ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

УПРАВЛІННЯ ПРОЄКТАМИ ТА ПРОГРАМАМИ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до самостійного вивчення дисципліни

здобувачами за освітньо-професійними програмами
другого (магістерського) рівня:

«Комп'ютерні науки та цифровий інтелект»
спеціальність 122 Комп'ютерні науки

«Комп'ютерне конструювання мехатронних систем»
спеціальність 133 Галузеве машинобудування

Рекомендовано Науково-методичною радою
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 1 від «08» вересня 2023 р.)
Обов'язково до розміщення в репозиторії

Запоріжжя 2023

metinvest polytechnic

Управління проєктами та програмами: методичні рекомендації до самостійного вивчення дисципліни для здобувачів за освітньо-професійними програмами другого (магістерського) рівня «Комп'ютерні науки та цифровий інтелект» (спеціальність 122 Комп'ютерні науки) та «Комп'ютерне конструювання мехатронних систем» (спеціальність 133 Галузеве машинобудування)/ Уклад. Шевченко Н.Ю. Запоріжжя, ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2023. 30 с.

Методичні рекомендації включають завдання для самостійної роботи студентів, стислі теоретичні відомості за темами завдань та перелік посилань на корисні джерела інформації.

Рекомендовано для магістрів спеціальностей 122 Комп'ютерні науки та 133 Галузеве машинобудування всіх форм навчання.

Затверджено на засіданні кафедри ЦТПАР
Протокол № 1 від «05» вересня 2023 р.

Узгоджено:
Секретар Редакційної ради


_____ Малій Х. В.
«06» вересня 2023 р.

Зміст

1. ОСНОВНІ ПОЛОЖЕННЯ

Управління проєктами та програмами в сфері інформаційних технологій допомагає ефективно впорядковувати та керувати процесами розробки програмного забезпечення, впровадження нових технологій, створення IT-інфраструктури тощо. Оскільки ця сфера стикається зі складністю технічних завдань, змінами вимог клієнтів і швидкими змінами технологій, професійні уміння управління проєктами стають невід'ємною частиною успішного розвитку IT-проєктів та бізнесу загалом.

IT-проєкти часто мають складну структуру, велику команду фахівців і вимагають синхронізації багатьох аспектів, таких як ресурси, терміни, бюджет, ризики та інші. Відповідне управління може допомогти уникнути затримок, перевищення бюджету та невдач в проєкті.

Ця дисципліна надає студентам розуміння ключових аспектів ефективного управління проєктами та програмами в сфері інформаційних технологій. Вона охоплює такі питання, як методи, інструменти та підходи до планування, виконання та контролю IT-проєктів для досягнення успіху і високої якості.

Студенти отримують інформацію для ознайомлення з поняттями проєктів та програм, для розуміння ролі управління проєктами в сфері IT, визначення фаз та життєвого циклу проєкта, для виконання планування проєкту (визначати мету та обсяг проєкта, розробляти робочий план, графік та розподіляти ресурси), оцінювання ризиків та формування стратегій їх управління, ведення проєктної документації.

Для формування перелічених навичок, а також навичок управління ресурсами, моніторингу виконання та контролю проєкту, комунікації та співпраці студентам рекомендується виконати додаткові практичні завдання в межах самостійного вивчення окремих тем дисципліни.

2. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

2.1. Засади управління на стадіях планування та реалізації проєктів/програм

Практичне завдання № 1. Життєвий цикл продукту.

Мета: набути практичних навичок визначення найбільш доцільної моделі життєвого циклу проєкту в залежності від специфіки продукту проєкту.

Завдання: поміркуйте і запропонуйте яку модель, методологію або фреймворк ви б використали для розробки ПЗ у наступних проєктах, та обґрунтуйте чому:

1) програмне забезпечення для міністерства оборони України;

2) будівництво метрополітену;

3) створення сайту для міського водоканалу у Вашому місті з великою кількістю користувачів;

4) керування процесом виробництва миючих засобів;

5) програмне забезпечення для світової електронної системи.

Практичне завдання № 2. Загальний огляд технологій: Backend / Frontend, API, mobile, DB, Git, CI/CD, deployment

Мета: набути знань щодо базових технологій розробки програмного забезпечення (Backend / Frontend, API, mobile, DB, Git, CI/CD/deployment) та отримати практичні навички роботи з сервісами GitHub.

Завдання: створити профіль на GitHub та опублікувати сайт користувача на username.github.io (див. Алгоритм створення сайту у п. 3.2).

Практичне завдання № 3. Командоутворення при реалізації IT-проєкту

Мета: набути практичних навичок формування команди IT-проєкту (у відповідності до теми дипломного проєкту).

Завдання:

1) визначити методологію розробки програмного забезпечення, яке передбачається розробити в межах дипломного проєкту, обґрунтувати вибір;
2) визначити структуру IT-команди за ролями, стисло описати функціонал кожної ролі та обґрунтувати її необхідність в команді;
3) визначте власний (фактичний) стиль управління.

2.2. Засади управління на етапах реалізації та закриття проєктів/програм

Практичне завдання № 1. Експертиза, моніторинг та контроль робіт.

Мета: навчитися формувати (розробляти) звіт за результатами реалізації проєкту.

Завдання: розробити Project Status Reports, використовуючи шаблон *.xlsx

Практичне завдання № 2. Впровадження змін та оцінка рівня задоволеності команди та стейкхолдерів.

Мета: навчитися планувати та управляти процесом управління змінами.

Завдання: розробити Change Management Plan, використовуючи шаблон *.xlsx

Всі шаблони для формування окремих проєктних документів надані в курсі на платформі Moodle.

3. СТИСЛІ ТЕОРЕТИЧНІ ВІДОМОСТІ

3.1. Життєвий цикл продукту[1]

Життєвий цикл ПЗ, **SDLC (Software development lifecycle)** – стадії, що проходить програмний продукт від появи ідеї до її реалізації в коді, імплементації у бізнес і подальшої підтримки.

Стандартні етапи ЖЦ (рис. 3.1):

1. Ініціалізація.
2. Аналіз вимог.
3. Проєктування.
4. Програмування.
5. Тестування і налагодження.
6. Експлуатація, супровід і підтримка.

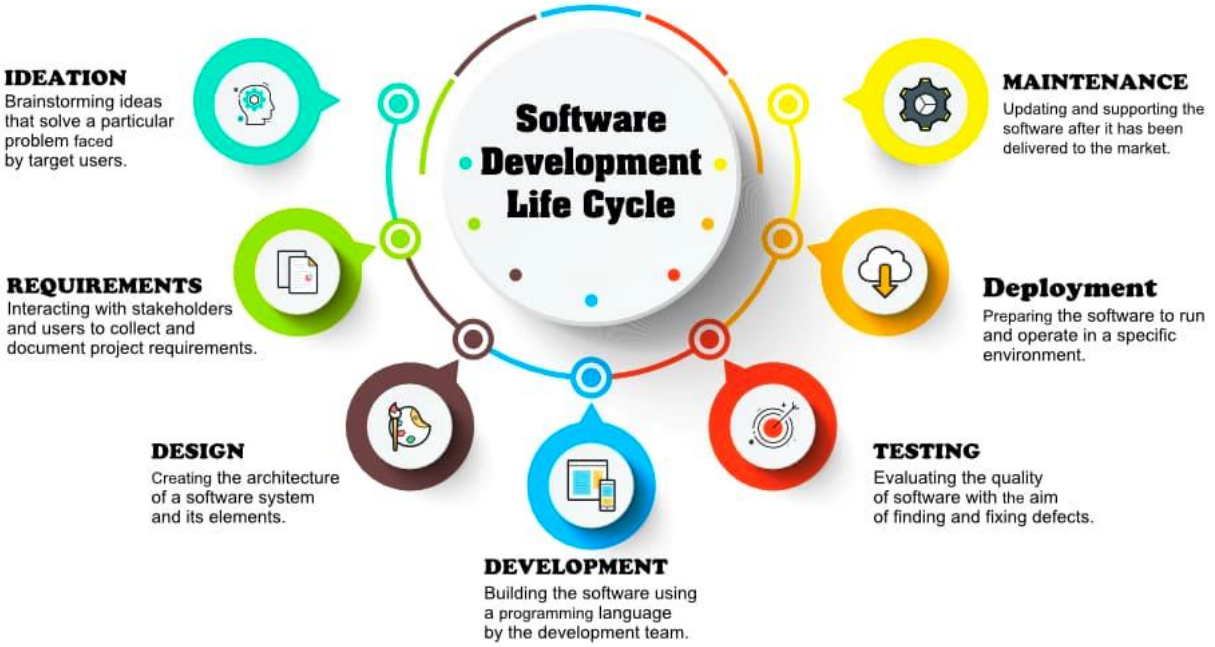


Рис. 3.1.1. Етапи ЖЦ продукту

1. Ініціалізація (планування системи). Фаза планування – найбільш важливий і критичний крок у створенні успішної системи. На цьому етапі:

– точно вирішується що саме потрібно зробити, розробити, які проблеми вирішити, які потреби закрити;

– визначаються проблеми, цілі і ресурси (таких, як персонал і витрати);

– вивчаються можливості альтернативних рішень шляхом зустрічей з клієнтами, постачальниками, консультантами та співробітниками;

– вивчається, як зробити продукт краще, ніж у конкурентів.

Після аналізу цих даних буде три варіанти: розробити нову систему, покращити існуючу або залишити систему як є.

На цьому етапі надзвичайно важлива комунікація з замовником.

[1] https://training.qatestlab.com/blog/technical-articles/popular-software-development-life-cycles/

2. Аналіз вимог (аналіз системи). Необхідно визначити і задокументувати вимоги кінцевого користувача системи – в чому його очікування і як їх здійснити.

