ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

VOLUME-6, ISSUE-2

DASTURIY TA'MINOT MUHANDISLIGIDA SOLID DIZAYN TAMOYILLARI

Mahkamov Shohruh Sarvar o'g'li, Jizzax, O'zbekistan
Orcid: 0000-0003-0124-1749
e-mail: karoll19910112@gmail.com
Rakhmonkulov Feruz Pardaboyevich, Jizzakh, Uzbekistan
e-mail: feruz0123@gmail.com
Ezhumalai Perumal
e-mail: ezhumalaiperumal16@gmail.com

Annotatsiya: *Ushbu maqolada dasturiy ta'minot muhandisligida muhim o'rin tutuvchi SOLID dizayn tamoyillarining nazariy asoslari hamda ularning zamonaviy dasturiy tizimlarni ishlab chiqish jarayonidagi ahamiyati tahlil qilinadi. Dasturiy mahsulotlarning murakkabligi ortib borayotgan sharoitda kodning barqarorligi, kengaytiriluvchanligi va qayta foydalanish imkoniyatini ta'minlash dolzarb masalalardan biri hisoblanadi. Tadqiqot davomida SOLID tamoyillarining besh asosiy prinsipi - Single Responsibility, Open–Closed, Liskov Substitution, Interface Segregation va Dependency Inversion - alohida ko'rib chiqilib, ularning obyektga yo'naltirilgan dasturlashdagi amaliy qo'llanilishi misollar orqali yoritilgan. Shuningdek, ushbu tamoyillarni qo'llash dasturiy arxitektura sifatini oshirish, kodni modullarga ajratish, tizimni qo'llab-quvvatlashni soddalashtirish hamda dasturiy mahsulotni rivojlantirish jarayonini samarali tashkil etishga yordam berishi ko'rsatib berilgan.*
Kalit so'zlar: *SOLID tamoyillari, dasturiy ta'minot muhandisligi, obyektga yo'naltirilgan dasturlash, dasturiy arxitektura, kod sifati, modulli dasturlash.*
Abstract: *This article analyzes the theoretical foundations of the SOLID design principles and their importance in modern software engineering practices. As software systems become increasingly complex, ensuring maintainability, scalability, and reusability of code has become a significant challenge in software development. The study examines the five fundamental SOLID principles-Single Responsibility, Open–Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion-and demonstrates their practical application in object-oriented programming through illustrative examples. The research also highlights how the implementation of these principles contributes to improving software architecture, enhancing modularity, simplifying system maintenance, and supporting the sustainable development of software products.*
Keywords: *SOLID principles, software engineering, object-oriented programming, software architecture, code quality, modular programming.*

KIRISH

Dasturiy ta'minot muhandisligida "yaxshi kod" deganda ko'pincha faqat algoritmik to'g'rilik emas, balki uzoq muddatli boshqaruv xarajatlari ham nazarda tutiladi. Shu nuqtada maintainability (texnik xizmatga yaroqlilik) - tizimni mo'ljallangan ushlab turuvchilar tomonidan samarali va tejamkor tarzda o'zgartira olish darajasi - markaziy mezonga aylanadi.
Maintainability ko'pincha bo'linadigan xususiyatlar orqali muhokama qilinadi. Modullik (modularity), qayta foydalanish mumkinligi (reusability), tahlil qilish mumkinligi (analyzability), o'zgartirilishi mumkinligi (modifiability), sinovdan o'tkazilishi mumkinligi (testability) kabi tarkibiy ko'rsatkichlar bu sifatni "operatsionallashtirish"ga yordam beradi.
SOLID tamoyillari aynan shu joyda amaliy qiymat beradi va u kodni shunday strukturaga keltirishga undaydiki, o'zgarishlar "kaskad" bo'lib ketmasin; yangi talab qo'shish eski kodni buzib qo'yish ehtimolini oshirmasin; test yozish "behad qimmat" bo'lib qolmasin. Bu fikrlar Open–Closed Principle (OCP) va Dependency Inversion Principle (DIP) kabi tamoyillarda ravshan ko'rinadi.

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

VOLUME-6, ISSUE-2

Shuningdek, dizaynni “ko‘rinarli” qilish uchun UML kabi modellashtirish tili ishlatiladi. Object Management Group spetsifikatsiyasiga ko‘ra, UML - taqsimlangan obyekt tizimlari artefaktlarini vizuallashtirish, spetsifikatsiya qilish, konstruksiya qilish va hujjatlashtirish uchun grafik til spesifikatsiyasidir.

Usullar

Ushbu ish uch qatlamli metod bilan bajarildi.

Birinchidan, adabiyotlar tahlili qilindi: SOLID tamoyillarining klassik ta’riflari va muhandislik motivatsiyasi uchun Object Mentor resurslaridagi prinsip maqolalari hamda Robert Martinning “Design Principles and Design Patterns” bob/materiali tayanch sifatida qabul qilindi.

OCP tarixiy ildizi uchun Bertrand Meyerning “Object-Oriented Software Construction” asari keltiriladi.

LSP’ning formal (semantik) yadro g‘oyasi uchun Barbara Liskov va Jeannette Wingning subtyping haqidagi seminal maqolasi asosiy manba bo‘lib xizmat qiladi.

