МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

О. С. Яшина, Т. С. Пісклова

УПРАВЛІННЯ ІТ-ПРОЕКТАМИ

Навчальний посібник

Харків «ХАІ» 2020

УДК 004.9(076.5)
Я 58

Рецензенти: д-р техн. наук, доц. М. В. Євланов,
д-р техн. наук, проф. І. П. Гамаюн

Яшина, О. С.

Я 58 Управління ІТ-проектами [Текст] : навч. посіб. / О. С. Яшина, Т. С. Пісклова. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2020. – 128 с.

ISBN 978-966-662-747-9

Описано основні підходи до ІТ-проектів та методи управління ними. Вивчено основні моделі життєвого циклу та галузі знань з управління проектами. Наведено відомості про поширені методологічні підходи і стандарт, а також інформаційні технології та інструментальні засоби управління ІТ-проектами. Розглянуто як класичні, так і сучасні підходи, методології управління проектами на основі Agile-технологій.

Для студентів спеціальності 122 «Комп'ютерні науки» при вивченні курсів «Управління ІТ-проектами», «Системні технології проектування», «Науково-прикладні питання проектування інформаційних систем» та спеціальності 126 «Інформаційні системи та технології» при вивченні курсу «Інформаційний менеджмент ІТ-проектів».

Іл. 63. Табл. 7. Бібліогр. : 10 назв

УДК 004.9(076.5)

ISBN 978-966-662-747-9

ВСТУП

Мета вивчення дисциплини – надання студентам знань, уміння, методичних прийомів і засобів, нових інформаційних технологій та інструментальних засобів для управління проектами в галузі інформаційних технологій.

Завдання – вивчення сучасних методологічних підходів до организациииї та управління проектами створення та експлуатації інформаційних систем, вивчення моделей життєвих циклів ІТ-проектів.

У результаті вивчення навчальної дисципліни студент повинен *знати:*

– основні моделі життєвого циклу (ЖЦ) проекту в галузі інформаційних систем і програмного забезпечення;
– основні стадії ЖЦ і фази управління проектами; зв'язок між ними;
– основні способи організації взаємодії між учасниками ІТ-проекту;
– основні методи планування ІТ-проекту;
– основні підході до економічної оцінки проектів у галузі інформаційних систем і програмного забезпечення;

уміти :

– обирати модель життєвого циклу проекту;
– планувати життєвий цикл проекту;
– будувати та розраховувати мережні моделі плану проекту;
– використовувати комп'ютерні засоби при вирішенні завдань управління проектами;
– оцінювати розроблення плану проекту з досягнення результатів і витрат ресурсів.

1 ОСНОВНІ ПОНЯТТЯ ТА ЗАВДАННЯ УПРАВЛІННЯ ІТ-ПРОЕКТАМИ

Наприкінці 50-х років у США для здійснення програми дослідних і конструкторських робіт зі створення ракети «Поларіс» уперше був використаний метод планування та управління, оснований на ідеї визначення, оцінювання ймовірних термінів і контролю так званого «критичного шляху» всього комплексу робіт. Результати перевершили всі очікування: по-перше, помітно зменшилася кількість збоїв у роботі через неузгодженість використовуваних ресурсів, різко скоротилася загальна тривалість виконання всього комплексу робіт, отримано величезний ефект через зниження сумарної потреби в ресурсах і, відповідно, зменшення загальної вартості програми. Після того як результати виконання програми «Поларіс» стали надбанням громадськості, за минулий з тих пір час метод «критичного шляху» не тільки набув широкого застосування у повсякденній практиці управління, а й зумовив появу спеціальної науково-прикладної дисципліни – управління проектами. У центрі уваги цієї дисципліни знаходяться питання планування, організаці, контролю і регулювання ходу виконання проектів, організації матеріально-технічного, фінансового та кадрового забезпечення проектів, оцінювання інвестиційної привабливості різних варіантів реалізації проектів. У сучасному діловому середовищі актуальність проектного управління як методу організації та управління виробництвом значно зросла. Це зумовлено об'єктивними тенденціями в глобальній реструктуризації бізнесу. Принцип концентрації виробничо-економічного потенціалу поступився місцем принципу зосередження на розвитку власного потенціалу організації. Великі виробничо-господарські комплекси конгломеративного типу швидко заміщуються гнучкими мережними структурами, серед яких домінує принцип переваги використання зовнішніх ресурсів внутрішніми (outsourcing). Тому виробнича діяльність все більше перетворюється в комплекс робіт зі складною структурою використовуваних ресурсів, складною організаційною топологією, сильною функціональною залежністю від часу і величезною вартістю.

1.1 Об'єкт проектного управління

Термін *проект*, як відомо, походить від латинського слова projectus, що в буквальному перекладі означає "кинутий вперед". Таким чином, відразу стає ясно, об'єкт управління, який можна подати у вигляді проекту, відрізняє можливість його перспективного розгортання, тобто можливість передбачити його стан у майбутньому.

Згідно з PMI PMBoK фактично визнаний міжнародним стандартом проекту. Це тимчасове підприємство, призначене для створення унікальних продуктів, послуг або результатів.

Хоча різні офіційні джерела трактують поняття проекту по-різному, в усіх визначеннях чітко проглядаються особливості проекту як об'єкта управління, зумовлені комплексністю завдань і робіт, чіткою орієнтацією цього комплексу на досягнення певних цілей і обмеженнями за часом, бюджетом, матеріальними і трудовими ресурсами.

Однак будь-яка діяльність, у тому числі і та, яку ніхто не збирається називати проектом, виконується протягом деякого періоду часу і пов'язана з витратами певних фінансових, матеріальних і трудових ресурсів. Крім того, будь-яка розумна діяльність, як правило, доцільна, тобто спрямована на досягнення певного результату. І тим не менше в одних випадках до управління діяльністю підходять як до управління проектом, а в інших випадках – ні.

Діяльність як об'єкт управління розглядається у вигляді проекту тоді, коли:

– вона об'єктивно має низку комплексних характеристик і для її ефективного управління важливе значення має аналіз внутрішньої структури всього комплексу робіт (операцій, процедур і т. п.);
– переходи від однієї роботи до іншої визначають основний зміст усієї діяльності;
– досягнення цілей діяльності пов'язано з послідовно-паралельним виконанням усіх елементів цієї діяльності;
– обмеження за часом, фінансовими, матеріальними і трудовими ресурсами мають особливе значення у процесі виконання комплексу робіт;
– тривалість і вартість діяльності явно залежать від організації всього комплексу робіт.

Тому *об'єктом проектного управління* прийнято вважати особливим чином організований комплекс робіт, спрямований на вирішення певної задачі або досягнення певної мети, який обмежений в часі, а також пов'язаний зі споживанням конкретних фінансових, матеріальних і трудових ресурсів. При цьому під *роботою* розуміється елементарна, неподільна частина даного комплексу дій.

Елементарність роботи – поняття умовне й відносне. Те, що недоцільно ділити в одній системі дій, корисно розукрупнювати в інший. Наприклад, якщо за елемент комплексу робіт зі складання автомобіля приймається технологічна операція, то однією з «робіт» може вважатися установлення складальником фари. Ця «робота» в цьому випадку є неподільною, так як залишаються неподільними її чинники – виконавець, предмет і об'єкт дії. Але щойно ми починаємо розглядати виконання цієї роботи як окреме завдання, вона сама перетворюється в комплекс.

Якщо завдання виникає регулярно, а його рішення перетворюється на рутинну діяльність, доведену до автоматизму, то немає ніякого особливого сенсу щоразу, приступаючи до його вирішення, розглядати і моделювати його складну структуру. Результат відомий заздалегідь і час,

витрачений на планування, буде просто втрачено. Тому об'єктом проектного управління є, як правило, комплекс взаємозв'язаних робіт, спрямованих на вирішення деякого оригінального завдання. Але справа в тому, що в сучасному діловому середовищі, при стрімкому розвитку техніки, технології та організації виробництва, при стрімкій зміні видів і різновидів товарів і послуг на ринках поява перед менеджером оригінальних завдань стала фактично звичайною ситуацією. Якщо наприкінці п'ятдесятих років, на зорі зародження проектного управління, об'єктами такого управління були виключно науково-дослідні і дослідно-конструкторські програми, то зараз уже мало кого можна здивувати технічними, організаційними, економічними і навіть соціальними проектами. Уже в самому визначенні типу проекту закладена характеристика області його застосування.

1.2 Характеристики проекту

Проект – це тимчасове підприємство, призначене для створення унікальних продуктів, послуг або результатів.

Це визначення є надто загальним. Для уточнення поняття проекту треба вказати його ключові характеристики.

Тимчасовість проекту

Термін "тимчасове" означає, що у будь-якому проекті є чіткий початок і чітке завершення. Завершення настає, коли досягнуті цілі проекту, або усвідомлено, що цілі проекту не будуть або не можуть бути досягнуті, або зникла необхідність у проекті, і він припиняється. «Тимчасовий» не обов'язково передбачає коротку тривалість проекту: багато проектів можуть тривати протягом декількох років, але в усіх випадках проект кінцевий. Проекти не є постійно триваючою діяльністю.

Крім того, «тимчасовий» не належить до створюваних в ході проекту продукту, послуги або результату. Більшість проектів створюється для досягнення стійкого, тривалого результату. Так, результатом проекту зі зведення монумента на центральній площі міста стане монумент, який буде прикрашати місто протягом століть. Проекти також можуть призводити до запланованих або незапланованих впливів на соціальну, економічну обстановку і навколишнє середовище, що перевищує тривалість самого проекту.

Тимчасова природа може бути характерна не тільки для проектів, але і для інших видів діяльності:

- сприятлива можливість, або ринкове вікно, може тривати досить обмежений час – деякі проекти обмежені часовими рамками для створення нового продукту або послуги;
- команда проекту як робоча одиниця рідко переживає проект, тому що команда, створена виключно для виконання проекту, наприкінці

проекту буде розпущена, а члени команди отримають нові призначення.

Унікальні продукти, послуги або результати

Унаслідок проекту виходять унікальні результати поставки, що є продуктами, послуги або результати. Проект містить:

– продукт і готовий виріб, який можна виміряти і який може бути як кінцевою ланкою виробничого ланцюга, так і елементом;
– здатність надати послуги, такі, як практичні функції, що сприяють виробництву або дистрибуції;
– результати, такі, як наслідки або документи. Наприклад, дослідницький проект отримує дані, які можна використовувати для визначення наявності тенденції або користі нового процесу для суспільства.

Унікальність є важливою характеристикою споруди результатів постановки проекту. Наприклад, численні споруджені офісні будівлі є унікальними, оскільки відрізняються одна від одної власниками, дизайном, місцем розташування, будівельними організаціями тощо. Наявність повторюваних елементів не порушує принципової унікальності кожного проекту.

Послідовне розроблення

Послідовне розроблення – це властивість проектів нарівні з поняттями тимчасовості і унікальності. Послідовне розроблення означає розвиток за етапами і протікання за кроками. Наприклад, зміст проекту формулюється в загальних рисах на ранніх стадіях проекту і згодом деталізується і конкретизується у міру того, як команда проекту розробляє більш чітке і повне уявлення про мету проекту та результати поставки.

Послідовне розроблення специфікацій проекту необхідно ретельно узгоджувати з правильним визначенням змісту проекту, особливо в разі виконання проекту за контрактом. Якщо зміст проекту (тобто склад робіт, які необхідно виконати) визначено правильно, то він повинен контролюватися в міру поступового уточнення специфікацій продукту і проекту.

Чим проекти відрізняються від операційної діяльності

Організації виконують роботи для досягнення ряду цілей. Зазвичай роботи можна подавати як проекти або як операції, хоча вони іноді можуть перетинатися. У них є ряд загальних характеристик:

– виконуються людьми;
– обмежені доступністю ресурсів;
– плануються, виконуються і управляються.

Операційна діяльність і проекти розрізняються, головним чином, тим, що оперативна діяльність – це тривалий у часі і повторюваний процес, тоді як проекти є тимчасовим і унікальними.

Кінцеві цілі проекту і операційної діяльності докорінно відрізняються. Завдання проекту – досягнення поставленої мети, після чого проект завершується. Операційна діяльність, навпаки, часто слугує для забезпечення нормального перебігу бізнесу. Проект відрізняється тим, що він завершується після виконання поставлених конкретних завдань, тоді як операції отримують нові цілі і продовжують виконуватися.

Проекти розробляться на всіх рівнях організації, до них можуть бути причетні як одна людина, так і багато тисяч учасників. Їх тривалість може становити від декількох тижнів до декількох років. У проекті можуть брати участь один або кілька підрозділів (наприклад, спільні підприємства або партнерства). Приклади проектів:

- розроблення нового продукту або послуги;
- здійснення змін у структурі, кадрах або стилі організації;
- розроблення нового транспортного засобу;
- розроблення або придбання нової або вдосконаленої інформаційної системи;
- будівництво будинків або споруд;
- створення водопровідної системи для міста або селища;
- проведення виборчої кампанії;
- упровадження нової процедури або нового процесу на підприємстві;
- виконання вимог контракту.

1.3 Галузі знань з управління проектами

Управління проектами – комплексна дисципліна, що містить знання з різних областей (рисунок 1.1).

Завдання проекту – досягнення конкретної бізнес-цілі при дотриманні обмежень «залізного трикутника» (рисунок 1.2). Це означає, что жоден з кутів трикутника не може бути змінений без впливу на інші.

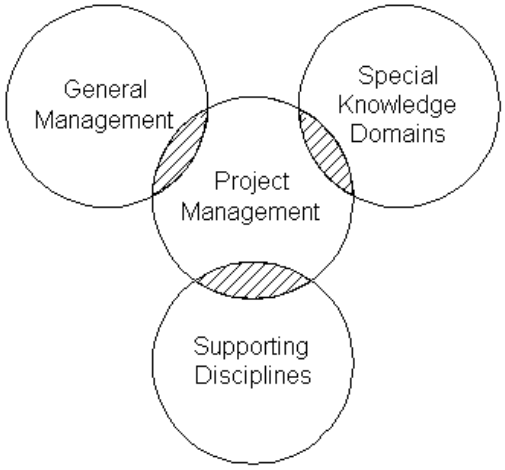


Рисунок 1.1 – Основні складові управління проектами

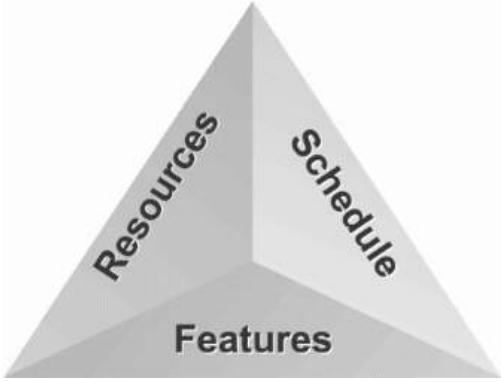


Рисунок 1.2 – Трикутник обмежень проекту

Згідно з PMI PMBoK можуть бути виділені такі області знань з управління проектами:

– управління змістом (зрозуміти бажання замовника і правильно їх реалізувати);
– управління часом (треба вкластись у строки);
– управління вартістю (треба вкластись у бюджет);
– управління якістю;
– управління ризиком;
– управління закупівлями;
– управління персоналом;
– управління комунікаціями;
– управління інтеграціями.

Вивчення вказаних галузей знань визначає подальший план курсу.

2 УПРАВЛІННЯ ЗМІСТОМ ІТ-ПРОЕКТУ

Управління змістом проекту містить процеси, що забезпечують уключення до проекту всіх тих і лише тих робіт, які необхідні для успішного виконання проекту.

2.1 Процеси управління змістом проекту

PMI PM BoK виділяє такі процеси управління змістом:

– **планування змісту** – створення плану управління змістом проекту, в якому документується процес формулювання, верифікації і контролю змісту проекту, а також процес створення і формулювання ієрархічної структури робіт (Work Breakdown Structure, WBS);
– **визначення змісту** – розроблення детального опису змісту проекту як основи для прийняття майбутніх рішень;
– **створення WBS** – розбиття великих результатів поставки проекту і проектних робіт на більш дрібні, більш керовані елементи;
– **підтвердження змісту** – формалізація прийняття завершених результатів постановки проекту;
– **управління змістом** – управління змінами змісту проекту.

У контексті управління проектами термін «зміст» може належати до таких понять:

– **зміст продукту** – властивості і функції, які характеризують продукт, послугу або результат;
– **зміст проекту –** роботи, які необхідно виконати, щоб отримати продукт, послугу або результат із зазначеними характеристиками і функціями.

Результатом проекту зазвичай є один продукт. Однак цей продукт може містити додаткові компоненти, кожен з яких має самостійний зміст, але всі складові змісту знаходяться у взаємозв'язку один з одним. Наприклад, нова телефонна система в загальному випадку буде містити чотири компоненти: апаратуру, програмне забезпечення, навчання і впровадження.

2.2 Опис змісту проекту

Опис змісту проекту докладно описує результати поставки проекту і роботи, необхідні для створення цих результатів поставки. Опис змісту проекту дає також загальне уявлення про зміст проекту всім учасникам проекту і описує основні цілі проекту. Він також дозволяє команді проекту проводити більш детальне планування і служить орієнтиром при виконанні роботи командою проекту; крім того, у разі надходження запитів на зміни або необхідності проведення незапланованих додаткових робіт на його основі визначається їх місце, тобто, чи перебувають вони в рамках проекту чи поза ними.

Ефективність управління загальним змістом проекту з боку команди управління проектом може залежати від того, наскільки детально і до якого рівня деталізації в описі змісту проекту буде визначено, як буде виконуватися робота і які роботи виключені. Управління змістом проекту, своєю чергою, може визначати, наскільки добре команда управління проектом може планувати, управляти і контролювати виконання проекту. Детальний опис змісту проекту безпосередньо або з посиланням на інші документи містить таке.

Цілі проекту. Цілі проекту містять вимірні критерії його успішності. Проекти можуть мати широкий спектр цілей – пов'язаних з бізнесом, вартістю і розкладом проекту, а також технічних і якісних цілей. Цілі проекту можуть також містять планові показники вартості, розкладу і якості проекту. У кожної мети проекту є атрибути (наприклад, вартість), одиниця виміру (наприклад, долар США) і абсолютне або відносне значення (наприклад, не більше 1,5 млн доларів).

Визначення змісту продукту. Описує характеристики продукту, послуги або результату, для створення яких розпочато проект. Вони зазвичай менш деталізовані на ранніх фазах проекту і стають більш докладними на пізніх фазах у міру поступового уточнення характеристик продукту. У той час як форма і зміст будуть відрізнятися, опис змісту повинен завжди бути досить докладним, щоб забезпечити майбутнє планування змісту проекту.

Вимоги до проекту. Описують умови до результатів поставки проекту для задоволення контракту, стандарту, специфікації або інших формально обов'язкових документів. Результати аналізу потреб, побажань

і очікувань усіх учасників проекту перетворюються в перелік вимог до пріоритетів.

Межі проекту. Визначають в цілому те, що міститься в проекті. Явно вказують, що не включається до проекту, щоб виключити ситуацію, коли учасник проекту помилково вважає, що деякий продукт, послуга або результат входять в проект.

Результати поставки проекту. Результати поставки містять як виходи, до яких відносяться створювані проектом продукт або послуга, так і побічні результати, такі як звіти і документація з управління проектом. Залежно від опису змісту проекту результати поставки можуть бути описані в узагальненій або у детальній формі.

Критерії приймання товару. Визначають порядок і критерії приймання готового продукту.

Обмеження проекту. Перелічує і описує обмеження проекту, пов'язані з його змістом і обмежують можливість вибору для команди проекту. До них належать, наприклад, затверджений попередній бюджет або необхідні дати (контрольні події розкладу), установлені замовником або виконуючою організацією. Коли проект виконується за контрактом, то обмеження зазвичай виступають як умови контракту. Обмеження, що перераховуються у докладному описі змісту проекту, традиційно більш численні і деталізовані.

Допущення проекту. Перераховує і описує допущення проекту, пов'язані з його змістом, і потенційний ефект цих припущень у разі, якщо вони виявляться помилковими. Команда проекту періодично ідентифікує, документує і стверджує допущення в рамках процесу планування. Допущення, що перераховуються в докладному описі змісту проекту, зазвичай більш численні і описуються докладніше, ніж припущення.

2.3 Аналіз вимог до інформаційної системи

Для визначення змісту IT0 проекту важливо розуміти вимоги користувача до розроблюваного програмного продукту.

Вимоги поділяються за рівнями. Рівні вимог пов'язані, з одного боку, з рівнем абстракції системи, з іншого – з рівнем управління на підприємстві.

На верхньому рівні представлені так звані бізнес-вимоги (business requirements). Приклади бізнес-вимог: система повинна скоротити термін оборотності оброблюваних на підприємстві замовлень у три рази. Бізнес-вимоги зазвичай формулюються топ-менеджерами, або акціонерами підприємства.

Наступний рівень – рівень вимог користувачів (user requirements). Приклад вимоги користувача: система повинна представляти діалогові засоби для введення вичерпної інформації про замовлення, подальшої фіксації інформації в базі даних і маршрутизації інформації про замовлення до співробітника, який відповідає за його планування і

виконання. Вимоги користувачів часто бувають погано структурованими, дублюючими, суперечливими.

Тому для створення системи важливий третій рівень, в якому здійснюється формалізація вимог. Третій рівень – функціональний (functional requirements). Приклад функціональних вимог (або просто функцій) щодо роботи з електронним замовленням: замовлення може бути створено, відредаговано і переміщено з дільниці на дільницю.

Функціональні вимоги регламентують функціонування або поведінку системи (behavioral requirements). Функціональні вимоги відповідають на питання «що повинна робити система» у тих чи інших ситуаціях. Функціональні вимоги визначають основний «фронт робіт» Розробника, і встановлюють цілі, завдання та сервіси, що надаються системою Замовнику. Функціональні вимоги записуються, як правило, при посередництві розпорядчих правил: «система повинна дозволяти комірникові формувати прибуткові та видаткові накладні». Іншим способом є так звані варіанти використання (uses cases) – популярний і дуже продуктивний спосіб подання вимог. Це – основний, визначальний вид вимог, який буде розглядатися протягом усього лекційного курсу. Нефункціональні вимоги відповідно регламентують внутрішні і зовнішні умови або атрибути функціонування системи: зовнішні інтерфейси (External Interfaces), атрибути якості (Quality Attributes), обмеження (Constraints).

2.4 Створення ієрархічної структури робіт (Work Breakdown Structure – WBS)

WBS – це узгоджена з результатами поставки ієрархічна декомпозиція робіт, які команда проекту має виконати для досягнення цілей проекту і створення зумовлених результатів поставки. З її допомогою структурується і визначається весь зміст проекту. WBS поділяє роботи проекту на більш дрібні і більш керовані частини, де на кожному нижчому рівні WBS дається більш детальне визначення проектних робіт. Для запланованих робіт, що відповідають елементам нижчого рівня WBS (їх ще називають пакетами робіт), можна визначати графік виконання, кошторисну вартість, здійснювати спостереження і контроль за ними.

Декомпозиція – це поділ результатів поставки проекту на більш дрібні і більш керовані елементи; декомпозиція виконується доти, доки робота і результати поставки не визначаються на рівні пакетів робіт. Рівень пакетів робіт є нижчим і являє собою точку, в якій вартість і графік робіт можуть бути оцінені з достатнім ступенем вірогідності. Рівень деталізації пакетів робіт буде варіюватися залежно від розміру і складності проекту.

Для декомпозиції елементів WBS верхнього рівня потрібно поділити на основні елементи, де елементи WBS перевіряють продукти, послуги або результати. Кожен елемент повинен бути чітко і повно визначений та

закріплений за конкретним виконуючим підрозділом, який відповідає за виконання даного елемента WBS.

Планування методом набігаючої хвилі

WBS і словник WBS відображують еволюцію змісту проекту в міру того, як він конкретизується – аж до рівня пакетів робіт. Планування методом набігаючої хвилі – це вид планування способом послідовного розроблення, при якому робота, яку треба буде виконати в найближчій перспективі, детально планується на нижчому рівні WBS, а далекоперспективна робота планується на порівняно високому рівні WBS. Планування робіт, передбачених на один-два найближчі звітні періоди, конкретизується в міру виконання робіт у поточному періоді. Тому на різних стадіях життєвого циклу проекту планові операції можуть мати різну ступінь конкретизації. На ранніх стадіях стратегічного планування, коли інформацію в повному обсязі визначено, операції можуть залишатися на рівні контрольних подій.

3 ЖИТТЄВИЙ ЦИКЛ ІТ-ПРОЕКТУ

3.1 Поняття життєвого циклу проекту

Менеджери проекту або організація можуть розділити проект на фази, щоб забезпечити більш якісне управління з відповідними посиланнями на поточні операції, які виконують організації. Сукупність цих фаз становить життєвий цикл проекту. Багато організацій в усіх своїх проектах використовують певний набір життєвих циклів.

Життєві цикли (ЖЦ) багатьох проектів мають ряд загальних характеристик:

– фази зазвичай йдуть послідовно і обмежуються передачею технічної інформації або здачею технічного елемента;
– рівень витрат і кількість персоналу невеликі на початку, збільшуються по ходу виконання проекту і швидко падають на завершальному етапі проекту. Ці зміни показано на рисунку 3.1.

Рівень невпевненості і, отже, ризик недосягнення цілей найбільш великі на початку проекту. Впевненість у завершенні проекту, як правило, збільшується під час виконання проекту.

Здатність учасників проекту вплинути на кінцеві характеристики продукту проекту і остаточну вартість проекту максимальні на початку проекту і зменшуються під час виконання проекту. Головна причина цього полягає в тому, що вартість внесення змін до проекту і виправлення помилок у загальному випадку зростає у ході виконання проекту.

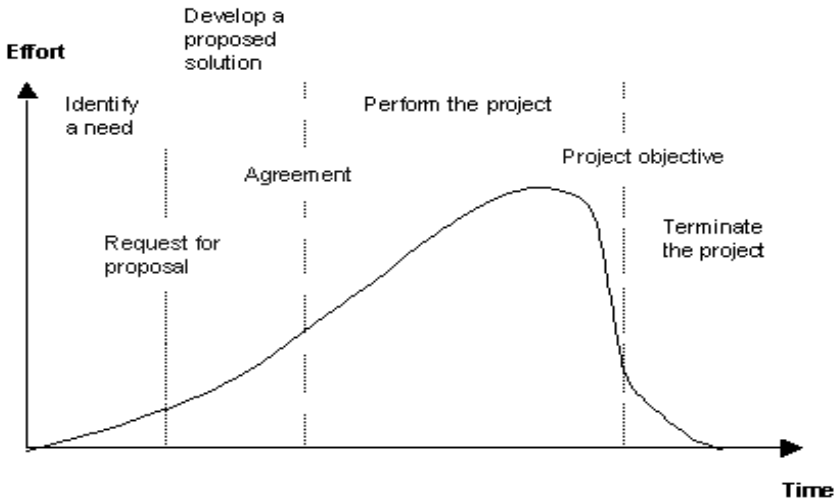


Рисунок 3.1 – Типовий приклад зміни рівня витрат і кількості задіяного персоналу протягом життєвого циклу проекту

3.2 Стадії життєвого циклу проекту

У найзагальнішому випадку прийнято виділяти такі стадії життєвого циклу проекту (рисунок 3.2):

– ініціація – відбуваються висування ідеї, а також підготовка проектних документів. Проводяться детальне обґрунтування, а також маркетингові дослідження, які послужать підмогою для реалізації наступних стадій;
– планування – визначення термінів реалізації задуму, поділ даних процесів на конкретні етапи, а також призначення виконавців і відповідальних осіб;
– виконання – починається відразу ж після того, як були затверджені плани. Мається на увазі реалізація у повному обсязі всіх намічених дій;
– завершення – аналіз отриманих даних і контроль на предмет відповідності їх запланованим. Цей обов'язок у більшості випадків покладається на керівництво.

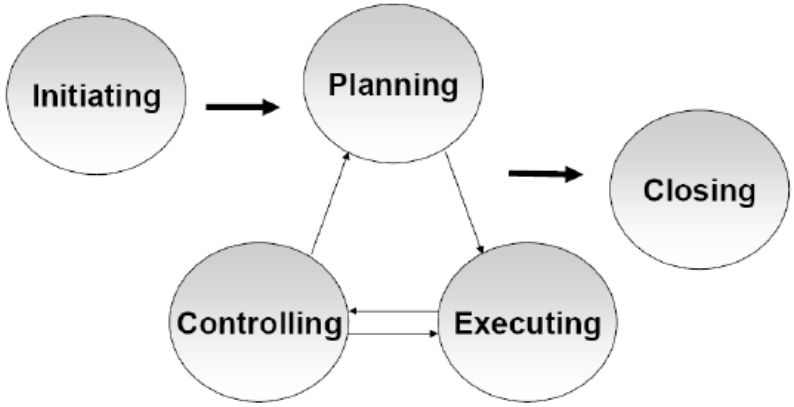


Рисунок 3.2 – Стадії життєвого циклу проекту

Слід розрізняти життєвий цикл проекту і життєвий цикл розроблення, який є його частиною (рисунок 3.3).

