƏDQİQİ

mövzusunda

MAGİSTRLİK DİSSERTASİYASI

İxtisas: 050615 – "İnformasiya təhlükəsizliyi"

İxtisaslaşma: "Məlumatların mühafizəsinin təşkili"

Elmi rəhbər: **Elçin Adil oğlu Əliyev**

BAKI – 2024

MÜNDƏRİCAT

AZƏRBAYCAN TEXNİKİ UNİVERSİTETİ

YÜKSƏK TƏHSİL İNSTİTUTU

MAGİSTRANTIN ANDI

Proqram kodlarının mürəkkəbliyinin ölçülməsi üsullarının tədqiqi mövzusunda təqdim etdiyimizmagistrlik dissertasiyasını elmi əxlaq normalarına və istinad qaydalarına tam riayətetməklə və istifadə etdiyim bütün mənbələri ədəbiyyat siyahısında əks etdirməklə yazdığımıza and içirik və magistrlik dissertasiyasının AzTU Kitabxana İnformasiya Mərkəzində saxlanılması, həmin mərkəz tərəfindən AzTU Rəqəmsal Repozitoriyasına daxil edilərək repozitoriyanın veb saytında yerləşdirilməsinə icazə veririk.

Qarayev İbrahim

Qüdrətov Orxan

İsrafilli İsmayıl

Tarix

İXTİSARLARIN SİYAHISI

CC - Cyclomatic Complexity (Tsiklomatik Mürəkkəblik)

CI/CD - Continuous Integration/Continuous Delivery (Davamlı İnteqrasiya və Davamlı Yerləşdirmə)

DIT - The Depth of Inheritance Tree (Varislik Ağacının dərinliyi)

LCOM - Lack of Cohesion of Methods (Metodların uyğun olmaması)

LOC - Lines of Code (Kod Sətirləri)

MI - Maintabiliyt Index (Davamlılıq İndeksi)

GİRİŞ

Mövzunun aktuallığı. Proqram kodlarının mürəkkəbliyinin ölçülməsi üsullarının tədqiqi mövzusunun seçilməsi müasir texnoloji mühitdə proqram təminatının inkişafının artan əhəmiyyətindən, onun hazırlanmasında və texniki xidmətində kod mürəkkəbliyinin oynadığı mühüm roldan irəli gəlir. Proqram sistemlərinin miqyası və mürəkkəbliyi artdıqca onun davamlılığını, etibarlılığını və ümumi proqram keyfiyyətini təmin etmək üçün kodun mürəkkəbliyini başa düşmək və idarə etmək daha da çətinləşir. Yüksək kod mürəkkəbliyi kodun xətalara meyliliyini daha da artırır və kodun oxunaqlılığını azaldır. Bu hal da öz növbəsində proqram təminatının inkişaf prosesinə mane olur və texniki xidmət xərclərini artırır. Məhz bu tədqiqat mövzusu kodun mürəkkəbliyi üçün mövcud ölçmə və metodologiyaları hərtərəfli qiymətləndirmək, onların güclü tərəflərinə, məhdudiyyətlərinə və praktiki təsirlərinə işıq salmaq ehtiyacı ilə seçilmişdir. Bu tədqiqat, mürəkkəblik ölçmə üsullarının tarixi inkişafını və müasir vəziyyətini araşdıraraq, ən yaxşı tətbiqləri tənzimləyə biləcək, proqram keyfiyyətini artıra biləcək və proqram mühəndisliyi metodologiyalarının davamlı olaraq təkmilləşdirilməsinə kömək edə biləcək qiymətli məlumatlar təmin etməyi hədəfləyir.

Tədqiqatın məqsədi və vəzifələri. Bu tədqiqatın məqsədi proqram təminatının inkişafı çərçivəsində proqram kodlarının mürəkkəbliyinin ölçülməsi üsullarını hərtərəfli araşdırmaq və qiymətləndirməkdir. Əsas məqsədlərə aşağıdakılar daxildir:

- Mövcud meyarların qiymətləndirilməsi;
- Eksperimental təhlil;
- Korrelyasiyaların müəyyən edilməsi;
- Təkmilləşdirmə tövsiyələri.

Bu məqsədlərə nail olmaqla, tədqiqat proqram tərtibatçılarını, tədqiqatçıları və digər maraqlı tərəfləri proqram kodlarının mürəkkəbliyini idarə etmək və təkmilləşdirmək üçün effektiv strategiyalar haqqında məlumatlandıra bilən və nəticədə proqram təminatının inkişafına töhfə verən dəyərli məlumatları təmin etmək məqsədi daşıyır.

Tədqiqatın vəzifələrinə aşağıdakılar daxildir:

- Müxtəlif mürəkkəblik meyarlarını təhlil etmək və bu meyarların güclü və zəif tərəflərini qiymətləndirmək;

- Tədqiqatın nəticələrinə əsasən praktiki tövsiyələr vermək;

- Alınan nəticələri təhlil etmək, ümumi nəticələrə əsaslanaraq akademik və peşəkar bilik bazasına töhfə vermək.

Tədqiqatın predmenti və obyekti. Tədqiqatın predmeti müasir proqram təminatı mühəndisliyində proqram kodlarının artan mürəkkəbliyinin yaratdığı problemlərin tanınması , təhlili və həll edilməsidir. Tədqiqatın obyekti isə proqram kodlarının mürəkkəbliyini ölçmək üçün istifadə olunan metodlardır.

Tədqiqat metodları. Bu tədqiqatda ədəbiyyat təhlili və sənədlərin nəzərdən keçirilməsi üsullarından istifadə edilir. Ədəbiyyat icmalı mövcud elmi araşdırmaları və proqram təminatının inkişaf etdirilməsi təcrübələrini araşdıraraq, kodun mürəkkəbliyinə təsir edən amilləri və ölçmə üsullarını müəyyən etmək üçün istifadə olunacaq. Sənədin nəzərdən keçirilməsi müxtəlif keyfiyyət ölçmə modellərini və kodun mürəkkəbliyini qiymətləndirmək üçün istifadə olunan ölçüləri təhlil etmək üçün istifadə olunacaq.

Tədqiqatın elmi yeniliyi. Bu işin elmi yeniliyi ondan ibarətdir ki, proqram təminatının yaradılması proseslərində kodun mürəkkəbliyini ölçmək və qiymətləndirmək üçün istifadə olunan üsullar hərtərəfli tədqiq edir. Bundan əlavə, tədqiqat kodun mürəkkəbliyini idarə etmək üçün tövsiyələt verməklə, proqramçılara rəhbərlik etmək məqsədi daşıyır.

Tədqiqatın praktiki və nəzəti əhəmiyyəti. Praktiki əhəmiyyət baxımından, proqram təminatın mürəkkəbliyi tətbiqin davamlılığı, təkrar istifadə oluna bilməsi, etibarlılığı və performansı kimi amillərə əhəmiyyətli təsir göstərir.Mürəkkəb kod səhvləri tapmaq və düzəltməyi çətinləşdirə, proqram təminatının ümumi işinə mənfi təsir göstərə, yeni funksiyaların əlavə edilməsinə və ya mövcud kodun dəyişdirilməsinə mane ola bilər. Buna görə də, mürəkkəbliyin ölçülməsi proqram təminatının hazırlanması prosesində keyfiyyətin yaxşılaşdırılması və xərclərin

azaldılması üçün mühüm vasitədir. Ölçülmüş mürəkkəblik səviyyələrinə əsasən tərtibatçılar kodu sadələşdirə, yenidən strukturlaşdıra və ya yenidən dizayn edə bilərlər.

Nəzəri əhəmiyyəti proqram təminatının mürəkkəbliyinin ölçülməsi fundamental problemdir. Bu, mürəkkəbliyin mücərrəd anlayışdan ölçülə bilən kəmiyyətə çevrilməsini tələb edir. Bu kontekstdə müxtəlif mürəkkəblik ölçüləri və ölçmə metodları üzrə araşdırmalar bizə mürəkkəbliyin müxtəlif aspektlərini başa düşmək və idarə etməkdə kömək edir. Bundan əlavə, bu tədqiqatlar proqram təminatının hazırlanması prosesinin nəzəri əsaslarını gücləndirir və proqram mühəndisliyi sahəsində irəliləyişlərə töhfə verir.

Müdafiə üçün təqdim edilən nəticələr. Ölçmə üsulları və texnikalar, onların üstünlük və çatışmazlıqları, mürəkkəbliyin proqram təminatına təsirləri və azaldılması üsulları.

Nəticələrin approbasiyası. Yekun olaraq, tədqiqat proqram kodlarının mürəkkəbliyini ölçmək üçün istifadə olunan üsulların vacibliyi təsdiqlədi. Mürəkkəbliyin ölçülməsi keyfiyyətin yaxşılaşdırılması, xərclərin azaldılması və proqram təminatının hazırlanması prosesində nəzəri əsasların möhkəmləndirilməsi üçün mühüm vasitədir. Buna görə də, mürəkkəbliyin ölçülməsi sahəsini daha da inkişaf etdirmək üçün perspektiv tədqiqatlara ehtiyac var.

Nəşrlər. Dissertasiya müddətində müəlliflər tərəfindən aşağıdakı elmi məqalə nəşr edilmişdir

Qarayev, İ., Qüdrətov, O,. İsrafilli, İ,. (2024). Kod mürəkkəbliyi və onun təsirləri. Azərbaycan xalqının ümumilli lider, görkəmli dövlət xadimi Heydər Əliyevin anadan olmasının 101 illiyinə həsr olunmuş tələbə və gənc tədqiqatçıların "Mütərəqqi texnologiyalar və innovasiyalar" mövzusunda IX Respublika elmi-texniki konfransı.

I FƏSİL. PROQRAM TƏMİNATININ KEYFİYYƏTİNİN ƏSAS ELEMENTLƏRİ

1.1. Proqram təminatının keyfiyyəti nədir

Proqram təminatı rəqəmsal dünyamızı gücləndirən görünməz qüvvədir. Bu, kompüterlərimizin, smartfonlarımızın, planşetlərimizin və bir sıra digər elektron cihazların ruhudur. Əslində bu, toxuna bildiyimiz və görə bildiyimiz fiziki komponentlər olan aparatın görünməz hissəsidir. Proqram təminatı kompüterə nə edəcəyini bildirən proqramlaşdırma dilində yazılmış bir sıra təlimatlardan ibarətdir. Bu təlimatlar kompüterin məlumatları necə emal etdiyini, hesablamaları yerinə yetirdiyini, istifadəçilər və digər qurğularla qarşılıqlı əlaqəsini özündə birləşdirir. O, sistemin aparat komponentləri ilə istifadəçi arasında vasitəçi rolunu oynayır, qarşılıqlı əlaqəni asanlaşdırır və hesablama resurslarından istifadə etməyə imkan verir. Müasir bir-biri ilə əlaqəli dünyada proqram təminatı həyatımızın demək olar ki, bütün aspektlərində mühüm rol oynayır, ünsiyyət və əyləncədən tutmuş biznes və sənayeyə qədər. Texnologiya irəliləməyə davam etdikcə, proqram təminatı inkişaf etməyə davam edəcək, innovasiyalara təkan verəcək və rəqəmsal dünyamızın gələcəyini formalaşdıracaq.

Keyfiyyətli proqram təminatı bir neçə əsas atributla xarakterizə olunur:

1. **Funksionallıq:** Keyfiyyətli proqram təminatı nəzərdə tutulduğu funksiyaları və tapşırıqları effektiv şəkildə yerinə yetirir. Müəyyən edilmiş tələblərə cavab verir, nəzərdə tutulan xüsusiyyət və imkanları təqdim edir.

Aşağı funksionallıq proqramın keyfiyyətinə necə təsir edir:

- İstifadəçinin narazılığı: Məhdud funksionallığı olan proqram təminatı istifadəçilərə tapşırıqlarını yerinə yetirmək və ya problemlərini həll etmək üçün lazım olan xüsusiyyət və imkanları təmin edə bilmir. Bu, məyusluğa, narazılığa və nəticədə zəif istifadəçi təcrübəsinə səbəb ola bilər;
- Artan dəstək yükü: İstifadəçiləri məhdudiyyətləri və ya həll yollarını aradan qaldırmaq üçün İT və ya müştəri xidməti qruplarından daha çox dəstək və yardım tələb edə bilər. Bu, dəstək xərclərini artıra və resursları gərginləşdirə bilər;

- Zərər riski: Bütün bunlar müştəri itkisinə və proqram təminatçısı üçün gəlir itkisinə səbəb olur.

1. **Etibarlılıq:** Keyfiyyətli proqram təminatı ardıcıl və proqnozlaşdırıla bilən şəkildə işləyir, dəqiq nəticələr verir və gözlənilməz uğursuzluqlar və ya səhvlər olmadan zamanla sabitliyi qoruyur.

Aşağı etibarlılığın proqram təminatının keyfiyyətinə necə təsir edir:

- Məhsuldarlığın azalması: Etibarsız proqram istifadəçilərin məhsuldarlığını pozur, fasilələrə, gecikmələrə və məlumat itkisinə səbəb olur. İstifadəçilər vaxt və səy sərf edərək tapşırıqları təkrarlamalı, proqramı yenidən başlatmalı və ya alternativ həll yolları axtarmalı ola bilər;
- Məlumatın itkisi və ya korrupsiya riski: Proqram təminatının nasazlığı məlumat itkisi, korrupsiya və ya digər təhlükəsizlik zəiflikləri ilə nəticələnə bilər.

Ümumən zəif etibarlılıq istifadəçi məyusluğuna səbəb olmaqla, məhsuldarlığı aşağı salmaqla, etibarı sarsıdaraq, reputasiyaya xələl gətirərək, dəstək yükünü artıraraq, təhlükəsizlik riskləri yaradaraq və texniki xidmət xərclərini artıraraq proqram təminatının keyfiyyətini aşağı salır.

2. **Performans:** Keyfiyyətli proqram təminatı yüksək iş yükü şəraitində belə, sürətli cavab müddəti, minimal gecikmə və optimal resurs istifadəsi ilə səmərəli şəkildə işləyir.
3. **Təhlükəsizlik:** Keyfiyyətli proqram təminatı təhlükəsizdir, məlumatların və resursların məxfiliyini, bütövlüyünü və mövcudluğunu təmin etmək üçün icazəsiz girişdən, məlumatların pozulmasından və təhlükəsizlik zəifliklərindən qoruyur.

Keyfiyyətli proqram təminatı istifadəçi ehtiyaclarını ödəmək, etibarlı şəkildə yerinə yetirmək, müsbət istifadəçi təcrübəsini təmin etmək, təhlükəsizlik və bütövlüyü qorumaq, texniki xidmət və yeniləmələri asanlaşdırmaq, effektiv miqyas vermək və hərtərəfli sınaqdan və yoxlamadan keçmək qabiliyyəti ilə xarakterizə olunur.

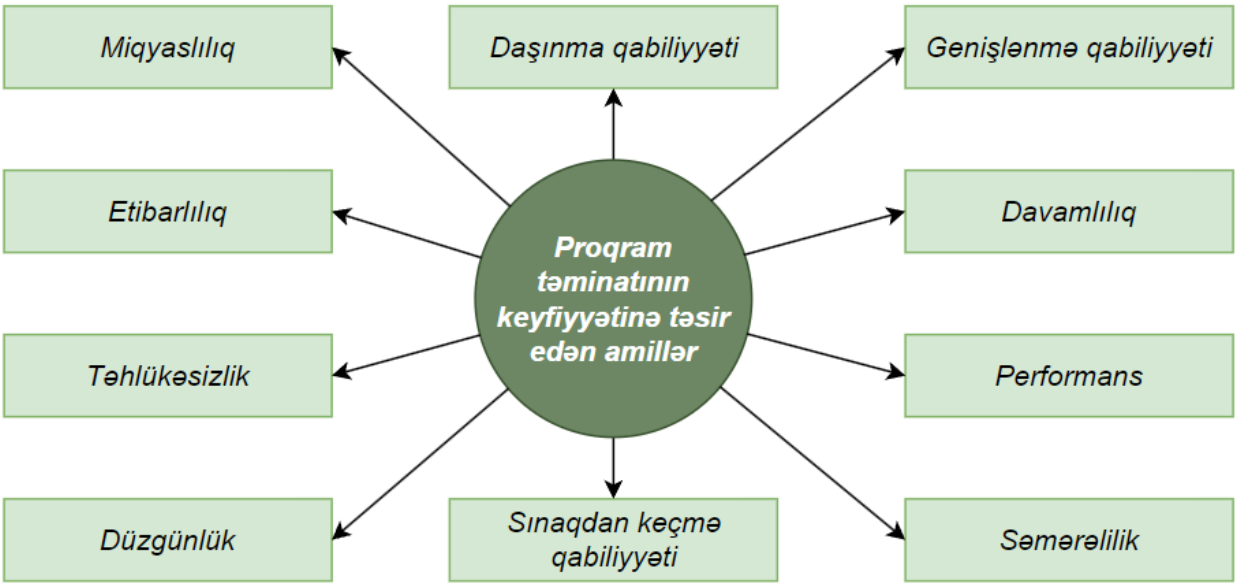
1.2. Proqram təminatının keyfiyyətinin tərifi

Proqram təminatının keyfiyyəti proqram məhsulunun arzu olunan xüsusiyyətlərini, onun bu xüsusiyyətlərə nə dərəcədə uyğun olduğunu və bu xüsusiyyətləri əldə etmək üçün edilən prosesləri, alətləri və texnikaları ifadə edir. Bununla belə, bu termin zamanla müxtəlif müəlliflər və təşkilatlar tərəfindən müxtəlif şəkildə müəyyən edilmişdir. Məsələn, Phil Crosby-ə görə keyfiyyət "tələblərə uyğunluq" (Crosby, 1979), Watts Humphrey-ə görə isə "mükəmməl səviyyədə istifadəyə uyğunluğu əldə etmək" deməkdir (Humphrey, 1989). Bu müxtəlif təriflər proqram keyfiyyətinin mürəkkəb bir anlayış olduğunu göstərir.

Daha yeni təriflərdə proqram keyfiyyəti proqram məhsulunun müəyyən şərtlər altında bəyan edilmiş və nəzərdə tutulan ehtiyacları ödəmək qabiliyyəti kimi ifadə edilir (ISO/IEC, 25019:2023). O, həmçinin proqram məhsulunun müəyyən edilmiş tələblərə nə dərəcədə cavab verdiyinə əsaslanan dərəcə kimi müəyyən edilir. Bununla belə, bu keyfiyyət göstərilən tələblərin maraqlı tərəflərin ehtiyaclarını, istəklərini və gözləntilərini dəqiq əks etdirib- əks etdirməməsindən asılıdır (ISO/IEC, 25019:2023). Hər iki tərif keyfiyyətin tələblərə uyğunluğun qayğısına qaldığını vurğulayır. Bununla belə, onlar nə funksional, nə də etibarlılıq, performans, etibarlılıq və ya digər xüsusiyyətlər kimi müəyyən edilmiş tələblərin digər növlərinə istinad etmirlər. Əhəmiyyətli məqam ondan ibarətdir ki, bu təriflər keyfiyyətin tələblərdən asılı olduğunu açıq şəkildə göstərir.

