|  |
| --- |
| Level 4 | Software Development Lifecycles |
| Abdulloh Xalilov |

**Unit 11:** Level 4 | Software Development Lifecycles

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

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

**Talaba ID:** 231397

**Taqdim etilgan sana:** 2.06.2025 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 | Software Development Lifecycles |
| **Topshiriq nomi:** | Level 4 | Software Development Lifecycles |
| **Topshiriq topshirilgan sana:** | 2.06.2025yil |

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:** 2.06.2025 yil |

Mundarija

[KIRISH 3](#_Toc199764724)

[LO1: P1 4](#_Toc199764725)

[P2 8](#_Toc199764726)

[M1 11](#_Toc199764727)

[D1 13](#_Toc199764728)

[LO2: P3 14](#_Toc199764729)

[P4 16](#_Toc199764730)

[M2 18](#_Toc199764731)

[D2 20](#_Toc199764732)

[LO3: P5 21](#_Toc199764733)

[P6 24](#_Toc199764734)

[M3 28](#_Toc199764735)

[M4 28](#_Toc199764736)

[D3 29](#_Toc199764737)

[LO4: P7 30](#_Toc199764738)

[M5 33](#_Toc199764739)

[M6 35](#_Toc199764740)

[D4 36](#_Toc199764741)

[Xulosa 38](#_Toc199764742)

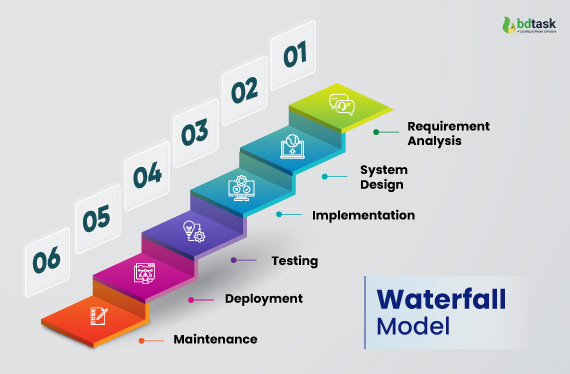
[ADABIYOTLAR RO'YXATI 38](#_Toc199764743)

KIRISH

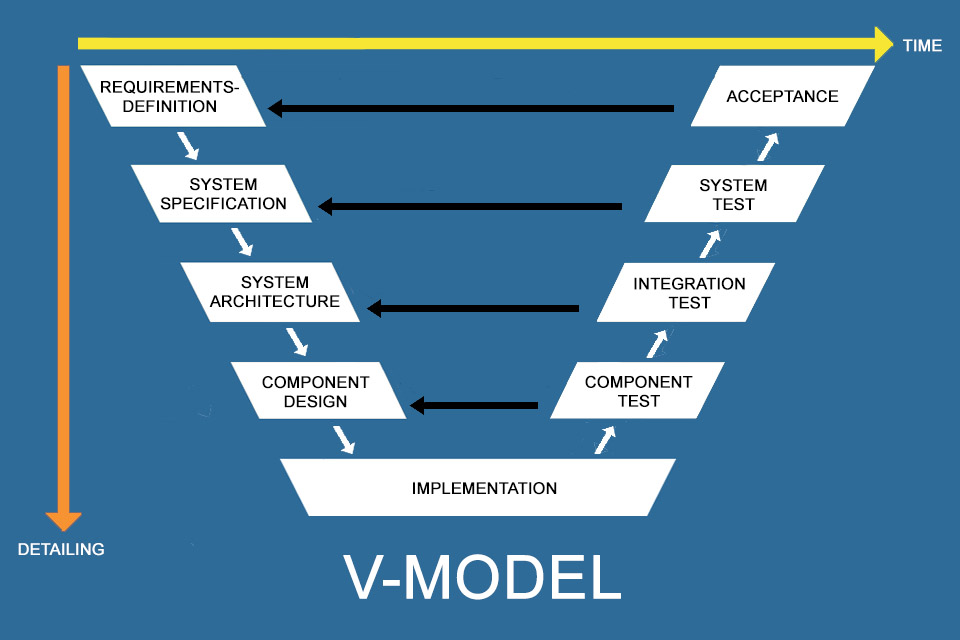
Zamonaviy dunyoda dasturiy ta'minotning roli tobora ortib borayotgan bir paytda, uni ishlab chiqish jarayonlariga tizimli va puxta yondashish muhim ahamiyat kasb etadi. Ushbu tahliliy ish dasturiy ta'minotni yaratishning turli jihatlarini, jumladan, loyihaning dastlabki texnik-iqtisodiy asoslanishidan tortib, ishlab chiqishning muqobil metodologiyalari va hayotiy siklning har bir bosqichida qo'llaniladigan vosita hamda usullargacha bo'lgan keng qamrovli masalalarni o'rganishga qaratilgan. Muhokamalar davomida dasturiy ta'minot loyihalarining muvaffaqiyatini ta'minlashda texnik-iqtisodiy asoslashning (TIA) ahamiyati, uning turli mezonlari (LO1, LO2), shuningdek, an'anaviy va Agile kabi ishlab chiqish yondashuvlarining (LO3) o'ziga xos xususiyatlari, afzalliklari va kamchiliklari ko'rib chiqildi. Bundan tashqari, talablarni tahlil qilishdan tortib, dizayn, amalga oshirish, sinovdan o'tkazish va joriy etishgacha bo'lgan dasturiy ta'minotni ishlab chiqish hayotiy siklining (SDLC) turli bosqichlarida qo'llaniladigan vositalar va usullarning maqsadi hamda samaradorligi (LO4) batafsil tahlil qilindi. Ushbu tahlil dasturiy ta'minotni ishlab chiqish sohasidagi nazariy bilimlar va amaliy yondashuvlar o'rtasidagi bog'liqlikni ochib berishga, shuningdek, turli darajadagi mezonlar (Pass, Merit, Distinction) doirasida chuqur mulohaza yuritishga intiladi. Maqsad – dasturiy ta'minotni ishlab chiqish jarayonining murakkabligini va unda qabul qilinadigan qarorlarning ko'p qirraliligini namoyish etishdir.

LO1: P1

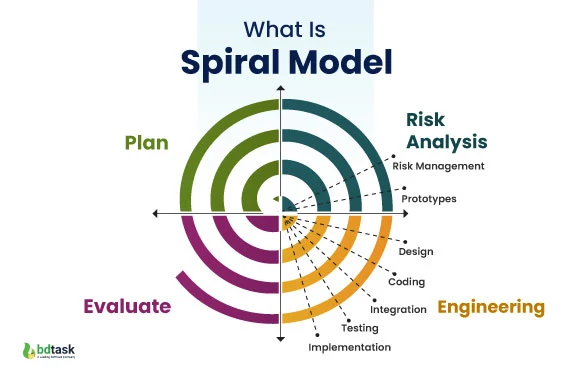
**Ikki iterativ va ikkita ketma-ket dasturiy ta'minotning hayot aylanishi modellarini tavsiflang.**

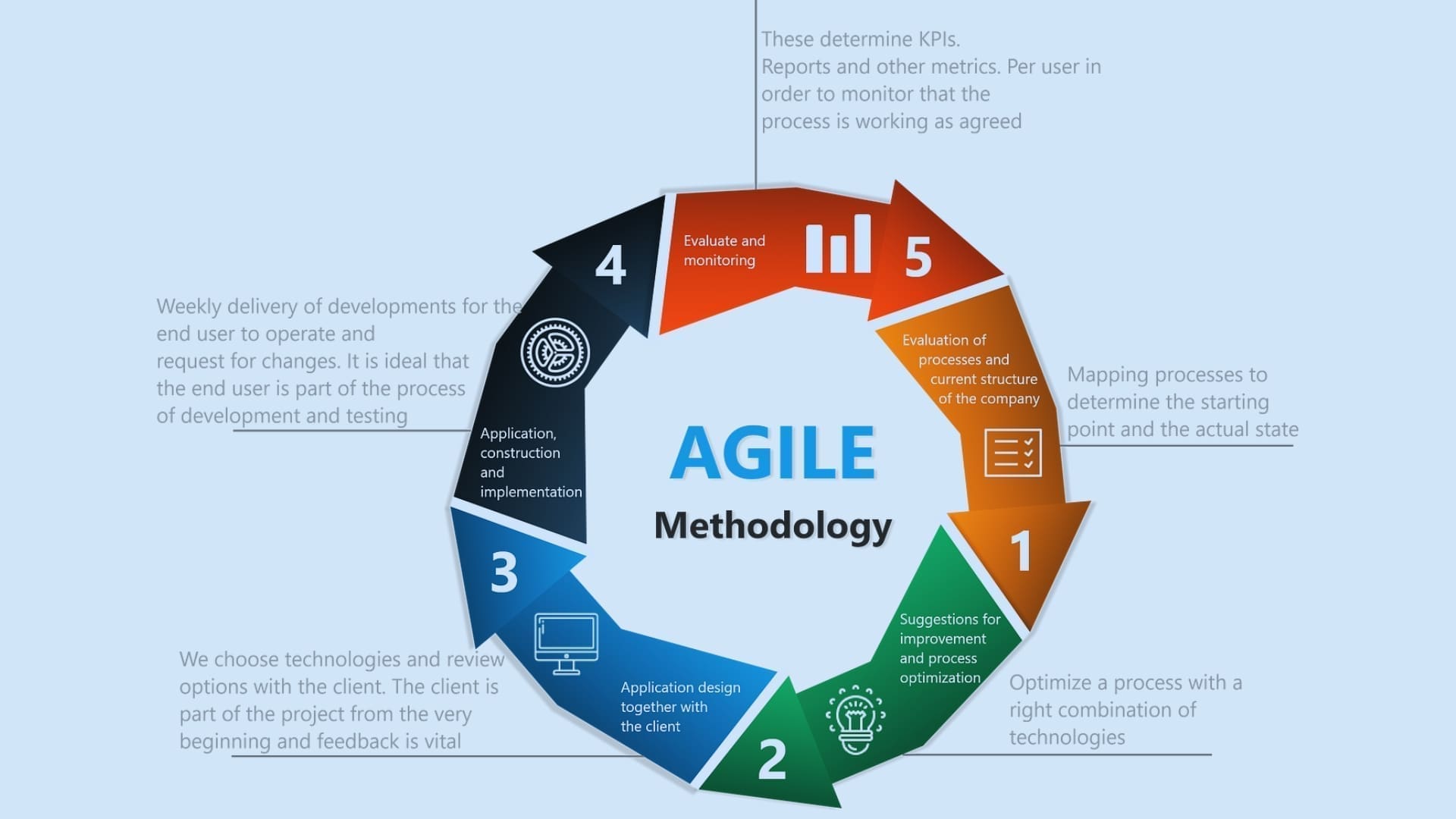
Dasturiy ta'minot ishlab chiqish hayotiy sikli modellaridan dastlab ketma-ket (sequential) turlarini ko'rib chiqamiz. Bu modellar ishlab chiqish jarayonini qat'iy tartibda, bir bosqich tugallanmasdan keyingisiga o'tilmaydigan tarzda amalga oshirishni nazarda tutadi.

Eng klassik ketma-ket model – bu **Waterfall Modeli**dir. Bu model dasturiy ta'minotni ishlab chiqish jarayoniga eng an'anaviy va qat'iy yondashuvlardan biri bo'lib, jarayon yuqoridan pastga qarab, bosqichma-bosqich, bir tomonlama harakat qiladi. Asosiy bosqichlari talablarni aniqlash va tahlil qilish, tizimni loyihalash, amalga oshirish (kodlash), integratsiya va sinov, joylashtirish hamda qo'llab-quvvatlash va ta'minotdan iborat. Waterfall modelining afzalliklari uning soddaligi, tushunarliligi va har bir bosqichda batafsil hujjatlashtirish talab etilishidir, bu esa loyihani boshqarishni osonlashtiradi. Biroq, uning jiddiy kamchiliklari ham mavjud: eng asosiysi – moslashuvchan emasligi. Talablar loyiha boshida to'liq va aniq belgilanishi shart, keyinchalik ularga o'zgartirish kiritish juda qiyin. Shuningdek, ishlaydigan dasturiy ta'minot faqatgina ishlab chiqish siklining oxirgi bosqichlaridagina paydo bo'ladi. Waterfall modeli, odatda, talablari juda aniq, barqaror va o'zgarish ehtimoli kam bo'lgan loyihalar uchun mos keladi.



**V-Modeli (V-Shaklidagi Model)** Waterfall modelining bir kengaytmasi bo'lib, ishlab chiqishning har bir bosqichiga mos keladigan sinov jarayonini parallel olib borishga alohida e'tibor qaratadi. Uning nomi V-shaklidagi diagrammadan kelib chiqqan: chap tomonda ishlab chiqish jarayonlari (talablarni tahlil qilish, tizimni loyihalash, arxitektura loyihasi, modulni loyihalash), V ning eng pastki nuqtasida kodlash, o'ng tomonda esa ularga mos keluvchi sinov jarayonlari (birlik testlari, integratsion testlash, tizimli testlash, qabul qilish testlari) joylashadi. Modelning asosiy g'oyasi verifikatsiya ("mahsulotni to'g'ri ishlab chiqyapmizmi?") va validatsiya ("to'g'ri mahsulotni ishlab chiqyapmizmi?") jarayonlarini har bir bosqichda amalga oshirishdir. Bu modelning asosiy afzalligi sinov jarayonlariga kuchli e'tibor qaratishidir, bu esa xatoliklarni erta aniqlashga yordam beradi. Ammo, V-modeli ham Waterfall kabi qat'iy bo'lib, talablar o'zgarganda moslashuvchan emas. V-modeli ko'pincha talablari aniq va barqaror bo'lgan, sinov jarayonlari juda muhim loyihalarda, masalan, yuqori ishonchlilik talab etiladigan tizimlarda qo'llaniladi.