Крім того, для проекту робиться техніко-економічне обґрунтування, яке з'ясовує, чи є проект організаційно, економічно, соціально, технологічно здійсненним.

Дуже важливо підтримувати хороший рівень комунікації з замовниками, щоб переконатися, що у вас є чітке бачення кінцевого продукту і його функцій.

Адже скільки людей, стільки й думок, важливо дійти до спільного знаменника.

3. Проєктування (дизайн системи). Фаза дизайну настає після того, коли досягнуто хорошого розуміння вимог споживача і ви точно знаєте, що саме треба втілити.

Ця фаза визначає елементи системи, компоненти, рівень безпеки, модулі, архітектуру, різні інтерфейси і типи даних, якими оперує система. Дизайн системи в загальних рисах може бути зроблений ручкою на листку паперу – він визначає, як система буде виглядати і як функціонувати.

Потім робиться розширений, детальний дизайн, з урахуванням всіх функціональних і технічних вимог, як логічно, так і фізично.

4. Розробка, впровадження і розгортання. Ця фаза йде після повного розуміння системних вимог і специфікацій. Це і є власне процес розробки системи, коли її дизайн вже повністю завершено. В життєвому циклі розробки системи саме тут пишеться код, а також фаза впровадження може включати в себе конфігурацію і налаштування під певні вимоги і функції. На цій стадії система готова до установки у замовника, до запуску в робочий режим. Можливо, кінцевим користувачам буде потрібний тренінг, щоб вони освоїлися з системою і знали, як її використовувати.

Фаза впровадження може бути дуже довгою – це залежить від складності системи.

5. Дослідна експлуатація та інтеграція. Тут відбувається об'єднання різних компонентів і підсистем в єдину цілісну систему. В систему подаються різні вхідні дані і аналіз виходу, поведінки і функціонування.

Тестування стає все важливішим для задоволення споживача, при цьому воно не вимагає знань коду, конфігурації обладнання чи дизайну.

Тестування може виконуватися справжніми користувачами або спеціальною командою співробітників. Воно може бути систематичним і автоматизованим, з тим, щоб упевнитися, що актуальні результати роботи системи збігаються з передбаченими і бажаними

6. Підтримка системи. На цій фазі здійснюється періодична технічна підтримка системи, щоб переконатися, що вона не застаріла.

Сюди входить:

– заміна старого обладнання і постійна оцінка продуктивності;

– здійснюються апдейти певних компонентів, щоб упевнитися, що система відповідає потрібним стандартам і новітнім технологіям, і не схильна до загроз безпеки.

Всі моделі ЖЦ можна розділити на дві великі групи: послідовні (**Waterfall та V-model**) та ітераційні моделі[2].

[2] 1) https://training.qatestlab.com/blog/technical-articles/popular-software-development-life-cycles/
2) https://evergreens.com.ua/ua/articles/software-development-metodologies.html

Waterfall (каскадна модель): етапи залежать один від одного і наступний починається, коли завершений попередній, утворюючи таким чином поступальний (каскадний) рух уперед.

Паралелізм етапів у каскадній моделі, хоч і обмежений, але можливий для абсолютно незалежних між собою робіт. При цьому інтеграція паралельних частин все одно відбувається на якомусь наступному етапі, а не в межах одного. Команди різних етапів між собою не комунікують, відповідаючи тільки за свій етап.

Застосовується на довготривалих і великих проєктах.

Переваги:

– всі стадії проєкту виконуються в чіткій послідовності;

– чіткість етапів дозволяє планувати терміни завершення всіх робіт і відповідні ресурси (грошові і людські);

– вимоги залишаються незмінними протягом усього циклу.

Мінуси:

– складності при формулюванні чітких вимог і неможливість їхньої зміни;

– тестування починається тільки з середини розвитку проєкту;

– до завершення процесу розробки користувачі не можуть переконатися, чи якісний продукт, який розробляється.

Внесення замовником значних змін під час процесу розробки або виникнення неврахованих ризиків призводять до перебудови та перепланування всього проєкту.

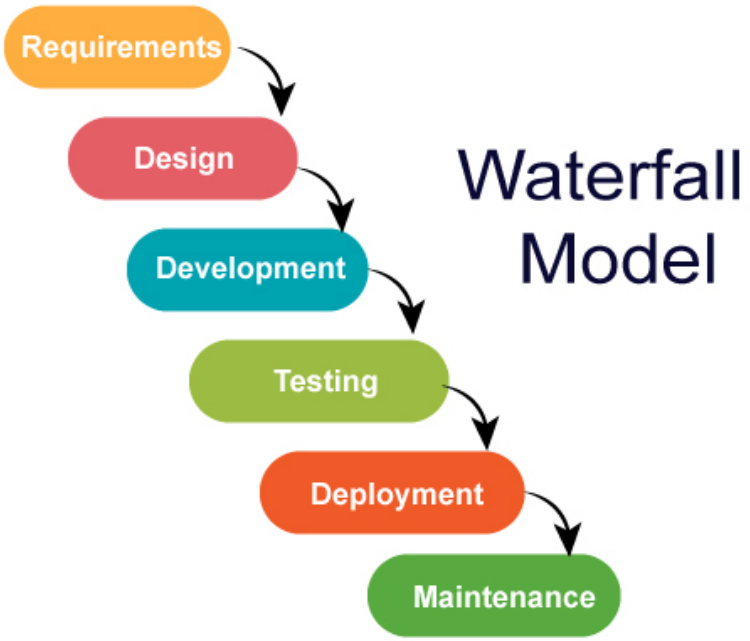


Рис. 3.1.2. Каскадна модель ЖЦ

V-модель (перевірка та валідація) як Waterfall ілюструє процес розробки програмного забезпечення в лінійному послідовному потоці, але вона глибоко демонструє взаємозв'язки між кожною фазою життєвого циклу розробки та пов'язаною з нею фазою тестування.

Замість лінійного руху вниз, етапи процесу згинаються вгору після фази кодування, утворюючи типову V-подібну форму.

Incremental model передбачає розбиття проєкту на частини (етапи, ітерації) і проходження етапів життєвого циклу на кожному з них. Кожен етап є закінченим сам по собі, сукупність етапів формує кінцевий результат. З кожним етапом розробка

наближається до кінцевого бажаного результату або уточнюються вимоги до результату по ходу розробки, і відповідно в будь-який момент поточна ітерація може виявитися останньою або черговою на шляху до завершення. Даний підхід дозволяє боротися з невизначеністю, знімаючи її етап за етапом, і перевіряти правильність технічного, маркетингового або будь-якого іншого рішення на ранніх стадіях.

Плюси:

– замовник може дати свій відгук щодо кожної версії продукту;

– є можливість переглянути ризики, які пов'язані з витратами і дотриманням графіка;

– звикання замовника до нової технології відбувається поступово.

Мінуси:

– функціональна система повинна бути повністю визначена на початку життєвого циклу для виділення ітерацій;

– при постійних змінах структура системи може бути порушена;

– терміни здачі системи можуть бути збільшені через обмеженість ресурсів (виконавці, фінанси).

Рис. 3.1.3. Інкрементна модель ЖЦ

Ітеративна модель не передбачає повного обсягу вимог для початку робіт над продуктом. Розробка програми може починатися з вимог до частини функціоналу, які можуть згодом доповнюватися і змінюватися. Процес повторюється, забезпечуючи створення нової версії продукту для кожного циклу.

У дещо спрощеному вигляді, ітеративна модель складається з чотирьох основних стадій, які повторюються у кожній з ітерацій (plan-do-check-act):

– визначення та аналіз вимог;

– дизайн та проектування – згідно вимогами. Причому дизайн може як розроблятися окремо для даної функціональності, так і доповнювати вже існуючий;

– розробка і тестування – кодування, інтеграція і тестування нового компонента;

– фаза рев'ю – оцінка, перегляд поточних вимог та пропозиції щодо доповнення цих вимог.

За результатами кожної ітерації приймається рішення – чи будуть використані її результати для доповнення існуючої функціональності в якості вхідної точки для

початку наступної ітерації (т.зв. інкрементальне прототипування). Зрештою, досягається точка, в якій всі вимоги були втілені в продукт – відбувається реліз.

Ітеративну модель використовують:

– для великих проєктів;

– коли відомі, принаймні, ключові вимоги;

– коли вимоги до проєкту можуть мінятися в процесі розробки.

Spiral model (спіральна модель): усі етапи життєвого циклу йдуть витками, на кожному з яких відбуваються проєктування, кодування, дизайн, тестування і т.д. Такий процес відображає суть назви: піднімаючись, проходиться один виток (цикл) спіралі для досягнення кінцевого результату. Причому не обов'язково, що один і той же набір процесів буде повторяться від витка до витка. Але результати кожного з витків ведуть до головної мети.

Плюси:

– приділяється особлива увага управлінню ризиками;

– додаткові функції можуть бути додані на пізніх етапах;

– є можливість гнучкого проєктування.

Мінуси:

– оцінка ризиків на кожному етапі є досить витратною;

– постійні відгуки і реакція замовника може провокувати все нові і нові ітерації, які можуть призводити до тимчасового затягування розробки продукту;

– більш застосовується для великих проєктів.