1.3. Proqram təminatının keyfiyyətinə təsir edən amillər

Proqram təminatının keyfiyyəti, onun hazırlanması prosesinin hər mərhələsində vacibdir. Çünki, proqram təminatının keyfiyyəti məhsulun nə qədər yaxşı və etibarlı olduğunu göstərir. Bu keyfiyətə təsir edən amillər kifayət qədər müxtəlifdir. Şəkil 1.1-də proqram təminatının keyfiyyətinə təsir edən əsas amillər göstərilmişdir.



Şək. 1.1 Proqram təminatının keyfiyyətinə təsir edən bir sıra amillər

Genişləmə qabiliyyəti – proqram təminatının keyfiyyətinə təsir edən ən zəruri amildir. Genişlənmə qabiliyyəti, proqram təminatına zərər vermədən ona müxətlif yeni funksiyaları əlavə etmək qabiliyyətidir.

Davamlılıq – proqram təminatının funksiyalarında müəyyən dəyişikliklər etmək və ya kiçik düzəlişlər etməklə sistemin dəyişdirilməsinə imkan verən xüsusiyyətdir.

Performans – proqram təminatına təsir edən ən əsas amillərdən biridir. İstifadəçilər maksimum dərəcədə az xərclə yüksək performanslı proqram təminatından istifadə edə bilməlidirlər.

Səmərəlilik – Proqram təminatının keyfiyyətinə böyük təsir göstərən mühüm amildir. Səmərəlilik istifadəçilərə istədikləri işləri tez bir zamanda yerinə yetirməyə kömək edən resurslardan istifadə deməkdir.

Miqyaslılıq – sistemin yükü artdıqca əməliyyatlara cavab vermə qabiliyyətidir. Miqyaslılıq dərəcəsinin zəif olması proqram təminatına olduqca təsir edir.

Etibarlılıq – etibarlılıq o deməkdir ki, sistem yüksək yük şəraitində belə davamlıdır. Etibarsız sistemin gələcək inkişaf ehtimalı yoxdur, çünki proqram təminatının keyfiyyəti aşağı olduğu üçün daha az istifadəçi cəlb edir.

Təhlükəsizlik – sistemin proqram təminatının keyfiyyətinə təsir edən mühüm xüsusiyyətdir, çünki hücumlar və təhlükəsizlik boşluqları istifadəçilərin etibarını sarsıda və sistemin işinə mənfi təsir göstərə bilər.

Düzgünlük – proqram təminatının funksionallığını və etibarlılığını müəyyən edir. Səhvləri asanlıqla və tez bir zamanda düzəltmək istifadəçi təcrübəsini yaxşılaşdırır və proqram təminatına nəzərdə tutulan vəzifələri düzgün yerinə yetirməyə imkan verir. Bu, proqram təminatının keyfiyyətini və istifadəsini artırır.

Sınaq qabiliyyəti – Yüksək keyfiyyətli proqram testlərdə daha yaxşı nəticə göstərən proqramdır. Buna görə də test proqram təminatının keyfiyyətinə təsir edən zəruri elementdir.

Daşınma qabiliyyəti – Proqram təminatı uyğunlaşdırıla bilən və müxtəlif platformalarda işləyə bilməlidir. Daşınma qabiliyyəti tez-tez çevikliklə əlaqələndirilir və istifadəçilərin ehtiyaclarına uyğun olaraq funksiyaları yerinə yetirməlidir.

1.4. Proqram təminatının keyfiyyət modelləri

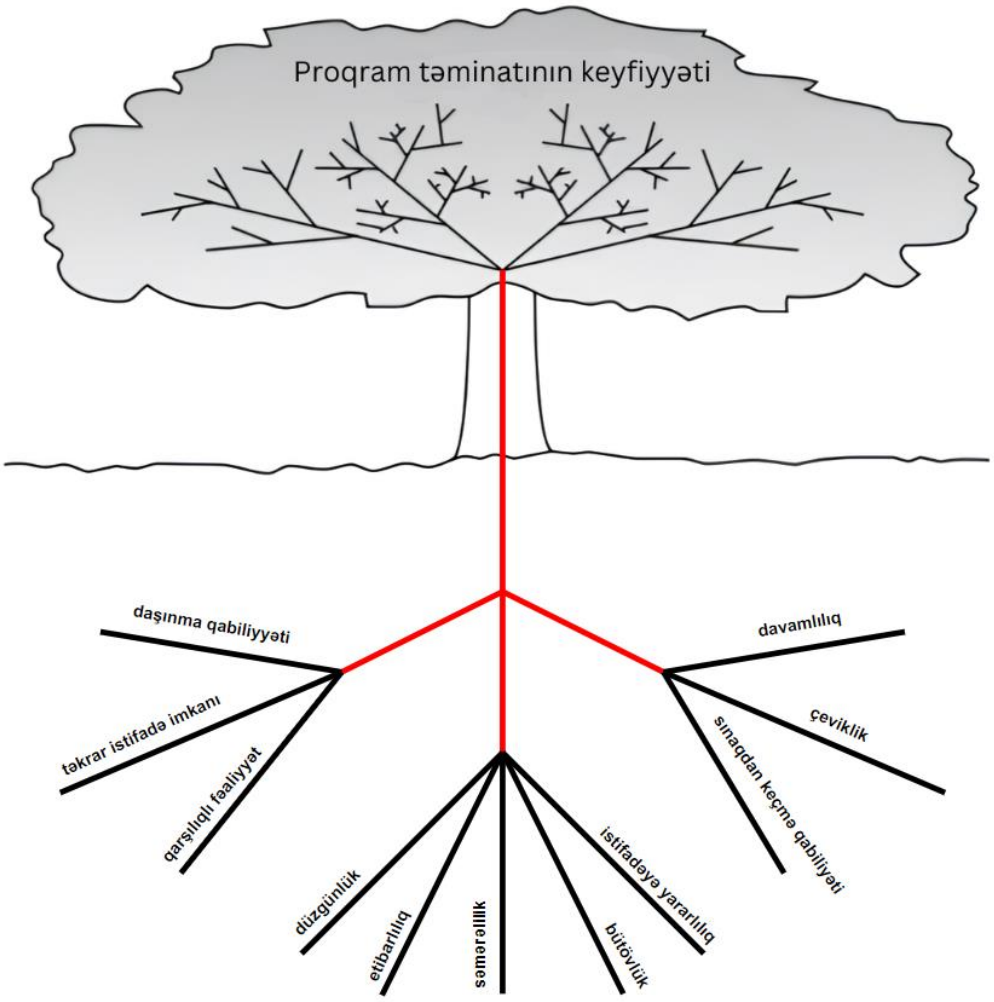
Proqram təminatının keyfiyyət modeli, proqram keyfiyyətinin xüsusiyyətlərini, atributlarını və onların necə əlaqəli olduğunu müəyyən edən freymvörkdür. Fərqli modellərin proqram təminatının keyfiyyətini qiymətləndirmək üçün fərqli perspektivləri, məqsədləri və meyarları ola bilər. Məsələn, bəzi modellər kodun keyfiyyəti, mürəkkəbliyi və modulluğu kimi proqram təminatının daxili aspektlərinə diqqət yetirir, digərləri isə istifadəçi məmnunluğu, funksionallıq və performans kimi xarici aspektlərə diqqət yetirə bilər.

Proqram təminatının keyfiyyəti müxtəlif modellər və standartlar vasitəsilə qiymətləndirilə bilər. Bu modellər proqram təminatının hazırlanması prosesi boyunca keyfiyyəti yaxşılaşdırmaq və ölçmək üçün istifadə olunur. Keyfiyyəti başa düşmək və ölçmək üçün tədqiqatçılar tez-tez keyfiyyət atributlarının bir-biri ilə necə əlaqəli olduğuna dair modellər yaratmışlar. Bu modellər proqram təminatının

hazırlanması prosesi boyunca keyfiyyəti yaxşılaşdırmaq və ölçmək üçün istifadə olunur.

Bu modellərdən biri də McCall keyfiyyət modelidir. McCall keyfiyyət modeli Jim McCall tərəfindən təqdim edilmiş keyfiyyət modelidir. Bu model ilk növbədə sistem tərtibatçıları və sistemin inkiçaf prosesi üçün nəzərdə tutulub. McCall keyfiyyət modeli həm istifadəçilərin fikirlərini, həm də proqramçıların prioritetlərini əks etdirən bir sıra proqram təminatı keyfiyyəti amillərinə diqqət yetirir, istifadəçilər və proqramçılar arasında fərqi aradan qaldırmağa çalışır (McCall, Richards, & Walters, 1977).

McCall keyfiyyət modeli Şəkil 1.2-də göstərildiyi kimi proqram məhsulunun keyfiyyətini müəyyən etmək üçün üç əsas perspektivə malikdir.



Şək. 1.2 McCall keyfiyyət modeli

Məhsulun yenidən nəzərdən keçirilməsi: Məhsulun dəyişikliklərə məruz qalma qabiliyyətinə aiddir və aşağıdakıları əhatə edir:

- Davamlılıq: Proqramdakı qüsurları tapmaq və düzəltmək üçün lazım olan səy;
- Çeviklik: İş mühiitində dəyişikliklər tələb edildiyi zaman onların yerinə yetirilməsinin asanlığı;
- Sınaq qabiliyyəti: proqramın xətasız olmasını və gözlənilən spesifikasiyalara cavab verməsini təmin etmək üçün onun sınaqdan çevirilməsinin asanlığı.

Məhsul keçidi: Məhsulun yeni mühitlərə uyğunlaşması ilə bağlıdır aiddir və aşağıdakıları əhatə edir:

- Portativlik: Proqramı bir mühitdən digərinə köçürmək üçün tələb olunan səy;
- Yenidən istifadə edilə bilənlik: Proqram təminatının fərqli kontekstdə təkrar istifadəsinin asanlığı;
- Birlikdə işləmə qabiliyyəti: Sistemi başqa bir sistemə birləşdirmək üçün tələb olunan səy.

Məhsulun əməliyyat perspektivi: Proqram təminatının istifadəçilər tərəfindən necə qəbul edildiyini və istifadə olunduğunu qiymətləndirir. Proqram təminatının əməliyyat keyfiyyəti aşağıdakılardan asılıdır:

- Dəqiqlik: Proqramın bütün spesifikasiyalara cavab vermə dərəcəsi olub, nəzərdə tutulmuş əməliyyatları yerinə yetirmək qabiliyyətidir;
- Etibarlılıq: Sistemin uğursuzluğa düçar olmamaq qabiliyyəti;
- Səmərəlilik: Proqram təminatının öz resurslarından, xüsusilə prosessor vaxtı və yaddaş kimi resurslardan nə dərəcədə səmərəli istifadə etməsi ilə bağlıdır;
- Dürüstlük: Proqram təminatının təhlükəsizliyinə aiddir, icazəsiz girişlərin qarşısını almağa kömək edir;
- Yararlılıq: Proqram təminatından istifadənin asanlığı.

McCall keyfiyyət modeli, proqram təminatının hazırlanması prosesində mühüm rol oynayır. İstifadəçilərin ehtiyaclarını ödəmək və proqram təminatının keyfiyyətini artırmaq üçün hazırlanmış bu model, dəqiqlik, etibarlılıq, səmərəlilik və digər amillərə diqqət yetirərək proqram təmiantının ümumi performansını

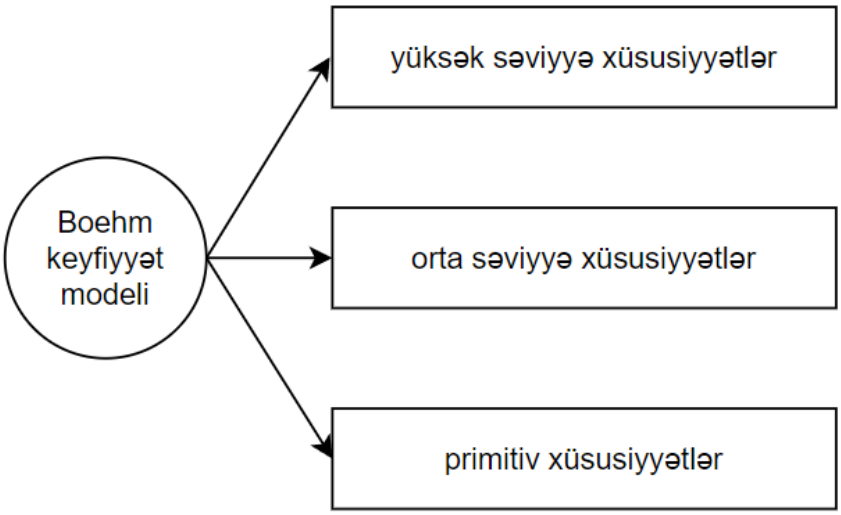
qiymətləndirir. McCall modeli, proqramçılara istifadəçi mərkəzli bir yanaşma tətbiq etməyə istiqamət verərək proqram təminatı layihələrinin uğurla başa çatdırılmasına kömək edir.

Digər mühüm keyfiyyət modellərindən biri də Boehm keyfiyyət modelidir. Boehm keyfiyyət modeli proqram təminatının keyfiyyətini qiymətləndirmək üçün istifadə edilən bir modeldir. Bu model 1978-ci ildə Barry W. Boehm tərəfindən hazırlanmışdır (Boehm, 1978). Əsasən, proqram təminatının keyfiyyətini müəyyən etmək üçün müxtəlif xüsusiyyətlərdən və ölçülərdən istifadə edir.

Boehm keyfiyyət modeli proqram təminatının keyfiyyətini keyfiyyət və kəmiyyət baxımından qiymətləndirmək məqsədi daşıyır. Proqram təminatının etibarlılığı, performansı, istifadəsi, texniki xidmətin asanlığı kimi müxtəlif xüsusiyyətlərə diqqət yetirir. Bu xüsusiyyətlərə əlavə olaraq, proqram təminatının inkişaf prosesində idarəetmə və təşkilati amilləri də nəzərə alır.

Modelin əsas məqsədi proqram təminatının hazırlanması prosesində keyfiyyətin artırılması və proqram təminatı layihələrinin uğurla başa çatdırılmasını təmin etməkdir. Bu məqsədlə proqram təminatının müxtəlif mərhələlərində keyfiyyət meyarları müəyyən edilir və daim nəzarətdə saxlanılır.

Boehm keyfiyyət modeli Şəkil 1.3-də görüldüyü kimi 3 səviyyədən ibarətdir.



Şək. 1.3 Bohem keyfiyyət modelinin strukturu

1. **Yüksək səviyyə xüsusiyyətlər**

Bu səviyyə proqram təminatının ümumi istifadə imkanlarını və funkionallığını əks etdirir. Burada üç əsas suala diqqət yetirilir:

- Mövcud istifadə: Proqram təminatı nə dərəcədə yaxşı istifadə edilə bilər;
- Davamlılıq: Proqram təminatı nə dərəcədə asanlıqla başa düşülə, dəyişdirilə və yenideən sınaqdan keçirilə bilər;
- Daşınma qabiliyyəti: Mühiti dəyişdikdə proqram təminatını hələdə istifadə etmək mümkündürmü.

2. Orta səviyyə xüsuiyyətlər

Bu səviyyə Boehm tərəfindən müəyyən edilmiş 7 keyfiyyət amilini təmsil edir. Bu amillər prqoram sisteminin gözlənilən keyfiyyətlərini təmsil edir: Etibarlılıq, məhsuldarlıq, səmərəlilik, sınaq qabiliyyəti, anlaşıqlılıq, çeviklik və daşıma qabiliyyəti.

3. Primitiv xüsusiyyətlər

Bu səviyyə proqram təminatının keyfiyyətini ölçmək üçün əsas ölçüləri təmin edir. Boehm öz keyfiyyət modelini hazırlayarkən, qarşıya qoyduğu məqsədlərdən biri də bu primitiv xüsusiyyətlərlə bağlı keyfiyyət ölçülərini müəyyən etmək idi. Bu ölçülər proqram təminatının keyfiyyətinin qiymətləndirilməsində mühüm rol oynayır.

Yüksək, orta və primitiv səviyyələrdə olan xüsusiyyətlər proqram təminatının müxtəlif aspektlərinin hərtərəfli qiymətləndirilməsinə imkan verir. Boehm keyfiyyət modeli proqram təminatı layihələrinin idarə edilməsində və onların keyfiyyətinin qiymətləndirilməsində mühüm rol oynayır, ona görə də proqram təminatının hazırlanması prosesində maraqlı tərəflər üçün dəyərli mənbədir.

Proqram təminatının keyfiyyət modellərindən biri də ISO/IEC 25010-da müəyyən edilmiş məhsulun keyfiyyət modelidir. Bu model aşağıdakı şəkildə göstərilən doqquz keyfiyyət xarakteristikasından ibarətdir:

Cədvəl 1.4

ISO/IEC 25010-da müəyyən edilmiş məhsulun keyfiyyət modeli

PROQRAM MƏHSULUNUN KEYFİYYƏTİ								
FUNKSİONAL UYĞUNLUQ	**PERFORMANS SƏMƏRƏLİLİYİ**	**UYĞUNLUQ**	**QARŞILIQLI ƏLAQƏ QABİLİYYƏTİ**	**ETİBARLILIQ**	**TƏHLÜKƏSİZLİK**	**DAVAMLILIQ**	**ÇEVİKLİK**	**ƏMNİYYƏT**
FUNKSİONAL TAMLIQ	ZAMAN DAVRANIŞI	BİRGƏ MÖVCUDLUQ	UYĞUNLUĞUN TANINMASI	QÜSURSUZLUQ	MƏXFİLİK	MODULLUQ	UYĞUNLAŞMA QABİLİYYƏTİ	ƏMƏLİYYAT MƏHDUDİYYƏTİ
FUNKSİONAL DÜZGÜNLÜK	RESURSDAN İSTİFADƏ	QARŞILIQLI FƏALİYYƏT	ÖYRƏNMƏ QABİLİYYƏTİ	MÖVCUDLUQ	BÜTÖVLÜK	TƏKRAR İSTİFADƏ İMKANI	MİQYASLILIQ	RİSKİN İDENTİFİKASİYASI
FUNKSİONAL ƏLÇATANLIQ	TUTUM		OPERATİVLİK	SƏHVLƏRƏ DÖZÜMLÜLÜK	DANILMAZLIQ	TƏHLİL ETMƏK QABİLİYYƏTİ	QURAŞDIRMA QABİLİYYƏTİ	GÜVƏNLİ MOD
			İSTİFADƏÇİ XƏTALARINDAN QORUNMA	BƏRPA QABİLİYYƏTİ	CAVABDEHLİK	DƏYİŞDİRİLƏ BİLƏNLİK	ƏVƏZ EDƏ BİLƏNLİK	TƏHLÜKƏ XƏBƏRDARLIĞI
			İSTİFADƏÇİ CƏLB EDİLMƏSİ		HƏQİQİLİK	SINAQDAN KEÇMƏ QABİLİYYƏTİ		TƏHLÜKƏSİZ İNTEQRASİYA
			İNKLÜZİVLİK		MÜQAVİMƏT			
			İSTİFADƏÇİ YARDIMI					
			ÖZÜNÜ TƏSVİR EMƏK QABİLİYYƏTİ					

Mənbə: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Funksional uyğunluq. Bu xüsusiyyət, məhsul və ya sistemin müəyyən şərtlərdə istifadə edildikdə ehtiyacları nə qədər qarşıladığını göstərir. Bu xüsusiyyət aşağıdakı alt xüsusiyyətlərdən ibarətdir:

- Funksional bütövlük - bütün işləri nə qədər yaxşı yerinə yetirdiyini göstərir;
- Funksional dəqiqlik - istifadəçilərə istədikləri nəticələri nə qədər dəqiq verdiyini göstərir;
- Funksional uyğunluq - tapşırıqları və məqsədləri nə qədər yaxşı yerinə yetirirdiyini göstərir.