Ikkinchidan, refaktoring va amaliy keis uchun “xulq-atvorni saqlagan holda kichik o‘zgarishlar ketma-ketligi” tamoyiliga tayandik: refaktoring - mavjud kod bazasining dizaynini nazoratli ravishda yaxshilash bo‘lib, mohiyatan testlar/ishlayotgan holatni buzmaydigan mayda transformatsiyalar yig‘indisidir.

Uchinchidan, baholash (evaluation) uchun metrika va checklist birga qo‘llanadi. Metrika sifatida CK metrikalaridan CBO, WMC, LCOM tanlandi, chunki ular coupling va cohesion bilan bog‘liq dizayn muammolarini son jihatdan ko‘rsatib bera oladi.

Checklist esa kod review’da tez ishlaydigan sifatli diagnostika vositasi bo‘lib, metrikani “signal” sifatida, sharhni esa “izoh/faktlar” bilan to‘ldirishga mo‘ljallangan.

Tadqiqot cheklovi real repozitoriy, commit tarixi, testlar qamrovi, jamoa konteksti - unspecified. Shuning uchun “natijalar” bo‘limidagi o‘lchovlar metod namunasidir.

Natijalar

SOLID prinsiplari taqqoslanishi

1 - Jadval

Prinsip	Fokus (nimani himoya qiladi)	Amaliy savol (“sinaladigan” signal)	Tipik buzilish (smell)	Tez-tez ishlatiladigan yechim yo‘li
SRP	O‘zgarish sabablarini ajratish	“Bu klass nechta sabab bilan o‘zgaradi?”	“God class”, past cohesion	Extract Class, modul ajratish
OCP	Kengaytirish arzonligi	“Yangi tur qo‘shsam eski kodni tahrirlashim shartmi?”	switch/if portlashi	Polimorfizm, Strategy/plug-in
LSP	Subtyping kontrakti	“Subtype base o‘rnida muammosiz ishlaydimi?”	Type-check, kutilmagan exception	Kontraktni tekislash, kompozitsiya
ISP	Mijozni keraksiz metodga bog‘lamaslik	“Mijoz interfeys metodlarining ko‘pini ishlatmayaptimi?”	“Fat interface”	Interfeyslarni rol bo‘yicha bo‘lish
DIP	Bog‘liqlik yo‘nalishini teskari qilish	“High-level modul detalga qaram bo‘lib qolganmi?”	new bilan qattiq bog‘lanish	Abstraksiyalar + DI

Bu jadvaldagi g‘oyalar SRP/OCP/LSP/ISP/DIP’ning klassik bayonlari bilan mos keladi.

S prinsipi

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

VOLUME-6, ISSUE-2

S – SRP (Single Responsibility Principle). SRPning klassik bayoni “klass o‘zgarishi uchun bittadan ortiq sabab bo‘lmasligi kerak”, ya’ni bitta klass ko‘p “reason for change”ga xizmat qilsa, u bir nechta mas’uliyatni aralashtirgan bo‘ladi.

SRPning ratsionali juda “insonga yaqin”: muhandis sifatida biz “bugun GUI o‘zgardi” degan kichik o‘zgarish ertaga “geometriya hisob-kitobi ham qayta test qilinsin” degan qimmat jarayonga aylanib ketishini xohlamaymiz. SRP buzilganda, bir mas’uliyatdagi o‘zgarish boshqasiga “tasodifan” ta’sir qiladi va rebuild/retest/redeploy xarajati oshadi.

SRP ko‘pincha hamjihatlik (cohesion)ni oshiradi, biriktirish (coupling)ni pasaytiradi; bu esa maintainability sub-xususiyatlari (modularity/ analyzability/ modifiability/ testability)ni yaxshilashga xizmat qiladi.

Keng tarqalgan buzilishlar “hisoblaydi + saqlaydi + email yuboradi + log yozadi” kabi birikmalar yoki UI/persistence kabi detallar domain hisob-kitobi bilan bitta klassga tiqishtirilishi.

SRP (Java): hisoblash va xabar yuborish aralashgan holat, so‘ng ajratish

```java
// BEFORE: SRP buzilishi (hisoblash + email jo'natish bitta klassda)
class InvoiceService {

    private final SmtpClient smtp = new SmtpClient(); // infra detali shu yerda


    public double total(Invoice inv) {

        return inv.subtotal() + inv.tax();

    }


    public void email(Invoice inv, String to) {

        smtp.send(to, "Invoice", "Total=" + total(inv));

    }

}
```

```java
// AFTER: SRP-ga yaqin (mas'uliyatlar ajratildi)

final class InvoiceCalculator {

    public double total(Invoice inv) {

        return inv.subtotal() + inv.tax();

    }

}
```

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025


```
interface MailSender {
    void send(String to, String subject, String body);
}

final class InvoiceEmailer {
    private final InvoiceCalculator calc;
    private final MailSender mail;

    InvoiceEmailer(InvoiceCalculator calc, MailSender mail) {
        this.calc = calc;
        this.mail = mail;
    }

    public void email(Invoice inv, String to) {
        mail.send(to, "Invoice", "Total=" + calc.total(inv));
    }
}
```

O prinsipi

O – OCP (Open–Closed Principle). OCPning mashhur ta'rifi "dasturiy obyektlar (klasslar, modullar, funksiyalar…) kengaytirish uchun ochiq, lekin modifikatsiya uchun yopiq bo'lishi kerak." OCP atamasining tarixiy ildizlari Bertrand Meyer ishlariga borib taqaladi; Robert Martin OCP'ni muhandislik kolonkasi sifatida bayon qilarkan, Meyerning 1988 yildagi yondashuvini aniq tilga oladi.