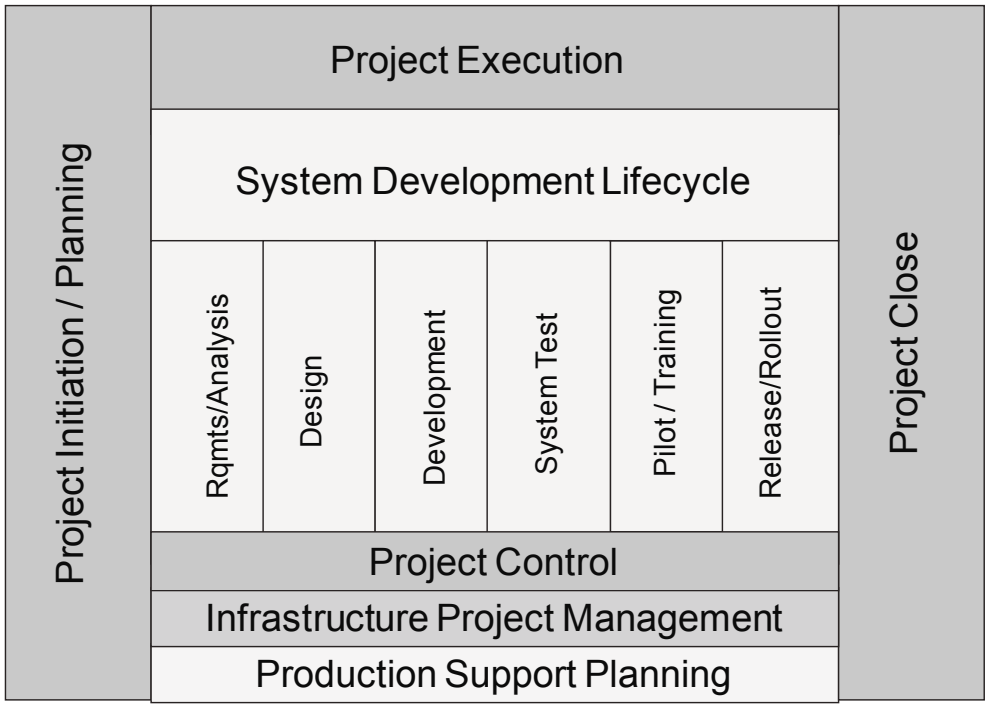


Рисунок 3.3 – Різниця між ЖЦ проекту і ЖЦ розроблення

3.3 Моделі ЖЦ проекту розроблення ПЗ

Каскадну (водоспадну) модель життєвого циклу проекту показано на рисуноку 3.4**.**

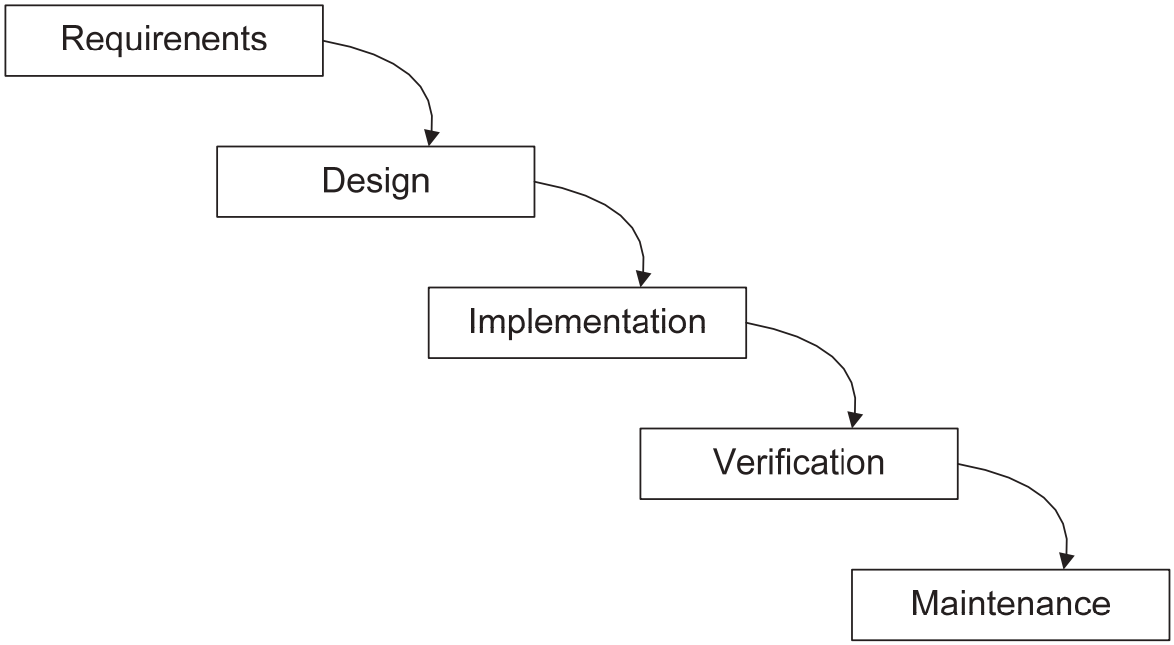


Рисунок 3.4 – Каскадна (водоспадна) модель життєвого циклу проекту

Незважаючи на те, що життєві цикли для кожного окремо взятого проекту або організації можуть істотно відрізнятися, існують деякі загальноприйняті моделі, які можуть послужити базовою основою. Однією з найпоширеніших є водоспадна, або каскадна модель, яка передбачає послідовне виконання кожної запланованої дії і характеризується такими особливостями:

– складання чіткого плану дій щодо досягнення поставлених цілей;
– за кожною дією визначаються певний перелік завдань, а також обов'язкових до виконання робіт;
– упровадження проміжних (контрольних) етапів, на яких буде проводитися контроль за дотриманням раніше розробленого плану.

Коли використовувати каскадну модель:

– тільки тоді, коли вимоги відомі, зрозумілі й зафіксовані. Суперечливих вимог немає;
– немає проблем з доступністю програмістів потрібної кваліфікації;
– у відносно невеликих проектах.

3.4 Спіральна модель

Спіральну модель показано на рисунку 3.5.

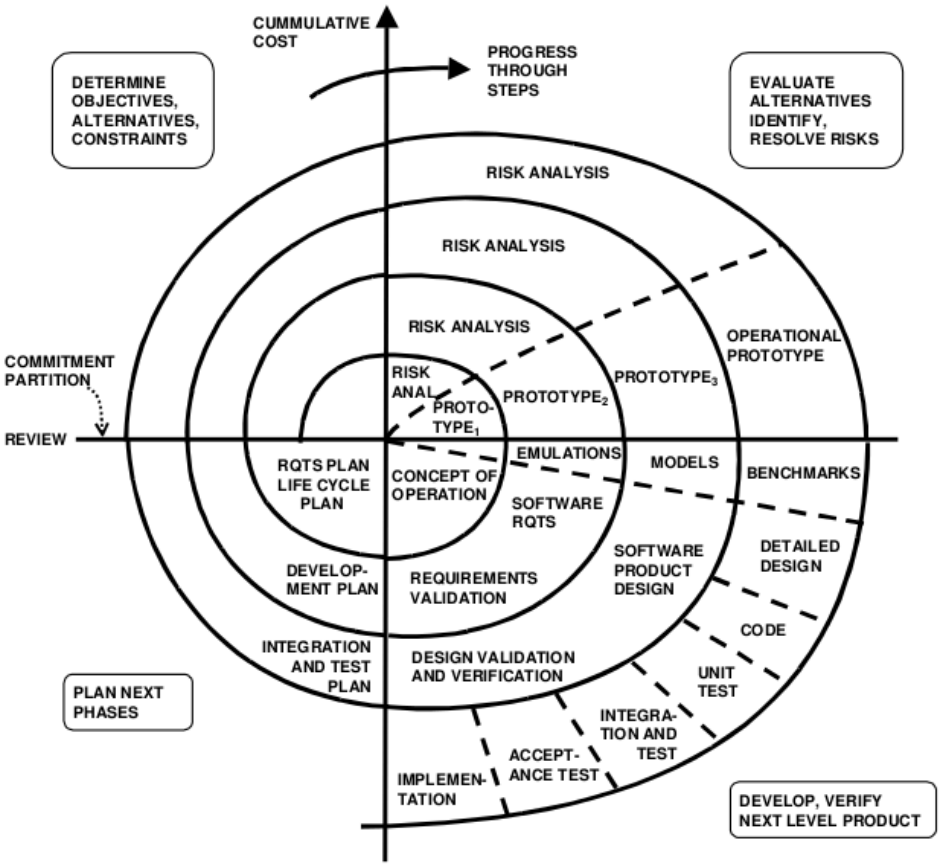


Рисунок 3.5 – Спіральна модель

Життєві цикли проекту, які відрізняються циклічністю, розробляються відповідно до спіральної моделі. На кожному витку визначається ефективність розроблення відповідно до її вартості. Ця модель відрізняється тим, що при її розробленні одна з ключових позицій відводиться ризиковій складовій, яка містить такі пункти:

– нестача кваліфікованих і досвідчених кадрів;
– можливість вийти за рамки бюджету або ж не вкластися у визначені терміни;
– втрата актуальності розроблення під час її реалізації;
– необхідність вносити зміни в процесі виробництва;
– ризики, пов'язані із зовнішніми факторами (перебої з поставками, зміна ринкової ситуації тощо);
– невідповідність виробничої потужності необхідного рівня;
– протиріччя в роботі різних підрозділів.

3.5 V-подібна модель (V-Model)

V-подібну модель (V-Model) показано на рисунку 3.6.

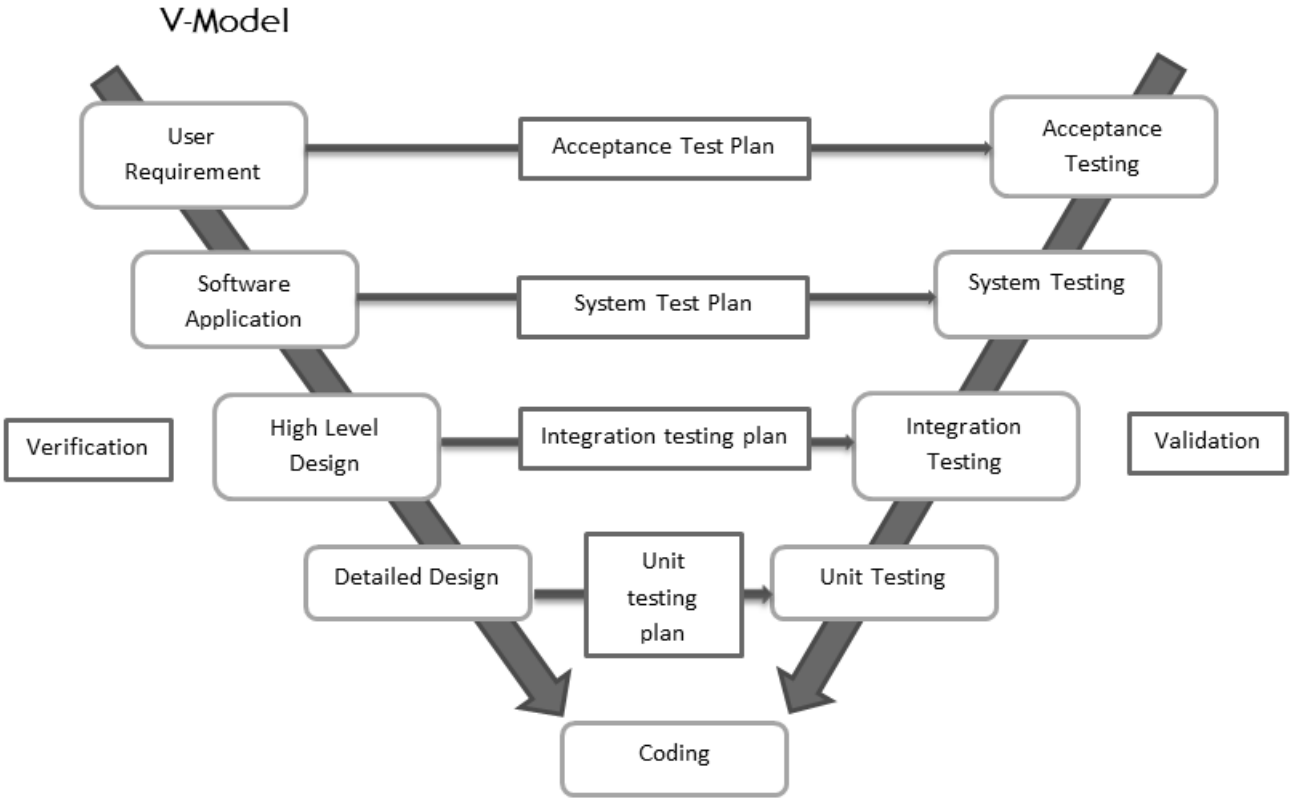


Рисунок 3.6 – V-подібна модель (V-Model)

V-подібна модель застосовна до систем, яким особливо важливе безперебійне функціонування.

Наприклад, прикладні програми в клініках для спостереження за пацієнтами, інтегроване ПЗ для механізмів управління аварійними подушками безпеки в транспортних засобах тощо.

Особливістю моделі можна вважати те, що вона спрямована на ретельну перевірку і тестування продукту, що знаходиться вже на початкових стадіях проектування.

Стадія тестування проводиться одночасно з відповідною стадією розроблення, наприклад, під час кодування пишуться модульні тести.

Коли використовувати V-модель:

– якщо потрібне ретельне тестування продукту, то V-модель виправдає закладену в себе ідею: validation and verification;
– для малих і середніх проектів, де вимоги чітко визначені і фіксовані;
– в умовах доступності інженерів необхідної кваліфікації, особливо тестувальників.

3.6 «Iterative Model» (ітеративна або ітераційна модель)

Ітеративну або ітераційну модель (Iterative Model) показано на рисунку 3.7.

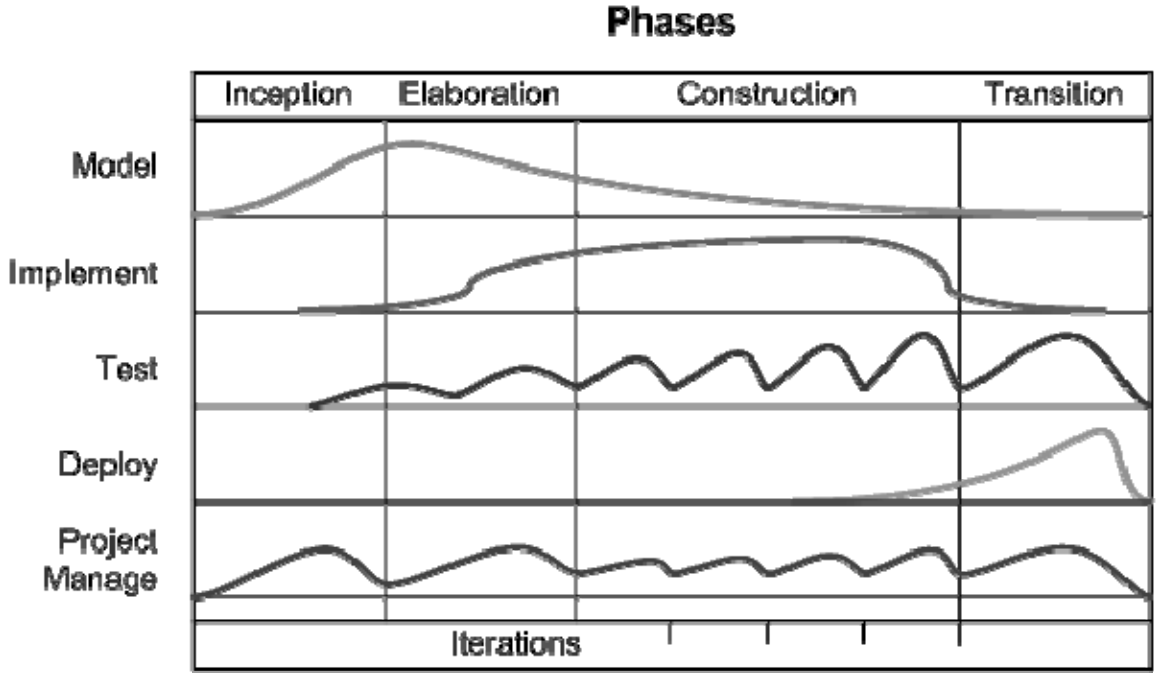


Рисунок 3.7 – Ітеративна або ітераційна модель (Iterative Model)

Ітеративне розроблення ПЗ – це процес створення програмного забезпечення, який здійснюється невеликими етапами, в ході яких проводиться аналіз отриманих проміжних результатів, висуваються нові вимоги і коригуються попередні етапи роботи.

Життєвий цикл проекту при ітераційному розробленні розбитий на послідовність ітерацій, кожна з яких, по суті, є проектом у мініатюрі, тобто містить всі процеси розроблення ПЗ (збір і аналіз вимог, складання специфікацій, безпосередню реалізацію, тестування і запуск), але в рамках однієї ітерації розробляється не весь проект, а тільки його версія чи окрема частина. Як правило, мета кожної ітерації – це отримання версії ПЗ, що містить як нові можливості, реалізовані в ході поточної ітерації, так і функціональність усіх попередніх ітерацій. Результат фінальної ітерації містить всю необхідну функціональність продукту. Бюджет і терміни, необхідні для реалізації фінальної версії, зазвичай спочатку не встановлюються, тому що не визначається загальний обсяг робіт і вимоги формуються при реалізації.

Ітераційна модель життєвого циклу не потребує повної специфікації вимог. Замість цього робота починається з реалізації частини функціоналу, це стає базою для визначення подальших вимог. Цей процес повторюється. Версія може бути неідеальна, головне, щоб вона працювала. Розуміючи кінцеву мету, ми прагнемо до неї так, щоб кожен крок був результативним, а кожна версія – працездатна.

Слід оптимально використовувати ітеративну модель:

– вимоги до кінцевої системі заздалегідь чітко визначені і зрозумілі;
– проект великий або дуже великий;
– основне завдання повинно бути визначено, але деталі реалізації можуть еволюціонувати з плином часу.

3.7 Інкрементна модель

Інкрементну модель показано на рисунку 3.8.

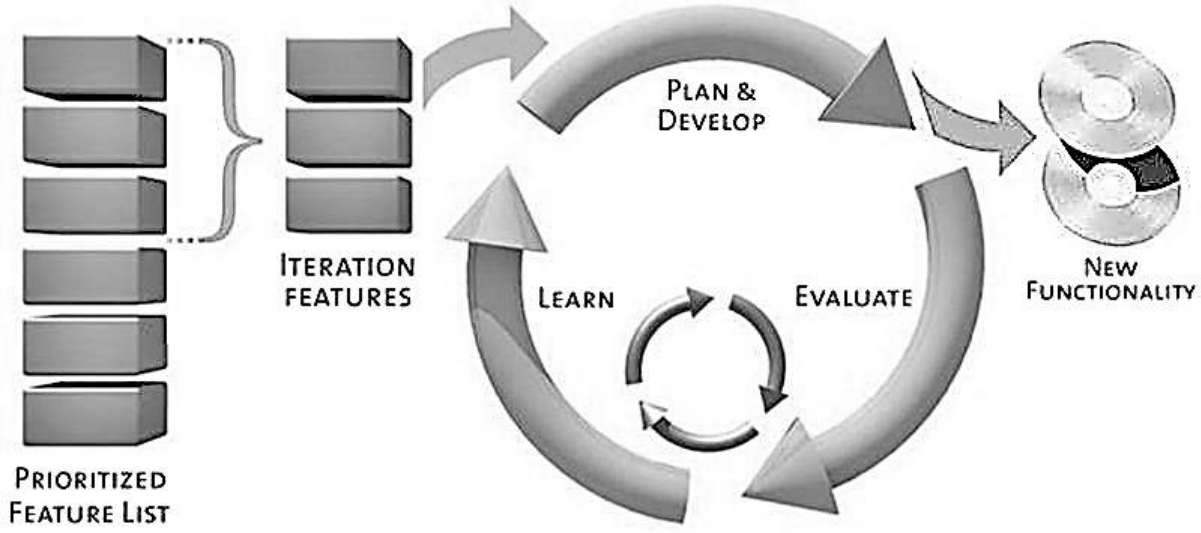


Рисунок 3.8 – Інкрементна модель

Життєві цикли проекту можуть бути розглянуті з точки зору інкрементної моделі. Найбільш актуальним і обґрунтованим її використанням буде в тому випадку, коли передбачається складна і масштабна робота з великою кількістю учасників. У цьому випадку об'ємний проект розчленовується на безліч дрібних складових, які, реалізуючись по частинах, згодом складаються в масштабний проект. Інкрементна модель не потребує одночасного вкладання всієї необхідної суми коштів. Можна поступово вносити невеликі суми, що покривають кожен з етапів. А оскільки весь проект розбитий на невеликі складові, то він є досить гнучким і дозволяє в будь-який час вносити відповідні зміни. І одним з найважливіших моментів є мінімізація ризиків, які рівномірно розподіляються між фазами (інкремент).

В інкрементній моделі повні вимоги до системи діляться на різні форми збирання. Термінологія часто використовується для опису поетапного складання ПЗ. Мають місце кілька циклів розроблення, і разом вони становлять життєвий цикл «мультиводоспад». Цикл розділений на більш дрібні легко створювані модулі. Кожен модуль проходить через фази визначення вимог, проектування, кодування, упровадження і тестування. Процедура розроблення за інкрементною моделлю передбачає випуск на першому великому етапі продукту в базовій функціональності, а потім – вже послідовне додавання нових функцій, так званих «інкрементів». Процес триває доти, доки не буде створена повна система.

Інкрементні моделі використовуються там, де окремі запити можуть бути легко формалізовані і реалізовані.

Коли використовувати інкрементну модель:

– коли основні вимоги до системи чітко визначені і зрозумілі. У той же час деякі деталі можуть доопрацьовуватися з плином часу;
– потрібне раннє виведення продукту на ринок;
– є кілька ризикових функцій або цілей.

Хоча інкрементна й ітеративна моделі багато в чому схожі, вони являють собою різні підходи до створення кінцевого продукту.

3.8 DevOps цикл

DevOps цикл показано на рисунку 3.9.

DevOps (Develpoment and Operations – Розроблення та Експлуатація) – це підхід, при якому інженери-розробники та інженери-адміністратори разом беруть участь у всьому життєвому циклі програмного продукту, від проектування і розроблення – до ретельної підтримки продукту. Таким чином DevOps покликана усунути традиційну роз'єднаність, де одна команда пише код, інша його тестує, третя – розгортає, а четверта відповідає за експлуатацію.

При DevOps співробітники-експлуатанти починають використовувати при підтримці довірених їм систем багато з прийомів, що закріпилися в арсеналі розробника. У DevOps системна інженерія вибудовується точно як потік завдань. Усі ресурси заносяться до системи обліку початкових кодів і покриваються відповідними тестами.

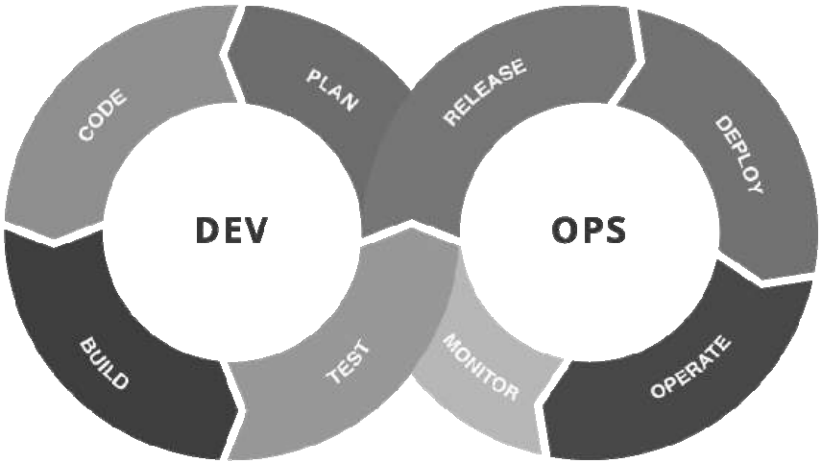


Рисунок 3.9 – DevOps цикл

Протягом життєвого циклу технічні фахівці мають постійно працювати над розумінням того, що повинна робити система і як вона працює в даний час. Безперервна інтеграція і безперервна доставка забезпечують основні методики інтеграції змін.

4 МЕРЕЖНЕ ПЛАНУВАННЯ

4.1 Управління термінами проекту

Управління термінами проекту містить процеси, що забезпечують своєчасне завершення проекту. Процеси управління термінами проекту містять наступне:

Визначення складу операцій – визначення конкретних планових операцій, які необхідно виконати для отримання різних результатів проекту.

Визначення взаємозв'язків операцій – виявлення і документування залежностей між плановими операціями.

Оцінювання ресурсів операції – оцінювання типів і кількості ресурсів, необхідних для виконання кожної планової операції.

Оцінювання тривалості операцій – оцінювання кількості робочих періодів, необхідних для виконання окремих операцій.

Розроблення розкладу – складання розкладу проекту з урахуванням послідовності операцій, їх тривалості, вимог до ресурсів і обмежень на терміни.

Управління розкладом – управління змінами розкладу проекту.

4.2 Основи мережного планування

Для опису, аналізу та оптимізації проектів найбільш придатними виявилися мережні моделі, які є різновидом орієнтованих графів.

У мережній моделі роль вершин графа можуть відігравати події, що визначають початок і закінчення окремих робіт, а дуги в цьому випадку будуть відповідати роботам. Таку мережну модель прийнято називати мережною моделлю з роботами на дугах (Activities on Arrows, AoA). У той же час, можливо, що в мережній моделі роль вершин графа відіграють роботи, а дуги відображують відповідність між закінченням однієї роботи і початком іншої. Таку мережну модель прийнято називати мережною моделлю з роботами в вузлах (Activities on Nodes, AoN).

Нехай безліч A = (a1, a2, a3, ... an) – комплекс робіт, виконання яких необхідне для вирішення певної задачі, наприклад будівництва будинку.

Тоді, якщо безліч V = (v1, v2, v3, ... vm) являтиме собою комплекс подій, що виникають у процесі виконання комплексу робіт, то мережна модель буде задаватися орієнтованим графом G = (V, A), в якому елементи множини V відіграють роль вершин, а елементи множини A – роль дуг, що з'єднують вершини, причому кожній дузі ai можна поставити в однозначну відповідність пару вершин (vsi, vfi), перша з яких буде визначати момент початку роботи ai, а друга – момент закінчення цієї роботи.

Така мережна модель буде мережною моделлю з роботами на дугах.

Тепер нехай безліч A = (a1, a2, a3, ... an), як і раніше, буде розглядатися як комплекс робіт, виконання яких необхідно для вирішення певної задачі, наприклад будівництво будинку.

Тоді, якщо безліч V = (v1, v2, v3, ... vm) представляє комплекс відносин передування-слідування робіт у процесі їх виконання, то мережна модель буде задаватися орієнтованим графом G = (A, V), в якому елементи множини A відіграють роль вершин, а елементи множини V – роль дуг, що з'єднують вершини, причому кожній дузі можна поставити в однозначну відповідність пару вершин (asi, afi), перша з яких буде безпосередньо попередньою роботою у даній парі, а друга – безпосередньо наступною.

Така мережна модель буде мережною моделлю з роботами у вузлах.

Мережна модель може бути подана: 1) мережним графіком, 2) у табличній формі, 3) у матричній формі, 4) у формі діаграми на шкалі часу.

Як буде показано нижче, перехід від однієї форми подання до іншої не становить великих труднощів.

Перевага мережних графіків і часових діаграм перед табличною і матричною формами полягає у їх наочності.

Однак ця перевага зникає прямо пропорційно тому, як збільшуються розміри мережної моделі.

Для реальних завдань мережного моделювання, в яких мова йде про тисячі робіт і подій, викреслювання мережних графіків і діаграм втрачає будь-який сенс.

Перевага табличної і матричної форм перед графічними поданням полягає у тому, що з їх допомогою зручно здійснювати аналіз параметрів мережних моделей; у цих формах застосовні алгоритмічні процедури аналізу, виконання яких не потребує наочного відображення моделі на площині.

Мережним графіком називається повне графічне відображення структури мережної моделі на площині.

Приклад мережного графіка для моделі типу AoN показано на рисунку 4.1.

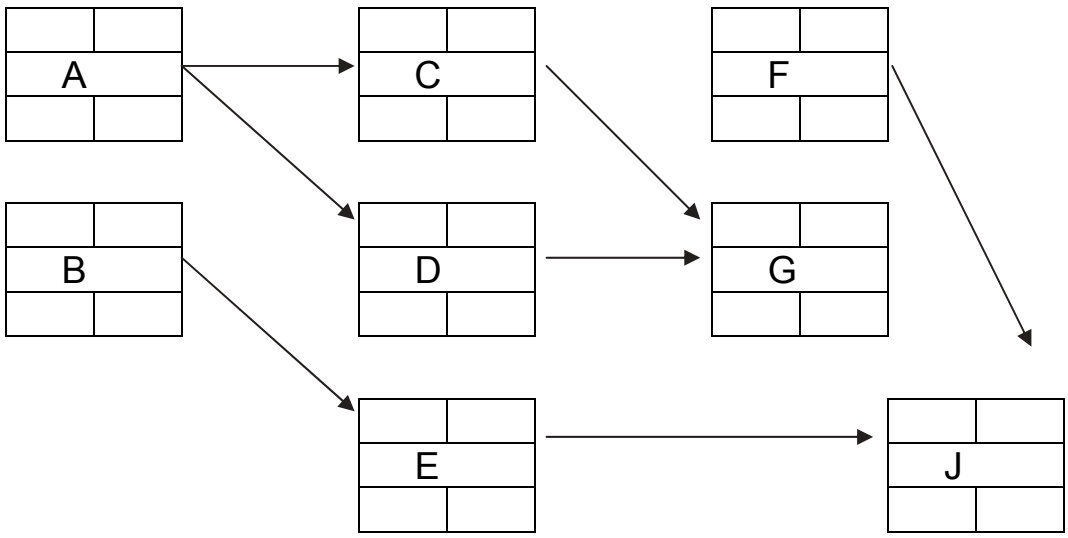


Рисунок 4.1 – Приклад мережного графіка моделі типу AoN

У **табличній формі** мережна модель задається безліччю {A, A (IP)}, де A – це безліч індексів робіт; A (IP) – безліч комбінацій робіт, які безпосередньо передують роботі A.

Для розглянутого вище прикладу таблична форма мережної моделі буде такою, яку наведено у таблиці 4.1.

Таблиця 4.1 – Табличная форма мережної моделі

{A}	{A (IP)}
A	-
B	-
C	A
D	A
E	B
F	C
G	C, D
...J	E, F

Опис мережної моделі у формі часової діаграми (або графіка Гантта) передбачає розміщення робіт у системі координат, де по осі абсцис (X) відкладають час (t), а по осі ординат (Y) – роботу.

Точкою початку відліку будь-якої з робіт буде момент закінчення всіх її попередніх робіт.

Якщо роботі не передує ніщо, то вона відкладується від початку часової шкали, тобто з самого лівого краю діаграми. На рисунку 4.2 показано графік Гантта.