Performans səmərəliliyi. Bu xüsusiyyət məhsulun müəyyən vaxt və səmərəlilik parametrləri daxilində öz funksiyalarını nə dərəcədə yaxşı yerinə yetirdiyini və resursların müəyyən şərtlərdə necə səmərəli istifadə edildiyini əks etdirir. Bu xüsusiyyət aşağıdakı alt xüsusiyyətlərdən ibarətdir:

- Zaman davranışı - məhsulun nə qədər sürətli çalışdığını göstərir;
- Resurs İstifadəsi - məhsulun nə qədər az resurs istifadə etdiyini göstərir;
- Tutum - məhsulun nə qədər böyük işlərin öhdəsindən gələ bildiyini göstərir.

Uyğunluq. Məhsulun, sistemin və ya komponentin digər məhsullar, sistemlər və ya komponentlərlə məlumat mübadiləsi edə bilmə dərəcəsi, eyni mühiti və resursları paylaşarkən tələb olunan funksiyaları yerinə yetirmək qabiliyyəti. Bu xüsusiyyət aşağıdakı alt xüsusiyyətlərdən ibarətdir:

- Birgə mövcudluq - məhsulun digər məhsullara zərər vermədən eyni mühiti və resursları digər məhsullarla paylaşaraq tələb olunan funksiyaları effektiv şəkildə yerinə yetirə bilmə dərəcəsi;
- Qarşılıqlı işləmə qabiliyyəti - sistemin, məhsulun və ya komponentin digər məhsullarla məlumat mübadiləsi və mübadilə edilən məlumatdan qarşılıqlı şəkildə istifadə edə bilmə dərəcəsi.

Qarşılıqlı əlaqə qabiliyyəti. Müəyyən istifadəçilərin məhsul və ya sistemdən istifadə edərək müxtəlif vəziyyətlərdə müəyyən tapşırıqları yerinə yetirə bilmə dərəcəsi. Bu xüsusiyyət aşağıdakı alt xüsusiyyətlərdən ibarətdir:

- Uyğunluğun tanınması - istifadəçilərin məhsul və ya sistemin ehtiyaclarına uyğun olub-olmadığını tanıya bilmə dərəcəsi;
- Öyrənilmə qabiliyyəti - məhsulun və ya sistemin funksiyalarının müəyyən bir müddət ərzində müəyyən istifadəçilər tərəfindən öyrənilə bilmə dərəcəsidir;
- Operativlik - məhsulun və ya sistemin istifadə və idarə edilməsini asanlaşdıran xüsusiyyətlərə malik olduğunu göstərir;
- İstifadəçi səhvlərindən qorunma - sistemin istifadəçiləri əməliyyat xətalarından qoruma dərəcəsi;
- İstifadəçinin cəlb edilməsi - bir interfeysin , istifadəçiləri cəlb edən və həvəsləndirən şəkildə funksiyaları və məlumatları təqdim etmə dərəcəsi;
- İnklüzivlik - məhsul və ya sistemin müxtəlif mənşəli insanlar (məsələn, müxtəlif yaş, qabiliyyət, mədəniyyət, etnik mənsubiyyət, dil, cins, iqtisadi vəziyyət və s.) insanlar tərəfindən istifadə oluna bilmə dərəcəsi;
- İstifadəçi yardımı - məhsulun konkret istifadə kontekstində konkret məqsədlərə nail olmaq üçün ən geniş xüsusiyyətlərə və imkanlara malik insanlar tərəfindən istifadə oluna bilmə dərəcəsi;
- Özünü təsvir etmək qabiliyyəti - məhsulun istifadəçiyə lazım olan müvafiq məlumatı təqdim etməklə, imkanlarını və istifadəsini dərhal anlamağa köməklik dərəcəsi.

Etibarlılıq. Məhsulun və ya sistemin, xüsusi funksiyaları müəyyən müddət və müəyyən şərtlər daxilində yerinə yetirmə dərəcəsi. Bu xüsusiyyət aşağıdakı alt xüsusiyyətlərdən ibarətdir:

- Qüsursuzluq - normal şərtlərdə müəyyən edilmiş funksiyaları səhvsiz yerinə yetirmə dərəcəsi;
- Mövcudluq - lazım olduqda sistem və ya məhsulun istifadəyə hazır və əlçatan olma dərəcəsi;
- Səhvlərə dözümlülük - aparat və ya proqram təminatı xətalarının mövcudluğuna baxmayaraq sistemin nəzərdə tutulduğu kimi işləmə dərəcəsi;
- Bərpa qabiliyyəti - nazaslıq və ya çökmə hallarında, zərər görmüş məlumatların bərpa və sistemin istənilən vəziyyətə qaytarılması.

Təhlükəsizlik. Məhsul və ya sistemin hakerlər tərəfindən hücum cəhdlərinə qarşı qorunma dərəcəsi və məlumatın fərdlər və ya digər sistemlər üçün müvafiq şəkildə əlçatan olma dərəcəsi. Bu xüsusiyyət aşağıdakı alt xüsusiyyətlərdən ibarətdir:

- Məxfilik - məlumatlara sadəcə səlahiyyətli şəxslərin əldə etməsi;
- Bütövlük - sistemin və ya məlumatların zərərli dəyişikliklərdən qorunması;
- Danılmazlıq - hərəkətlərin və ya hadisələrin sonradan inkar edilə bilinməməsi üçün baş verdiyinin sübut olunma dərəcəsi;
- Cavabdehlik - müəssisənin hərəkətlərinin həmin qurum tərəfindən izlənilə bilmə dərəcəsi;
- Həqiqilik - şəxsiyyətin sübut olunma dərəcəsi;
- Müqavimət - hücumu altında olarkən məhsul və ya sistemin öz əməliyyatlarını saxlama dərəcəsi;

Davamlılıq. Bu xüsusiyyət məhsulun və ya sistemin ətraf mühitdəki dəyişikliklərə nə qədər asan və səmərəli şəkildə dəyişdirilə biləcəyini, düzəldildiyini və ya uyğunlaşdırıla biləcəyini ifadə edir. Bu xüsusiyyət aşağıdakı alt xüsusiyyətlərdən ibarətdir:

- Modulluq - sistem və ya proqramın, bir komoponentdə edilən dəyişikliyin digər komponentlər üzərində minimum təsirə sahip olacağı şəkildə ayrı modullaradan ibarət olma dərəcəsi;
- Təkrar istifadə imkanı - məhsulun birdən çox sistemdə istifadə oluna bilmə dərəcəsi;
- Təhlik etmək qabiliyyəti - dəyişikliyin məhsula və ya sistemə təsirini qiymətləndirmək, məhsuldakı çatışmazlıqları və ya nasazlıqların səbəblərini müəyyən etmənin mümkün olan effektivlik və səmərəlilik dərəcəsi;
- Dəyişdirilə bilənlik - məhsulun və ya sistemin keyfiyyətini aşağı salmadan effektiv və səmərəli şəkildə dəyişdirilə bilmə dərəcəsi;
- Sınaqdan keçmə qabiliyyəti - testlərin həyata keçirilə biləcəyi effektivlik və səmərəlilik dərəcəsi.

Çeviklik. Məhsulun dəyişikliklərə uyğunlaşdırıla bilmə dərəcəsi. Bu xüsusiyyət aşağıdakı alt xüsusiyyətlərdən ibarətdir:

- Uyğunlaşma qabiliyyəti - məhsulun müxtəlif aparat, proqram təminatı və ya istifadə mühitlərinə nə qədər asan və səmərəli uyğunlaşa bilmə dərəcəsi;
- Miqyaslılıq - məhsulun artan və ya azalan iş yüklərinə tab gətirə bilməsi və ya dəyişkənliyin öhdəsindən gəlmək üçün öz qabiliyyətini uyğunlaşdıra bilmə dərəcəsi;
- Quraşdırma qabiliyyəti - məhsul və ya sistemin müəyyən bir mühitdə uğurla quraşdırıla və ya çıxarıla biləcəyi effektivlik və səmərəlilik dərəcəsi;
- Əvəz edilə bilənlik - məhsulun eyni mühitdə eyni məqsəd üçün başqa bir xüsusi proqram məhsulunu əvəz edə bilmə dərəcəsi.

Əmniyyət. Bu xüsusiyyət məhsulun müəyyən şərtlər altında insan həyatına, sağlamlığına, əmlakına və ya ətraf mühitə təhlükə yaratmadan vəziyyətin qarşısını almaq dərəcəsini əks etdirir. Bu xüsusiyyət aşağıdakı alt xüsusiyyətlərdən ibarətdir:

- Əməliyyat məhdudiyyəti - əməliyyat təhlükəsi ilə qarşılaşdıqda məhsulun və ya sistemin təhlükəsiz parametrlər və ya vəziyyətlər daxilində işləməsinin məhdudlaşdırılma dərəcəsi;

- Riskin identifikasiyası - məhsulun həyatı, əmlakı və ya ətraf mühiti qəbuledilməz riskə məruz qoya biləcək hadisə və ya prosesi müəyyən edə bilmə dərəcəsi;
- Güvənli mod - məhsulun nasazlıq halında avtomatik olaraq özünü təhlükəsiz iş rejiminə qoya bilməsi və ya təhlükəsiz vəziyyətə qayıtması dərəcəsi;
- Təhlükə xəbərdarlığı - məhsul və ya sistemin əməliyyatları davam etdirməsi üçün kifayət qədər vaxt ərzində cavab verə bilmələri məqsədilə əməliyyatlara və ya daxili nəzarətlərə qəbuledilməz risklər barədə xəbərdarlıq etmə dərəcəsi;
- Təhlükəsiz inteqrasiya - məhsulun bir və ya bir neçə komponentlə inteqrasiyası zamanı təhlükəsizliyini qoruya bilmə dərəcəsi.

1.5. Proqram təminatının keyfiyyətinin monitorinqi

Proqram təminatının keyfiyyətinin monitorinqi proqram təminatı inkişafının həyat dövrü ərzində müxtəlif keyfiyyət atributlarının və ölçülərinin davamlı müşahidəsini, ölçülməsini və qiymətləndirilməsini əhatə edir. O, maraqlı tərəflərə proqram məhsulunun keyfiyyəti ilə bağlı anlayışlar təqdim etmək, potensial problemləri və ya təkmilləşdirilməli sahələri müəyyən etmək və məlumatlara əsaslanan qərarların qəbulunu dəstəkləmək məqsədi daşıyır.

Proqram təminatının keyfiyyətinə nəzarətin icmalı:

- Əsas keyfiyyət ölçmələri: Proqram təminatının keyfiyyətinin monitorinqi funksionallıq, etibarlılıq, istifadə imkanları, performans, təhlükəsizlik, davamlılıq və digər müvafiq atributlarla bağlı əsas keyfiyyət göstəricilərinin izlənilməsini və təhlilini əhatə edir. Bu ölçülərə qüsur sıxlığı, kod dəyişikliyi, test əhatə dairəsi, cavab müddətləri, səhv dərəcələri və müştəri məmnuniyyəti balları daxil ola bilər;
- Testlərin avtomatlaşdırılması: Testlərin avtomatlaşdırılması proqram təminatının funksionallığını, performansını və etibarlılığını yoxlamaq üçün avtomatlaşdırılmış testlərin icrasına imkan verməklə proqram təminatının

keyfiyyətinin monitorinqində mühüm rol oynayır. Xətaları tez bir zamanda aşkar etmək üçün vahid testləri, inteqrasiya testləri və s. testlər avtomatlaşdırılmış sınaq çərçivələri müntəzəm olaraq həyata keçirilir;

- Qüsurların izlənməsi və idarə edilməsi: Qüsurların izlənilməsi sistemləri test zamanı aşkar edilmiş qüsurları qeyd etmək, izləmək və idarə etmək üçün istifadə olunur. Bu sistemlər qüsurlar haqqında məlumatları, o cümlədən onların ciddiliyi, statusu və həlli ilə bağlı məlumatları toplayır, keyfiyyət məsələlərini dərhal həll etmək üçün komanda daxilində əməkdaşlığı asanlaşdırır;
- Performans monitorinqi: Performans monitorinqi cavab müddəti, məhsuldarlıq və resurs istifadəsi kimi əsas performans göstəricilərini izləyir və təhlil edir. Performans monitorinqi, sistem performansını optimallaşdırmağa və proqram təminatının performans tələblərinə cavab verməsini təmin etməyə kömək edir;
- Uyğunluq və təhlükəsizlik monitorinqi: Proqram təminatının keyfiyyətinin monitorinqi həmçinin sənaye standartlarına, qaydalara və təhlükəsizlik üzrə ən yaxşı təcrübələrə uyğunluğun monitorinqini əhatə edir. Uyğunluğun monitorinqi vasitələri, proqram təminatının kodunu və konfiqurasiyalarını uyğunluq pozuntuları, təhlükəsizlik zəiflikləri və potensial risklər üçün skan edərək, proqram məhsullarının təhlükəsizlik və normativ tələblərə cavab verməsini təmin etməyə kömək edir.

Ümumilikdə, proqram təminatının keyfiyyətinin monitorinqi proqram təminatının inkişaf dövrü ərzində yüksək keyfiyyət standartlarını qorumaq üçün vacibdir, təşkilatlara istifadəçilərin və maraqlı tərəflərin ehtiyac və gözləntilərinə cavab verən etibarlı, təhlükəsiz proqram məhsulları təqdim etməyə imkan verir.

Proqram təminatının keyfiyyətinin monitorinqi olmadan həm proqram məhsuluna, həm də onu inkişaf etdirən və ya yayınlayan təşkilata təsir edən bir sıra mənfi nəticələr yarana bilər:

- Müştəri məmnunluğunun azalması: Monitorin edilmədikdə proqram təminatı

qüsurları və keyfiyyət problemləri diqqətdən kənarda qala bilər ki, bu da istifadəçi təcrübəsinin pisləşməsinə və müştəri məmnuniyyətinin azalmasına səbəb olur. Səhvlər, xətalar və ya istifadə problemi ilə qarşılaşan istifadəçilərin proqram təminatından məyus olma və narazı olma ehtimalı daha yüksəkdir ki, bu da potensial olaraq mənfi rəylər, itkilər və ya müştərilərin itirilməsi ilə nəticələnir;

- Yüksək dəstək və texniki xidmət xərcləri: Aşkar edilməmiş keyfiyyət problemləri dəstək və texniki xidmət xərclərinin artmasına səbəb ola bilər, çünki təşkilatlar müştərilərin şikayətlərini həll etmək, qüsurları aradan qaldırmaq üçün resurslar ayırmalıdır.

Ümumiyyətlə, proqram təminatının keyfiyyətinin monitorinqi olmadan təşkilatlar istifadəçi gözləntilərinə cavab verməyən, daha yüksək dəstək və texniki xidmət xərclərinə məruz qalan, təhlükəsizlik risklərini artıran və bazarda rəqabət üstünlüyünü itirən aşağı səviyyəli proqram məhsulları təqdim etmək riski daşıyır. Proqram təminatının keyfiyyətinin monitorinqi təşkilatlara bu riskləri azaltmağa, yüksək keyfiyyətli proqram məhsulları təqdim etməyə və uzunmüddətli uğur əldə etməyə kömək edir.

Proqram təminatının keyfiyyətinin monitorinqi və qiymətləndirilməsinin məqsədi çoxşaxəlidir və proqram təminatı inkişafının həyat dövrü ərzində müxtəlif məqsədləri və faydaları əhatə edir. Proqram təminatının keyfiyyətinin monitorinqi və qiymətləndirilməsinin əsas məqsədləri bunlardır:

- Xətaların erkən aşkarlanması: Proqram təminatının keyfiyyətinə davamlı olaraq nəzarət etməklə təşkilatlar qüsurları, səhvləri və problemləri inkişaf prosesinin əvvəlində müəyyən edə bilərlər. Erkən aşkarlama operativ həll etməyə imkan verir;
- Tələblərin təsdiqi: Keyfiyyət monitorinqi proqram məhsullarının müəyyən edilmiş tələblərə və istifadəçi gözləntilərinə cavab verməsini təmin edir. Proqram təminatını əvvəlcədən müəyyən edilmiş keyfiyyət meyarlarına əsasən qiymətləndirməklə, təşkilatlar proqram təminatının nəzərdə tutulduğu

kimi işləməsini və istədiyiniz funksiyaları və imkanları təqdim etdiyini təsdiq edə bilər;

- İnkişaf proseslərinin təkmilləşdirilməsi: Proqram təminatının keyfiyyətinin monitorinqi və qiymətləndirilməsi inkişaf proseslərinin və təcrübələrinin effektivliyinə dair təsəvvürləri təmin edir. Keyfiyyət göstəricilərini təhlil edərək, tendensiyaları müəyyən edərək və performansı ölçməklə təşkilatlar ümumi keyfiyyəti artırmaq üçün təkmilləşdirmə sahələri müəyyən edə və inkişaf iş axınlarını optimallaşdıra bilər;
- Risklərin azaldılması: Keyfiyyət monitorinqi proqram təminatı qüsurları, təhlükəsizlik zəiflikləri, uyğunluq problemləri və layihə gecikmələri ilə bağlı riskləri azaltmağa kömək edir. Riskləri proaktiv şəkildə müəyyən etmək və həll etməklə təşkilatlar keyfiyyətlə bağlı problemlərin layihənin uğuru və biznes məqsədlərinə təsir ehtimalını və təsirini minimuma endirə bilər.

Ümumilikdə, proqram təminatının keyfiyyətinin monitorinqi və qiymətləndirilməsinin məqsədi proqram məhsullarının keyfiyyət standartlarına cavab verməsini, istifadəçi ehtiyaclarını ödəməsini, riskləri azaltmasını və proqram təminatının inkişaf dövrü ərzində davamlı təkmilləşməsini təmin etməkdir. Keyfiyyətli monitorinq və qiymətləndirməyə üstünlük verməklə təşkilatlar istifadəçilərin, maraqlı tərəflərin və bazarın tələblərinə cavab verən yüksək keyfiyyətli proqram məhsulları təqdim edə bilər.