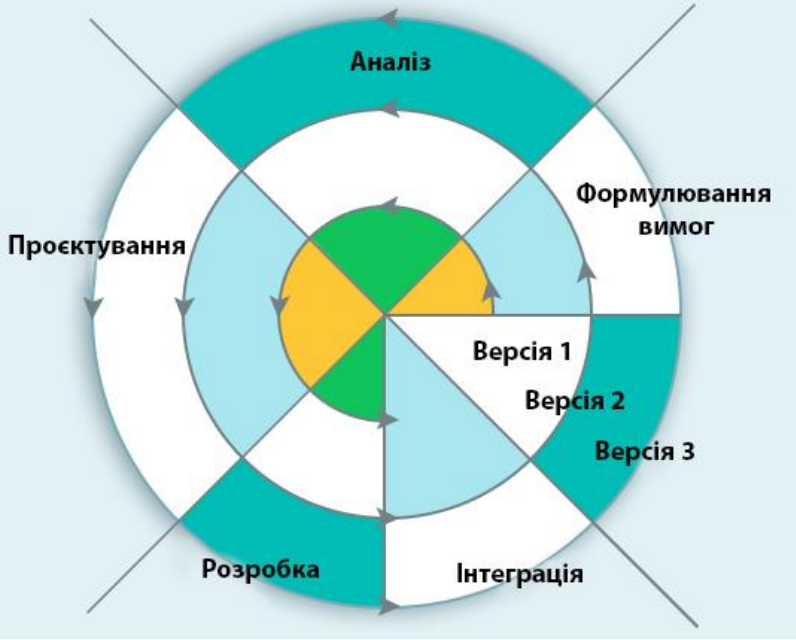


Рис. 3.1.4. Спіральна модель ЖЦ

Agile – набір принципів гнучкої розробки (12 принципів[3]) та ідей.

Основні ідеї Agile:

1. **Люди та співпраця** важливіші за процеси та інструменти.
2. **Працюючий продукт** важливіший за вичерпну документацію.
3. **Співпраця із замовником** важливіша за обговорення умов контракту.
4. **Готовність до змін** важливіша за дотримання плану.

[3] https://agilemanifesto.org/iso/uk/manifesto.html

Один з принципів – взаємодія – має на увазі, що замовник взаємодіє з командою, команда з замовником – усі між собою. Це дозволяє обмінюватися досвідом між учасниками команди і клієнтом і кожному з них впливати на прийняття рішень. За рахунок такого підходу знижуються ризики втрати часу і грошей і підвищується здатність команди вирішувати складні нестандартні завдання з високим ступенем невизначеності.

Однак взаємодії всіх і з усіма можуть вилитися у хаос, що впливає на всі сфери розробки. Тому використовуючи Agile потрібно розуміти обмеження: команди повинні бути невеликі, учасники повинні бути компетентні та мотивовані, ітерації короткі з максимально зрозумілими цілями, встановлені чіткі обмеження за часом і кінцевий результат повинен бути очевидним.

Плюси:

– швидке прийняття рішень завдяки постійним комунікаціям;

– мінімізація ризиків;

– полегшена робота з документацією.

Мінуси:

– велика кількість мітингів і обговорень, що може збільшити час розробки продукту;

– складно планувати процеси, так як вимоги постійно змінюються;

– рідко використовується для реалізації великих проєктів.

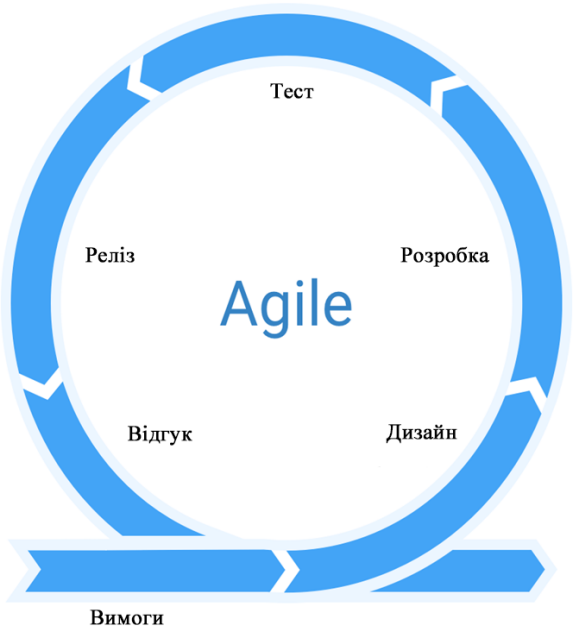


Рис. 3.1.5. Agile-підхід до ЖЦ

Scrum – це гнучка модель розробки ПЗ, в якій робиться акцент на якісному контролі процесу розробки. Ролі в методології (Scrum Master, Product Owner, Team) дозволяють чітко розподілити обов'язки в процесі розробки. За успіх Scrum в проєкті відповідає Scrum Master і є сполучною ланкою між менеджментом і командою. За розробку продукту відповідає Product Owner, який також ставить завдання і приймає остаточні рішення для команди.

Команда – це єдине ціле, в ній результати оцінюються не по кожному окремому учаснику, а по тому, що виходить в результаті у всіх.

Спринти в даній методології тривають від 1 до 4 тижнів. Після кожного спринту команда надає варіант закінченого продукту.

Плюси:

– швидкий зворотній зв'язок від фахівців в різних сферах (дизайнерів, архітекторів, тестувальників та ін.);

– завдяки залученості тестувальника в роботу відбувається швидке додавання нового функціоналу і швидкий запуск продукту з мінімальними функціями;

– самостійна і самоорганізована команда.

Мінуси:

– деякі люди, які знають продукт, стають незамінними, так як документація не надається в процесі розробки;

– неможливо спланувати точну дату завершення, так як все уточнюється за результатами попереднього спринту;

– замовники не завжди можуть зрозуміти суть даної методології і необхідно витратити час на роз'яснення.

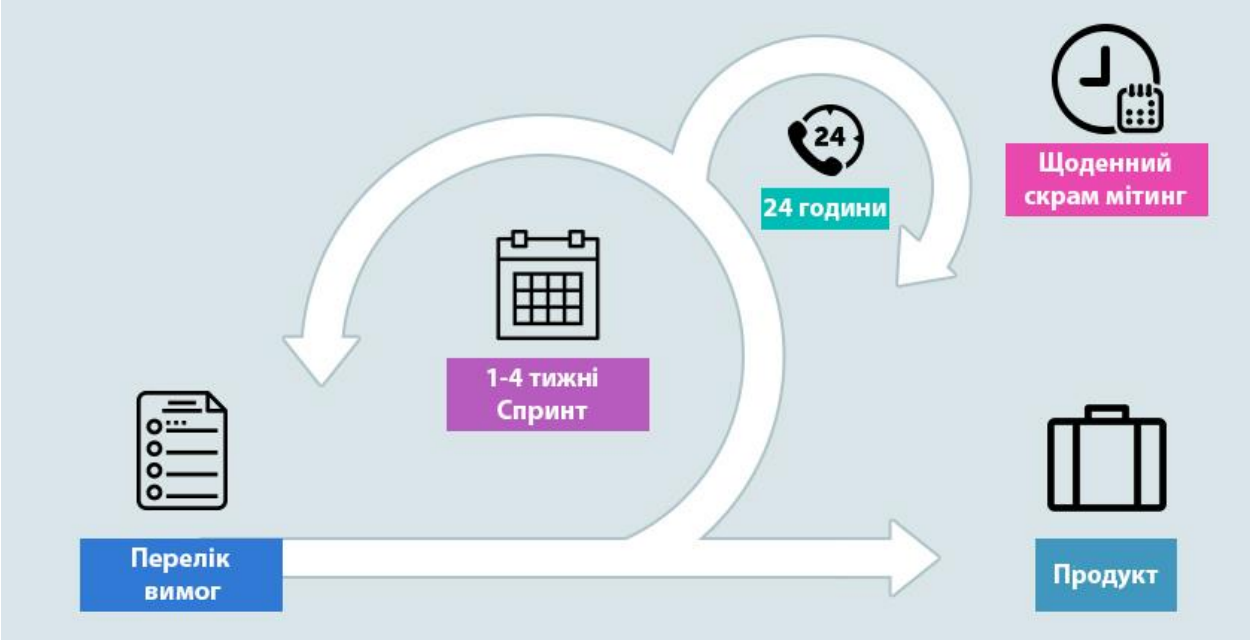


Рис. 3.1.6. Розробка ПЗ за моделлю SCRUM

Lean – ідея ощадливого ставлення до ресурсів (у тому числі часу) і вирішення завдання найпростішим способом. Наприклад: не робиться весь продукт, щоб зрозуміти, що він нікому не потрібен – робиться лендінг і форма підписки і дається на нього реклама, щоб перевірити, що цей продукт викликає інтерес клієнтів. Далі приймається усвідомлене рішення про необхідність розробки.

Коли доходить до розробки продукту, або робиться якесь поліпшення, виробниче або інженерне, спочатку робиться його **MVP (minimum viable product)**. **MVP** – така версія продукту, що виконує свою головну функцію і при цьому її не відхиляють клієнти і визнають її корисність. Звичайно, її можна покращувати і покращувати, але загалом продукт на стадії MVP повинен бути корисним, зрозумілим клієнтові і таким, щоб можна було прийняти рішення про його подальше поліпшення або визнати експеримент невдалим і тестувати нову гіпотезу, витративши при цьому якомога менше ресурсів.

Загалом Lean підхід орієнтується на тестування потреб і цінностей і потрапляння в очікування ринку мінімальними засобами. Lean-підхід не про «технічні» проблеми і їхні рішення інженерними засобами, а про підприємницький підхід до вирішення завдань. Lean передбачає, що команда шукає найпростіше рішення для досягнення результату: технічно, організаційно і т.п., спрощуючи все, що не є дійсно важливим.

У спрощенні сила і головна «пастка» Lean: прагнення все спростити іноді призводить до ситуацій, в яких продукт спрощують настільки, що губляться дійсно важливі функції і продукт по суті виявляється непотрібним, оскільки не несе цінності користувачеві.

За **Kanban** методологією проєкт ділиться на етапи, що візуалізуються у вигляді канбан-дошки. Етапи, це наприклад: планування, розробка, тестування, реліз і ет.п. Завдання у вигляді «карток» переміщуються з етапу на етап. Нові завдання можна додавати у будь-який час. Завдання закривається не по закінченню конкретного часу, а по зміні статусу на «завершено».

