|  |
| --- |
| Level 4 | Programming |
| Abdulloh Xalilov |

**Unit 4:** Level 4 | Programming

**O‘qituvchi:** Ilyosxo`ja Ikromxo`jayev

**Guruh identifikatori:** 23-412 Software (C#)

**Talaba ID:** 231397

**Taqdim etilgan sana:** 1.05.2025 yil

**BTEC o'quvchilar topshiriqlarini baholash va deklaratsiya**

Baholash uchun ishlarni taqdim etganda, har bir o'quvchi ish o'ziniki ekanligini tasdiqlovchi deklaratsiyani imzolashi kerak.

|  |  |
| --- | --- |
| **O‘quvchi (talaba) identifikatori:** | 231397 |
| **Baholovchi nomi:** | Ilyosxo`ja Ikromxo`jayev |
| **BTEC dasturi nomi:** | Pearson BTEC Higher Nationals in Information Technologies |
| **Birlik yoki komponent raqami va nomi:** | Level 4 | Programming |
| **Topshiriq nomi:** | Level 4 | Programming |
| **Topshiriq topshirilgan sana:** | 1.05.2025yil |

Iltimos, har bir topshiriq uchun berilgan ishlarni sanab o'ting. Ishlarni topish mumkin bo'lgan sahifa raqamlarini ko'rsating yoki ishlarning mohiyatini tavsiflang (masalan, grafik, rasm).

|  |  |  |
| --- | --- | --- |
| **Topshiriq vazifasi ma'lumoti** | **Ishlar taqdim etildi** | **Sahifa** |
|  |  |  |
|  |  |  |
|  |  |  |

|  |
| --- |
| **O'quvchi deklaratsiyasi**  Men ushbu topshiriq uchun taqdim etilgan ish meniki ekanligini tasdiqlayman. Ishda foydalanilgan manbalarga aniq havola qildim. Men noto'g'ri deklaratsiya noto'g'ri ishlashning bir shakli ekanligini tushunaman.  **O‘quvchi imzosi:**  **Sana:** 1.05.2025 yil |

Mundarija

[KIRISH 3](#_Toc199699179)

[LO1: P1 6](#_Toc199699180)

[M1 8](#_Toc199699181)

[D1 9](#_Toc199699182)

[LO2: P2 12](#_Toc199699183)

[M2 15](#_Toc199699184)

[D2 19](#_Toc199699185)

[LO3: P3 22](#_Toc199699186)

[M3 24](#_Toc199699187)

[D3 27](#_Toc199699188)

[LO4: P4 30](#_Toc199699189)

[P5 32](#_Toc199699190)

[M4 34](#_Toc199699191)

[D4 37](#_Toc199699192)

[Xulosa 40](#_Toc199699193)

[ADABIYOTLAR RO'YXATI 43](#_Toc199699194)

KIRISH

Inson salomatligi va hayotining beqiyos qadriyati zamonaviy tibbiyot oldiga bemorlarga nafaqat samarali davolash usullarini taqdim etish, balki ularning operatsiyadan keyingi tiklanish jarayonlarini ham sifatli va xavfsiz tashkil etishdek murakkab vazifani qo'yadi (OM-1 da "Inson hayoti va salomatligi bebahodir" deya ta'kidlanganidek). Jarrohlik amaliyotlari ko'pincha bemor hayotini saqlab qolish yoki uning sifatini yaxshilash uchun zaruriy chora bo'lsa-da, davolash jarayoni operatsiya xonasida yakunlanmaydi; aksincha, operatsiyadan keyingi davr bemorning to'liq sog'ayishi va hayotga qaytishi uchun hal qiluvchi bosqich hisoblanadi. Biroq, aynan shu bosqichda bir qator muammolar va qiyinchiliklar yuzaga kelishi mumkin. Bemorning kasalxonada qolish muddati operatsiya turiga bog'liq bo'lib, imkon qadar qisqa bo'lishi maqsadga muvofiq sanalsa-da (OM-1), bu uning sog'lig'i uchun paradoksal ravishda yangi xavflarni keltirib chiqarishi mumkin: yotoqda uzoq vaqt harakatsiz qolish yotoq yaralar, venoz emboliya va tromboz kabi jiddiy asoratlarga, hatto insult yoki yurak xurujiga olib kelishi ehtimoldan xoli emas. Operatsiyadan so'ng bemor dastlab tez yordam xonasiga (E.R.) o'tkazilib, u yerda uning ahvoli doimiy ravishda maxsus jihozlar va malakali shifokorlar tomonidan nazorat qilinadi. Ammo, bemor E.R.ni tark etib, reabilitatsiya xonasiga o'tkazilgach va ayniqsa, kasalxonadan uyiga javob berilgandan so'ng, shifokor va bemor o'rtasidagi aloqaning susayishi yoki butunlay uzilishi (OM-1 "Kasalxonadan chiqqandan keyin bemor va shifokor o'rtasida aloqa yo'qligi"), kutilmagan alomatlarning o'z vaqtida va rasmiy tarzda qayd etilmasligi (OM-1 "Kutilmagan alomatlarning rasmiy qaydi yo'q") hamda bemorning o'z tiklanish jarayoni haqidagi noaniq his-tuyg'ulari (OM-1 "Bemorning tiklanish haqidagi noaniq his-tuyg'ulari") kabi muammolar bemorning to'liq sog'ayishiga to'sqinlik qilishi va jiddiy asoratlarning rivojlanishiga sabab bo'lishi mumkin. Mavjud cheklangan kasalxona sig'imlari va tibbiyot xodimlarining bandligi (OM-1) ham bu muammolarni yanada chuqurlashtiradi.

Aynan shu kabi dolzarb muammolarni hal etish va operatsiyadan keyingi tiklanish jarayonini optimallashtirish maqsadida "PATIENT’S RECOVERY SYSTEM" (PRS) nomli axborot tizimini ishlab chiqish taklif etilmoqda. Ushbu tizimning asosiy maqsadi – bemorning kasalxonada va undan keyin uy sharoitida tiklanish jarayonini samarali kuzatib borish, tibbiy xodimlarga bemor ahvoli haqida o'z vaqtida va aniq ma'lumotlar yetkazib berish, shuningdek, bemorlarga o'z sog'lig'ini boshqarishda faol ishtirok etish imkonini yaratish orqali nojo'ya ta'sirlar va asoratlarning oldini olishdir (OM-1 "Tashkilotning asosiy maqsadi va vazifasi nojo'ya ta'sirlar va asoratlarning oldini olish uchun tiklanish davrida bemorning sog'lig'ini kuzatish va nazorat qilishdir"). Taklif etilayotgan yechim bemor tomonidan qayd etilgan kasallik alomatlari va ko'rilgan choralarni o'zida saqlash, tana harorati, qon bosimi, EKG kabi muhim ko'rsatkichlarni monitoring qilish, shunga o'xshash holatlarda qo'llanilgan samarali davolash usullari haqida tibbiy xodimlarga ma'lumot berish, zaruriy qo'shimcha tekshiruvlarni eslatib turish kabi funksiyalarni o'z ichiga oladi. Eng muhimi, tizim bemor kasalxonadan chiqqandan keyin ham u bilan interaktiv aloqani (kuniga ikki yoki undan ko'p marta intervyu) davom ettirib, uning ahvolini masofadan nazorat qilish, noqulayliklar yuzaga kelganda shifokor bilan maslahatlashish zarurligini eslatish, ba'zi hollarda esa dori-darmonlarni almashtirish yoki yuzaga kelgan holat normal ekanligi haqida ma'lumot berish kabi imkoniyatlarni taqdim etadi. Shuningdek, tizim reabilitatsiya jarayonini ham kuzatib borishi, bemor va shifokorga uning borishi haqida ma'lumot berishi hamda shifokorga shunga o'xshash holatlarda samarali bo'lgan reabilitatsiya usullari bo'yicha ko'rsatmalar taqdim etishi ko'zda tutilgan. Har qanday favqulodda vaziyat yuzaga kelganda, tizim avtomatik ravishda tez yordam xizmatlarini chaqirishi ham mumkin. Bunday murakkab va mas'uliyatli tizimni ishlab chiqish, o'z navbatida, dasturiy ta'minot muhandisligi sohasidagi fundamental bilimlarni, xususan, samarali algoritmlarni aniqlash va ularni amalga oshirish (LO1), turli dasturlash paradigmalarining (protsessual, ob'ektga yo'naltirilgan, hodisalarga asoslangan) xususiyatlarini chuqur tushunish va ularni o'rinli qo'llay olish (LO2), integratsiyalashgan ishlab chiqish muhitlaridan (IDE) unumli foydalanish (LO3) hamda puxta diskretlash jarayonlarini amalga oshirish va qat'iy kodlash standartlariga rioya qilish (LO4) kabi ko'nikmalarni talab etadi. Ushbu hisobot aynan "PATIENT’S RECOVERY SYSTEM"ni ishlab chiqish jarayonining ana shu jihatlarini batafsil yoritishga qaratilgan.

LO1: P1

**Algoritm nima ekanligini ta'riflang va jarayonni tavsiflang**

**ilova yaratish**

Dasturiy ta'minotni ishlab chiqish sohasida algoritm tushunchasi markaziy o'rinni egallaydi. U, keng ma'noda, aniq belgilangan masalani yechish yoki muayyan natijaga erishish uchun bajarilishi lozim bo'lgan, cheklangan sondagi, tartiblangan amallar ketma-ketligi sifatida ta'riflanadi (Knuth, 1997). Har bir samarali algoritm bir nechta muhim xususiyatlarga ega bo'lishi kerak: aniqlik (har bir amalning bir ma'noli ifodalanishi), cheklanganlik (algoritmning cheklangan vaqt ichida yakunlanishi), kiritish (algoritmga beriladigan boshlang'ich ma'lumotlar), chiqarish (algoritm natijasida olinadigan qiymatlar) va samaradorlik (amallarning bajarilishi mumkinligi va maqsadga muvofiqligi) (Cormen et al., 2009). "PATIENT’S RECOVERY SYSTEM" loyihasi doirasida algoritmlar bemorning jarrohlik amaliyotidan keyingi tiklanish jarayonini kuzatish, tahlil qilish va boshqarish uchun asosiy vosita bo'lib xizmat qiladi. Misol uchun, bemorning hayotiy ko'rsatkichlarini (tana harorati, qon bosimi, EKG) doimiy monitoring qilish algoritmi ishlab chiqilishi mumkin; bu algoritm belgilangan me'yorlardan chetga chiqish holatlarini aniqlab, zudlik bilan tibbiy xodimlarni ogohlantiradi (OM-3 "Diagnose", "Monitor" vazifalari). Shuningdek, bemor tomonidan kiritilgan sub'ektiv shikoyatlar (masalan, og'riq darajasi, uyqu sifati) va uning tibbiy tarixi (OM-4 "Medical History") asosida potensial asoratlarni bashorat qilish yoki tavsiya etilgan davolash usullari (OM-1 "similar cases") haqida ma'lumot berish uchun ham alohida algoritmlar zarur bo'ladi. Ushbu tizim uchun dastur yaratish jarayoni bir necha uzviy bog'liq bosqichlardan iborat. Dastlab, loyihaning maqsadi va funksional imkoniyatlari aniqlanadi, bunda "PATIENT’S RECOVERY SYSTEM"ning asosiy muammosi – kasalxonadan chiqqandan keyin bemor va shifokor o'rtasidagi aloqaning yetishmasligi, kutilmagan alomatlarning rasmiy qayd etilmasligi (OM-1) kabi jihatlar chuqur tahlil qilinadi. Ushbu talablar (requirements) yig'indisi "Software Requirements Specification" (SRS) hujjatida o'z aksini topadi (Pressman and Maxim, 2015). Keyingi bosqichda, ushbu talablar asosida tizimning arxitekturasi, ma'lumotlar bazasining tuzilishi (masalan, MySQL yoki PostgreSQL) va foydalanuvchi interfeyslari (ham tibbiy xodimlar, ham bemorlar uchun) loyihalanadi, bunda MVC (Model-View-Controller) arxitekturasi kabi zamonaviy yondashuvlar qo'llanilishi mumkin. Loyihalash bosqichidan so'ng, tanlangan dasturlash muhitida (masalan, Java Spring Boot texnologiyalari to'plami) algoritmlar kod ko'rinishida implementatsiya qilinadi. Dasturiy ta'minotning alohida komponentlari ishlab chiqilgach, ular o'zaro integratsiya qilinadi va tizimning funksionalligi, ishonchliligi hamda samaradorligini tekshirish uchun turli darajadagi (birlik, integratsion, tizimli va qabul qilish) test sinovlari o'tkaziladi (Myers et al., 2011). Muvaffaqiyatli sinovlardan so'ng, tizim foydalanishga topshiriladi (deployment) va uning uzluksiz ishlashini ta'minlash, yuzaga kelishi mumkin bo'lgan xatoliklarni bartaraf etish hamda yangi talablarga moslashtirish maqsadida doimiy ravishda qo'llab-quvvatlanadi (maintenance) (Sommerville, 2016).