Ratsional OCP yomon dizaynning tipik belgisini "kaskad o'zgarishlar" deb ko'rsatadi: bitta o'zgarish ko'plab bog'liq modullarga tarqalib ketadi va tizim fragil/rigid bo'lib qoladi. OCP bunga qarshi: ishlayotgan kodni "o'zgartirish" o'rniga xulqni "yangi kod qo'shish" bilan kengaytirishga undaydi.

Bu prinsipda regressiya xavfi kamayadi; release'lar barqarorroq; yangi talablar "extension point"lar orqali kiradi; testlash strategiyasi soddalashadi.

Keng tarqalgan buzilishlarga "if/else yoki switch portlashi" - har safar yangi tur qo'shish uchun markaziy funksiyani tahrirlashga majbur bo'lish.

OCP (Java): switch portlashi, so'ng Strategy orqali kengaytirish

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

VOLUME-6, ISSUE-2

```
// BEFORE: OCP buzilishi
class DiscountCalculator {
    public double discount(String tier, double amount) {
        return switch (tier) {
            case "SILVER" -> amount * 0.02;
            case "GOLD"   -> amount * 0.05;
            default       -> 0.0;
        };
    }
}
```

```
// AFTER: OCP-ga yaqin (yangi "tier" -> yangi klass)
interface DiscountPolicy {
    double discount(double amount);
}

final class SilverDiscount implements DiscountPolicy {
    public double discount(double amount) { return amount * 0.02; }
}

final class GoldDiscount implements DiscountPolicy {
    public double discount(double amount) { return amount * 0.05; }
}

final class DiscountService {
```

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025


```
  private final java.util.Map<String, DiscountPolicy> policies;


  DiscountService(java.util.Map<String, DiscountPolicy> policies) {

    this.policies = policies;

  }


  public double discount(String tier, double amount) {

    return policies.getOrDefault(tier, a -> 0.0).discount(amount);

  }

}
```

L prinsipi

L – LSP (Liskov Substitution Principle). LSP'ning muhandislik bayoni (Robert Martinning parafrazi): base klassga pointer/reference bilan ishlaydigan funksiyalar derived klass obyektlari bilan ham, buni bilmagan holda ishlay olishi kerak.

Bu g'oyaning formal-semantik tomoni "Subtype Requirement" sifatida: supertype haqida isbotlangan xossa subtype uchun ham saqlanishi kerak degan talab bilan ifodalanadi.

Ratsional LSP buzilsa, chaqiruvchi kod subtype'larni taniy boshlaydi (type-check, RTTI, if (x is Y)), natijada polimorfizm "yo'qoladi" va OCP ham bosim ostida qoladi. Robert Martin buni juda aniq bog'laydi: LSP'ga bo'ysunmaydigan funksiya yangi derived klass chiqqanda o'zgartiriladi - bu esa OCP'ga zid.

Ushbu prinsipda kontraktlar aniq bo'ladi; vorislash (inheritance) haqiqiy substitutability'ni beradi; testlar "base contract" darajasida yozilib, hamma subtype'lar uchun ishlaydi.

Keng tarqalgan buzilishlarga subtype metodlarida NotSupportedException/ UnsupportedOperationException; invariant buzilishi; "Square-Rectangle" kabi nozik semantik tuzoqlar.

LSP (Java): Rectangle/Square nozik buzilishi va kompozitsiya yo'li

```
// BEFORE: LSP buzilishi (klassik nozik misol)

class Rectangle {

  protected int w, h;

  public void setWidth(int w)  { this.w = w; }

  public void setHeight(int h) { this.h = h; }

  public int area() { return w * h; }
```

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025


```
}


class Square extends Rectangle {

    @Override public void setWidth(int w)  { this.w = this.h = w; }

    @Override public void setHeight(int h) { this.w = this.h = h; }

}


static void resize(Rectangle r) {

    r.setWidth(5);

    r.setHeight(10);

    // Rectangle uchun 50 kutiladi; Square bo'lsa 100 -> substitutability buziladi.

}
```

```
// AFTER: kompozitsiya / alohida Shape kontrakti

interface Shape { int area(); }


final class Rect implements Shape {

    private final int w, h;

    Rect(int w, int h) { this.w = w; this.h = h; }

    public int area() { return w * h; }

}


final class Sq implements Shape {

    private final int s;

    Sq(int s) { this.s = s; }
```

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

VOLUME-6, ISSUE-2

```
    public int area() { return s * s; }

}
```

I prinsipi

I – ISP (Interface Segregation Principle). ISPning to‘g‘ridan-to‘g‘ri ta’rifi: “mijozlar o‘zlari ishlatmaydigan interfeyslarga qaram bo‘lishga majbur qilinmasin.”