Ketma-ket modellardan farqli o'laroq, iterativ modellar dasturiy ta'minotni bosqichma-bosqich, takrorlanuvchi sikllar orqali ishlab chiqishni nazarda tutadi. **Prototiplash Modeli** ana shunday yondashuvlardan biri bo'lib, uning asosiy maqsadi – dastlabki bosqichlarda foydalanuvchilarning talablarini aniqlashtirish va tizimning asosiy funksionalligini namoyish etuvchi ishchi model (prototip) yaratishdir. Ishlash tartibi talablarni yig'ish va aniqlashtirish, tezkor dizayn, prototipni yaratish, foydalanuvchi tomonidan baholash va prototipni takomillashtirish kabi qadamlarni o'z ichiga oladi. Bu jarayon talablar aniqlashguncha bir necha marta takrorlanishi mumkin, so'ngra yakuniy mahsulot ishlab chiqiladi. Prototiplashning eng katta yutug'i foydalanuvchilarning talablarini aniqlashtirishda va noto'g'ri tushunishlarni kamaytirishda samarali ekanligidir. Kamchiliklari esa foydalanuvchilarning ba'zan prototipni deyarli tayyor mahsulot deb qabul qilishlari va jarayonni boshqarishning qiyinligidir. Bu model, ayniqsa, foydalanuvchi interfeysi murakkab bo'lgan, talablar noaniq yoki innovatsion mahsulotlar uchun juda mos keladi.

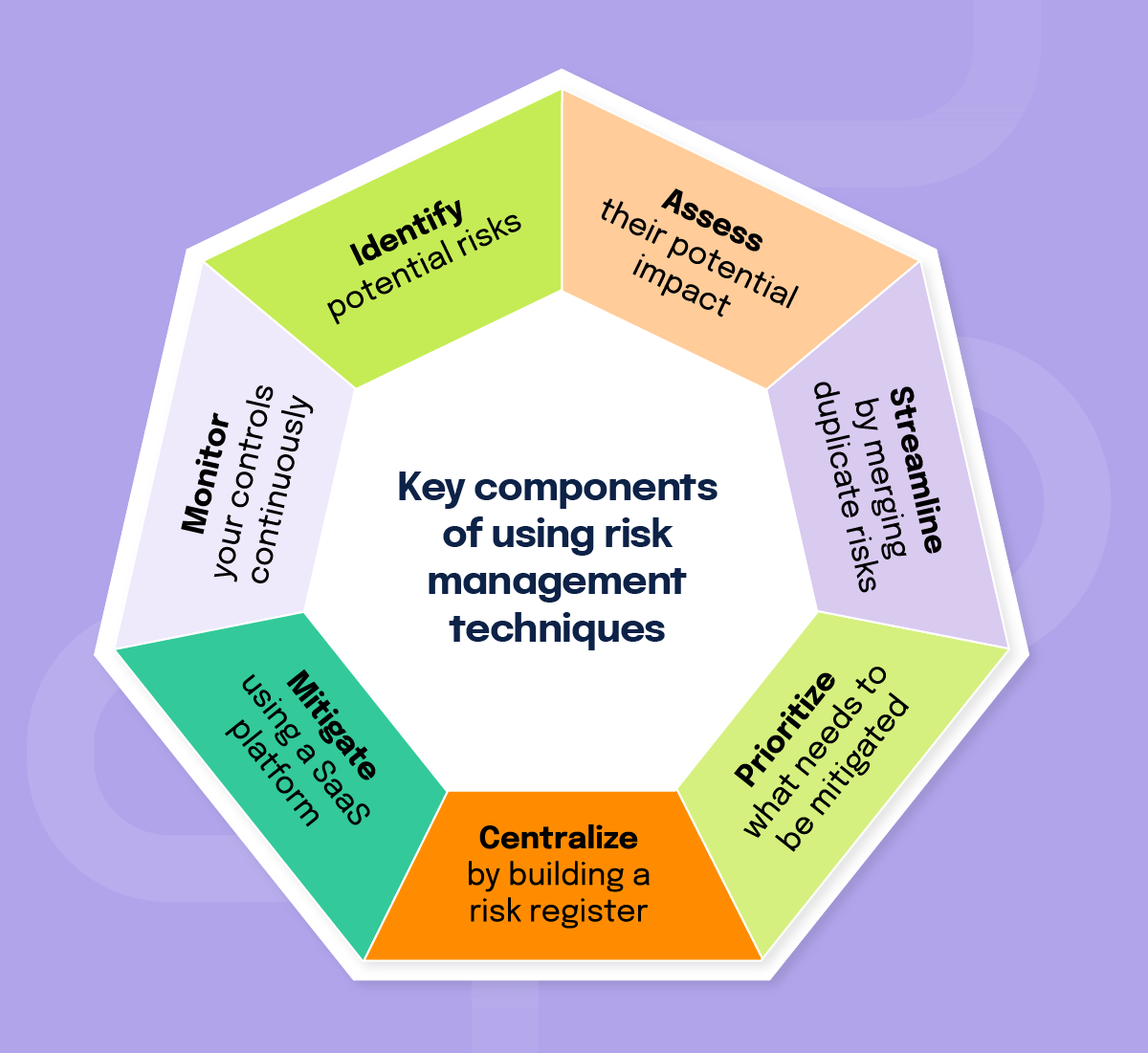
**Spiral Modeli**, Barry Boehm tomonidan taklif etilgan bo'lib, dasturiy ta'minotni ishlab chiqish jarayoniga risklarga yo'naltirilgan, iterativ yondashuvni taqdim etadi. Ushbu model Waterfall modelining tizimli yondashuvi bilan Prototiplash modelining iterativ tabiatini birlashtiradi va har bir iteratsiyada risklarni tahlil qilishga alohida urg'u beradi. Spiral modelining ishlash tartibi har bir aylanishda takrorlanadigan to'rtta asosiy kvadrant yoki faoliyat guruhidan iborat: maqsadlarni, muqobil yechimlarni va cheklovlarni aniqlash; risklarni baholash va tahlil qilish; mahsulotni ishlab chiqish va tekshirish; va keyingi iteratsiyani rejalashtirish. Spiral modelining afzalliklari uning risklarni boshqarishga kuchli e'tibor qaratishida va talablarning loyiha davomida o'zgarishiga moslashish imkonini berishidadir. Biroq, bu model ancha murakkab bo'lishi va uni samarali qo'llash uchun risklarni baholash sohasida yuqori malakali mutaxassislarni talab qilishi mumkin. Kichik loyihalar uchun u haddan tashqari ko'p vaqt va resurs talab qilishi mumkin. Spiral modeli asosan katta, qimmat va murakkab dasturiy ta'minot loyihalariga to'g'ri keladi, bu yerda risklarni boshqarish hal qiluvchi ahamiyatga ega.

**Agile (chaqqon, epchil) Modeli** aslida bitta aniq model emas, balki dasturiy ta'minotni ishlab chiqishga iterativ va inkremental yondashuvlarni o'zida mujassam etgan qadriyatlar va tamoyillar to'plamidir. Agile yondashuvining asosiy maqsadi – bu o'zgaruvchan talablarga moslasha oladigan, yuqori sifatli dasturiy ta'minotni tez-tez va muntazam ravishda yetkazib berishdir. Agile modelining ishlash tartibi qisqa iteratsiyalarga (odatda "sprint" deb ataladi) asoslanadi. Har bir iteratsiya rejalashtirish, talablarni tahlil qilish, dizayn, kodlash, sinovdan o'tkazish va foydalanuvchiga ishlaydigan dasturiy ta'minotning kichik bir qismini (inkrement) taqdim etish kabi barcha standart ishlab chiqish bosqichlarini qamrab oladi. Agile modelining afzalliklari uning yuqori darajadagi moslashuvchanligida, mijozning jarayonga faol jalb etilishida va xatolarning erta aniqlanishidadir. Kamchiliklari shundaki, u loyihaning boshida aniq muddatlar va byudjetni belgilashni qiyinlashtirishi mumkin va Agile muvaffaqiyati ko'p jihatdan jamoa a'zolarining tajribasiga bog'liqdir. Agile modelining qo'llanilishi ayniqsa talablari tez-tez o'zgarib turadigan, innovatsion yoki noaniqlik darajasi yuqori bo'lgan loyihalar uchun juda mos keladi.

Top of Form

P2

**Ushbu modellarda xavf qanday boshqarilishini tushuntiring.**



Dasturiy ta'minotni ishlab chiqish jarayonida risklarni boshqarish loyihaning muvaffaqiyatli yakunlanishi uchun hal qiluvchi ahamiyatga ega. Risklar – bu loyihaning maqsadlariga (masalan, muddatlar, byudjet, sifat, funksionallik) salbiy ta'sir ko'rsatishi mumkin bo'lgan har qanday noaniq hodisa yoki shart-sharoitdir. Turli SDLC modellari risklarni boshqarishga turlicha yondashadi.

**Waterfall Modelida Risklarni Boshqarish:** Waterfall modeli o'zining qat'iy ketma-ket tabiati tufayli risklarni boshqarishda unchalik kuchli emas. Ushbu modelda risklarni aniqlash va baholash asosan loyihaning eng boshlang'ich bosqichlarida, ya'ni talablarni aniqlash va rejalashtirishda amalga oshirilishi kutiladi. Biroq, loyiha boshlanganidan keyin yangi risklar paydo bo'lsa yoki dastlabki baholashlar noto'g'ri bo'lsa, ularga moslashish qiyin kechadi. Chunki model bir bosqichdan keyingisiga o'tishni nazarda tutadi va orqaga qaytish cheklangan. Agar loyihaning keyingi bosqichlarida jiddiy risklar yuzaga chiqsa (masalan, texnologik muammolar yoki talablarning noto'g'ri tushunilganligi), bu katta kechikishlarga va xarajatlarning oshishiga olib kelishi mumkin (Murch, 2012). Risklarni kamaytirishning asosiy usuli – bu dastlabki bosqichlarda imkon qadar puxta rejalashtirish va talablarni to'liq aniqlashdir. Biroq, bu har doim ham amaliy jihatdan mumkin emas.

**V-Modelida Risklarni Boshqarish:** V-modeli Waterfall modeliga qaraganda risklarni boshqarishga biroz ko'proq e'tibor qaratadi, chunki u har bir ishlab chiqish bosqichiga parallel ravishda sinov bosqichini qo'shadi. Bu yondashuv xatoliklarni va nomuvofiqliklarni (ya'ni, potentsial risklarni) ishlab chiqish jarayonining ertaroq bosqichlarida aniqlash imkonini beradi. Masalan, talablarni tahlil qilish bosqichida qabul qilish testlari rejalashtiriladi, bu esa talablarning to'g'ri tushunilganligini va ularning sinovdan o'tkazilishi mumkinligini tekshirishga yordam beradi. Shunga qaramay, V-modeli ham asosan ketma-ket model bo'lib, talablar o'zgarganda yoki kutilmagan yirik risklar paydo bo'lganda moslashuvchanlikda cheklovlarga ega. Risklarni boshqarish asosan sinov jarayonlari orqali amalga oshiriladi, bu esa reaktiv yondashuvdir, ya'ni risklar yuzaga kelgandan keyin ularni aniqlashga qaratilgan.

**Prototiplash Modelida Risklarni Boshqarish:** Prototiplash modeli risklarni, ayniqsa talablar bilan bog'liq risklarni boshqarishda ancha samaraliroqdir. Foydalanuvchilarga dastlabki bosqichlardanoq ishlaydigan prototipni taqdim etish orqali, talablarning noto'g'ri tushunilishi yoki to'liq bo'lmasligi kabi risklar kamaytiriladi (Dennis va Haley, 2009). Foydalanuvchilarning fikr-mulohazalari asosida prototipni takomillashtirish orqali, loyiha noto'g'ri yo'nalishda ketishining oldi olinadi. Shuningdek, texnik risklarni baholash uchun ham prototiplardan foydalanish mumkin; masalan, yangi texnologiyaning qo'llanilishi yoki murakkab algoritmlarning ishlashi prototip orqali tekshirilishi mumkin. Biroq, prototiplash jarayonining o'zi ham ba'zi risklarni keltirib chiqarishi mumkin, masalan, foydalanuvchilarning prototipga nisbatan noto'g'ri kutishlari yoki prototipni yaratishga haddan tashqari ko'p vaqt sarflanishi.

**Spiral Modelida Risklarni Boshqarish:** Spiral modeli risklarni boshqarishga eng ko'p e'tibor qaratadigan model hisoblanadi. Aslida, risklarni tahlil qilish va boshqarish uning markaziy elementidir. Har bir spiral aylanishi risklarni aniqlash, baholash va ularni kamaytirish choralari bilan boshlanadi (Boehm tomonidan ta'kidlanganidek). Bu model loyihaning har bir bosqichida eng muhim risklarni aniqlashga va ularga qarshi choralar ko'rishga imkon beradi. Agar risklar juda yuqori bo'lsa, loyihani to'xtatish yoki o'zgartirish to'g'risida qaror qabul qilinishi mumkin, bu esa katta yo'qotishlarning oldini oladi. Prototiplar, simulyatsiyalar va boshqa risklarni kamaytirish usullari Spiral modelining ajralmas qismidir. Bu yondashuv, ayniqsa, katta, murakkab va noaniqlik darajasi yuqori bo'lgan loyihalar uchun juda muhimdir. Biroq, Spiral modelini samarali qo'llash uchun risklarni tahlil qilish sohasida yuqori malakali mutaxassislar talab etiladi.

**Agile Modelida Risklarni Boshqarish:** Agile modellari ham risklarni boshqarishga o'ziga xos yondashuvga ega. Garchi Agile manifestida "risk menejmenti" atamasi to'g'ridan-to'g'ri tilga olinmasa-da, uning ko'plab tamoyillari va amaliyotlari risklarni kamaytirishga yordam beradi. Qisqa iteratsiyalar (sprintlar) orqali ishlaydigan dasturiy ta'minotni tez-tez yetkazib berish, loyiha noto'g'ri yo'nalishda ketayotganini erta aniqlash imkonini beradi (bu "bozor riski" yoki "mahsulotga moslik riski"ni kamaytiradi). Mijoz bilan doimiy hamkorlik va fikr-mulohazalarni olish talablar bilan bog'liq risklarni kamaytiradi. Kundalik uchrashuvlar va jamoaning yaqin hamkorligi muammolarni (potentsial risklarni) tezda aniqlash va hal qilishga yordam beradi. O'zgarishlarga tezkor javob berish qobiliyati esa kutilmagan risklar paydo bo'lganda ularga moslashish imkonini beradi. Shuningdek, Agile ko'pincha texnik mukammallikka e'tibor qaratadi (masalan, doimiy integratsiya, avtomatlashtirilgan testlash), bu esa texnik risklarni kamaytirishga yordam beradi (Ferguson, 2014). Biroq, Agile yondashuvida loyihaning umumiy strategik risklarini (masalan, uzoq muddatli byudjet yoki resurslar bilan bog'liq) boshqarish ba'zan qiyinroq bo'lishi mumkin, chunki asosiy e'tibor qisqa muddatli iteratsiyalarga qaratiladi.