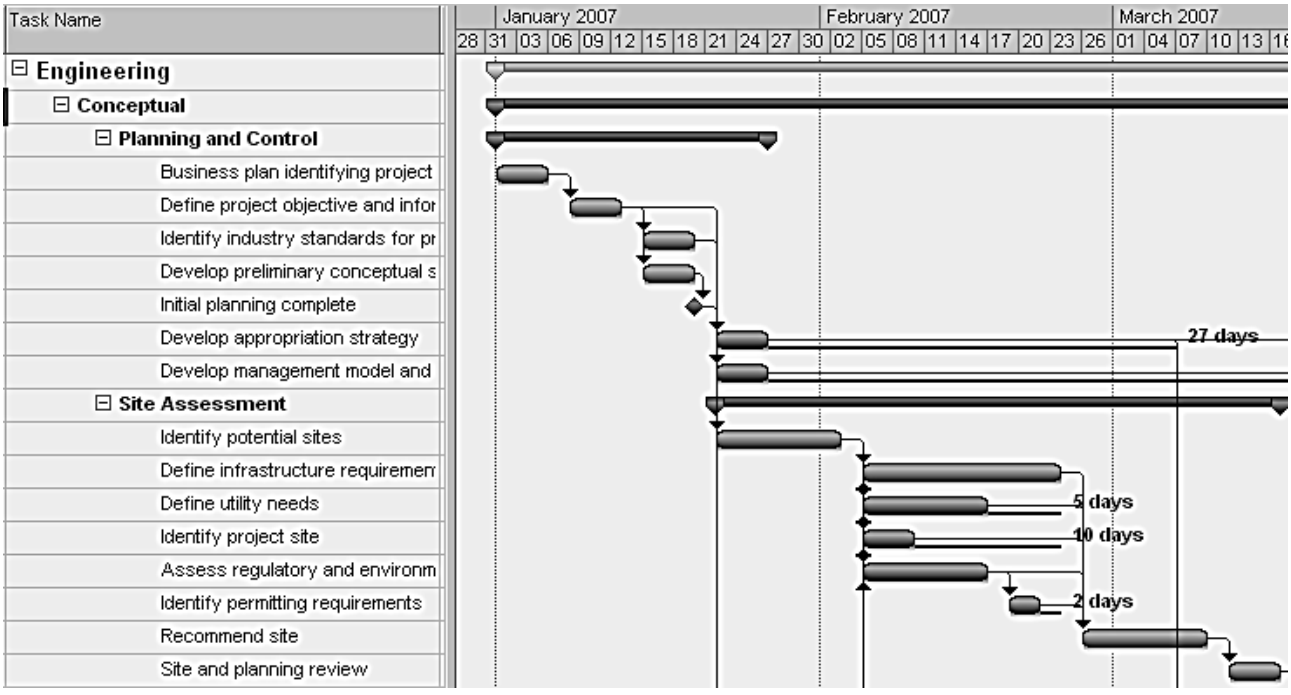


Рисунок 4.2 – Приклад діаграми Гантта

4.3 Часові параметри мережної моделі

Часові параметри (або часові характеристики) мережної моделі є головними елементами аналітичної системи проектного управління. Саме

для їх визначення і подальшого поліпшення виконується вся підготовча, допоміжна робота зі складання мережної моделі проекту і її подальшої оптимізації.

Розрізняють такі часові параметри:

- тривалість роботи (ti) – календарний час, який займає виконання роботи;
- ранній час початку роботи (ESTi) – найбільш ранній з можливих термінів початку виконання роботи;
- ранній час закінчення роботи (EFTi) – дорівнює ранньому часу початку роботи плюс її тривалість;
- пізній час закінчення роботи (LFTi) – найбільш пізній з допустимих термінів закінчення роботи;
- пізній час початку роботи (LSTi) – дорівнює пізньому часу закінчення роботи мінус її тривалість;
- повний резерв часу виконання роботи (TFi) – це максимально можливий запас часу для виконання даної роботи понад тривалість самої роботи за умови, що в результаті затримки кінцева подія для даної роботи настане не пізніше свого пізнього терміну;
- вільний резерв часу виконання роботи (FFi) – це запас часу, на який можна розраховувати при виконанні даної роботи в припущенні, що попередня і наступна події цієї роботи наступають в свої ранні терміни;
- незалежний резерв часу виконання роботи (IFi) – це запас часу, на який можна відкласти початок виконання роботи без ризику вплинути на будь-які терміни настання будь-яких подій в моделі взагалі.

4.4 Критичний шлях

Будь-яка послідовність робіт в мережній моделі називається *шляхом*. Шляхів у мережній моделі може бути дуже багато, але при цьому шляхи, які пов'язують вихідні і завершальні події мережної моделі, називаються *повними*, а всі інші – *неповними*. Сума тривалостей виконання робіт, що становлять той чи інший шлях, називається *тривалістю* цього шляху.

Найтриваліший з усіх повних шляхів називається *критичним шляхом* мережної моделі. Таким чином, тривалість критичного шляху дорівнює сумі тривалостей всіх робіт, які становлять цей шлях.

Роботи, що лежать на критичному шляху, називаються *критичними роботами*, а події – *критичними подіями*.

Уже одного визначення критичного шляху мережній моделі проекту досить для організації управління усім комплексом робіт. Жорстко контролюючи календарні терміни виконання критичних робіт, можна в

підсумку уникнути втрат. У робіт, які не перебувають на критичному шляху, як правило, є резерви часу, що дозволяють на деякий час відкладати їх виконання, якщо це необхідно.

4.5 Методи розрахунку часових параметрів і критичного шляху канонічної мережної моделі проекту

Якщо розміри мережного графіка невеликі, то його часові параметри і критичний шлях можуть бути знайдені шляхом безпосереднього розгляду графіка вершина за вершиною, робота за роботою. Природно, у міру збільшення масштабів моделі ймовірність появи помилки в розрахунках буде зростати в геометричній прогресії. Тому навіть при невеликих розмірах моделі доцільно скористатися одним з найбільш придатних алгоритмічних методів розрахунку, що дозволяють підійти до цього завдання формально.

Найбільш поширеними методами розрахунку часових параметрів мережної моделі є табличний і матричний. Тому, навіть якщо вихідна інформація з мережної моделі подана у вигляді мережного графіка або часової діаграми, приступаючи до аналізу, її слід привести до табличної або матричної форми.

Табличний метод. Складається таблиця, кількість рядків в якій дорівнює кількості робіт, що містить такі стовпчики (у порядку їх слідування зліва направо):

- індекс роботи;
- індекси безпосередньо попередніх робіт;
- індекси безпосередньо наступних робіт;
- тривалість виконання роботи t;
- ранній час початку виконання роботи EST;
- пізній час початку виконання роботи LST;
- ранній час закінчення виконання роботи EFT;
- пізній час закінчення виконання роботи LFT;
- повний резерв часу роботи TF;
- вільний резерв часу роботи FF;
- незалежний резерв часу роботи IF.

Вихідна інформація, пов'язана з описом топології мережної моделі, міститься в стовпцях (1), (2) і (4). Суть табличного методу розрахунку часових параметрів мережної моделі полягає у послідовному заповненні інших стовпців цієї таблиці.

Алгоритм табличного методу передбачає виконання таких послідовних кроків.

КРОК 1. Визначення індексів безпосередньо наступних робіт.

Розглядаємо роботу з індексом [i]. Безпосередньо наступні роботи – це ті роботи, для яких робота [i] є безпосередньо попередньою. Отже, індекси безпосередньо наступних робіт – це індекси тих робіт, у яких в стовпці (2) міститься індекс роботи [i].

КРОК 2. Визначення раннього часу початку EST і раннього часу закінчення робіт EFT.

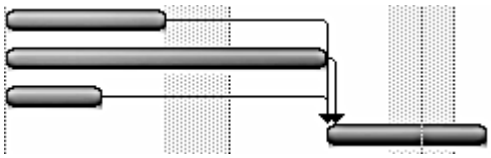
Визначення раннього часу початку і раннього часу закінчення робіт, тобто заповнення стовпців (5) і (7) має здійснюватися одночасно, оскільки час початку наступних робіт залежить від часу попередніх.

Заповнення зазначених стовпців здійснюється послідовно від початку мережної моделі до її кінця, тобто зверху вниз таблиці. При цьому діють такі правила:

- ранній час закінчення EFF даної роботи дорівнює ранньому часу її початку EST (зі стовпця (5)) плюс тривалість роботи t (зі стовпця (4));
- ранній час початку EST виконання роботи дорівнює 0, якщо цій роботі безпосередньо не передує жодна з робіт мережної моделі, або дорівнює максимальному значенню раннього часу закінчення серед усіх робіт, які безпосередньо передують цій робіт (зі стовпця (7)).

Тривалість критичного шляху дорівнює максимальному значенню в стовпці (7).

Принцип визначення ранніх строків виконання роботи показано на рисунку 4.3.



EST equals to largest EFT of predecessors
EFT = EST + Duration

Рисунок 4.3 – Визначення ранніх строків роботи

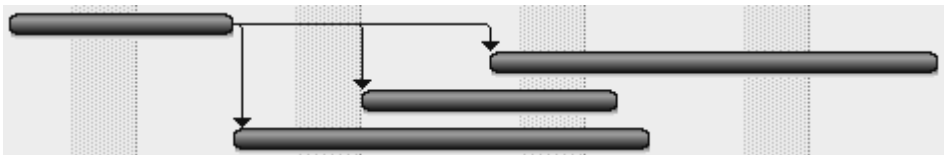
КРОК 3. Визначення пізнього часу закінчення LFT і пізнього часу початку робіт LST.

Визначення пізнього часу закінчення LFT і пізнього часу початку робіт LST, тобто заповнення стовпців (6) і (8) таблиці потрібно проводити одночасно, оскільки час початку попередніх робіт залежить від часу закінчення наступних.

Заповнення зазначених стовпців здійснюється послідовно від кінця мережної моделі до її початку, тобто знизу вгору таблиці. При цьому діють такі правила:

- пізній час початку даної роботи LST дорівнює пізньому часу її закінчення LFT (зі стовпця (8)) мінус тривалість роботи t (зі стовпця (4)).
- пізній час закінчення виконання роботи LFT дорівнює тривалості критичного шляху, якщо за цією роботою немає жодної безпосередньо наступної роботи (зі стовпця (3)) мережної моделі, або дорівнює мінімальному значенню пізнього часу початку LST серед усіх безпосередньо наступних робіт (зі стовпця (6)).

Принцип визначення пізніх термінів виконання роботи показано на рисунку 4.4.



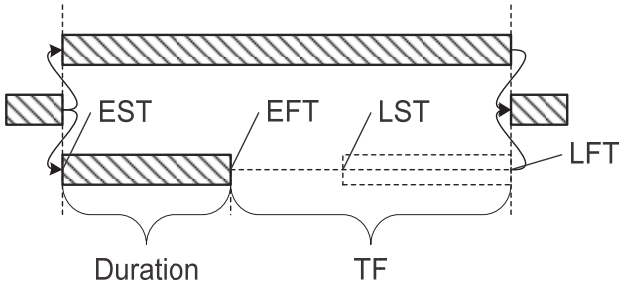
LFT equals to smallest LST of successors
LST = LFT – Duration

Рисунок 4.4 – Визначення пізніх термінів роботи

КРОК 4. Визначення повного резерву часу виконання роботи.

Повний резерв часу роботи [i] знаходиться як різниця значень її пізнього та раннього часу закінчення (відповідно, стовпці (8) і (7)), або як різниця значень її пізнього та раннього початку виконання (відповідно, стовпці (6) і (5)).

Принцип визначення резерву часу виконання роботи показано на рисунку 4.5.



Float = LST – EST or ***Float = LFT – EFT***

Рисунок 4.5 – Визначення повного резерву роботи

За наведеними вище правилами заповнити таблицю 4.2.

Таблиця 4.2 – Приклад розрахунку критичного шляху

Робота	Безпосередньо попередня	Безпосередньо наступна	t	EST	LST	EFT	LFT	TF
A	-	D,E	4	0	0	4	4	0
B	-	H,I,J	7	0	7	7	14	7
C	-	F,G	2	0	20	2	22	20
D	A	M	8	4	18	12	26	14
E	A	H,I,J	10	4	4	14	14	0
F	C	K,L	7	2	22	9	29	20
G	C	N	6	2	25	8	31	23
H	B,E	M	12	14	14	26	26	0
I	B,E	-	6	14	30	20	36	16
J	B,E	K,L	3	14	26	17	29	12
K	F,J	-	4	17	32	21	36	15
L	F,J	N	2	17	29	19	31	12
M	D,H	-	10	26	26	36	36	0
N	G	-	5	19	31	24	36	12

4.6 Узагальнена мережна модель

Узагальнена мережна модель доповнює канонічну модель низкою властивостей і обмежень, що дозволяє побудувати більш реалістичний план.

У цьому методі існує чотири типи залежностей або відносин передування (рисунок 4.6).

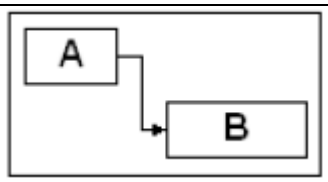
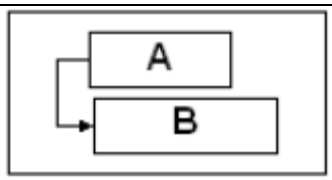
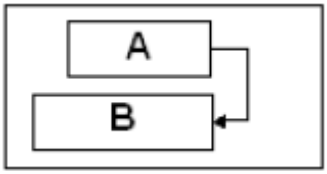
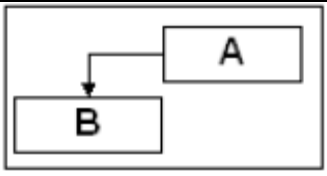
Finish-to-Start (FS) -B can not start till A finishes -A: Construct fence; B: Paint Fence		**Start-to-Start (SS)** -B can not start till A starts -A: Pour foundation; B: Level concrete	
Finish-to-Finish (FF) -B can not finish till A finishes -A: Add wiring; B: Inspect electrical		**Start-to-Finish (SF)** -B can not finish till A starts (rare)	

Рисунок 4.6 – Залежності між роботами

Фініш-старт. Ініціація наступної операції залежить від завершення попередньої операції.

Фініш-фініш. Завершення наступної операції залежить від завершення попередньої операції.

Старт-старт. Ініціація наступної операції залежить від ініціації попередньої операції.

Старт-фініш. Завершення наступної операції залежить від ініціації попередньої операції.

Найчастіше використовується відношення передування типу «фініш-старт». Відносини «старт-фініш» використовуються рідко.

Крім того, накладається ряд обмежень на терміни виконання робіт (рисунок 4.7).

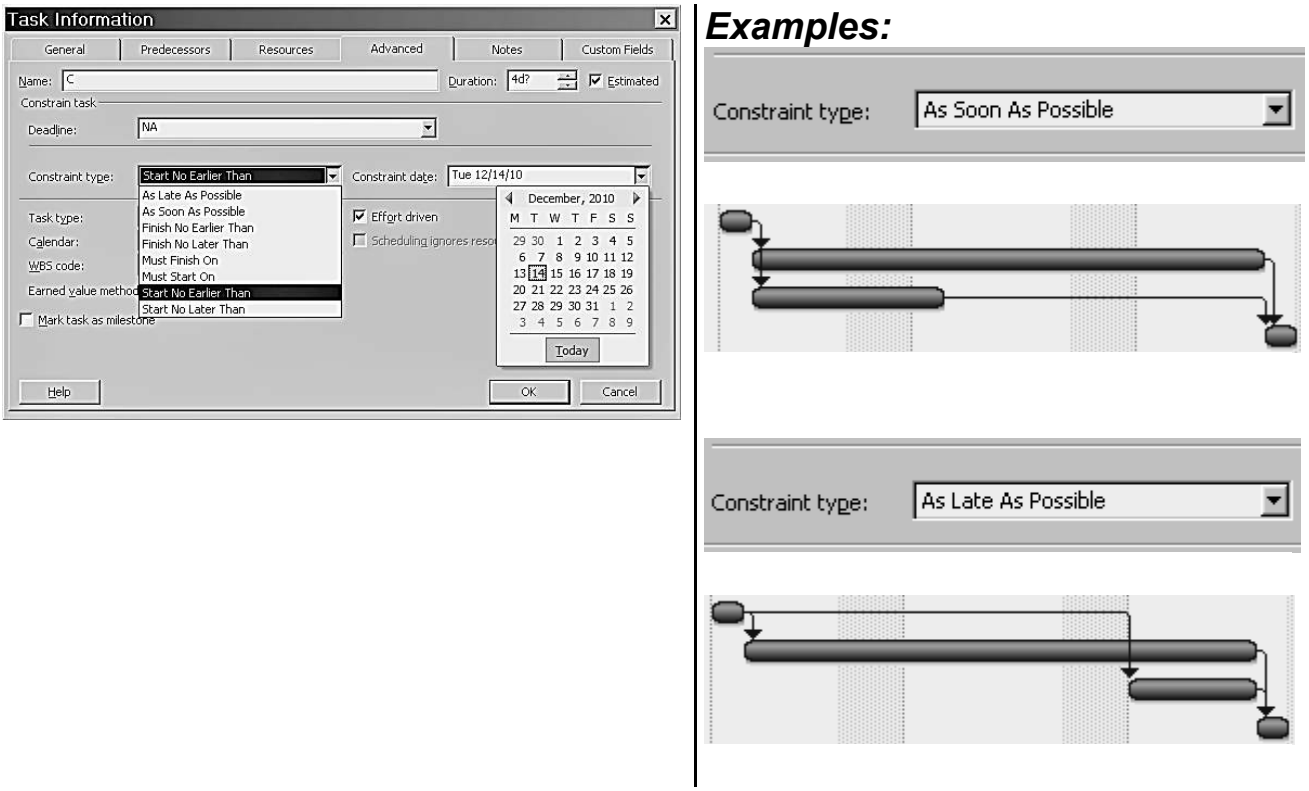


Рисунок 4.7 – Обмеження на терміни виконання роботи

5 ОРГАНІЗАЦІЙНА СТРУКТУРА ІТ-ПРОЕКТУ

5.1 Типові організаційні структури управління проектами

Зміст проекту висуває вимоги щодо оптимальної організаційної структури проекту з точки зору внутрішнього організаційного устрою проекту, тобто з точки зору поділу праці, що закладається в організаційній структурі.

Усе розмаїття організаційних структур, можливих для управління проектом, можна подати у вигляді континууму, межі якого позначають можливі рішення з розподілу праці – вертикальний (функціонально-адміністративне) поділ праці і горизонтальний (проектно-цільове). При цьому слід зазначити, що в цьому випадку розуміється під «вертикальним» поділом праці. Під цим поняттям тут мається на увазі не традиційний розгляд організації за рівнями ієрархії, а поділ праці залежно від участі в різних вертикальних процесах управління та управлінської функції.

Під «горизонтальним» поділом праці розуміється структура діяльності співробітників організації залежно від їх участі в горизонтальних, технологічних процесах виконання робіт. Класичним варіантом реалізації пріоритету поділу праці за вертикальними процесами є функціональна організаційна структура.

Приклад класичної функціональної організаційної структури показано на рисунку 5.1.

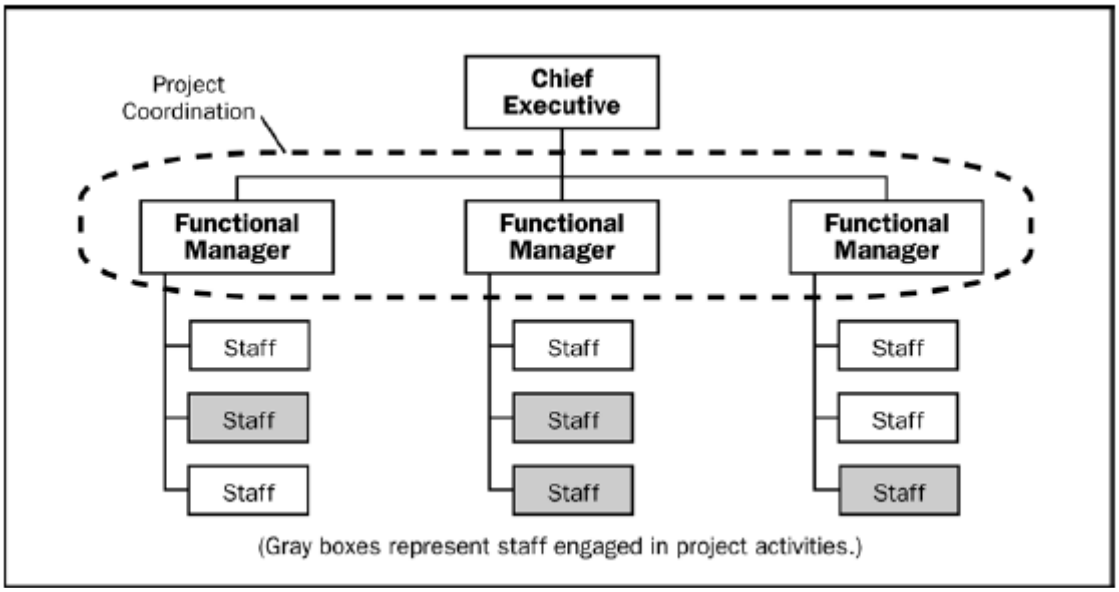


Рисунок 5.1 – Функціональна організаційна структура

Загальні переваги та недоліки функціональної організаційної структури наведено в таблиці 5.1.

У рамках функціональних організаційних структур можуть іноді використовуватися механізми, що дозволяють посилити горизонтальну інтеграцію і таким чином дещо згладити негативні сторони функціональних структур, що розривають єдині процеси на різні операційні відрізки, ефективність виконання яких оптимізується, а взаємодія між якими погіршується, що призводить до зниження ефективності виконання процесу в цілому.

Найбільш часто вживаними механізмами горизонтальної інтеграції функціональних структур є посередники і команди. Найбільш прості елементи горизонтальних зв'язків можуть бути організовані у вигляді так званих посередників.

Таблиця 5.1 – Переваги і недоліки функціональних організаційних структур

Переваги	**Недоліки**
Стимулює ділову і професійну спеціалізацію	Стимулює функціональну ізольованість
Зменшує дублювання зусиль і підвищує ефективність використання ресурсів у функціональних областях	Підвищує кількість міжфункціональних конфліктів і знижує ефективність досягнення загальних цілей
Покращує координацію у функціональних областях	Підвищує кількість взаємодій між окремими учасниками наскрізних, горизонтальних процесів, таким чином знижуючи ефективність комунікацій
Сприяє підвищенню технологічності виконання операцій у функціональних областях	Установлюється функціональна технологічність, що не сприяє вирішенню комплексних, міждисциплінарних проблем
Співробітники мають чітку перспективу кар'єрного росту та професійного розвитку	При залученні співробітників для реалізації проекту вони істотно знижують мотивацію

Посередники – це окремі люди або групи людей, які полегшують взаємодію між підрозділами. Вони становлять один підрозділ в іншому, знижуючи, таким чином, можливість конфлікту і розвантажуючи вертикальні зв'язки. Зазвичай посередники діють на нижніх рівнях ієрархії і запобігають розвитку розбіжностей вже на ранній стадії їх розвитку. Коли питання, що розглядаються посередниками, стають більш складними і важливими або коли більше двох підрозділів мають потребу в координації, тоді замість посередників організовуються команди. Команди автоматично створюються «над» наявними функціональними зв'язками і діють як самостійні організаційні одиниці. Команди можуть створюватися як для вирішення тимчасових завдань, так і на більш постійній основі.

Командам можуть бути делеговані досить широкі повноваження, але також можуть бути створені команди або комітети чисто дорадчого характеру.

При всій своїй корисності посередники і команди допомагають згладжувати недоліки функціональних структур, але все ж мають обмежену застосовність. Для повноцінної горизонтальної інтеграції на

вертикальну функціональну структуру накладається проектно-цільова структура, утворюючи, таким чином, матричну організаційну структуру, приклад якої показано на рисунку 5.2.

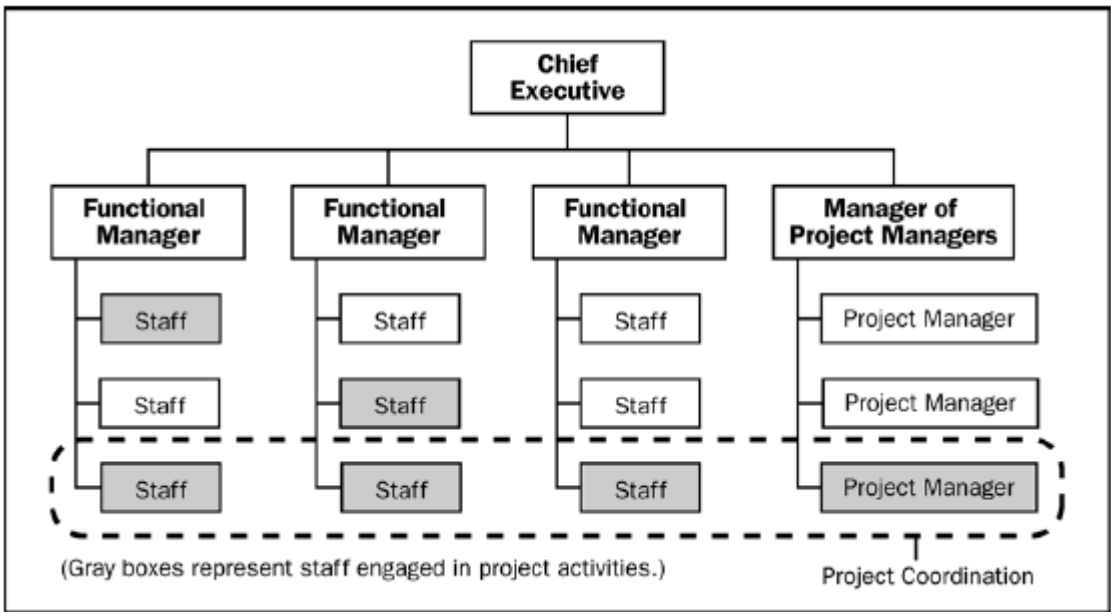


Рисунок 5.2 – Матрична організаційна структура

Будучи комбінацією проектної та функціональної структур, найбільш матрична організація може набувати найрізноманітніших форм залежно від того, до якого «краю» організаційного спектра вона тяжіє в кожному конкретному випадку.

Матричні організаційні структури зазвичай розрізняються за повнотою повноважень керівника проекту (або особи, відповідальної за реалізацію, не завжди це буває керівник проекту), за кількістю тих, хто бере участь у проектній діяльності організації і ролі постійного штату з управління проектом.

Усі види матричних організаційних структур мають переваги і недоліки (таблиця 5.2).

Матричні організаційні структури отримують досить суперечливі оцінки як в теорії, так і в практиці управління проектами. Одні фахівці вважають, що матриці дозволяють поєднати переваги функціональних і проектних організаційних структур і згладити їх недоліки. Інші ж – навпаки переконані, що недоліки обох структур у матрицях залишаються і ще поповнюються власне матричними.

Проте, не зволікаючи на всю складність і такі неоднозначні судження з приводу ефективності використання матричних структур, вони знаходять дуже широке застосування в управлінні проектами в багатьох галузях економіки: хімічній промисловості, банківській справі та страхуванні, виробництві товарів.

Таблиця 5.2 – Переваги і недоліки матричних організаційних структур

Переваги	Недоліки
Проект і його цілі перебувають у центрі уваги так само, як і потреби клієнтів	Виникають конфлікти між проектною та функціональною структурами, які створюють великі проблеми при прийнятті рішень щодо проекту
Зберігаються всі переваги функціональних структур щодо оптимізації діяльності в функціональних областях і використанні ресурсів для потреб декількох проектів	Виникає необхідність координувати діяльність декількох проектів, наприклад, з таких питань, як розподіл обмежених ресурсів
Істотно знижується занепокоєння персоналу з приводу кар'єри після закінчення проекту	Виникає серйозна проблема розподілу повноважень між керівниками проектів і керівниками функціональних підрозділів
З'являється можливість гнучко «налаштовувати» організаційну структуру в рамках широкого спектра: від слабкої до сильної матриці	Порушується принцип єдиноначальності, що дезорієнтує персонал і спричиняє безліч конфліктів

Матричні організаційні структури ефективно використовуються для досягнення одночасної вертикальної, функціональної спеціалізацій і проектно-цільової (проектної, продуктової, ринкової, географічної та ін.) горизонтальної інтеграції.

У загальному випадку матричні структури використовуються для реалізації проекту в рамках одного підприємства і в разі необхідності керування кількома проектами одночасно на постійній основі.

5.2 Загальний вигляд організаційної структури проекту ІС

Організаційна структура проекту зі створення ІС має бути динамічною, вона повинна пристосовуватися до особливостей різних фаз ЖЦ проекту. Але в той же час вона має забезпечувати чіткий поділ обов'язків між виконавцями.

Єдиних рекомендацій щодо формування ОС проекту не існує. Кращим є формування цілісного колективу фахівців з жорсткою організаційною структурою. Але можлива і така схема, при якій в проекті

постійно задіяні тільки менеджер, архітектор і ключові розробники, а решта фахівців залучаються в міру потреби.

Такий підхід дозволяє економити кошти і значно скорочувати тривалість розроблення. Деяким фірмам удається домогтися 24-годинного робочого дня, залучаючи до розроблення інженерів, які живуть на різних континентах. У цьому випадку взаємодія керівників і ключових розробників з виконавцями здійснюється за допомогою мережі Internet. Однак у цьому випадку нелегко домогтися високої виконавської дисципліни. Можливі й юридичні конфлікти, пов'язані з присвоєнням фахівцями, які працюють вдома, інтелектуальної власності компанії. Тому при будь-якій формі побудови ОС інтеграція готового продукту повинна здійснюватися в офісі компанії.

ОС проекту зі створення ІС показано на рисунку 5.3.