Ratsional “fat/polluted interface” paydo bo‘lsa, interfeysdagi bitta metodga o‘zgartirish kiritish hatto undan foydalanmaydigan klientlarni ham qayta kompilyatsiya/redeploy qilishga majbur qilishi mumkin. ISP buni “interfeyslarni klientlar ehtiyojiga mos bo‘lib bo‘lish” orqali yumshatadi.

Ushbu prinsipda coupling kamayadi; mocking/stubbing yengillashadi; interfeyslar “rol” tiliga yaqinlashadi (masalan, “faqat o‘qiydi”, “faqat yozadi”).

Keng tarqalgan buzilishlarha hamma narsani bitta interfeysga tiqish; ba’zi implementatsiyalar majburan “bo‘sh metod” yoki exception otishga o‘tishi.

ISP (Java): “semiz” interfeys va rol bo‘yicha ajratish

```
// BEFORE: ISP buzilishi

interface MultiFunctionDevice {

    void print(String text);

    void scan();

    void fax();

}



final class OldPrinter implements MultiFunctionDevice {

    public void print(String text) { /* ok */ }

    public void scan() { throw new UnsupportedOperationException(); }

    public void fax()  { throw new UnsupportedOperationException(); }

}
```

```
// AFTER: ISP-ga yaqin

interface Printer { void print(String text); }

interface Scanner { void scan(); }

interface Fax { void fax(); }
```

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

VOLUME-6, ISSUE-2

```
final class SimplePrinter implements Printer {
    public void print(String text) { /* ok */ }
}

final class OfficeMfd implements Printer, Scanner, Fax {
    public void print(String text) { /* ... */ }
    public void scan() { /* ... */ }
    public void fax()  { /* ... */ }
}
```

D prinsipi

D - DIP (Dependency Inversion Principle). DIPning klassik formulasi ikki banddan iborat.

(A) High-level modullar low-level modullarga bog‘liq bo‘lmasin, ikkalasi ham abstraksiyalarga tayanishi kerak.

(B) Abstraksiyalar detallarga bog‘liq bo‘lmasin, detallar abstraksiyalarga bog‘liq bo‘lsin.

Ratsional Robert Martin “inversion” so‘zi nega ishlatilishini ham izohlaydi. an’anaviy protseduraviy strukturalarda dependency ko‘pincha yuqoridan pastga oqadi, OO dizaynda esa yaxshi arxitektura bu oqimni “inverted” qiladi. Bu, uning talqinida, siyosat (policy) qatlamini detallar (mechanism/utility) o‘zgarishidan himoya qiladi.

Ushbu prinsipda high-level (biznes) logikani qayta ishlatish osonlashadi; testda detallarni almashtirish (mock/fake) qulay bo‘ladi; o‘zgarishlar “pastki qatlam”da bo‘lsa ham “yuqori qatlam”ni majburan o‘zgartirmaydi.

Amaliy ko‘prik ko‘p platformalarda DIP’ni joriy qilishning eng mashhur mexanizmi - Dependency Injection (DI). Masalan, Spring hujjatida DI obyektlar dependency’larini konstruktor/factory/property orqali “ta’riflash”, konteyner esa yaratish paytida ularni “inject” qilishi sifatida bayon qilinadi.

Xuddi shuningdek, Microsoftning .NET hujjatlari built-in service container va konstruktor injection mexanizmlarini tushuntiradi.

Keng tarqalgan buzilishlarga domain/service ichida new SqlRepository() yoki new SmtpClient(); yashirin global singleton; yuqori qatlamning pastki qatlam detallariga qattiq “compile-time” bog‘lanishi.

DIP (Java): concretionsga qattiq bog‘lanish va abstraksiya orqali ajratish

```
// BEFORE: DIP buzilishi (high-level modul detalni o'zi yaratadi)

final class OrderService {
```

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

VOLUME-6, ISSUE-2

```
    private final SqlOrderRepository repo = new SqlOrderRepository();


    public void place(Order order) {
        repo.save(order);
    }
}
```

```
// AFTER: DIP-ga yaqin
interface OrderRepository {
    void save(Order order);
}


final class SqlOrderRepository implements OrderRepository {
    public void save(Order order) { /* DB logic */ }
}


final class OrderService2 {
    private final OrderRepository repo;


    OrderService2(OrderRepository repo) { this.repo = repo; }


    public void place(Order order) {
        repo.save(order);
    }
}
```

Amaliy refaktoring keisi

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

VOLUME-6, ISSUE-2

Quyidagi mini-keis (domen/real tizim - unspecified) “checkout” jarayoniga o‘xshash ssenariy asosida qurilgan. “Oldin” holatda SRP, OCP va DIP buzilishlari bir joyga jamlangan, maqsad - refaktoring orqali dependency’ni ajratish va kengaytirish nuqtalarini tabiiy qilish. Refaktoring yondashuvi “mayda, xavfsiz, xulq-atvorni saqlovchi” o‘zgarishlar konsepsiyasiga mos keladi.