M1

Dasturiy ta'minotni yozma kod holatidan to uning bevosita bajarilishigacha bo'lgan jarayon bir nechta ketma-ket amalga oshiriladigan va o'zaro bog'liq bo'lgan texnik bosqichlarni o'z ichiga oladi; bu bosqichlar dasturning manba kodini mashina tushunadigan shaklga o'tkazish va uni ijro etish uchun zarurdir. "PATIENT’S RECOVERY SYSTEM" kabi Java platformasida ishlab chiqilayotgan tizim uchun bu jarayon quyidagicha kechadi: Dastlab, dasturchi tomonidan Java tilining sintaksis va semantikasiga muvofiq yozilgan manba kodi (*.java fayllari) maxsus dastur – kompilyator (Java Development Kit tarkibidagi javac) tomonidan qayta ishlanadi. Kompilyatsiya jarayonida kod bir necha analiz bosqichlaridan o'tadi: leksik analizator kodni tokenlarga (kalit so'zlar, identifikatorlar, operatorlar) ajratadi; sintaktik analizator ushbu tokenlar ketma-ketligining til grammatikasiga mos kelishini tekshirib, sintaktik daraxt hosil qiladi; semantik analizator esa kodning ma'no jihatdan to'g'riligini (masalan, o'zgaruvchilar turlarining mosligi, metodlarning to'g'ri chaqirilishi) nazorat qiladi (Aho et al., 2007). Java uchun kompilyatsiya natijasida platformadan mustaqil bo'lgan oraliq kod – bayt-kod (*.class fayllari) hosil bo'ladi. Ushbu bayt-kod bevosita protsessor tomonidan bajarila olmaydi, balki Java Virtual Mashinasi (JVM) uchun mo'ljallangan instruktsiyalar to'plamidir (Lindholm et al., 2014). Keyingi bosqich, agar dastur bir nechta alohida kompilyatsiya qilingan modullardan (masalan, turli .class fayllari yoki JAR arxivlari) yoki tashqi kutubxonalardan tashkil topgan bo'lsa, bog'lash (linking) jarayonini o'z ichiga olishi mumkin. Java'da bu asosan dinamik bog'lash shaklida, ya'ni dasturning ishlashi davomida JVM tomonidan kerakli klasslar yuklanganda amalga oshiriladi. Yuklash (loading) bosqichida JVMning maxsus komponenti – ClassLoader – kerakli .class fayllarini (bayt-kodni) xotiraga yuklaydi. Yuklangan bayt-kod JVM tomonidan yana bir bor tekshiruvdan o'tkaziladi (bytecode verification), bu esa xavfsizlikni ta'minlash (masalan, xotiraga ruxsatsiz murojaat qilishga urinishlarning oldini olish) uchun muhimdir, ayniqsa "PATIENT’S RECOVERY SYSTEM" kabi tibbiy ma'lumotlar bilan ishlaydigan tizimlarda. Nihoyat, bajarish (execution) bosqichida JVMning ijro etuvchi mexanizmi (execution engine) bayt-kod instruktsiyalarini birma-bir interpretatsiya qiladi yoki Just-In-Time (JIT) kompilyatori yordamida ularni bevosita kompyuterning markaziy protsessori (CPU) tushunadigan mashina kodiga o'girib, ijro etadi. Shu bilan birga, JVM dasturning xotira bilan ishlashini (obyektlarni yaratish, keraksiz obyektlarni avtomatik tarzda o'chirish – garbage collection) boshqaradi (Venners, 1998). Ushbu murakkab jarayonlar muvaffaqiyatli yakunlangandan so'nggina "PATIENT’S RECOVERY SYSTEM" to'liqqonli ishga tushib, o'z funksiyalarini bajara oladi.

D1

Algoritmning tanlangan dasturlash tilida samarali va to'g'ri amalga oshirilishini, shuningdek, uning dastlabki yozma (masalan, psevdokod yoki blok-sxema ko'rinishidagi) varianti bilan yakuniy kod varianti o'rtasidagi uzviy bog'liqlikni baholash, ishlab chiqilayotgan dasturiy ta'minotning sifati, ishonchliligi va kelajakda qo'llab-quvvatlanish qulayligini (maintainability) ta'minlashda hal qiluvchi ahamiyatga ega. "PATIENT’S RECOVERY SYSTEM" loyihasi uchun Java dasturlash tili va Spring Boot freymvorkining tanlanishi bir qator omillar bilan asoslanadi: Java tilining ob'ektga yo'naltirilgan paradigmasi (OYDP) bemorlar, shifokorlar, tibbiy muolajalar kabi real dunyo obyektlarini tabiiy ravishda modellashtirish imkonini beradi (AM-1 agent modellari); Spring Boot esa MVC arxitekturasini qo'llab-quvvatlashi, keng ko'lamli tizimlarni tez va samarali ishlab chiqish uchun tayyor vositalar (masalan, ma'lumotlar bazasi bilan ishlash, xavfsizlikni ta'minlash) taqdim etishi bilan ajralib turadi (Walls, 2016). Keling, bemorning kasalxonadan chiqqandan keyingi holatini masofadan monitoring qilish va kiritilgan ma'lumotlar (masalan, kunlik simptomlar, dori-darmonlarga reaksiya) asosida shifokorga xabarnoma yuborish yoki favqulodda vaziyatda avtomatik ravishda tez yordam xizmatini chaqirish (scenario description) uchun mo'ljallangan algoritmni ko'rib chiqaylik. Ushbu algoritm dastlab psevdokod yoki batafsil tavsif ko'rinishida ifodalanadi, unda mantiqiy shartlar, qaror qabul qilish nuqtalari va bajariladigan amallar aniq ko'rsatiladi. Masalan:

Code

IF bemor\_holati\_kritik (simptomlar, hayotiy\_ko'rsatkichlar) THEN

AVTOMATIK\_TEZ\_YORDAM\_CHAQIRISH()

SHIFOKORGA\_OGOHLANTIRISH\_YUBORISH (bemor\_ID, holat\_tavsifi)

ELSE IF bemor\_holati\_xavotirli (simptomlar) THEN

SHIFOKORGA\_MASLAHAT\_UCHUN\_XABAR\_YUBORISH (bemor\_ID, simptomlar\_tavsifi)

TAVSIYA\_BERISH (bemor\_ID, umumiy\_tavsiyalar)

ELSE

MUNTАZAM\_KUZATUVNI\_DAVOM\_ETTIRISH()

KEYINGI\_INTERVYUNI\_REJALASHTIRISH (bemor\_ID)

END IF

Ushbu yozma algoritmni Java kodiga o'tkazishda har bir mantiqiy blok (IF-ELSE IF-ELSE), har bir shart (bemor\_holati\_kritik, bemor\_holati\_xavotirli) va har bir amal (AVTOMATIK\_TEZ\_YORDAM\_CHAQIRISH, SHIFOKORGA\_OGOHLANTIRISH\_YUBORISH) Java sinflari, metodlari va boshqaruvchi konstruksiyalari yordamida aniq implementatsiya qilinishi lozim. Yozma algoritm va kod varianti o'rtasidagi munosabat bir-biriga mos kelishi, ya'ni kod algoritmdagi har bir detalni to'g'ri aks ettirishi shart. Baholash jarayonida quyidagi mezonlarga e'tibor qaratiladi: **To'g'rilik (Correctness):** Kodning barcha mumkin bo'lgan kirish ma'lumotlari (turli simptomlar, bemorning turli holatlari) uchun algoritm mantig'iga mos ravishda kutilgan natijalarni berishi (TM-1 "Diagnose" va TM-2 "Monitor" vazifalarining sifat o'lchovlari). **Samaradorlik (Efficiency):** Algoritmning bajarilish tezligi va xotiradan foydalanish darajasi, ayniqsa ko'p sonli bemorlarga xizmat ko'rsatishda (OM-5 "Yagona tibbiy mutaxassis juda ko'p sonli bemorlarni qisqa vaqt ichida hal qilishi mumkin"). **O'qiluvchanlik va Qo'llab-quvvatlanuvchanlik (Readability and Maintainability):** Kodning tushunarli yozilganligi, izohlarning mavjudligi, modullarga ajratilganligi va kelajakda o'zgartirishlar kiritish yoki xatoliklarni tuzatish qanchalik osonligi (Martin, 2008). **Ishonchlilik va Xatoliklarga Bardoshlilik (Robustness and Fault Tolerance):** Tizimning kutilmagan vaziyatlar (masalan, noto'g'ri ma'lumot kiritilishi, tarmoq uzilishlari) yoki tizimdagi ichki xatoliklarga nisbatan qanchalik barqaror ishlashi – bu tibbiy tizimlar uchun o'ta muhimdir. **Testlanuvchanlik (Testability):** Kodning alohida qismlarini (birlik testlar) va umumiy tizimni (integratsion testlar) sinovdan o'tkazish qanchalik osonligi. Ushbu algoritmni Java'da implementatsiya qilishda, masalan, bemor holatini baholash mantig'i alohida servis klassiga chiqarilishi, har bir ogohlantirish turi uchun maxsus xabarnoma shablonlari ishlatilishi va favqulodda vaziyatlarni qayd etish uchun loglash mexanizmlari qo'llanilishi mumkin. Yozma algoritm va kod o'rtasidagi farqlar (masalan, kodda qo'shimcha xatoliklarni qayta ishlash bloklari yoki optimallashtirishlar kiritilishi) aniq hujjatlashtirilishi va asoslantirilishi lozim, chunki bu tizimning umumiy sifatiga va ishonchliligiga bevosita ta'sir ko'rsatadi. Shuningdek, "PATIENT’S RECOVERY SYSTEM" kabi murakkab tizimlarda algoritmlarning nafaqat texnik jihatdan to'g'ri implementatsiya qilinishi, balki ularning tibbiy protokollar va axloqiy me'yorlarga mos kelishi ham muhimdir (Beauchamp and Childress, 2013), bu esa algoritmni baholashda qo'shimcha mas'uliyat yuklaydi.

LO2: P2

Dasturlash paradigmalarini tushunish har qanday dasturchi uchun, ayniqsa, sanoat sohalarini endigina o'zgartirayotgan mutaxassislar uchun muhim ahamiyat kasb etadi, chunki ular dasturiy ta'minotni loyihalash va ishlab chiqishga bo'lgan yondashuvlarni fundamental darajada belgilaydi. Asosiy dasturlash paradigmalaridan uchta keng tarqalgani – protsessual, ob'ektga yo'naltirilgan va hodisalarga asoslangan dasturlashdir.

**Protsessual dasturlash (Procedural Programming - PP)** paradigmasi dasturni bajarilishi kerak bo'lgan bir qator protseduralar yoki funksiyalar (ba'zan subrutinalar deb ham ataladi) ketma-ketligi sifatida ko'rib chiqadi. Ushbu yondashuvda asosiy e'tibor amallarni bajarishga, ya'ni "qanday qilish kerak" degan savolga javob topishga qaratiladi. Dastur odatda yuqoridan pastga qarab (top-down design) loyihalanadi, ya'ni katta masala kichikroq, boshqarilishi osonroq bo'lgan protseduralarga bo'linadi (Hanly and Koffman, 2016). Ma'lumotlar ko'pincha global o'zgaruvchilarda saqlanadi yoki protseduralar o'rtasida parametrlar sifatida uzatiladi. Protsessual dasturlash tillariga dastlabki misollar sifatida FORTRAN, COBOL va C kabi tillarni keltirish mumkin. Uning asosiy xususiyatlari qatoriga modullilik (protseduralar orqali), kodning qayta ishlatilishi (protseduralarni bir necha marta chaqirish imkoniyati) va nisbatan sodda boshqaruv oqimi kiradi. "PATIENT’S RECOVERY SYSTEM" kontekstida, agar tizimning ayrim kichik, mustaqil vazifalari (masalan, bemorning yoshini hisoblash yoki ma'lum bir davolash kursi uchun umumiy xarajatni kalkulyatsiya qilish) bo'lsa, ularni alohida protseduralar sifatida amalga oshirish mumkin edi. Biroq, tizim murakkablashgan sari, global ma'lumotlarga tayanish va protseduralar o'rtasidagi bog'liqliklarning ortishi kodni tushunish, o'zgartirish va xatoliklarni tuzatishni qiyinlashtirishi mumkin (Sebesta, 2016).