1.6. Proqram təminatının keyfiyyətininin monitorinqi və qiymətləndirilməsinin əhatə dairəsi

Proqram təminatının keyfiyyətinin monitorinqi və qiymətləndirilməsinin əhatə dairəsi proqram məhsullarının həyat dövrü ərzində keyfiyyətinin qiymətləndirilməsi, saxlanması və artırılmasına yönəlmiş hərtərəfli fəaliyyətləri əhatə edir. Proqram təminatının keyfiyyətinə nəzarətin qiymətləndirilməsinin əhatə dairəsinə ümumi baxış:

1. Funksional test: Proqram təminatının müəyyən edilmiş tələblərə cavab

verməsini və nəzərdə tutulan vəzifələri dəqiq və səmərəli şəkildə yerinə yetirməsini təmin etmək üçün onun funksionallığının qiymətləndirilməsi. Buraya funksionallığı təsdiqləmək üçün fərdi xüsusiyyətlərin, istifadəçi qarşılıqlı əlaqələrinin və sistem iş axınının sınaqdan keçirilməsi daxildir.

2. Qeyri-funksional test: Performans, etibarlılıq, istifadəyə yararlılıq, təhlükəsizlik və davamlılıq kimi proqram təminatının qeyri-funksional aspektlərinin qiymətləndirilməsi. Bu, yüksək keyfiyyətli istifadəçi təcrübəsini təmin etmək üçün cavab müddətləri, səhvlərin idarə edilməsi, əlçatanlıq və miqyaslılıq kimi test amillərini əhatə edir.
3. Avtomatlaşdırılmış test: Təkrarlanan testləri və qiymətləndirmələri səmərəli şəkildə həyata keçirmək üçün avtomatlaşdırılmış sınaq çərçivələrinin və alətlərinin tətbiqi. Buraya qüsurları aşkar etmək və ardıcıl olaraq proqram təminatının keyfiyyətini təmin etmək üçün avtomatlaşdırılmış bölmə testləri, inteqrasiya testləri, reqressiya testləri və performans testləri daxildir.
4. Manual test: Proqram təminatının funksionallığını, istifadəyə yararlılığını və istifadəçi təcrübəsini yoxlamaq üçün əl ilə sınaq fəaliyyətinin həyata keçirilməsi. Manual test qüsurları aşkar etmək və istifadəçi nöqteyi-nəzərindən ümumi proqram keyfiyyətini qiymətləndirmək üçün kəşfiyyat testini, istifadəçi qəbulu testini, istifadəyə yararlılıq testini əhatə edir.
5. Təhlükəsizlik qiymətləndirmələri: Potensial zəiflikləri, təhlükəsizlik risklərini müəyyən etmək və azaltmaq üçün proqram təminatının təhlükəsizliyinin qiymətləndirilməsi. Təhlükəsizliyin qiymətləndirilməsi proqram sistemlərinin kibertəhlükələrə qarşı möhkəm və dayanıqlı olmasını təmin etmək üçün nüfuzetmə testi, zəifliyin skan edilməsi və kod təhlilinin aparılmasını əhatə edir.
6. Performans monitorinqi: Sistemin sabitliyini, genişlənməsini və səmərəliliyini qiymətləndirmək üçün proqram sistemlərinin performansının monitorinqi. Performans monitorinqi sistemin işini optimallaşdırmaq və müsbət istifadəçi təcrübəsini təmin etmək üçün cavab müddəti, ötürmə

qabiliyyəti, resurs istifadəsi və səhv dərəcələri kimi göstəricilərin toplanması və təhlilini əhatə edir.

Nəticə olaraq, proqram təminatının keyfiyyətinin monitorinqi və qiymətləndirilməsinin əhatə dairəsi genişdir. Tələblərin təsdiqlənməsi və funksional testlərin aparılmasından təhlükəsizlik qiymətləndirmələrinin aparılmasına və istifadəçi rəylərinin təhlilinə qədər proqram təminatının keyfiyyətinin monitorinqi və qiymətləndirilməsinin əhatə dairəsi proqram təminatının işlənməsinin həyat dövrünün bütün aspektlərini əhatə edir.

1.7. Proqram təminatının keyfiyyətinin yüksəldilməsində texnologiyaların rolu

Texnologiyalar prosesləri sadələşdirən, səmərəliliyi artıran və ümumi məhsulun etibarlılığını artıran alətlər, çərçivələr və metodologiyalar təmin etməklə proqram təminatının keyfiyyətinin inkişafında mühüm rol oynayır. Texnologiyaların proqram təminatının keyfiyyətinin inkişafında mühüm rol oynayır.

1. Avtomatlaşdırılmış test alətləri: Selenium, JUnit və TestNG kimi texnologiyalar vahid testi, inteqrasiya testi və reqressiya testi daxil olmaqla müxtəlif sınaq proseslərini avtomatlaşdırır. Avtomatlaşdırılmış sınaq insan səhvini azaldır, sınaq əhatəsini artırır və qüsurların müəyyən edilməsini sürətləndirir və bununla da proqram təminatının keyfiyyətini artırır.
2. Kodun statik təhlili alətləri: SonarQube, Checkstyle və ESLint kimi alətlər mənbə kodunu potensial səhvlər, təhlükəsizlik zəiflikləri və kodlaşdırma standartlarına riayət üçün təhlil edir. Statik təhlili tərtibatçılara inkişaf prosesinin əvvəlində problemləri müəyyən etməyə və həll etməyə kömək edir, daha təmiz, daha davamlı kod və təkmilləşdirilmiş ümumi proqram keyfiyyətinə gətirib çıxarır.
3. Monitorinq və müşahidə alətləri: Prometheus, Grafana və New Relic kimi alətlər proqram sistemlərinin performansını izləmək üçün monitorinq və

müşahidə imkanlarını təmin edir. Monitorinq vasitələri problemlərin proaktiv aşkarlanmasına, performans göstəricilərinin təhlilinə və sistem resurslarının optimallaşdırılmasına imkan verir ki, bu da etibarlılığın və istifadəçi təcrübəsinin artmasına gətirib çıxarır.

4. Təhlükəsizlik skanı və təhlili: OWASP ZAP, Nessus və Fortify kimi təhlükəsizlik skan alətləri proqram kodu və konfiqurasiyalarındakı təhlükəsizlik zəifliklərini müəyyən etməyə kömək edir.

Ümumilikdə, texnologiyalar prosesləri avtomatlaşdırmaq, əməkdaşlığı asanlaşdırmaq, kodun keyfiyyətini təmin etmək, performansı optimallaşdırmaq, təhlükəsizliyi artırmaq və davamlı təkmilləşdirmə üçün anlayışlar təqdim etməklə proqram təminatının keyfiyyətinin inkişafında mühüm rol oynayır. Bu texnologiyalardan səmərəli istifadə etməklə təşkilatlar istifadəçi ehtiyaclarına cavab verən, gözləntiləri üstələyən və biznes uğurunu təmin edən yüksək keyfiyyətli proqram məhsulları təqdim edə bilər.

II FƏSİL. Kod mürəkkəbliyinin proqram təminatının istismarına təsiri

2.1. Kodun mürəkkəbliyini başa düşmək

Kod mürəkkəbliyi, proqram kodunu anlamaq və davamlılğını qorumaqdakı çətinlik səviyyəsidir. Bu səviyyə, kodu başa düşmək , üzərində refaktorinq etmək üçün nə qədər səy lazım olduğunun ölçüsüdür. Proqram kodunun mürəkkəb olması gələcəkdə kodda olan xətaların aşkar edilməsi, aradan qaldırılması, kodların test olunması kimi məsələlərdə çətinliyə və vaxt itkisinə səbəb olur. Ona görə də, proqram təminatlarındakı təhlükəsizlik boşluqlarını öncədən aşkar etmək üçün kodlar müəyyən keyfiyyət meyarları ilə ölçülməlidir. Bu anlayış kodun oxunuşunu, miqyaslanmasını artırmağa və proqram təminatının hazırlanması və saxlanması zamanı xətaların və ya uğursuzluqların yaranma riskini azaltmağa kömək edir.

2.2. Kod mürəkkəbliyinin ölçülməsi üsulları

Kodun mürəkkəbliyinin ölçülməsi ilə bağlı ilk müzakirələr 1970-ci illərə təsadüf edir. İlk dəfə proqram kodlarının ölçülməsi ilə bağlı fikirləri ortaya qoyan tədqiqatçılardan biri Maurice H. Halstead-dir. Halstead, “Elements of Software Science” adlı kitabında kod mürəkkəbliyinin ölçülməsi ilə bağlı öz fikirlərini təqdim etmişdir (Halstead, 1977).

Digər mühüm şəxs, Thomas J. McCabe-dir. McCabe yazdığı “A Complexity Measure” adlı məqaləsində , proqramın idarəetmə axını (dövrlər, şərtlər və s.) vasitəsilə proqram kodundakı bütün mümkün icra yollarının sayını hesablayan Tsiklomatik Mürəkkəblik konsepsiyasını təqdim etdi (McCabe, 1976).

Kodun mürəkkəbliyi çoxşaxəli anlayışdır və onu ölçməyin vahid, qəti yolu yoxdur. Mürəkkbəliyi ölçmək üçün bir çox meyarlar var. Bu meyarlar mürəkkəbliyin müxtəlif aspektlərini ölçməyə çalışır.

Onlardan ən geniş yayılmışı Tsiklomatik Mürəkkəblikdir (ing: cyclomatic complexity). Tsiklomatik mürəkkəblik proqram kodundakı bütün mümkün icra yollarının sayıdır.

Tsiklomatik Mürəkkəblik dəyərini hesablamaq üçün müxtəlif düsturlar mövcuddur. Ən məşhurlarından biri də McCabe düsturudur. Düstur aşağıdakı şəkildədir:

$$CC = E - N + 2P$$

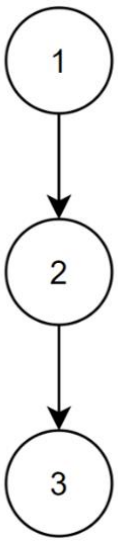
Burada E – qrafın tillərinin sayı, N – təpə nöqtələrinin sayı, P - əlaqəli komponentlərin sayıdır.

Bu formul, proqramın mürəkkəbliliyini nisbətən ölçür. Əgər CC dəyəri artarsa, proqramın mürəkkəbliliyi artamağa başlayır və test etmə çətinləşir.
Düsturdakı dəyişənlərin dəyərlərini yerinə qoyduqda alınan nəticə 1-10 aralığındadırsa bu kod sadə və az riskli olduğu mənasına gəlir. Nəticə 11-20 aralığındadırsa nisbətən mürəkkəb və orta riskli olduğu, 21-50 aralığındadırsa kodun mürəkkəbliyinin çox və yüksək riskli olması mənasına gəlir. Əgər nəticə 50 və ya 50-dən artıqdırsa bu kod çox yüksək riskli və test edilə bilinməyən kod sayılır.

İndi isə sadə və ona nisbətən mürəkkəb olan kodun tsiklomatik mürəkkəblik dəyərini ölçək. Şəkil 2.1-də Java dilində 2 ədədin toplanması kodu və onun iş axını qrafiki göstərilmişdir.

Node	Statement

(1)
(2)
(3)
	void sum(int a , int b) {
	int sum = a + b;
	System.out.println("Cəm: " + sum);
	}



Şək. 2.1 Java dilində 2 ədədin toplanması proqramı

Bu sadə kodun Tsiklomatik Mürəkkəblik dəyərinin ölçülməsi aşağıdakı şəkildədir.

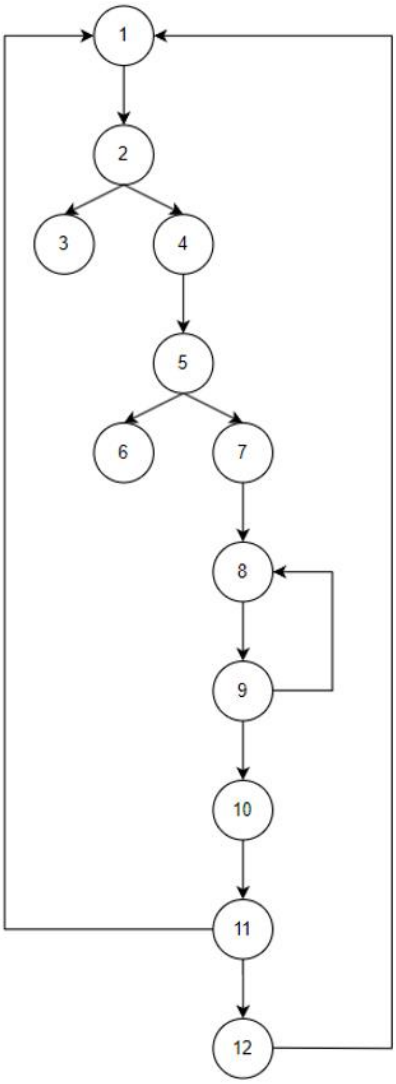
Kodda qrafın tillərinin sayı 2 olduğundan E=2 , təpə nöqtələrin sayı 3 olduğundan N=3 , əlaqəli komponentlərin sayı 1 olduğundan P=1 yazırıq. Bu komponentləri düsturda yerinə qoyaq.

$$CC = E - N + 2P = 2 - 3 + 2 \times 1 = 1$$

Nəticədə 1 dəyərini ədı etdik. Tsiklomatik Mürəkkəblik dəyəri 1 olduğundan, bu funksiyanı unit test ilə əhatə etmək üçün minimal bir test ssenarisi yazmaq kifayət edəcəkdir.

Şəkil 2.2-də Java dilində ədədin sadə olub olmamasının yoxlayan kod və onun iş axını qrafiki göstərilmişdir.

Node	Statement
(1)	void checkPrime (int number) {
(2)	if (number == 0) {
(3)	System.out.println("0 ədəd deyil !");
(4)	} else {
(5)	if (number == 1) {
(6)	System.out.println(number + " nə sadədir,nə də mürəkkəb");
(7)	} else {
(8)	for (int i = 2; i < number; i++) {
(9)	if (number % i == 0) {
(10)	System.out.println(number + " sadə ədəd deyil");
(11)	return;
	}
	}
(12)	System.out.println(number + " sadə ədəddir");
	}
	}
	}



Şək. 2.2 Java dilində ədədin sadə olub olmamasını yoxlayan proqram

Digər kod nümunəsinə nisbətən mürəkkəb olan bu kodun Tsiklomatik Mürəkkəblik dəyərinin ölçülməsi aşağıdakı şəkildədir.

Kodda qrafın tillərinin sayı 14 olduğundan E=14 , təpə nöqtələrin sayı 10 olduğundan N=3 , əlaqəli komponentlərin sayı 1 olduğundan P=1 yazırıq. Bu komponentləri düsturda yerinə qoyaq.

$$CC = E – N + 2P = 14 – 10 + 2 = 6$$

Nəticədə 6 dəyərini əldə etdik. Tsiklomatik Mürəkkəblik dəyəri 6 olduğundan, bu funksiyanı unit test ilə əhatə etmək üçün minimal 6 test ssenarisi yazmaq kifayət edəcəkdir.

Tsiklomatik Mürəkkəblik meyarının üstünlükləri

- Kod mürəkkəbliyini ölçmə. Tsiklomatik Mürəkkəblik meyarı proqram təminatının mürəkkəbliyini obyektiv şəkildə ölçür. Bu da proqramçılara proqram təminatında mürəkkəb hissələri müəyyələşdirmək və düzəltməyə rəhbərlik edir;

- Test əhatəsini müəyyənləşdirmə. Yüksək tsiklomatik mürəkkəblik dəyəri proqram təminatını test ilə tam əhatə etmək üçün daha çox sınaq ssenarisi tələb edir. Bu amil proqram təminatının test prosesini planlaşdırarkən mühüm aspektdir;

- Daha asan saxlanma. Aşağı tsiklomatik dəyər ümumilikdə kodu daha başa düşülən və saxlanılması daha asan edir. Bu da kodun daha sürətli dəyişdirilməsinə və təkrar istifadəsinə imkan verir;

- Refaktorinq ehtiyaclarını müəyyənləşdirmə. Yüksək tsiklomatik mürəkkəblik dəyəri kodun yenidən dəyişdirilməsinin və ya refaktorinq edilməsinin lazım olduğunu göstərə bilir.

Tsiklomatik Mürəkkəblik meyarının çatışmazlıqları

- Struktur mürəkkəbliyini ölçür. Tsiklomatik mürəkkəblik təkcə kodun struktur mürəkkəbliyini ölçür. Metodlar arası əlaqələr, məlumat axını və obyektlərin əlaqəsi kimi digər mürəkkəblik növlərini nəzərə almır;

- Tək başına yetərli deyil. Kod mürəkkəbliyini ölçmək üçün tək başına yetərli deyil;

- Proqramlaşdırma dilindən asılılıq. Fərqli proqramlaşdırma dillərində və fərqli layihələrdə eyni tsiklomatik mürəkkəblik dəyərinə malik kod müxtəlif mürəkkəblik səviyyələrinə malik ola bilər;

- İş axınını nəzərə alır. Tsiklomatik mürəkkəblik yalnız iş axınını nəzərə alır , məlumat axını və digər kod strukturlarını nəzərə almır. Buna görə də nəticə bəzi halllarda natamam və ya yanıltıcı ola bilər.

Bu üstünlükləri və çatışmazlıqları nəzərə alaraq belə nəticəyə gəlmək olar ki, tsiklomatik mürəkkəblik meyarı mürəkkəbliyin ölçülməsi üçün faydalı bir vasitədir, lakin kod keyfiyyətini müəyyən etmək üçün tək başına kifayət deyil.

Kod sətirləri

Kod sətirləri (ing. Lines Of Code və ya Source Lines Of Code), proqram təminatının hazırlanması zamanı istifadə olunan köhnə və fundamental meyardır. Bu meyar proqram təminatındakı kod sətirlərini sayaraq onun ölçüsünü və mürəkkəbliyini qiymətləndirmək üçün istifadə olunur. Hesablama zamanı proqram təminatının mənbə kodunda şərhlər və başlıq sətirləri nəzərə alınmır. Çünki bu sətirlər, proqramçılar tərəfindən digər proqramçılar üçün kodun izahı və ya kodun sənədləşdirilməsi üçün istifadə olunur. Buna görə də, bu sətirlərin kodun funksinallığı ilə birbaşa əlaqəsi yoxdur (Bhatia & Malhotra, 2014).

LOC meyarı , yalnız kodun həcmini ölçür və eyni proqramlaşdırma dili və kodlaşdırma standartlarını istifadə edən layihələr arasında müqayisə və ya qiymətləndirmə məqsədilə istifadə edilə bilər. Müxtəlif proqramlaşdırma dillərindən və ya kodlaşdırma standartlarından istifadə edən layihələr arasında LOC dəyərini müqayisə etmək dəqiq ölçü olmaya bilər (Bhatia & Malhotra, 2014). Bunun səbəbi, LOC meyarınım yalnız kodun həcminin ölçməsi və kod üzərində hansısa dəyişiklik zamanı bunun kodun həcminə təsir etməsidir. Bu səbəbdən müxtəlif proqramlaşdırma dili və ya müxtəlif kodlaşdırma dili istifadə edilən layihələrdə LOC dəyərini qarşılaşdırmaq kodun mürəkkəbliyi və ya keyfiyyəti haqqında dəqiq

məlumat vermir. Həcmdən asılı olmayaraq birbaşa müqayisə etməyin yaxşı yolu proqram təminatının mürəkkəbliyini ölçməkdir (Aswini & Yazhini, 2017).