Xulosa qilib aytganda, har bir SDLC modeli risklarni boshqarishga o'ziga xos yondashuvni taklif etadi. Waterfall va V-modellari kabi ketma-ket modellar risklarni asosan oldindan rejalashtirish va sinov orqali boshqarishga harakat qilsa, Prototiplash, Spiral va Agile kabi iterativ va moslashuvchan modellar risklarni jarayon davomida aniqlash, baholash va ularga moslashish orqali boshqarishga ko'proq e'tibor qaratadi. Loyiha uchun model tanlashda risklarni boshqarish qobiliyati muhim mezonlardan biri bo'lishi kerak.

M1

**Muayyan hayot aylanishi modeli nima uchun tanlanganligini misol bilan mu-hokama qiling rivojlanish muhiti uchun.**

Dasturiy ta'minotni ishlab chiqish uchun hayotiy sikl modelini tanlash loyihaning o'ziga xos xususiyatlari, jamoaning tajribasi, mavjud resurslar va tashkiliy madaniyat kabi ko'plab omillarga bog'liq. Muayyan bir rivojlanish muhiti uchun nima uchun ma'lum bir model tanlanganini tushunish uchun aniq bir misolni ko'rib chiqish foydalidir.

**Misol:** Keling, **katta bir bank uchun yangi onlayn banking tizimini ishlab chiqish loyihasi**ni ko'rib chiqaylik. Bunday muhit bir nechta muhim xususiyatlarga ega: yuqori darajadagi xavfsizlik talablari, mavjud tizimlar bilan integratsiya zarurati, tartibga soluvchi organlar tomonidan qat'iy nazorat va talablarning loyiha boshida nisbatan aniq va barqaror bo'lishi (garchi ba'zi o'zgarishlar bo'lishi mumkin bo'lsa ham). Shuningdek, bunday loyihalarda batafsil hujjatlashtirish muhim ahamiyatga ega.

Ushbu rivojlanish muhiti uchun **V-Modeli** yoki **Waterfall modelining modifikatsiyalangan varianti** (masalan, bosqichlar orasida cheklangan qaytish imkoniyati bilan) maqbul tanlov bo'lishi mumkin. Nima uchun?

Birinchidan, bank tizimlari uchun **xavfsizlik va ishonchlilik** birinchi darajali ahamiyatga ega. V-modeli har bir ishlab chiqish bosqichiga mos keladigan sinov bosqichini parallel ravishda olib borishga urg'u beradi. Bu esa, xatoliklarni va potentsial zaifliklarni ishlab chiqish jarayonining ertaroq bosqichlarida aniqlash va tuzatish imkonini beradi. Qabul qilish testlari, tizimli testlash, integratsion testlash va birlik testlarining aniq belgilanishi va rejalashtirilishi tizimning barcha jihatlari sinchkovlik bilan tekshirilishini ta'minlaydi (Murch, 2012).

Ikkinchidan, **talablarning nisbatan barqarorligi**. Garchi onlayn banking sohasida yangi funksiyalar paydo bo'lishi mumkin bo'lsa-da, asosiy bank operatsiyalari (hisob raqamini ko'rish, pul o'tkazish, to'lovlarni amalga oshirish) standartlashgan va loyiha boshida aniq belgilanishi mumkin. Waterfall va V-modellari aynan shunday, talablari oldindan yaxshi aniqlangan loyihalar uchun mo'ljallangan.

Uchinchidan, **batafsil hujjatlashtirish zarurati**. Bank sohasidagi loyihalar tartibga soluvchi organlar tomonidan tekshirilishi va auditdan o'tkazilishi sababli, har bir bosqichda batafsil hujjatlarning (talablar spetsifikatsiyasi, dizayn hujjatlari, test rejalari va natijalari) mavjudligi shart. Waterfall va V-modellari hujjatlashtirishga katta e'tibor qaratadi (freetutes.com).

To'rtinchidan, **mavjud tizimlar bilan integratsiya**. Yangi onlayn banking tizimi bankning mavjud ichki tizimlari (masalan, asosiy bank tizimi, mijozlar ma'lumotlar bazasi) bilan uzviy bog'lanishi kerak. V-modelidagi integratsion testlash bosqichi bu integratsiyaning to'g'ri ishlashini tekshirish uchun juda muhimdir.

Agar ushbu loyiha uchun, masalan, **Agile modeli** tanlangan bo'lsa, ba'zi qiyinchiliklar yuzaga kelishi mumkin edi. Agile talablarning loyiha davomida o'zgarishiga moslashuvchan bo'lsa-da, bank kabi yuqori darajada tartibga solinadigan sohada talablarning tez-tez va keskin o'zgarishi maqsadga muvofiq emas. Shuningdek, Agile yondashuvidagi hujjatlashtirishga kamroq e'tibor qaratilishi audit talablariga javob bermasligi mumkin. Biroq, Agile elementlari (masalan, qisqa iteratsiyalar orqali ayrim yangi funksiyalarni joriy etish yoki foydalanuvchi interfeysini takomillashtirish) V-modeli yoki Waterfall modeliga qo'shimcha sifatida qo'llanilishi mumkin.

Xulosa qilib aytganda, onlayn banking tizimini ishlab chiqish kabi muhit uchun V-modelining tanlanishi uning sinov jarayonlariga kuchli e'tibor qaratishi, hujjatlashtirishni qo'llab-quvvatlashi va nisbatan barqaror talablar bilan ishlashga mosligi bilan izohlanadi. Bu tanlov loyihaning muvaffaqiyatli yakunlanishi uchun zarur bo'lgan xavfsizlik, ishonchlilik va tartibga soluvchi talablarga muvofiqlikni ta'minlashga yordam beradi.

D1

**Waterfall hayot aylanishi modelini katta hajmdagi dasturiy ta'minotni ishlab chiqish loyihasiga qo'llashning afzalliklarini baholang.**

Waterfall modelini katta hajmdagi dasturiy ta'minotni ishlab chiqish loyihalariga qo'llash masalasi bugungi kunda ko'plab munozaralarga sabab bo'lsa-da, ma'lum sharoitlarda uning ayrim afzalliklari namoyon bo'lishi mumkin. Ushbu afzalliklarni baholashda loyihaning o'ziga xos xususiyatlarini va muhitini hisobga olish muhimdir.

Katta hajmdagi dasturiy ta'minot loyihalari odatda ko'p sonli jamoa a'zolarini, murakkab o'zaro bog'liqliklarni va uzoq muddatli rejalashtirishni o'z ichiga oladi. Bunday sharoitda Waterfall modelining **tuzilmaviy va tartibli yondashuvi** ma'lum bir darajada boshqaruvni osonlashtirishi mumkin. Har bir bosqichning aniq belgilangan maqsadlari, natijalari va muddatlari katta jamoalarni muvofiqlashtirish va loyihaning umumiy borishini kuzatish uchun asos yaratadi. Loyihaning boshida **talablarni to'liq va batafsil aniqlashga qaratilgan urinish**, garchi amalda bunga erishish qiyin bo'lsa-da, agar muvaffaqiyatli amalga oshirilsa, keyingi bosqichlarda noaniqliklarni va o'zgarishlarni kamaytirishi mumkin. Bu, ayniqsa, byudjet va muddatlar qat'iy belgilangan va o'zgarishlarga yo'l qo'yish imkoniyati cheklangan hollarda muhim bo'lishi mumkin (Murch, 2012).

Waterfall modelining yana bir potentsial afzalligi uning **hujjatlashtirishga kuchli urg'u berishi** bilan bog'liq. Katta loyihalarda bilim almashinuvi, yangi jamoa a'zolarini jalb qilish va tizimni uzoq muddatli qo'llab-quvvatlash uchun batafsil hujjatlarning (talablar spetsifikatsiyasi, dizayn hujjatlari, test rejalari) mavjudligi muhim ahamiyat kasb etadi. Waterfall modeli har bir bosqichda bunday hujjatlarni yaratishni talab qiladi, bu esa loyiha haqida markazlashtirilgan ma'lumot manbasini shakllantirishga yordam beradi (freetutes.com). Bu, ayniqsa, xodimlar almashinuvi yuqori bo'lgan yoki loyiha bir necha bo'limlar yoki hatto tashkilotlar ishtirokida amalga oshiriladigan holatlarda foydalidir.

Bundan tashqari, agar katta hajmdagi loyiha **avval yaxshi o'rganilgan va tushunilgan sohada** amalga oshirilayotgan bo'lsa, ya'ni texnologiyalar barqaror va talablar nisbatan o'zgarmas bo'lsa, Waterfall modelining ketma-ket yondashuvi samara berishi mumkin. Bunday hollarda, loyihani oldindan batafsil rejalashtirish va bosqichma-bosqich amalga oshirish nisbatan kamroq risk bilan kechishi mumkin. Masalan, mavjud yirik tizimning yangi versiyasini yaratish (bu yerda asosiy funksionallik o'zgarmaydi) yoki standartlashtirilgan jarayonlarni avtomatlashtirish kabi loyihalar uchun Waterfall qisman qo'llanilishi mumkin.

Biroq, shuni ta'kidlash kerakki, Waterfall modelini katta hajmdagi zamonaviy dasturiy ta'minot loyihalariga sof shaklda qo'llash ko'pincha jiddiy kamchiliklarga olib keladi. Talablarning loyiha davomida o'zgarishi tabiiy hol bo'lib, Waterfall modelining bunga moslasha olmasligi, ishlaydigan dasturiy ta'minotning faqatgina oxirgi bosqichlarda paydo bo'lishi va risklarni samarali boshqarishdagi cheklovlari uning keng qo'llanilishini cheklaydi (IJCSI, 2010). Ko'pgina hollarda, katta loyihalar uchun iterativ, inkremental yoki Agile yondashuvlarining elementlarini o'z ichiga olgan gibrid modellar (masalan, bosqichli yetkazib berish bilan Waterfall yoki "Wagile" – Waterfall va Agile kombinatsiyasi) maqbulroq bo'lishi mumkin.

Xulosa qilib aytganda, Waterfall modelini katta hajmdagi dasturiy ta'minot loyihasiga qo'llashning afzalliklari asosan uning tuzilmaviyligi, hujjatlashtirishga urg'u berishi va talablari juda barqaror bo'lgan ma'lum turdagi loyihalarda rejalashtirishni osonlashtirishi bilan bog'liq. Ammo bu afzalliklar ko'pincha modelning moslashuvchan emasligi va o'zgarishlarga sekin javob berishi kabi jiddiy kamchiliklari bilan qoplanadi. Shuning uchun, bunday modelni tanlashdan oldin loyihaning barcha xususiyatlari va potentsial risklari sinchkovlik bilan baholanishi kerak.

LO2: P3

**Maqsadini tushuntiring fizibilite hisoboti.**

Texnik-iqtisodiy asoslash (TIA) hisoboti – bu taklif etilayotgan loyiha yoki tashabbusning hayotiyligini va amalga oshirishga arziydiganligini baholash uchun o'tkaziladigan tahlil natijalarini jamlovchi muhim hujjatdir. Uning asosiy maqsadi – qaror qabul qiluvchilarga (masalan, rahbariyat, investorlar, manfaatdor tomonlar) loyihani davom ettirish, o'zgartirish yoki undan voz kechish to'g'risida asoslangan qaror qabul qilishlari uchun zarur bo'lgan obyektiv ma'lumotlarni taqdim etishdir. Fizibilite hisoboti loyihaning turli jihatlarini, jumladan, texnik, iqtisodiy, huquqiy, operatsion va jadvalga muvofiqlik (TELOS – Technical, Economic, Legal, Operational, Schedule feasibility) kabi mezonlar bo'yicha sinchkovlik bilan o'rganadi (Dennis va Haley, 2009).

Fizibilite hisobotining birinchi va eng muhim maqsadi – bu **loyihaning texnik jihatdan amalga oshirilishi mumkinligini aniqlashdir**. Bu degani, taklif etilayotgan tizimni yaratish uchun zarur bo'lgan texnologiyalar mavjudmi, jamoaning texnik malakasi yetarlimi va mavjud infratuzilma loyihani qo'llab-quvvatlay oladimi kabi savollarga javob topishdir. Agar loyiha texnik jihatdan imkonsiz yoki haddan tashqari murakkab bo'lsa, uni boshlash mantiqsiz bo'ladi.

Ikkinchi asosiy maqsad – bu **loyihaning iqtisodiy jihatdan maqsadga muvofiqligini baholashdir**. Bu xarajatlar va foydalarni tahlil qilishni (cost-benefit analysis) o'z ichiga oladi. Hisobot loyihani amalga oshirish uchun ketadigan barcha xarajatlarni (masalan, dasturiy ta'minot, uskunalar, ishchi kuchi, o'qitish) va loyihadan kutilayotgan foydalarni (masalan, daromadning oshishi, xarajatlarning kamayishi, samaradorlikning ortishi) baholaydi. Maqsad – loyihaning moliyaviy jihatdan o'zini oqlashini va investitsiyalarning qaytimini (ROI) aniqlashdir (Lejk va Deeks, 2002). Agar loyiha iqtisodiy jihatdan foydasiz bo'lsa, u qanchalik texnik jihatdan mukammal bo'lmasin, amalga oshirilmasligi mumkin.

Uchinchidan, fizibilite hisoboti **potentsial risklarni aniqlash va ularni baholashga** xizmat qiladi. Har qanday loyiha ma'lum darajada risklar bilan bog'liq. Hisobot ushbu risklarni (masalan, texnologik risklar, bozor risklari, moliyaviy risklar, operatsion risklar) oldindan ko'ra bilishga va ularni kamaytirish yoki bartaraf etish bo'yicha strategiyalarni ishlab chiqishga yordam beradi. Bu esa, loyihaning kutilmagan muammolarga duch kelish ehtimolini kamaytiradi.

To'rtinchidan, hisobot **turli xil muqobil yechimlarni ko'rib chiqish va ularni taqqoslash** imkonini beradi. Ko'pincha, ma'lum bir muammoni hal qilishning bir nechta yo'li mavjud bo'ladi. Fizibilite hisoboti har bir muqobil yechimning afzalliklari, kamchiliklari, xarajatlari va risklarini tahlil qilib, eng maqbul variantni tanlashga yordam beradi.