**Ob'ektga yo'naltirilgan dasturlash (Object-Oriented Programming - OOP)** paradigmasi esa dasturni o'zaro ta'sir qiluvchi "ob'ektlar" to'plami sifatida tasavvur qiladi. Har bir ob'ekt real dunyodagi yoki abstrakt tushunchaning namunasi bo'lib, o'zining ma'lumotlari (atributlar yoki holatlar) va ushbu ma'lumotlar ustida amallar bajaruvchi xatti-harakatlari (metodlar)ga ega bo'ladi (Booch et al., 2007). OOP ning asosiy tamoyillari quyidagilardir: **Inkapsulyatsiya (Encapsulation)** – ob'ektning ma'lumotlarini tashqi aralashuvlardan himoyalash va ularga faqat ob'ektning o'z metodlari orqali murojaat qilish imkoniyatini berish; bu ma'lumotlarning yaxlitligini ta'minlaydi. **Merosxo'rlik (Inheritance)** – mavjud klass (ota-klass) asosida yangi klasslar (bola-klasslar) yaratish imkonini beradi, bunda bola-klass ota-klassning atributlari va metodlarini meros qilib oladi hamda ularni kengaytirishi yoki o'zgartirishi mumkin; bu kodning qayta ishlatilishini sezilarli darajada oshiradi. **Polimorfizm (Polymorphism)** – "ko'p shakllilik" degan ma'noni anglatib, turli klasslarga mansub ob'ektlarning bir xil nomdagi metod chaqiruviga o'ziga xos tarzda javob berish qobiliyatini ifodalaydi; bu dasturning moslashuvchanligini va kengaytirilishini osonlashtiradi. "PATIENT’S RECOVERY SYSTEM" uchun OOP juda mos keladi: Bemor, Shifokor, TibbiyTarix, DoriDarmon, ReabilitatsiyaJarayoni kabi tushunchalar alohida klasslar sifatida loyihalanib, ularning namunalari (ob'ektlari) tizimda o'zaro aloqada bo'ladi (AM-1 agent modellari bunga yaqqol misol). Masalan, Bemor klassi bemorning shaxsiy ma'lumotlari, kasallik tarixi va joriy alomatlarini o'zida saqlashi, simptomlarniYangilash() yoki konsultatsiyaSoʻrash() kabi metodlarga ega bo'lishi mumkin. OOP protsessual dasturlashning murakkablikni boshqarishdagi ba'zi cheklovlarini yengishga yordam beradi, chunki u ma'lumotlar va ular ustida amallar bajaruvchi funksiyalarni bitta birlik – ob'ekt ichida birlashtiradi (Deitel and Deitel, 2017).

**Hodisalarga asoslangan dasturlash (Event-Driven Programming - EDP)** paradigmasi dasturning bajarilish oqimini tashqi yoki ichki "hodisalar" (events) tomonidan boshqarilishini nazarda tutadi. Hodisalar foydalanuvchining harakatlari (masalan, sichqoncha tugmasini bosish, klaviaturada belgi kiritish), sensorlardan keladigan signallar, tizim xabarlari yoki boshqa dasturlardan keladigan chaqiruvlar bo'lishi mumkin (Gamma et al., 1995). Dastur hodisalarni kutish holatida bo'ladi va biror hodisa yuzaga kelganda, unga mos keladigan maxsus funksiya yoki metod – "hodisa ishlovchisi" (event handler yoki event listener) ishga tushiriladi. EDP ayniqsa grafik foydalanuvchi interfeyslari (GUI) va real vaqt rejimida ishlaydigan tizimlar uchun keng qo'llaniladi. "PATIENT’S RECOVERY SYSTEM" veb-ilova sifatida ishlab chiqilishi (Java Spring Boot + Thymeleaf) ko'zda tutilganligi sababli, EDP muhim rol o'ynaydi. Masalan, tibbiy xodim bemorning yangi ma'lumotlarini veb-shakl orqali kiritib, "Saqlash" tugmasini bosganda, "tugmaBosildi" hodisasi yuzaga keladi va tegishli hodisa ishlovchisi ma'lumotlarni ma'lumotlar bazasiga yozish amalini bajaradi. Yoki, agar bemorning hayotiy ko'rsatkichlarini kuzatuvchi sensor (agar tizim bunday integratsiyani nazarda tutsa) xavfli o'zgarishni aniqlasa, "kritikHolat" hodisasini generatsiya qilishi va bu hodisa shifokorni ogohlantiruvchi mexanizmni ishga tushirishi mumkin.

Ushbu uch paradigma o'rtasidagi munosabatlarga kelsak, ular bir-birini butunlay inkor etmaydi, aksincha, ko'pincha birgalikda qo'llaniladi. Masalan, hodisalarga asoslangan dasturlarda hodisa ishlovchilari ko'pincha protsessual funksiyalar yoki ob'ektga yo'naltirilgan metodlar sifatida yoziladi. Ob'ektga yo'naltirilgan tizimlar o'z navbatida protsessual dasturlash elementlarini (masalan, metodlar ichidagi amallar ketma-ketligi) o'z ichiga oladi. Zamonaviy dasturiy ta'minot ishlab chiqishda, ayniqsa "PATIENT’S RECOVERY SYSTEM" kabi murakkab tizimlarda, ko'pincha ushbu paradigmalarning gibrid kombinatsiyasi qo'llaniladi: OOP tizimning umumiy strukturasini va ma'lumotlarni modellashtirishni ta'minlasa, EDP foydalanuvchi bilan interaktiv aloqani va asinxron operatsiyalarni boshqaradi, PP esa alohida, kichik vazifalarni bajaruvchi metodlar yoki funksiyalar ichida o'z ifodasini topadi (Stroustrup, 2013). Dasturchilar uchun ushbu paradigmalarning har birining kuchli va zaif tomonlarini, shuningdek, ularning qachon va qanday qo'llanilishi maqsadga muvofiq ekanligini tushunish, samarali va sifatli dasturiy mahsulotlar yaratishning kalitidir.

M2

Berilgan dasturiy ta'minotning manba kodida protsessual, ob'ektga yo'naltirilgan va hodisalarga asoslangan paradigmalarning qo'llanilishini solishtirish va taqqoslash ularning amaliy jihatlarini chuqurroq anglash imkonini beradi. "PATIENT’S RECOVERY SYSTEM" (PRS) misolida ushbu paradigmalarning kod darajasida qanday namoyon bo'lishini ko'rib chiqamiz.

Agar PRS **protsessual yondashuv** asosida ishlab chiqilgan bo'lsa, uning manba kodi asosan bir qator funksiyalardan tashkil topgan bo'lar edi. Bemor ma'lumotlari (masalan, ismi, yoshi, kasallik tarixi, joriy simptomlari), shifokorlar ro'yxati, davolash protokollari kabi ma'lumotlar global o'zgaruvchilarda, massivlarda yoki strukturalarda saqlanishi mumkin edi. Tizimning asosiy funksiyalari bemorMalumotlariniKiritish(bemor\_id, yangi\_simptomlar), shifokorgaXabarYuborish(shifokor\_id, xabar\_matni), reabilitatsiyaJarayoniniYangilash(bemor\_id, bajarilgan\_mashqlar) kabi alohida protseduralar orqali amalga oshirilar edi. Ushbu yondashuvda ma'lumotlar va ularni qayta ishlovchi funksiyalar bir-biridan ajratilgan bo'ladi. Masalan, bemorning kasallik tarixini olish uchun getKasallikTarixi(bemor\_id) funksiyasi chaqirilsa, unga yangi yozuv qo'shish uchun addKasallikTarixigaYozuv(bemor\_id, yangi\_yozuv) funksiyasi ishlatilishi mumkin. Bunday tuzilmada, tizim kengaygan sari, qaysi funksiya qaysi ma'lumotlarga ta'sir qilishini kuzatish qiyinlashadi va ma'lumotlar yaxlitligini ta'minlash murakkablashadi (OM-4 da "Medical History (Computer system)" va "Medical History (paper form)" ning farqlari va kompyuter tizimining afzalliklari bunga ishora qiladi).

Aksincha, PRS **ob'ektga yo'naltirilgan paradigma (OOP)** asosida qurilganida, manba kodi Patient, Doctor, Appointment, MedicalRecord, Symptom, TreatmentPlan kabi klasslar to'plamidan iborat bo'ladi. Har bir klass o'ziga tegishli ma'lumotlarni (atributlarni) va ushbu ma'lumotlar ustida amallar bajaruvchi metodlarni o'z ichiga oladi (inkapsulyatsiya). Masalan, Patient klassi patientId, name, dateOfBirth, List<MedicalRecord> medicalHistory, List<Symptom> currentSymptoms kabi atributlarga va addSymptom(Symptom newSymptom), getMedicalHistory(), requestAppointment(Doctor preferredDoctor, Date preferredDate) kabi metodlarga ega bo'lishi mumkin. Shifokorlar uchun Doctor klassi, tibbiy yozuvlar uchun MedicalRecord klassi va hokazo. Merosxo'rlik tamoyili orqali, masalan, Surgeon (Jarroh) klassi umumiy Doctor klassidan vorislik qilib, unga qo'shimcha maxsus xususiyatlar (masalan, specialization, surgicalProceduresPerformed) va metodlar qo'shishi mumkin. Polimorfizm esa, aytaylik, turli xil Notification (ogohlantirish) turlari (masalan, EmailNotification, SMSNotification, InAppNotification) umumiy send() metodiga ega bo'lib, har biri xabarni o'ziga xos tarzda yuborishini ta'minlashi mumkin. Bunday yondashuvda kod ancha modulli, qayta ishlatilishi oson va kengaytirish uchun qulay bo'ladi, chunki har bir ob'ekt o'z mas'uliyat doirasiga ega (OM-2 "Structure" va "People" bo'limlari bunday rollarni ajratishga moyillikni ko'rsatadi).

PRS tizimida **hodisalarga asoslangan dasturlash (EDP)** ayniqsa foydalanuvchi interfeysi (masalan, Thymeleaf bilan qurilgan veb-sahifalar) va real vaqtda javob berishni talab qiluvchi funksiyalar (masalan, bemorning ahvoli yomonlashganda tezkor ogohlantirishlar) uchun markaziy o'rin tutadi. Manba kodida bu hodisalarni tinglovchi (event listeners) va ularga javob beruvchi (event handlers) komponentlar mavjud bo'ladi. Masalan, bemor o'zining uyda kuzatilayotgan simptomlarini veb-interfeys orqali yuborganda ("submit\_symptoms\_button\_clicked" hodisasi), server tomonida bu hodisani qabul qilib oluvchi maxsus kontrollyor metodi (Spring MVC da @PostMapping bilan belgilanadigan metod) ishga tushadi. Bu metod o'z navbatida tegishli servis obyektining metodini chaqirib, ma'lumotlarni qayta ishlaydi va kerak bo'lsa, boshqa hodisalarni (masalan, "new\_critical\_symptom\_reported\_event") generatsiya qilishi mumkin. Agar tizim mikroservislar arxitekturasida (D mezonlari) qurilgan bo'lsa va RabbitMQ kabi xabar almashish navbatlaridan foydalansa, bir servis tomonidan generatsiya qilingan hodisa (xabar) navbatga qo'yilib, boshqa bir yoki bir nechta qiziqqan servislar tomonidan asinxron ravishda qabul qilib olinishi va qayta ishlanishi mumkin. Bu tizim komponentlarining bir-biriga bog'liqligini kamaytiradi (loose coupling) va tizimning umumiy javobgarligini (responsiveness) hamda masshtablanuvchanligini (scalability) oshiradi.