Davamlılıq İndeksi

Davamlılıq indeksi (ing. maintability Index meyarı ilk dəfə 1992-ci ildə Proqram Təminatına Baxım üzrə Beyxəlxalq Konfransda Paul Oman və Jack Hagemeister tərəfindən təklif edilmişdir (Oman & Hagemeister, 1992). Bu meyar, kodun baxımının və dəyişdirilməsinin asanlığını ölçür, sonda proqram təminatının ümumi davamlılığını əks etdirən bir dəyər qaytarır. Bu dəyərin yüksək olması proqram təminatının yüksək davamlılığa malik olduğunu göstərir. Davamlılğın yüksək olması da, proqram təminatına baxımın daha asan olması deməkdir. Dəyərin aşağı olması isə proqram təminatının davamlılığının aşağı olması , yəni ona baxımın çətin olması deməkdir (Oman, P., & Hagemeister, J.,1992). Davamlılıq Indeksini hesablamaq üçün Kod sətirləri, Tsiklomatik mürəkkəblik, Halstead həcmi və kodda olan şərhlərin ümumi faizindən ibarət düstürdan istifadə olunur. Bu düsturun müxtəlif növləri var. Növlər arasında əsas fərqlər meyar seçimində və (və ya) seçilən meyarın davamlılıq üzərindəki təsirinin miqdarındadır. Oman və Hagemeister tərəfindən təklif olunan düstur aşağıdakı şəkildədir (Oman & Hagemeister, 1992):

$$MI = 171 - 3.42xln(aveE) - 0.23xaveV(g) - 16.2xln(aveLOC)$$

aveE – modul başına ortalama Halstead həcmidir;
aveV(g) – modul başına ortalama Tsiklomatik Mürəkkəblik dəyəridir;
aveLOC – modul başına ortalama kod sətridir.

Bu düstur bizə 0-100 aralığında dəyər qaytarır (Counsell, Liu, Eldh, Tonelli, Marchesi, Concas, & Murgia, 2015). Bu aralıqlar aşağıdakı kimi şərh olunur :

- 0-9 : dərhal refaktorinq tələb edən, davamlılğı çox aşağı olan kod;
- 10-19 : vacib ölçüdə refaktorinqə ehtiyacı olan, davamlılığı aşağı olan kod;
- 20-29 : təkmilləşdirməyə ehtiyacı olan, davamlılığı orta səviyyədə olan kod;
- 30-100 : üzərində daha asan düzəlişlər edilə bilən, davamlılığı yaxşı olan kod;

Proqramçılar davamlılığa diqqət yetirməklə, kodun keyfiyyətini artıra, səhvləri azalda və kod üzərində təkmilləşdirmə və modifikasiya işlərini daha da asanlaşdıra bilərlər.

Davamlılıq indeksinin üstünlükləri:

- Kod keyfiyyətinin ölçülməsi. Davamlılıq indeksi kodun mühüm keyfiyyət atributlarını ölçür. Bu da proqramçılara kodun keyfiyyətini obyektiv qiymətləndirməyə imkan verir;

- Fərqli meyarları nəzərə alma. Davamlılıq indeksi kodun ölçüsü , mürəkkəbliyi və strukturu kimi mühüm amilləri nəzərə alır. Nəticədə yalnız bir metrikdən istifadə edərək kodu hərtərəfli qiymətləndirmək mümkün olur;

- Fərqli proqramlaşdırma dillərdə və mühitlərdə istifadə imkanı. Davamlılıq indeksi müxtəlif proqramlaşdırma dillərində və müxtəlif layihə növlərində istfiadə edilə bilər.

Davamlılıq indeksinin çatışmazlıqları:

- Yetərsizliklər və anlaşılmazlıqlar. Baxım İndeksi bəzi kod keyfiyyəti atributlarını dəqiq ölçməyə və ya yanlış nəticələr verə bilər. Məsələn, bəzi xüsusiyyətlərin və ya kod strukturlarının təsirlərini tam əks etdirməyə bilər;

- Tək başına yetərli deyil. Təkcə davamlılıq indeksi kodun keyfiyyətini müəyyən etmək üçün kifayət deyil. Kodun ümumi keyfiyyətini qiymətləndirmək üçün digər ölçülərdən istifadə edilməlidir;

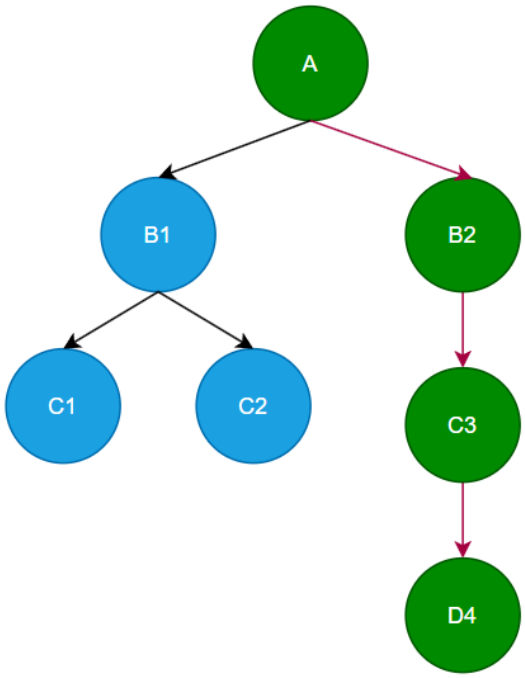
- İnsan faktorunun nəzərə alınmaması. Davamlılıq indeksi istifadəçilərin və ya müştərilərin ehtiyaclarını, istifadəçi təcrübəsini və ya biznes tələblərini nəzərə almır. Buna görə də kodun real dünya şəraitində necə işləyəcəyini qiymətləndirmək yetərli olmaya bilər.

Varislik ağacının dərinliyi

Varislik ağacının dərinliyi (ing. The Depth of Inheritance Tree) meyarı ilk dəfə Shyam R. Chidamber və Chris F. Kemerer tərəfindən ortaya qoyulmuşdur (Chidamber & Kemerer, 1994). Bu meyar proqram təminatı mühəndisliyi sahəsində,

proqram təminatının keyfiyyətini ölçmək üçün önəmli bir meyar sayılır. DIT meyarı, digər meyarlarla birlikdə proqram sistemlərində, o cümlədən açıq mənbəli proqram təminatlarındakı nasazlıqları, yenidən istifadənin mümkünlüyünü və digər keyfiyyət göstəriciləri-ni qiymətləndirmək üçün istifadə olunur (Sergiy Prykhodko, Natalia Prykhodko & Smykodub, 2021). Həmçinin DIT meyarı proqram təminatının test edilməsinə ayrılan məbləğə təsir edən bir faktor olaraq qəbul edilir (Shaheen & du Bousquet, 2008).

Bu meyar, bir sinifin obyekt yönümlü proqramlaşdırma iyerarxiyasının təpəsindən başlayaraq miras səviyyəsinin nə qədər dərinlikdə olduğunu ölçür. Yəni bir sinifin özünü birbaşa miras alan sinif sayını deyil, eyni zamanda bu siniflərin miras aldığı başqa siniflərin neçə səviyyə dərinlikdə olduğunu göstərir. Polimorfizm hallarında sinif səviyyəsində DIT meyarının dəyəri düyün nöqtəsindən (yəni sinifdən) ağacın kökünə qədər olan maksimum uzunluqdur. DIT dəyərinin 0-a bərabər olması kökü göstərir (Chidamber & Kemerer, 1994). Dəyərlərin əksəriyyəti 2-dən aşağı olarsa, bu obyekt yönümlü dizayn və mirasın üstünlüklərindən lazımı qədər istifadə edilmədiyini göstərir. DIT dəyərinin maksimum 5 olması tövsiyyə olunur, çünki daha çox köklü ağaclarda daha çox metod və siniflər olur (Sergiy Prykhodko, Natalia Prykhodko & Smykodub, 2021). Bu da ümumi strukturu daha da mürəkkəbləşdirir. Şəkil 2.3-də DIT dəyərinin 4 olduğu nümunə göstərilmişdir.



Şək. 2.3 Varislik ağacının dərinliyi

DIT meyarının üstünlükləri

- Sinif iyerarxiyasını anlamaq. Varislik ağacının dərinliyi siniflər arasındakı əlaqələri başa düşmək haqqında fikir verir. Bu, proqram tərtibatçılarına kodun ümumi strukturunu daha yaxşı başa düşmək və təhlil etmək imkanı verir;

- Varislik iyerarxiyasının sadələşdirilməsi. Daha aşağı dəyər daha çevik, elastik kod bazaları yarada bilər. Bu da, kodun müxtəlif situasiyalarda təkrar istifadəsini təmin edir;

- Performans və yaddaşın idarə edilməsi. Daha aşağı dəyər obyektin yaradılması və idarə edilməsini daha sürətli və səmərəli edə bilər;

DIT meyarının çatışmazlıqları

- Yüksək asılılıq riski. Daha dərin varislik ağacı siniflər arasında daha çox asılılıq və əlaqələr deməkdir. Nəticədə ən kiçik dəyişiklərin edilməsi halında belə asılılıq olan bütün siniflərdə dəyişiklik etmək lazım olur;

- Kodun təkrar istifadə imkanının azalması. Varislik ağacının dərin olması kodun daha konkret və əlaqəli olmasına səbəb ola bilər ki , bu da kodun təkrar istifadəsini azalda bilər;

- Xətaları aşkarlamaqda çətinlik. Varislik ağacının dərin olması xətaların aşkarlanması proseslərini çətinləşdirə bilər. Çünki xəta aşkarlandıqda onun mənbəyini müəyyən etmək və izləmək çətin ola bilər.

Metodların Uyğun olmaması

Kod mürəkkəbliyini ölçmək üçün istifadə edilən üsullardan biri də LCOM meyardır. Metodların uyğun olmaması (ing. Lack of Cohesion of Methods) mənasını verən LCOM, sinif daxilində metodların bir-birinə uyğun olmama dərəcəsini ölçmək üçün istifadə olunan bir meyardır.

Siniflər obyekt yönümlü proqram təminatının əsas vahidləridir. Sinifdə bir çox metod və dəyişən var. Bu metod və dəyişənlər arasındakı əlaqə, sinifin uyğunluğu haqqında təsəvvür yaradır. Bu əlaqə, uyğunluğu artıraraq kodun

keyfiyyətinə töhfə verir. Uyğunluğun artırılması, kodun davamlılığını və başa düşülməsini artırır.

Uyğunluq (ing: cohesion) sinifdəki metodlar və dəyişənlər arasında qarşılıqlı təsir dərəcəsi kimi ölçülür. Uyğunluq adətən kodun strukturuna baxmaqla hesablanır. Hər bir uyğunluq meyarı, uyğunluğu müxtəlif yollarla ölçür. Müvafiq olaraq, hər bir meyar eyni kod üçün fərqli nəticələr əldə edir. Əgər bir metod bir dəyişəni istifadə edirsə, bu metod və dəyişən arasında bir əlaqə var. Həmçinin bir metod başqa bir metodu çağırırsa, bu iki metod bir-birilə əlaqəlidir. Buna birbaşa əlaqə (ing: direct relation) deyilir. Tutaq ki, bir sinfin m1, m2 və m3 adında üç ayrı metodu var. Əgər m1 metodu m2 metodunu, m2 metodu isə m3 metodunu çağırırsa, m1 və m3 metodları arasında dolayı əlaqə vardır (Linda Badri & Mourad Badri, 2004).

Bəzi ölçülər yalnız metod-atribut əlaqəsinə əsaslanır. Bəziləri birbaşa atribut-metod və metod-metod əlaqəsini nəzərdən keçirərkən, bəzi ölçülər də dolayı əlaqələri nəzərə alaraq hesablamalar aparır. Sinfin uyğunluq dəyəri uyğunluğun ölçülməsindən sonra əldə edilən nəticəyə görə yüksək və ya aşağı kimi şərh edilir. Bu şərhi etmək üçün meyarın normallaşdırılması vacibdir. Normallaşdırıldıqda, əldə edilən dəyər daha asan şərh edilə bilər. Uyğunluq dəyəri 0-a yaxın olduqda uyğunluq aşağı, 1-ə yaxın olduqda uyğunluq yüksək kimi şərh olunur (Dallal & Briand, 2012). Uyğunluq dəyəri yüksək olduqda kodun başa düşülməsi, davamlılığı və etibarlılığı artır. Bu situasiya, birbaşa kodun keyfiyyətində özünü göstərir.

Uyğunluq dəyəri aşağı olduqda SOLID prinsipinin ilk konsepsiyası olan tək məsuliyyət (ing. single responsibility) prinspini pozmuş oluruq. Buna görə də sinifi bölmək məcburiyyətində qalırıq. Keyfiyyətli proqram təminatı üçün ən vacib amillərdən biri yüksək uyğunluqdur.

LCOM meyarının üstünlükləri

- Kodun anlaşıqlılığının artırma. Metodlar arasında əlaqənin kəsilməsi sinifin nə iş gördüyünü daha aydın edir. Nəticədə kod daha asan anlaşılır və onun baxımı daha asan olur;

- Kodun təkrar istifadə edilməsini təmini. Bir biri ilə bağlı olmayan metodlar müxtəlif kontekslərdə daha asan təkrar istifadə edilə bilir. Metodun dəyişdirilməsi və ya yenilənməsi digərlərinə təsir etmir, kodu daha çevik və təkrar istifadə edilə bilən edir.

LCOM meyarının çatışmazlıqları

- Performans problemləri. Yüksək əlaqə mübadilə mübadiləsi səbəbindən performans problemlərinə səbə ola bilər;

- Baxım və sazlama problemləri. Metodlar arasında aşağı əlaqənin olması kodun saxlanması və sazlanmasını çətinləşdirə bilər. Çünki bir metodda səhvi dəyişdirərkən və ya taparkən digər metodların necə təsirlənəcəyini proqnozlaşdırmaq daha çətin ola bilər;

- Kompozit funksiyaların müəyyən edilməsində çətinliklər. Metodlar arasında əlaqənin olmaması bəzən oxşar funksiyaları olan metodların ayrı-ayrı siniflərə aid olması anlamına gələ bilər. Bu halda, oxşar funksiyaları birləşdirmək və ya yenidən konfiqurasiya etmək çətin ola bilər.

Testin əhatə dairəsi

Testin əhatə dairəsi (ing. test coverage) proqram təminatının keyfiyyətinin mühüm göstəricisidir və proqram təminatına texniki xidmətin vacib hissəsidir. Texniki xidmət müddətində proqram təminatının test edilməsi mühüm fəaliyyət hesab olunur. Test etmə proqram təminatının keyfiyyətini müəyyən etmək və yaxşılaşdırmaq üçün istifadə olunur. Bunun üçün çoxsaylı metodlar mövcuddur, məsələn, funksionallıq testləri, performans testləri, istifadəçi interfeysi testləri və s. Bunlar proqramın doğruluğunu, effektivliyini və istifadəçi təcrübəsini təmin etmək üçün əhəmiyyətlidir. Həmçinin, istifadəçi geri bildirimləri, beta testləri və QA

prosesləri kimi tədbirlər də proqram təminatının keyfiyyətini yaxşılaşdırmaqda vacib rol oynayır.

Test proseslərinə proqram təminatında testin əhatə dairəsinin əldə edilməsi də daxildir. Əhatə dairəsi , test ilə əhatə olunmuş elementlərin ümümi faizidir. Əgər əhatə dairəsi 100% deyilsə, əhatədən kənar elementləri yoxlamaq üçün daha çox testlər yazılaraq testin əhatə dairəsi artırılır (ISTQB, 2007). Testin əhatə dairəsi testin keyfiyyətinə nəzarət etməyə və sınaqdan keçirilməmiş sahələri əhatə edən test nümunələri yazmağa kömək edir Grinwald, Harel, Orgad, Ur, & Ziv, 1998).

Əhatə dairəsinin ölçülməsinin nəticəsi , test prosesini təkmilləşdirmək üçün bir neçə yolla istifadə edilə bilər (Shahid, Ibrahim, & Mahrin, 2011). O, həmçinin istifadəçiyə yoxlama prosesinin vəziyyəti haqqında məlumat verir, testdəki boşluqların, yəni əhatə edilməyən sahələrin tapılmasına kömək edir Grinwald, Harel, Orgad, Ur, & Ziv, 1998). Testin əhatə dairəsi eyni zamanda reqressiya testinə, test işinin prioritetləşdirilməsinə, test paketinin artırılmasına və ya azaldılmasına kömək edir.

2.3 Proqram Təminatı İnkişafının Həyat Dövrü

Proqram təminatı inkişafının həyat dövrü proqram təminatı hazırlanmasının əvvəlindən sonuna qədər olan prosesi təsvir edir. Bu proses 6 və ya 7 mərhələdən ibarət olur. Bu mərhələlər Şəkil 2.4-də göstərilmişdir.

Şək. 2.4 Proqram təməinatı inkişafının həyat dövrü

Kodun mürəkkəbliyi proqram təminatı inkişafının həyat dövrünün bütün mərhələrinə əhəmiyyətli dərəcədə təsir edir.

Planlaşdırma. Planlaşdırma mərhələsi layihənin uğurlu icrası üçün əsas mərhələdir. Bu mərhələdə tələblər aydınlaşdırılır, məqsədlər və məhdudiyyətlər müəyyən edilir, resurslar təyin edilir və vaxt qrafiki müəyyən edilir. Həmçinin planlaşdırma

mərhələsində komanda yerinə yetiriləcək tapşırıqları və onların yerinə yetirilməsi qaydasını müəyyən edən layihə planı yaradır. Bu mərhələ layihə komandasının aydın bir məqsədə doğru irəliləməsini və layihənin uğurla başa çatmasını təmin edir. Kod mürəkkəbliyi planlaşdırma mərhələsinə təsir edən ən mühüm amillərdən biridir. Mürəkkəb kod bazası, layihənin resurslara və vaxt qrafikinə olan ehtiyaclarını artıra bilər. Bu halda resursların düzgün bölüşdürlməsi və vaxt qrafikinin yenidən nəzərdən keçiriləsi tələn oluna bilər. Həmçinin mürəkkəb kod bazaları layihənin risk səviyyəsini də artıra bilər. Bundan əlavə, mürəkkəb kod bazaları layihənin maraqlı tərəfləri arasında əlaqəni və layihənin gedişatını izləməyi çətinləşdirə bilər. Buna görə də, planlaşdırma mərhələsində kommunikasiya strategiyasını və monitorinq planını müəyyən etmək çox vacibdir.

- **Tələblərin toplanması və təhlili.** Tələblərin toplanması və təhlili proqram təminatı inkişafının həyat dövrünün mühüm mərhələsidir. Bu mərhələdə inkişaf qrupu proqram layihəsinin ehtiyaclarını, məqsədlərini və məhdudiyyətlərini başa düşmək və sənədləşdirmək üçün maraqlı tərəflərlə sıx əməkdaşlıq edir. Bu zaman müvafiq məlumatlar toplamaq üçün araşdırmalar, müsahibələr və müzakirələr aparılır. Nəticədə toplanmış tələblər təhlil edilir və layihənin ümumi məqsədlərinə uyğun olmasını təmin etmək üçün prioritetləşdirilir. Bu mərhələ, inkişaf prosesi zamanı yarana biləcək hər hansı potensial problem və ya riskləri müəyyən etməyə kömək edir.