Beshinchidan, fizibilite hisoboti **qaror qabul qilish uchun asos bo'lib xizmat qiladi**. U loyihaning barcha muhim jihatlari bo'yicha xolis va tizimli ma'lumotlarni taqdim etgani uchun, rahbariyat loyihani davom ettirish, kechiktirish, o'zgartirish yoki bekor qilish to'g'risida ongli ravishda qaror qabul qilishi mumkin. Bu esa, resurslarning noto'g'ri sarflanishining oldini oladi va tashkilotning strategik maqsadlariga erishishiga yordam beradi.

Xulosa qilib aytganda, fizibilite hisobotining maqsadi – taklif etilayotgan loyihaning umumiy hayotiyligini har tomonlama baholash, potentsial muammolar va imkoniyatlarni aniqlash hamda loyihaning kelajagi bo'yicha asoslangan qarorlar qabul qilish uchun mustahkam zamin yaratishdir. Bu hujjat loyihani rejalashtirish jarayonining ajralmas qismi bo'lib, loyihaning muvaffaqiyatli amalga oshirilishi uchun poydevor qo'yadi.

P4

**Qanday texnik ekanligini tasvirlab bering yechimlarni solishtirish mumkin.**

Dasturiy ta'minot loyihasini amalga oshirishda ko'pincha bir nechta potentsial texnik yechimlar mavjud bo'ladi. Ushbu yechimlarni samarali solishtirish va eng maqbulini tanlash loyihaning muvaffaqiyati uchun muhim ahamiyatga ega. Texnik yechimlarni solishtirish jarayoni bir necha asosiy usullar va mezonlarga tayanadi.

Birinchi navbatda, solishtirish uchun **aniq mezonlar to'plamini belgilab olish** zarur. Bu mezonlar loyihaning o'ziga xos talablari va ustuvorliklaridan kelib chiqishi kerak. Umumiy mezonlar qatoriga quyidagilar kirishi mumkin: **funksional muvofiqlik**, ya'ni yechimning belgilangan talablarni qanchalik to'liq qondira olishi; **texnik amalga oshiriluvchanlik**, ya'ni mavjud texnologiyalar, resurslar va jamoaning malakasi bilan yechimni yaratish imkoniyati; **unumdorlik**, jumladan tezlik, kengayuvchanlik va ishonchlilik kabi ko'rsatkichlar; **xarajatlar**, shu jumladan ishlab chiqish, joriy etish, qo'llab-quvvatlash va umumiy egalik qiymati (TCO); **xavfsizlik** darajasi; **qo'llab-quvvatlanuvchanlik va kengaytiriluvchanlik**, ya'ni kelajakda tizimni o'zgartirish va yangilash qanchalik oson bo'lishi; hamda **mavjud tizimlar bilan integratsiya** qilish imkoniyatlari (Dennis va Haley, 2009).

Ushbu mezonlar belgilangandan so'ng, turli **solishtirish usullaridan** foydalanish mumkin. Eng keng tarqalgan usullardan biri bu **vaznli ballash matritsasi (weighted scoring matrix)** hisoblanadi. Ushbu usulda har bir mezonga uning loyiha uchun muhimligiga qarab vazn beriladi. Keyin har bir texnik yechim har bir mezon bo'yicha baholanadi (masalan, 1 dan 5 gacha ball bilan) va olingan ballar mos vaznlarga ko'paytirilib, umumiy ball hisoblanadi. Eng yuqori umumiy ball to'plagan yechim odatda eng maqbul deb topiladi (Lejk va Deeks, 2002).

Yana bir muhim usul – bu **prototiplash yoki kontseptsiya isboti (Proof of Concept - PoC)** yaratishdir. Ayniqsa, yangi yoki murakkab texnologiyalar ko'rib chiqilayotganda, har bir yechimning kichik miqyosdagi ishchi modelini yaratish ularning texnik jihatdan hayotiyligini, potentsial muammolarini va unumdorligining ayrim jihatlarini amalda tekshirish imkonini beradi. Bu, shuningdek, foydalanuvchilarning dastlabki fikr-mulohazalarini olish uchun ham foydalidir.

**Xarajat-foyda tahlili (cost-benefit analysis)** ham texnik yechimlarni, ayniqsa ularning iqtisodiy samaradorligi nuqtai nazaridan solishtirishda muhim rol o'ynaydi. Har bir yechim bilan bog'liq bo'lgan barcha kutilayotgan xarajatlar va potentsial foydalar sinchkovlik bilan baholanadi va sof foyda yoki investitsiyalarning qaytimi (ROI) kabi ko'rsatkichlar bo'yicha taqqoslanadi.

Bundan tashqari, **ekspert baholashi va jamoaviy muhokama** ham qaror qabul qilish jarayonining ajralmas qismidir. Tajribali mutaxassislar, arxitektorlar va ishlab chiquvchilarning fikrlari, ularning texnik bilimlari va avvalgi tajribalari turli yechimlarning kuchli va zaif tomonlarini chuqurroq tushunishga yordam beradi.

Shuningdek, har bir texnik yechim bilan bog'liq bo'lgan **risklarni baholash** ham muhimdir. Ba'zi yechimlar texnologik jihatdan ilg'or bo'lishi mumkin, ammo ular bilan bog'liq risklar (masalan, yetarli malakali mutaxassislarning yo'qligi, texnologiyaning yetarlicha sinovdan o'tmaganligi) yuqori bo'lishi mumkin. Risklarni boshqarish imkoniyatlari ham solishtirishda hisobga olinishi kerak.

Nihoyat, agar tashqi komponentlar yoki tayyor dasturiy ta'minot (off-the-shelf software) ko'rib chiqilayotgan bo'lsa, **yetkazib beruvchining obro'si, qo'llab-quvvatlash xizmati sifati va litsenziyalash shartlari** kabi omillar ham solishtirish mezonlariga kiritilishi lozim.

Xulosa qilib aytganda, texnik yechimlarni solishtirish – bu ko'p qirrali jarayon bo'lib, u aniq belgilangan mezonlar, tizimli tahlil usullari va ekspert bilimlarining kombinatsiyasini talab qiladi. Maqsad – loyihaning muvaffaqiyatini ta'minlaydigan, uning talablariga eng mos keladigan, iqtisodiy jihatdan samarali va minimal riskka ega bo'lgan yechimni tanlashdir.

M2

**Texnik-iqtisodiy hisobotning tarkibiy qismlarini muhokama qiling.**

Texnik-iqtisodiy asoslash (TIA) hisoboti taklif etilayotgan loyihaning hayotiyligini har tomonlama baholash uchun tuzilgan muhim hujjat bo'lib, u odatda bir nechta standart tarkibiy qismlardan iborat bo'ladi. Ushbu qismlar birgalikda qaror qabul qiluvchilarga loyiha haqida to'liq va obyektiv ma'lumot berishga xizmat qiladi.

Hisobot odatda **kirish qismi** bilan boshlanadi. Bu yerda loyihaning umumiy tavsifi, uning maqsadlari va vazifalari, shuningdek, TIA o'tkazilishining sabablari va ko'lami bayon etiladi. Kirish qismi o'quvchiga hisobotning qolgan qismlarini tushunish uchun kontekst yaratadi.

Keyingi muhim tarkibiy qism – bu **mavjud tizimning tahlili (agar mavjud bo'lsa) yoki muammoning bayoni**dir. Bu bo'limda hozirgi vaziyat, mavjud muammolar, kamchiliklar yoki yangi imkoniyatlar batafsil yoritiladi. Agar yangi tizim mavjud tizimni almashtirish yoki takomillashtirish uchun taklif etilayotgan bo'lsa, mavjud tizimning kuchli va zaif tomonlari tahlil qilinadi. Bu taklif etilayotgan loyihaning zarurligini asoslashga yordam beradi.

Shundan so'ng, **taklif etilayotgan yechim(lar)ning tavsifi** keladi. Bu bo'limda loyihaning asosiy g'oyasi, uning qanday ishlashi, qanday funksiyalarni bajarishi va qanday natijalarga erishishi kutilayotgani batafsil bayon etiladi. Agar bir nechta muqobil yechimlar ko'rib chiqilayotgan bo'lsa, har biri alohida tavsiflanadi.

Hisobotning markaziy qismlaridan biri – bu **fizibilitet tahlilining o'zi**dir. Bu odatda bir necha kichik bo'limlarga ajratiladi, ularning eng keng tarqalganlari TELOS (Technical, Economic, Legal, Operational, Schedule) tizimi bo'yicha tahlillardir (Dennis va Haley, 2009).

**Texnik fizibilitet (Technical Feasibility):** Taklif etilayotgan tizimni yaratish uchun zarur bo'lgan texnologiyalar mavjudligi, ularning yetukligi, jamoaning texnik malakasi va mavjud infratuzilmaning mosligi baholanadi.

**Iqtisodiy fizibilitet (Economic Feasibility):** Xarajatlar va foydalar tahlili (cost-benefit analysis) o'tkaziladi. Loyihaning umumiy qiymati, kutilayotgan daromadlar, investitsiyalarning qaytimi (ROI), naqd pul oqimi va boshqa moliyaviy ko'rsatkichlar hisoblanadi (Lejk va Deeks, 2002).

**Huquqiy fizibilitet (Legal Feasibility):** Loyihaning amaldagi qonunchilikka, litsenziyalash talablariga, ma'lumotlar maxfiyligi qoidalariga va boshqa huquqiy me'yorlarga mosligi tekshiriladi.

**Operatsion fizibilitet (Operational Feasibility):** Taklif etilayotgan tizim joriy etilgandan so'ng tashkilotning kundalik faoliyatiga qanday ta'sir qilishi, foydalanuvchilar tomonidan qanchalik yaxshi qabul qilinishi va mavjud ish jarayonlariga mos kelishi baholanadi. O'qitish ehtiyojlari ham shu yerda ko'rib chiqiladi.

**Jadval fizibiliteti (Schedule Feasibility):** Loyihani belgilangan muddatlarda amalga oshirish imkoniyati baholanadi. Loyihaning asosiy bosqichlari uchun taxminiy vaqt jadvallari tuziladi va ularning realistikligi tekshiriladi.

Keyingi muhim bo'lim **risklarni baholash va ularni kamaytirish choralari**ga bag'ishlanadi. Bu yerda loyiha bilan bog'liq bo'lishi mumkin bo'lgan barcha potentsial risklar (texnik, moliyaviy, operatsion, bozor va hk.) aniqlanadi, ularning yuzaga kelish ehtimoli va ta'sir darajasi baholanadi hamda ularni oldini olish yoki oqibatlarini yumshatish bo'yicha tavsiyalar beriladi.

Hisobotda, shuningdek, **muqobil yechimlarni solishtirish (agar mavjud bo'lsa)** bo'limi ham bo'lishi mumkin. Bunda har bir muqobil variantning afzalliklari, kamchiliklari, xarajatlari va risklari yuqorida sanab o'tilgan fizibilitet mezonlari asosida taqqoslanadi.

Nihoyat, hisobot **xulosalar va tavsiyalar** bilan yakunlanadi. Bu bo'limda o'tkazilgan tahlillar asosida umumiy xulosalar chiqariladi va loyihani davom ettirish, o'zgartirish yoki undan voz kechish bo'yicha aniq tavsiyalar beriladi. Agar loyihani davom ettirish tavsiya etilsa, keyingi qadamlar ham belgilanishi mumkin.

Ba'zi hollarda, TIA hisobotiga **ilovalar** ham kiritilishi mumkin, ular batafsil hisob-kitoblar, diagrammalar, so'rovnomalar natijalari yoki boshqa qo'shimcha ma'lumotlarni o'z ichiga olishi mumkin.

Umuman olganda, texnik-iqtisodiy hisobotning tarkibiy qismlari loyihaning barcha muhim jihatlarini qamrab olishga va qaror qabul qiluvchilarga asosli tanlov qilish uchun zarur bo'lgan keng qamrovli ma'lumotlarni taqdim etishga qaratilgan.

D2

**Dasturiy ta'minot tekshiruvida turli fizibilite mezonlarining ta'sirini baholang**

Dasturiy ta'minotni tekshirish (software investigation) yoki loyihani dastlabki baholash bosqichida turli fizibilite mezonlarining ta'siri nihoyatda katta. Ushbu mezonlar – texnik, iqtisodiy, huquqiy, operatsion va jadval (TELOS) – loyihaning hayotiyligini va uni amalga oshirishga arziydiganligini aniqlash uchun asosiy yo'naltiruvchi omillardir. Ularning har biri loyihaning kelajagi haqida muhim qarorlar qabul qilishga bevosita ta'sir ko'rsatadi.

**Texnik fizibilitetning ta'siri:** Texnik fizibilitet loyihani amalga oshirish uchun zarur bo'lgan texnologiya, vositalar va tajribaning mavjudligini baholaydi. Agar dasturiy ta'minot tekshiruvi natijasida taklif etilayotgan yechim uchun zarur texnologiyalar hali yetarlicha yetuk emasligi, juda qimmatligi yoki jamoada ularni qo'llash uchun malaka yetishmasligi aniqlansa, bu loyihaning yo'nalishini keskin o'zgartirishi yoki hatto uni to'xtatishi mumkin. Masalan, agar bir kompaniya sun'iy intellektga asoslangan murakkab tahliliy tizimni yaratmoqchi bo'lsa, ammo uning IT bo'limida bunday tizimlarni ishlab chiqish va qo'llab-quvvatlash uchun tajriba bo'lmasa yoki bozorda bunday mutaxassislarni topish qiyin bo'lsa, loyihaning texnik fizibiliteti past deb baholanadi. Bu esa, loyiha ko'lamini kichraytirish, oddiyroq texnologiyalarni tanlash yoki loyihadan butunlay voz kechish qaroriga olib kelishi mumkin. Aksincha, agar barcha texnik shart-sharoitlar mavjud bo'lsa, bu loyihaga "yashil chiroq" yoqadi.