Solishtirganda, **protsessual yondashuv** PRS kabi murakkab tizim uchun yetarlicha struktura va boshqaruv imkoniyatini bera olmaydi, chunki unda ma'lumotlar va funksiyalar o'rtasidagi bog'liqliklar chalkashib ketishi ehtimoli yuqori. **OOP** esa PRS ning asosiy domen mantig'ini, ma'lumotlar modellarini va biznes qoidalarini aniq, tushunarli va kengaytiriladigan tarzda ifodalash uchun kuchli vosita hisoblanadi. U kodning qayta ishlatilishini, modulliligini va testlanuvchanligini yaxshilaydi. **EDP** esa PRS ning interaktivligini, real vaqtda javob berish qobiliyatini va asinxron operatsiyalarni samarali boshqarishni ta'minlaydi. Amalda, PRS kabi zamonaviy tizimlar ko'pincha ushbu paradigmalarning kombinatsiyasidan foydalanadi: OOP tizimning asosiy arxitekturasini tashkil etadi (masalan, Spring Boot da servislar, repozitoriylar, domen obyektlari klasslar sifatida), EDP esa foydalanuvchi interfeysi bilan aloqani va tizim ichidagi komponentlar o'rtasidagi asinxron muloqotni (masalan, RabbitMQ orqali) boshqaradi. Hatto OOP metodlari ichida ham kichik, aniq vazifalarni bajaruvchi protsessual kod fragmentlari mavjud bo'ladi. Shunday qilib, har bir paradigmaning o'ziga xos kuchli tomonlari PRS ning turli qismlarida maqsadli ravishda qo'llanilishi mumkin.

D2

Dasturiy ta'minotning manba kodini protsessual, ob'ektga yo'naltirilgan va hodisalarga asoslangan paradigmalarni amalga oshirishi nuqtai nazaridan tanqidiy baholash, ularning kod tuzilishi va funksional xususiyatlariga qanday ta'sir qilishini chuqur tahlil qilishni talab etadi. "PATIENT’S RECOVERY SYSTEM" (PRS) uchun ushbu paradigmalarning qo'llanilishini bir nechta muhim jihatlar bo'yicha baholaymiz: kodning modulliligi, kengaytiriluvchanligi, qayta ishlatiluvchanligi, ma'lumotlar yaxlitligi va xatoliklarga bardoshliligi.

Agar PRS ning biror bir muhim funksiyasi, masalan, **"Bemorning hayotiy ko'rsatkichlari (masalan, qon bosimi) belgilangan me'yordan keskin chetga chiqqanda shifokorni zudlik bilan ogohlantirish"** funksiyasini ko'rib chiqaylik.

**Protsessual implementatsiya**da bu funksiya bir yoki bir nechta global funksiyalar orqali amalga oshirilishi mumkin edi. Masalan, monitorVitalSigns(patientId) funksiyasi davriy ravishda chaqirilib, bemorning ma'lumotlarini (ehtimol global massiv yoki fayldan o'qib) tekshiradi. Agar qon bosimi kritik chegaradan oshsa, triggerAlert(doctorId, patientId, alertMessage) funksiyasi chaqiriladi.

* **Tanqidiy baho:** Bunday yondashuvda kod tuzilishi nisbatan sodda bo'lishi mumkin, ammo bir qator jiddiy kamchiliklarga ega. **Modullilik** past darajada bo'ladi, chunki hayotiy ko'rsatkichlarni tekshirish logikasi va ogohlantirish yuborish logikasi bir-biriga qattiq bog'lanib qolishi mumkin. Agar ogohlantirish usuli (masalan, SMS o'rniga email) o'zgarsa yoki yangi turdagi hayotiy ko'rsatkich (masalan, qand miqdori) qo'shilsa, mavjud funksiyalarni sezilarli darajada o'zgartirish kerak bo'ladi, bu esa **kengaytiriluvchanlik**ni qiyinlashtiradi. Ma'lumotlar (bemor holati, shifokor kontaktlari) global miqyosda yoki ko'plab funksiyalar orqali uzatilganligi sababli, **ma'lumotlar yaxlitligi**ni ta'minlash murakkablashadi va xatoliklarga moyillik ortadi. **Qayta ishlatiluvchanlik** ham cheklangan bo'ladi, chunki funksiyalar o'ziga xos ma'lumotlar tuzilmalariga bog'liq bo'lishi mumkin.

**Ob'ektga yo'naltirilgan implementatsiya**da esa bu funksiya bir nechta o'zaro ta'sir qiluvchi ob'ektlar yordamida amalga oshiriladi. Patient obyekti o'zining VitalSign obyektlari to'plamiga (masalan, BloodPressure, HeartRate) ega bo'lishi mumkin. Har bir VitalSign obyekti o'zining joriy qiymatini va kritik chegaralarini biladi. Patient obyekti checkVitalSigns() metodiga ega bo'lishi mumkin, bu metod barcha hayotiy ko'rsatkichlarni tekshiradi. Agar biror ko'rsatkich kritik bo'lsa, Patient obyekti AlertService (yoki NotificationManager) ob'ektining sendCriticalAlert(doctor, patient, details) metodini chaqiradi. AlertService esa kerakli Doctor ob'ektini topib, unga ogohlantirish yuboradi.

* **Tanqidiy baho:** Bu yondashuvda kod tuzilishi ancha yaxshilanadi. **Modullilik** yuqori: Patient, VitalSign, AlertService, Doctor kabi har bir klass o'zining aniq mas'uliyatiga ega. Bu esa kodni tushunishni, testlashni va alohida qismlarini mustaqil ravishda o'zgartirishni osonlashtiradi. **Kengaytiriluvchanlik** ham yaxshi: yangi turdagi hayotiy ko'rsatkichni qo'shish uchun yangi VitalSign voris klassini yaratish yoki yangi ogohlantirish usulini qo'shish uchun AlertServicega yangi strategiya (masalan, Strategy pattern yordamida) qo'shish mumkin. **Inkapsulyatsiya** tufayli bemorning ma'lumotlari Patient obyekti ichida himoyalangan bo'ladi, bu esa **ma'lumotlar yaxlitligi**ni oshiradi. Metodlar va klasslar turli kontekstlarda **qayta ishlatilishi** mumkin. Masalan, AlertService nafaqat hayotiy ko'rsatkichlar, balki boshqa turdagi kritik hodisalar uchun ham ogohlantirish yuborishda ishlatilishi mumkin. OM-5 da ta'kidlanganidek, tizim "bemorning muammosi bo'yicha qaror qabul qilish uchun shifokorni qo'llab-quvvatlashga harakat qiladi" va OOP bunday qo'llab-quvvatlashni amalga oshiruvchi komponentlarni aniq ajratishga yordam beradi.

**Hodisalarga asoslangan implementatsiya** (ko'pincha OOP bilan birgalikda) quyidagicha bo'lishi mumkin: VitalSignMonitor (bu ham bir ob'ekt bo'lishi mumkin) bemorning hayotiy ko'rsatkichlaridagi o'zgarishlarni doimiy kuzatib boradi. Agar kritik o'zgarish yuz bersa, u CriticalVitalSignEvent hodisasini generatsiya qiladi. Tizimda ushbu hodisani tinglaydigan bir yoki bir nechta EventListener (masalan, AlertNotificationListener, EmergencyProtocolTriggerListener) mavjud bo'ladi. AlertNotificationListener hodisani qabul qilib, tegishli shifokorga ogohlantirish yuborish logikasini ishga tushiradi (bu ham AlertService ob'ektidan foydalanishi mumkin).

* **Tanqidiy baho:** Bu yondashuv komponentlar o'rtasidagi bog'liqlikni minimal darajaga tushiradi (**loose coupling**). VitalSignMonitor ogohlantirish qanday yuborilishini yoki qanday favqulodda protokollar ishga tushirilishini bilishi shart emas; u faqatgina hodisani e'lon qiladi. Bu tizimning **moslashuvchanligi** va **kengaytiriluvchanligi**ni yanada oshiradi. Yangi reaksiya turini qo'shish uchun shunchaki yangi EventListener yaratish kifoya. Bu, ayniqsa, PRS ning kelajakda yangi funksiyalar bilan boyitilishi yoki boshqa tizimlar bilan integratsiya qilinishi (masalan, shifoxonaning umumiy axborot tizimi bilan) ehtimolini hisobga olganda muhimdir. RabbitMQ kabi xabar almashish tizimlaridan foydalanish (D mezonlari) bunday hodisaga asoslangan arxitekturani mikroservislar darajasida samarali amalga oshirish imkonini beradi, bu esa tizimning **xatoliklarga bardoshliligi**ni (bir komponentning ishdan chiqishi boshqalariga minimal ta'sir qiladi) va **masshtablanuvchanligi**ni (alohida komponentlarni mustaqil ravishda masshtablash) yaxshilaydi. Biroq, haddan tashqari ko'p hodisalar va ishlovchilar tizimning umumiy oqimini kuzatishni qiyinlashtirishi mumkin.

Xulosa qilib aytganda, "PATIENT’S RECOVERY SYSTEM" kabi hayotiy muhim va murakkab tizim uchun **protsessual yondashuv**ning o'zi yetarli bo'lmaydi va jiddiy cheklovlarga olib keladi. **Ob'ektga yo'naltirilgan paradigma** tizimning asosiy tuzilishini, ma'lumotlar modellarini va biznes mantig'ini samarali tashkil etish uchun mustahkam poydevor yaratadi. **Hodisalarga asoslangan paradigma** esa, ayniqsa OOP bilan birgalikda qo'llanilganda, tizimning interaktivligini, real vaqtda javob berish qobiliyatini, moslashuvchanligini va kengaytiriluvchanligini sezilarli darajada oshiradi. Tanqidiy nuqtai nazardan, eng maqbul yechim bu paradigmalarning kuchli tomonlarini birlashtirgan gibrid yondashuv bo'lib, unda OOP tizimning yadrosini tashkil etadi, EDP esa dinamik o'zaro ta'sirlarni va asinxron jarayonlarni boshqaradi. Java Spring Boot va RabbitMQ kabi texnologiyalardan foydalanish aynan shunday zamonaviy, modulli va ishonchli tizimni yaratishga imkon beradi.

LO3: P3

Integratsiyalashgan ishlab chiqish muhiti (Integrated Development Environment - IDE) yordamida algoritmni amalga oshiradigan dastur yozish zamonaviy dasturiy ta'minot ishlab chiqish jarayonining ajralmas qismidir. "PATIENT’S RECOVERY SYSTEM" (PRS) loyihasi uchun, masalan, bemorning kunlik o'z-o'zini hisobot qilish ma'lumotlarini (masalan, og'riq darajasi, uyqu sifati, yangi paydo bo'lgan noqulayliklar) qayta ishlaydigan va ushbu ma'lumotlar asosida shifokorga ogohlantirish yuborish yoki bemorga qo'shimcha ko'rsatmalar berish zarurligini aniqlaydigan algoritmni amalga oshirish talab etiladi. Bunday dasturni Java dasturlash tilida, Spring Boot freymvorkidan foydalangan holda, IntelliJ IDEA yoki Eclipse kabi mashhur IDE yordamida ishlab chiqish mumkin.

Dastur yozish jarayoni IDE ichida yangi loyiha yaratishdan boshlanadi. IntelliJ IDEA da bu "New Project" opsiyasi orqali amalga oshiriladi va Spring Initializr (Spring Boot loyihalarini tezkor sozlash vositasi) integratsiyasidan foydalanib, loyihaning asosiy bog'liqliklari (dependencies), masalan, Spring Web (veb-ilovalar uchun), Spring Data JPA (ma'lumotlar bazasi bilan ishlash uchun), Thymeleaf (foydalanuvchi interfeysi shablonlari uchun) va tanlangan ma'lumotlar bazasi drayveri (MySQL yoki PostgreSQL) darhol loyihaga qo'shiladi (OM-5 "veb-ilovalarni qo'llab-quvvatlaydigan ma'lumotlar bazasining yangi uslubi talab qilinadi"). Loyiha tuzilmasi (masalan, src/main/java kod uchun, src/main/resources konfiguratsiya fayllari va shablonlar uchun) avtomatik tarzda yaratiladi.