Keis (Java, oldin): SRP+OCP+DIP buzilishlari jamlangan checkout

```
final class CheckoutService {

   public void checkout(Order order, String paymentType) {

      // SRP buzilishi: hisoblash + to'lov + saqlash + email bitta metodda

      double total = order.subtotal() + order.tax();


      // OCP buzilishi: yangi payment turi -> shu yerga tegiladi

      if ("CARD".equals(paymentType)) {

         new CardProcessor().pay(total);    // DIP buzilishi

      } else if ("CASH".equals(paymentType)) {

         new CashProcessor().pay(total);    // DIP buzilishi

      } else {

         throw new IllegalArgumentException("Unknown payment");

      }


      new SqlOrderRepository().save(order);   // DIP buzilishi

      new SmtpClient().send(order.email(), "Receipt", "Total=" + total);

   }

}
```

Keis (Java, keyin): DIP + OCP + SRPga yaqinlashtirilgan checkout

```
interface PaymentProcessor { void pay(double amount); }


final class CardProcessor implements PaymentProcessor {
```

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

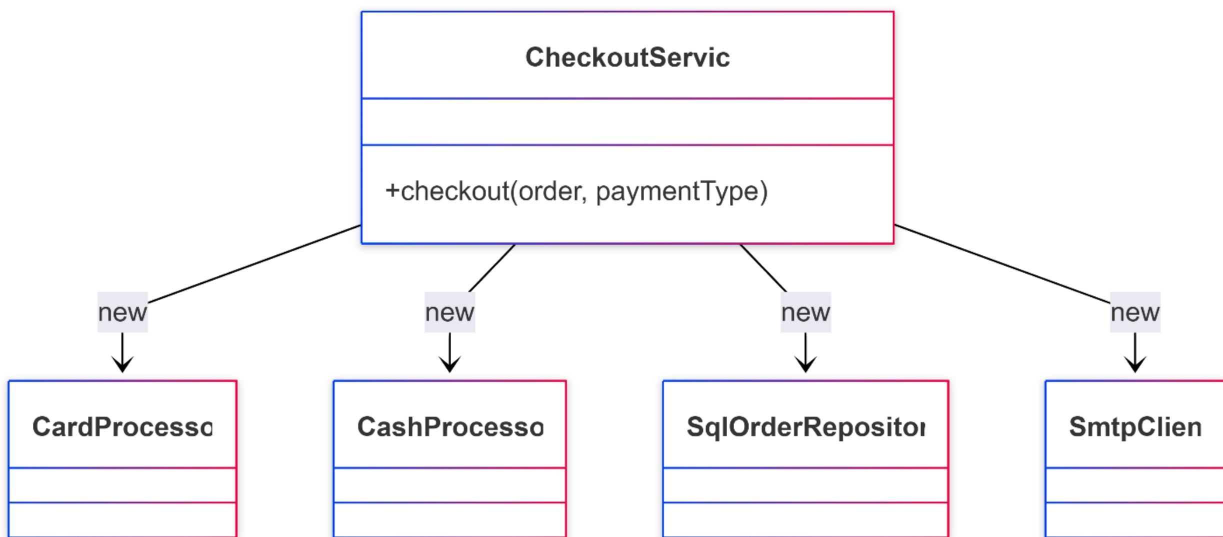
VOLUME-6, ISSUE-2

```
    public void pay(double amount) { /* ... */ }
}
final class CashProcessor implements PaymentProcessor {
    public void pay(double amount) { /* ... */ }
}

interface OrderRepository { void save(Order order); }
final class SqlOrderRepository implements OrderRepository {
    public void save(Order order) { /* ... */ }
}

interface MailSender { void send(String to, String subject, String body); }
final class SmtpMailSender implements MailSender {
    public void send(String to, String subject, String body) { /* ... */ }
}

/**
 * CheckoutService2: siyosat (policy) darajasi.
 * Detallar (repo/mail/payment impl) tashqaridan inject qilinadi.
 */
final class CheckoutService2 {
    private final OrderRepository repo;
    private final MailSender mail;
    private final java.util.Map<String, PaymentProcessor> payments;
```

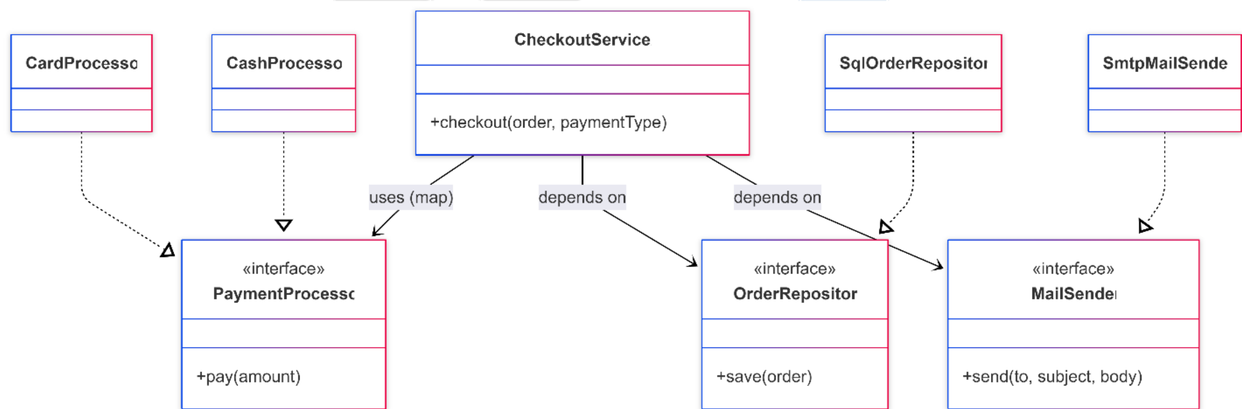
ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025


```
  CheckoutService2(OrderRepository repo, MailSender mail,
                   java.util.Map<String, PaymentProcessor> payments) {
    this.repo = repo;
    this.mail = mail;
    this.payments = payments;
  }

  public void checkout(Order order, String paymentType) {
    double total = order.subtotal() + order.tax();

    PaymentProcessor p = payments.get(paymentType);
    if (p == null) throw new IllegalArgumentException("Unknown payment");
    p.pay(total);

    repo.save(order);
    mail.send(order.email(), "Receipt", "Total=" + total);
  }
}
```