**Iqtisodiy fizibilitetning ta'siri:** Iqtisodiy fizibilitet loyihaning moliyaviy jihatdan o'zini oqlashini tahlil qiladi. Dasturiy ta'minot tekshiruvi davomida xarajatlar (ishlab chiqish, uskunalar, litsenziyalar, o'qitish, qo'llab-quvvatlash) va kutilayotgan foydalar (daromadning oshishi, xarajatlarning kamayishi, samaradorlikning ortishi) sinchkovlik bilan baholanadi. Agar xarajatlar foydadan ancha yuqori bo'lsa yoki investitsiyalarning qaytim muddati (ROI) juda uzoq bo'lsa, loyiha iqtisodiy jihatdan samarasiz deb topilishi mumkin (Lejk va Deeks, 2002). Masalan, kichik bir biznes uchun juda qimmat CRM tizimini joriy etish, agar u keltiradigan foyda xarajatlarni qoplay olmasa, iqtisodiy jihatdan oqlanmaydi. Bunday holda, arzonroq muqobil yechimlarni izlash yoki loyihani keyinga qoldirish qarori qabul qilinishi mumkin. Ijobiy iqtisodiy baho esa loyihani moliyalashtirish uchun asos bo'ladi.

**Huquqiy fizibilitetning ta'siri:** Huquqiy fizibilitet loyihaning amaldagi qonunlar, me'yoriy hujjatlar va litsenziyalash talablariga muvofiqligini tekshiradi. Agar dasturiy ta'minot tekshiruvi loyihaning ma'lumotlar maxfiyligi to'g'risidagi qonunlarga (masalan, GDPR), intellektual mulk huquqlariga yoki sohaviy standartlarga zid ekanligini aniqlasa, bu jiddiy huquqiy muammolarga va katta jarimalarga olib kelishi mumkin. Masalan, bemorlarning shaxsiy ma'lumotlarini qayta ishlaydigan tibbiy dasturiy ta'minot HIPAA (AQShda) kabi maxsus qonunlarga qat'iy rioya qilishi shart. Agar muvofiqlikni ta'minlash imkonsiz yoki haddan tashqari qimmat bo'lsa, loyiha to'xtatilishi mumkin. Huquqiy jihatdan "toza" bo'lish loyihaning barqarorligi uchun muhimdir.

**Operatsion fizibilitetning ta'siri:** Operatsion fizibilitet taklif etilayotgan tizimning tashkilotning kundalik faoliyatiga qanchalik mos kelishini va foydalanuvchilar tomonidan qanday qabul qilinishini baholaydi. Dasturiy ta'minot tekshiruvi shuni ko'rsatishi mumkinki, yangi tizim mavjud ish jarayonlarini tubdan o'zgartirishni talab qiladi yoki xodimlar uni o'zlashtirishga qarshilik ko'rsatishi mumkin. Masalan, agar kompaniya xodimlari yillar davomida ma'lum bir dasturiy ta'minotdan foydalanishga o'rganib qolgan bo'lsa, ularni butunlay yangi va murakkab interfeysli tizimga o'tkazish qiyin bo'lishi va samaradorlikning vaqtincha pasayishiga olib kelishi mumkin (Dennis va Haley, 2009). Bunday hollarda, qo'shimcha o'qitish dasturlarini ishlab chiqish, tizimni bosqichma-bosqich joriy etish yoki foydalanuvchilarga qulayroq yechim izlash kabi choralar ko'rilishi mumkin. Agar operatsion muammolar hal etib bo'lmas darajada bo'lsa, loyihadan voz kechish ham mumkin.

**Jadval fizibilitetining ta'siri:** Jadval fizibiliteti loyihani belgilangan muddatlarda amalga oshirish imkoniyatini tahlil qiladi. Dasturiy ta'minot tekshiruvi davomida loyihaning hajmi, murakkabligi va mavjud resurslar hisobga olingan holda realistik vaqt jadvali tuzishga harakat qilinadi. Agar loyihani bozorga chiqarish uchun qat'iy muddatlar bo'lsa (masalan, raqobatchilardan oldin yangi mahsulotni taqdim etish), ammo tekshiruv uni bu muddatda yakunlash imkonsizligini ko'rsatsa, loyiha strategiyasi qayta ko'rib chiqilishi kerak. Bu loyiha ko'lamini qisqartirish (MVP – minimal hayotiy mahsulot yaratish), qo'shimcha resurslar jalb qilish yoki muddatlarni uzaytirish (agar iloji bo'lsa) kabi qarorlarga olib kelishi mumkin. Agar belgilangan muddatlarga rioya qilish mutlaqo imkonsiz bo'lsa, loyiha boshlanmasligi ham mumkin.

Xulosa qilib aytganda, dasturiy ta'minot tekshiruvida har bir fizibilite mezonining (texnik, iqtisodiy, huquqiy, operatsion, jadval) ta'siri juda muhim. Ushbu mezonlar bo'yicha o'tkazilgan sinchkov tahlil loyihaning kuchli va zaif tomonlarini, potentsial xavf-xatarlarini va imkoniyatlarini ochib beradi. Natijada, manfaatdor tomonlar loyihani davom ettirish, o'zgartirish yoki undan voz kechish to'g'risida asosli va xabardor qaror qabul qila oladilar, bu esa resurslarning samarali ishlatilishini va loyihaning muvaffaqiyat ehtimolini oshiradi.

LO3: P5

**Biznesga bo'lgan ehtiyojni qondirish uchun dasturiy ta'minot tekshiruvini o'tkazing.**

Dasturiy ta'minotni ishlab chiqish jarayoni murakkab va ko'p bosqichli bo'lib, uni samarali amalga oshirish uchun turli xil yordamchi vositalardan foydalaniladi. Ushbu vositalar ishlab chiquvchilarning mehnat unumdorligini oshirish, xatoliklarni kamaytirish, jamoaviy ishlashni yaxshilash va umumiy sifatni oshirishga yordam beradi. Ularning har biri o'ziga xos maqsadga ega.

**Integrallashgan ishlab chiqish muhitlari (Integrated Development Environments - IDEs)** dasturiy ta'minotni ishlab chiqishda eng asosiy vositalardan biridir. IDElar odatda kod muharriri, kompilyator/interpretator, tuzatuvchi (debugger) va ba'zan versiyalarni boshqarish tizimi bilan integratsiya kabi bir nechta vositalarni bitta dasturiy ta'minot paketida birlashtiradi. Ularning maqsadi – dasturchiga kod yozish, uni sinovdan o'tkazish va tuzatish uchun qulay va markazlashtirilgan platforma taqdim etishdir. Misollar: Visual Studio, IntelliJ IDEA, Eclipse, VS Code. Bu vositalar kodni avtomatik to'ldirish, sintaksisni ajratib ko'rsatish va xatoliklarni tezda aniqlash kabi funksiyalar orqali ishlab chiqish jarayonini sezilarli darajada tezlashtiradi.

**Versiyalarni boshqarish tizimlari (Version Control Systems - VCS)** loyiha fayllaridagi o'zgarishlarni vaqt o'tishi bilan kuzatib borish uchun mo'ljallangan. Ularning asosiy maqsadi – bir nechta ishlab chiquvchiga bir vaqtning o'zida bitta loyiha ustida ishlash imkonini berish, o'zgarishlarni birlashtirish (merge), ziddiyatlarni (conflicts) hal qilish va kerak bo'lganda fayllarning oldingi versiyalariga qaytish imkoniyatini ta'minlashdir. Eng keng tarqalgan VCS bu Git bo'lib, GitHub, GitLab va Bitbucket kabi platformalar bilan birgalikda ishlatiladi. Bu vositalar kodning yo'qolishini oldini oladi va jamoaviy hamkorlikni samarali tashkil etishga yordam beradi.

**Loyiha boshqaruvi vositalari (Project Management Tools)** loyiha vazifalarini rejalashtirish, taqsimlash, kuzatib borish va jamoa a'zolari o'rtasidagi muloqotni boshqarish uchun ishlatiladi. Ularning maqsadi – loyihaning belgilangan muddatlarda va byudjet doirasida bajarilishini ta'minlash, resurslarni samarali taqsimlash va loyihaning borishi haqida shaffoflikni oshirishdir. Misollar: Jira, Trello, Asana, Microsoft Project. Bu vositalar vazifalarning holatini kuzatish, muddatlarni belgilash va jamoa a'zolari o'rtasida axborot almashinuvini osonlashtiradi.

**Avtomatlashtirilgan sinov vositalari (Automated Testing Tools)** dasturiy ta'minotni sinovdan o'tkazish jarayonini avtomatlashtirish uchun mo'ljallangan. Ularning maqsadi – sinovlarni tezroq, samaraliroq va kamroq xato bilan o'tkazish, shuningdek, regressiya testlarini (kodga o'zgartirish kiritilgandan keyin mavjud funksionallik buzilmaganligini tekshirish) osonlashtirishdir. Misollar: Selenium (veb-ilovalar uchun), JUnit (Java uchun), PyTest (Python uchun). Avtomatlashtirilgan testlash dasturiy ta'minot sifatini oshirishga va xatoliklarni ishlab chiqarish muhitiga chiqishidan oldin aniqlashga yordam beradi.

**Doimiy integratsiya va doimiy yetkazib berish (Continuous Integration/Continuous Delivery - CI/CD) vositalari** kod o'zgarishlarini avtomatik ravishda yig'ish (build), sinovdan o'tkazish va ishlab chiqarish muhitiga (yoki uning bir qismiga) yetkazib berish jarayonlarini avtomatlashtiradi. Ularning maqsadi – dasturiy ta'minotni tez-tez va ishonchli tarzda yangilab turish, xatoliklarni erta aniqlash va ishlab chiqish siklini qisqartirishdir. Misollar: Jenkins, GitLab CI/CD, GitHub Actions, CircleCI. CI/CD amaliyotlari Agile va DevOps yondashuvlarining muhim qismidir.

**Dizayn va prototiplash vositalari (Design and Prototyping Tools)** foydalanuvchi interfeyslari (UI) va foydalanuvchi tajribasini (UX) loyihalash, shuningdek, interaktiv prototiplarni yaratish uchun ishlatiladi. Ularning maqsadi – dizaynerlarga va ishlab chiquvchilarga dasturiy ta'minotning ko'rinishi va ishlashi haqida aniq tasavvurga ega bo'lish, foydalanuvchilarning dastlabki fikr-mulohazalarini olish va ishlab chiqishdan oldin dizayn muammolarini aniqlash imkonini berishdir. Misollar: Figma, Adobe XD, Sketch, InVision.

Bular dasturiy ta'minotni ishlab chiqishda qo'llaniladigan yordamchi vositalarning ayrim misollari xolos. Ularning har biri ishlab chiqish jarayonining ma'lum bir jihatini osonlashtirish va samaradorligini oshirishga xizmat qiladi. To'g'ri vositalarni tanlash va ulardan samarali foydalanish loyihaning muvaffaqiyatiga sezilarli hissa qo'shishi mumkin.

P6

**Dasturiy ta'minotni tekshirish va qo'llab-quvvatlovchi hujjatlarni yaratish uchun tegishli dasturiy ta'minotni tahlil qilish vositalari/texnikalaridan foydalaning.**

Dasturiy ta'minotni ishlab chiqishda ikki asosiy yondashuv mavjud: an'anaviy (ko'pincha Waterfall yoki ketma-ket modellar bilan bog'liq) va Agile (moslashuvchan) yondashuvlar. Ularning har biri o'zining afzalliklari va kamchiliklariga ega bo'lib, loyihaning o'ziga xos xususiyatlariga qarab tanlanadi.

An'anaviy Yondashuvlar (masalan, Waterfall)

Afzalliklari:

Aniq tuzilma va rejalashtirish: An'anaviy modellar loyihaning boshida batafsil rejalashtirishni va har bir bosqichning aniq belgilanishini talab qiladi. Bu katta, murakkab loyihalarda tartibni saqlashga va loyihaning borishini kuzatishni osonlashtirishga yordam beradi (Murch, 2012). Har bir bosqichning aniq natijalari (deliverables) bo'ladi.

Hujjatlashtirishga e'tibor: Ushbu yondashuvlar har bir bosqichda batafsil hujjatlashtirishni talab qiladi (talablar spetsifikatsiyasi, dizayn hujjatlari, test rejalari). Bu bilim almashinuvi, yangi jamoa a'zolarini o'qitish va tizimni uzoq muddatli qo'llab-quvvatlash uchun foydalidir (freetutes.com). Ayniqsa, qat'iy tartibga solinadigan sohalarda (masalan, aviatsiya, tibbiyot) bu muhim.

Resurslarni boshqarish osonligi (nazariy jihatdan): Loyihaning boshida barcha bosqichlar va vazifalar aniq belgilanganligi sababli, resurslarni (xodimlar, byudjet) taqsimlash va boshqarish nazariy jihatdan osonroq bo'lishi mumkin.

Talablari barqaror loyihalar uchun mos: Agar loyihaning talablari boshida to'liq aniq va o'zgarmas bo'lsa, an'anaviy yondashuv samarali bo'lishi mumkin.

Kamchiliklari:

Moslashuvchanlikning pastligi: An'anaviy modellar talablarning loyiha davomida o'zgarishiga yomon moslashadi. Agar talablar o'zgarsa, bu katta kechikishlarga va xarajatlarning oshishiga olib kelishi mumkin (IJCSI, 2010).

Ishlaydigan dasturiy ta'minotning kech paydo bo'lishi: Foydalanuvchilar yoki manfaatdor tomonlar ishlaydigan dasturiy ta'minotni faqat loyihaning eng oxirgi bosqichlarida ko'rishadi. Bu esa, agar talablar noto'g'ri tushunilgan bo'lsa, katta muammolarga olib kelishi mumkin.

Risklarni boshqarishdagi cheklovlar: Risklar asosan loyihaning boshida aniqlanadi. Loyiha davomida paydo bo'ladigan yangi risklarga tezkor javob berish qiyin.

Mijoz bilan hamkorlikning cheklanganligi: Mijoz asosan loyihaning boshida (talablarni aniqlashda) va oxirida (qabul qilish testlarida) ishtirok etadi. Jarayon davomida fikr-mulohazalarni olish imkoniyati kam.

"Katta portlash" integratsiyasi: Turli komponentlar alohida ishlab chiqilib, faqat oxirida birlashtiriladi, bu esa integratsiya muammolarini kech aniqlashga olib kelishi mumkin.