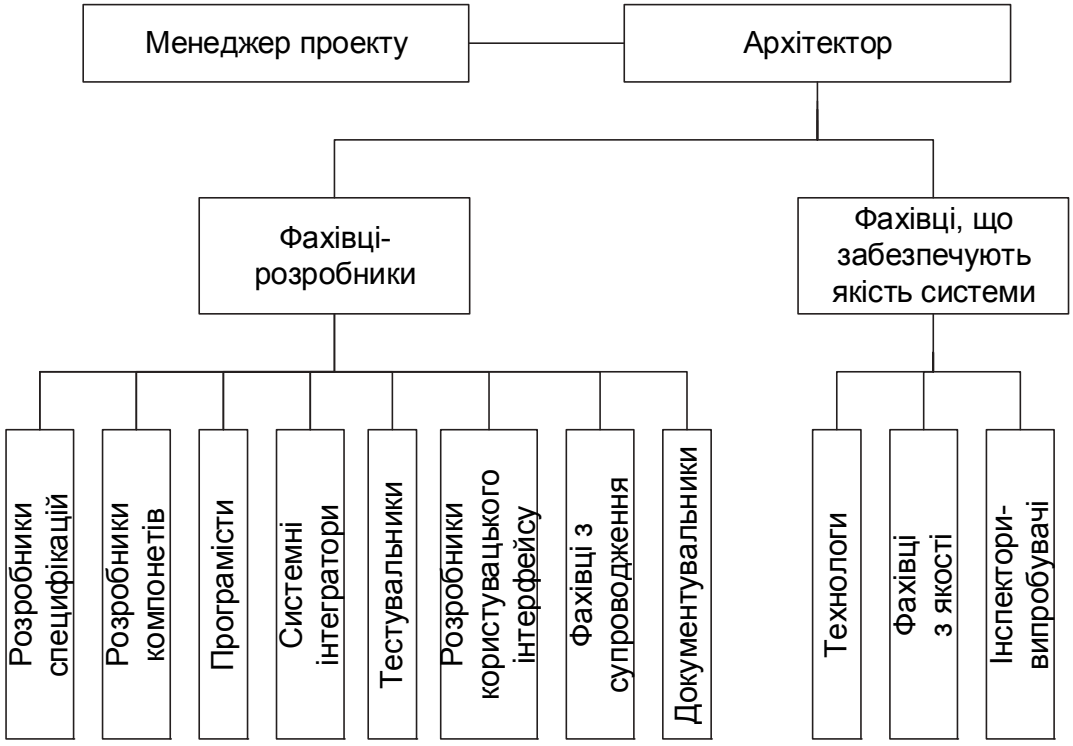


Рисунок 5.3 – Організаційна структура проекту зі створення ІС

Розглянемо основні обов'язки учасників проекту.

Загальне керівництво проектом здійснюють менеджер проекту й архітектор.

Менеджер проекту

Менеджер проекту здійснює загальне організаційне керівництво проектом. До його основних обов'язків належать:

– планування і контроль проекту;
– переговори з замовником;
– розв’язання конфліктів.

Архітектор

Архітектор визначає загальну ідеологію розроблення проекту та здійснює загальне технічне керівництво проектом. До його основних обов'язків належать:

– аналіз і моделювання предметної області;
– розроблення основ архітектури і керівництво роботами з її деталізації;
– координація робіт з проектування та побудови компонентів, інтеграції та випробувань системи;
– оцінювання проміжних і остаточних результатів проекту.

Розробники специфікацій компонентів

Готують специфікації компонентів системи, досить детальні для коректного розроблення початкового програмного коду.

Розробники компонентів і програмісти

Створюють і налагоджують програмні компоненти. У складних системах для реалізації окремих компонентів використовуються різні мови програмування.

Системні інтегратори

Компонують систему з готових компонентів.

Тестувальники (тестери)

Розробляють тести і плани тестування для перевірки правильності інтеграції системи і її функціонування.

Розробники призначеного для користувача інтерфейсу

Розробляють інтерфейс кінцевого користувача. Як правило, графічний користувальницький інтерфейс збирається зі стандартних компонентів за допомогою засобів візуального програмування. Така робота не потребує високої кваліфікації, але необхідна велика увага, оскільки невдалий інтерфейс може зробити програму марною для користувача.

У деяких випадках до розроблення призначеного для користувача інтерфейсу залучаються професійні дизайнери.

Фахівці із супроводу

Здійснюють супровід і модифікацію систем.

Документатори

Забезпечують підготовку технологічної та експлуатаційної документації.

Технологи

Відповідають за придбання інструментарію для розроблення проекту. Проектують технологічний процес розроблення. Навчають фахівців-розробників кваліфікованому застосуванню інструментальних засобів і технологій.

Фахівці з якості

Забезпечують експлуатацію системи якості проекту на основі стандартів і методик фірми. Відповідають за виявлення відхилень від заданих показників якості продукту і процесів і аналіз їх наслідків. Розробляють заходи щодо усунення відхилень.

Інспектори-випробувачі

Проводять випробування програмних продуктів і системи в цілому. Для них обов'язкові знання стандартів і нормативних документів, що регламентують якість ІУС.

5.3 Закон Брукса

Закон, описаний в класичній книзі Ф. Брукса «Міфічний людино-місяць», говорить: «залучення нових працівників не скорочує, а подовжує графік робіт зі створення програмного продукту».

У цього твердження є дуже просте і математично точне пояснення. Жодне обговорення команд розробників не проходить без згадування даного принципу.

Залежність часу, необхідного на проект розроблення, від кількості задіяних виконавців, Брукс встановлює в такий спосіб. Загальна кількість роботи складається з трудовитрат (рисунок 5.4) на:

– неподільні завдання – час виконання цих завдань не залежить від кількості співробітників;
– колективні завдання – можуть бути розділені на довільну кількість підзадач; час виконання зменшується з ростом кількості співробітників в обернено пропорційній залежності;
– частково розділяються завдання, що потребують взаємодії – можуть бути розділені на обмежену кількість незалежних підзадач; додавання співробітників виправдано доти, доки можливий поділ подзадач між ними, подальше додавання співробітників не має ефекту;
– обмін інформацією – Брукс пише буквально таке: «Якщо всі завдання повинні бути окремо скоординовані між собою, то витрати зростають як n (n-2)/2». Мається на увазі, що при наявності n співробітників кількість трудовитрат, вироблених на координацію «усіх з усіма», пропорційна кількості зв'язків у повному графі.

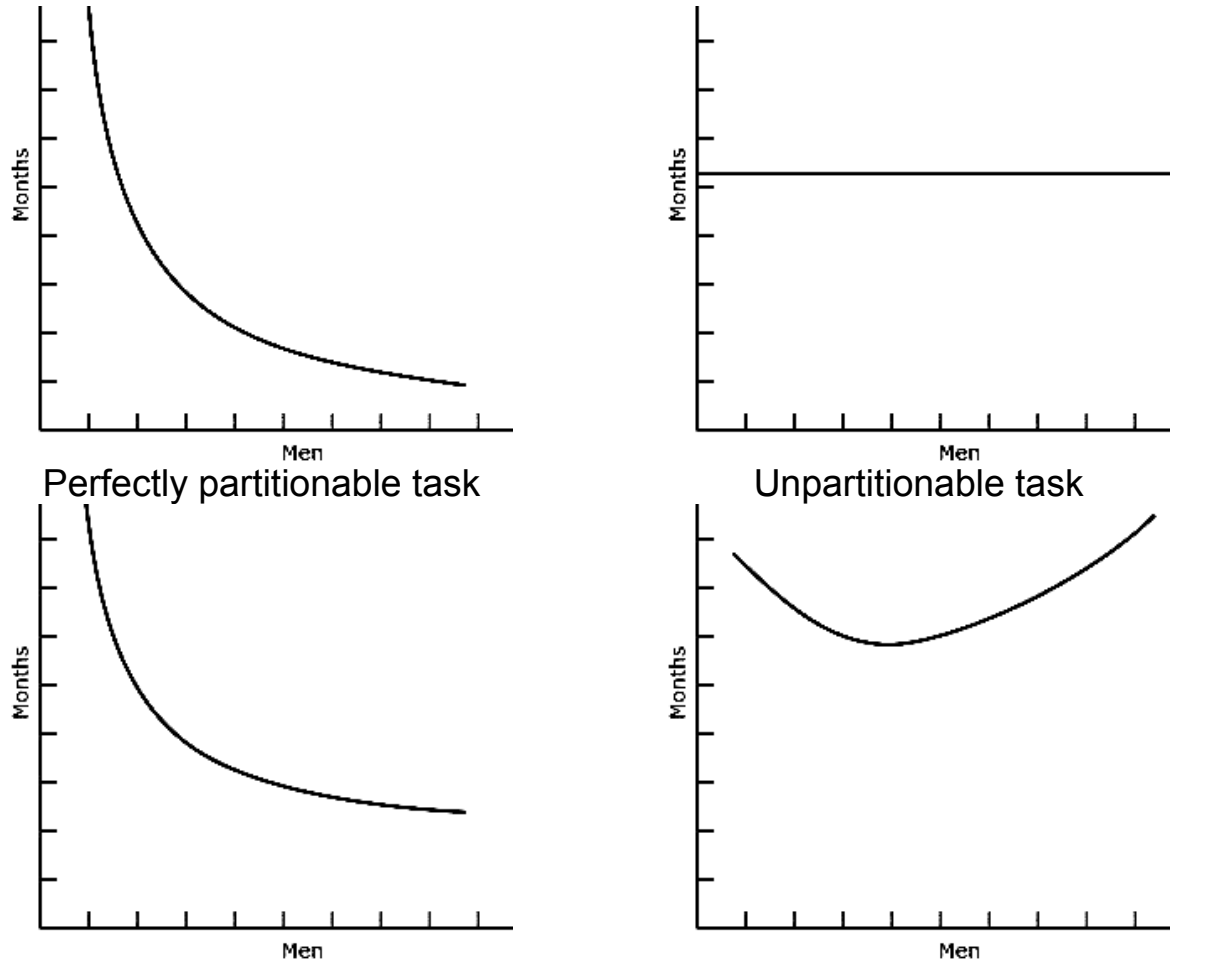


Рисунок 5.4 – Загальна кількість роботи складається з трудовитрат

Якщо колектив розростається, здатність кожного його члена вступати у виразне спілкування з іншим її членом різко падає – безцінний ресурс марнується. Занадто багато пересічних ліній і зустрічних потоків. У таких випадках не є рідкістю, коли група несподівано починає розпадатися – за соціальними або функціональними інтересами – на підгрупи. На робочому місці виникають непорозуміння, взаємне нерозуміння і навіть протилежні цілі. Наради, що раніше тривали кілька хвилин, тепер можуть тривати годинами. Колектив утратив свою універсальність.

У більшості випадків рекомендується обмежити команду розробників 7 – 9 особами. Великі команди не можуть функціонувати ефективно.

Закон Брукса неодноразово піддавався жорсткій критиці. Ерік С. Реймонд в статті «Собор і Базар» критикує класичні технології розроблення (централізоване будівництво собору) і стверджує, що Open Source розроблення (децентралізований демократичний базар) долають обмеження Брукса: «Але якби закон Брукса відбивав всю картину в цілому, проект Linux був би неможливий».

Однак «Собор і Базар» також викликав запеклу критику. Найбільш відома стаття Миколи Безрукова «Повторний погляд на Собор і Базар»: «Насправді всі великі проекти з відкритим кодом мають ієрархічну структуру ... З тих же причин взаємний обмін знаннями також має певні межі в співтоваристві розробників відкритих вихідних кодів ... Сама собою доступність вихідних текстів не тягне за собою автоматичного доступу до найбільш важливих і критичних архітектурних відомостей».

Павло Протасов у статті «Беззаконня Брукса?» наводить ще один аргумент: «Зі збільшенням кількості призначених для користувача програм стає помітною тенденція до скорочення кількості програмістів, які працюють над окремим проектом.

Так, вихідні коди відкриті, так, можна зробити до них патч – але вже мізерно малий відсоток користувачів конкретної програми на це здатний».

6 УПРАВЛІННЯ ВАРТІСТЮ ІТ-ПРОЕКТУ

6.1 Процеси управління вартістю

Управління вартістю проекту об'єднує процеси, що виконуються у ході планування, розроблення бюджету та контролювання витрат, і забезпечує завершення проекту в рамках затвердженого бюджету з використанням процесів з даної галузі знань:

– **Вартісне оцінювання –** визначення приблизної вартості ресурсів, необхідних для виконання операцій проекту.

– **Розроблення бюджету витрат –** підсумовування оцінювання вартості окремих операцій або пакетів робіт і формування базового плану щодо вартості.

– **Управління вартістю –** вплив на фактори, що викликають відхилення за вартістю, і управління змінами бюджету проекту.

6.2 Структура витрат на розроблення ІС

Розроблення ІС – один з найбільш дорогих видів проектів. Однак вартість проекту на початкових етапах ЖЦ оцінити досить складно. Некоректні оцінювання призводять до того, що в 60…70 % випадків реальна вартість ІС виявляється істотно (іноді багаторазово) вище оцінної. Разом з тим планування проекту, складання і узгодження із замовником його бюджету потребують якщо не точного, то хоча б укрупненого оцінювання витрат. Розглянемо основні складові витрат на розроблення ІС і моделі їх визначення.

Витрати на супроводження ІС

Витрати на супровід системи нерідко можна порівняти з витратами на її створення, а іноді вони істотно перевищують їх. Вартість модифікації

системи, що складається з великої кількості взаємодіючих компонентів, може в 2-3 рази перевищити початкові витрати на її розроблення.

Наприклад, витрати на створення системи управління для винищувача F-16 об'ємом 236 KSLOC становлять $ 85 млн, а зусилля із супроводу програм комплексу, їх поліпшення і усунення помилок потребувало додатково $ 250 млн. Тільки в 2000 р. на супровід електронної системи торгів NASDAQ було витрачено $ 55 млн.

Витрати на забезпечення основних функціональних можливостей

До цієї категорії належать такі види витрат:
- витрати на аналіз і проектування архітектури;
- витрати на придбання готових компонентів;
- витрати на створення нових компонентів;
- витрати на придбання інструментальних засобів;
- витрати на придбання апаратних засобів.

Розподіл витрат між цими складовими не завжди однаковий. Використання готових компонентів знижує витрати на програмування, але збільшує витрати на інтеграцію і придбання готового ПЗ. Комп'ютерна техніка коштує дорого, і необхідність створення унікальних технічних засобів може багаторазово збільшити вартість проекту. Але в більшості випадків нові системи працюють з апаратним забезпеченням, що вже є у замовника. Зниження витрат на управління може спричинити зростання витрат на перероблення дефектного ПЗ.

У загальному випадку (але не завжди!) слід дотримуватися досить простої закономірності. Трудомісткість і вартість розроблення ІС пропорційна обсягу (складності) ПЗ, вираженого кількістю рядків вихідного коду SLOC: тому при оцінюванні вартості обсяг програми в SLOC використовується як базовий параметр.

Реальна тривалість і вартість розроблення залежать від кількості фахівців, їх кваліфікації та продуктивності праці, зарплати, а також ряду додаткових чинників.

Розроблення простої програми обсягом 103-104 SLOC може зайняти кілька місяців. Однак за умови забезпечення високого рівня якості з проведенням сертифікаційних випробувань ЖЦ проекту займає не менше півроку. Ці значення можна прийняти за нижню межу складності і тривалості розроблення.

Розроблення складних систем об'ємом 106-107 SLOC займає від 2 до 4 років. При більш тривалому розробленні ПЗ морально старіє і втрачає актуальність. Тому наведені значення можна прийняти за верхню межу складності і тривалості проекту. Більш складні системи необхідно розробляти і впроваджувати за частинами.

Витрати на забезпечення якості

У розробленні ІС, як і в багатьох інших областях, діє правило, відоме як принцип 80/20: 80 % можливостей досягаються після застосування 20 % зусиль, а для того щоб отримати решту 20 % можливостей, необхідно докласти 80 % зусиль. Це правило нерідко розцінюється як жарт, але справедливе майже завжди. Реалізація основних функціональних можливостей ІС, як правило, не потребує значних витрат. Набагато більших витрат і зусиль потребує забезпечення коректної роботи системи, її надійності, продуктивності і т. п., тобто інформації про програму до рівня повноцінного програмного продукту.

Таким чином, необхідність забезпечення високого рівня якості призводить до істотного (іноді у багато разів) подорожчання відносно базового оцінювання вартості проекту.

До основних складових витрат на забезпечення якості ІС належать такі.

Витрати на забезпечення коректності при реалізації вимог

Додають до вартості ІС додатково 20…30 % від вартості основних функціональних можливостей.

Витрати на взаємодію із зовнішнім середовищем

Сюди належать візуалізація інформації для кінцевого користувача, а також кошти взаємодії з операційною системою, з іншими прикладними системами, використання телекомунікаційних технологій тощо.

Ці витрати додають до вартості ІС додатково 10…20 % від вартості основних функціональних можливостей.

Витрати на захист від навмисних і випадкових загроз

Сюди належать усі засоби захисту системи: від елементарної простої, до повноцінного захисту від злому.

Ці витрати додають до вартості ІС додатково від 10…20 % для забезпечення правильної реакції на аномальні дії некваліфікованого користувача до 200…300 % для повноцінного захисту від несанкціонованого доступу.

Витрати на забезпечення високої надійності

Додають до вартості ІС додатково від 10…20 % у звичайних системах до 200…300 % у системах, які повинні працювати без перезавантаження десятки тисяч годин (серверні вузли розподілених ІС, системи управління електростанціями, промисловими об'єктами з безперервним виробничим циклом, системами життєзабезпечення тощо).

Витрати на забезпечення практичності

Сюди входять підготовка інструкцій для користувача, оформлення документації, створення довідкової системи (Help), розроблення електронних підручників, навчальних прикладів тощо. Ці витрати додають до вартості ІС додатково до 20 % від вартості основних функціональних можливостей.

Витрати на випробування системи

До цієї категорії належать такі види витрат:

– витрати на розроблення тестів;
– витрати на створення випробувальних стендів;
– витрати на проведення випробувань.

Для розроблення тестових прикладів необхідно детально проаналізувати поведінку зовнішнього середовища. Витрати на розроблення тестів, що охоплюють 90…95 % усіх можливих варіантів використання, можуть бути порівнянні з витратами на реалізацію основних функціональних можливостей.

Для виконання випробувань нерідко створюються випробувальні стенди. Це може бути фізичний стенд для натурних випробувань, який підключається до комп'ютера за допомогою пристрою сполучення з об'єктом, або програма, що імітує зовнішнє середовище. Подібний імітаційний стенд являє собою самостійний, іноді досить складний програмний продукт обсягом 103-104 SLOC. Самі випробування можуть бути пов'язані з введенням великих обсягів вихідних даних та аналізом великої кількості вихідної інформації.

6.3 Основні моделі для визначення обсягів робіт при розробленні інформаційних систем

Метод функціональних точок

Термінологія

Метод функціональних точок (МФТ) використовує вельми специфічну термінологію. Понятійний апарат програмування призначений для знаходження способів реалізації програм. В області користувальницьких вимог до функціональності реалізація нецікава і несуттєва, тут головне – точне знаходження відповідей на запитання «що програма повинна робити?», тому починати треба зі знайомства з термінами, які в МФТ звучать звично, але їх специфіка і семантика відображують різницю між двома згаданими областями.

Можна навести таблицю (таблиця 6.1), яка містить найбільш важливі терміни та їх визначення в МФТ і в цілому в програмній інженерії [2].

Таблиця 6.1 – Короткий словник ІТ-МФТ

Термін	**Прийняте в ІТ значення**	**Значення МФТ**
Користувач (user)	Фізична особа, яка використовує ПЗ або визначає вимоги до нього	Особа, група осіб, організація, додаток, пристрій і т. д., які тією чи іншою мірою зумовлюють вимоги до функцій, виконуваних ПЗ
Додаток (application)	Фізична реалізація програмного продукту. Межі «додатків» часто чітко визначені суто на апаратному або програмному рівні	Утворює єдиний цілий набір автоматично виконуваних процедур і необхідних даних, що підлягають строго визначеному переліку цілей
Проект (project)	Залежно від організаційних особливостей може містити як процес проектування нового виробу, так і процеси його модифікації, удосконалень і т. д.	Безліч оцінок функціональності і удосконалень в усіх точках життєвого циклу ПЗ
Файл (file)	Набір даних, що існує на фізичному рівні (зазвичай на рівні конкретної файлової системи, наприклад, «вхідний файл»)	Зумовлена користувачем група пов'язаних даних безвідносно їх типу, уявлення і фізичної реалізації

Методика оцінювання функціональності ПЗ на основі вимог користувача.

Загальна мета аналізу за методом функціональних точок – це підрахунок нормованої кількості функціональних точок. Цей процес містить п'ять кроків:

1. Визначення меж ПЗ.
2. Ідентифікація та оцінювання функціональності даних (показники ILF, EIF).
3. Ідентифікація та оцінювання функціональності транзакцій (показники EI, EO, EQ).
4. Визначення нормує значення фактора (VAF).
5. Підрахунок нормованої кількості функціональних точок.

Джерелами інформації для виконання всіх цих кроків можуть служити:

– загальна специфікація на ПЗ (технічне завдання – ТЗ), що містить функціональну специфікацію і специфікацію якості;
– наявна на момент оцінювання документація за інтерфейсами;

– наявна на момент оцінювання документація розробника;
– звіти за іншими метриками ПЗ;
– спілкування з користувачами;
– прототип керівництва користувача;
– результати функціонального моделювання;
– логічна модель даних;
– діаграми потоків даних;
– результати системного аналізу, отримані з використанням різних методик, наприклад, таблиць рішень, мереж Петрі, діаграм UML і т. п.

1. Визначення меж ПЗ.

Межі ПЗ можуть бути встановлені вже на ранніх етапах життєвого циклу продукту. Наприклад, якщо ПЗ розробляється в ході заміщення застарілого проекту, то його межі повинні, очевидно, бути подібними (а, можливо, і повністю збігатися) з межами існуючого ПЗ. Якщо ПЗ є принципово новим, то, щоб правильно визначити його межі, необхідно встановити межі інших ПЗ, що працюють спільно з даним.

Межі для додатків Internet/Intranet визначаються також, як і для звичайного ПЗ. Для них межі формуються не для призначеного користувача інтерфейсу або декількох екранів, а для програми в цілому. Часто Internet/Intranet додаток розробляється як заміна або розширення існуючого ПЗ, і в цьому випадку, очевидно, некоректно розглядати такий Internet/Intranet додаток як принципово нове ПЗ.

Межі для додатків клієнт/сервер повинні формуватися і для клієнта, і для сервера. Причина полягає в тому, що ні клієнт, ні сервер не забезпечують окремо вимог користувача. Тобто окремо вони не є завершеним ПЗ.

2. Ідентифікація та оцінкювання функціональності даних (ILF і EIF).

Загальна функціональність ПЗ є сумою двох складових: функціональності даних і функціональності транзакцій.

Функціональність даних визначається шляхом аналізу логічних груп даних, які використовуються і підтримуються ПЗ.

Функціональність транзакцій визначається шляхом аналізу інформації, що вводиться і виводиться користувачем.

При аналізі за методом функціональних точок розглядаються два види груп даних:

Внутрішній логічний файл (ILF – Internal Logical File) – логічно пов'язана група даних, що визначається користувачем і знаходиться всередині меж ПЗ.

Зовнішній інтерфейсний файл (EIF – External Interface File) – логічно пов'язана група даних, що забезпечує ПЗ інформацією, але лежить за його межами і підтримується іншим ПЗ.

Функціональність логічних файлів (ILF і EIF) оцінюється шляхом підрахунку кількості типів елементів записів (RET) і кількості типів

елементів даних (DET), що входять до відповідних логічних груп даних. При цьому під кількістю RET зазвичай розуміється кількість різних логічних підгруп даних, що виділяються в файлі з точки зору користувача, або кількість різних використовуваних форматів записів, а під кількістю DET – кількість різних елементарних полів у цих записах.

Зазвичай функціональність даних подається файлами, таблицями баз даних, об'єктами та іншими одиницями зберігання інформації.

У цілому завдання оцінювання кількості RET у методі функціональних точок – одне з найскладніших і результат його вирішення достатньо сильно залежить від точки зору оцінювача.

3. Ідентифікація та оцінювання функціональності транзакцій (EI, EO, EQ).

Транзакції – це елементарні процеси, тобто найменші одиниці активності, що мають сенс для користувача, які відбуваються всередині ПЗ і які породжуються вхідною і вихідною інформацією. В аналізі, основаному на методі функціональних точок, виділяють три види транзакцій:

Зовнішнє введення (EI – External Input) – процес уведення даних і керуючої інформації в ПЗ. Керуюча інформація – це така, що необхідна для правильного оброблення даних. Дані, що надходять на вхід ПЗ, використовуються для підтримки внутрішнього логічного файла. Зазвичай процеси виду EI використовуються для додавання, зміни або видалення інформації.

Зовнішній вивід (EO – External Output) – процес, що генерує дані або керуючу інформацію, які надходять на вихід ПЗ. Зазвичай процесом виду EO є формування різних екранів, звітів, повідомлень.

Зовнішній запит (EQ – External Inquiry) – діалогове введення, яке призводить до негайної відповіді ПЗ в формі діалогового виведення. При цьому діалогове введення в самому ПЗ не зберігається, а діалоговий вивід не потребує виконання обчислень. У цьому полягає головна відмінність EQ від EI і EO.

Кожній з виявлених характеристик функціональності ПЗ (EI, EO, EQ, ILF, або EIF) ставиться у відповідність низький, середній або високий рівні складності, а потім присвоюється певна числова оцінка.

Таким чином, ненормована кількість функціональних точок (UFPC – Unadjusted Function Point Count) обчислюється як зважена сума оцінок складності всіх вимог.

Для отримання остаточного результату аналізу, тобто нормованої кількості функціональних точок (AFPC – Adjusted Function Point Count), необхідно також урахувати ряд загальних вимог до проекту, для чого ненормовану кількість функціональних точок множать на спеціальним чином розрахований нормуючий фактор (VAF – Value Adjustment Factor).

4. Визначення значення нормуючого фактора (VAF).

У методі функціональних точок нормуючий фактор (VAF) визначається шляхом аналізу 14 основних характеристик системи (GSC – General System Characteristics), метою яких і є облік загальних вимог до проекту. У методі функціональних точок надано такі основні характеристики системи.

Кожна характеристика системи оцінюється експертним способом числом від 0 до 5.

Значення всіх 14 характеристик підсумовуються для отримання підсумкового ступеня впливу (TDI – Total Degree of Influence). Нормуючий фактор (VAF) розраховується за формулою VAF=0,65+(0,01·TDI).

Таким чином, нормуючий фактор може набувати значеннь від 0,65 до 1,35.

5. Підрахунок нормованої кількості функціональних точок.

Нормована кількість функціональних точок являє собою множення ненормованої кількості функціональних точок на нормуючий фактор *AFPC=UFPC·VAF*.

Надалі нормована кількість функціональних точок може бути використана для отримання оцінювання кількості рядків вихідного коду (SLOC – Source Lines o Code) у ПЗ за допомогою методу зворотного запуску (Backfire Method).

Метод зворотного запуску оснований на використанні так званого «мовного множника», який являє собою середню кількість рядків вихідного коду конкретної алгоритмічної мови, що припадає на одну нормовану функціональну точку. Таким чином, якщо мову реалізації обрано, то можна оцінити кількість рядків вихідного коду розроблюваного ПЗ шляхом множення нормованої кількості функціональних точок на відповідний мовний множник *SLOC=AFPC·LM*, де LM – мовний множник мови програмування.

Вимірювання розміру ПЗ є ключовим фактором для точного оцінювання трудомісткості його розроблення та планування робіт щодо його реалізації.

Модель оцінювання вартості COCOMO

Однією з найпопулярніших, відкритих і добре документованих моделей оцінювання вартості програмного забезпечення (ПЗ) є модель COCOMO II (COnstructive COst MOdel – конструктивна модель вартості) [4]. Модель COCOMO дозволяє оцінити трудомісткість і вартість проекту зі створення ПЗ.

Модель COCOMO II розроблено в 1997 р. на основі моделі COCOMO, що застосовувалася до цього з 1981 р. Обидві моделі розроблені під керівництвом Б. Боема, USC Center for Software Engineering.

Модель COCOMO II розроблено з урахуванням особливостей життєвого циклу (ЖЦ) проекту ПЗ і фактично вона містить три моделі, які

дають оцінки з різним ступенем точності залежно від повноти інформації, наявної на різних фазах ЖЦ:

Модель етапу створення прототипів (Application Composition) дає оцінки низької точності на основі грубих вхідних даних. Її доцільно застосовувати для раннього оцювання високоризикованих проектів.

Модель ранніх етапів розроблення (Early Design) може використовуватися на фазі проектування. Дає оцінки помірної точності. Потребує чіткого розуміння особливостей проекту.

Постархітектурна модель (Post-Architecture) може застосовуватися на фазах побудови і впровадження. Дає високоточні оцінки на основі детального опису проекту.

У цій роботі розглянуто постархітектурну модель. Модель COCOMO II використовує множину параметрів, значення яких отримані шляхом оцінювання (calibration) безлічі проектів і можуть періодично переглядатися.

Оцінювання трудомісткості проекту

Трудомісткість проекту в людино-місяцях (Person Months) визначається за формулою

$$PM = A \cdot E \cdot Size^{B}, \qquad (6.1)$$

де *PM* – обсяг роботи в людино-місяцях (1 людино-місце = 152 людино-години);

A – масштабуючі коефіцієнти;

E – уточнюючий фактор, що характеризує предметну область, персонал, середовище й інструментарій;

Size – розмір кінцевого продукту (обсяг коду), вимірюваний в тисячах рядків вихідного коду (SLOC);

B – показник, що характеризує економію при великих масштабах, притаманну тому процесу, який використовується для створення кінцевого продукту, зокрема, здатність процесу уникати непродуктивних видів діяльності (доробок, бюрократичної тяганини, накладних витрат на взаємодію).

Коефіцієнт *A*1 є постійним і приймається таким:

$$A = 2{,}45.$$

Визначення коефіцієнта масштабування B

Коефіцієнт масштабування *B* характеризує можливість економії при великих масштабах проекту. Чим вище значення цього коефіцієнта, тим більша перевитрата коштів матиме місце. Значення коефіцієнта *B* визначається як комбінація п'яти чинників:

$$B = 1{,}01 + 0{,}01\sum_{i=1}^{5} SF_i . \qquad (6.2)$$

Визначення поправкового коефіцієнта E

Поправковий коефіцієнт *EAF* визначається як добуток 17 коефіцієнтів, що характеризують різні аспекти програмного продукту:

$$E = \prod_{i=1}^{17} AF_i \ . \qquad (6.3)$$

7 УПРАВЛІННЯ РИЗИКАМИ ІТ-ПРОЕКТУ

Ризик проекту – це невизначена подія або умова, яка в разі виникнення має позитивний або негативний вплив щонайменше як на одну з цілей проекту, наприклад, терміни, вартість, зміст або якість.

Управління ризиками проекту містить процеси, пов'язані з плануванням управління ризиками, їх ідентифікацією та аналізом, реагуванням на ризики, моніторингом та управлінням ризиками проекту. Процеси управління ризиками проекту містять таке:

Планування управління ризиками – вибір підходу, планування і виконання операцій з управління ризиками проекту.