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025
THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY
VOLUME-6, ISSUE-2



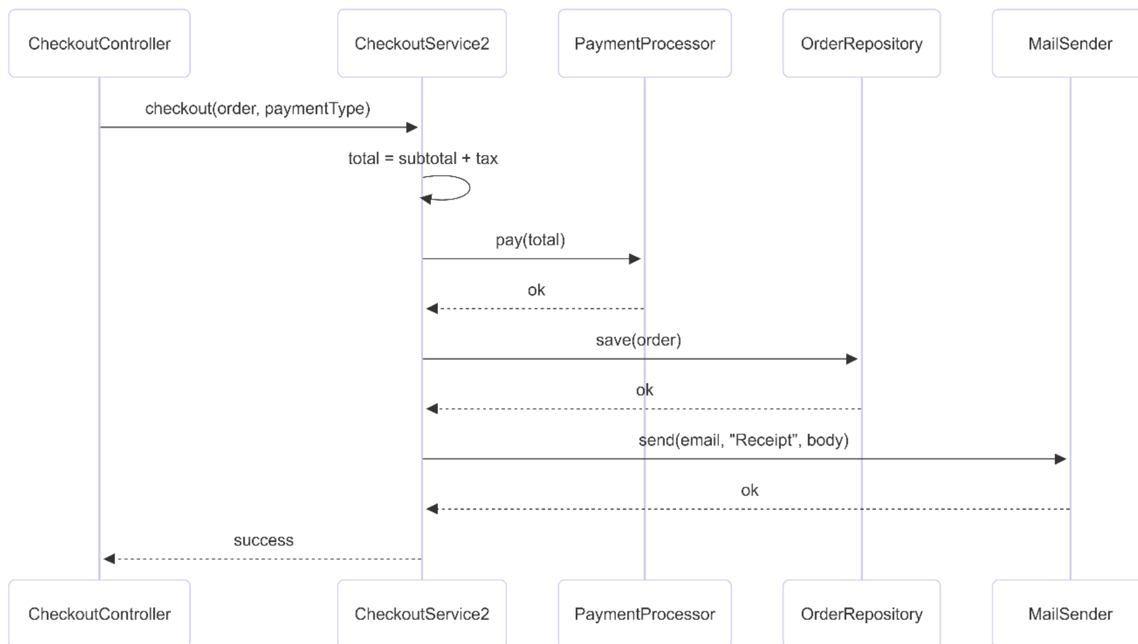
1-rasm. Keis (oldin). UML class diagram - high-level modul detallarni "new" qiladi



2-rasm. Keis (keyin). UML class diagram - abstraksiya orqali ajratilgan dependencylar

DIP ta'rifidagi "abstraksiyalarga tayanish" shu tarzda amaliy struktura sifatida ko'rinadi.

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025
THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY
VOLUME-6, ISSUE-2



3-rasm. Keis (keyin). UML sequence diagram - run-time hamkorlik

Sequence diagram "policy" va "detail" ajratilganini dinamik darajada ham ko'rsatadi bu DIP ratsionaliga mos.

SOLID compliance checklist va metrik baholash

2-jadval

Prinsip	Tekshiruv savollari (amaliy)
SRP	Klass nomini "va/and" bilan tushuntirayapsizmi? O'zgarish sabablari 2+ ta bo'lib ko'rinadimi?
OCP	Yangi tur qo'shish uchun mavjud switch/if'ni tahrir qilasizmi? "Kaskad o'zgarish" sezilyaptimi?
LSP	Subtype base kontraktini toraytiryaptimi (kutilmagan exception, invariant buzilishi)?
ISP	Interfeys "fat"mi: klient metodlarning katta qismini ishlatmayaptimi?
DIP	High-level modul concretions'ga compile-time bog'langanmi (new, static global)?

Checklistdagi OCP/LSP/ISP/DIP savollari prinsiplardagi aniq muammolar tasviriga tayangan.
Metrika nuqtayi nazari coupling va cohesion - maintainabilityga bevosita ta'sir qiladigan dizayn omillari sifatida ko'riladi, NIST taqdimotida maintainability zaifliklari orasida "excessive coupling" kabi holatlar ham tilga olinadi.
CK metrikalari ta'riflari (bu maqolada ishlatilgani):
CBO - klass nechta boshqa klass bilan "coupled" ekanini sanaydi.
WMC - klass metodlari soni/og'irligi bo'yicha murakkablikka yaqin ko'rsatkich (CK'da "weighted methods" sifatida ta'riflanadi).
LCOM - metodlar va ularning instans o'zgaruvchilarni bo'lishishi asosida cohesion yetishmasligini o'lchaydi.