Таблиця 3.1.1 – Відмінності між Scrum і Kanban

Scrum	Kanban
Команда приймає участь у конкретній ітерації	Необов'язкова участь
Використання **швидкості** як засобу покращення процесу	Використання **часових рамок** у якості засобів для покращення процесів
Попередня оцінка	Попередня оцінка
Виконання спрінта належить одній команді	Канбан-дошка може бути розділена між декількома командами
Включає в себе використання як мінімум 3 ролей (власник продукту, скрам-майстер, скрам команда)	Відсутність ролей
Скрам-дошка змінюється між спринтами	Канбан-дошка незмінна
Для кожного спринта пріоритет встановлюється в backlog спринта	Призначення пріоритетів не обов'язкове

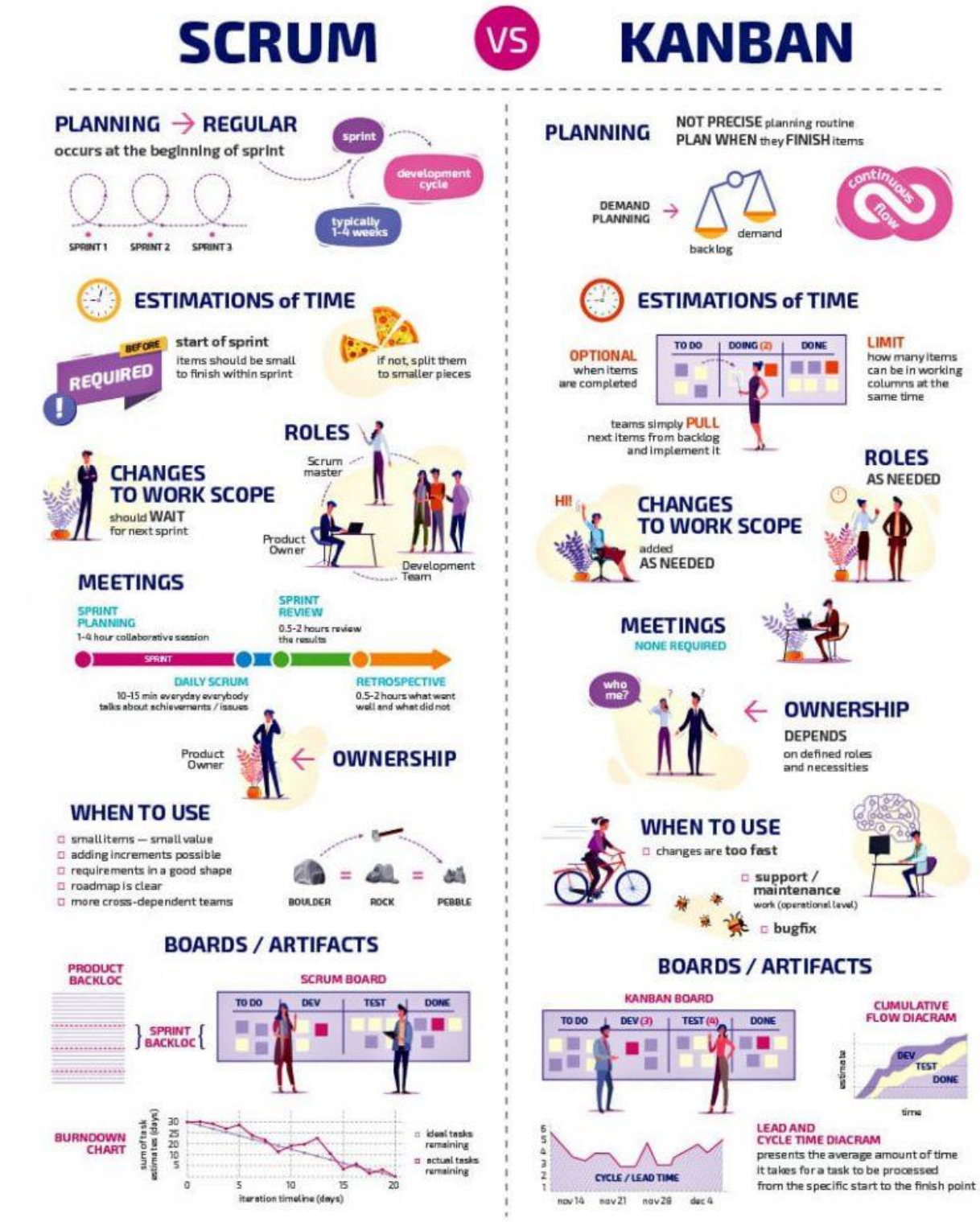


Рис. 3.1.7. Відмінності між Scrum та Kanban

Таблиця 3.1.2 – Що і коли використовувати?

Методологія	Коли
Waterfall Model	Тільки тоді, коли вимоги відомі, зрозумілі та зафіксовані. Суперечливих вимог немає. Немає проблем із доступністю програмістів потрібної кваліфікації. У відносно невеликих про'ктах.
V-Model	Якщо потрібно ретельне тестування продукту. Для малих та середніх проєктів, де вимоги чітко визначені та фіксовані. У разі доступності інженерів необхідної кваліфікації, особливо тестувальників.
Incremental Model	Коли основні вимоги до системи чітко визначені та зрозумілі. У той самий час деякі деталі можуть допрацьовуватися з часом. Потрібно раннє виведення товару ринку. Є кілька ризикових фіч чи цілей.
RAD Model» (rapid application development), Incremental Model	Може використовуватися лише за наявності висококваліфікованих та вузькоспеціалізованих архітекторів. Бюджет проекту великий, щоб сплатити цих фахівців разом із вартістю готових інструментів автоматизованого збирання. RAD-модель може бути обрана за впевненого знання цільового бізнесу та необхідності термінового виробництва системи протягом 2-3 місяців.
Agile Model	Коли потреби користувачів постійно змінюються у динамічному бізнесі.
Iterative Model	Вимоги до кінцевої системи наперед чітко визначені та зрозумілі. Проєкт великий чи дуже великий. Основне завдання має бути визначено, але деталі реалізації можуть еволюціонувати з часом.
Spiral Model	Доцільна для складних і дорогих проєктів, наприклад, таких як розробка системи документообігу для банку, коли кожен наступний крок вимагає більшого аналізу для оцінки наслідків, ніж програмування.

3.2. Загальний огляд технологій: Backend / Frontend, API, mobile, DB, Git, CI/CD, deployment[4]

BACK-END – це програмно-апаратна частина. Головне завдання бекенду – зв'язати базу даних з фронтендом, який у свою чергу повинен відобразити дані у зручному для користувача вигляді. І навпаки, усе що відбувається на фронтенд-частині має надходити у базу-даних через бекенд.

Бекенд складається з трьох базових компонентів: сервера, бази даних і програмних застосунків.[5]

Backend-частина використовує такі мови програмування як: PHP, Ruby, Java, Perl, Python, Node.js, React.js та інші[6].

FRONT-END – це клієнтська сторона, що являє собою користувацький інтерфейс, тобто усе, що бачить та з чим взаємодіє користувач, коли браузер завантажує сторінку. До фронтенду відноситься: дизайн, верстка, наповнення, функціонал (кнопки, форми, віджети), які доступні користувачу. Frontend-частина здебільшого використовує такі мови веб-програмування як: HTML, CSS, Java Script[7].

HTML (з англ. HyperText Markup Language) – спеціальна мова розмітки веб-сторінок, що повідомляє браузеру як саме відображати сайт і де повинні знаходитись його конкретні частини.

CSS (з англ. Cascade Style Sheet – каскадні таблиці стилів) – спеціальний код, який описує стилі веб-сторінок: колір, дизайн, товщина ліній і так далі, тобто відповідає за "косметичну" сторону сайтів.

Java Script – об'єктно-орієнтована, скриптова мова веб-програмування, яка створює скрипти і сценарії, які виконуються на боці користувача та роблять веб-сторінку інтерактивною, динамічною.

FULL-STACK розробники – багатопрофільні спеціалісти, які володіють технологіями та навичками відразу обох частин веб-розробки Frontend і Backend.

API – Application Programming Interface (інтерфейс програмування додатків, програмний інтерфейс програми).

API – це набір способів і правил взаємодії та обміну даними між різними програмами.

API виступає в ролі посередника, що дозволяє програмі 1 отримати доступ до даних і навіть деяких можливостей програми 2. Це дозволяє розробникам поліпшити функціональність продуктів, що випускаються і пов'язати їх з іншими додатками.

MOBILE APP (мобільний додаток) – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях, яке

[4] https://happymonday.ua/hto-takyj-back-end-developer-i-yak-nym-staty
[5] https://nachasi.com/tech/2023/06/01/khto-takii-back-end-rozrobnik/
[6] https://kr-labs.com.ua/blog/shho-take-frontend-i-backend/
[7] https://kr-labs.com.ua/blog/shho-take-frontend-i-backend

завантажується з онлайн магазинів мобільних додатків, таких як App Store, Google Play Market тощо[8].

Розробка програмного забезпечення для мобільних пристроїв потребує врахування їх обмежень та можливостей (функція геолокації, наявність камери, широкий спектр розмірів дисплею, різні технічні характеристики та конфігурації).

Для пристроїв на Android підходить розробка мобільних додатків на java, kotlin. Для пристроїв на iOS використовуються мови програмування Objective-C і Swift.

DATABASE (база даних) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами. В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти[9].