Agile Yondashuvlar (masalan, Scrum, Kanban)

Afzalliklari:

Yuqori moslashuvchanlik: Agile yondashuvlari o'zgaruvchan talablarga tezda moslasha oladi. Qisqa iteratsiyalar (sprintlar) orqali talablarni qayta ko'rib chiqish va o'zgartirishlar kiritish mumkin (Ferguson, 2014).

Mijoz bilan doimiy hamkorlik: Mijoz yoki uning vakili ishlab chiqish jarayoniga faol jalb qilinadi, bu esa yakuniy mahsulotning uning ehtiyojlariga maksimal darajada mos kelishini ta'minlaydi.

Ishlaydigan dasturiy ta'minotni tez-tez yetkazib berish: Har bir iteratsiya oxirida ishlaydigan dasturiy ta'minotning kichik bir qismi (inkrement) taqdim etiladi. Bu xatoliklarni erta aniqlashga va foydalanuvchilarning fikr-mulohazalarini tezda olishga imkon beradi.

Sifatni oshirish: Doimiy sinov va fikr-mulohazalar orqali sifat nazorati jarayonning ajralmas qismiga aylanadi. Xatolar erta aniqlanadi va tuzatiladi.

Jamoaviy ish va motivatsiyani oshirish: O'z-o'zini tashkil etuvchi jamoalar, shaffoflik va doimiy muloqot jamoa a'zolarining mas'uliyatini va motivatsiyasini oshiradi.

Risklarni samarali boshqarish: Qisqa iteratsiyalar va doimiy fikr-mulohazalar risklarni (masalan, bozor riski, texnik risklar) erta aniqlash va ularga tezkor javob berish imkonini beradi.

Kamchiliklari:

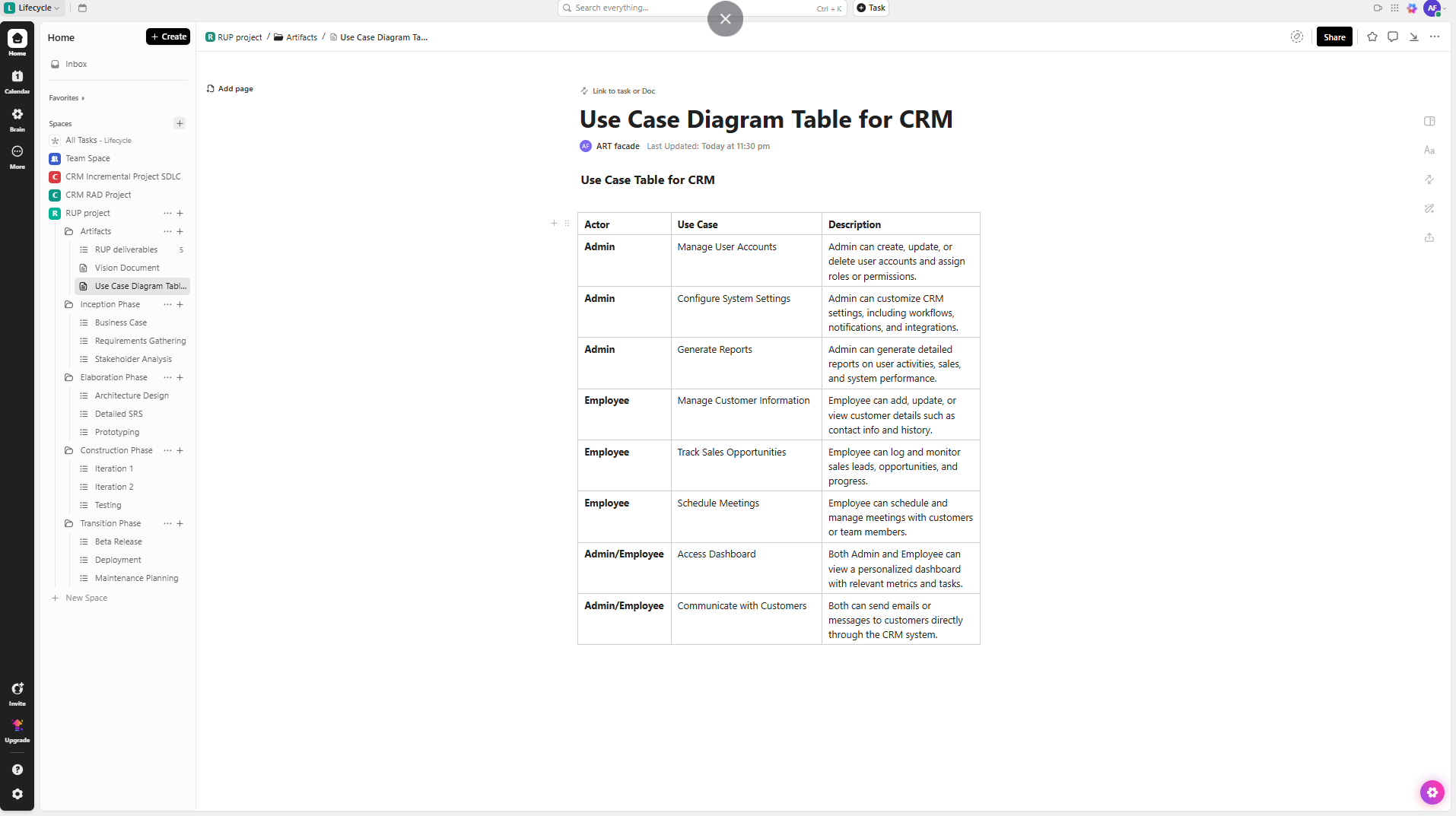
Loyiha ko'lami va muddatlarini aniq bashorat qilish qiyinligi: Talablarning o'zgaruvchanligi sababli, loyihaning yakuniy hajmini, muddatlarini va byudjetini boshida aniq belgilash qiyin bo'lishi mumkin.

Tajribali jamoa va mijoz talab etiladi: Agile muvaffaqiyati ko'p jihatdan jamoa a'zolarining tajribasi, o'zaro hamkorlik qobiliyati va mijozning jarayonga faol jalb etilishiga bog'liq.

Katta loyihalarni boshqarish murakkabligi: Katta, ko'p sonli jamoalardan iborat loyihalarda Agile yondashuvini muvofiqlashtirish va boshqarish qiyin bo'lishi mumkin (garchi Scaled Agile Framework - SAFe kabi yechimlar mavjud bo'lsa ham).

Hujjatlashtirishga kamroq e'tibor (ba'zan): Agile "ishlaydigan dasturiy ta'minot hujjatlardan muhimroq" tamoyiliga asoslanadi. Bu ba'zi hollarda, masalan, tizimni uzoq muddatli qo'llab-quvvatlash yoki bilim almashinuvi uchun yetarli hujjatlarning yo'qligiga olib kelishi mumkin.

"Scope creep" (loyiha hajmining nazoratsiz kengayishi) xavfi: Agar talablar va ustuvorliklar qat'iy boshqarilmasa, loyiha hajmi doimiy ravishda kengayib borishi mumkin.

Xulosa qilib aytganda, an'anaviy yondashuvlar talablari barqaror, yaxshi aniqlangan va hujjatlashtirish muhim bo'lgan loyihalar uchun mos kelishi mumkin. Agile yondashuvlari esa talablari tez-tez o'zgarib turadigan, innovatsion va tezkor natijalar talab qilinadigan loyihalar uchun afzalroqdir. Ko'pgina zamonaviy loyihalar ushbu ikki yondashuvning elementlarini o'zida mujassam etgan gibrid modellardan foydalanadi. Tanlov loyihaning o'ziga xos ehtiyojlari, jamoaning imkoniyatlari va tashkiliy madaniyatga bog'liq.

ClickUp dasturiy ta’minotida RUP modeli ishlab chiqildi:

Document public link: <https://doc.clickup.com/90181207647/d/h/2kzkg1jz-598/dcd69ceba2d8c10>

RUP Loyiha Link: <https://app.clickup.com/90181207647/v/s/90184741891>

M3

**Dasturiy ta'minotga bo'lgan talablar dasturiy ta'minotning ishlash muddati davomida qanday kuzatilishi mumkinligini tahlil qiling.**

Dasturiy ta’minotga bo‘lgan talablarning butun hayot aylanishi davomida uzluksiz va aniqlik bilan kuzatilishi mahsulot sifatini ta’minlashda, shuningdek, keyingi bosqichlarda yuzaga kelishi mumkin bo‘lgan nomuvofiqliklar va xatoliklarni oldini olishda muhim rol o‘ynaydi. Talablarni kuzatish (traceability) — bu dasturiy ta’minotni ishlab chiqish jarayonida aniqlangan har bir talabning keyingi dizayn, kod, test va ekspluatatsiya bosqichlarida o‘z ifodasini topishini ta’minlashdir.

Zamonaviy dasturiy ta’minot loyihalarida talablarni kuzatish uchun maxsus vositalar, masalan, IBM DOORS yoki Jira kabi talablarni boshqarish tizimlari keng qo‘llaniladi. Bu vositalar har bir talab uchun noyob identifikatorlar yaratish, ularning hayot aylanishi davomida o‘zgarishini nazorat qilish va loyiha ishtirokchilari o‘rtasida samarali kommunikatsiyani tashkil etish imkonini beradi. Talablar o‘zgarishi yoki yangilanishi natijasida ushbu o‘zgarishlarning boshqa komponentlarga ta’sirini tezda aniqlash, shuningdek, test ssenariylarini va kod qismlarini moslashtirish imkoniyati vujudga keladi.

Bundan tashqari, talablarni kuzatish tizimlari dasturiy ta’minotni sertifikatlash, auditdan o‘tkazish va ekspluatatsiya bosqichida yuzaga keladigan muammolarni aniqlashda ham yordam beradi. Masalan, foydalanuvchi tomonidan bildirilgan muammo yoki taklif qaysi dastlabki talablarga bog‘liqligini aniqlash orqali, mazkur muammoning ildiz sababini va uni bartaraf etish uchun zarur bo‘lgan o‘zgarishlarni tez va aniq belgilash mumkin. Shu tariqa, talablarning kuzatiluvchanligi dasturiy mahsulotning hayot aylanishi davomida barqaror sifat va funksionallikni ta’minlash uchun zarur zamin yaratadi.

M4

**Dasturiy ta'minot sifatini yaxshilashning ikkita yondashuvini muhokama qiling.**

Dasturiy ta’minot sifatini oshirish sohasida bir nechta yondashuvlar mavjud bo‘lib, ular orasida jarayon yo‘naltirilgan (process-oriented) va mahsulot yo‘naltirilgan (product-oriented) yondashuvlar eng ko‘p uchraydi. Ushbu yondashuvlarning har biri dasturiy mahsulot sifatini turli jihatdan ta’minlashga xizmat qiladi.

Jarayon yo‘naltirilgan yondashuvda asosiy e’tibor dasturiy ta’minotni yaratish bo‘yicha qo‘llaniladigan metodologiyalarga, standartlarga va tartiblarga qaratiladi. Bu yondashuvda CMMI (Capability Maturity Model Integration) yoki ISO 9001 kabi sifat menejmenti tizimlaridan foydalanish orqali dasturiy mahsulotni ishlab chiqish jarayonining har bir bosqichida qat’iy nazorat va uzluksiz monitoring amalga oshiriladi. Natijada, jarayonning standartlashtirilishi va optimallashtirilishi orqali xatoliklar ehtimoli kamayadi, ishonchlilik va takrorlanuvchanlik esa oshadi. Jarayon yo‘naltirilgan yondashuv, odatda, yirik va murakkab loyihalarda, hamda regulyativ talablari kuchli sohalarda samarali hisoblanadi.

Aksincha, mahsulot yo‘naltirilgan yondashuvda asosiy e’tibor yakuniy dasturiy mahsulotning sifat ko‘rsatkichlariga, ya’ni funksional to‘liqlik, ishlash tezligi, ishonchlilik, foydalanish qulayligi va boshqa texnik parametrlarga qaratiladi. Ushbu yondashuvda sifatni oshirish uchun avtomatlashtirilgan testlash, kodni qayta ko‘rib chiqish, doimiy integratsiya (CI/CD) va foydalanuvchi fikr-mulohazalarini doimiy yig‘ish kabi amaliyotlar keng qo‘llaniladi. Bu yondashuv eng avvalo mahsulotning real ishlashini va foydalanuvchi ehtiyojlariga qanchalik mos kelishini asosiy mezon sifatida oladi.

Ta’kidlash joizki, zamonaviy dasturiy ta’minot loyihalarida ushbu ikki yondashuv bir-birini to‘ldiradi. Jarayon yo‘naltirilgan yondashuv mahsulot sifatining asosiy kafolati bo‘lsa, mahsulot yo‘naltirilgan yondashuv orqali foydalanuvchilarning real ehtiyojlari va kutgan natijalari qondiriladi. Shu tariqa, dasturiy ta’minot sifatini oshirishda kompleks va muvozanatli yondashuv eng yuqori samaradorlikni ta’minlaydi.

D3

**Dasturiy ta'minot sifatini yaxshilashda uning samaradorligi nuqtai nazaridan tizim tekshiruvini o'tkazish jarayonini baholang.**

Dasturiy ta’minot sifatini oshirishda tizim tekshiruvlari, ya’ni verifikatsiya va validatsiya jarayoni, hal qiluvchi ahamiyat kasb etadi. Tizim tekshiruvining asosiy maqsadi — yaratilgan mahsulotning belgilangan talablar va foydalanuvchi ehtiyojlariga to‘liq mos kelishini ishonchli tarzda aniqlashdir. Ushbu jarayon samaradorlik nuqtai nazaridan baholanganida, bir necha muhim jihatlar ko‘zga tashlanadi.

Avvalo, tizim tekshiruvining to‘g‘ri tashkil etilishi dasturiy mahsulotni ishlab chiqish jarayonida xatoliklarni erta aniqlashga, ularni tez va nisbatan kam xarajat evaziga bartaraf etishga imkon yaratadi. Natijada, dasturiy ta’minotni ishlab chiqish va uni ekspluatatsiya qilish xarajatlari sezilarli darajada kamayadi. Yaxshi amalga oshirilgan tizim tekshiruvlari mahsulot sifatining oshishiga, foydalanuvchilar ishonchi va qoniqishining ortishiga xizmat qiladi, chunki yakuniy mahsulotda jiddiy xatoliklar yoki nomuvofiqliklar deyarli uchramaydi.