Keyinchalik, yuqorida aytib o'tilgan algoritmni amalga oshirish uchun zarur bo'lgan Java klasslari yaratiladi. Masalan, bemorning kunlik hisobotini ifodalovchi DailyReport nomli domen klassi (Plain Old Java Object - POJO), ushbu hisobotlarni qayta ishlovchi ReportProcessingService nomli servis klassi va foydalanuvchi so'rovlarini qabul qiluvchi ReportController nomli kontrollyor klassi tuzilishi mumkin. IDE ning kod muharriri dasturchiga bir qator qulayliklar yaratadi: sintaksisni yoritish (syntax highlighting) kodni o'qishni osonlashtiradi; avtomatik kod to'ldirish (code completion) metod nomlari, o'zgaruvchilar va klasslarni tezda kiritishga yordam beradi, bu esa imlo xatolarini kamaytiradi; real vaqtda xatoliklarni aniqlash (on-the-fly error detection) esa kompilyatsiyadan oldinroq potensial muammolarni ko'rsatib beradi (McConnell, 2004).

ReportProcessingService klassi ichida algoritmning asosiy mantig'i yoziladi. Bu yerda bemorning kiritgan ma'lumotlari (masalan, og'riq darajasi 10 ballik shkalada 7 dan yuqori bo'lsa yoki yangi, xavotirli simptom qayd etilsa) tahlil qilinadi. Agar algoritm biror shartni (masalan, kritik holatni) aniqlasa, u NotificationService (alohida servis) orqali shifokorga xabar yuborish amalini chaqirishi mumkin. IDE ning refaktoring vositalari (masalan, metod nomini o'zgartirish, kod qismlarini alohida metodga chiqarish) kod strukturasini yaxshilash va uni yanada tushunarli qilish imkonini beradi (Fowler, 2018). Masalan, bemorning ma'lumotlarini ma'lumotlar bazasidan olish yoki saqlash uchun Spring Data JPA repozitoriylaridan foydalaniladi va IDE bu interfeyslarni yaratish hamda ularning metodlarini avtomatik generatsiya qilishda yordam berishi mumkin. Algoritmni testlash uchun JUnit kabi test freymvorklari bilan integratsiya mavjud bo'lib, IDE ichida birlik testlarni (unit tests) yozish va ishga tushirish mumkin, bu esa algoritmni amalga oshiruvchi kodning to'g'ri ishlashini tekshirish uchun muhimdir. Shunday qilib, IDE nafaqat kod yozish vositasi, balki algoritmni sifatli va samarali amalga oshirish uchun keng qamrovli yordamchi sifatida xizmat qiladi.

M3

Integratsiyalashgan ishlab chiqish muhiti (IDE) dasturiy ta'minotni ishlab chiqish jarayonini boshqarishda shunchaki kod yozish vositasidan ancha kengroq imkoniyatlarni taqdim etadi. "PATIENT’S RECOVERY SYSTEM" (PRS) kabi murakkab loyihada IDE ning boshqaruv funksiyalari loyihaning muvaffaqiyatli yakunlanishi uchun muhim ahamiyatga ega.

Birinchidan, IDE lar odatda versiyalarni boshqarish tizimlari (Version Control Systems - VCS) bilan chuqur integratsiyaga ega bo'ladi, masalan, Git. IntelliJ IDEA yoki Eclipse kabi IDE lar Git omborxonalari (repositories) bilan bevosita ishlash imkonini beradi: o'zgarishlarni indekslash (staging), kommit qilish (committing), yangi shoxobchalar (branches) yaratish, shoxobchalarni birlashtirish (merging) va masofaviy omborxonalar (masalan, GitHub, GitLab) bilan sinxronizatsiya qilish kabi amallarni IDE interfeysidan chiqmasdan bajarish mumkin (Chacon and Straub, 2014). PRS loyihasida bir nechta dasturchi bir vaqtda ishlayotgan bo'lsa, bu funksiyalar kod o'zgarishlarini samarali boshqarish, konfliktlarni hal qilish va loyiha tarixini kuzatib borish uchun juda zarurdir. Har bir dasturchi o'zining alohida funksiyasi yoki moduli ustida ishlashi va keyinchalik o'zgarishlarini asosiy shoxobchaga integratsiya qilishi mumkin.

Ikkinchidan, IDE lar loyihani tuzish (build) va bog'liqliklarni (dependencies) boshqarish vositalari bilan integratsiyalashgan bo'ladi. PRS loyihasi Spring Boot asosida qurilganligi sababli, u odatda Maven yoki Gradle kabi tuzish vositalaridan foydalanadi. IDE ushbu vositalarning konfiguratsiya fayllarini (masalan, pom.xml Maven uchun yoki build.gradle Gradle uchun) tushunadi va tahrirlash imkonini beradi. Loyihaga yangi kutubxona (masalan, ma'lumotlarni vizualizatsiya qilish uchun grafik kutubxonasi) qo'shish kerak bo'lsa, buni konfiguratsiya fayliga bir necha qator qo'shish orqali amalga oshirish mumkin va IDE avtomatik ravishda kerakli fayllarni yuklab oladi va loyihaga bog'laydi. Shuningdek, IDE loyihani kompilyatsiya qilish, testlarni ishga tushirish va bajariladigan paketni (masalan, JAR yoki WAR faylini) yaratish kabi tuzish jarayonlarini bir tugmani bosish orqali boshqarish imkonini beradi (OM-5 "Tizimni muntazam yangilash va texnik xizmat ko'rsatish juda muhimdir" bandi bunday jarayonlarni osonlashtirish zarurligini ko'rsatadi).

Uchinchidan, IDE lar dasturni diskretlash (debugging) uchun kuchli vositalarni taqdim etadi. Dasturchi kodning istalgan joyiga to'xtash nuqtalarini (breakpoints) qo'yishi, dasturning bajarilishini qadamma-qadam kuzatishi, o'zgaruvchilarning joriy qiymatlarini ko'rishi va hatto ularni o'zgartirishi mumkin. PRS da bemorning ma'lumotlarini qayta ishlash algoritmidagi mantiqiy xatolikni topish uchun diskretlash vositalari juda foydali bo'ladi (bu haqda LO4 da batafsil to'xtalib o'tiladi).

To'rtinchidan, IDE lar kodni tahlil qilish va refaktoring qilish uchun ilg'or funksiyalarga ega. Statik kod analizatorlari potensial xatoliklarni, kod uslubidagi nomuvofiqliklarni yoki samarasiz kod qismlarini aniqlashga yordam beradi. Refaktoring vositalari esa (masalan, o'zgaruvchi yoki metod nomini xavfsiz o'zgartirish, klassni alohida faylga chiqarish, merosxo'rlik ierarxiyasini o'zgartirish) kod sifatini uning funksionalligini o'zgartirmagan holda yaxshilash imkonini beradi (Fowler, 2018). PRS ning uzoq muddatli qo'llab-quvvatlanishi va kengaytirilishi uchun bu juda muhim.

Beshinchidan, IDE lar testlash jarayonini boshqarishni osonlashtiradi. JUnit yoki TestNG kabi test freymvorklari bilan integratsiya birlik testlarni, integratsion testlarni yozish, ishga tushirish va natijalarini ko'rish uchun qulay interfeysni ta'minlaydi. Test qamrovi (test coverage) vositalari esa kodning qancha qismi testlar bilan qoplanganligini ko'rsatib beradi, bu esa tizim ishonchliligini oshirishga yordam beradi (Myers et al., 2011). PRS kabi tibbiy tizim uchun testlashning ahamiyati juda yuqori (OM-5 "Tizimni sinab ko'rish va tasdiqlash uchun tibbiy mutaxassislar eski bemorning ma'lumotlarini berish orqali yaxshi ish qilishlari mumkin").

Xulosa qilib aytganda, IDE dasturiy ta'minotni ishlab chiqish jarayonining deyarli barcha bosqichlarini – loyihani sozlashdan tortib, kod yozish, diskretlash, testlash, versiyalarni boshqarish va loyihani tuzishgacha – bitta integratsiyalashgan muhitda boshqarish imkonini beradi. Bu esa dasturchilarning samaradorligini oshiradi, xatoliklarni kamaytiradi va "PATIENT’S RECOVERY SYSTEM" kabi murakkab loyihalarni muvaffaqiyatli amalga oshirishga yordam beradi.

D3

Ilovalarni ishlab chiqish uchun integratsiyalashgan ishlab chiqish muhitidan (IDE) foydalanishni IDE dan foydalanmaslik (masalan, oddiy matn muharriri va buyruqlar satri vositalaridan foydalanish) bilan taqqoslaganda, IDE ning bir qator muhim afzalliklari yaqqol namoyon bo'ladi, ayniqsa "PATIENT’S RECOVERY SYSTEM" (PRS) kabi keng ko'lamli va murakkab loyihalar uchun.

**Samaradorlik va Ishlab chiqarish Unumdorligi:** IDE dan foydalanish dasturchining ish unumdorligini sezilarli darajada oshiradi. Kodni avtomatik to'ldirish, sintaksisni yoritish, real vaqtda xatoliklarni aniqlash, loyiha bo'ylab tezkor navigatsiya, shablonlardan foydalanish (code snippets) kabi funksiyalar kod yozish jarayonini tezlashtiradi va mexanik xatolarni kamaytiradi. IDE dan foydalanmaganda, dasturchi bularning barchasini qo'lda bajarishi yoki alohida-alohida vositalarni sozlab ishlatishi kerak bo'ladi, bu esa ko'proq vaqt va e'tibor talab qiladi. PRS loyihasida ko'plab klasslar, metodlar va konfiguratsiya fayllari mavjud bo'lishini hisobga olsak, IDE ning bu yordami beqiyosdir.

**Xatoliklarni Aniqlash va Diskretlash:** IDE lar kompilyatsiya vaqtida va hatto kod yozish jarayonida potensial xatoliklarni aniqlash uchun kuchli vositalarga ega. O'rnatilgan diskretlash vositalari (debugger) dasturning bajarilishini chuqur tahlil qilish, xatoliklar manbasini topish va ularni tuzatish jarayonini ancha osonlashtiradi. IDE siz, diskretlash asosan print bayonotlarini kodga qo'shish yoki buyruqlar satridan ishga tushiriladigan alohida diskretlash vositalaridan (masalan, GDB C/C++ uchun, JDB Java uchun) foydalanish orqali amalga oshiriladi, bu esa kamroq qulay va ko'proq vaqt talab qiladi (Rosenberg, 2000). PRS da bemorning hayotiga bevosita ta'sir qilishi mumkin bo'lgan xatoliklar bo'lmasligi shart (OM-5 "Ba'zi tanqidiy tomonlari bor, chunki u bevosita inson hayoti bilan bog'liq"), shuning uchun IDE ning ilg'or diskretlash imkoniyatlari juda muhim.

**Loyiha Boshqaruvi va Tuzilishi:** IDE lar loyihaning umumiy tuzilishini boshqarish, fayllar va papkalarni tartibga solish, bog'liqliklarni (dependencies) boshqarish (Maven/Gradle integratsiyasi orqali) va loyihani tuzish (build) jarayonlarini avtomatlashtirish uchun markazlashgan platformani taqdim etadi. IDE siz, bu vazifalarning har biri alohida buyruqlar satri vositalari yordamida bajarilishi kerak, bu esa loyiha konfiguratsiyasini murakkablashtiradi va xatolarga moyillikni oshiradi. PRS kabi ko'p komponentli tizimda loyihani samarali boshqarish uchun IDE ning bu imkoniyatlari juda qo'l keladi.

**Versiyalarni Boshqarish Integratsiyasi:** Aksariyat zamonaviy IDE lar Git kabi versiyalarni boshqarish tizimlari bilan uzviy integratsiyaga ega. Bu dasturchilarga kod o'zgarishlarini IDE ichida osongina kuzatish, kommit qilish, shoxobchalar bilan ishlash va jamoaviy ishlashni samarali tashkil etish imkonini beradi. IDE siz, barcha VCS operatsiyalari buyruqlar satri orqali yoki alohida grafik mijozlar yordamida bajarilishi kerak.

**Refaktoring va Kod Sifati:** IDE lar kodni xavfsiz refaktoring qilish uchun kuchli vositalarni (masalan, o'zgaruvchi nomini loyiha bo'ylab o'zgartirish, metodni boshqa klassga ko'chirish) taklif qiladi. Shuningdek, ular kod uslubini tekshirish (linting) va statik analiz vositalari bilan integratsiyalashib, kod sifatini oshirishga yordam beradi. IDE siz, refaktoring ancha xavfli va ko'p mehnat talab qiladigan jarayonga aylanishi mumkin.