Kodun mürəkkəbliyi tələblərin toplanması və analizi mərhələsinə əhəmiyyətli dərəcədə təsir göstərir. Mürəkkəb kod bazası layihənin tələblərini başa düşməyi, müəyyən etməyi, mövcud kod bazasını nəzərdən keçirməsini və tələblərin necə həyata keçiriləcəyini başa düşməsini çətinləşdirir. Bundan əlavə, kodun mürəkkəbliyi tələblərin aydın şəkildə göstərilməsinə və sənədləşdirilməsinə mane ola bilər. Bu halda layihənin tələblərini dəqiq müəyyən etmək və prioritetləşdirmək çətinləşir. Buna görə də tələblərin toplanması və təhlili mərhələsində kodun mürəkkəbliyini nəzərə almaq və müvafiq strategiyaları müəyyən etmək vacibdir.

- **Dizayn və arxitektura.** Dizayn və arxitektura, proqram təminatının inkişaf dövründə həlledici rol oynayır. Buraya proqram sistemi üçün planın yaradılması, onun strukturu və komponentlərinin təsviri daxildir. Dizayn mərhələsi, sistemi daha kiçik, idarə edilə bilən modullara bölmək və onların qarşılıqlı əlaqəsini müəyyən etməklə kod mürəkkəbliyini həll etmək məqsədi daşıyır. Yaxşı dizayn edilmiş arxitektura ilə proqramçılar proqram təminatının davamlılığını və çevikliyini artıra bilərlər. Bu mərhələdə proqramçılar kod mürəkkəbliyini nəzərə alaraq, yarana biləcək səhvlərin riskini effektiv şəkildə azalda və proqram təminatının ümumi keyfiyyətini artıra bilər.

Kod mürəkkəbliyinin dizayn mərhələsinə təsiri vacibdir. Mürəkkəb kod bazası dizayn prosesini çətinləşdirərək nəticədə daha çox resurs və vaxt tələb edəcək. Bundan əlavə, mürəkkəb kod bazaları tez-tez performansa və miqyaslanmağa mənfi təsir göstərir. Buna görə də dizayn mərhələsində kodun mürəkkəbliyini azaltmaq və daha səmərəli sistem yaratmaq vacibdir.

- **Kodlaşdırma.** Kodlaşdırma mərhələsi proqram təminatı inkişafının həyat dövrünün həlledici mərhələsidir. Bu mərhələdə proqramçılar tələbləri təhlil edərək proqram təminatının formalaşmasını təmin edən kodları yazır. Bu zaman oxunaqlı olmayan mürəkkəb kodlar yazılarsa, mərhələnin effektivliyinə və səmərəliliyinə əhəmiyyətli təsir göstərə bilər.
- **Test etmə.** Test və keyfiyyət təminatı proqram təminatının işlənib hazırlanmasında mühüm rol oynayır. Bu mərhələdə proqram mühəndisləri kodun funksionallığını, performansını və etibarlılığını yoxlamaq üçün müxtəlif növ testlər həata keçirirlər. Potensial qüsurları müəyyən etmək və proqram təminatının müəyyən edilmiş tələblərə cavab verməsini təmin etmək üçün blok testi (ing. unit testing), inteqrasiya testi (ing. integration testing) və sistem testi kimi üsullardan istifadə edirlər. Həyata keçirilən test və keyfiyyətin təminatı fəaliyyətləri, proqram təminatının təkmilləşdirilməyə ehtiyacı olan sahələrini müəyyən etməyə, kodlaşdırma standartlarına və ən

yaxşı təcrübələrə (best practices) uyğun olmasını təmin edir. Effektiv sınaq və keyfiyyət təminatı prosesləri yarana biləcək səhvlərin ehtimalını azaltmaq, proqram təminatının ümumi keyfiyyətini yaxşılaşdırmaq və istifadəçi təcrübəsini artırmaq üçün vacibdir. Mürəkkəb kod bazası test ssenarilərinin hazırlanmasını çətinləşdirir və sınaq qabiliyyətinin artırılması tələb oluna bilər.

- **Yerləşdirmə və texniki xidmət.** Yerləşdirmə və texniki xidmət proqram təminatı inkişafının həyat dövrü mühüm mərhələləridir.

Yerləşdirmə mərhələsində hazırlanmış proqram təminatı son istifadəçilərin mühitinə köçürülür və quraşdırılır. Bu proses uyğunluğun təmin edilməsini, lazımi parametrlərin konfiqurasiya edilməsini və istənilən inteqrasiyanın həyata keçirilməsini əhatə edir. Yerləşdirildikdən sonra proqram təminatı davamlı dəstək və yeniləmələrin təmin olunduğu texniki xidmət mərhələsinə keçir. Texniki ximət fəaliyyətlərinə səhvlərin aradan qaldırılması, performansın optimallaşdırılması və uyğunluq problemlərinin həlli daxildir. Kod mürəkkəbliyi həm yerləşdirmədə, həm də texniki xidmətdə mühüm rol oynayır, çünki mürəkkəb kod quraşdırma zamanı çətinliklərə səbəb ola bilər və texniki xidmət tapşırıqlarının səmərəliliyinə mane ola bilər.

2.4 Proqram təminatının inkişafı metodologiyası

Proqram təminatının inkişafı metodologiyası prosesləri planlaşdırmaq, idarə etmək və nəzarət etmək üçün istifadə olunan çərçivədir (Pargaonkar, 2023). Proqram təminatının təkmilləşdirilməsi metodologiyası proqram təminatının inkişafının həyat dövrü kimi tanınır və bir çox mühəndislik, sənaye sahələrində və s. Bir çox sahələrdə istifadə olunur. Dünyada bir çox tədqiqatçı tərəfindən müzakirə olunmuş, tədqiq edilmiş və nəticədə öz güclü və zəif tərəfləri olan bir çox model yaradılmışdır. Agile, Waterfall və s. Bir çox model mövcuddur. Hər biri özünəməxsus metodologiyaya sahibdir və müəyyən nəticələr əldə etmək, son məhsul istehsal etmək üçün atılmalı olan addımlar ardıcıllığına malikdir.

Çevik proqram təminatının inkişafı (ing. agile software development) paradiqması son bir neçə ildə getdikcə populyarlaşdı, çünki o, aşağı xərclər, daha yaxşı məhsuldarlıq, daha keyfiyyətli və daha yaxşı biznes məmnuniyyəti tələb edir (Deepti Mishra & Alok Mishra, 2011).

Çevik metodlar işlənib hazırlanacaq proqram təminatının qeyri-səlis və ya dəyişən tələblərə malik olduğu və layihənin həyat dövrü ərzində dəyişən tələblərin öhdəsindən gələ bildiyi hallarda əlverişli həll təklif edir (Cohn & Ford, 2003). Bu metodologiya çeviklik, əməkdaşlıq və iterativ çatdırılmanı vurğulayan proqram təminatının hazırlanmasına müasir yanaşmadır (Al-Saqqa, Sawalha, & AbdelNabi, 2020). O, 90-cı illərin sonlarında təqdim edilib və hazırda bütün dünyada "əsas" proqram mühəndisliyi kimi qəbul edilir. Uyğunlaşa bilən quruluşa görə xüsusilə mürəkkəb layihələr üçün uyğundur. Çevik metodologiyalar mürəkkəb layihələri daha kiçik, idarə oluna bilən tapşırıqlara bölərək komandalara dəyişən tələblərə tez cavab verməyə və ənənəvi xətti yanaşmaların tələlərindən qaçmağa imkan verir. Ümumilikdə, Agile metodologiyası proqram təminatı layihələrinin mürəkkəbliklərini effektiv şəkildə idarə edə bilən çevik və dinamik çərçivə təmin edir.

Bu üsullar ənənəvi proqram təminatının işlənməsi ilə müqayisədə daha yüksək çeviklik və çevikliyə malik olduğunu sübut etdi (Qumer & Henderson-Sellers, 2008) və daha qısa müddət ərzində daha yüksək keyfiyyətli proqram təminatı istehsal etmək üçün istifadə olunur (Livermore, 2007).

Digər metologiya da, Waterfall metodologiyasıdır. Waterfall metodologiyası mürəkkəb layihələrdə geniş istifadə olunan proqram təminatının hazırlanmasına xətti, ardıcıl yanaşmadır. O, tələblərin toplanması, sistemin dizaynı, həyata keçirilməsi, sınaqdan keçirilməsi və istismara verilməsi daxil olmaqla, əvvəlcədən müəyyən edilmiş mərhələlər toplusunu izləyir. Hər bir mərhələ əvvəlkinin tamamlanması üzərində qurulur və dəyişikliklər ciddi şəkildə idarə olunur. Bu üsul xüsusilə son məhsulun aydın başa düşülməsinin vacib olduğu dəqiq müəyyən edilmiş və sabit tələbləri olan layihələr üçün uyğundur. Waterfall metodologiyası

növbəti mərhələyə keçməzdən əvvəl hər bir addımın tamamlanmasını təmin edən strukturlaşdırılmış yanaşma təqdim edir. O, sənədləşdirmə və planlaşdırmanı vurğulayaraq onu tənzimləyici və ya uyğunluq tələbləri olan layihələr üçün faydalı edir. Bununla belə, onun sərtliyi mürəkkəbliklə qarşılaşdıqda bir dezavantaj ola bilər, çünki hər hansı gözlənilməz dəyişiklik bütün prosesi poza bilər. O, həmçinin rəyi qəbul etmək və dəyişən müştəri ehtiyaclarına uyğunlaşmaq üçün çevikliyə malik deyil. Ümumiyyətlə, Waterfall metodologiyası aşağı mürəkkəbliyi, aydın tələbləri və başa çatdırılması üçün proqnozlaşdırıla bilən yolu olan layihələr üçün uyğundur (Aroral, 2021).

2.5. Kod mürəkkəbliyinin proqram təminatına təsirləri.

- **Kod mürəkkəbliyinin proqram təminatının inkişaf müddətinə təsiri**

Artan mürəkkəblik inkişaf müddətinə əhəmiyyətli dərəcədə təsir göstərir. Mürəkkəblik artdıqca, məhsul və ya sistemin inkişafı üçün tələb olunan vaxt da artır. Bu bir neçə faktorla bağlıdır. İlk olaraq, mürəkkəb layihələr daha çox tələblərə, xüsusiyyətlərə və komponentlərə malikdir ki, bu da təbii olaraq daha çox vaxt tələb edir. Həmçinin, mürəkkəblik müxtəlif komponentlər arasında asılılığı və qarşılıqlı əlaqəni artıra bilər ki, bu da layihənin idarə edilməsində daha çox planlaşdırma və nəzarət tələb edir.

Digər faktor, mürəkkəb sistemlərin inkişaf prosesini daha da gecikdirən səhvlərə və inteqrasiya problemlərinə daha çox meylli olmasıdır. Buna görə də, təşkilatlar üçün inkişaf qrafiklərini effektiv şəkildə planlaşdırmaq və idarə etmək üçün mürəkkəbliyin nəticələrini anlamaq və həll etmək çox vacibdir.

- **Kod mürəkkəbliyinin proqram təminatın texniki qulluq xərclərinə təsiri**

Proqram təminatına texniki qulluq proqram təminatının hazırlanması prosesinin vacib komponentidir (Shukla & Misra, 2008). Proqram təminatına texniki xidmət, xətaların düzəldilməsi, yeniliklərin əlavə edilməsi və s. yaxşılaşdırılmaların edilməsi prosesidir. Proqram təminatının ölçüsü böyüdükcə belə proqram təminatının mürəkkəbliyini ölçmək lazım gəlir. Çünki mürəkkəblik artıqca proqram

təminatının ölçüsü də artır . Bu da öz növbəsində təşkilatlarda proqram təminatına texniki qulluq xərclərinin artmasına səbəb olur. Çünki təşkilatlar artıq proqram sistemlərindən tamamilə asılıdır və onlara milyonlarla dollar sərmayə qoyublar. Onların sistemləri kritik biznes aktivləridir və bu aktivlərin dəyərini qorumaq üçün texniki qulluq xərclərini artırmalıdırlar (Shukla & Misra, 2008).

- **Kod mürəkkəbliyinin proqram təminatının performansına təsiri**

Mürəkkəb kodlar daha yavaş çalışır. Bu yavaşlıq lazımsız dövrlər, həddən artıq resur istifadəsi və ya sürətli çalışmayan alqoritmlər bağlı ola bilər. Məsələn lazımsız dövrlər əməliyyat sistemini boşuna yoraraq performansa təsir edə bilər. Resurslarından səmərəli şəkildə istifadə edilməməsi sistemin resurslarının tükənməsinə və əlavə xərclərə səbəb ola bilər. Səmərsiz çalışan alqoritmlər gec nəticələr qaytararaq proqram təmiantının performansına mənfi təsir edə bilər.

- **Kod mürəkkəbliyinin proqram təminatının məhsuldarlığına təsiri**

Texnologiyanın inkişafı və tələbatın artması ilə proqram təminatı hazırlayan şirkətlərin sayı da sürətlə artır. Bu cür şirkətlərin sayı artdıqca rəqabət də getdikcə artmaqdadır. Şirkətlər bu bazar rəqabətində uğur qazanmaq üçün təkcə məhsulun keyfiyyətini yaxşılaşdırmamalı, həm də proqram təminatının işlənib hazırlanma proseslərini də daha da səmərəli etməlidir (Card, Clark, & Berg, 1987).

Proqram təminatının idarə edilməsi üç əsas amildən ibarətdir: texnologiya, insanlar və proseslər (Chiang & Mookerjee, 2004). Bu amillər proqram təminatının səmərəliliyini artırmaq üçün vacib yollardır.

Proqram təminatının yaradılması prosesində alətlərdən səmərəli istifadə etmək bacarığı (Boehm, 1987), dəyişən tələblər və natamam metodlar kimi amillər məhsuldarlığa təsir edən problemlər kimi ortaya çıxır. Bununla belə, layihənin mürəkkəbliyi insanların yaradıcılığına, yeni bacarıqları öyrənmək istəyinə və müvafiq olaraq məhsuldarlığa ən çox təsir edən amildir.

Mürəkkəb və ya çətin başa düşülən kod proqram təminatının hazırlanması prosesini ləngidə, və xətaların baş vermə ehtimalını artıra bilər. Həmçinin mürəkkəb kodlar komandaya yeni qatılan üzvlərin layihəyə uyğunlaşmasını çətinləşdirə bilər. Bunun

nəticəsində də həm proqam təminatınını məhsuldarlığı həm də komandanın məhsuldarlığı azala bilər.

- **Kod mürəkkəbliyinin proqram təminatının istifadəçi təcrübəsinə təsiri**

Performans problemləri istifadəçi təcrübəsinə də təsir edə bilər. Bunun nəticəsində də proqram təminatı daha az istifadəçi tərəfindən seçilməsinə səbəb ola bilər. Çünki adətən istifadəçilər yavaş çalışan proqram təminatlarından narazı qalaraq daha sürətli və səmərəli çalışan proqram təminatına yönəlirlər. Bu da öz növbəsində həm proqram təminatının uğuruna həm də istifadəçi məmnunluğuna mənfi təsir göstərir. Bu səbəbdən proqram təminatının hazırlanması zamanı kod mürəkkəbliyinin azaldılması və istifadəçi təcrübəsinin ön planda tutulması vacibdir.

- **Kod mürəkkəbliyinin proqram təminatının təhlükəsizliyinə təsiri**

Kod mürəkkəbliyinin proqram təminatının təhlükəsizliyinə təsiri olduqca vacibdir. Çünki təhlükəsizlik boşluqları yarada biləcək amillərdən biri də məhz mürəkkəb kodlardır. Aşağıda mürəkkəb kodların proqram təminatının təhlükəsizliyinə təsirləri sadalanmışdır:

- **Zəifliklərin gizlənməsi:** Mürəkkəb kodlar daxilində zəifliklər asanlıqla gizlənə bilər. Çünki mürəkkəb kodlar daxilində boşluqları aşkarlamaq olduqca çətindir. Bu zəifliklər aşkarlanmadığı halda, proqram təminatı potensial hücumlara qarşı müdafiəsiz qalır.
- **Boşluqların aşkarlanması və aradan qaldırıması müddəti:** Mürəkkəb kodlar təhlükəsizlik boşluqlarının aşkarlaması və aradan qaldırıması prosesini çətinləşdirə bilər. Mürəkkəb strukturlar daxilində təhlükəsizlilə bağlı problemləri müəyyənləşdirmək və aradan qaldırmaq daha çox zaman tələb edir.
- **Təhlükəsizlik testlərinin mürəkkəbliyi:** Mürəkkəb kodlar təhlükəsizlik testlərini həyat keçirməsini çətinləşdirir. Çünki test ediləcək çox ssenari olduğundan testerlər təhlükəsizlik testlərini həyata keçirmək üçün daha çox səy göstərməli ola bilərlər.

2.11. Proqram təminatının hazırlamasında mürəkkəbliyin azaldılması

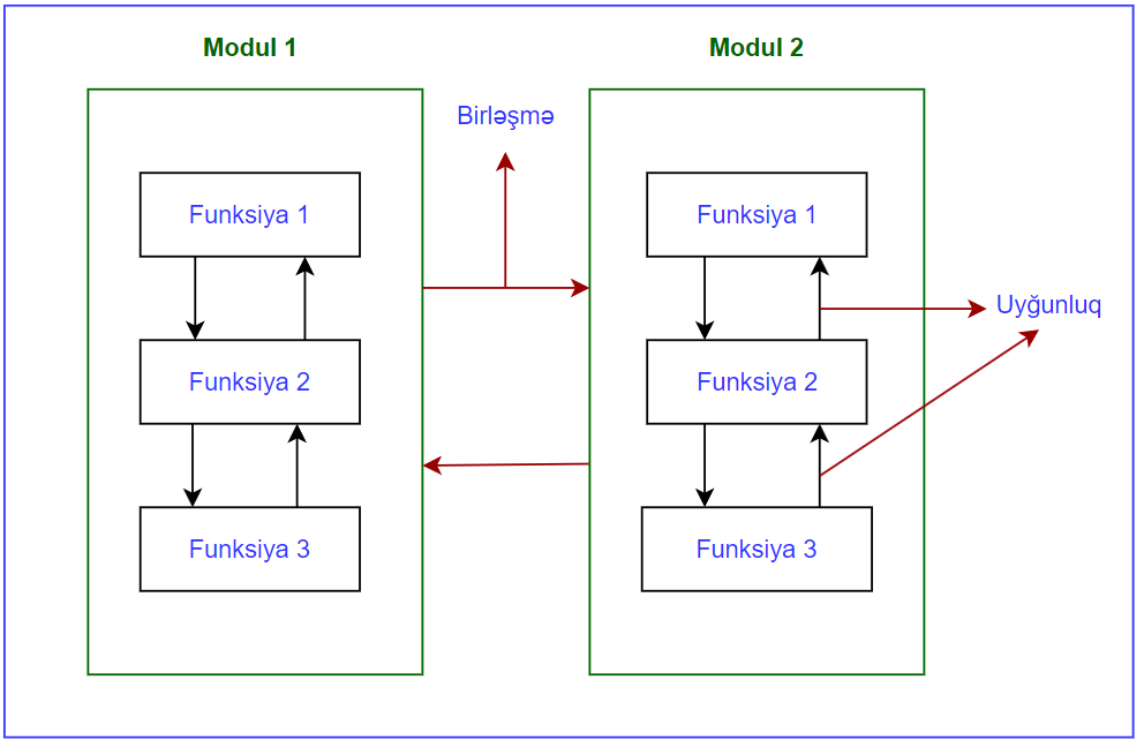
Proqram təminatının hazırlanmasında mürəkkəbliyin azaldılması kod bazasının sadələşdirilməsinə və texniki xidmətin yaxşılaşdırılmasına yönəlmiş müxtəlif üsulları əhatə edir. Bu üsulları tətbiq etməklə proqramçılar mürəkkəb kodun yaratdığı problemləri azalda bilərlər. Refaktorinq, modul dizayn və dizayn nümunələrinin istifadəsi kimi texnikalar bu məqsədə çatmaqda həlledici rol oynayır. Bu üsullardan istifadə edərək tərtibatçılar kodu sadələşdirə, təkrarlanmanı aradan qaldıra və ümumi kodun keyfiyyətini yaxşılaşdıra bilərlər. Bu üsullar proqram təminatının tərtibatçılar üçün başa düşülməsini asanlaşdırmaqla yanaşı, gələcəkdə dəyişikliklər etmək və yeni funksiyalar əlavə etmək qabiliyyətini də təkmilləşdirir.