Gipotetik (unspecified loyiha) o'lchovlar: oldin vs keyin

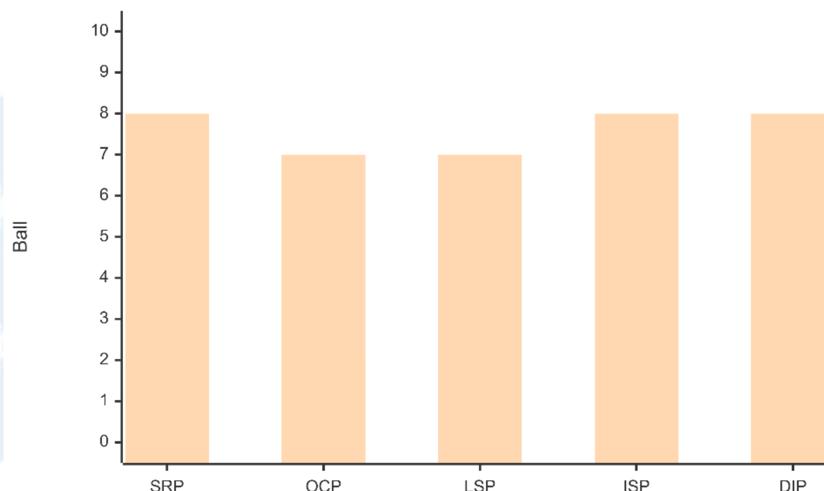
ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025
THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY
VOLUME-6, ISSUE-2

3-jadval

Ko‘rsatkich	Oldin (Kod 11)	Keyin (Kod 12)	Izoh (interpretatsiya)
CBO (Checkout servis)	7	3	Coupling pasaydi (DIP/ISP yo‘nalishi)
WMC (Checkout servis)	12	5	“God-service” yengillashdi (SRP ta’siri)
LCOM (Checkout servis)	18	7	Cohesion yaxshilandi (SRP)
switch/if branch soni	3	1	Kengaytirish “yangi implementatsiya”ga ko‘chdi (OCP)

(Eslatma: raqamlar namunaviy, metodologiyani ko‘rsatish uchun.)

Bu ko‘rsatkichlar talqini CK metrikalarining coupling/cohesion va murakkablik bilan bog‘liqligi haqidagi ta’riflarga mos keladi.



5-rasm. SOLID moslik ballari (0–10), gipotetik: oldin vs keyin (bar chart)

Ballar checklist + metrik signallar asosida gipotetik tarzda qo‘yilgan, real tizimda buni kod review protokoli va avtomatlashtirilgan statik tahlil bilan mustahkamlash tavsiya etiladi.

Muhokama

SOLID’ni amalda foydali qiladigan narsa - u “quruq nazariya” emas, balki o‘zgarishlar bilan yashaydigan kodning ichki iqtisodiyotini tushuntirib beradi. Masalan, OCP “eski kodni o‘zgartirma” demaydi u “eski kodni buzish riskini oshirmaydigan extension point qidir” deydi va bunu “kaskad o‘zgarishlar” misolida asoslaydi.

LSP va OCP o‘rtasidagi bog‘liqlik ayniqsa muhim. LSP buzilganda chaqiruvchi kod subtype’larni tanishga majbur bo‘ladi, natijada har yangi subtype - chaqiruvchi kodni modifikatsiya qilishni talab qiladi, bu esa OCP’ni sindiradi. Bu munosabat klassik LSP maqolasida ochiq aytiladi.

DIP esa ko‘p jamoalarda “faqat DI container ishlatish” bilan adashtiriladi. Aslida DIP - dependency yo‘nalishini arxitektura darajasida “policy detaldan yuqori turadi” degan tamoyilga moslashtirish. DI (Spring yoki .NET) esa bu tamoyilni joriy qilishning amaliy mexanizmlaridan biri, xolos.

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

THE MULTIDISCIPLINARY JOURNAL OF SCIENCE AND TECHNOLOGY

VOLUME-6, ISSUE-2

Metrikalar bo‘yicha ogohlantirish CBO/WMC/LCOM kabi ko‘rsatkichlar foydali, lekin ular dizayn niyatini 100% ushlamaydi. Shu sabab, metrikani “qizil bayroq” (signal), checklist va dizayn muhokamasini esa “diagnostika” sifatida birga qo‘llash ko‘proq ishonch beradi. CK maqolasining o‘zi ham metrikalar nazariy asoslari va baholash mezonlarini muhokama qiladi, bu metrikalarning “o‘lchash nazariyasi” bilan bog‘liq ehtiyotkorlikni eslatadi.

Maintainability nuqtayi nazaridan qaraganda, excessive coupling kabi holatlar ko‘pincha amaliy xarajatni oshiradi, NIST taqdimoti ham maintainability zaifliklari sifatida coupling bilan bog‘liq muammolarni tilga oladi. Shu sabab SOLID va metrikalarni birgalikda ko‘rish “nazariya + o‘lchov” uyg‘unligini beradi.