Ідентифікація ризиків – визначення того, які ризики можуть вплинути на проект, і документальне оформлення їх характеристик.

Якісний аналіз ризиків – розташування ризиків за ступенем їх пріоритету для подальшого аналізу або оброблення шляхом оцінювання і підсумовування ймовірності їх виникнення та впливу на проект.

Кількісний аналіз ризиків – кількісний аналіз потенційного впливу ідентифікованих ризиків на загальні цілі проекту.

Планування реагування на ризики – розроблення можливих варіантів і дій, що сприяють підвищенню сприятливих можливостей і зниженню загроз для досягнення цілей проекту.

Моніторинг та управління ризиками – відстеження ідентифікованих ризиків, моніторинг залишкових ризиків, ідентифікація нових ризиків, виконання планів реагування на ризики і оцінювання їх ефективності протягом життєвого циклу проекту.

7.1 Ідентифікація ризиків

Ідентифікація ризиків передбачає визначення ризиків, здатних вплинути на проект. Зазвичай виходи ідентифікації ризиків містяться в документі, який можна назвати реєстром ризиків.

У цьому списку є перелік і опис ідентифікованих ризиків, включаючи основні причини їх виникнення та невизначені допущення проекту.

Основні ризики ІТ-проектів: 1. Масштаб проекту. 2. Кадрове забезпечення. 3. Управління ризиками. 4. Нереальні терміни.

5. Фінансування. 6. Організаційна політика. 7. Численні затримки виконання робіт. 8. Несподівані функціональні проломи. 9. Питання взаєморозуміння. 10. Опір нововведенням.

Вибір класифікації ризиків буде залежати від специфіки IT-проекту і професійних переваг менеджера проекту. Обмовимося, що одні й ті ж ризики можуть трохи відрізнятися за змістом для різних видів діяльності і різних типів проектів. Але в основному ризики IT-проекту можна класифікувати так:

Технічні ризики. Практично в будь-якому IT-проекті існують ризики, пов'язані з технікою (відмова і збої в роботі обладнання, помилки в монтажі тощо).

Ризики оцінювання термінів. Для більшості IT-проектів (особливо в проектах щодо розроблення і впровадження програмного забезпечення) характерні помилки в оцінюванні термінів робіт проекту.

Інтеграційні ризики. Інтеграційні ризики в IT-проектах, особливо у великих компаніях, завжди високі, оскільки будь-яке IT-рішення повинно бути інтегровано в існуючу інфраструктуру. Найбільш характерні ризики переходу на нову систему, які містять витрати на зупинку підприємства під час впровадження IT рішень, навчання персоналу тощо.

Ризики неприйняття продукту проекту користувачами. Будь-який проект, у тому числі в IT сфері – це, в першу чергу, зміна технології роботи. Технічна складова будь-якого проекту – безумовно важлива, але не менш важлива організаційна частина.

Комерційні ризики. Це ризики, пов'язані з вибором технології і постачальника. Необхідно оцінити успішність технології на ринку, її актуальність протягом життєвого циклу IT-проекту, доступність необхідного апаратного та програмного забезпечення, його якість, частоту модернізації.

Ризики недотримання технології. Ці ризики виникають у разі, якщо менеджер проекту має одноосібне рішення за ризиками (ідентифікація, аналіз, вибір методу реагування). Чим більше і складніше проект, тим вище цей ризик.

7.2 Якісний аналіз ризиків

Якісний аналіз ризиків містить розстановку пріоритетів для ідентифікованих ризиків, результати чого використовуються згодом. Якісний аналіз ризиків – це зазвичай швидкий і недорогий спосіб установлення пріоритетів у процесі планування реагування на ризики, за необхідності служить основою для проведення кількісного аналізу ризиків. Якісний аналіз ризиків підлягає уточненню протягом усього життєвого циклу проекту і повинен відображати всі зміни, які стосуються ризиків проекту.

Матриця ймовірності і наслідків (рисунок 7.1). Розстановка ризиків за пріоритетом відповідає потенційному ступеню значущості їх наслідків для досягнення цілей проекту.

Організація встановлює поєднання ймовірності і впливу, на підставі яких ступінь ризику визначається як «високий», «середній» або «низький», що своєю чергою визначає значущість для планування реагування на даний ризик. Ці поєднання в процесі планування управління ризиками можуть переглядатися і адаптуватися до конкретного проекту. Відносна шкала наслідків розробляється кожним підприємством самостійно. Шкала містить тільки описові позначення, наприклад, «дуже низький», «низький», «середній», «високий» і «дуже високий», розташовані в порядку зростання максимальної сили впливу ризику згідно з визначенням даної організації.

Те ж саме можна зробити інакше – шляхом присвоєння даних наслідків цифрових значень, які можуть бути лінійними і нелінійними, наприклад, 0,1 – 0,3 – 0,5 – 0,7 – 0,9 або 0,05 – 0,1 – 0,2 – 0,4 – 0,8.

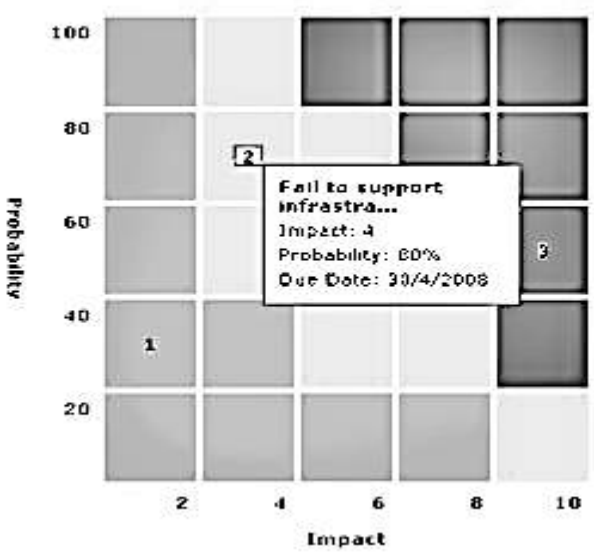


Рисунок 7.1 – Матриця ймовірності і наслідків

7.3 Кількісний аналіз ризиків

Кількісний аналіз проводиться щодо тих ризиків, які в процесі якісного аналізу ризиків були кваліфіковані як ті, що потенційно або істотно впливають на конкурентоспроможні властивості проекту.

У процесі кількісного аналізу ризиків оцінюється ефект від таких ризикових подій і таким ризикам присвоюється цифровий рейтинг.

Аналіз також є кількісним підходом до прийняття рішень в умовах невизначеності.

У ході цього процесу використовуються такі методи, як моделювання Монте-Карло і аналіз дерева рішень; вони використовуються для:

– визначення кількості можливих виходів проекту і ступеня їх вірогідності;
– оцінювання ймовірності досягнення конкретних цілей проекту;
– ідентифікації ризиків, які потребують найбільшої уваги, шляхом кількісного оцінювання їхнього відносного внеску в загальний ризик проекту;
– визначення реалістичних і досяжних цілей за вартістю, розкладом або змістом з урахуванням ризиків проекту;
– визначення кращого рішення з управління проектом у ситуації, коли деякі умови або виходи залишилися невизначеними.

PERT-аналіз. При моделюванні виконання проекту в умовах невизначеності для встановлення тривалості і вартості окремих робіт найбільш часто використовують названі нижче ймовірні ЗР. Для визначення щільності розподілу тривалості роботи, масштабних характеристик: математичного очікування μ і дисперсії D або середньоквадратичного відхилення (СКВ) σ запишемо такі вирази:

β -розподіл:

$$f(t)=\left[\frac{1}{M}\cdot\frac{\frac{1}{M}+1}{B-A}\cdot\left(\frac{1}{M}-1\right)\right](t-B)(t-A)^{\left(\frac{1}{M}-1\right)},$$

де (тут і далі) *A* – нижня оцінка; *B* – верхня оцінка; *M* – найбільш імовірна оцінка.

Масштабні характеристики:

$$\mu=\frac{1}{6}(A+4M+B),$$

$$\sigma=\sqrt{D}=\frac{1}{6}(B-A).$$

При асиметричному розподілі параметри визначають шляхом експертного оцінювання. β-розподіл має місце в разі, коли параметри робіт схильні до дії невеликої кількості випадкових чинників.

Такий розподіл запропоновано використовувати при розгляді методу PERT без строгого теоретичного обгрунтування, а скоріше – як ілюстрацію. Проте багато в чому завдяки авторитету творців методу, β-розподіл набув поширення в практиці УП. Статистичні дослідження підтверджують, що в більшості випадків параметри робіт досить добре узгоджуються з β-розподілом.

Аналіз дерева рішень. Зазвичай структура аналізу дерева рішень будується на основі діаграми дерева рішень (рисунок 7.2), яка описує розглянуту ситуацію з урахуванням кожної з наявних можливостей вибору і можливого сценарію. Вона об'єднує вартість кожної можливості вибору, ймовірність виникнення кожного можливого сценарію, а також винагороди

за кожен альтернативний логічний шлях. Побудова дерева рішень дає можливість провести аналіз очікуваної грошової вартості з кожної альтернативи за умови, що всі винагороди і відповідні рішення вже мають кількісне вираження.

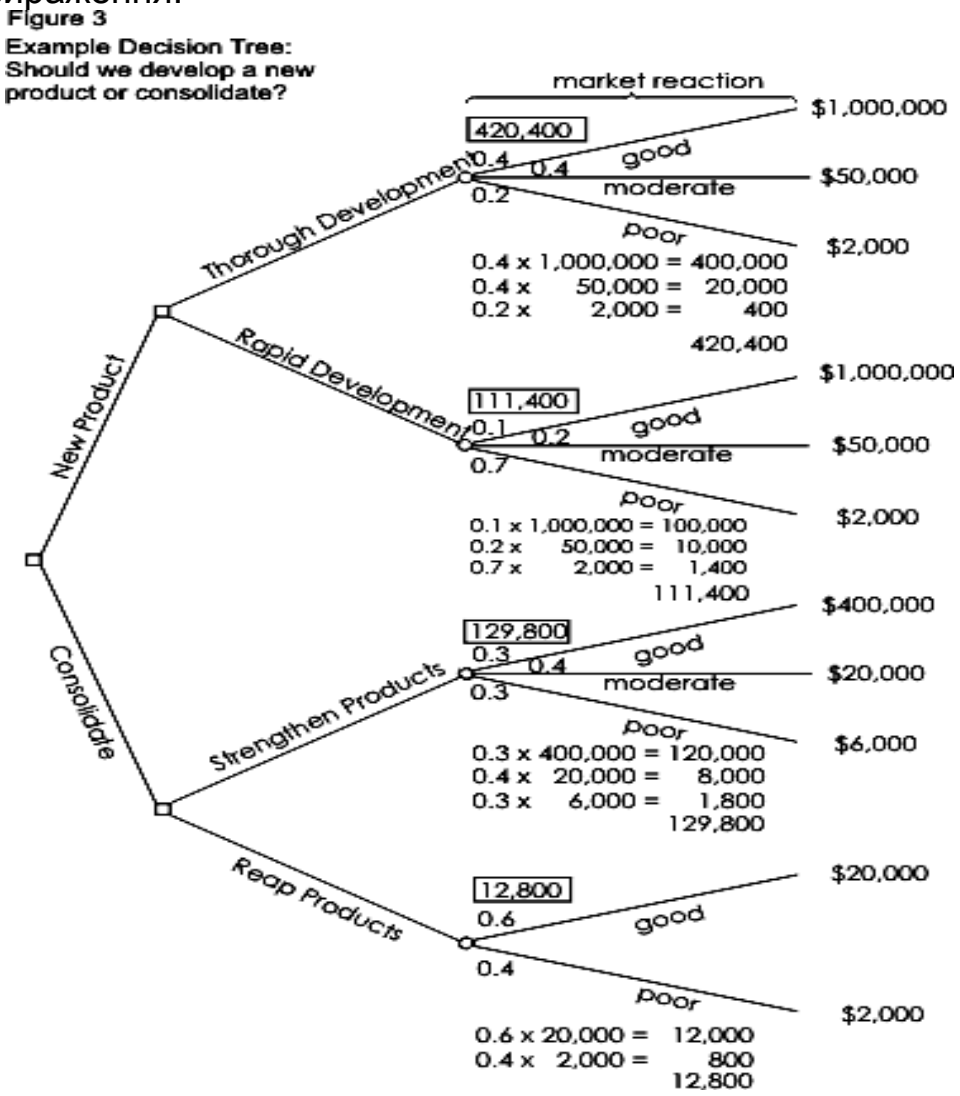


Рисунок 7.2 – Структура аналізу дерева рішень

Моделювання та імітація. При моделюванні проекту використовується модель для визначення наслідків від впливу детально описаних невизначеностей на результати проекту в цілому. Моделювання зазвичай проводиться за допомогою методу Монте-Карло. При моделюванні модель проекту розраховується багато разів (ітеративно), при цьому входи рандомізовані з функції розподілу ймовірності (наприклад, вартість елементів проекту або тривалість планових операцій), обраної для кожної ітерації з розподілу ймовірності кожної змінної. Розраховується розподіл ймовірностей (наприклад, загальна вартість або дата завершення).

Імітаційне моделювання – найбільш універсальний підхід до розрахунку стохастичних мереж. Для імітаційного моделювання стохастичних мереж розроблені методи статистичного моделювання, основані на спільному використанні методів Монте-Карло і методів розрахунку детермінованих мереж. Суть їх полягає в багаторазовому розрахунку детермінованої мережі при різних значеннях випадкових параметрів. Моделювання виконують так (рисунок 7.3):

– відповідно до індивідуальних ЗР розігрують випадкові значення мікрохарактеристик робіт;
– будь-яким методом розрахунку детермінованої мережі визначають макрохарактеристику проекту;
– після багаторазового прогону моделі накопичену інформацію про випадкові значення макрохарактеристик обробляють статистичними методами.

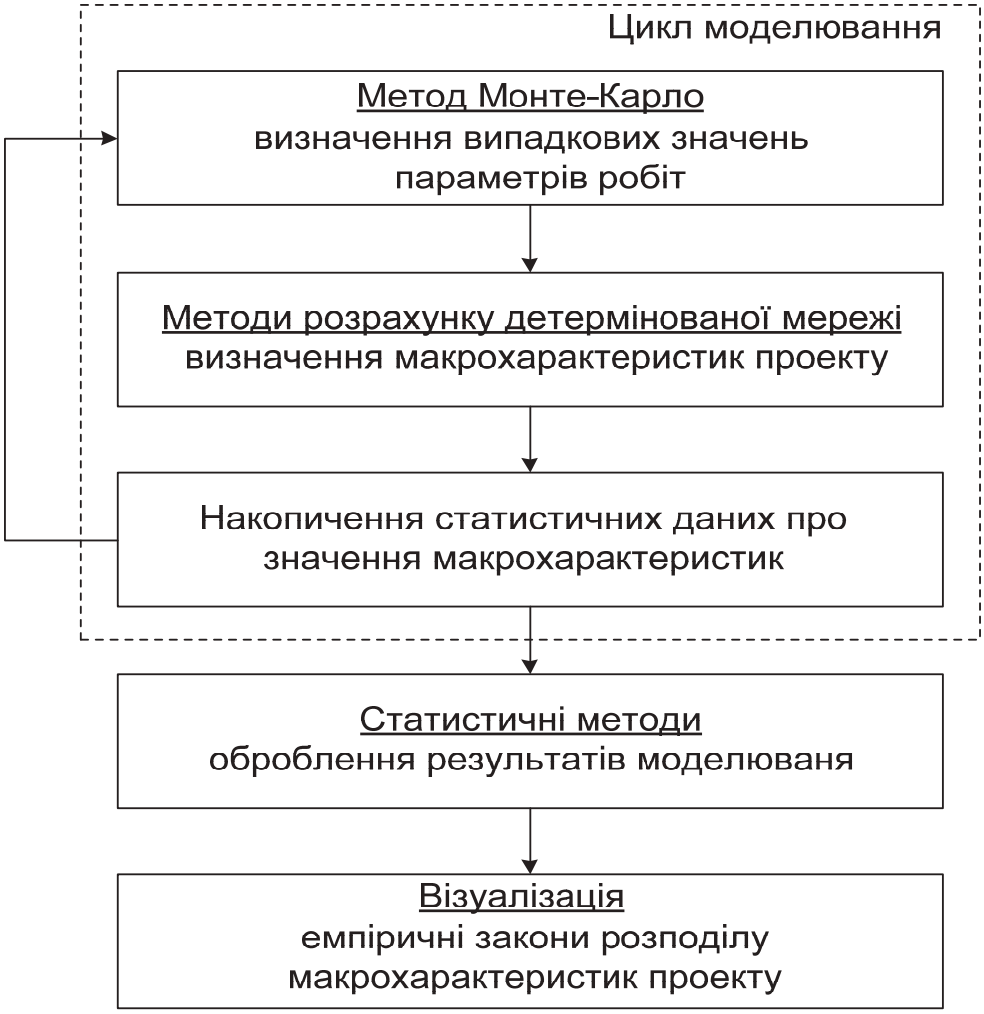


Рисунок 7.3 – Імітаційне моделювання

У результаті моделювання отримують емпіричні ЗР макрохарактеристики проекту в формі гістограм.

Перевага такого виду моделювання – універсальність. Теоретично таким шляхом можна вирішувати будь-які завдання, пов'язані з аналізом і оптимізацією стохастичних мереж.

Недолік полягає у великій трудомісткості. Аналіз стохастичної мережі потребує багаторазового прорахунку детермінованої мережі тієї ж структури, а оптимізація мережі пов'язана з багаторазовим виконанням операцій аналізу.

7.4 Планування реагування на ризики

Причиною виникнення ризику є невизначеність, яка присутня в усіх проектах. Відомі ризики – це ті ризики, які ідентифіковані і піддані аналізу. Щодо таких ризиків можна спланувати відповідні дії за допомогою процесів, описаних в цьому розділі. Але для невідомих ризиків спланувати відповідні дії неможливо. У таких випадках розумним рішенням для команди проекту є виділення загального резерву на непередбачувані обставини, до яких будуть включені ці невідомі ризики, а також усі відомі ризики, для яких розроблення конкретних заходів реагування не є економічно ефективним чи можливим.

Існують три типові стратегії реагування на появу загроз або ризиків, здатних вплинути на досягнення результатів проекту. Такими стратегіями є: ухилення, передача або зниження.

Ухилення. Ухилення від ризику передбачає зміну плану управління проектом таким чином, щоб виключити загрозу, спричинену негативним ризиком, захистити мету проекту від наслідків ризику або послабити мету, що знаходяться під загрозою (наприклад, розширити рамки розкладу або зменшити зміст проекту). Деяких ризиків, що виникають на ранніх стадіях проекту, можна уникнути за допомогою уточнення вимог, отримання інформації, поліпшення комунікації або проведення експертизи.

Передача. Передача ризику це перекладення негативних наслідків загрози з відповідальністю за реагування на ризик на третю сторону. Передача ризику просто переносить відповідальність за його управління іншій стороні; ризик при цьому не усувається. Передача відповідальності за ризик є найбільш ефективною щодо фінансових ризиків. Передача ризику практично завжди передбачає виплату премії за ризик стороні, що бере на себе ризик. Інструменти передачі ризиків численні і різноманітні; вони містять, зокрема, використання страховки, гарантії виконання контракту, гарантійні зобов'язання і т. д. Умови передачі відповідальності за певні ризики третій стороні можуть зазначатися в контракті. У багатьох випадках у контракті з оплатою фактичних витрат витрати на ризики можуть перекладатися на покупця,

Зниження. Зниження ризиків передбачає зниження ймовірності та / або наслідків негативної ризикованої події до прийнятних меж. Вжиття

запобіжних заходів щодо зниження ймовірності настання ризику або його наслідків часто виявляються більш ефективними, ніж зусилля щодо усунення негативних наслідків, що мають місце після настання події ризику. Як приклади заходів щодо зниження ризиків можна навести: впровадження менш складних процесів, проведення більшої кількості випробувань або вибір постачальника, поставки якого мають більш стабільний характер. Для зниження ризиків може знадобитися розроблення прототипу, на основі якого проводиться пропорційне збільшення ймовірності ризику від стендової моделі до процесу або продукту. Якщо неможливо знизити ймовірність, ослаблення ризику повинно бути спрямовано на наслідки ризику, а саме на ті зв'язки, які визначають їх серйозність. Наприклад, розроблення дублюючої підсистеми може скоротити наслідки відмови основної системи.

Ухвалення. Загальна стратегія реагування на загрози і сприятливі можливості. Ця стратегія використовується в тих випадках, коли виключити всі ризики з проекту малоймовірно. Ця стратегія означає, що команда проекту прийняла рішення не змінювати план проекту в зв'язку з ризиком або не знайшла іншої підходящої стратегії реагування на ризики. Ця стратегія може бути застосована або до погроз, або до сприятливих можливостей. Вона може бути або активною, або пасивною. Пасивне прийняття даної стратегії не передбачає проведення будь-яких запобіжних заходів, залишаючи команді проекту право діяти на власний розсуд у разі настання події ризику. Найбільш поширеною формою активного прийняття даної стратегії є створення резерву на непередбачені обставини, які містять час, гроші або ресурси для управління,

Стратегія реагування на непередбачені обставини. Деякі способи реагування призначені для використання тільки в разі виникнення певних подій. Стосовно деяких ризиків команда проекту може задіяти план реагування на ризики, який може бути введений в дію тільки при заздалегідь визначених умовах – якщо є впевненість і достатня кількість ознак того, що даний план буде успішно виконаний. Необхідно визначити і відстежувати події, які призводять до дії механізм реагування на непередбачені обставини, наприклад, відсутність проміжних контрольних подій або привласнення певного постачальника високого рівня пріоритетності.

7.5 Метод критичного ланцюга

Метод критичного ланцюга (Critical Chain Method, CCM) був запропонований Еліяху Голдраттом (Eliyahu Goldratt) у 1997 р. CCM – це метод планування та управління проектами, який звертає більшу увагу на обмеження, пов'язані з ресурсами проекту. Він оснований на методах і алгоритмах теорії обмежень. Цей метод протилежний методам критичного

шляху або PERT у тому сенсі, що він не передбачає жорсткої послідовності завдань і жорсткого планування. Навпаки, календарний план, складений з використанням ССМ, передбачає вирівняне навантаження ресурсів за часом, але потребує від виконавців завдань бути гнучкими щодо часу початку виконання завдань і швидко переключатися між завданнями і ланцюжками завдань (але не працювати над ними одночасно) з метою утримати весь проект у рамках запланованого часу.

Тобто ССМ пропонує сконцентрувати увагу не на досягненні оцінок завдань і проміжних віх, а на досягненні єдино важливої дати – обіцяної дати завершення проекту.

ССМ уводить таке поняття, як критичний ланцюг завдань, або просто критичний ланцюг. Критичний ланцюг – це послідовність завдань, від тривалості яких залежить загальна тривалість усього проекту.

Робота ресурсів над завданням у традиційному менеджменті проектів займає весь відведений час огляду комбінації таких причин: наявність жорстких дат закінчення завдання і «безпечних» оцінок завдання, що містять резерви часу. Для усунення цих проблем ССМ пропонує такі дії:

Створювати календарний план, використовуючи досить щільні оцінки тривалості завдань. Найчастіше в ССМ за тривалість завдання приймається оцінка з 50 %-ним забезпеченням ризику, так звана агресивна оцінка (рисунок 7.4).

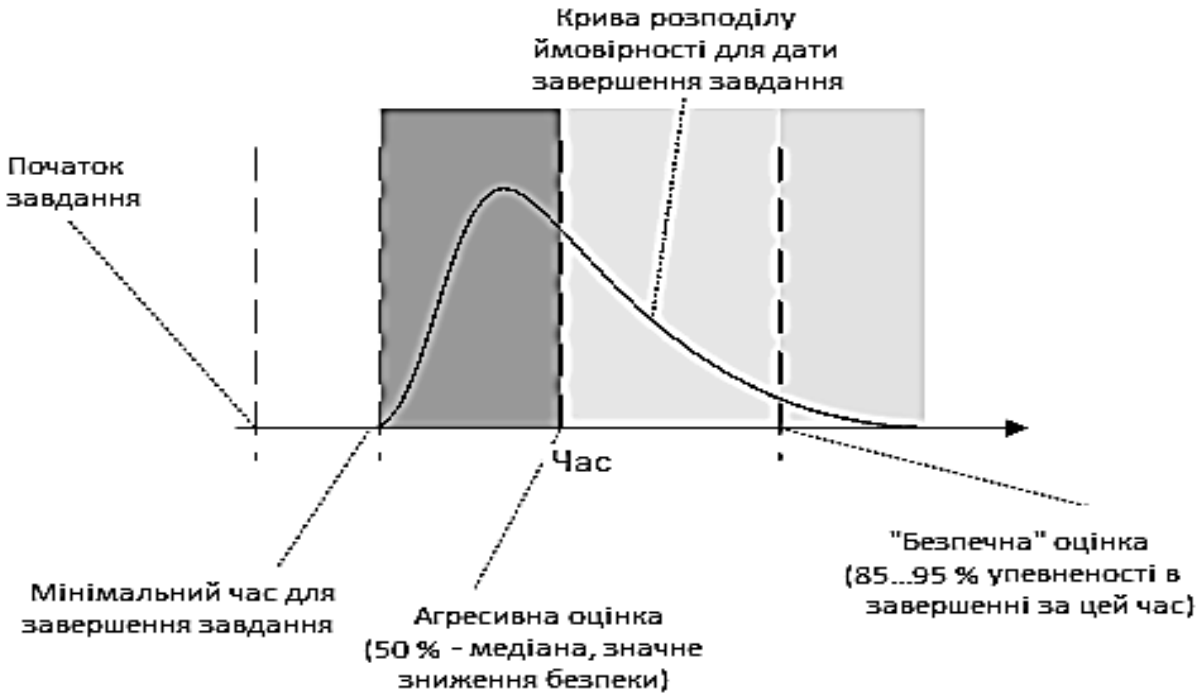


Рисунок 7.4 – Час закінчення завдання як розподіл імовірності

Відмова від жорстких дат закінчення завдання (але не проекту). Безумовно, завдання, як і раніше, оцінюються і мають дату закінчення у календарному плані. Але ця дата не розглядається як зобов'язання виконавців закінчити роботу над завданням саме в зазначений термін.

У матричній організаційній структурі має сенс наділити менеджерів проектів достатньою повнотою влади, щоб вони могли захистити ресурси проекту від «терміновіших» завдань інших проектів або підрозділів.

ССМ умовно ділить ресурси на дві категорії: ресурси, які виконують критичні завдання, і ресурси некритичних завдань. У цьому контексті ті ресурси, про які ми дійсно повинні дбати, – це ресурси критичних завдань.

По-перше, необхідно зібрати інформацію з ресурсів: за скільки потрібно їх попереджати, що вони повинні перервати свою поточну роботу і переключитися на більш важливі завдання критичного ланцюга. По-друге, вимагати, щоб ресурси періодично надавали оцінки про час, необхідний для завершення їх поточних завдань (буфер попередження).

Для того щоб захистити дату закінчення всього проекту від варіацій завдань, ССМ використовує буфери ресурсів і часу. Акумулюємо резервний час усіх завдань ланцюга, який становив від 50 до 90 % покриття невизначеності, залишаючи для самих завдань тільки 50 % покриття. Ці розмазані за всіма завданнями резерви підсумовуються в єдиний буфер часу, який поміщається в кінці ланцюга (рисунок 7.5). Таким чином, варіації в критичному ланцюзі не мають прямого впливу на обіцяну дату закінчення проекту, тому що вони гасяться буфером часу.

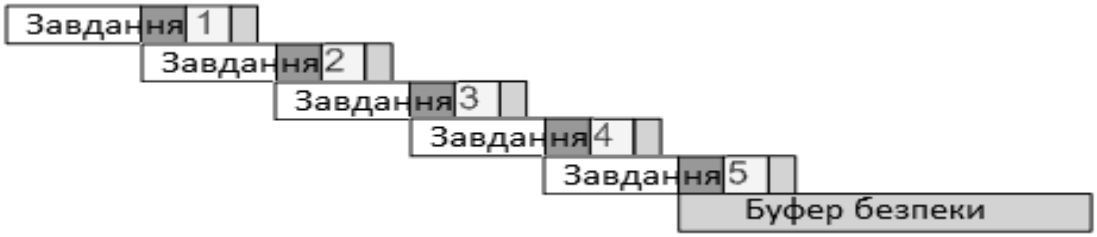


Рисунок 7.5 – Буфер часу в кінці ланцюга

Оскільки ми залишаємо тільки 50 % покриття ризиків в оцінюванні завдання, ми можемо очікувати, що в половині випадків завдання закінчаться раніше запланованих, а в половині – пізніше. У ССМ ми активно використовуємо переваги раннього завершення завдань. Щодо запізнілих завдань, то їх буде компенсувати буфер ланцюга.

ССМ використовується не тільки у вільний час (float) для некритичних задач, але й у випадку з буфером у кінці ланцюга (тепер уже некритичного), який був описаний для критичних завдань (рисунок 7.6). Цей буфер, назвемо його «буфер, що живить», охороняє залежні критичні ланцюги від варіацій за часом у некритичних ланцюгах.

Для некритичного ланцюга ми не використовуємо жодних додаткових інструментів для того, щоб уникнути наслідків затримки виконання завдань.

Особливо цікавим інструментом у ССМ є використання буферів ресурсів. Можна умовно виділити два види таких буферів.

По-перше, той час, за який ми попереджаємо виконавця про те, що скоро у нього виникне завдання з критичної ланцюга, і є ресурсним буфером.

Другий тип буфера – це виділення альтернативних (додаткових) ресурсів для завдань критичного ланцюга. Цей буфер має сенс, коли завдання можуть піддаватися частим змінам. У цьому випадку додавання ресурсів означає захист від ризиків фінальної дати закінчення проектів.