CI/CD/CD. CONTINUOUS INTEGRATION / CONTINUOUS DELIVERY / CONTINUOUS DEPLOYMENT

Безперервна інтеграція (CI), безперервна доставка (CD) та безперервне розгортання (CD) – DevOps[10]-підхід до розробки та апгрейду ПЗ, що передбачає безперервне конвеєрне тестування, складання, доставку та розгортання оновлень. Можливе як окреме застосування компонентів цього підходу (CI або CI+CD), так і їхнє послідовне використання в рамках єдиного процесу (CI+CD+CD).

Безперервна інтеграція (CI) – це метод розробки програмного забезпечення, у якому зміни коду постійно інтегруються у репозиторій. Далі інтеграція автоматично збирається та тестується, що допомагає виявити та усунути конфлікти та помилки. CI покращує якість та стабільність ПЗ та дозволяє прискорити цикл випуску.

Безперервна доставка (CD) – CI + CD. Наступний після CI рівень. Тепер нова версія не тільки створюється і тестується при кожній зміні коду, що реєструється в репозиторії, але і може бути оперативно запущена по одному натисканні розгортання кнопки. Однак запуск розгортання все ще відбувається вручну, ту саму кнопку все ж таки треба комусь натиснути. Цей метод дозволяє випускати зміни невеликими партіями, які легко змінити чи усунути у разі потреби.

Безперервне розгортання (CD) – CI + CD + CD. Після автоматизації релізу залишається один ручний етап: схвалення та запуск розгортання у продакшен. Усі зміни автоматично розгортаються.

Розгортання програмного забезпечення (**deployment**) – це усі дії, що роблять програмну систему готовою до використання.

---

[8] https://bissoft.org/ua/app

[9] https://intellipaat.com/blog/what-is-database/
https://www.javatpoint.com/what-is-database
https://www.atlassian.com/ru/git/tutorials/what-is-git

[10] DevOps (development & operations) — низка практик, призначених для пожвавлення взаємодії розробників із фахівцями інформаційно-технологічного обслуговування та зближення їхніх робочих процесів одне з одним

GIT (version control system, VCS)[11] – розподілена система керування версіями файлів та спільної роботи.

Git – найпопулярніший та безкоштовний інструмент, в якому зберігається код та історія його змін.

До основних завдань Git належить[12]:

- збереження коду та історії змін;
- збереження інформації про користувачів, які змінюють код;
- можливість відкотити код до будь-якої версії;
- можливість об'єднувати різні версії, зміни версій;
- підготовка кінцевого коду до релізу.

GITHUB є одним із множини сервісів на основі Git. Для легшого розуміння можна уявити собі соціальну мережу для розробників, де ті переглядають коди один одного, допомагають в розробці, залишають коментарі тощо.

GitHub дозволяє[13]:

- зберігати код;
- використовувати інструменти для спільної роботи;
- оцінювати роботи інших розробників;
- створювати приватні та публічні репозиторії (за приватні стягується плата – користування від трьох і більше осіб).

GitHub – це також інструмент для зберігання та управління репозиторіями Git, за допомогою якого можна[14]:

- взаємодіяти з репозиторіями;
- керувати правами доступу;
- відстежувати помилки;
- автоматизувати процеси;
- можлива інтеграція з різними CI-системами (CI – Continuous Integration).

GITHUB PAGES – це загальнодоступні вебсторінки, розміщені та опубліковані через GitHub.

GitHub Pages можна використати, щоб продемонструвати деякі проєкти з відкритим кодом, розмістити блог або поділитися своїм резюме.

Найшвидший спосіб розпочати роботу – скористатися інструментом вибору тем Jekyll для завантаження попередньо створеної теми. Потім ви можете змінити вміст і стиль своїх сторінок GitHub.

---

[11] https://www.youtube.com/watch?app=desktop&v=ySKJF3ewfVk
https://nachasi.com/tech/2018/06/05/shho-take-github/
https://w3schoolsua.github.io/githubpages/index.html#gsc.tab=0

[12] https://training.qatestlab.com/blog/technical-articles/what-is-github-and-how-to-work/

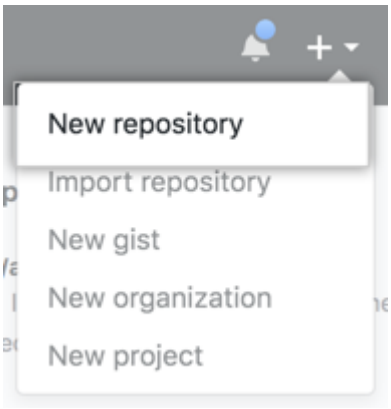
[13] https://training.qatestlab.com/blog/technical-articles/what-is-github-and-how-to-work/

[14] https://training.qatestlab.com/blog/technical-articles/what-is-github-and-how-to-work/

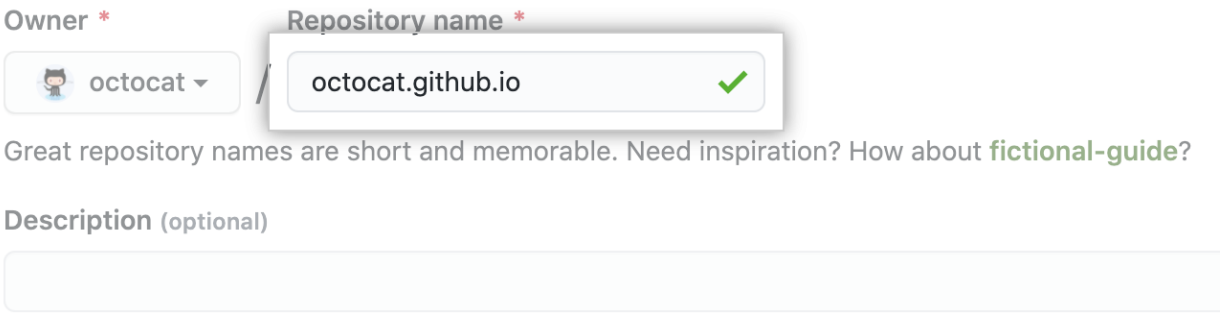
Алгоритм створення сайту[15]:

0. Зареєструйтесь на GitHub:
(див. відео https://www.youtube.com/watch?v=xxVHjPJ8mUE).

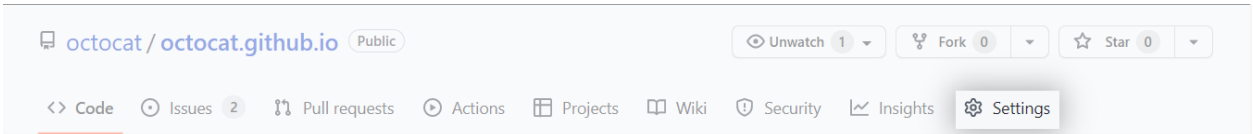
1. У верхньому правому куті будь-якої сторінки скористайтеся спадним меню та виберіть "New repository" ("Новий репозиторій").



2. Введіть `username.github.io` як назву репозиторію. Замініть `username` своїм власним іменем користувача GitHub. Наприклад, якщо ваше ім'я користувача `octocat`, ім'я репозиторію має бути таким: `octocat.github.io`.



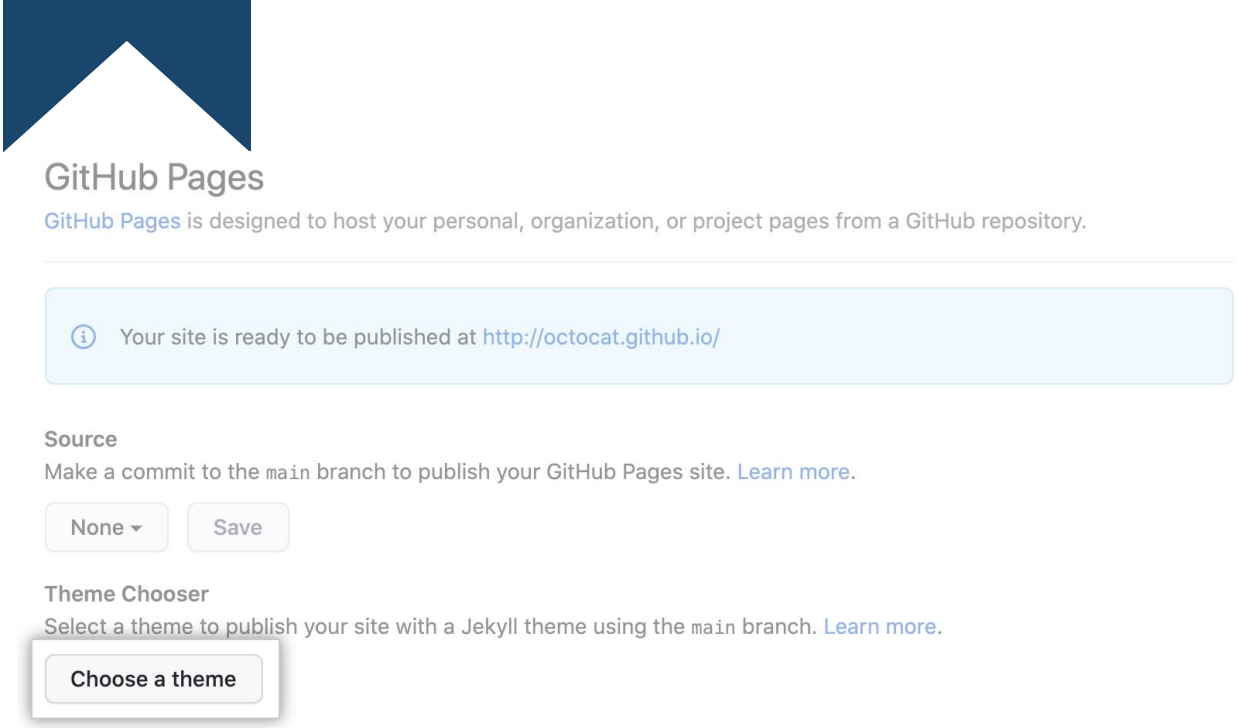
3. Під назвою свого репозиторію клікніть меню "Settings" ("Налаштування").