**O'rganish Egri Chizig'i va Resurslardan Foydalanish:** Albatta, IDE larning ham o'ziga yarasha kamchiliklari bo'lishi mumkin. Ular odatda oddiy matn muharrirlariga qaraganda ko'proq tizim resurslarini (xotira, protsessor vaqti) talab qiladi. Shuningdek, IDE ning barcha funksiyalarini o'rganish uchun ma'lum vaqt kerak bo'lishi mumkin (o'rganish egri chizig'i). Ba'zi dasturchilar IDE larni haddan tashqari "og'ir" yoki keraksiz funksiyalarga to'la deb hisoblashlari va yengilroq vositalarni afzal ko'rishlari mumkin. Biroq, PRS kabi yirik loyiha uchun IDE taqdim etadigan afzalliklar bu kamchiliklarni deyarli har doim oqlaydi. IDE siz ishlash dasturchidan yuqori darajadagi intizom, ko'plab vositalarni mukammal bilish va konfiguratsiya qilish qobiliyatini talab etadi, bu esa loyihaning umumiy vaqtini va xarajatlarini oshirishi mumkin. Ayniqsa, jamoaviy ishda yagona IDE va loyiha sozlamalaridan foydalanish standartlashtirishni va ish jarayonining silliqligini ta'minlaydi.

Xulosa qilib aytganda, "PATIENT’S RECOVERY SYSTEM" kabi dasturiy ta'minotni ishlab chiqishda IDE dan foydalanish samaradorlikni oshirish, xatoliklarni kamaytirish, kod sifatini yaxshilash va loyihani boshqarishni osonlashtirish orqali muhim afzalliklarni beradi. IDE siz ishlash nazariy jihatdan mumkin bo'lsa-da, bu amalda ancha qiyin, ko'p vaqt talab qiladigan va xatolarga moyil jarayon bo'lar edi, bu esa loyihaning muvaffaqiyatiga salbiy ta'sir ko'rsatishi mumkin. Shuning uchun, bunday loyihalar uchun zamonaviy IDE lardan foydalanish deyarli standart amaliyot hisoblanadi (Bloch, 2018).

LO4: P4

Integratsiyalashgan ishlab chiqish muhitida (IDE) amalga oshiriladigan diskretlash jarayoni hamda unda mavjud bo'lgan maxsus vositalar dasturiy ta'minotda yuzaga keladigan xatoliklarni, ya'ni "bugs" deb ataluvchi nuqsonlarni aniqlash, ularning tub sabablarini ochish va samarali bartaraf etishda beqiyos ahamiyatga ega. "PATIENT’S RECOVERY SYSTEM" (PRS) singari murakkab tuzilishga ega va inson salomatligi bilan bevosita bog'liq bo'lgan tizimlar uchun (OM-5 da ta'kidlanganidek, "Ba'zi tanqidiy tomonlari bor, chunki u bevosita inson hayoti bilan bog'liq"), dasturiy xatoliklarning o'z vaqtida topilishi va tuzatilishi tizimning umumiy ishonchliligi hamda foydalanuvchilar uchun xavfsizligini ta'minlashda hal qiluvchi omil bo'lib xizmat qiladi. Diskretlash jarayoni o'zi bir necha mantiqiy bosqichlarni qamrab oladi. Dastlab, xatolikni barqaror ravishda qayta tiklashga erishish lozim, chunki tasodifiy yuzaga keladigan nuqsonlarni tahlil qilish ancha mushkul kechadi; dasturchi xatolikka olib keluvchi aniq shart-sharoitlar yoki amallar ketma-ketligini sinchkovlik bilan aniqlashi talab etiladi. Masalan, PRS tizimida bemor tomonidan ma'lum bir noyob simptomlar kombinatsiyasi kiritilganda tizimning kutilmagan xatti-harakat namoyon etishi kuzatilsa, aynan shu holatni takrorlash orqali muammoni o'rganish mumkin bo'ladi. Keyingi muhim qadam – xatolikni izolyatsiya qilish, ya'ni kodning aynan qaysi qismida, qaysi modul yoki metodda ushbu nuqson kelib chiqayotganini aniqlashdir. Bu bosqichda IDE ning diskretlash vositalari, xususan, to'xtash nuqtalari (breakpoints) va qadamma-qadam bajarish imkoniyatlari faol qo'llaniladi. Xatolikning joylashuvi aniqlangach, uning asl sababini chuqur anglab yetish zarur; bu mantiqiy nomuvofiqlikmi, ma'lumotlarni noto'g'ri qayta ishlash oqibatimi yoki resurslardan foydalanishdagi kamchilikmi – bularning barchasi tahlil qilinadi. Shundan so'nggina kodga tegishli o'zgartirishlar kiritilib, xatolik bartaraf etiladi. Yakuniy bosqichda esa kiritilgan tuzatishning to'g'riligini tekshirish, ya'ni dastlabki xatolik endi yuzaga kelmasligiga ishonch hosil qilish va, eng muhimi, kiritilgan o'zgartirishlar tizimning boshqa qismlarida yangi muammolarni (regressiyalarni) keltirib chiqarmaganligini sinovdan o'tkazish talab etiladi, bunda avtomatlashtirilgan testlar muhim rol o'ynaydi (Kaner et al., 2013).

Zamonaviy IDE lar, masalan, IntelliJ IDEA yoki Eclipse, Java dasturlash tili uchun diskretlash jarayonini sezilarli darajada yengillashtiradigan va samaradorligini oshiradigan keng qamrovli vositalar majmuasini taqdim etadi. To'xtash nuqtalari dasturchiga kodning ixtiyoriy bajariladigan qatorida dastur ijrosini vaqtincha to'xtatish imkonini beradi, bu esa o'sha paytdagi tizim holatini, o'zgaruvchilarning qiymatlarini va boshqa muhim parametrlarni sinchkovlik bilan o'rganish uchun qulay sharoit yaratadi. PRS da, misol uchun, bemorning kiritgan ma'lumotlari asosida tavsiya shakllantiruvchi algoritmning qaror qabul qilish blokidan oldin to'xtash nuqtasi o'rnatilishi mumkin. Bundan tashqari, shartli to'xtash nuqtalari (conditional breakpoints) mavjud bo'lib, ular faqat ma'lum bir mantiqiy shart (masalan, bemorning yoshi ma'lum bir qiymatdan katta bo'lganda yoki biror o'zgaruvchi kutilmagan qiymatga ega bo'lganda) bajarilgandagina dastur ijrosini to'xtatadi, bu esa faqat muayyan holatlarda yuzaga keladigan murakkab xatoliklarni aniqlashda juda qo'l keladi. Dastur ijrosi to'xtash nuqtasida to'xtagach, IDE kodni qadamma-qadam bajarish imkoniyatini beradi: "step over" buyrug'i joriy qatorni bajarib, keyingisiga o'tishni ta'minlasa (agar joriy qatorda metod chaqiruvi bo'lsa, u metod ichiga kirmasdan bajariladi), "step into" buyrug'i esa, aksincha, metod chaqiruvi mavjud bo'lsa, o'sha metodning ichiga kirib, uning bajarilishini batafsil kuzatishga imkon beradi. "Step out" esa joriy metoddan chiqib, uni chaqirgan joyga qaytish uchun xizmat qiladi. Bunday qadamma-qadam tahlil PRS algoritmlarining ichki mantig'ini chuqur o'rganishda muhimdir. Shuningdek, diskretlash paytida IDE maxsus oynada barcha joriy o'zgaruvchilarning (lokal o'zgaruvchilar, obyektlarning maydonlari) nomlari va ularning o'sha paytdagi qiymatlarini aks ettiradi. Dasturchi nafaqat bu qiymatlarni ko'rishi, balki ularni diskretlash jarayonida o'zgartirishi ham mumkin, bu esa turli farazlarni tezda tekshirib ko'rish va xatolik sababini aniqlashga yordam beradi. PRS da bemorning tibbiy tarixi yoki joriy davolash rejasining holatini tahlil qilishda bu imkoniyat juda foydalidir. Bundan tashqari, "ifodalarni baholash" (expression evaluation) vositasi yordamida dasturchi dastur to'xtagan paytda ixtiyoriy Java ifodasini yoki metod chaqiruvini kiritib, uning natijasini darhol ko'rishi mumkin. Bu, masalan, murakkab bir mantiqiy shartning haqiqiyligini tekshirish yoki kichik bir funksiyaning kutilgan natijani berayotganini tezda aniqlash uchun qulaydir. Chaqiruvlar steki (call stack) esa joriy to'xtash nuqtasiga qadar qaysi metodlar ketma-ketlikda chaqirilganligini ko'rsatib beradi, bu dasturning joriy holatga qanday yetib kelganligini tushunishga va xatolik manbasini yuqoriroq darajadagi chaqiruvlardan izlashga yordam beradi. Ushbu vositalarning majmuasi diskretlash jarayonini tizimli va samarali amalga oshirishga, natijada esa "PATIENT’S RECOVERY SYSTEM" kabi mas'uliyatli tizimlarning sifatini va ishonchliligini oshirishga xizmat qiladi (Rosenberg, 2000).

P5

Dasturiy ta'minotni ishlab chiqish jarayonida kodlash standarti muhim o'rin tutadi; u dasturchilar jamoasi yoki yakka tartibdagi mutaxassis tomonidan kod yozishda rioya etilishi lozim bo'lgan qoidalar, yo'riqnomalar va eng ilg'or tajribalar (best practices) majmuasini o'zida mujassam etadi. "PATIENT’S RECOVERY SYSTEM" (PRS) loyihasini Java dasturlash tilida amalga oshirish rejalashtirilganligi sababli, ushbu til uchun sanoatda keng e'tirof etilgan va o'zini amalda oqlagan kodlash standartlaridan birini tanlash hamda unga izchil amal qilish loyihaning muvaffaqiyati uchun muhim shartlardan biridir. Ushbu PRS loyihasi doirasida kod yozish uchun **Google Java Style Guide** (Google, n.d.) standarti asos sifatida qabul qilinishi tavsiya etiladi. Mazkur standart o'zining aniq va tushunarli qoidalari, kodning bir xilligiga (consistency) va o'qiluvchanligiga (readability) qaratilgan kuchli urg'usi bilan ajralib turadi va ko'plab yirik dasturiy loyihalarda muvaffaqiyatli qo'llaniladi. Bu standart dasturiy kodning turli jihatlarini, jumladan, nomlash qoidalaridan tortib, formatlash uslublari, izohlar yozish tartibi va hatto ba'zi bir dasturlash amaliyotlarigacha qamrab oladi.

Xususan, nomlash qoidalariga ko'ra, paketlar (packages) faqat kichik harflar bilan, so'zlar esa nuqtalar orqali ajratilgan holda yozilishi (masalan, com.example.prs.patient.service), klasslar va interfeyslarning nomlari esa har bir so'zi bosh harf bilan boshlanuvchi "CamelCase" uslubida (masalan, PatientMedicalRecord yoki TreatmentRecommendationEngine) ifodalanishi belgilanadi. Metodlar va o'zgaruvchilar uchun esa birinchi so'zi kichik harf bilan, keyingi so'zlari bosh harf bilan yoziladigan "lowerCamelCase" (masalan, calculateRiskScore yoki lastConsultationDate) qo'llaniladi. Konstantalar, ya'ni o'zgarmas qiymatlar esa barcha harflari katta va so'zlar orasida pastki chiziqcha ishlatiladigan "UPPER\_SNAKE\_CASE" (masalan, DEFAULT\_MONITORING\_INTERVAL\_HOURS) uslubida nomlanadi. Bunday yagona nomlash tizimi kodning turli elementlarini bir qarashda ajratib olishni va ularning vazifasini tezroq anglashni osonlashtiradi.