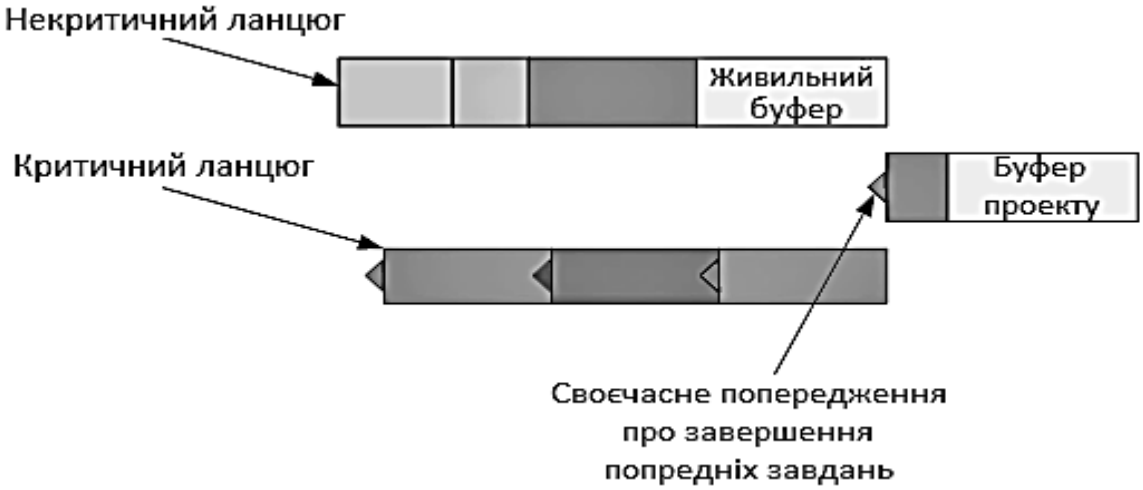


Рисунок 7.6 – Складання розкладу щодо ССМ

На рисунку 7.7 показано критичнй ланцюг, який утворено обмеженими ресурсами. Буфери ресурсів часу в цьому випадку використовуються для управління ризиками, пов'язаними з обмеженими ресурсами. Живильні буфери знаходяться перед завданнями критичних ресурсів, оберігаючи таким чином завдання критичного ланцюга від зсуву за часом у разі затримки завдань некритичних ланцюгів.

Прогрес проекту і точність планування з використанням ССМ часто відстежуються не за класичною технікою аналізу освоєного обсягу (earned value analysis), а за відсотком використаних буферів. Відстеження залишку буферів часу для завдання використовується в ССМ для моніторингу статусу завдання: при досягненні мінімального порогового значення буфера необхідно зробити коригувальні дії. Аналогічно, відсоток використання проектного буфера служить тригером для визначення здійсненності обіцяної дати завершення, так і показником успішності проекту.

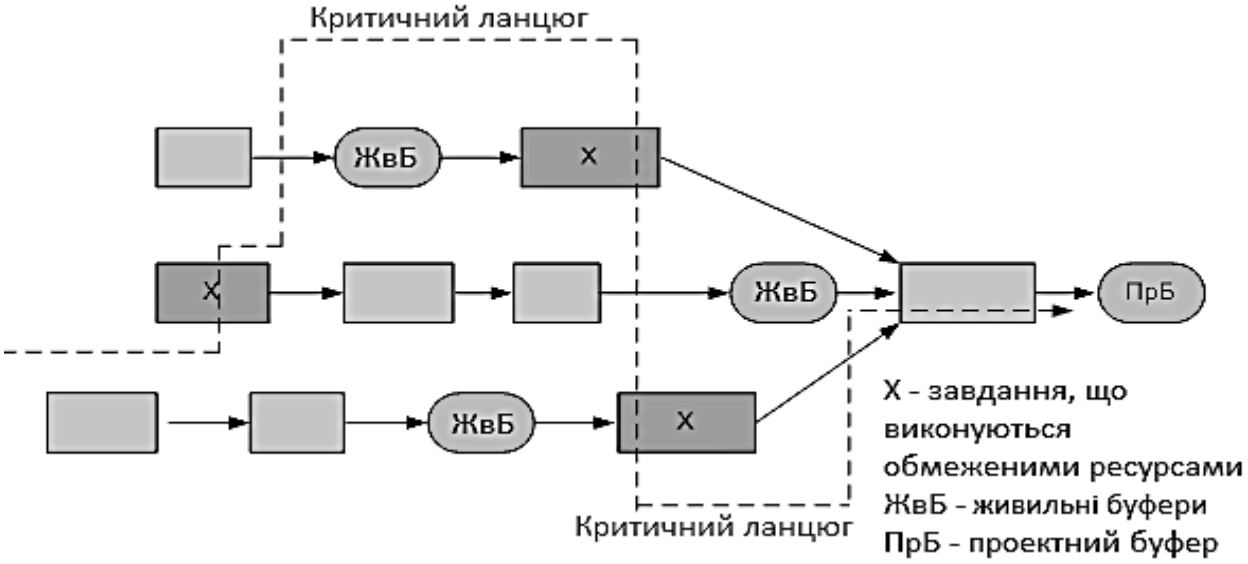


Рисунок 7.7 – Критичний ланцюг

8 КОНТРОЛЬ ВИКОНАННЯ ПРОЕКТУ

Ви не можете контролювати те, що не можете виміряти

Том де Марко

8.1 Моніторинг проекту

Моніторинг – постійне спостереження за будь-яким процесом з метою виявлення його відповідності бажаному результату або первісним припущенням, а також спостереження, оцінювання і прогноз стану навколишнього середовища в зв'язку з господарською діяльністю. Моніторинг – це безперервний зворотний зв'язок з об'єктом для отримання інформації про виконану роботу, оцінювання ефективності проекту. Може містити збір, вимір і поширення інформації про ефективність проекту.

Моніторинг передбачає оцінювання і аналіз ряду метрик, що характеризують як продукт, так і процес його створення.

Метрики програмного забезпечення (англ. Software metric) – штучно введена числова міра, що дозволяє виділити і оцінити певні властивості програмного забезпечення або його специфікацій.

Тенденції зміни метрик у часі дозволяють зрозуміти, в яких напрямках змінюються продукт і процес. Для багатьох метрик важливі не стільки абсолютні значення, скільки напрями зміни.

У цілому можна використовувати такий підхід до вибору метрик для проекту:

– метрики етапів ЖЦ і календарного плану: стежити за графіком робіт за етапами ЖЦ і порівнювати фактичні і заплановані значення;
– метрики витрат за проектами/доданою вартістю: стежити за значеннями кумулятивної величини витрат порівняно з бюджетом, а

також загальною вартістю проекту, постійно оновлюючи дані в міру реалізації проекту;
– метрики відстеження змін у вимогах: кількість змін у вимогах у масштабах проекту;
– метрики процесу розроблення: стежити за кількістю реалізованих у моделі вимог порівняно із загальною кількістю вимог у проекті;
– метрики типів відмов: відстежувати причини відмов ПЗ;
– решта – метрики за дефектами: графічне подання кількості відмов на місяць за місяцями протягом усього часу виконання проекту;
– огляд метрики ефективності: відстежувати щільність помилок за фазами і використовувати діаграми для визначення «піків» і «провалів» на кривій, а також перевищень гранично допустимих значень.

Основні діагностичні метрики

– Бюджет виконаних робіт (плановий і фактичний): постійно змінюється величина (з наростаючим підсумком) сумарної вартості всіх запланованих робіт за проектом, завершених на цей момент.
– Відношення фактичного бюджету до планового бюджету: якщо цей коефіцієнт не перевищує 1, то проект укладається в бюджет у середньостроковій перспективі; якщо ні, то швидше за все бюджет буде перевитрачено.
– Дисперсія витрат: різниця між запланованим бюджетом робіт, які повинні були бути завершені на поточний момент згідно з календарним планом, і фактичним бюджетом, що дозволяє оцінити розмір перевитрат або неосвоєного бюджету в проекті.
– Виконання розкладу: ставлення фактичних трудовитрат на виконання завершених робіт до запланованих на виконання цих робіт трудовитрат (планованої трудомісткості). Цей параметр використовується для оцінювання ризиків можливості недотримання графіка.
– Дисперсія календарного плану: різниця між фактичними затратами для виконаних робіт і плановаными трудозатратами. Це абсолютне вираження того ж параметра, який поданий у вигляді коефіцієнта.
– Відставання від графіка: сумарна тривалість часу затримки задач, які не уклалися в план-графік.
– Коефіцієнт закритих робіт: відношення кількості завершених (закритих) робіт до загальної кількості запланованих до завершення на даний момент робіт.
– Продуктивність праці, що обчислюється як відношення обсягу робіт до витраченого часу (відхилення від запланованих значень).

– Фактичні значення результатів перевірки вимог SLA (Service Level Agreement – угода про рівень обслуговування) або вимог до якості для розроблюваного Вами програмного продукту або сервісу.

Кожна з названих величин вимірюється на регулярній основі. Можна будувати графіки або подавати цифри в табличному вигляді.

8.2 Правило 80/20

Правило 80/20 – також відоме як принцип Парето – говорить, що 20 % укладених зусиль дають 80 % отриманого результату. Відповідно, щоб отримати решту 20 % результату, нам доведеться докласти 80 % зусиль. Універсальний принцип з'явився на початку 1897 року в наукових публікаціях Вільфреда Парето, однак він не став далі розвивати свою думку і лише озвучив деякі приватні закономірності, що стосуються розподілу нерухомості в Італії. Через півстоліття – у 1951 році видатний теоретик менеджменту Джозеф Джуран випустив у світ свою головну наукову публікацію в області якості, яка дотепер не втрачає своєї популярності серед фахівців у різних професійних областях. У своїй науковій праці Джозеф Джуран виявив закономірність розподілу процентного співвідношення 80 до 20 у багатьох галузях, а також у повсякденному житті. Правило 80/20 популярне в бізнес-дослідженнях, продажах, економіці та багатьох інших областях. Правда, співвідношення 80/20 тут є тільки приблизними, а коефіцієнти можуть фактично зміщуватися в будь-яку сторону. Прогрес проекту надзвичайно нелінійний. Завершення проекту, забезпечення належного рівня якості потребують значно більших зусиль, ніж реалізація основного функціоналу (рисунок 8.1).

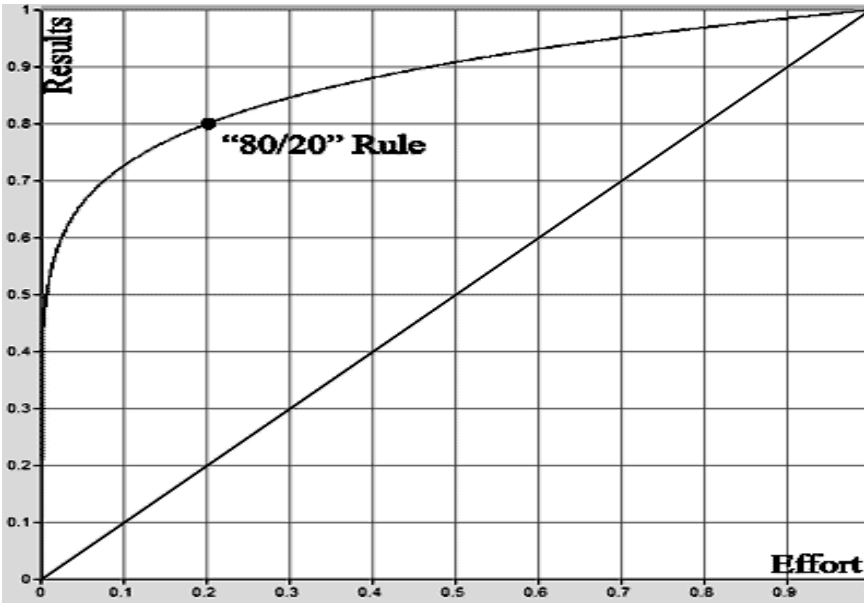


Рисунок 8.1 – Реалізація основного функціоналу

Правило Парето також пояснює «синдром 90 %»: проект готовий на 90 % протягом більшої частини свого життєвого циклу. Насправді, якщо проект готовий на 90 %, для його завершення буде потрібно ще стільки ж часу, скільки вже витрачено.

8.3 Метод освоєного обсягу

Під час відстеження проекту керівнику потрібно вміти визначати, чи вкладається проект у запланований бюджет і чи буде він завершений в заплановані терміни. Для цього мало збирати фактичні дані про хід робіт – потрібно ще й правильно їх аналізувати.

У цьому посібнику Ви ознайомитеся з методикою аналізу ходу проектних робіт і навчитеся використовувати її при відстеженні проекту. Ви дізнаєтеся, які інструменти для застосування цієї методики пропонує MS Project і освоїте їх застосування на практиці. Найцікавіше в цій методиці й інструментах те, що з їх допомогою Ви зможете зрозуміти перспективи Вашого проекту і вжити заходів для досягнення потрібних результатів та уникнення несприятливого розвитку подій.

Проект характеризується обмеженістю в часі й ресурсах і в процесі виконання повинен укластися в запланований бюджет і терміни. Тому під час його відстеження керівнику потрібно вміти визначати динаміку ходу робіт. Однак неможливо визначити, чи вкладається проект в бюджетні і часові рамки, на основі простих даних про фактичні витрати і трудовитрати. Уявімо два проекти з бюджетом 10 000 руб і тривалістю 10 місяців. Після двох місяців роботи керівник першого проекту повідомляє про те, що проект укладається в бюджет: витрачено всього 1500 руб. Керівник же іншого проекту повідомляє про перевищення бюджету: після двох місяців роботи витрачено 2500 руб.

Припустимо, що витрати повинні розподілятися рівномірно у міру виконання проекту, і ми очікували, що після двох місяців роботи буде витрачено 2000 руб. Іншими словами, два місяці становлять 20 % тривалості проекту, 2000 руб – 20 % його бюджету.

Витрати в обох проектах відрізняються від запланованих, але на підставі цих даних можна оцінити стан проекту. На перший проект витрачено на 500 руб менше запланованого, але чи виконано хоча б 20 % запланованих трудовитрат? На другий проект витрачено на 500 руб більше, ніж планувалося, але чи виконано більше 20 % запланованих робіт?

Щоб відповісти на ці запитання, потрібно проаналізувати проектні дані, беручи до уваги дані як про витрати, так і про трудовитрати. Для цього призначений аналіз за методикою освоєного обсягу (earned value analysis). Оскільки аналіз вимірює швидкість витрачання коштів і

виконання роботи, він завжди виконується до певного моменту часу (дати звіту MS Project).

Методика використовує для визначення стану проекту три величини.

Базова вартість запланованих робіт (BCWS, БСЗР) означає зведену вартість робіт, які повинні були бути здійснені до поточного моменту. Іншими словами, параметр означає, які повинні бути витрати на проект на поточний момент за базовим планом.

Фактична вартість виконаних робіт (ACWP, ФСВР) означає зведену фактичну вартість трудовитрат на поточний момент, тобто скільки фактично витрачено на проект до поточного моменту.

Базова вартість виконаних робіт (BCWP, БСВР) означає заплановану за базовим планом вартість фактично виконаних робіт, тобто скільки планувалося витратити на здійснення тих трудовитрат, що були фактично здійснені. Цей параметр часто називається освоєним обсягом.

Кожна з величин визначається в грошових одиницях, і завдяки цьому методика дозволяє аналізувати одночасно дані про витрати і трудовитрати. Назва методики часто перекладається як «придбана вартість», і цей переклад допомагає зрозуміти її суть. Трудовитрати розглядаються як засіб, завдяки якому проект «набуває» вартості (освоює обсяг). Відповідно в кожен момент відомо, яку вартість проект повинен був придбати (BCWS, БСЗР), яку вартість він придбав (BCWP, БСВР) і скільки було витрачено на її придбання (ACWP, ФСВР). Саме тому BCWP (БСВР) часто називається освоєним обсягом (чи набутою вартістю, earned value).

Щоб визначити, наскільки хід робіт відповідає календарному плану, порівнюються BCWP (БСВР) і BCWS (БСЗР). Якщо базова вартість виконаних робіт менше базової вартості запланованих робіт, то хід робіт відстає від розкладу. Якщо ж вартість виконаних робіт перевищує вартість запланованих робіт, то хід робіт випереджає розклад.

Щоб визначити, чи вкладається проект у бюджет, порівнюються BCWP (БСВР) і ACWP (ФСВР). Якщо фактична вартість виконаних робіт більше запланованої в базовому плані, то проект перевищує бюджет. Якщо ж фактична вартість нижче запланованої, то це означає, що кошти витрачаються економно.

Індикатори

Щоб позбавити керівника проекту необхідності порівнювати між собою параметри, віднімаючи від одного інший, при аналізі освоєного обсягу використовують похідні від основних параметрів індикатори, що дозволяють легко визначити, як хід робіт співвідноситься з планом.

Індикаторів методики освоєного обсягу досить багато.

Для визначення співвідношення між виконаними роботами (BCWP, БСВР) і запланованими на поточний момент (BCWS, БСЗР) служить індекс відхилення від календарного плану (SPI, ІОКП). Його значення

визначається шляхом ділення базової вартості виконаних робіт BCWP (БСВР) на базову вартість запланованих робіт BCWS (БСЗР). Якщо значення індексу дорівнює одиниці, це означає, роботи виконуються точно за календарним планом. Якщо значення перевищує одиницю, значить, хід робіт випереджає календарний план, а якщо воно менше одиниці – значить, роботи виконуються з відставанням.

Тепер повернемося до індикаторів, що використовуються в таблицях Earned Value (Освоєний обсяг) і Earned Value Cost Indicators (Показники витрат (освоєний обсяг)). Вони служать для визначення перспектив проекту при збереженні поточного ходу робіт.

Планові зведені витрати на проект (або завдання) позначаються індикатором бюджету після завершення (BAC, БПЗ). Його значення відповідає витратам, запланованим на проект (завдання) у базовому плані (поле Baseline Cost, Базові витрати). У нашому проекті бюджет завдання дорівнює 100 руб (100 годин роботи вартістю 1 руб/годину).

Коли фактичний хід робіт за проектом відхиляється від запланованого, зведені витрати за проектом також відхиляються від планових. Для визначення зведених витрат на проект при збереженні поточного темпу робіт служить індикатор попереднього оцінювання щодо завершення. Значення цього індикатора визначається складанням фактичної вартості виконаних робіт (ACWP, ФСВР) і вартості робіт, що залишилися. Ця вартість визначається відніманням запланованої вартості виконаних робіт (BCWP, БСВР) з бюджету щодо завершення (BAC, БПЗ) і діленням результату на індекс відхилення вартості (CPI, ІОС). Наприклад, в нашому проекті при збереженні темпів економії бюджету на 20 % ми витратимо на виконання завдання не 100 руб, а тільки 80 руб.

Різниця між бюджетом після завершення (BAC, БПЗ) і попереднім оцінюванням після завершення (EAC, ПОПЗ) позначена як відхилення після завершення (VAC, ОПЗ). Наприклад, у нашому випадку відхилення становить 20 руб, оскільки при збереженні поточних темпів робіт ми витратимо на 20 руб. менше, ніж планувалося. Нульове значення цього індикатора означає відповідність бюджету плану при збереженні поточного темпу робіт. Якщо ж індикатор набуває негативного значення, отже, витрати на проект можуть перевищити заплановані.

За допомогою останнього індикатора – показника ефективності виконання (TCPI, ПЕВ) – можна визначити співвідношення між рештою обсягу робіт і бюджетом, що залишився. Індикатор обчислюється шляхом ділення результату віднімання придбаної вартості (BCWP, БСВР) з бюджету після завершення (BAC, БПЗ) на результат віднімання фактичної вартості виконаних робіт (ACWP, ФСВР) з бюджету після завершення (BAC, БПЗ).

Якщо значення індикатора дорівнює одиниці, це означає, що проект виконується точно за планом і робота, що залишилася, буде виконана в рамках бюджету.

Якщо значення індикатора більше одиниці, значить, обсяг роботи, що залишилася, перевищує бюджет і потрібно або збільшити його, або працювати з більшою ефективністю. Якщо ж значення індикатора менше одиниці, значить, у проекту є запас бюджету і можна поліпшити якість роботи, реалізувати додаткові завдання і т. п. У нашому проекті значення індикатора 0,99, тобто у проекту є невеликий запас бюджету.

Для зручності всі відомості про індикатори, включаючи формули розрахунку і трактування значень, зібрано в таблицю 8.1.

Таблиця 8.1 – Індикатори методики освоєного обсягу

Назва	**Формула обчислення**	**Значення**	**Трактування**
Відхилення від календарного плану SV, ОКП	SV = BCWP-BCWS ОКП = БСВР-БСЗР	<0	Відставання від плану
		= 0	Виконання в строк
		> 0	Випередження плану
Відхилення за вартістю CV, ОПС	CV = BCWP-ACWP ОПС = БСВР-ФСВР	<0	Перевищення витрат
		= 0	Витрати за планом
		> 0	Економія коштів
Відносне відхилення за СТО і ОСТІ CV %, ОНПС	CV% = [(BCWP--ACWP)/BCWP] x 100 ОНПС = [(БСВР--ФСВР)/БСВР] x 100	<0	Перевищення витрат
		= 0	Витрати за планом
		> 0	Економія коштів
Індекс відхилення вартості CPI і ОС	CPI = BCWP/ACWP ІОС = БСВР/ФСВР	<1	Перевищення витрат
		= 1	Витрати за планом
		> 1	Економія коштів
Відносне відхилення від календарного плану SV %, ООКП	SV% = (SV/BCWS) x xIOO ООКП = (ОКП/БСЗР) x x100	<0	Відставання від плану
		= 0	Виконання в строк
		> 0	Випередження плану
Індекс відхилення від календарного плану SPI, ІОКП	SPI = BCWP/BCWS ІОКП = БСВР/БСЗР	<1	Відставання від плану
		= 1	Виконання в строк
		> 1	Випередження плану
Попередня оцінка після завершення EAC, ПОПЗ	EAC = ACWP+(BAC--BCWP)/CPI ПОПЗ = ФСВР+(БПЗ--БСВР)/ІОС	<	Економія коштів
		BAC (БПЗ)	
		=	Витрати за планом
		BAC (БПЗ)	
		<BAC (БПЗ)	Перевищення витрат
Відхилення після завершення VAC, ОПЗ	VAC = BAC-EAC ОПЗ = БПЗ-ПОПЗ	<0	Перевищення витрат
		= 0	Витрати за планом
		> 0	Економія коштів

Продовження таблиці 8.1

Назва	Формула обчислення	Значення	Трактування
Показник ефективності виконання TCPI, ПЕВ	TCPI = (BAC--BCWP)/(BAC-ACWP) TCPI = (BAC--BCWP)/(BAC – ACWP)	<1	Засоби економляться, можна підвищити якість робіт
		= 1	Хід робіт відповідає плану
		> 1	Можливе перевищення витрат, потрібно підвищити ефективність

Графічний сенс освоєного обсягу

Припустимо, що якийсь проект відстає від графіка за термінами, а фактичні витрати на виконання робіт при цьому перевищують планові. Тоді, вибравши деяку звітну дату на графіку і побудувавши криву освоєного обсягу і фактичних витрат (також накопиченим підсумком), побачимо таку картину (рисунок 8.2).

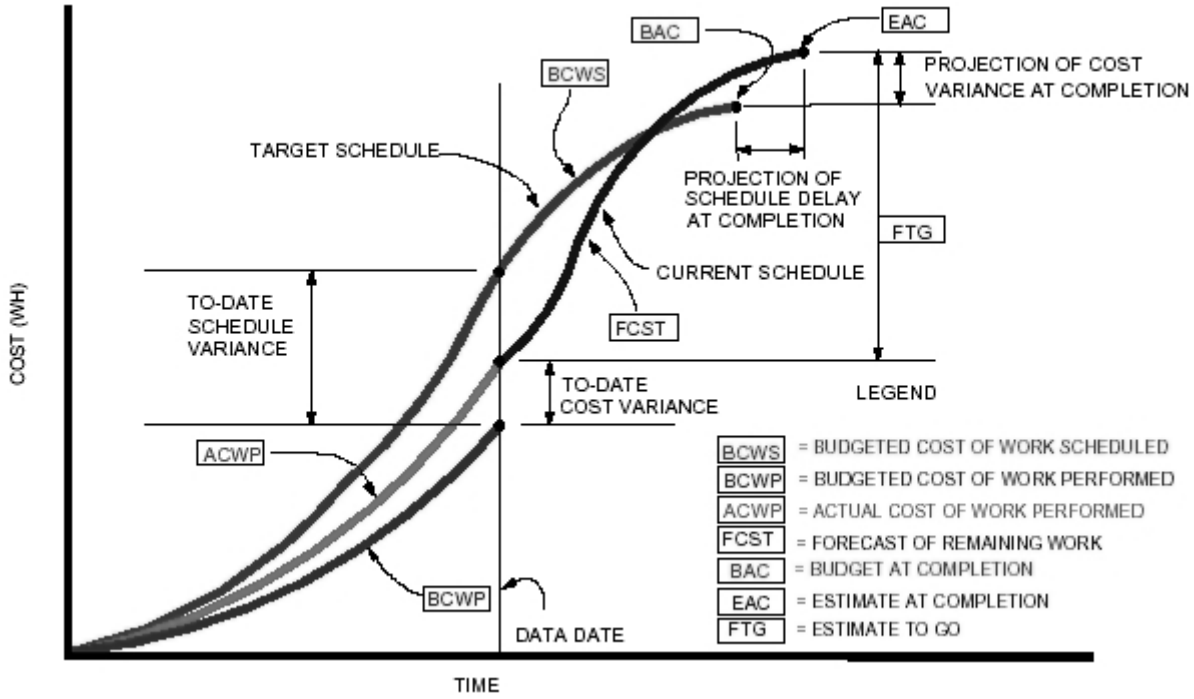


Рисунок 8.2 – Графічний сенс освоєного обсягу

Як видно з графіка, криві планового і освоєного обсягів, а також фактичних витрат наочно відображують стан проекту на звітну дату з точки зору недоосвоєння бюджету і перевитрати коштів. Що ми бачимо? Що в певний момент часу планова вартість виконаних робіт «не дотягує» до необхідного рівня, при цьому фактичні витрати, понесені за проектом, уже значно перевищують планові.

Зауважимо, що вже знайомі нам показники відхилення за термінами і вартістю також видно на цьому графіку особливо наочно. Більш того, до

наочності грошових показників відхилень додалося ще й відображення тимчасової затримки проекту.

9 СТАНДАРТИ ТА МЕТОДОЛОГІЧНІ ПІДХОДИ ДО УПРАВЛІННЯ ІТ-ПРОЕКТОМ

Моделі або методології процесів розроблення ІТ-проектів прийнято класифікуваті за «вагою» – кількістю формалізованіх процесів (більшість процесів або тільки основні) і детальної їх регламентації. Чим більше процесів документовано, чим детальніше описані, тим більше «вага» моделі. SW-CMM. У середині 80-х років минулого сторіччя Міністерство оборони США задумалося про те, як вибирати розробників ПЗ при реалізації великомасштабних програмних проектів. За замовленням військових інститут програмної інженерії, що входив до складу Університету Карнегі–Меллона, розробив SW-CMM, Capability Maturity Model for Software [9] як еталонну модель організації розроблення програмного забезпечення. Ця модель визначає п'ять рівнів процессу розроблення ПЗ.

1. Початковий – процес розроблення має хаотичний характер. Він лише зосереджується на формуванні відрядження конкретних виконавців.

2. Повторювання – встановлені основні процеси управління проектами: відстежування витрат, термінів і функціональності. упорядкування цих процесів і патентів для того, щоб повторити попередні досягнення на аналогічніх проектах.

3. Визначення – процеси розроблення ПЗ і управління проектами, описані й упроваджені в єдину систему процесів компанії. В усіх проектах використовується стандартний для організації процес розроблення і підтримки програмного забезпечення, адаптованості під конкретний проект.

4. Керування – збирання детальних кількісних даних з функціонування процесів розроблення й якості кінцевого продукту. Аналізуються значення і динаміка цих даних.

5. Оптимізація – постійне поліпшення процесів грунтується на кількісних даних за процесами і на пробному впровадженні нових ідей і технологій.

Документація з повним описом SW-CMM займає близько 500 сторінок і налічує 312 вимог, яким має відповідати організація, якщо вона планує атестуватися за цим стандартом на 5-й рівень зрілості.

9.1 Міжнародний стандарт ISO 12207: 1995

Стандарт визначає ряд процесів, причому досить великих. Фази ЖЦ не виділяються, не визначається також послідовність процесів. Процеси діляться на три великі групи.

1. Основні процеси:
– процес придбання визначає дії покупця;
– процес поставки визначає дії постачальника;
– процес розроблення визначає підприємство-розробника програмного продукту;
– процес функціонування визначає дії підприємство-оператора, який забезпечує обслуговування системи в процесі функціонування;
– процес супроводу визначає дії персоналу.

2. Допоміжні процеси:
– процес вирішення проблем;
– процес документування;
– процес управління конфігурацією;
– процес забезпечення якості;
– процес верифікації;
– процес атестації;
– процес спільної оцінки;
– процес аудиту.

3. Організаційні процеси:
– процес управління;
– процес створення інфраструктури;
– процес удосконалення (інших процесів);
– процес навчання.

Допоміжні та організаційні процеси є невід'ємною частиною ЖЦ і не повинні розглядатися як необов'язкові. Стандарт ISO 12207 визначає лише основні принципи розроблення ІУС, що не деталізує їх і не пропонує конкретних методик. Це забезпечує високу адаптивність стандарту до різних видів проектів, умов різних галузей, методів, що застосовується в різних фірмах.

9.2 Методологія Oracle CDM/PJM

Методологія розроблення систем для замовлення Custom Development Methodology (CDM) і розроблення проектів Project Development Methodology (PJM) запропонована фірмою Oracle, призначена для розроблення систем, що активно взаємодіють з великою базою даних.

Модель ЖЦ – каскадна, процесно-орієнтрована. Методологія Oracle CDM/PJM передбачає широке використання засобів розроблення фірми Oracle, тому її доцільно застосовувати в тих випадках, коли проект передбачає широке використання продуктів даної фірми.

У Oracle CDM/PJM виділяється ряд процесів (рисунок 9.1).

Пропонуються також два спрощених підходи. Швидкий підхід (CDM Fast Track), призначений для проектів середнього рівня складності, містить

три фази ЖЦ. Полегшений підхід (CDM Light), призначений для невеликих проектів, містить усього дві фази ЖЦ.