Modul dizayn. Proqram təminatı çoxlu alt sistemləri özündə birləşdirən mürəkkəb strukturdur. Buna görə də, hər bir funksionallığı özündə birləşdirən vahid sistemin dizaynı çətin və səhvlərə səbəb ola bilər. Bu problemi həll etmək üçün proqram təminatının müstəqil modullara bölünür. Modul dizayn hər modulu müstəqil tərtib etməyə, üzərində dəyişiklik etməyə imkan verir. Bu da öz növbəsində proqram təminatının hazırlanması prosesini asanlaşdırır və yaranan xətaların akarlanıb aradan qaldırılması proseslərini sürətləndirir.

Proqram təminatı sisteminin modullarının müstəqilliyi 2 meyar vasitəsilə ölçülə bilər: uyğunluq və birləşmə.

- Uyğunluq modul daxilindəki müxtəlif funksiyalar arasında əlaqənin gücünün ölçüsüdür;

- Birləşmə proqram daxilindəki modullar arasındakı əlaqənin gücünün ölçüsüdür.



Şək. 2.5 İki modul arasında birləşmə və uyğunluq

Refaktorinq. Refaktorinq, mövcud kodun daxili strukturunu və dizaynını yaxşılaşdırmaq üçün proqram təminatının hazırlanmasında istifadə olunan bir prosesdir. Bu proses kodun xarici davranışını dəyişmədən daxili strukturunu təkmilləşdirir. Oxunma qabiliyyətini artırmaq, mürəkkəbliyi azaltmaq və davamlılığı yaxşılaşdırmaq üçün kod bazasında kiçik, artımlı dəyişikliklərin edilməsini nəzərdə tutur. Refaktorinq vasitəsilə proqramçılar təkrarlanma və ya həddindən artıq mürəkkəb məntiqli kodları aradan qaldıra və kodun ümumi keyfiyyətini yaxşılaşdıra bilərlər. Proqramçılar müntəzəm olaraq refaktorinq etməklə texniki borcların yığılmasının qarşısını ala, yeni funksiyalar əlavə etməyi və proqram təminatına gələcək dəyişiklikləri asanlaşdıra bilərlər.

Refaktorinqin məqsədləri aşağıdakılarda ifadə edilir (Fowler, 2018) :

- Refaktorinq proqram təminatının dizaynını təkmilləşdirir;
- Refaktorinq proqram təminatının başa düşülməsini asanlaşdırır;
- Refaktorinq səhvləri tapmağa kömək edir;
- Refaktorinq daha sürətli proqramlaşdırmaya kömək edir.

Dizayn nümunələri. Dizayn nümunələri proqram təminatının inkişafında rast gəlinən ümumi dizayn problemlərinin sübut edilmiş həlləridir. Proqramçılar bu

dizayn nümunələrindən istifadə edərək yaxşı qurulmuş, təkrar istifadə edilə bilən həllər qəbul etməklə mürəkkəbliyi azalda bilərlər. Dizayn nümunələri ən yaxşı təcrübələri təşviq edən və kodun keyfiyyətini yaxşılaşdıran ortaq lüğət və təlimatlar toplusunu təmin edir. Şəkil 2.6-da dizayn nümunələri göstərilmişdir.

Şək. 2.6 Dizayn nümunələri

Bu dizayn nümunələrindən istifadə etməklə proqramçılar mürəkkəbliyi azalda və kodun davamlılığını yaxşılaşdıra bilərlər.

Abstraksiya. Abstraksiya mürəkkəbliyi azaltmaq və kodu daha başa düşülən etmək üçün istifadə olunan üsullardan biridir. Abstraksiya obyektin və ya komponentin mürəkkəb daxili strukturlarını gizlədərək, yalnız zəruri detalları göstərməsini təmin edir. Bu da o deməkdir ki, proqamçılar uyğun komponentlərdən istifadə edərkən daxili iş məntiqini tam başa düşməyə məcbur deyillər, onları yalnız həmin komponentin təmin etdiyi interfeyslər və funksiyalar maraqlandırır. Abstraksiya proqram təminatının hazırlanması prosesində iki şəkildə istifadə olunur.

1. Məlumatın abstraksiyası. Məlumatın abstraksiyası, məlumatların mürəkkəb daxili strukturunu gizlədir və yalnız zəruri məlumatların görünməsini təmin edir. Yəni ki, məlumatların necə saxlandığı, işləndiyi və s. Təfərrüatlar gizlədilərək yalnızca istifadəçiyə təqdim olunan hissələrlə məşğul olunur.

2. Əməliyyatın abstraksiyası. Əməliyyatın abstraksiyası, komponentin necə işlədiyinə dair detalları gizlədərək, yalnız onun təklif etdiyi funksionallığı göstərir.

Abstraksiya proqram təminatının hazırlanması prosesində kod mürəkkəbliyini azaltmaq üçün güclü vasitədir, çünki o, proqram təminatının müxtəlif komponentləri arasında asılılığı azaldır və kodu təkrar istifadə edilə bilən edir.

Düzgün adlandırma. Adlandırma kodun anlaşıqlılığını artırmaq üçün istifadə olunan ən əsas və ən sadə texnikalardan biridir. Düzgün və açıqlayıcı adlar, koda ilk baxışda onun nəyi təmsil etdiyini və nə etdiyini aydın şəkildə çatdırır. Düzgün adlandırma üçün diqqət edilməli bəzi məqamlar bunlardır.

- Siniflərin, metodların, dəyişənlərin və digər komponentlərin mənalı ada sahib olması vacibdir;

- Adların qısa, anlaşıqlı və yığcam olmasına üstünlük verilməlidir. Çünki uzun və mürəkkəb adlar kodun oxunmasını çətinləşdirə bilər;

- Adlandırma zamanı ingilis dilindən istifadə etmək lazımdır;

Şərhlər. Şərhlər koda nəzər keçirildiyi zamanı onun başa düşülməsi üçün mühüm rol oynayır. Yaxşı yazılmış şərhlər, kodun niyə o şəkildə yazıldığını, necə çalışdığını və onun məhdudiyyətlərini izah edə bilər. Şərh yazarkən diqqət edilməli məqamlar aşağıdakılardır;

- Şərhləri daha çox kodun daha mürəkkəb və ya çətin başa düşüldüyü yerlərə yazmaq vacibdir;

- Şərhlər kodun nə etdiyi və niyə o strukturda yazıldığını izah etməlidir;

- Şərhlər yenilənməlidir. Kod üzərində dəyişlik edərkən mütləq şərhlərin da yenilənməsi vacibdir;

- Həddindən artıq şərh yazmaqdan qaçınılmalıdır.

Avtomatlaşdırılmış testlər və davamlı inteqrasiya. Avtomatlaşdırılmış testlər, proqramın funksionallığını, performansını və təhlükəsizliyini yoxlamaq üçün avtomatik olaraq hazırlanmış test skriptləri və alətləri istifadə etməklə proqramın davamlı olaraq test edilməsini təmin edir. Bu proses, proqram təminatının

hazırlanması prosesində müəyyən bir səviyyədə təhlükəsizlik üçün kritik bir elementdir. Avtomatlaşdırılmış testlər, proqramın müxtəlif fəaliyyət və məlumat axını altında düzgün bir şəkildə işləyib işləmədiyini yoxlmağa kömək edir.

Davamlı inteqrasiya proqram təminatının hazırlanması kod dəyişikliklərinin müəyyən edilib birləşdirilməsini və test edilməsini təmin edir. Bu proses tez-tez və avtomatik olarak yerinə yetirilərək, dəyişikliyiklərin avtomatik olarak kod bazasına yansımasını mümkün edir. Davamlı inteqrasiya səhvləri erkən aşkar etməyə imkan verir və kodun sabitliyini artırır. O, həmçinin proqramçılar arasında əməkdaşlığı təşviq edir və kod bazasının daim güncəl olmasını təmin edir.

Davamlı inteqrasiya və avtomatlaşdırılmış testlər üçün çoxsaylı alətlər və platformalar mövcuddur. Mürəkkəbliyin növünə və tələblərinə görə, müxtəlif alətlərdən istifadə olunur. Bunlardan ən çox istifadə edilənlər aşağıdakılardır.

Jenkins: Jenkins, java dilində yazılmış açıq mənbəli (ing. open source) davamlı inteqrasiya və yerləşdirmə alətlərindən biridir. Müxtəlif proqramlaşdırma dillərini və platformalarını dəstəkləyir. Jenkins, proqram təminatının hazırlanması proseslərini avtomatlaşdırmağa kömək edir, test avtomatlaşdırma alətləri və frameworklər üçün çoxsaylı hazır plaginlərdən istifadə etməyə imkan verir. O, llayihənin mənbə kodunun saxlandığı yerdən dəyişiklikləri aşkar edərək konfiqurasiya, qurmaq (build), sınaqdan keçirmək və yerləşdirmə kimi manual prosesləri avtomatik həyata keçirir və bu proseslərdədə baş verə biləcək nasazlıqları minimuma endirir.

GitLab CI/CD : GitLab CI/CD proqram təminatının hazırlanması proseslərinin avtomatlaşdırılması və sürətləndirilməsi üçün istifadə edilən alətdir. GitLab-ın bir hissəsi kimi təklif olunan xidmətdir və GitLab layihə depoları ilə inteqrasiya olunur. Proqramçılar GitLab CI/CD tərəfindən təmin edilən xüsusi YAML fayllarından istifadə edərək layihə üçün fərdiləşdirilmiş iş axınlarını müəyyən edə bilərlər. Bu iş axınları kod dəyişikliklərini aşkarlaya və avtomatik olaraq xüsusi addımları yerinə yetirə, sınaqdan keçirə və yerləşdirə bilir (deploy).

Travis CI: Travis CI xüsusilə açıq mənbə layihələri üçün populyar davamlı inteqrasiya və yerləşdirmə xidmətidir. O, GitHub ilə səmərəli şəkildə inteqrasiya edir və yeni kodun əlavə edilməsi kimi müəyyən hadisələri avtomatik birləşdirir və sınaqdan keçirir.

CircleCI : CircleCI, davamlı inteqrasiya və yerləşdirmə (CI/CD) platformasıdır. Proqram təminatının hazırlanması proseslərinin avtomatlaşdırılması və sürətləndirilməsi üçün istifadə olunur.

Selenium : Selenium , testerlərin proqram təminatının keyfiyyətini artırmaq üçün istifadə etdiyi avtomatlaşdırılmış alətdir. Müxətlif proqramlaşdırma dillərini dəstəkləyir və bir çox brauzerlərdə çalışa bilir. Bu alət, veb proqramların fərqli komponentlərini avtomatik yoxlama və test etmə imkanı verir. Selenium vasitəsilə testerlər manual test ssenarilərini asanlıqla yarada , həmçinin brauzerlə qarşılıqlı əlaqəni simulyasiya edə və istifadəçi təcrübəsini yoxlaya bilərlər. Bu alət, proqram təminatının ümumi keyfiyyətini, dayanıqlığını yaxşılaşdırmaqla yanaşı, səhvləri əvvəlcədən aşkar etməyə və düzəltməyə imkan verir.

JIRA : JIRA, kod bazasının idarə olunması , proyekt idarəetməsi və iş axınlarının idarə edilməsi üçün istifadə olunan platformadır. Davamlı inteqrasiya və avtomatlaşdırılmış test proseslərini də idarə etmək üçün pluginləri mövcuddur. Komandalar bu pluginlər vasitəsilə proqram təminatını effektiv şəkildə test edə bilərlər.

III FƏSİL. Proqram təminatlarında kod mürəkkəbliyinin komanda işinə təsiri

3.1. Mürəkkəbliyin komandanın dinamikasına təsiri və əməkdaşlıq

Kod bazasının mürəkkəbliyi, bu komandalar daxilində dinamikanın formalaşmasında mühüm rol oynayır. Kodun mürəkkəbliyi komanda dinamikasında ünsiyyətə, əməkdaşlığa, və məhsuldarlığa böyük təsir göstərir. Effektiv ünsiyyət uğurlu komanda dinamikasının təməl daşıdır.

- Kommunikasiyaların pozulması: Yüksək kod mürəkkəbliyi proqram təminatının inkişaf etdirilməsi komandaların daxilində kommunikasiyaların pozulmasına səbəb ola bilər. Proqramçılar fikirlərini ifadə etməkdə və ya bir-birlərinin kodunu başa düşməkdə çətinlik çəkə bilər, bu da ünsiyyətdə anlaşılmazlıqlara səbəb ola bilər. Nəticə etibarilə, proqramçılar problemin həlli və innovasiya üzərində fəal əməkdaşlıq etmək əvəzinə uzun müddət mürəkkəb kodu anlamağa vaxt sərf edə bilərlər. Üstəlik, kodun mürəkkəbliyi komanda daxilində bilik mübadiləsinə və ötürülməsinə mane ola bilər. Çünki proqramçılar mürəkkəb kodları araşdırmaq istəməyə və ya bunu effektiv etmək üçün təcrübəyə malik olmaya bilər;
- Komanda məmnuniyyəti: Kod mürəkkəbliyinin təsiri texniki çətinliklərdən kənara çıxır və komanda üzvlərinin əhval-ruhiyyəsinə və fəaliyyətinə təsir göstərir. Həddindən artıq mürəkkəb kod bazaları ilə məşğul olmaq, proqraçıları ruhdan sala və əsəbiləşdirə bilər. Mürəkkəb kodla uzun müddət işləmək komanda üzvləri arasında işdən ayrılma, məyusluq və tükənmə hisslərinə gətirib çıxara, motivasiya və iş həvəsini azalda bilər;
- Məhsuldarlıq və səmərəlilik: Kodun mürəkkəbliyi proqram təminatının inkişaf etdirilməsi komandalarının məhsuldarlığına və səmərəliliyinə əhəmiyyətli dərəcədə təsir göstərir. Mürəkkəb kod bazaları səhvlərə, xətalara daha çox meyllidir, qiymətli vaxt və resurslar sərf edən geniş təkmilləşdirmə və problemlərin aradan qaldırılması səylərini tələb edir. Bundan əlavə, kodun

mürəkkəbliyi kodun nəzərdən keçirilməsi və keyfiyyətin təminatı prosesini çətinləşdirir. Mürəkkəb kod bazalarının yaratdığı çətinlikləri dərk etməklə və onları azaltmaq üçün strategiyaları həyata keçirməklə komandalar innovasiya və uğura kömək edən daha əlverişli və ahəngdar iş mühiti yarada bilərlər;

- Kod mürəkkəbliyin komanda daxilində xüsusi bacarıq və biliyə ehtiyaca təsir etməsi: Mürəkkəb kod çox vaxt mürəkkəb alqoritmləri, mürəkkəb məlumat strukturlarını və müxtəlif texnologiyaların inteqrasiyasını əhatə edir. Bu, yalnız proqramlaşdırma dillərini mükəmməl bilən, həm də qabaqcıl proqramlaşdırma konsepsiyalarını tətbiq etməkdə mahir olan komanda üzvlərini tələb edir. Yüksək kod mürəkkəbliyi proqram arxitekturasının və dizayn nümunələrinin hərtərəfli başa düşülməsini tələb edir. Genişləndirilə bilən və davamlılılq qabiliyyətinə malik olan sistemlər dizayn etmək bacarığına malik komanda üzvləri mürəkkəbliyin effektiv idarə olunması üçün çox vacibdir. Çünki mürəkkəb kod bazalarını qabaqcadan görmək və həll etmək , çətin ola biləcək problemlər və boşluqlar yada bilər. Güclü analitik və problem həll etmə bacarığı olan insanlar əvəzolunmazdır, çünki onlar innovativ həllər hazırlaya və sistemli şəkildə çətin məsələlərin öhdəsindən gələ bilərlər. Kod daha mürəkkəbləşdikcə, ənənəvi test strategiyaları kifayət etməyə bilər. Komanda proqram təminatının etibarlı, səmərəli və təhlükəsiz olmasını təmin etmək üçün avtomatlaşdırılmış test, performans testi və təhlükəsizlik testi kimi qabaqcıl sınaq metodologiyalarında bacarıqlı şəxslərə ehtiyac duyur.

3.2. Mürəkkəblik və Layihə idarəetmə

Kodun mürəkkəbliyi və onun idarə edilməsi proqram təminatının hazırlanmasında nəzərə alınmalı vacib amillərdir. Kodun mürəkkəbliyi kodun konstruksiya, struktur, məntiqi və memarlıq mürəkkəbliyinin göstəricisidir. Kod nə qədər mürəkkəb olsa, onu düzəltmək və ya yeni funksiyalar əlavə etmək bir o qədər çətindir. Səhvlərin baş verməsinin qarşısını almaq onları tapmaq və düzəltməkdən

daha asan və qənaətcildir. Mürəkkəbliyi azaltmaqla biz proqram təminatının keyfiyyətini yaxşılaşdıra, texniki xidmət üçün vaxt və xərcləri azalda bilərik. Xüsusilə, proqram təminatının davamlılığına, yəni onun nasazlıq tapma, təmir etmə və ya təkmilləşdirmə qabiliyyətinə baxmaq. Mürəkkəblik əsas etibarilə baxımlılığa mənfi təsir göstərir. Proqram təminatı layihəsi menecerinin rolu proqram məhsulunu vaxtında və büdcə daxilində istehsal etməkdir. Menecer həm təmir xərclərini, həm də inkişaf xərclərini azaltmaq istəyir.