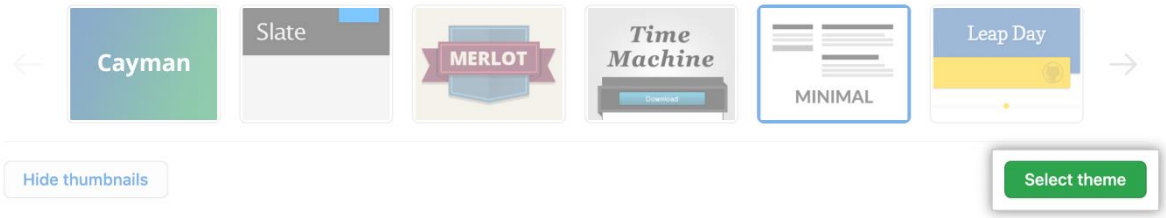
4. У розділі "Code and automation" ("Код і автоматизація") бічної панелі клікніть **Pages**.

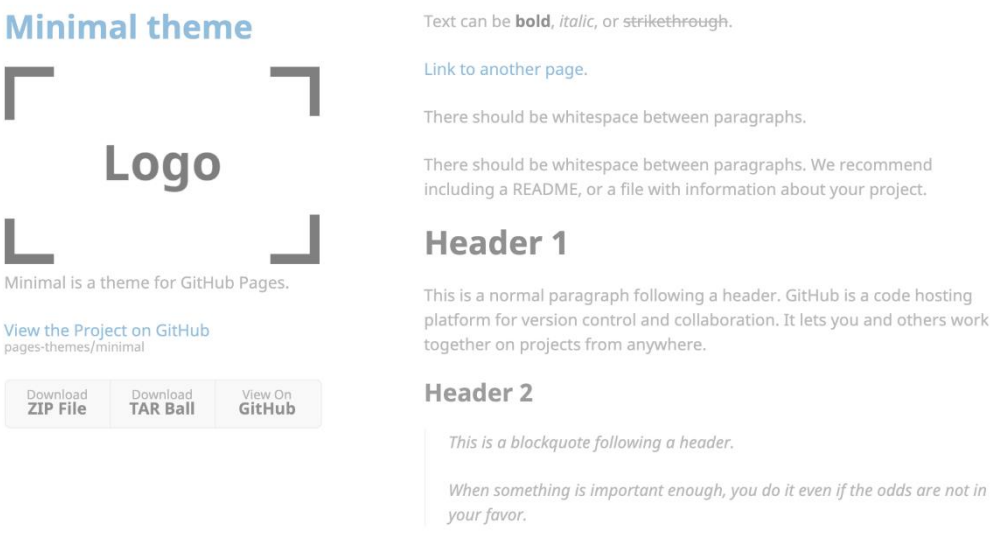
Клікніть "Choose a theme" ("Вибрати тему").

[15] https://w3schoolsua.github.io/githubpages/#gsc.tab=0



Відкриється вікно вибору теми. Перегляньте доступні теми, а потім натисніть "Select theme" ("Вибрати тему"), щоб вибрати тему. Пізніше легко змінити тему, тож якщо ви не впевнені, просто виберіть одну із запропонованих.





5. Після вибору теми файл `README.md` вашого репозиторію відкриється в редакторі файлів. У файлі `README.md` ви будете писати вміст для свого сайту. Ви можете відредагувати файл або залишити вміст за умовчанням.

Завершивши редагування файлу, натисніть "Commit changes" ("Затвердити зміни").

6. Відвідайте `username.github.io`, щоб переглянути свій новий вебсайт.

Примітка: публікація змін на вашому сайті може тривати до 20 хвилин після того, як ви надішлете зміни на GitHub. Просто трохи зачекайте!

За замовчуванням назва вашого сайту – `username.github.io`. Ви можете змінити назву, відредагувавши файл `_config.yml` у своєму репозиторії. Ви також можете додати опис для свого сайту.

Клікніть вкладку **Code** вашого репозиторію.

У списку файлів клікніть `_config.yml` щоб відкрити файл.

Клікніть щоб редагувати файл.

Файл `_config.yml` уже містить рядок, який визначає тему для вашого сайту. Додайте новий рядок із `title:`, а потім потрібним заголовком. Додайте новий рядок із `description:`, а потім потрібним описом.

Наприклад:

```
theme: jekyll-theme-minimal
title: Домашня веб-сторінка Octocat
description: Додайте це в закладки, щоб стежити за оновленнями мого проєкту!
```

Після завершення редагування файлу натисніть "Commit changes" ("Затвердити зміни").

Словник за матеріалами https://sebweo.com/scho-take-git-ta-github-kerivnitstvo-dlya-pochatkivtsiv/[16]

Github

Найпопулярніше віддалене сховище для git-репозиторіїв. Особливості: він дозволяє встановлювати права доступу до проектів, відстежувати і відправляти помилки, приймати запити на поліпшення, підписуватися на повідомлення сховища, використовувати графічний інтерфейс, а не командний рядок. Репозиторії за замовчуванням відкриті, але платні акаунти можуть мати приватні репозиторії.

Фіксація (commit)

Думайте про це як про збереження вашої роботи. Коли ви фіксуєте репозиторій, це схоже на те, що ви збираєте файли в тому вигляді, в якому вони існують в даний момент, і поміщаєте їх в капсулу часу. Фіксація буде існувати тільки на вашому локальному комп'ютері, поки вона не буде відправлена на віддалений репозиторій.

Відправлення (push)

Фіксація поміщає ваші файли в капсулу часу, а відправка – це те, що запускає капсулу в космос. Відправлення – це, по суті, синхронізація ваших збережень (фіксацій, коммітів) з хмарою (знову ж таки, ймовірно, Github). Ви також можете використовувати кілька коммітів одночасно. Ви можете працювати в автономному режимі, зробити багато роботи, а потім передати все це на Github.

[16] https://sebweo.com/scho-take-git-ta-github-kerivnitstvo-dlya-pochatkivtsiv/
https://drukarnia.com.ua/articles/yak-zapustiti-svii-sait-na-github-jpZO_

Гілка (branch)

Уявіть свій git-репо у вигляді дерева. Стовбур дерева, програмне забезпечення, яке запускається, називається майстер-гілкою (Master Branch). Це те, що є онлайн. Гілки цього дерева називаються, як не дивно, гілками. Це окремі екземпляри коду, який відрізняється від основної бази коду. Ви можете відгалузити одну функцію або експериментальний патч. Розгалужуючись, ви можете зберегти цілісність основного програмного забезпечення і мати можливість відкотитися, якщо зробите щось зовсім божевільне. Це також дозволяє вам працювати над своїм завданням, не впливаючи на вашу команду (або вона на вас).

Об'єднання (merge)

Коли гілка виправлена, не містить помилок (наскільки ви принаймні можете судити) і готова стати частиною первинної бази коду, вона буде об'єднана з головною гілкою. Об'єднання – це злиття двох гілок. Будь-який новий або оновлений код стане офіційною частиною кодової бази. Той, хто відгалужується від точки злиття, також буде мати цей код в своїй гілці.

Клонування (clone)

Клонування репо – бере весь онлайн-репозиторій і робить точну копію на вашому локальному комп'ютері. Вам потрібно буде зробити це по ряду причин, і не в останню чергу для збереження сумісності.

Відгалуження (fork)

Форкінг багато в чому схожий на клонування, тільки замість того, щоб робити копію існуючого репо на вашому локальному комп'ютері, ви отримуєте абсолютно новий репо цього коду під своїм власним ім'ям. Ця функція в основному використовується для взяття існуючої кодової бази і перехід з нею в абсолютно новому напрямку, що часто трапляється в програмному забезпеченні з відкритим вихідним кодом; розробники бачать базову ідею, яка працює, але хочуть піти іншим шляхом. Форкінг дозволяє цьому статися. Ви також можете взяти участь в репозиторії іншого розробника, як у своїй особистій пісочниці. І якщо ви робите щось, що, на вашу думку, може йому сподобатися, ви можете зробити попередній запит на об'єднання.

Запит на підтвердження (Pull Request)

Запит на підтвердження – це коли ви відправляєте запит з внесеними вами змінами (або в гілці, або в відгалуженні), які повинні бути перенесені (або об'єднані) в основну гілку (Master Branch) сховища. Це великий час, і тут відбувається диво. Якщо запит на підтвердження буде схвалений, ви офіційно внесете свій внесок в програмне забезпечення, і Github завжди буде показувати, що саме ви зробили. Однак, якщо запит відхиляється з якої-небудь причини, ревізор зможе дати відгук про те, чому запит був відхилений і що ви можете зробити, щоб він був прийнятий.

3.3. Командоутворення при реалізації ІТ-проєкту[17]

Команда проєкту – група осіб, яка підтримує керівника проєкту у виконанні проєкту для досягнення цілей проєкту.

Команда проєкту створюється на період реалізації проєкту та поєднує людей таким чином, що робота набувала синергічного ефекту.

Життєвий цикл команди послідовно проходить декілька етапів, які змінюють один одний:

– формування команди (постановка цілей, розподілення ролей в команді);

– притирання (осмислення цілей, визначення спільного вектору руху);

– нормалізація (досягнення цілей внаслідок компромісів та налагодження комунікації);

– функціонування (збільшення продуктивності за рахунок оптимізації процесів розробки і самоорганізації членів команди);

– розформування команди (отримання результатів, завершення проєкту).

Провідні компанії спочатку визначаються з методологією впровадження проєкту, а вже потім створюють ідеальну команду.