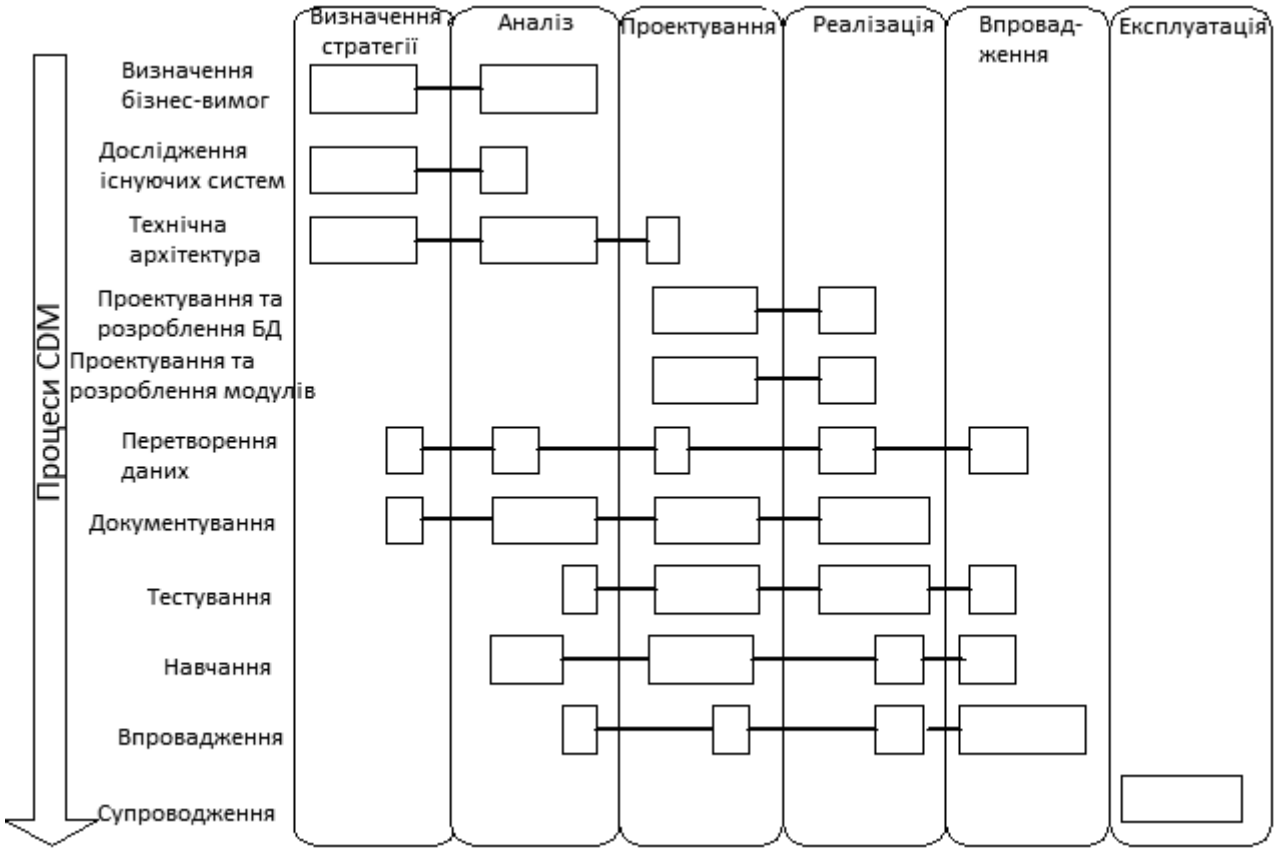


Рисунок 9.1 – Методологія Oracle Custom Development Method (CDM). Класичний підхід до організації життєвого циклу проекту

9.3 Rational Unified Process

RUP. Уніфікований процес (Rational Unified Process, RUP) був розроблений Пилипом Крачтеном (Philippe Kruchten), Іваром Якобсоном (Ivar Jacobson) та іншими співробітниками компанії «Rational Software» як доповнення до мови моделювання UML. Модель RUP описує абстрактний загальний процес, на основі якого організація або проектна команда повинна створити конкретний спеціалізований процес, орієнтований на її споживання (рисунок 9.2).

Саме межа RUP викликає основнову критику – оскільки вона може бути чим завгодно, її не можна вважаті нічиїм визначенням.

У результаті такої загальної побудови RUP можна використовувати і як основу для самого традіційного стилю водоспаду розробки, так і гнучкого процесу.

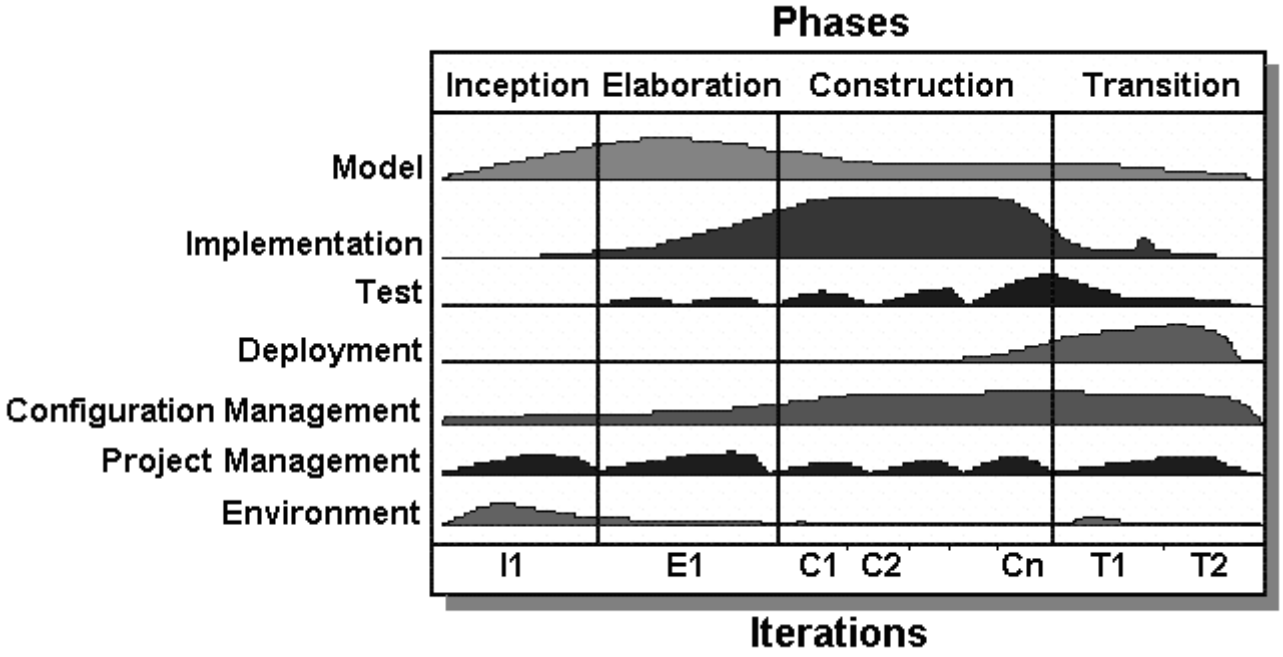


Рисунок 9.2 – Уніфікованій процес RUP

9.4 Microsoft Solutions Framework

MSF. Microsoft Solutions Framework (MSF) (рисунок 9.3) – це гнучка і достатньо легка модель, побудована на основі ітеративного розроблення. Привабливою особливістю MSF є велика увага до створення ефективної і небюрократизованої проектної команди. Для досягнення цієї мети MSF пропонує достатньо нестандартні підходи до організаційної структури розподілу відповідальності і принципів взаємодії усередині команди. PSP/TSP – одна з останніх розробок інституту програмної інженерії

Personal Software Process/Team Software Process – визначає вимоги до компетенції розробника.

Згідно з цією моделлю КОЖЕН програміст повинен уміти: враховувати час, витрачений на роботу над проектом; враховувати знайдені дефекти; класифікувати типи дефектив; оцінювати розмір завдання; здійснювати систематичний підхід до опису результатів тестування; планувати програмні завдання; розподіляти їх за часом і складати графік роботи; виконувати індівідуальну перевірку проекту і архітектури; здійснювати індівідуальну перевірку коду; виконувати регресійне тестування.

Team Software Process робить ставку на самокеровані команди кількістю 3 – 20 розробників. Команди мають: установити власні цілі; скласти свій процес і плани; відстежувати роботу; підтримувати мотивацію і максимальну продуктивність.

Послідовне застосування моделі PSP/TSP дозволяє зробити нормою в організації п'ятий рівень CMM.

Рисунок 9.3 – Microsoft Solutions Framework (MSF)

9.5 Підходи та методи Agile

Основна ідея усіх гнучких моделей полягає в тому, що вживаний в розробленні ПЗ процес повинен бути адаптивним. По суті так звані гнучкі методології – це не методології, а набір практик, які можуть дозволити (а можуть і не дозволити) добиватися ефективного розроблення ПЗ, грунтуючись на ітеративності інкрементальності, самокерованості команди й адаптивності процессу.

Вибір моделі – процес важкий і легкий, моделі виробничого процесу мають свої переваги і свої недоліки. Ефективність сильно залежить від індивідуальніх здібностей, потребує більш кваліфікованої універсальної і стабільної команди. Обсяг і складність виконуваних проектів обмежені. Ті, хто намагається слідувати моделям, які описано в книгах, не аналізуючи їх, застосовують у конкретній ситуації, не враховуючи свідчення і протипоказання, уподібнюються послідовникам культу «Карго».

Алістер Коуберн, один з авторів «Маніфесту гнучкого розроблення ПЗ» проаналізував дуже різні програмні проекти, які виконувалися за різними моделям від абсолютно полегшених і «гнучких» до важких (CMM-5) за останні 20 років. Він не виявив кореляції між успіхом або провалом проектів і моделями процесу розроблення, які застосовувалися у проектах. Звідси він зробив висновок про те, що ефективність розроблення ПЗ не залежить від моделі процесу, а також про те, що:

– у кожного проекту має бути своя модель процесу розроблення;
– у кожної моделі має бути свій час.

Це означає, що не існує єдиного правильного процесу розроблення ПЗ, у кожному новому проекті процес повинен визначатися щораз наново, залежно від проекту, продукту і персоналу, відповідно до «Закону 4 П»: Проект, Продукт, Персонал, Процес.

Абсолютно різні процеси мають застосовуватися у проектах, у яких беруть участь п'ять осіб, і в проектах, у яких беруть участь 500 осіб. Якщо продуктом проекту є критичне ПЗ, наприклад, система управління атомною електростанцією, то процес розроблення повинен сильно відрізнятися від розроблення, наприклад, сайту «відпочинь.ру». І, нарешті, по-різному слід організовувати процес розроблення в команді вчорашніх студентів і в команді професіоналів.

10 AGILE-МЕТОДОЛОГІЇ

Agile – гнучка методологія розроблення та управління проектами – застосовується і добре працює в невеликих компаніях.

Доцільно використовувати Agile, коли виконується щось нове, у крайньому разі – нове для конкретної команди розроблення. Якщо команда робить те, що вона вже робила неодноразово, вона, ймовірно, не потребує гнучкого підходу.

Суть agile-підходу викладено в «маніфесті», але для замовника її можна коротко сформулювати так:

– розроблення ведеться короткими циклами (ітераціями) тривалістю 1–4 тижні;
– у кінці кожної ітерації замовник отримує цінний для нього додаток (або його частинау), який можна використовувати в бізнесі;
– команда розроблення співпрацює із замовником у ході всього проекту;
– зміни в проекті вітаються і їх швидко включають в роботу.

Історія виникнення цього підходу стала відповіддю на запити галузі:

1. Замовник не може сформувати чіткі вимоги до ПЗ.

2. Нові технології посилили конкуренцію і потребували оперативного застосування в бізнесі; з появою веб-додатків упровадження нової функціональності стало відбуватися швидше: потрібно розгорнути додаток тільки на сервері – і всі користувачі отримують доступ до нього. Ця інновація серйозно посилила конкуренцію між компаніями.

3. Замовники і розробники ПЗ не задоволені процесом взаємодії.

Історія Agile починається з публікації в 2001 році «Маніфесту гнучкого розроблення ПЗ», що складається з 12 принципів.

10.1 Agile-маніфест розроблення програмного забезпечення

Ми постійно відкриваємо для себе більш досконалі методи розроблення програмного забезпечення, займаючись розробленням безпосередньо і допомагаючи в цьому іншим. Завдяки виконаній роботі ми змогли усвідомити, що:

люди і взаємодія важливіші за процеси та інструменти;
працюючий продукт важливіший за вичерпану документацію;
співпраця з замовником важливіша за узгодження умов контракту;
готовність до змін важливіша за проходження щодо попереднього плану.

Тобто, не заперечуючи важливості того, що те, що знаходиться справа, ми все більше цінуємо ніж те, що знаходиться зліва.

Основоположні принципи Agile-маніфесту

Ми слідуємо таким принципам:

– Найвищим пріоритетом для нас є задоволення потреб замовника завдяки регулярному і ранньому постачанню коштовного програмного забезпечення.
– Зміна вимог вітається, навіть на пізніх стадіях розроблення.
– Agile-процеси дозволяють використовувати зміни для забезпечення замовнику конкурентної переваги.
– Працюючий продукт слід випускати якомога частіше – з періодичністю від кількох тижнів до кількох місяців.
– Протягом усього проекту розробники і представники бізнесу мають щодня працювати разом.
– Над проектом повинні працювати мотивовані професіонали. Щоб робота була зроблена, створіть умови, забезпечте підтримку і повністю довіртеся їм.
– Безпосереднє спілкування є найбільш практичним і ефективним способом обміну інформацією як з самою командою, так і всередині команди.
– Працюючий продукт – основний показник прогресу.
– Інвестори, розробники і користувачі повинні мати можливість підтримувати постійний ритм. Agile допомагає налагодити такий стійкий процес розроблення.
– Постійна увага до технічної досконалості і якості проектування підвищує гнучкість проекту.
– Простота – мистецтво мінімізації зайвого клопоту – вкрай необхідна.
– Найкращі вимоги, архітектурні та технічні рішення народжуються у самоорганізованих команд.

– Команда повинна систематично аналізувати можливі способи поліпшення ефективності і відповідно коригувати стиль своєї роботи.

10.2 Методи Agile-розроблення

Scrum

Своїй назві «Scrum» зобов'язаний регбі.

Метод успішно застосовують такі компанії, як Microsoft, Yahoo, Siemens Healthcare.

Оскільки Scrum – каркас розроблення, в кожному наступному прикладі він може значно відрізнятися від попереднього.

Ключові елементи:

- **Product Backlog** – список вимог за проектом;
- **Sprint Backlog** – список вимог, які потрібно виконати в найближчий спринт;
- **спринти** (Ітерації) – відрізки часу на виконання певного ряду завдань;
- **щоденні п'ятнадцятихвилинні «міт-апи»** (зустрічі) – задавайте по три запитання кожному з команди: що робив вчора, що буде сьогодні, що заважає виконати завдання;
- **огляди робочих частин продукту** – із залученням до перегляду і обговорення стейкхолдерів (зацікавлених осіб);
- **ретроспектива** – обговорення проблеми і пошук рішення і шляхів поліпшення процесу після кожного спринту.

eXtreme Programming (XP)

Він застосовується виключно в сфері розроблення ПЗ і будується навколо чотирьох процесів:

- **кодування** – відповідно до єдиних у команді стандартів оформлення;
- **тестування** – тести пишуться самими програмістами до написання коду, який будуть тестувати;
- **планування** – як фінального білда, так і окремих ітерацій. Останнє проходить у середньому раз на два тижні.
- **слухання** – як розробників, так і клієнта, у ході якого зникають неясності, визначаються вимоги і цінності.

Crystal Methodologies

Алістер Кокберн, один з авторів «Маніфесту ...» пропонує проводити класифікацію проектів за кольорами критеріїв кількості осіб у команді: від 2 (Crystal Clear) до 100 (Crystal Red). Під більш масштабні проекти виділені кольори Maroon, Blue і Violet.

Crystal-проекти мають відповідати трьом основним показникам:

- швидкій доставці робочого коду – розвитку ідеї ітеративної моделі розроблення Agile;
- досконалості через рефлексію – нова версія ПЗ поліпшується на основі даних про попередню;
- «осмотичній» взаємодії – метафора комунікації та обміну інформацією між розробниками ПЗ в одній кімнаті.

Dynamic Software Development Method (DSDM)

Над розробленням DSDM працював консорціум із 17 англійських компаній. DSDM, як і екстремальне програмування, використовується переважно для створення програмного забезпечення.

Особливу роль відводять участі кінцевого споживача (користувача) у процесі розроблення. Крім цього принципу до базових належать:

- часті випуски робочих версій продукту;
- автономність розробників у плані прийняття рішень;
- тестування протягом усього робочого циклу.

DSDM ділиться на версії, які оновлюються у міру розвитку технологій, появи нових вимог до розроблення ПЗ.

Спочатку команда вивчає реальність розроблення програми і область застосування. Далі робота ділиться на три взаємозв'язаних цикли:

- **цикл функціональної моделі** – створення аналітичної документації і прототипів;
- **цикл проектування і конструювання** – зведення системи в робочий стан;
- **цикл реалізації** – розгортання системи.

Feature Driven Development (FDD)

Хоча в FDD теж застосовується ітераційна модель розроблення, від Agile вона відрізняється таким:

- більше уваги приділяють моделюванню;
- підвищена (порівняно з Agile) важливість побудови звітності й графіків;
- націлене на корпоративне розроблення.

Feature Driven Development складається з таких циклічних етапів:

- **створення загальної моделі** – бачення проекту на основі попередніх даних;
- **розроблення списку властивостей** – аналог product backlog у методиці Scrum;
- **планування за властивостями** – оцінювання складності властивостей кожним членом команди;

– **властивості кожної моделі** – технічний дизайн і реалізація – фінальна стадія, після закінчення якої властивість йде в продукт, і цикл повторюється.

Lean Software Development

Lean Software Development – скоріше не методологія, а набір принципів бережливого виробництва, який спрямований на підвищення ефективності процесу розроблення, мінімізацію витрат.

До набору входять такі сім принципів:

– **позбавлення втрат** – все, що не додає цінності продукту для кінцевого споживача;
– **постійне навчання** – безперервний розвиток команди збільшує можливості ефективного виконання завдань;
– **прийняття рішення так пізно, як тільки можна** – пріоритет не за спонтанним рішенням, а за продуманим, розробленим на основі отриманих знань;
– **швидка доставка** – за суттю – основа ітеративної моделі;
– **посилення команди** – проектна команда – основа успішного завершення завдань;
– **цілісність і якість** – потрібно спочатку виготовляти якісний продукт, щоб не витрачати час і ресурси на подальше тестування і позбавлення від багів;
– **бачення цілісної картини** – розбиття проекту на окремі частини неможливе без розуміння поточного статусу розроблення, цілей, концепції і стратегії ПЗ, що розробляється.

Agile Modeling (AM)

Agile Modeling – набір цінностей, принципів і практик для моделювання програмного забезпечення.

AM використовують як складову повноцінної методики розроблення ПЗ – наприклад, екстремального програмування або Rapid Application Development.

Принципи Agile Modeling такі:

– ефективна взаємодія між проектними стейкхолдерами;
– прагнення розробити найбільш просте з можливих рішень, яке підійде до всіх вимог;
– постійне одержання зворотного зв'язку;
– сміливість приймати рішення і відповідати за його виконання;
– розуміння того, що Ви не знаєте абсолютно все.

Agile Data Method (ADM)

ADM – набір ітеративних методик гнучкого розроблення програмного забезпечення, які роблять упор на формування вимог і рішень щодо

проекту через співпрацю окремих команд. Як і AUP, це не самоцінна методика.

Суть Agile Data Method визначається шістьма положеннями:

- **дані** – основа створення будь-якої програми;
- **проблеми з проектом** – їх можна виявити тільки при чіткому розумінні мети й концепції проекту;
- **робочі групи** – крім безпосередньої команди розробників є enterprise groups, які підтримують інші робочі групи;
- **унікальність** – немає ідеальної методики, під кожен проект треба комбінувати інструменти з різних методологій;
- **командна робота** – спільна робота набагато ефективніше, ніж поодинці;
- **«солодка пляма»** – пошук оптимального вирішення проблеми («солодкої плями»), уникаючи крайнощів.

Agile Unified Process (AUP)

AUP – спрощена версія іншої методології розроблення ПЗ – Rational Unified Process (RUP).

Від 2012 року її замінили на Disciplined Agile Delivery (DAD), але подекуди AUP ще зустрічається. Ключові позиції Agile Unified Process:

- Ваша команда знає, що робить;
- простота – понад усе;
- відповідність принципам гнучкої методології розроблення;
- сфокусованість на цінних для проекту активностях;
- незалежність у виборі інструментів;
- настроювання AUP під потреби конкретного проекту.

Essential Unified Process (EssUP)

Розроблення шведського вченого Івара Якобсона, створене для поліпшення Rational Unified Process. EssUP оперує поняттям практики, до якої належать:

- сценарій використання – опис поведінки системи;
- ітераційне розроблення – створення робочих шматків коду короткими циклами в кілька тижнів;
- командні практики – спрямовані на згуртування команди і підвищення її ефективності;
- процесуальні практики – наприклад, «Думай глобально, починай з малого» або «Залучайте стейкхолдерів до бізнес-процесів».

Усі практики в тому чи іншому вигляді зустрічаються в методологіях RUP, CMMI і гнучкої методики розроблення.

Getting Real (GR)

Ефективна для стартапів і початківців команд методологія, яка пропонує по максимуму використовувати особливості невеликих проектів і компаній: мобільність, гнучкість, пошук нових рішень, відсутність жорсткої заплутаною ієрархії і т. д. GR – збірна солянка з десятка інструментів гнучкого розроблення, які використовуються для мінімізації:

- можливостей;
- опцій і налаштувань;
- структури компанії;
- зустрічей;
- обіцянок.

Незвичайна концепція не набула поширення, хоча окремі елементи використовують інші методики.

OpenUP (OUP)

Незалежна від інструментів методологія розроблення ПЗ без жорсткої структури, яка містить такі практики:

- вимір швидкості роботи команди;
- проведення щоденних зустрічей і ретроспектив після завершення ітерацій;
- концепція мікрокроків і раннього тестування з використанням чекліст;
- методика гнучкого моделювання (AMDD).

Практики реалізуються на основі чотирьох принципів:

- узгодження інтересів і досягнення спільного бачення під час спільної роботи;
- безперервне вдосконалення через постійний зворотний зв'язок;
- фокусування на архітектурі додатків на ранніх стадіях для мінімізації ризиків;
- максимізація цінності для кінцевого споживача.

10.3 Особливості гнучких підходів до розроблення

Методи гнучкого розроблення орієнтовані на ітеративний і інкрементальний процеси програмування, в яких розроблення ПЗ розбивається на короткі цикли, що називаються ітераціями, або «Спринт». Тривалість ітерацій варіюється від тижня до місяця. Протягом ітерації команда розробників додає невеликий набір функціональності, грунтуючись на певних клієнтських пріоритах, тестує на предмет коректної роботи і перевіряє. Постійна участь клієнта дозволяє команді виявляти проблеми і відхилення, що дає можливість розробникам вчасно вносити корективи. Завдання полягає в отриманні готового до постачання ПЗ у кінці

ітерації, навіть якщо це всього лише невелика частина необхідного кінцевого продукту.

Залучення клієнтів

Співпраця з клієнтами завжди підвищує шанси проекту на успіх. Це правильно як у водоспадних проектах, так і проектах гнучкого розроблення. У водоспадних проектах клієнти зазвичай з самого початку виділяють значний час на допомогу бізнес-аналітикам у розумінні, документуванні та перевірці вимог. Клієнти повинні також брати участь в етапах приймального тестування, надаючи інформацію про відповідність продукту їхнім потребам. Однак на стадії розроблення клієнти задіяні мало. У проектах гнучкого розроблення клієнти (або власник продукту) беруть участь на всіх стадіях проекту, а власники продукту, клієнти і кінцеві користувачі беруть участь у написанні користувальницьких історій та інших вимог. Вони також повинні тестувати і надавати свої зауваження щодо нових функцій. Призначені для користувача історії зазвичай менш докладні за традиційні функціональні вимоги, тому клієнти повинні мати доступ у процесі ітерацій, щоб надавати свої відгуки та уточнення. Написані некваліфікованим учасником призначені для користувача історії скоріше надають недостатньо точну інформацію про вимоги. Незалежно від того, хто пише для користувача історії, хтось з солідним досвідом і знаннями бізнес-аналітика має редагувати історії до того, як команда почне їх реалізовувати.

Ставлення до документації

Тісна співпраця клієнтів з розробниками в проектах гнучкого розроблення зазвичай означає, що вимоги можуть документуватися менш докладно, ніж у традиційних проектах. Це не означає, що документація не потрібна. Методи гнучкого розроблення повинні сприяти створенню мінімального обсягу документації, яке дає точні вказівки розробникам і тестувальникам. Документація може бути спрощеною і вестися у вигляді карток або записів в електронних базах.

Ітеративна розробка

Однак у проекті гнучкого розроблення не документують докладно всі вимоги на самому початку проекту, а насамперед виявляють високорівневі вимоги, щоб відразу можна було розпочати планування і розстановку пріоритетів. Вимоги можуть розроблятися невеликими порціями протягом усього проекту, і навіть після випуску. Однак важливо якомога раніше визначити нефункціональні вимоги, щоб можна було спроектувати архітектуру системи, що забезпечує критично важливі продуктивність, зручність використання, доступність та інші цілі за якістю.

Розрахунок на неминучі зміни

В організаціях знають, що в проектах будуть зміни. Важливо перебудуватися, якщо це сприятиме конкурентоспроможності проекту. Потрібно управляти змінами, щоб мінімізувати їх негативний вплив на проект, але необхідно усвідомити неминучість і навіть користь змін. Це не означає, що потрібно створювати дизайн, розрахований на всі можливі майбутні вимоги. Але маючи можливість зазирнути трохи в майбутнє, розробники можуть створити більш розширювану і надійну архітектуру, яка полегшує додавання нових функцій.

10.4 Плюси і мінуси використання Agile

Плюси:

- **залучення стейкхолдерів** (зацікавлених осіб) – у команди з'являється більше можливостей зрозуміти бажання клієнта, а рання і часта доставка ПЗ посилює довіру стейкхолдерів до проектної команди і ще залучає до проекту;
- **рання і передбачувана доставка** – модель розроблення через ітерації (короткі проміжки від 1 до 6 тижнів) дає гнучкість, прискорює випуск релізу продукту;
- **фокусування на бізнес-цінності** – колаборація з клієнтом забезпечує розуміння командою того, як зробити продукт максимально цінним для споживача;
- **безперервне поліпшення якості** – тестування під час кожної ітерації, розподіл фінального билда на окремі шматки робочого коду дозволяють поліпшувати ПЗ і справлятися з її помилками до виходу фінального продукту.

Мінуси і обмеження:

- **підвищені вимоги до команди і клієнтів** – без тісної взаємодії між проектною командою і користувачами неможливо домогтися виходу якісного продукту з високою цінністю. Велика кількість інструментів і методів в Agile для впровадження потребує досвідченої команди;
- **не підходить для аутсорс** і проектів, де учасники взаємодіють один з одним тільки онлайн;
- **ризик ніколи не випустити фінальну версію ПЗ –** цей мінус, як не дивно, випливає з ітеративного розроблення і безперервного вдосконалення продукту – плюсів Agile;

– **не працює без чіткого бачення бізнес-цілей проекту** – так як Agile-команда орієнтується на стейкхолдерів, то без вироблення цілей і концепції продукту розроблення неможливе.

11 SCRUM

Scrum (рисунок 11.1) – це фреймворк, призначений для розроблення, поставки та підтримки складних продуктів. Цей фреймворк містить опис Scrum, він розповідає про ролі, події, артефакти і правила. Творцями Scrum є Кен Швабер і Джефф Сазерленд.

Scrum – це фреймворк, який допомагає вирішувати завдання, що змінюються в процесі роботи, щоб продуктивно і творчо поставляти клієнтам продукти з максимально можливою цінністю. Scrum – це процесний фреймворк, який почали використовувати для управління роботою над складними продуктами на початку дев'яностих років. Scrum не є процесом, технікою або вичерпним методом. Навпаки, Scrum – це фреймворк, в якому можна використовувати різноманітні процеси і методи.

Фреймворк – це набір базових елементів і правил, свого роду каркас, на якому будується процес розроблення. Основними елементами фреймворка є Scrum-команди і пов'язані з ними ролі, події, артефакти і правила. Кожен елемент фреймворка служить визначеній меті й є обов'язковим для успішного використання Scrum. Суть Scrum – це маленька команда людей. Кожна окрема команда надзвичайно гнучка й адаптивна. Ці переваги виявляються при поширенні на будь-яку кількість команд в організації: одну, кілька або цілі мережі команд, які розробляють, випускають, здійснюють експлуатацію і підтримку продуктів, таким чином об'єднують працю тисяч людей. Вони спільно працюють і взаємодіють завдяки просунутим архітекторам і сучасним середовищам розроблення.

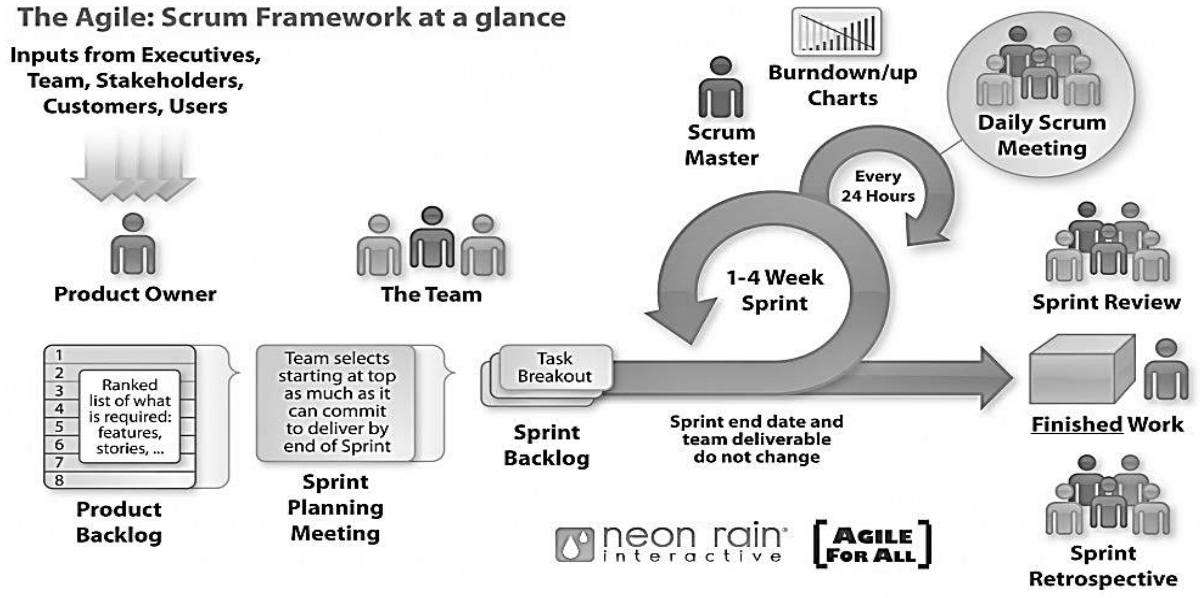


Рисунок 11.1 – Scrum

11.1 Scrum-команда

Scrum-команда складається з власника продукту, команди розробки і Scrum-майстра. Scrum-команди є здатними до самоорганізації та крос-функціональними. Команди, які самоорганізуються, самостійно вирішують, як виконувати свою роботу, а не слідують зовнішнім вказівками. Крос-функціональні команди мають усі необхідні компетенції для виконання роботи і не залежать від людей, які не входять у команду.