Biroq, tizim tekshiruvining samaradorligi faqatgina testlarni avtomatlashtirish yoki keng qamrovli test ssenariylarini tuzish bilangina emas, balki test jarayonining rejalashtirilishi, test qamrovi va foydalanuvchi ishtirokining ta’minlanishi bilan ham bevosita bog‘liq. Samarali tekshiruvlar doimiy feedback va iterativ testlash orqali mahsulotni ishlab chiqish tsiklini qisqartiradi, takroriy xatolarning oldini oladi hamda tizimning real ekspluatatsiya sharoitlariga mos kelishini ta’minlaydi.

Shuni ham ta’kidlash joizki, tizim tekshiruvlari samaradorligini oshirish uchun zamonaviy test avtomatlashtirish vositalari va CI/CD (Continuous Integration/Continuous Delivery) amaliyotlaridan foydalanish keng qo‘llanmoqda. Bu yondashuvlar kodda yuzaga keladigan o‘zgarishlarni avtomatik ravishda sinovdan o‘tkazishni ta’minlab, inson omilidan kelib chiqadigan xatoliklarni kamaytiradi va mahsulotni ishlab chiqarishga tezroq chiqarish imkonini beradi. Natijada, tizim tekshiruvining samaradorligi nafaqat sifatni oshirish, balki biznes uchun strategik ustunliklarni ham ta’minlaydi.

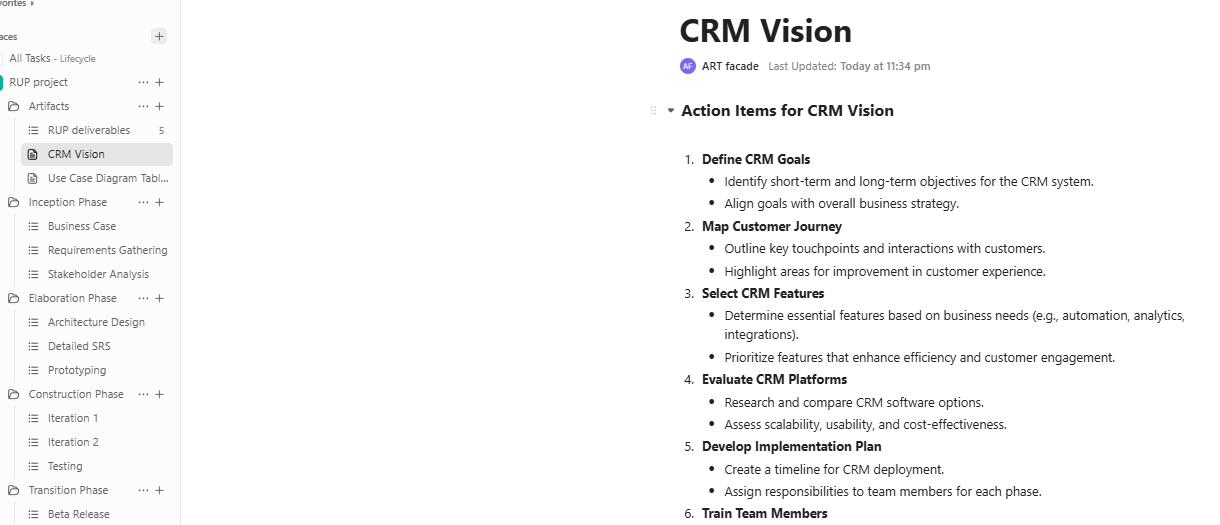
LO4: P7

**Dasturiy ta'minotning xatti-harakatlarini loyihalash usullarining mosligini misollar yordamida muhokama qiling.**

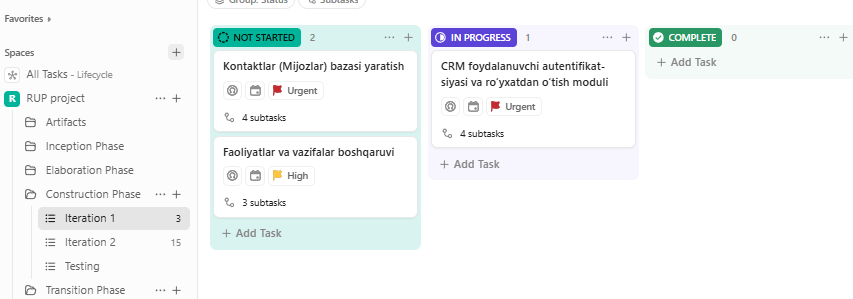
Dasturiy ta’minotning xatti-harakatlarini loyihalashda turli usullarning mosligi, ishlab chiqilayotgan tizimning murakkabligi, foydalanuvchi ehtiyojlari va texnik talablarga bevosita bog‘liq holda namoyon bo‘ladi. Bu jarayonda mos dizayn usulini tanlash dasturiy mahsulotning keyingi ishlash jarayoniga, uning ishonchliligi, tushunarliligi va rivojlantirish imkoniyatlariga sezilarli ta’sir ko‘rsatadi. Quyida Clickupda RUP modelinign loyihalash misoli keltiriladi:  
A screenshot of a computer

AI-generated content may be incorrect.

Dastlab RUP asosiy Deliverable larini aniqlashtirib olindi va so’ngra Use Case Diagram, Vision Document aniqlashtirildi:



RUP modelining asosiy 4 fazasi Inception, Elaboration, Construction va Transition fazalari aniqlashtirildi. Har bir faza uchun ‘task’lar belgilandi. CRM asosiy boshlang’ich 2 ta iterationda olib borildi. Contruction fazasida dasturni ishlab chiqish boshqichlarining 2 ta asosiy Iterationlari qurildi:



Bu rasimda Itaration 1 keltirilgan. Rejadagi Iteration 2:  
A white background with black text

AI-generated content may be incorrect.

Shuningdek birinchi Iterationning sanalari belgilab ketildi:

A screenshot of a computer

AI-generated content may be incorrect.

Masalan, xatti-harakatlarni loyihalashda eng ko‘p uchraydigan usullardan biri sifatida Unified Modeling Language (UML) asosidagi ketma-ketlik (sequence) diagrammalari tilga olinadi. Ushbu diagrammalar tizim ichidagi obyektlar o‘rtasidagi o‘zaro aloqani va vaqt bo‘yicha xabarlar almashuvini vizual tarzda ifodalash imkonini beradi. Bu, ayniqsa, murakkab biznes jarayonlarini, masalan, onlayn buyurtma berish yoki bank tranzaksiyalarini modellashtirishda juda qulaydir. Ketma-ketlik diagrammasi orqali jarayondagi har bir bosqichning aniq tartibi va harakatlar ketma-ketligi ishlab chiqilishi mumkin, bu esa dasturchilarga va manfaatdor tomonlarga tizimning ishlash logikasini to‘liq va ravshan ko‘rsatib beradi (Fowler, 2003).

Boshqa bir misolda, holat diagrammalari (state diagrams) tizimning turli holatlari va holatlar o‘rtasidagi o‘tishlarni aniqlash uchun qo‘llaniladi. Bu usul, odatda, avtomatlashtirilgan nazorat tizimlari yoki foydalanuvchi interfeysi komponentlari kabi holatga bog‘liq xatti-harakatlarni loyihalashda samarali hisoblanadi. Masalan, bankomat dasturida kartaning kiritilishi, PIN-kodni tasdiqlash, pul yechish va transaktsiyani yakunlash kabi turli holatlar va ular o‘rtasidagi o‘tishlar aynan holat diagrammasi yordamida aniq va izchil tarzda loyihalanadi (Sommerville, 2011).

Yana bir keng tarqalgan yondashuv — faoliyat diagrammalari (activity diagrams) bo‘lib, ular jarayon ichida bir nechta parallel va ketma-ket ish faoliyatini ifodalash imkonini beradi. Bu usul, ayniqsa, murakkab biznes jarayonlarida yoki ko‘p foydalanuvchi ishtirok etuvchi tizimlarda harakatlar oqimini modellashtirish va optimallashtirish uchun qulaylik yaratadi. Masalan, elektron tijorat platformasida foydalanuvchi buyurtma berish, to‘lovni amalga oshirish va yetkazib berishni kuzatish kabi faoliyatlarning har biri va ular o‘rtasidagi bog‘liqlik activity diagramma yordamida aniq ko‘rsatiladi (Dennis va boshqalar, 2015).

Dasturiy ta’minotning xatti-harakatlarini loyihalashda mos usulni tanlash, avvalo, tizimning o‘ziga xos xususiyatlari, murakkablik darajasi va foydalanuvchi ehtiyojlariga tayanadi. Har bir metodning o‘ziga xos afzallik va chegaralari mavjud: masalan, ketma-ketlik diagrammalari interaktiv jarayonlar uchun, holat diagrammalari esa holatga bog‘liq tizimlar uchun, faoliyat diagrammalari esa murakkab jarayonlarni soddalashtirish uchun eng mos keladi. Shu sababli, har bir loyiha uchun eng mos yondashuvni tanlash, natijada yaratiladigan dasturiy mahsulotning sifatini va rivojlantirish samaradorligini belgilovchi asosiy omil sifatida qaraladi.

M5

Dasturiy ta’minotning xatti-harakatlarini modellashtirish, tahlil qilish va hujjatlashtirish, SDLCning (Software Development Life Cycle) barcha zamonaviy modellarida muhim ahamiyatga ega (Dennis va Haley, 2009; Ferguson, 2014). Xatti-harakatlarni ifodalash vositalari va usullari yordamida tizimning dinamik jihatlari, foydalanuvchi va dastur o‘rtasidagi o‘zaro ta’sirlar, holatlar va tranzitsiyalar aniqlik bilan tasvirlanadi. Bu, o‘z navbatida, tizimda yuzaga keladigan muammolarni erta aniqlash, talablarga mos keluvchi yechimlarni ishlab chiqish va dastur sifatini oshirish imkonini beradi.

**Xatti-harakatlarni modellashtirish vositalari va usullari**

**Ketma-ketlik diagrammalari (Sequence Diagrams)**

Unified Modeling Language (UML) doirasidagi ketma-ketlik diagrammalari dasturiy tizimdagi obyektlar yoki komponentlar o‘rtasida vaqt bo‘yicha xabarlar almashuvini vizual tarzda ifodalaydi. Bu diagrammalar, ayniqsa, foydalanuvchi so‘rovi, server javobi va oraliq jarayonlarning aniq tartibini ko‘rsatishda muhimdir. Masalan, onlayn bank xizmatida foydalanuvchi login qilishdan boshlab balansni so‘rashgacha bo‘lgan jarayonlar har bir obyekt ishtirokida va harakat ketma-ketligida ko‘rsatiladi. Ushbu yondashuv tizimning interaktiv va asinxron xatti-harakatlarini tahlil qilishda samarali hisoblanadi (Dennis va Haley, 2009).

**Holat diagrammalari (State Diagrams)**

Holat diagrammalari tizimning turli holatlari, ushbu holatlar o‘rtasidagi o‘tishlar va har bir o‘tishni faollashtiruvchi hodisalarni aniqlaydi. Cheklangan holat mashinasi yoki boshqaruv tizimlari asosida qurilgan dasturlar uchun holat diagrammasi eng samarali vositalardan hisoblanadi. Masalan, bankomat dasturida "Kartani kiritish", "PINni kiritish", "Pul yechish", "Chiqish" kabi holatlar va ular o‘rtasidagi o‘tishlar aniq ko‘rsatiladi (Lejk va Deeks, 2002).

**Faoliyat diagrammalari (Activity Diagrams)**

Faoliyat diagrammalari jarayon ichida bir nechta parallel yoki ketma-ket ish faoliyatini, shartli o‘tishlarni va tarmoqlanishlarni aniqlaydi. Ular murakkab biznes jarayonlari yoki ko‘p foydalanuvchi ishtirokidagi tizimlarni modellashtirish uchun qulaydir. Masalan, onlayn tijorat platformasida buyurtma berish, to‘lov amalga oshirish va yetkazib berish kabi faoliyatlarning har biri va ularning o‘zaro bog‘liqligi activity diagramda aks ettiriladi (Dennis va Haley, 2009).

**Cheklangan holat mashinasi (Finite State Machine, FSM)**

FSM asosan deterministik va oddiy xatti-harakatlarni ifodalash uchun ishlatiladi. Tizim holatlarining aniq chegaralangan to‘plami mavjud bo‘lib, har bir holat uchun faqatgina ruxsat etilgan o‘tishlar va ularni faollashtiruvchi hodisalar belgilanadi. Bu usul, masalan, turniket yoki signalizatsiya tizimi kabi qurilmalarda keng qo‘llaniladi (Sakarovitch, 2009).

**Kengaytirilgan holat mashinasi (Extended FSM, EFSM)**

EFSM oddiy FSM imkoniyatlarini kengaytirib, holatlarga bog‘liq bo‘lgan ma’lumotlarni, murakkab shart va harakatlarni ham modellashtirishga imkon beradi. Bu, ayniqsa, real hayotdagi murakkab tizimlar uchun, masalan, bankomat yoki boshqaruv tizimlarida zarurdir (Alur va Dill, 1994).

**Prototiplash va simulyatsiya vositalari**

Interaktiv prototiplash vositalari (masalan, Balsamiq, Figma) va simulyatsiya muhitlari dasturiy ta’minot xatti-harakatlarini foydalanuvchi ishtirokida tezkor sinovdan o‘tkazish, muammolarni erta aniqlash va foydalanuvchi ehtiyojlariga moslashtirish imkonini beradi (Ferguson, 2014).

**Tahlil va xulosa**

Dasturiy ta’minot xatti-harakatlarini modellashtirish vositalari va usullari tizim murakkabligi, talablari va foydalanish sohasiga qarab tanlanishi zarur. To‘g‘ri tanlangan yondashuv dasturiy mahsulot sifatini, barqarorligini va kengaytiriluvchanligini oshiradi. SDLCning barcha asosiy modellarida (Sharshara, Prototiplash, Spiral, Agile) ushbu modellashtirish yondashuvlari samarali ishlatiladi va har bir modelning o‘ziga xos xususiyatlariga moslashtiriladi (Murch, 2012; Dennis va Haley, 2009).