Formatlash jihatlariga kelsak, Google Java Style Guide kod chekinishlari (indentation) uchun odatda ikkita bo'sh joy (space) ishlatilishini tavsiya etadi va tabulyatsiya belgilaridan foydalanishni ma'qullamaydi, bu esa turli matn muharrirlari va IDE larda kodning bir xil ko'rinishini ta'minlaydi. Qatorlarning maksimal uzunligi ham belgilangan (odatda 100 belgi), bu esa kodni gorizontal skroll qilmasdan o'qish imkonini beradi. Figurali qavslarni ({ }) ishlatish tartibi ham aniq ko'rsatilgan; masalan, if, for, while kabi boshqaruvchi konstruksiyalar uchun ular har doim, hatto blok bitta qatordan iborat bo'lsa ham, ishlatilishi shartligi va ochiluvchi qavsning odatda o'zidan oldingi bayonot bilan bir qatorda joylashishi kabi qoidalar kodning vizual tuzilishini bir xillashtiradi.

Izohlar yozish borasida esa, standart kodning murakkab yoki tushunilishi qiyin bo'lgan qismlarini sharhlab beruvchi, ammo ortiqcha yoki eskirgan ma'lumotlardan xoli bo'lgan izohlardan foydalanishni rag'batlantiradi. Ayniqsa, ommaviy (public) klasslar, interfeyslar va metodlar uchun Javadoc formatidagi izohlarni yozish muhim sanaladi, chunki bu loyiha uchun avtomatik tarzda texnik hujjatlar generatsiya qilish imkonini beradi (Bloch, 2018). Importlar tartibi, bayonotlarni joylashtirish, bo'sh qatorlardan foydalanish kabi boshqa ko'plab mayda detallar ham standart tomonidan qamrab olinib, ularning barchasi kodning umumiy sifati va o'qiluvchanligini oshirishga xizmat qiladi. PRS loyihasida ushbu standartga amal qilish, ayniqsa jamoa bo'lib ishlaganda, har bir dasturchi tomonidan yozilgan kodning boshqalar uchun tushunarli bo'lishini, kodni ko'zdan kechirish (code review) jarayonlarining samaraliroq o'tishini va loyihani kelajakda qo'llab-quvvatlashni ancha yengillashtiradi. Ko'pgina zamonaviy IDE lar, jumladan IntelliJ IDEA, bunday kodlash standartlarini avtomatik tarzda qo'llash va tekshirish uchun maxsus sozlamalar va plaginlarga ega, bu esa standartga rioya qilish jarayonini soddalashtiradi va inson omili bilan bog'liq xatoliklarni kamaytiradi.

M4

Nosozliklarni tuzatish, ya'ni diskretlash jarayoni, dasturiy ta'minot ishlab chiqish siklining shunchaki bir bosqichi bo'lib qolmasdan, u yaratilayotgan ilovalarning xavfsizroq va mustahkamroq (robust) bo'lishiga fundamental hissa qo'shadigan muhim amaliyot sifatida qaralishi lozim. "PATIENT’S RECOVERY SYSTEM" (PRS) kabi tibbiyot sohasiga mo'ljallangan va inson sog'lig'i bilan bevosita ishlaydigan dasturiy ta'minot uchun bu jihatlar ayniqsa dolzarbdir, chunki tizimdagi har qanday kichik nuqson yoki zaiflik bemorning ahvoliga salbiy ta'sir ko'rsatishi, noto'g'ri davolash tavsiyalariga olib kelishi va hatto hayot uchun xavf tug'dirishi mumkin (Leveson, 1995; OM-5 da "Noto'g'ri xulosalar va noto'g'ri tavsiflar mumkin"ligi ta'kidlangan). Diskretlash jarayonining ilovalarni xavfsizroq va mustahkamroq qilishga yordam berishining bir necha muhim yo'nalishlari mavjud.

Avvalo, puxta amalga oshirilgan diskretlash dasturchilarga o'zlari yozgan kodning ichki ishlash mantig'ini va ma'lumotlar oqimini chuqurroq anglab yetish imkoniyatini beradi. Dasturning bajarilishini to'xtash nuqtalari orqali nazorat qilish, o'zgaruvchilarning qiymatlarini real vaqtda kuzatish hamda kodni qadamma-qadam tahlil qilish orqali dasturchi dasturning turli kirish ma'lumotlari va sharoitlarida o'zini qanday tutishini aniq ko'ra oladi. Bu nafaqat mavjud xatolikni topib tuzatish uchun, balki kodning kutilmagan xatti-harakatlarini, yashirin mantiqiy nomuvofiqliklarni va potensial zaif nuqtalarini fosh qilish uchun ham qimmatlidir. PRS tizimida, masalan, bemorning bir nechta kasalliklari va qabul qilayotgan dori-darmonlari o'rtasidagi o'zaro ta'sirni hisobga oluvchi murakkab algoritmni diskretlash orqali uning barcha holatlar uchun to'g'ri ishlayotganiga ishonch hosil qilish mumkin.

Bundan tashqari, diskretlash jarayoni dasturning chekka holatlarga (edge cases) va kutilmagan yoki noto'g'ri formatdagi kirish ma'lumotlariga nisbatan qanchalik chidamli ekanligini baholash uchun muhim vositadir. Ko'pincha dasturlar standart, kutilgan sharoitlarda benuqson ishlaydi, ammo chegaraviy qiymatlar (masalan, juda katta yoki juda kichik sonlar, bo'sh satrlar, null qiymatlar) yoki foydalanuvchi tomonidan xato kiritilgan ma'lumotlar bilan to'qnashganda kutilmagan xatoliklarga yoki hatto ishdan chiqishga olib kelishi mumkin. Diskretlash paytida bunday nostandart holatlarni sun'iy ravishda yaratib (masalan, IDE vositalari yordamida o'zgaruvchilarning qiymatlarini o'zgartirib) yoki maxsus test ma'lumotlari orqali tizimga uzatib, uning bunday vaziyatlarga qanday munosabatda bo'lishini sinchkovlik bilan tekshirish mumkin. Bu esa PRS tizimining mustahkamligini, ya'ni uning kutilmagan vaziyatlarda ham nazorat ostida, oldindan belgilangan tartibda ishdan chiqishini (graceful degradation) yoki xatolikni to'g'ri qayta ishlab, foydalanuvchiga tushunarli xabar berishini ta'minlashga yordam beradi, bu esa tizimning umumiy ishonchliligini oshiradi (McConnell, 2004).

Shuningdek, samarali diskretlash amaliyoti dasturiy ta'minotdagi xavfsizlik zaifliklarini aniqlash va ularni bartaraf etishda muhim rol o'ynashi mumkin. Ba'zi keng tarqalgan dasturlash xatolari, masalan, buferning to'lib ketishi (buffer overflow), SQL in'ektsiyasi uchun sharoit yaratuvchi ma'lumotlarni yetarlicha validatsiya qilmaslik yoki resurslarga ruxsatsiz kirish imkonini beruvchi mantiqiy kamchiliklar, jiddiy xavfsizlik muammolariga olib kelishi mumkin. Diskretlash jarayonida ma'lumotlarning dastur ichida qanday harakatlanishini, qayta ishlanishini va saqlanishini batafsil kuzatish orqali bunday potensial xavf tug'diruvchi nuqtalarni aniqlash imkoniyati paydo bo'ladi. PRS tizimi bemorlarning o'ta maxfiy shaxsiy va tibbiy ma'lumotlarini (OM-4 da "Medical History" va "Patient's unexpected symptoms" kabi bilim aktivlari keltirilgan) saqlashi va qayta ishlashi sababli, uning xavfsizligini ta'minlash masalasi birinchi darajali ustuvorlikka ega va diskretlash bu borada muhim himoya qatlamini yaratishga yordam beradi.

Nihoyat, diskretlash jarayonida aniqlangan har bir xatolik va uning kelib chiqish sabablari ko'pincha mavjud test holatlarini (test cases) takomillashtirish va yangilarini yaratish uchun qimmatli ma'lumot manbai bo'lib xizmat qiladi. Har bir muhim xatolik topilib, tuzatilgandan so'ng, aynan shu xatolikni fosh qiluvchi avtomatlashtirilgan test (odatda regressiya testi deb ataladi) yozilishi lozim. Bu kelajakda kodga o'zgartirishlar kiritilganda yoki yangi funksiyalar qo'shilganda aynan shu xatolikning tasodifan qayta yuzaga kelishining oldini olishga yordam beradi. Bunday yondashuv PRS tizimining uzoq muddatli barqarorligini, ishonchliligini va sifatini bosqichma-bosqich oshirib borish imkonini beradi. Xulosa qilib aytganda, diskretlash shunchaki yuzaga kelgan muammolarni bartaraf etish usuli emas, balki proaktiv tarzda dasturiy ta'minotning sifatini, xavfsizligini va mustahkamligini oshirishga qaratilgan uzluksiz takomillashtirish jarayonining ajralmas qismidir (Brooks, 1995).

D4

Kodlash standartining zaruriyati va ahamiyati dasturiy ta'minot ishlab chiqish jarayonida, ayniqsa ko'p sonli ishtirokchilarni o'z ichiga olgan jamoaviy loyihalarda ham, yakka tartibda ishlayotgan individual dasturchi uchun ham birdek muhimdir va bu holat "PATIENT’S RECOVERY SYSTEM" (PRS) kabi mas'uliyatli tizimlar misolida yanada yaqqolroq namoyon bo'ladi. Ushbu standartlarga rioya qilish shunchaki tartib-intizom masalasi bo'lib qolmasdan, balki loyihaning umumiy sifati, samaradorligi va uzoq muddatli istiqbollariga bevosita ta'sir ko'rsatuvchi fundamental omil sifatida baholanishi lozim.

Jamoa miqyosida yagona kodlash standartining joriy etilishi bir qator muhim afzalliklarni taqdim etadi. Eng avvalo, u kod bazasining umumiy **o'qiluvchanligi va tushunarliligini** keskin oshiradi. Barcha jamoa a'zolari bir xil nomlash qoidalari, formatlash uslublari va izoh yozish tamoyillariga amal qilganda, har bir dasturchi uchun nafaqat o'zi yozgan, balki hamkasblari tomonidan yaratilgan kod qismlarini ham tezda anglab yetishi va kerak bo'lganda ularga o'zgartirish kiritishi ancha osonlashadi (Martin, 2008). Aks holda, har kim o'zicha yozgan kodlar yig'indisi vaqt o'tishi bilan chalkash, tushunilishi qiyin bo'lgan "spagetti kod"ga aylanib qolishi mumkin, bu esa yangi xodimlarning loyihaga kirishishini (onboarding) sekinlashtiradi va umumiy ish unumdorligini pasaytiradi. PRS tizimining murakkab tibbiy mantig'i va ko'plab o'zaro bog'liq komponentlarini (OM-1, OM-2, OM-3 da keltirilgan tashkiliy va vazifa modellari bunga ishora qiladi) hisobga olsak, kodning har bir qismi jamoaning barcha a'zolari uchun birdek tushunarli bo'lishi loyihaning muvaffaqiyati uchun hal qiluvchi ahamiyatga ega. Qolaversa, standartlashtirilgan kod **xatoliklarni kamaytirishga** ham xizmat qiladi; ko'pgina kodlash standartlari xatolarga moyil bo'lgan amaliyotlardan tiyilishni va xatoliklarni qayta ishlashning ishonchli usullarini qo'llashni tavsiya etadi, bu esa dastlabki bosqichlardayoq ko'plab nuqsonlarning oldini olishga yordam beradi. Bir xil uslubda yozilgan kodni ko'zdan kechirish (code review) jarayoni ham ancha samaraliroq o'tadi, chunki tekshiruvchilar uslubiy kelishmovchiliklarga chalg'imasdan, ko'proq kodning mantig'i va to'g'riligiga e'tibor qaratishlari mumkin. Shuningdek, dasturiy ta'minotning hayotiy sikli davomida unga **texnik xizmat ko'rsatish va o'zgartirishlar kiritish** zarurati tug'ilganda (PRS uchun bu muntazam yangilanishlar va yangi tibbiy protokollarga moslashishni anglatishi mumkin, OM-5), standartlarga asoslangan kod bazasi bilan ishlash ancha oson va xavfsizroq kechadi, hatto loyihaning dastlabki ishlab chiquvchilari jamoada bo'lmasa ham. Bu esa jamoaning umumiy **samaradorligini oshiradi** va dasturchilarga kodning tashqi ko'rinishi yoki formatlanishi kabi ikkilamchi masalalarga emas, balki asosiy funksionallikni yaratishga ko'proq vaqt ajratish imkonini beradi.