Mürəkkəb proqram təminatı layihələrinin idarə edilməsi onların uğurlarına əhəmiyyətli dərəcədə təsir edə biləcək çoxsaylı problemlərin həllini nəzərdə tutur.

Bu problemlər texniki, təşkilati və insan amillərini əhatə edir:

- Əhatə dairəsinin sürüşməsi: Layihənin əhatə dairəsi vaxta, resurslara və ya büdcəyə düzəlişlər etmədən ilkin parametrlərindən kənara çıxdıqda genişlənir. İnkişaf edən tələblər, maraqlı tərəflərin gözləntiləri və ya layihənin mürəkkəbliyinin lazımi səviyyədə qiymətləndirilməməsi səbəbindən mürəkkəb layihələrdə ümumi problemdir;
- Texniki borc: Texniki borc, qısamüddətli məqsədəuyğunluq üçün seçilmiş, lakin uzunmüddətli problemlər yaradan zəif proqram arxitekturasının və ya mürəkkəb proqram təminatı layihələrinin istənməyən sonudur. Texniki borcun idarə edilməsi və azaldılması mürəkkəb layihələrdə kodun keyfiyyətinin və davamlılığının pisləşməsinin qarşısını almaq üçün çox vacibdir;
- Risklərin idarə edilməsi: Mürəkkəb proqram təminatı layihələrində risklərin müəyyən edilməsi, qiymətləndirilməsi və azaldılması davamlı problemdir. Risklər texniki məsələlərdən tutmuş, təhlükəsizlik zəifliyinə və bazar dinamikasına qədər dəyişə bilər. Effektiv risklərin idarə edilməsi layihənin uğuru üçün vacibdir.

Layihənin idarə edilməsi proqram layihələrində mürəkkəbliyin mənfi təsirlərinin azaldılmasında mühüm rol oynayır. Layihələr miqyası böyüdükcə, onların idarə olunması ilə bağlı çətinliklər yaranır. Effektiv layihə idarəetməsi bu çətinliklərin

öhdəsindən gəlməyə kömək edə bilər. Layihələrin mürəkkəbliyi artdıqca, daha yüksək xərclər, gecikmələr, keyfiyyətin aşağı düşməsi və maraqlı tərəflərin məmnunluğunun azalması kimi bir sıra problemlərlər yaranır.

Layihənin idarə edilməsinin mürəkkəbliyin mənfi təsirlərini azaltmağa kömək edən əsas yolları:

- Risklərin aradan qaldırılması: Layihənin idarə edilməsinin əsas komponentlərindən biri risklərin müəyyən edilməsi, təhlili və azaldılmasıdır. Mürəkkəb layihələr ölçüsünə, əhatə dairəsinə və hissələrinin qarşılıqlı asılılığına görə daha çox risk daşıyır. Sistematik risklərin idarə edilməsi prosesləri vasitəsilə layihə menecerləri potensial problemləri qabaqcadan görə bilər və layihəyə mənfi təsir etməzdən əvvəl onları həll etmək üçün strategiyalar həyata keçirə bilərlər;
- Resursların bölgüsü: Mürəkkəb layihələr çox vaxt insan resursları, büdcələr və texnologiyalar da daxil olmaqla əhəmiyyətli resursları əhatə edir. Layihənin idarə edilməsi bu resursların səmərəli, effektiv şəkildə bölüşdürülməsini təmin edir və məhsuldarlığı maksimuma çatdırır;
- Uyğunlaşma: Mürəkkəb layihələrin tez-tez əhatə dairəsi, resursları və məqsədləri dəyişikliyə məruz qalır. Layihənin idarə edilməsi metodologiyaları, layihəni pozmadan bu dəyişikliklərə uyğunlaşmaq üçün çeviklik təmin edir. Bu uyğunlaşma mürəkkəb layihələrin qeyri-müəyyənliklərini və dinamikasını idarə etmək üçün əsas nüanslardan biridir;
- Keyfiyyətə nəzarət: Yekun nəticələrin keyfiyyətinin təmin edilməsi layihənin idarə edilməsinin digər mühüm roludur. Davamlı keyfiyyət təminatı təcrübələri və metodologiyaları vasitəsilə layihə menecerləri layihənin bütövlüyünü qoruya, maraqlı tərəflərin gözləntilərini qarşılaya və ya üstələyə, layihənin lazımı standartlara və qaydalara uyğun olmasını təmin edə bilərlər.

3.3. Keys tədqiqatları və emprik sübutlar

Kodun mürəkkəbliyi kontekstində həm keys tədqiqatları, həm də empirik sübutlar proqram təminatının inkişafının incəliklərini və onun layihə nəticələrinə təsirini başa düşməkdə mühüm rol oynayır. Onlar mürəkkəb hadisələrə dair dəyərli fikirlər təqdim edir, nəzəriyyənin inkişafına və sınaqdan keçirilməsinə töhfə verir, qərarların qəbulu və praktikasına məlumat verir, bilik və anlayışı inkişaf etdirir. Ciddi metodologiyanı real dünyada müşahidə və təcrübə ilə birləşdirərək, tədqiqatçılar praktiki əhəmiyyətə və nəzəri əhəmiyyətə malik etibarlı, təsirli fikirlər yarada bilir. Bu yanaşmaların inteqrasiyası kodun mürəkkəbliyinin hərtərəfli başa düşülməsinə gətirib çıxara bilər və onu effektiv idarə etmək üçün strategiyaları məlumatlandıra bilər.

Keys tədqiqatları: Keys tədqiqatları və empirik sübutlar proqram təminatının inkişafı və layihənin idarə edilməsi də daxil olmaqla müxtəlif sahələrdə tədqiqatın mühüm komponentləridir. Tədqiqatçılara kod strukturu, arxitektura, inkişaf prosesləri və layihənin nəticələri kimi amilləri araşdıraraq xüsusi proqram layihələrinin mürəkkəbliklərini dərindən araşdırmaq imkanı verir.

Keys tədqiqatları real dünya mühitində xüsusi layihələri, tətbiqləri və ya sistemləri araşdıraraq kod mürəkkəbliyinin kontekstual başa düşülməsini təklif edir. Bu kontekstdə kod mürəkkəbliyinə təsir edən layihə tələbləri, komanda dinamikası, inkişaf prosesləri və texnologiya yığınları kimi amillər daxildir.

Mürəkkəb məsələlərin tədqiqi: Keys tədqiqatları kodun mürəkkəbliyini və onun mənfi təsirlərini azaltmaq üçün istifadə olunan strategiyalar haqqında məlumat verir. Tədqiqatçılar komandaların refaktorinq, modullaşdırma, kod təhlili və digər üsullar vasitəsilə mürəkkəb kod bazalarına necə müraciət etdiyini araşdıra bilərlər. Keys tədqiqatları yalnız kəmiyyət metodlarından istifadə etməklə asanlıqla öyrənilə bilməyən mürəkkəb məsələləri və ya hadisələri araşdırmaq üçün xüsusilə faydalıdır. Onlar tədqiqatçılara kəmiyyət təhlillərində ortaya çıxmayan nüansları, ziddiyyətləri və gözlənilməz nəticələri üzə çıxarmağa imkan verir.

Empirik sübutlar: Empirik tədqiqatlar tez-tez tsiklomatik mürəkkəblik, kod xətləri və ya birləşmə meyarları kimi mürəkkəbliyi ölçmək üçün kod ölçülərinin kəmiyyət

təhlilini əhatə edir. Bu tədqiqat sistemli müşahidəyə və ciddi metodologiyaya əsaslanması ilə xarakterizə olunur. Empirik tədqiqat obyektivliyi və etibarlılığı təmin edərək məlumatların toplanması, təhlili və şərhi üçün sistematik metodlara əməl edir. Bu, tədqiqat nəticələrinin etibarlılığını artırır və digər tədqiqatçılar tərəfindən təkrarlanma və yoxlamaya imkan verir.

Empirik sübutlar proqram təminatının inkişafında kod mürəkkəbliyini başa düşmək və idarə etməkdə mühüm rol oynayır. Empirik sübutlar müxtəlif ölçülər vasitəsilə kodun mürəkkəbliyinin kəmiyyətcə ölçülməsinə imkan verir. Bu ölçülər kodun mürəkkəbliyinin obyektiv ölçülərini təmin edərək tədqiqatçılara müxtəlif kod bazalarını təhlil etməyə, müqayisə etməyə, mürəkkəbliyin proqram təminatının keyfiyyətinə və davamlılığına təsirini qiymətləndirməyə imkan verir.

Empirik tədqiqatlar kod mürəkkəbliyi ölçüləri ilə müxtəlif proqram atributları arasındakı əlaqəni araşdıra bilər. Tədqiqatlar göstərdi ki, çox mürəkkəb kodda qüsurlar və səhvlər olma ehtimalı daha yüksəkdir və bu, texniki xidmət səylərinin artmasına və proqram təminatının etibarlılığının azalmasına səbəb olur. Empirik məlumatları təhlil edərək, kodun mürəkkəbliyi ölçüləri ilə bu proqram atributları arasında korrelyasiyaları müəyyən edə, mürəkkəbliyin ümumi proqram təminatının keyfiyyətinə və layihənin uğuruna təsiri barədə məlumat əldə edə bilərik.

3.4. Komandalarda mürəkkəbliyin idarə edilməsi

Komandalarda mürəkkəbliyin idarə edilməsi komanda mühitində yaranan müxtəlif mürəkkəbliklərin müəyyən edilməsi, başa düşülməsi və həlli prosesinə aiddir. Buraya həm yerinə yetirilən işlə bağlı texniki mürəkkəbliklər, həm də komandanın dinamikası ilə bağlı şəxsiyyətlərarası mürəkkəbliklər daxildir. Komandalarda, xüsusən də proqram təminatının işlənib hazırlanmasında mürəkkəbliyin idarə edilməsi texniki strategiyaların və şəxsiyyətlərarası bacarıqların birləşməsini tələb edir.

Komandalarda kod mürəkkəbliyini idarə etmək üçün aşağıdakı addımlara diqqət yetirmək lazımdır:

1. Ümumi kod standartlarını təyin edin.
2. Modul dizaynı təşviq edin.
3. Müntəzəm olaraq kodu nəzərdən keçirin.
4. Mövcud kodu refaktor edin.
5. Komanda üzvlərini maarifləndirin və bilik mübadiləsini təşviq edin.
6. Avtomatlaşdırma vasitələrindən istifadə edin.
7. Ünsiyyət və əməkdaşlığı artırın.

NƏTİCƏ

Bu tədqiqat proqram kodlarının mürəkkəbliyini ölçmək üçün istifadə olunan üsulları qiymətləndirmək üçün aparılmışdır. Tədqiqatın nəticələri göstərir ki, proqram təminatı keyfiyyətinin mühüm komponenti olan kod mürəkkəbliyinin dəqiq ölçülməsi və qiymətləndirilməsi proqram təminatının hazırlanması prosesində mühüm rol oynayır.

Proqram təminatının keyfiyyətinə təsir edən amillərin təhlili, onun inkişaf prosesində hansı sahələrə diqqət yetirilməli olduğunu müəyyən etmək üçün vacib bir addımdır. Bu amilləri başa düşmək proqramçılara, proqram təminatının keyfiyyətinin yaxşılaşdırılması strategiyalarını optimallaşdırmağa və resurslardan ən səmərəli şəkildə istifadə etməyə kömək edə bilər.

Bu tədqiqatda kod mürəkkəbliyini qiymətləndirmək üçün istifadə oluna müxtəlif mürəkkəblik meyarları araşdırıldı. Belə bir nəticəyə gəlindi ki, kod mürəkkəbliyi çoxşaxəli anlayışdır və tək bir ölçmə metodu bütün mürəkkəblik elementlərini əhatə etməyə bilər.

Bundan əlavə, bu tədqiqata araşdırılan ölçmə üsullarının proqram təminatının hazırlanması prosesində qarşılaşılan praktiki çətinlikləri həll etmək üçün kifayət etmədiyi müşahidə edilmişdir. Buna görə də, gələcək tədqiqatlar daha əhatəli və effektiv ölçmə metodlarının və modellərinin işlənib hazırlanmasına yönəldilməlidir.

Nəticə olaraq, bu tədqiqat proqram təminatı keyfiyyəti və kod mürəkkəbliyi haqqında dərin anlayış təmin etdi. Gələcək tədqiqatlar, proqram təminatının hazırlanması müddətində keyfiyyətin idarə edilməsi və kod mürəkkəbliyini azaltma strategiyalarını təkmilləşdirilməsinə yönəldilməlidir. Bu, sənayedə daha etibarlı və davamlı olan proqram təminatı hazırlanmasına tövhə verəcək.

İSTİFADƏ OLUNMUŞ ƏDƏBİYYAT SİYAHISI

All Dallal, J., & Briand, L. C. (2012). A precise method-method interaction-based cohesion metric for object-oriented classes. ACM Transactions on Software Engineering and Methodology (TOSEM), 21(2), 1-34. https://doi.org/10.1145/2089116.2089118

Al-Saqqa, S., Sawalha, S., & AbdelNabi, H. (2020). Agile software development: Methodologies and trends. International Journal of Interactive Mobile Technologies, 14(11). https://doi.org/10.3991/ijim.v14i11.13269

Aroral, H. K. (2021). Waterfall process operations in the fast-paced world: project management exploratory analysis. International Journal of Applied Business and Management Studies, 6(1), 91-99. https://www.ijabms.com/wp-content/uploads/2021/05/05_ARORAL_PB.pdf

Badri, L., & Badri, M., A Proposal of a new class cohesion criterion: an empirical study, Journal of Object Technology, 2004, 145-159, 3(4). https://doi.org/10.5381/jot.2004.3.4.a8

Bhatia, S., & Malhotra, J. (2014). A survey on impact of lines of code on software complexity. 2014 International Conference on Advances in Engineering & Technology Research (ICAETR-2014), 1-4. https://doi.org/10.1109/ICAETR.2014.7012875

Boehm, B. W. (1978). Characteristics of Software Quality. (First ed., Vol. 1). North Holland Publishing Company.

Boehm, B. W., Improving software productivity, Computer, 1987, 43-57, 20(09). https://doi.org/10.1109/MC.1987.1663694

Card, D. N., Clark, T. L., & Berg, R. A. (1987). Improving software quality and productivity. Information and Software Technology, 29(5), 235-241. https://doi.org/10.1016/0950-5849(87)90343-0

Chiang, I. R., & Mookerjee, V. S. (2004). Improving software team productivity. Communications of the ACM, 47(5), 89-93.

https://doi.org/10.1145/986213.986217

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design. IEEE Transactions on Software Engineering, 20(6), 476-493. https://doi.org/10.1109/32.295895

Cohn, M., & Ford, D. (2003). Introducing an agile process to an organization [software development]. Computer, 36(6), 74-78. https://doi.org/10.1109/MC.2003.1204378

Counsell, S., Liu, X., Eldh, S., Tonelli, R., Marchesi, M., Concas, G., & Murgia, A. (2015, August). Re-visiting the 'maintainability index' metric from an object-oriented perspective. In 2015 41st Euromicro Conference on Software Engineering and Advanced Applications (pp. 84-87). IEEE. https://doi.org/10.1109/SEAA.2015.41

Crosby, P. B., Quality Is Free, McGraw-Hill, 1979

Fowler, M. (2018). Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional. https://silab.fon.bg.ac.rs/wp-content/uploads/2016/10/Refactoring-Improving-the-Design-of-Existing-Code-Addison-Wesley-Professional-1999.pdf

Grinwald, R., Harel, E., Orgad, M., Ur, S., & Ziv, A. (1998, May). User defined coverage—a tool supported methodology for design verification. In Proceedings of the 35th annual Design Automation Conference (pp. 158-163). https://doi.org/10.1109/DAC.1998.724458

Halstead, M. H. (1977). Elements of Software Science. Elsevier Science Inc.

https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Humphrey, W. S. (1989). Managing the Software Process. Addison-Wesley Longman Publishing Co., Inc.

ISO/IEC, 25019:2023 Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE). https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25019:ed-1:v1:en

ISTQB. International Software Testing Qualification Board (version 2.0), 2007. www.istqb.org

Qumer, A., & Henderson-Sellers, B. (2008). A framework to support the evaluation, adoption and improvement of agile methods in practice. Journal of Systems and Software, 81(11), 1899-1919. https://doi.org/10.1016/j.jss.2007.12.806

Livermore, J. A. (2007, March). Factors that impact implementing an agile software development methodology. In Proceedings 2007 IEEE SoutheastCon (pp. 82-86). IEEE. https://doi.org/10.1109/SECON.2007.342860

McCabe, T.J. (1976). A complexity measure. IEEE Transactions on Software Engineering, (4), 308-320. http://doi.org/10.1109/TSE.1976.233837

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). Factors in software quality: Concepts and definitions of software quality. Rome Air Development Center, Air Force Systems Command. https://apps.dtic.mil/sti/pdfs/ADA049014.pdf

Mishra, D., & Mishra, A. (2011). Complex software project development: agile methods adoption. Journal of Software Maintenance and Evolution: Research and Practice, 23(8), 549-564. https://doi.org/10.1002/smr.528

Oman, P., & Hagemeister, J. (1992, January). Metrics for assessing a software system's maintainability. In Proceedings Conference on Software Maintenance 1992 (pp. 337-338). IEEE Computer Society. https://doi.org/10.1109/ICSM.1992.242525

Pargaonkar, S. (2023). A Comprehensive Research Analysis of Software Development Life Cycle (SDLC) Agile & Waterfall Model Advantages,

Disadvantages, and Application Suitability in Software Quality Engineering. International Journal of Scientific and Research Publications (IJSRP), 13(08). https://doi.org/10.29322/IJSRP.13.08.2023.p14015

Prykhodko, S., Prykhodko, N., & Smykodub, T. (2021). A statistical evaluation of the depth of inheritance tree metric for open-source applications developed in Java. Foundations of Computing and Decision Sciences, 46(2), 159-172. https://doi.org/10.2478/fcds-2021-0011

Shaheen, M. R., & du Bousquet, L. (2008, April). Relation between depth of inheritance tree and number of methods to test. In 2008 1st International Conference on Software Testing, Verification, and Validation (pp. 161-170). IEEE. https://doi.org/10.1109/ICST.2008.34

Shahid, M., Ibrahim, S., & Mahrin, M. N. R. (2011). A study on test coverage in software testing. Advanced Informatics School (AIS), Universiti Teknologi Malaysia, International Campus, Jalan Semarak, Kuala Lumpur, Malaysia. https://www.researchgate.net/publication/228913406_A_Study_on_Test_Coverage_in_Software_Testing

Aswini, S., & Yazhini, M. (2017). An assessment framework of routing complexities using LOC metrics. In 2017 Innovations in Power and Advanced Computing Technologies (i-PACT) (s. 1-6).

Shukla, R., & Misra, A.K. (2008, February). Estimating software maintenance effort: a neural network approach. In Proceedings of the 1st India Software Engineering Conference (pp. 107-112). https://doi.org/10.1145/1342211.1342232