Xulosa va amaliy tavsiyalar

SOLID tamoyillari OOP tizimlarida strukturaviy qarorlarni boshqarish uchun qudratli “kompas” bo‘la oladi. Ular maintainability (modularity/ analyzability/ modifiability/ testability) bilan bog‘liq amaliy muammolarni kamaytirishga qaratilgan.

Quyidagi tavsiyalar real kod bazasi (unspecified) bo‘lsa ham umumiy qo‘llanadi:

Birinchi tavsiya: refaktoringni “katta portlash” emas, kichik qadamlar sifatida rejalashtiring, har qadamda xulq-atvor saqlanishi kerak (test, kompilyatsiya, smoke-check).

Ikkinchi tavsiya: SRP diagnostikasini jamoaviy odatga aylantiring. Agar bitta klassni tushuntirishda “va/and” so‘zi ko‘p ishlatilsa, yoki bir klassga turli stakeholder’lar (UI, DB, domain) birvarakay “buyruq” berayotgan bo‘lsa - mas’uliyatlarni ajrating.

Uchinchi tavsiya: OCP uchun “switch/if portlash nuqtalari”ni egallab oling. Yangi tur qo‘shish jarayoni eski kodni edit qilishga majbur qilsa, bu joyga Strategy/plug-in kabi kengaytirish mexanizmini kiriting, maqsad - ishlayotgan kodga tegmasdan xulq qo‘shish.

To‘rtinchi tavsiya: LSP’ni “vorislash bor ekan, hammasi joyida” deb qabul qilmang. Subtype’lar base kontraktini toraytirmasin, type-check/RTTI ko‘payishi LSP buzilganining kuchli signali. LSP buzilishi ko‘pincha OCP’ni ham yemiradi.

Beshinchi tavsiya: ISP’ni interfeys dizayni standartiga aylantiring. “mijozlar ishlatmaydigan metodga qaram bo‘lmasin”. Fat interface’larni rol bo‘yicha parchalash test yozishni ham yengillashtiradi.

Oltinchi tavsiya: DIP’ni arxitektura darajasida tasdiqlang high-level policy detallarni bilmasin. Amaliy mexanizm sifatida DI’dan (Spring yoki .NET) foydalanish mumkin, lekin DI - DIPning o‘zi emas, uni qo‘llash vositasidir.

Yettinchi tavsiya: o‘lchash (measurement)ni joriy qiling. Minimal “dashboard” sifatida CBO/WMC/LCOM’ni kuzatib boring: ular coupling/cohesion muammolarini tez ko‘rsatadi. Maintainability zaifliklaridan biri sifatida excessive coupling eslatilishi ham bu yondashuvni amaliy jihatdan qo‘llab-quvvatlaydi.

Foydalanilgan adabiyotlar

1. Robert C. Martin. *Design Principles and Design Patterns* (2000).
2. Robert C. Martin. *SRP: The Single Responsibility Principle* (Object Mentor PDF).
3. Robert C. Martin. *The Open-Closed Principle* (Object Mentor PDF), OCP va uning motivatsiyasi, shuningdek Meyerga ishora.
4. Bertrand Meyer. *Object-Oriented Software Construction* (OOSC, PDF).
5. Robert C. Martin. *The Liskov Substitution Principle* (1996, PDF).
6. Barbara Liskov, Jeannette Wing. *A Behavioral Notion of Subtyping* (1994, PDF).
7. Robert C. Martin. *The Interface Segregation Principle* (1996, PDF).
8. Robert C. Martin. *The Dependency Inversion Principle* (1996, PDF).
9. Shyam R. Chidamber, Chris F. Kemerer. *A Metrics Suite for Object Oriented Design* (1994, PDF) - CBO/WMC/LCOM ta’riflari.

ISSN: 2582-4686 SJIF 2021-3.261,SJIF 2022-2.889, 2024-6.875 ResearchBib IF: 10.25 / 2025

10. Martin Fowler. *Refactoring* (Refactoring kitobi sahifasi) - refaktoring ta'rifi va "small behavior-preserving" yondashuvi.
11. NIST (NIST CSRC'da joylashgan taqdimot): ISO/IEC 25010'ga mos sifat atributlari, maintainability sub-xususiyatlari va zaifliklar (masalan, coupling).
12. OMG: *UML 2.5.1 About UML* - UML ta'rifi va versiya ma'lumoti.
13. Spring Framework Reference: IoC/DI ta'rifi (dependency injection jarayoni).
14. Microsoft .NET: Dependency injection overview (konstruktor injection va built-in container).
15. Mahkamov, S. (2024). JAVA MVC ARXITEKTURASI: MODULYARLIK, XAVFSIZLIK VA PERFORMANCE MASALALARI. *SAMBHRAM XABARNOMASI*, *1*(1), 94-97.
16. Aynakulov, T., Mahkamov, S., & Ulashev, A. (2025, September). TASVIRLARDAN XUSUSIYATLARNI AVTOMATIK AJRATIB OLISH, OBYEKTLARNI ANIQLASH VA KLASSIFIKATSIYA QILISH. In *Scientific practical conference* (Vol. 1, No. 1, pp. 46-50).
17.