Individual dasturchi nuqtai nazaridan ham kodlash standartlariga rioya qilish bir qator muhim foyda keltiradi. Birinchidan, bu **shaxsiy intizomni shakllantiradi** va dasturchining o'z ishiga bo'lgan professional yondashuvini tarbiyalaydi. Ikkinchidan, vaqt o'tib, dasturchi o'zi ilgari yozgan kodga qayta murojaat qilganida, agar u standartlarga amal qilgan bo'lsa, o'sha kodning mantig'ini eslashi va tushunishi ancha oson bo'ladi – bu "kelajakdagi o'zim uchun" yozish tamoyiliga mos keladi. Uchinchidan, shaxsiy darajada ham standartlarga rioya qilish ko'plab mayda xatoliklarning oldini olishga yordam beradi va diskretlashga sarflanadigan qimmatli vaqtni tejaydi. To'rtinchidan, sanoatda keng qo'llaniladigan kodlash standartlarini bilish va ularga amal qilish dasturchining yangi jamoalarga va loyihalarga **integratsiyalashuvini osonlashtiradi** hamda uning mehnat bozoridagi raqobatbardoshligini oshiradi.

Albatta, ba'zi hollarda kodlash standartlari haddan tashqari qat'iy yoki dasturchining individual ijodkorligini cheklovchi omil sifatida qabul qilinishi mumkin. Ba'zi dasturchilar ma'lum bir formatlash uslubi yoki nomlash qoidasi ularning shaxsiy preferensiyalariga mos kelmasligini ta'kidlashlari mumkin. Biroq, tanqidiy nuqtai nazardan shuni anglash kerakki, kodlash standartlarining asosiy maqsadi shaxsiy erkinlikni bo'g'ish emas, balki jamoaviy ishda tartibsizlikning oldini olish, dasturiy ta'minotning umumiy sifatini oshirish va uning uzoq muddatli barqarorligini ta'minlashdir (Fowler, 1999). Yaxshi ishlab chiqilgan kodlash standarti odatda qat'iy qoidalar bilan birga ma'lum darajada moslashuvchanlikni ham nazarda tutadi va eng muhimi, u jamoa tomonidan kelishilgan va qabul qilingan bo'lishi kerak. "PATIENT’S RECOVERY SYSTEM" kabi inson hayoti va sog'lig'iga daxldor bo'lgan tizim uchun kodlash standartlarining yo'qligi yoki ularga e'tiborsizlik bilan qarash jiddiy salbiy oqibatlarga, jumladan, bemor xavfsizligiga tahdid soluvchi yashirin xatoliklarning paydo bo'lishiga olib kelishi mumkin. Shu bois, Google Java Style Guide kabi keng qamrovli va sinovdan o'tgan standartni tanlash va unga barcha jamoa a'zolari tomonidan qat'iy rioya etilishini ta'minlash nafaqat tavsiya etiladigan amaliyot, balki PRS loyihasining muvaffaqiyati va ishonchliligi uchun zaruriy shart hisoblanadi. Bu standartlar shunchaki qog'ozdagi qoidalar bo'lib qolmasdan, balki muntazam kod ko'zdan kechirishlari (code reviews), avtomatlashtirilgan linting vositalari va jamoa ichidagi doimiy muloqot orqali amalda qo'llab-quvvatlanib borilishi lozim.

Xulosa

"PATIENT’S RECOVERY SYSTEM" (PRS) loyihasini ishlab chiqish va uning funksional imkoniyatlarini tahlil qilish shuni ko'rsatadiki, zamonaviy axborot texnologiyalari tibbiyot sohasidagi, xususan, operatsiyadan keyingi bemorlarni parvarish qilish va ularning tiklanish jarayonlarini boshqarishdagi ko'plab dolzarb muammolarni hal etishda ulkan salohiyatga ega. Mazkur tizim, bemorning kasalxonadan chiqqandan keyingi davrda shifokor bilan aloqasining uzilib qolishi, kutilmagan salbiy alomatlarning o'z vaqtida aniqlanmasligi va bemorning o'z sog'lig'i borasidagi noaniqliklari kabi bir qator jiddiy kamchiliklarni bartaraf etishga qaratilgan bo'lib, u nafaqat bemorlar uchun, balki tibbiyot muassasalari va ularning xodimlari uchun ham sezilarli afzalliklarni taqdim etishi mumkin. Tizimning bemor ahvolini masofadan turib muntazam monitoring qilish, kiritilgan ma'lumotlar asosida individual tavsiyalar berish, shifokorlarni o'z vaqtida ogohlantirish va hatto favqulodda vaziyatlarda avtomatik tarzda tez yordam chaqirish kabi funksiyalari (scenario description) bemor parvarishining sifatini oshirish, asoratlarning oldini olish va tiklanish muddatlarini qisqartirishga xizmat qiladi. Bu esa, o'z navbatida, bemor uchun stresssiz uy sharoitida tezroq sog'ayish imkonini yaratib (OM-5 "Nomoddiy imtiyozlar"), kasalxonalarning cheklangan resurslaridan (OM-1 "Cheklangan kasalxona sig'imi") yanada samaraliroq foydalanishga, xususan, bitta tibbiy mutaxassisning qisqa vaqt ichida ko'proq bemorlarga sifatli xizmat ko'rsatishiga (OM-5 "Yagona tibbiy mutaxassis juda ko'p sonli bemorlarni qisqa vaqt ichida hal qilishi mumkin") zamin yaratadi.

Bunday murakkab va mas'uliyatli tizimni muvaffaqiyatli yaratish va joriy etish uchun dasturiy ta'minot muhandisligining barcha jabhalariga jiddiy e'tibor qaratish talab etiladi. Loyihaning asosini tashkil etuvchi algoritmlarning puxta ishlab chiqilishi va ularning tanlangan dasturlash tilida (masalan, Java Spring Boot texnologiyalari yordamida) samarali implementatsiya qilinishi (LO1) tizimning to'g'ri va ishonchli ishlashining garovidir. Protsessual, ob'ektga yo'naltirilgan va hodisalarga asoslangan dasturlash paradigmalarining o'ziga xos xususiyatlarini chuqur anglagan holda, ularni loyihaning turli komponentlarida maqsadli ravishda qo'llash (LO2) esa tizimning modulliligini, kengaytiriluvchanligini va qo'llab-quvvatlanish qulayligini ta'minlaydi. Integratsiyalashgan ishlab chiqish muhitlarining (IDE) keng imkoniyatlaridan unumli foydalanish (LO3) nafaqat kod yozish jarayonini tezlashtiradi, balki loyihani boshqarish, versiyalarni nazorat qilish va jamoaviy ishni samarali tashkil etishda ham muhim rol o'ynaydi. Va nihoyat, eng muhimi, "PATIENT’S RECOVERY SYSTEM" kabi inson hayoti bilan bevosita bog'liq bo'lgan tizim uchun diskretlash jarayonlariga alohida mas'uliyat bilan yondashish, xatoliklarni sinchkovlik bilan aniqlash va bartaraf etish hamda qat'iy kodlash standartlariga (masalan, Google Java Style Guide) izchil rioya qilish (LO4) tizimning xavfsizligi, mustahkamligi va umumiy sifatini kafolatlaydi. Ushbu jihatlarning barchasi birgalikda PRS ning nafaqat texnik jihatdan mukammal, balki amaliyotda ham bemorlar va tibbiyot xodimlari uchun haqiqiy qiymat yaratadigan vositaga aylanishiga xizmat qiladi. Kelajakda bunday tizimlarni sun'iy intellekt va mashinali o'qitish texnologiyalari bilan yanada boyitish orqali kasalliklarni bashorat qilish aniqligini oshirish, davolash rejalarini yanada individuallashtirish va tibbiy yordam ko'rsatish samaradorligini yangi bosqichga olib chiqish mumkinligi shubhasiz (Yu et al., 2018). Shu bois, "PATIENT’S RECOVERY SYSTEM" loyihasi nafaqat muayyan bir muammoning yechimi, balki sog'liqni saqlash sohasida raqamli transformatsiyalarning muhim bir yo'nalishi sifatida qaralishi lozim.

ADABIYOTLAR RO'YXATI

1. Aho, A.V., Lam, M.S., Sethi, R. and Ullman, J.D. (2007). *Compilers: Principles, Techniques, and Tools*. 2nd edn. Boston: Pearson/Addison-Wesley.
2. Beauchamp, T.L. and Childress, J.F. (2013). *Principles of Biomedical Ethics*. 7th edn. New York: Oxford University Press.
3. Bloch, J. (2018). *Effective Java*. 3rd edn. Boston, MA: Addison-Wesley Professional.
4. Booch, G., Maksimchuk, R.A., Engle, M.W., Young, B.J., Conallen, J. and Houston, K.A. (2007). *Object-Oriented Analysis and Design with Applications*. 3rd edn. Upper Saddle River, NJ: Addison-Wesley Professional.
5. Brooks, F.P. Jr. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. 2nd edn. Reading, MA: Addison-Wesley Professional.
6. Chacon, S. and Straub, B. (2014). *Pro Git*. 2nd edn. New York, NY: Apress.
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2009). *Introduction to Algorithms*. 3rd edn. Cambridge, MA: MIT Press.
8. Deitel, P.J. and Deitel, H.M. (2017). *Java How to Program, Early Objects*. 11th edn. Harlow: Pearson Education Limited.
9. Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Reading, MA: Addison-Wesley Professional. (Shuningdek, Fowler, M. (2018). *Refactoring: Improving the Design of Existing Code*. 2nd edn. Boston, MA: Addison-Wesley Professional. ham muhim).
10. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.
11. Google (n.d.). *Google Java Style Guide*. [Online]. Available at: <https://google.github.io/styleguide/javaguide.html> (Accessed: 01 June 2025).
12. Hanly, J.R. and Koffman, E.B. (2016). *Problem Solving and Program Design in C*. 8th edn. Boston: Pearson.
13. Hunt, A. and Thomas, D. (2000). *The Pragmatic Programmer: From Journeyman to Master*. Reading, MA: Addison-Wesley.
14. Kaner, C., Falk, J. and Nguyen, H.Q. (2013). *Testing Computer Software*. 2nd edn. New York: John Wiley & Sons. (Yoki Myers, G.J., Sandler, C. and Badgett, T. (2011). *The Art of Software Testing*. 3rd edn. Hoboken, NJ: John Wiley & Sons.)
15. Knuth, D.E. (1997). *The Art of Computer Programming, Volumes 1-3 Boxed Set*. 3rd edn. Reading, Massachusetts: Addison-Wesley Professional.
16. Leveson, N.G. (1995). *Safeware: System Safety and Computers*. Reading, MA: Addison-Wesley.
17. Lindholm, T., Yellin, F., Bracha, G. and Buckley, A. (2014). *The Java® Virtual Machine Specification, Java SE 8 Edition*. Boston, MA: Addison-Wesley.
18. Martin, R.C. (2003). *Agile Software Development, Principles, Patterns, and Practices*. Upper Saddle River, NJ: Prentice Hall.
19. Martin, R.C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall.
20. McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction*. 2nd edn. Redmond, WA: Microsoft Press.
21. Pressman, R.S. and Maxim, B.R. (2015). *Software Engineering: A Practitioner's Approach*. 8th edn. New York: McGraw-Hill Education.
22. Rosenberg, J. (2000). *How Debuggers Work: Algorithms, Data Structures, and Architecture*. New York: John Wiley & Sons.
23. Sebesta, R.W. (2016). *Concepts of Programming Languages*. 11th edn. Boston: Pearson.
24. Shortliffe, E.H. and Cimino, J.J. (eds.) (2014). *Biomedical Informatics: Computer Applications in Health Care and Biomedicine*. 4th edn. London: Springer.
25. Sommerville, I. (2016). *Software Engineering*. 10th edn. Harlow: Pearson Education Limited.
26. Stroustrup, B. (2013). *The C++ Programming Language*. 4th edn. Upper Saddle River, NJ: Addison-Wesley Professional.
27. Venners, B. (1998). *Inside the Java Virtual Machine*. New York: McGraw-Hill.
28. Walls, C. (2016). *Spring Boot in Action*. Shelter Island, NY: Manning Publications.
29. Yu, K.-H., Beam, A.L. and Kohane, I.S. (2018). Artificial intelligence in healthcare. *Nature Biomedical Engineering*, 2(10), pp.719-731.