M6

Cheklangan holat mashinasi (FSM) va kengaytirilgan FSM (EFSM) dasturiy ta’minot xatti-harakatlarini matematik va vizual asosda modellashtirish uchun mo‘ljallangan formal modellardir. Ular tizimning har bir holati, holatlar o‘rtasidagi o‘tishlar va o‘tishlarni faollashtiruvchi hodisalarni aniq belgilaydi (Sakarovitch, 2009). Ushbu modellar SDLCning barcha bosqichlarida, ayniqsa, talablar aniqligi va dizayn mustahkamligi muhim bo‘lgan tizimlarda keng qo‘llaniladi.

**FSM va EFSM o‘rtasidagi asosiy farqlar**

FSM o‘z tabiatiga ko‘ra oddiy va aniq chegaralangan holatlarga ega. Har bir o‘tish faqat tashqi kiruvchi signal yoki hodisaga bog‘liq bo‘lib, tizim holatlari soni va o‘tishlar to‘plami qat’iy belgilanadi. Masalan, turniket dasturida faqat “qulfli” va “ochiq” holatlari mavjud bo‘lib, har bir holat uchun ruxsat etilgan signal (masalan, chipta qo‘yish, o‘tish) bo‘yicha o‘tish amalga oshiriladi.

EFSM esa, FSMning imkoniyatlarini sezilarli kengaytiradi. EFSMda holat o‘zgarishi nafaqat tashqi hodisa, balki tizimning ichki ma’lumotlari, shartli tekshiruvlar va harakatlar orqali boshqariladi. Bu modelda o‘tishlar o‘zgaruvchilar qiymati, oldingi harakat natijalari yoki murakkab shartli ifodalar asosida amalga oshiriladi. Masalan, bankomat dasturida foydalanuvchi tomonidan kiritilgan PIN kod to‘g‘ri yoki noto‘g‘ri ekanligini tekshirish, noto‘g‘ri urinishlar sonini hisoblash va kartani bloklash jarayonlari EFSM orqali aniq modellashtiriladi (Alur va Dill, 1994).

Ushbu modelda faqat ikki holat va ikkita asosiy o‘tish mavjud. Har bir o‘tish faqat bir kirish signaliga bog‘liq va holatlar soni qat’iy chegaralangan.

Bu misolda tizim holati nafaqat tashqi signal (masalan, kartani kiritish, pin kiritish) balki ichki o‘zgaruvchilar (urinishlar soni, pin kodi) va shartli tekshiruvlar asosida ham boshqariladi. Natijada, tizim xatti-harakatlari murakkabligi va moslashuvchanligi yuqori darajaga chiqadi.

FSM va EFSM modellarining har biri dasturiy ta’minot xatti-harakatlarini modellashtirishda muhim ahamiyatga ega. FSM oddiy va deterministik tizimlar uchun yetarli bo‘lsa, EFSM murakkab, ko‘p parametrli va shartli boshqaruvni talab qiladigan sohalarda yuqori samaradorlikka ega. SDLCning barcha modellarida, ayniqsa, risk va murakkablik darajasi yuqori bo‘lgan loyihalarda, ushbu yondashuvlar tizimni ishonchli va barqaror ishlab chiqish uchun asosiy vosita hisoblanadi (Dennis va Haley, 2009; Murch, 2012).

D4

**Ma'lumotlarga asoslangan dasturiy ta'minot dasturiy ta'minotning ishonchliligi va samaradorligini qanday oshirishi haqidagi dalillarni taqdim eting.**

Dasturiy ta'minotni ishlab chiqish hayotiy siklining (SDLC) har bir bosqichida qo'llaniladigan vositalar va usullarning samaradorligi loyihaning yakuniy muvaffaqiyatiga bevosita ta'sir ko'rsatadi. Samaradorlikni baholashda ularning nafaqat belgilangan vazifalarni bajarish qobiliyati, balki loyiha maqsadlariga erishishga, resurslarni tejashga va sifatni oshirishga qo'shgan hissasi ham inobatga olinishi lozim.

**Talablarni tahlil qilish bosqichida** qo'llaniladigan intervyular, so'rovnomalar va JAD (Joint Application Development) kabi usullarning samaradorligi, birinchi navbatda, manfaatdor tomonlarning ehtiyojlarini qanchalik to'liq va aniq qamrab olish bilan o'lchanadi. Agar bu usullar to'g'ri qo'llanilsa, ular noto'g'ri yoki yetishmayotgan talablar tufayli yuzaga keladigan keyingi bosqichlardagi qimmatli o'zgartirishlarning oldini oladi (Sommerville, 2011). Masalan, yaxshi o'tkazilgan JAD sessiyasi turli manfaatdor tomonlar o'rtasida umumiy tushunchani shakllantirib, ziddiyatlarni erta hal qilish orqali samaradorlikni oshiradi. Talablarni boshqarish vositalarining (masalan, Jira, IBM DOORS) samaradorligi esa talablarning kuzatilishini (traceability), o'zgarishlarni boshqarishni va ular bo'yicha kommunikatsiyani osonlashtirish qobiliyati bilan belgilanadi. Bu vositalar talablarning yo'qolishi yoki noto'g'ri talqin qilinishi xavfini kamaytirib, loyiha samaradorligini oshiradi.

**Dizayn bosqichida** UML diagrammalari kabi modellashtirish vositalarining samaradorligi ularning dizayn g'oyalarini aniq, tushunarli va bir ma'noli tarzda ifodalay olishida namoyon bo'ladi. Yaxshi dizayn vositasi murakkab tizimlarni soddalashtirishga, jamoa a'zolari o'rtasida umumiy tasavvurni shakllantirishga va potensial dizayn kamchiliklarini erta aniqlashga yordam beradi (Fowler, 2003). Arxitektura va dizayn naqshlarining qo'llanilishi esa sinovdan o'tgan yechimlardan foydalanish orqali ishlab chiqish vaqtini qisqartiradi va tizimning sifatini (masalan, kengayuvchanlik, qo'llab-quvvatlanuvchanlik) oshiradi, bu esa ularning yuqori samaradorligidan dalolat beradi. UI prototiplash vositalarining samaradorligi esa foydalanuvchi fikr-mulohazalarini tez va arzon olish imkonini berib, yakuniy mahsulotning foydalanuvchilar tomonidan yaxshi qabul qilinishini ta'minlash bilan o'lchanadi.

**Amalga oshirish (kodlash) bosqichida** zamonaviy Integrallashgan Ishlab Chiqish Muhitlari (IDEs) kod yozish, tuzatish va refaktoring jarayonlarini sezilarli darajada tezlashtirib, dasturchilarning unumdorligini oshiradi. Ularning avtomatik to'ldirish, sintaksisni tekshirish kabi funksiyalari xatoliklarni kamaytiradi. Versiyalarni boshqarish tizimlari (masalan, Git) jamoaviy ishlashni samarali tashkil etish, kodlar tarixini saqlash va ziddiyatlarni boshqarish orqali loyiha samaradorligiga katta hissa qo'shadi. Kodlash standartlari va kodni qayta ko'rib chiqish kabi usullar esa kod sifatini, o'qilishini va qo'llab-quvvatlanuvchanligini oshirib, uzoq muddatli samaradorlikni ta'minlaydi.

**Sinov bosqichida** avtomatlashtirilgan sinov vositalarining (masalan, Selenium, JUnit) samaradorligi juda yuqori, chunki ular ko'p marta takrorlanadigan testlarni tez va aniq bajarishga, regressiya testlarini osonlashtirishga va xatoliklarni erta aniqlashga imkon beradi. Bu esa, o'z navbatida, ishlab chiqarish muhitiga xatolar bilan chiqish xavfini kamaytiradi va tuzatish xarajatlarini tejaydi (Whittaker, 2000). Xatolarni kuzatish tizimlari (bug trackers) esa aniqlangan xatolarni tizimli ravishda boshqarish, ularning holatini kuzatib borish va tuzatilishini nazorat qilish orqali sinov jarayonining samaradorligini oshiradi.

**Joriy etish va qo'llab-quvvatlash bosqichlarida** CI/CD (Continuous Integration/Continuous Delivery) vositalarining samaradorligi dasturiy ta'minotni tez-tez, ishonchli va minimal inson aralashuvi bilan yetkazib berish qobiliyatida namoyon bo'ladi. Bu esa bozor o'zgarishlariga tezkor javob berish va raqobatbardoshlikni saqlab qolish imkonini beradi. Monitoring vositalari tizimning ishlashini real vaqt rejimida kuzatib borish, muammolarni proaktiv tarzda aniqlash va ularning biznesga ta'sirini kamaytirish orqali operatsion samaradorlikni oshiradi.

Umuman olganda, SDLCning har bir bosqichida qo'llaniladigan vosita va usullarning samaradorligi ularning nafaqat texnik imkoniyatlariga, balki loyihaning o'ziga xos ehtiyojlariga, jamoaning malakasiga va tashkiliy madaniyatga qanchalik mos kelishiga ham bog'liq. To'g'ri tanlangan va samarali integratsiya qilingan vositalar to'plami loyiha muddatlarini qisqartirish, xarajatlarni kamaytirish, sifatni oshirish va pirovardida biznes maqsadlariga erishish uchun hal qiluvchi ahamiyatga ega. Aksincha, noto'g'ri vositalarni tanlash yoki ulardan samarasiz foydalanish loyihaning muvaffaqiyatsizligiga olib kelishi mumkin

Xulosa

Ushbu keng qamrovli muhokama natijasida dasturiy ta'minotni ishlab chiqish jarayonining ko'p bosqichli va kompleks tabiati yana bir bor oydinlashdi. Loyihaning boshlang'ich g'oyasidan tortib, yakuniy mahsulotni yetkazib berishgacha bo'lgan yo'lda texnik-iqtisodiy asoslashning (TIA) hal qiluvchi roli, uning tarkibiy qismlari va turli fizibilite mezonlarining loyiha taqdiriga ta'siri ko'rib chiqildi. An'anaviy va Agile kabi muqobil ishlab chiqish metodologiyalarining o'ziga xos jihatlari, ularning afzalliklari va kamchiliklari hamda loyiha xususiyatlariga qarab tanlanishining ahamiyati ta'kidlandi. Shuningdek, prototiplashning turli turlari va Rapid Application Development (RAD) kabi tezkor yondashuvlarning an'anaviy SDLCga nisbatan ustunliklari baholandi.

Bundan tashqari, dasturiy ta'minotni ishlab chiqish hayotiy siklining har bir bosqichida – talablarni tahlil qilish, dizayn, amalga oshirish, sinovdan o'tkazish va joriy etish – qo'llaniladigan maxsus vositalar va usullarning maqsadi hamda samaradorligi tahlil qilindi. Ushbu vosita va usullarning to'g'ri tanlanishi va ulardan oqilona foydalanish loyiha muddatlarini qisqartirish, xarajatlarni optimallashtirish, sifatni oshirish va pirovardida foydalanuvchilar ehtiyojlarini qondiradigan mahsulot yaratish uchun zamin yaratishi qayd etildi.

Xulosa qilish mumkinki, muvaffaqiyatli dasturiy ta'minotni ishlab chiqish nafaqat texnik mahoratni, balki strategik rejalashtirishni, moslashuvchanlikni, jamoaviy hamkorlikni va mavjud vosita hamda usullardan unumli foydalanishni talab etadi. Dasturiy ta'minot sohasi doimiy rivojlanishda ekanligini inobatga olgan holda, yangi texnologiyalar, metodologiyalar va vositalarni o'rganish hamda ularni amaliyotga tatbiq etish jarayoni uzluksiz davom etishi lozim. Bu esa, o'z navbatida, kelajakdagi dasturiy ta'minot loyihalarining yanada samarali va sifatli bo'lishiga xizmat qiladi.

ADABIYOTLAR RO'YXATI

1. Bass, L., Clements, P. and Kazman, R. (2012) *Software Architecture in Practice*. 3rd edn. Upper Saddle River, NJ: Addison-Wesley Professional.
2. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2001) *Manifesto for Agile Software Development*. Available at: [http://agilemanifesto.org](http://agilemanifesto.org/) (Accessed: 31 May 2025).
3. Dennis, A., Wixom, B.H. and Haley, D.T. (2009) *Systems Analysis and Design*. 4th edn. Hoboken, NJ: John Wiley & Sons. (Eslatma: Suhbatda 2009 yil keltirilgan, bu odatda ushbu kitobning 4-nashriga to'g'ri keladi).
4. Dennis, A., Wixom, B.H. and Tegarden, D. (2015) *Systems Analysis and Design: An Object-Oriented Approach with UML*. 5th edn. Hoboken, NJ: Wiley.
5. Fowler, M. (2003) *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd edn. Boston: Addison-Wesley Professional.
6. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.
7. Highsmith, J. (2009) *Agile Project Management: Creating Innovative Products*. 2nd edn. Boston, MA: Addison-Wesley Professional.
8. Larman, C. (2004) *Agile and Iterative Development: A Manager's Guide*. Boston, MA: Addison-Wesley Professional.
9. Lejk, M. and Deeks, D. (2002) *An Introduction to System Analysis Techniques*. 2nd edn. Harlow: Prentice Hall.
10. Martin, J. (1991) *Rapid Application Development*. New York: Macmillan Publishing Company.
11. McConnell, S. (1996) *Rapid Development: Taming Wild Software Schedules*. Redmond, WA: Microsoft Press.
12. Myers, G.J., Sandler, C. and Badgett, T. (2011) *The Art of Software Testing*. 3rd edn. Hoboken, NJ: John Wiley & Sons.
13. Pressman, R.S. and Maxim, B.R. (2020) *Software Engineering: A Practitioner's Approach*. 9th edn. New York: McGraw-Hill Education.
14. Royce, W.W. (1970) 'Managing the Development of Large Software Systems', *Proceedings of IEEE WESCON*, August, pp. 1-9.
15. Schwaber, K. and Sutherland, J. (2020) *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. Available at: [https://scrumguides.org](https://scrumguides.org/) (Accessed: 31 May 2025).
16. Sommerville, I. (2011) *Software Engineering*. 9th edn. Boston: Pearson Education. (Eslatma: Suhbatda 2011 yil keltirilgan, bu odatda 9-nashrga to'g'ri keladi. Keyingi nashrlar ham mavjud).
17. Whittaker, J.A. (2000) 'What is software testing? And why is it so hard?', *IEEE Software*, 17(1), pp. 70-79.
18. Wiegers, K. and Beatty, J. (2013) *Software Requirements*. 3rd edn. Redmond, WA: Microsoft Press.