В управлінні проєктами значну роль відіграють застосовані підходи до того чи іншого процесу: жорсткі (орієнтовані на задачу), гнучкі (орієнтовані на команду) та розподілені команди з високим ступенем незалежності.

Важливим фактором психологічного впливу керівника на групу є його авторитет. Авторитет формується з урахуванням особистісних особливостей керівника, його організаторського й мотиваційного потенціалу, стилю керівництва й т. п.

Важливою якістю керівника є вміння користуватися різними стилями керівництва та здатність їх застосовувати залежно від характеру розв'язуваних завдань, специфіки конкретної обстановки, соціально-психологічних особливостей співробітників.

Зазначають три стилі керівництва: авторитарний, демократичний, ліберальний. Великий вплив на стиль керівництва роблять індивідуальні якості особистості керівника.

Уміння керувати – це вміння змінювати стиль керівництва, проявляючи гнучкість.

Основою оптимального динамічного стилю керівництва повинен стати демократичний стиль, для якого характерний тісний зв'язок керівництва з працівниками, розвинене почуття відповідальності перед ними, вміння вступати в контакти з різними людьми, шанобливе ставлення до підлеглих, постійна турбота про них.

[17] Штифурак В.С. Основи успішної управлінської діяльності: (психогігієнічний аспект) Навч-метод. посіб. – Вінниця, 2008. - 194 с. https://studfile.net/preview/7792541/page:38/

Тест «стиль управління» (автор є. І. Рогов)[18]

Інструкція

Тест дозволяє оцінити стиль управління з точки зору співвідношення демократичних і формально-організаційних факторів. Перед вами 40 тверджень. Постарайтеся оцінити своє відношення до цих тверджень відповідно до ваших звичних думок і поводження. При цьому використовуйте наступну шкалу:

С – явище спостерігається систематично (у 80–100 % випадків від того, наскільки це взагалі можливо;

Ч – явище спостерігається часто (60-80 % випадків);

І – явище спостерігається іноді (40-60 %);

Р – явище спостерігається рідко (20-40 %);

Н – явище не спостерігається (0– 20 %).

Твердження:

1. У критичних ситуаціях проводжу в колективі обстеження соціально-психологічного клімату, думок, настроїв людей.
2. У роботі колективу використовуються, де необхідно стандартні правила, методичні вказівки, інструкції й інші управлінські документи.
3. Я обґрунтовую і відстоюю думку колективу (якщо переконаний в його справедливості) перед вищим керівництвом.
4. Ретельно планую роботу своїх заступників.
5. Докладаю всіх зусиль, щоб домогтися від підлеглих виконання усіх указівок.
6. Мої підлеглі чітко знають свої і спільні задачі, що стоять перед колективом.
7. Я особисто вирішую, що і як повинно робитися в колективі для досягнення поставлених цілей, надаючи підлеглим виконавські функції.
8. Допускаю в роботі підлеглих прояв високого рівня ініціативи і самостійності у виборі способів досягнення мети, що стоять перед ними.
9. Допускаю це не тільки у виборі способів, але й у самому процесі вироблення цілей за умови, що підлеглі обґрунтовують їхню важливість і напруженість.
10. Мені як керівникові часто приходиться вирішувати конфліктні ситуації, що виникають між підлеглими.
11. Для забезпечення контролю за дисципліною, вимагаю, щоб мої помічники інформували мене про те, що відбувається у колективі.
12. Допускаю, щоб підлеглі установлювали свій власний темп виконання якої-небудь роботи, якщо це не відбивається негативно на кінцевих результатах.
13. Здійснюючи керівництво, консультуюся і раджуся в розумній мірі з підлеглими.
14. Намагаюся підтримувати в колективі певний етикет, стиль відносин і поводження, а також стежу, щоб підлеглі дотримувалися їх.
15. Заохочую кар’єрне зростання своїх підлеглих.

[18] Управління закладом освіти: Підручник для здобувачів другого рівня вищої освіти педагогічних університетів / С. Г. Немченко, В. В. Крижко, О. С. Боднар, В. В. Радул, О. М. Старокожко, Ю. І. Кондратенко. 2-е вид. перероб. і допов. Бердянськ: БДПУ, 2022. 506 с.

16. Вважаю, що в сучасних умовах кращі результати в управлінні (якість, надійність, точність і т.д.) досягаються, коли людина або колектив працює в умовах примусового режиму, що задається ззовні.

17. У роботі колективу, яким я керую, бувають збої.

18. Інформую колектив про події, що відбуваються в ньому, і в загальному положенні справ у системі управління.

19. Підтримую свій зовнішній вигляд, одяг, порядок у кабінеті, манери поводження на належному рівні.

20. Оцінка і стимулювання праці в колективі здійснюються відповідно до реального внеску кожного у спільний результат.

21. Як керівник я проводжу в житті довгострокову кадрову політику (дотримуюся на практиці певних, відомих колективові принципів розподілу навантаження і обов'язків членів колективу).

22. Аналізуючи роботу своїх підлеглих, доходжу висновку, що багато хто з них – недостатньо знаючі і вмілі працівники, у них не вистачає ініціативи й інших необхідних якостей.

23. У керівництві використовую особистий позитивний приклад як засіб впливу на підлеглих і створити сприятливий соціально-психологічний клімат у колективі.

24. У колективі, яким керую, бувають конфлікти.

25. Створюю умови, при яких підлеглі мають сприятливі можливості висловити думку і впливати на навчально-виховний процес.

26. У керівництві використовую розподіл повноважень (залишаю за собою вирішення найбільш важливих питань, а другорядні делегую на нижчі рівні).

27. Читаю книги і слухаю поради фахівців про те, як працювати з людьми в процесі керівництва.

28. Як керівник дотримуюся на практиці відомих мені теоретичних і прикладних рекомендацій по роботі з людьми.

29. Вважаю, що для підвищення віддачі від людей у сфері управління провідну роль повинні відігравати організаційно-технічні фактори, а в другому плані повинні знаходитися соціально-психологічні фактори.

30. Результати колективу, яким я керую, бувають високими.

31. Як керівник я створюю умови для забезпечення фізичного здоров'я підлеглих на роботі й у побуті, спонукаю їх зміцнювати своє здоров'я.

32. Для забезпечення високих результатів створюю в колективі умови для прояву творчості, новаторства, ініціативи.

33. Вимагаю від підлеглих точних обґрунтувань при формуванні заходів щодо удосконалювання навчально-виховного процесу.

34. Часто доводиться відсувати на другий план вирішення таких питань колективу, як аналіз і поліпшення соціально-психологічного клімату.

35. Докладаю зусиль, щоб домогтися від підлеглих забезпечення високої дисципліни.

36. Робота колективу здійснюється на основі чіткого балансу прав, обов'язків, функцій, відповідальності, їхнього справедливого роз поділу між членами колективу.

37. Для досягнення високих показників у навчально-виховному процесі в колективі здійснюється професійне навчання і заохочується самостійна робота з підвищення кваліфікації.

38. Велику увагу (як керівник) я приділяю контролю дій підлеглих і якості їхньої роботи.

39. Стиль керівництва, якого я дотримуюся, впливає на поводження членів колективу, їхнє відношення до роботи і загальний соціально-психологічний клімат.

40. Стиль керівництва, якого я дотримуюся, впливає на загальні результати колективу.

Обробка результатів

1. Обведіть кружечком порядкові номери наступних тверджень: 7, 10, 16, 17, 22, 24, 29, 34.

2. Поставте по одному балу поруч з тими порядковими номерами, на які дані відповіді Р («рідко»), або Н («ніколи»).

3. Тепер поставте по одному балу поруч з іншими порядковими номерами (крім 7, 10, 16, 17, 22, 24, 29, 34), на які дана відповідь С – («систематично») або Ч – («часто»).

4. У таблицю внесіть бали, які ви поставили по твердженнях з номерами:

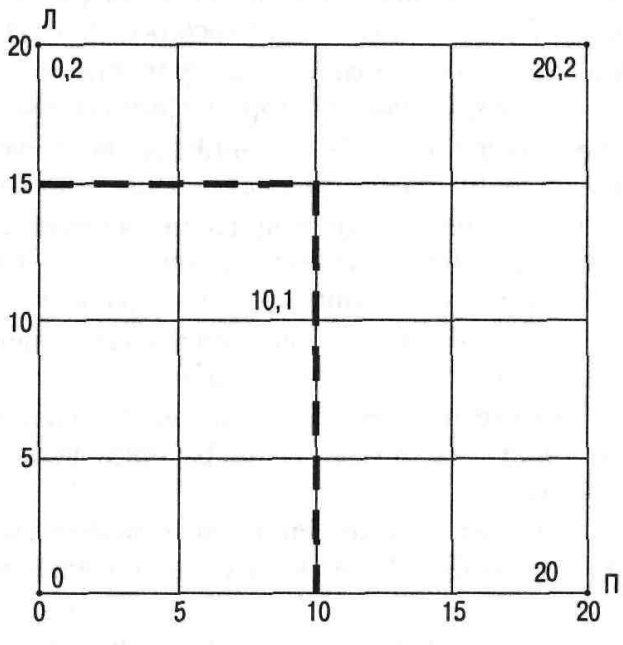
1	3.	7.	8	9.	12.	13.	15.	18.	19.	20.	21.	23.	24.	25.	26.	27.	31.	39.

У тому випадку, якщо бал не поставлений, відповідний квадрат у таблиці залишається порожнім.

5. Підрахуйте кількість внесених у таблицю одиниць – це число Л.

6. Підрахуйте кількість порожніх квадратів – це число П.

7. Внесіть отримані значення Л і П на відповідні осі графіка. Проведіть з цих точок перпендикуляри до осей і знайдіть точки перетину цих перпендикулярів між собою на графіку.



Графік

Аналіз отриманих результатів

Значення Л відбиває зацікавленість у процесі керівництва на формування і підтримку сприятливого соціально-психологічного клімату в колективі, на «людські відносини», на людей.

Значення П відбиває орієнтованість на досягнення певних цілей, опору на формальну організацію і владу, керівника.

Та чи інша точка перетину перпендикулярів, проведених на графіку від отриманих значень Л і П, указує на конкретне значення кількісної оцінки стилю керівництва. Ця оцінка лежить у межах наступних чотирьох крайніх (екстремальних) стилів, ближче до якогось із них.

Стиль 0.0. При цьому типі стилю керівник виявляє дуже мало турботи як про досягнення поставлених цілей навчально-виховному процесі, так і про створення сприятливого соціально-психологічного клімату в колек-тиві. Фактично керівник усунувся від роботи, пустив усе на самоплив.

Стиль 20.20. Це ідеальний стиль керівництва. У керівника з таким стилем у рівному і при тому максимальному ступені виявляється орієн-тованість на досягнення високих результатів і на турботу про створення сприятливого соціально-психологічного клімату в колективі. Такий стиль, як правило, дозволяє домагатися успішного вирішення різних задач у сполученні з умовами для найбільш повного розкриття творчих здібностей членів колективу.

Стиль 20.0. Даний тип стилю властивий найчастіше керівникам-автократам, що піклуються тільки про роботу, ігноруючи людський фактор. Нерідко такий керівник перетворюється на погонича і діє за принципом «давай-давай», що згодом зживає себе настільки, що перестає приносити успіх у досягненні поставлених цілей.

Стиль 0.20. При такому стилі корінник дуже мало піклується про робочий процес, якщо взагалі піклується про нього. Вся увага керівника тут спрямована на підтримку і збереження гарних, приятельських відносин з підлеглими. Створюється такий соціально-психологічний клімат, де усі розслаблені, дружні. І цей психологічний клімат відволікає колектив, відсуваючи на другий план вирішення про-блем. У кінцевому рахунку, така орієнтація на людські відносини не тільки ускладнює досягнення виробничих результатів, але й приводить до підриву сформованого затишного соціально-психологічного клімату. Це може привести до втрати керівником авторитету лідера.

3.4. Експертиза, моніторинг та контроль робіт[19]

Звіт про стан проєкту (Project Status Reports) – це документ, який описує хід проєкту протягом певного періоду часу та порівнює його з планом проєкту.

Менеджери проєктів використовують звіти про стан, щоб інформувати зацікавлені сторони про прогрес та відстежувати витрати, ризики, час та роботу. Звіти про стан проєкту дозволяють менеджерам проєктів та зацікавленим сторонам візуалізувати дані проєкту за допомогою діаграм та графіків.

[19] https://www.projectmanager.com/guides/status-report

Звіти про стан проєкту збираються на кожному етапі його виконання, щоб підтримувати графік та тримати всіх у курсі подій.

Звіт зазвичай включає:

- Робота, яка була завершена.
- План подальших дій.
- Короткий виклад бюджету та графіка проєкту.
- Список дій.
- Будь-які проблеми та ризики, що робиться для їх вирішення.

Справжня цінність звіту про стан проєкту полягає не лише у його використанні як каналу зв'язку. Він також надає документовану історію проєкту. Це формує історичні дані, тому наступного разу, коли PM плануватимете аналогічний проєкт, зможете уникнути помилок або вузьких місць.

Причини, через які менеджери проєктів створюють звіти:

- Допомога команді управління проєктом відстежувати витрати, завдання та терміни.
- Порівняння прогнозу бюджету та часу з фактичними витратами та тривалістю завдання.
- Поліпшення комунікації всередині організації.
- Спрощення процесу спілкування.
- Інформування зацікавлених сторін.
- Донесення ключових результатів до цільової аудиторії.
- Поліпшення організаційної підтримки проєкту чи команди.

Типи звітів про стан проєкту: щоденні, щотижневі, щомісячні або щоквартальні, залежно від вимог до управління проєктами.

Щоденний звіт фіксує, над чим кожен член проєктної команди працював упродовж дня. Він не лише підкреслює те, над чим вони працюють у цей час, а й усуває будь-які проблеми, які заважають їм виконати свої завдання. Він включає резюме сьогоднішньої роботи і того, що було виконано напередодні.

Щотижневий звіт аналогічний щоденному звіту про стан, за винятком того, що він охоплює весь робочий тиждень, а не лише один день. Він включає назву проєкту, дату звіту про стан, короткий опис того, яка робота була виконана за цей період часу, та план дій над чим працювати наступного тижня. Також буде розділ, в якому будуть перелічені будь-які проблеми, ризики та плани реагування на них.

Щомісячний звіт надає аналогічну оновлену інформацію про проєкт чи проєкти, але за період на місяць. Він надає відповідну інформацію для кращого управління проєктом або проєктами. Як і у випадку з іншими звітами, команда повідомляє про досягнуті результати, підбиває підсумки місяця та планує заходи наступного місяця.

Щоквартальний звіт – це короткий знімок проєкту, який легко засвоюється, за певний період часу, в даному випадку за чотири місяці або квартал року. Він включає ту саму інформацію, що й інші звіти про стан, і, ймовірно, включатиме графіки та інші візуальні елементи, щоб полегшити розуміння всіх даних.

Існує безліч різних типів звітів, які можна створювати під час управління проєктом. Деякі з них більше призначені для керівника проєкту, а інші для зацікавлених сторін, власників або клієнтів.

Звіт про стан (Project Status Reports) не слід плутати зі звітом про хід роботи (progress report). Хоча звіт про стан містить дані про прогрес за звітний період, існує безліч іншої інформації, крім простого ходу проєкту.

3.5. Впровадження змін та оцінка рівня задоволеності команди та стейкхолдерів[20]

План управління змінами (Change Management Plan) є важливою частиною будь-якого плану проєкту і може бути вирішальним фактором між успіхом проєкту та провалом.

План управління змінами — це процес, який реалізує зміну чи зміни у проєкті чи у всій організації.

PM можете думати про план управління змінами як про дорожню карту, яка показує всі кроки, які необхідно зробити, від виявлення зміни до його реалізації. Цей план не тільки встановлює курс, за яким можна виконати зміну, але й запитує, як вона вплине на проєкт чи організацію, як це вплине на робочі процеси і чи це змінить ваші відносини з клієнтами чи командами.

Елементи плану управління змінами:

- Ролі в управлінні змінами. По-перше, хто і що робитиме у плані управління змінами? Хто має право подавати запити на зміну, хто їх розглядає та хто їх авторизує? Необхідно визначити ролі та обов'язки для ефективного контролю змін.
- Рада контролю змін. Укомплектуйте свою раду щодо контролю змін людьми, які отримуватимуть запити на зміни та мають право схвалювати їх або накладати вето.
- Розробка процесу/алгоритму. Потрібний опис процесу для ефективного надсилання, оцінки, авторизації, управління та контролю запитів на зміни. Без регламентування процесу керування змінами стає некерованим.
- Форма запиту зміну. Важливо, щоб інформація, яка збирається, була однаковою протягом усього проєкту.
- Журнал змін. Це місце для збору та подальшого відстеження всіх замовлень на зміни. Без центрального пункту, де можна ідентифікувати зміни, затверджувати запити та документувати завдання, неможливо дізнатися, чи досягнуто будь-якого прогресу.
- Програмне забезпечення для планування проєкту може допомогти відстежувати зміни на кожному етапі проєкту, доки він не буде остаточно вирішено.

В відео[21] Дженніфер Бріджес, PMP, розповідає, як створити план управління змінами. Вона пояснює, як зміни вимірюються в порівнянні з базовим планом проєкту, в якому описані час, вартість, масштаб та якість проєкту. Таким чином, план управління змінами буде ґрунтуватися на цій базовій лінії та ставити питання, як, що, коли, де, чому і як визначити зміни та як ними керувати.

[20] https://www.projectmanager.com/training/how-to-make-a-change-management-plan

[21] https://www.youtube.com/watch?time_continue=10&v=wV1KlBpv24k&embeds_referring_euri=https%3A%2F%2Fwww.projectmanager.com%2F&source_ve_path=Mjg2NjY&feature=emb_logo

Порада для професіоналів: існують різні типи управління змінами, і найкращий спосіб отримати цілісне уявлення про це – використовувати потрійне обмеження. Існують процеси, які допомагають у процесі управління змінами, і дуже важливо знати їх. Крім того, майте на увазі, що процедури керування змінами різняться залежно від галузі.

4. ПЕРЕЛІК КОРИСНИХ ПОСИЛАНЬ

1. Стандарт з управління проєктами та Настанова до зводу знань з управління проєктами (Настанова Project Management Body of Knowledge - PMBOK). 2021. Сьоме видання: переклад з англ.. Newtown Square, Pennsylvania: Project Management Institute, Inc., 2021. P. 370. URL: Project Management Body of Knowledge
2. Project Management Institute (PMI)): офіційний сайт. URL: https://www.pmi.org
3. Міжнародна асоціація управління проектами IPMA. – URL: https://www.ipma.world/
4. Інститут проєктного менеджменту України: офіційний сайт. URL: https://pmiukraine.org/about-2/
5. ProjectManager Knowledge Base. URL: https://learn.projectmanager.com/
6. Services for librarians. Cambridge Core. URL: https://www.cambridge.org/core/services/librarians
7. https://www.projectmanager.com/
8. https://www.ba.in.ua/
9. https://happymonday.ua/media
10. https://dou.ua/
11. https://pmdoc.ua/documentation/
12. https://www.wrike.com/guides/
13. https://www.projectpractical.com/
14. http://www.iiba.org/
15. https://www.edx.org/learn/business-analysis
16. https://www.bridging-the-gap.com/