Scrum-команди постачають продукт ітеративно і інкрементально, максимально використовуючи можливості для отримання зворотного зв'язку. Завдяки цьому працездатна і потенційно корисна версія продукту доступна в будь-який момент.

Власник Продукту (Product Owner)

Власник продукту несе відповідальність за досягнення максимальної цінності продукту як результату роботи, яку виконує команда розробки. Власник продукту – єдина людина, яка відповідає за управління Backlogom продукту. Власник продукту може виконувати цю роботу як самостійно, так і делегувати її виконання членам команди розробки. Проте відповідальність за Product Backlog лежить на плечах власника продукту.

Роль власника продукту виконує одна людина, а не група людей. Власник продукту може відображати побажання зацікавлених осіб у Backloge продукту, але будь-хто, хто хоче змінити пріоритет елемента, повинен звертатися до власника продукту.

Роль Product Owner

- формулює вимоги;
- пріоритизує вимоги;
- коригує пріоритети на кожному спринті;
- несе персональну відповідальність за цінність вимог для ринку/користувачів;
- відповідає за взаємодію з ринком;
- тільки одна людина.

Власник продукту – це представник підрозділу, який володіє розробленим продуктом. Наприклад, у банку це може бути департамент карткових продуктів. Правильно визначити власника продукту не просто, тому що ця роль потребує поєднання таких якостей:

- мати особисту залученість до проекту і його результатів;
- мати гарні навички написання вимог.

Команда Розробки

Команда розробки складається з професіоналів, які працюють над постачанням до кінця спринту готового до випуску інкремент продукту.

Організації формують команди розробки і наділяють їх усіма повноваженнями для самостійної організації і управління своєю роботою.

Команди розробки мають ряд характеристик.

- Вони самоорганізуються: ніхто (навіть Scrum-майстер) не може вказувати команді розробки, як перетворити Product Backlog у готові до релізу інкременти.
- Вони крос-функціональні: команда володіє всіма навичками, які необхідні для створення інкремент продукту.
- Розробник – єдина роль для членів команди розробки незалежно від типу завдань, які він виконує. Scrum не визнає інших ролей у команді розробки, це правило не має винятків.
- Scrum не визнає підкоманду в команді розробки, незалежно від областей, над якими необхідно працювати (наприклад, тестування, архітектура, експлуатація або бізнес-аналітика). Це правило не має винятків.
- Команда розробки несе колективну відповідальність за створення інкремент продукту. При цьому окремі члени команди розробки можуть володіти різними спеціалізованими навичками і експертизою.

Оптимальний розмір команди розробки досить малий, щоб команда залишалася гнучкою, і досить великий, щоб виконувати значущу роботу за час спринту.

Коли в команді менше трьох осіб, взаємодія і продуктивність знижуються. Невеликі команди розробки можуть зіткнутися з проблемою нестачі навичок для створення готового до випуску інкремент продукту. Команди розміром більше дев'яти осіб зазнають труднощів з координацією роботи. Складність роботи у великих команд розробки зростає настільки, що емпіричний процес стає непридатним.

Scrum-майстер

Scrum-майстер несе відповідальність за просування і підтримку Scrum. Він досягає цих цілей, допомагаючи всім зрозуміти теорію, практики, правила і цінності Scrum.

Scrum-майстер – це лідер-слуга (Servant Leader) для Scrum-команди. Мається на увазі, що Scrum-майстер не привілейований носій влади, як це відбувається у випадку з класичним менеджментом, а лідер, який залучає учасників Scrum-команди в процес роботи, не має формальної влади. Людям поза командою він допомагає зрозуміти, що з їх взаємодій з Scrum-командою корисно, а що ні. Scrum-майстер допомагає змінити ці взаємодії так, щоб вони максимально збільшували цінність, яку створює Scrum-команда.

Роль Scrum Master:

– стежить за коректним застосуванням принципів Agile і процесів (ритуалів) Scrum;
– організовує роботу команди і забезпечує її усім необхідним;
– захищає команду, несе відповідальність за її ефективність;
– тільки одна людина.

Дуже складна роль. У класичному project management є керівник проекту. У Scrum така роль не передбачена. Кращим синонімом ролі Scrum Master буде «адміністратор». Scrum майстер організовує роботу команди проекту, але не втручається в її роботу.

– Scrum майстер не призначає людей на завдання – це робить сама команда;
– майстер не змушує людей виконувати роботу – це відповідальність команди;
– майстер не вказує Product Owner, які вимоги він повинен написати – це робота власника продукту.

Проте якщо Scrum-процес проходить з порушеннями (будь-хто з команди спізнюється на daily-meeting), то майстер повинен втрутитися і виправити ситуацію.

11.2 Події Scrum (ритуали, процеси в Scrum)

Обов'язкові події Scrum передбачені для того, щоб процес був регулярним і інші збори були б не потрібні. Кожна подія обмежена за часом і не може перевищувати максимальну встановлену тривалість. Тривалість Спринту не може бути змінена після його старту. Решта подій можуть бути завершені достроково, якщо мети заходів досягнуто. Це дозволяє уникнути втрат часу.

Кожна подія в Scrum (крім Спринту) – це формальна можливість для інспекції та адаптації. Спринт – виняток, він є контейнером для інших подій. Події створені спеціально для забезпечення максимальної прозорості та інспекції. Відмова від будь-якого з них веде до зниження прозорості і є упущеною можливістю для інспекції та адаптації.

Спринт (Sprint)

Спринт служить ядром Scrum. Спринт – часовий відрізок тривалістю місяць або менше, протягом якого створюється «готовий», тобто придатний до використання і випуску інкремент продукту.

Бажано зберігати незмінну тривалість спринту протягом усього процесу розроблення. Новий спринт починається відразу після закінчення попереднього.

Спринт складається з планування спринту, щоденного Scrum, розроблення, огляду спринту і ретроспективи спринту.

Кожен спринт можна вважати проектом, який триває не більше одного місяця.

Планування Спринту

Завдання, над якими буде працювати команда розробки під час спринту, визначаються на плануванні спринту. План створюється спільними зусиллями всієї Scrum-команди.

Планування спринту обмежено за часом. Для спринту тривалістю один місяць планування не повинно займати більше 8 годин. Якщо спринт коротше, то і планування проводиться швидше.

За результатами планування спринту Scrum-команда вирішує:

- яким буде інкремент в кінці спринту;
- як організувати роботу, щоб отримати готовий інкремент продукту.

Щоденний Scrum

Щоденний Scrum – це зустріч команди розробки, яка проводиться щодня під час спринту. Зустріч не повинна тривати більше 15 хвилин, за які команда розроблення планує свою роботу на найближчі 24 години. Команда оптимізує взаємодію між її членами і підвищує свою продуктивність, аналізуючи зроблене за останню добу і прогнозуючи, що залишилося на цей спринт роботи. Для спрощення щоденний Scrum проводиться щодня в один і той же час.

Тільки члени команди розробки беруть активну участь у щоденному скрамі. Решта учасників Scrum-команди можуть бути присутніми, однак їх участь не є обов'язковою.

Команда сама визначає формат зустрічі, але акцент завжди залишається на досягненні цілі спринту. Якісь команди проведуть дискусію, якісь будуть використовувати запитання, наприклад:

- Що я зробив учора, що допомогло команді розробки наблизитися до мети спринту?
- Що я зроблю сьогодні, щоб допомогти команді розробки досягти цілей спринту?
- Бачу я будь-які перешкоди, які можуть перешкодити мені або команді розробки досягти цілей спринту?

Scrum-майстер стежить, щоб зустріч команди розробки відбулася, але за проведення щоденного скраму відповідає сама команда. Scrum-майстер навчає команду розробки проводити щоденний Scrum за 15 хвилин або швидше.

Щоденний Scrum – це внутрішня зустріч команди розробки. Якщо на ній присутній хтось ще, Scrum майстер стежить, щоб вони не заважали зустрічі.

Зустріч команди ефективно проводити відповідно Kanban дошки, на якій відображені всі завдання спринту. Зустріч проходить тільки стоячи (stand-up meeting).

Огляд Спринту

Огляд спринту проводиться в кінці спринту з метою інспекції інкременту і, в разі потреби, адаптації Backloga продукту. Scrum-команда і зацікавлені особи під час огляду спринту спільно обговорюють, що було зроблено за спринт. Ці дані, як і будь-які зміни Backloga продукту протягом спринту, служать підставою для обговорення наступних кроків до оптимізації цінності продукту.

Огляд спринту – не статусна зустріч, а неформальна. На ній проводиться демонстрація інкремент продукту для отримання зворотного зв'язку і розвитку співпраці. Для спринту тривалістю один місяць тривалість зустрічі не перевищує чотирьох годин. Чим коротше спринт, тим коротше його огляд.

Ключові елементи огляду спринту:

- до кількості учасників зустрічі входять Scrum-команда і ключові зацікавлені особи, яких запрошує власник продукту;
- власник продукту пояснює, які елементи Backloga готові, а які ні;
- команда розробки розповідає про те, що відбулося під час спринту, які виникли проблеми і як вони були вирішені;
- команда розробки демонструє готову роботу і відповідає на запитання про інкремент;
- власник продукту описує поточний стан Backloga продукту. За необхідності він прогнозує можливі дати завершення розроблення продукту, грунтуючись на поточних показниках прогресу;
- усі присутні обговорюють, над чим варто працювати далі. Таким чином огляд спринту надає цінні дані для наступного планування спринту;
- проводиться огляд, як зміни ринку або потенційне використання продукту могли змінити те, що потрібно зробити в першу чергу;
- виконується огляд термінів, бюджету, можливостей і позицій на ринку для майбутніх релізів або можливостей продукту.

Результатом огляду спринту є переглянутий Product Backlog. Він містить елементи, які можуть увійти в наступний спринт. Також Product Backlog може бути змінений, якщо з'явилися нові бізнес-можливості.

Ретроспектива Спринту

Ретроспектива спринту – це можливість для Scrum-команди провести інспекцію, спрямовану на себе, і створити план поліпшень командної роботи в наступному спринті.

Ретроспектива спринту проводиться після огляду спринту і перед плануванням наступного спринту.

Кожну ретроспективу спринту Scrum-команда планує дії для поліпшення якості продукту, удосконалюючи робочий процес або адаптуючи критерій готовності, якщо це необхідно і не суперечить специфікації продукту і стандартам організації.

До кінця ретроспективи Scrum-команда повинна запланувати конкретні поліпшення, які вона реалізує в наступному спринті.

11.3 Артефакти Scrum

Артефакти Scrum відображують роботу або цінність. Вони забезпечують прозорість і створюють нові можливості для інспекції та адаптації. Артефакти Scrum спеціально розроблені для забезпечення максимальної прозорості ключової інформації, щоб всі учасники процесу однаково її розуміли.

Product Backlog (журнал, Беклог Продукту)

Product Backlog – це впорядкований список відомих вимог до продукту. Це єдине джерело вимог будь-яких необхідних змін у продукті. Власник продукту є відповідальним за Product Backlog, включаючи його вміст, доступність і впорядкованість.

Product Backlog ніколи не буває завершеним: на ранній стадії він містить тільки спочатку відомі і найбільш зрозумілі вимоги. Product Backlog еволюціонує разом з продуктом і середовищем, в якому він буде використовуватися. Щоб продукт залишався актуальним, конкурентоспроможним і корисним, його Backlog постійно змінюється слідом за змінами вимог до продукту. Поки існує продукт, існує і його Product Backlog.

Product Backlog містить дані, які визначають необхідність змін у наступних випусках продукту. До таких даних можуть належати нові характеристики або нові функції продукту, вимоги, інформація про шляхи удосконалення продукту і усунення дефектів.

Уточнення Product Backlog – це діяльність, спрямована на уточнення, оцінювання й упорядкування елементів у Backloge продукту. Йдеться про безперервний процес, у рамках якого власник продукту і Команда Розробки обговорюють деталі елементів Backloga продукту, тим самим перевіряючи і переглядаючи ці елементи.

Елементи у верхній частині списку зазвичай краще деталізовані, ніж елементи внизу. Чим детальніше і ясніше опис елементів Backloga продукту, тим точнішою може бути їх оцінка. Своєю чергою, чим нижче знаходяться елементи в Product Backlog, тим менше вони деталізовані.

Відстеження прогресу на шляху до цілей

У будь-який момент часу робота, що залишилася для досягнення мети, може бути підсумувана. Власник продукту відстежує загальний обсяг цієї роботи як мінімум на кожному огляді спринту, він порівнює його з обсягом, який залишався на попередніх оглядах спринту. Це дозволяє оцінити прогрес виконання запланованої роботи в бажаний термін і досягнення мети. Ця інформація робиться прозорою для всіх зацікавлених осіб.

Для прогнозування прогресу можуть бути використані різні практики, наприклад діаграми згоряння робіт (burn-down), burn-up діаграми або кумулятивні діаграми потоку.

Sprint Backlog (Беклог Спринту)

Sprint Backlog – це набір елементів Backloga, взятих у спринт, плюс план з досягнення цілі спринту і постачання Інкремент Продукту. Для забезпечення безперервного вдосконалення Sprint Backlog містить принаймні одне пріоритетне поліпшення під час попередньої ретроспективи спринту. Sprint Backlog являє собою досить детальний план, тому прогрес може бути визначений у рамках щоденного Scrum. Команда розробки змінює Sprint Backlog протягом усього спринту, тому Sprint Backlog прояснюється. Це відбувається в міру того, як команда розробки працює над планом і дізнається нові деталі про роботу, необхідну для досягнення цілі спринту.

У міру появи нової роботи команда розробки додає її до Sprint Backlog. Коли частина робіт виконана і завершена, оновлюється оцінювання робіт, що залишилися. При цьому елементи плану можуть бути видалені, якщо команда вважає, що вони втратили актуальність. Під час спринту змінювати Sprint Backlog може тільки команда розробки.

Інкремент (Increment)

Інкремент – це сума завершених під час спринту елементів Backloga продукту і всіх інкрементів попередніх спринтів.

До кінця спринту інкремент повинен бути «готовий». Мається на увазі його відповідність критеріям готовності Scrum-команди і готовність до використання. Інкремент – це сукупність виконаних робіт, яка підтримує емпіричний підхід і яку можна інспектувати в кінці спринту. Він повинен бути готовий до використання незалежно від позитивного чи негативного рішення власника продукту про його випуск.

Критерії Готовності (Acceptance Criteria)

Рішення про готовність інкременту продукту приймається виходячи з критеріїв готовності, прийнятих Scrum-командою. Ці ж критерії

допомагають команді розробки під час планування спринту визначити, скільки елементів Backloga продукту їй варто взяти в роботу.

11.4 Метрики і показники

З огляду на різноманітність інструментів, практик, методів і методологій в Agile потрібно вибрати інструмент, який допоможе визначити ефективність кожного з них. Таким інструментом є метрики.

Для більшості проектів вистачить чотири напрями метрик:

- *продуктивність* – до неї належать Velocity і WIP. Перша підійде не для всіх проектів, оскільки вимірюється кількістю виконаних завдань в ітерацію, а вони нерівнозначні. Метрика Work-in-Progress визначає ліміт завдань на різних стадіях: і чим він вищий, тим гірше;
- *прогнозування* – метрика capacity: визначення кількості ідеальних годин, доступних в наступному спринті. Відповідно можна зрозуміти, скільки часу є на роботу, наскільки ефективно виконання завдань і як спланувати кількість завдань для спринту;
- *якість* – наприклад, індекс стабільності вимог, який розраховується за формулою = (загальна кількість оригінальних бізнес-вимог + кількість вимог, які помінялися до цього часу + кількість доданих вимог + кількість прибраних вимог) / (загальна кількість оригінальних вимог). За допомогою метрики визначається кількість часу, витраченого на перероблення завдань;
- *цінності* – у кожному випадку розраховується індивідуально залежно від формату проекту. Наприклад, стартап AirBnb як метрика, яка визначає кінцеву цінність продукту для користувачів, вибрала кількість завантажених фотографій високої якості. З їх збільшенням пропорційно зростала і кількість споживачів.

До метрик застосовні ті ж правила, що і до інших Agile-інструментів.

Їх потрібно постійно переглядати, відкидати застарілі і додавати нові у міру необхідності. Вона має бути зрозуміла і доступна всій команді, не перетворюватися на самоціль.

Немає єдино правильної або потрібної вашому проекту метрики. Метрика заради метрики – погане рішення.

Burndown Chart – діаграма згоряння

На рисунку 11.2 показано: «ідеальну криву», якби завдання виконувалися по N штук в день щодня; «фактичну криву», яка показує реальну кількість (або обсяг в story points) виконаних у конкретний день завдань.

Таким чином, графік згоряння завдань показує загальний обсяг робіт, що залишилися на певний момент часу, а не відстежує кількість і стан окремих функціональних вимог або функцій (розмір яких може бути різним). Графік згоряння завдань допомагає уникнути синдрому «90 %», наочно показуючи обсяг, а не роботу, яка не відображує неминуче збільшення меж проекту. Нахил графіка згоряння завдань також відображує прогнозовану кінцеву дату проекту – точку, коли backlog буде порожній.

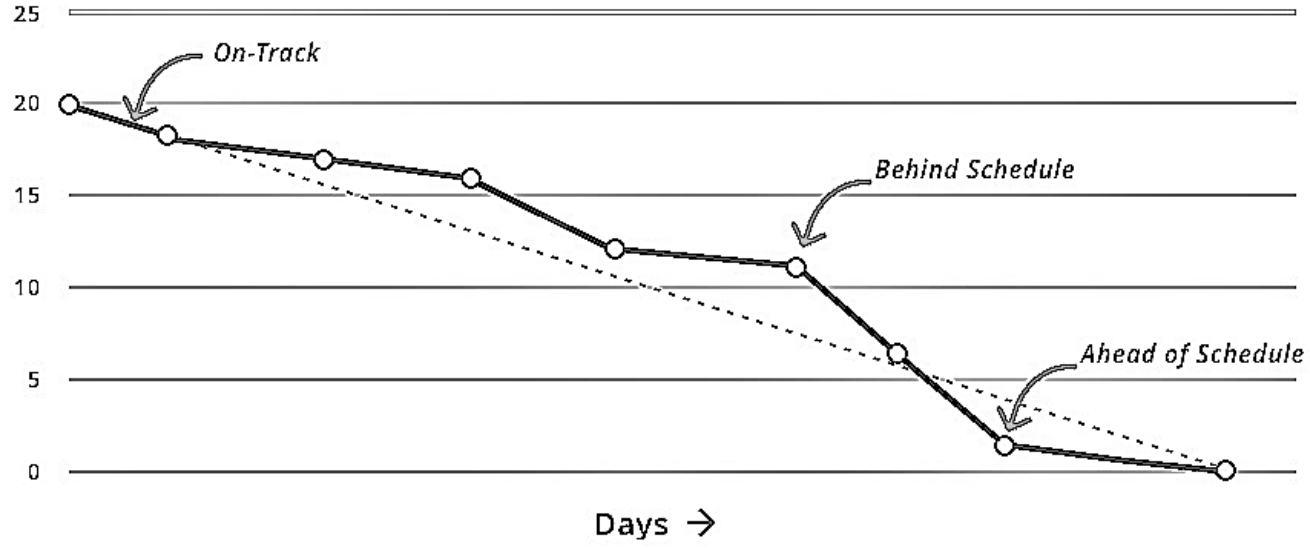


Рисунок 11.2 – Burndown Chart – діаграма згоряння

Burnup Chart

Burnup Chart – діаграма згоряння навпаки (рисунок 11.3). Сенс приблизно той же, тільки графіки йдуть не зверху вниз, а знизу вгору. Верхня межа відзначається кривою «усі завдання», коли фактична крива до неї доходить – стоп, усе завдання виконано.

Ключова відмінність від Burndown полягає ось у чому. Протягом спринту список завдань може змінюватися: додаватися нові, прибирати не актуальні.

Отже, ідеальна крива теж буде змінювати своє положення (ставати крутіше, якщо при цьому дедлайн залишити на місці).

Але якщо такі перетасовки завдань відбулися в ході роботи над спринтом, то на підсумковому графіку класичного Burndown цього не видно – зрушилася лінія, але коли конкретно і на скільки пунктів, не зрозуміло. У Burnup пропонується візуалізувати і ці зміни теж.

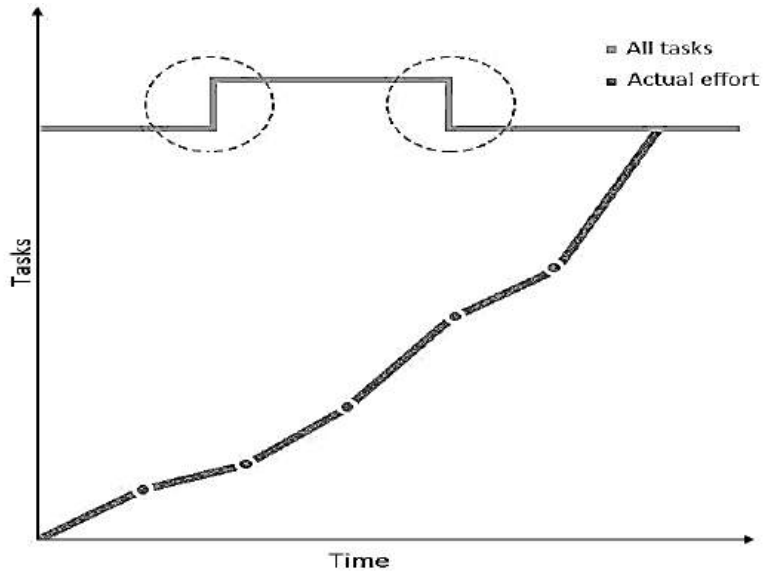


Рисунок 11.3 – Burnup Chart – діаграма згоряння навпаки

Velocity Chart – діаграма продуктивності

Velocity Chart – діаграма продуктивності (рисунок 11.4), призначена для з'ясування, наскільки фактично зроблена кількість завдань у спринті співвідноситься з плановими. Більш «глобальна» метрика дозволяє оцінити, наскільки команда справляється з планом у кожному спринті, і зробити прогноз на майбутнє. На горизонтальній осі – час, на вертикальній – кількість завдань у спринті. Поруч є два стовпці: перший – фактично виконані завдання, другий – план на спринт. Важливо: для того аби оперувати показником Velocity, потрібно, щоб тривалість спринту і кількість осіб у команді не змінювалися (як, втім, і склад команди).

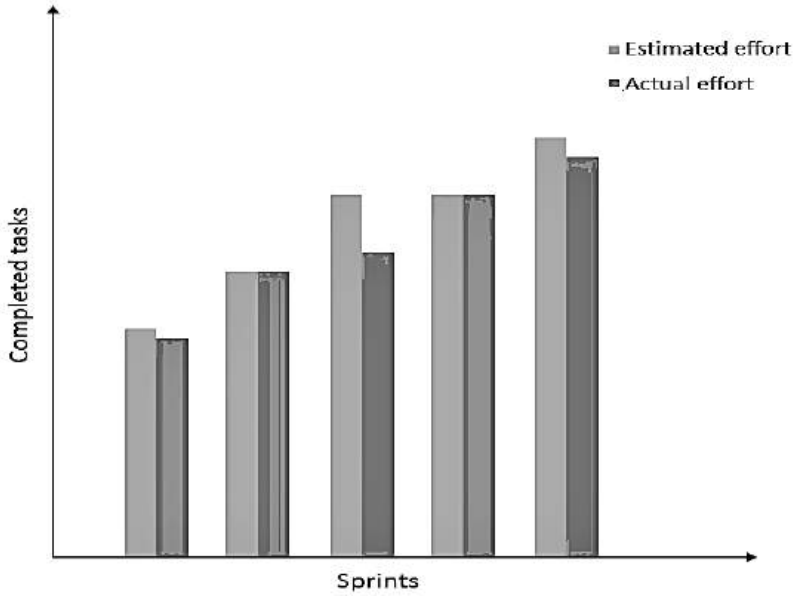


Рисунок 11.4 – Velocity Chart – діаграма продуктивності

11.5 Інтерпретація «Діаграми згоряння завдань»/Burndown Chart

Розглянемо кілька типових прикладів «Діаграми згоряння завдань» (Burndown Chart), показаних на рисунках 11.5 – 11.12.

Занадто рано. На діаграмі рисунка 11.5 видно, що команда все завдання виконала раніше терміну. Така ситуація теж не є позитивною, оскільки це означає ряд проблем, що виникнули:

Команда неправильно оцінила майбутню роботу.

У разі швидкого виконання завдань розробники не додавали завдання з наступного спринту.

Вимоги були погано опрацьовані, що не дозволило взяти до спринту достатню кількість завдань.

Команда сильно перестрахувалась, включивши спочатку додатковий термін.

У разі такої проблеми найчастіше Scrum Master запитує команду про можливість додавання додаткових завдань з Product Backlog.

Запізнення (рисунок 11.6). Також один з видів негативних діаграм згоряння завдань.

Однією з можливих причин тут може бути постійне додавання нових завдань під час спринту, що збільшує навантаження.

Іншою частою проблемою є недоробленість завдань, коли завдання зроблені наполовину. Такі завдання, як висловився Джефф Сазерленд, «є мотлохом».

У такій ситуації на Daily Scrum Meeting обов'язково потрібно говорити про проблеми, що заважають йти до мети рівною дорогою. Як тільки лінія реальних завдань пішла вище, відразу треба вирішувати проблему – це також один з постулатів методології Scrum.

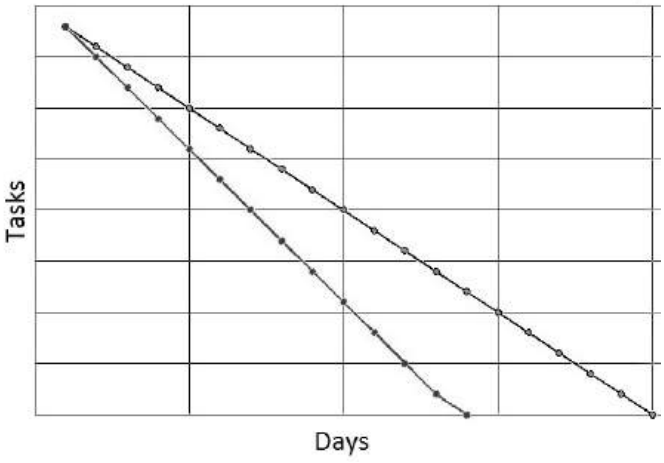


Рисунок 11.5 – Завдання виконані занадто швидко

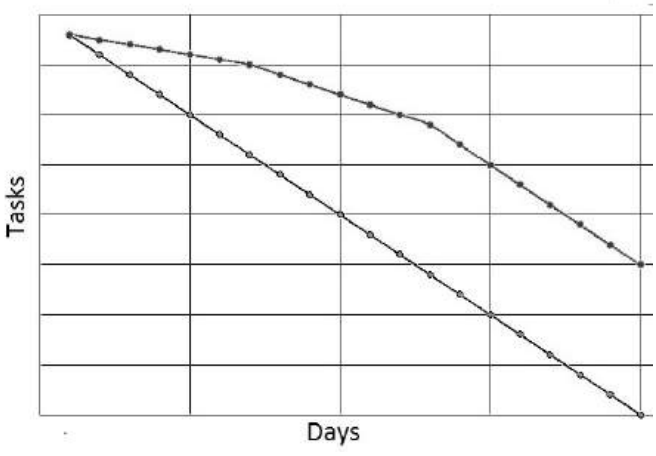


Рисунок 11.6 – Виконання завдань із запізненням

Без оцінок/Кінцева оцінка (рисунок 11.7). Може бути навіть команда і працювала, тільки забула або не захотіла використовувати діаграму згоряння завдань. Незважаючи на закінчений Sprint, усі підсумкові оцінки були внесені в діаграму згоряння в самий останній день після завершення роботи.

Це рівнозначно тому, коли закінчені завдання взагалі не вносяться. За даним графіком неможливо зробити висновки про правильність роботи команди, і, навіть більше того, можна припустити, що команда не прагне до розвитку.

Удосконалення (рисунок 11.8). Scrum Team на поточних показниках виглядає досить добре. За лініями видно, що на самому початку були труднощі, але під час Daily Scrum Meeting усі питання розкривалися і Scrum Master виправляв роботу, ведучи команду до мети. Також можливо, що група робила принципове прискорення для досягнення мети.

Ще однією причиною, наприклад, може бути те, що команда брала додаткові завдання.

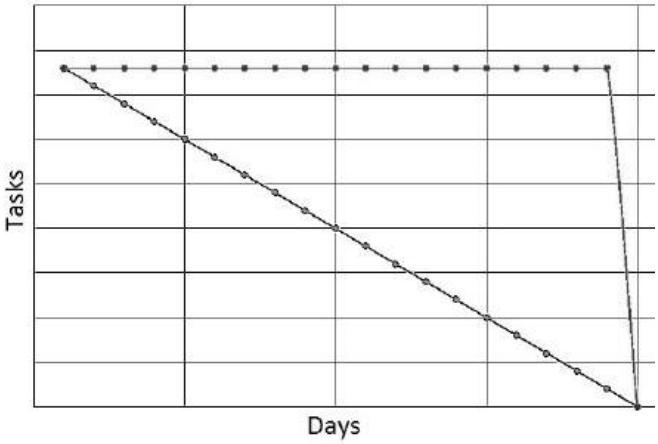


Рисунок 11.7 – Відсутність проміжних оцінок

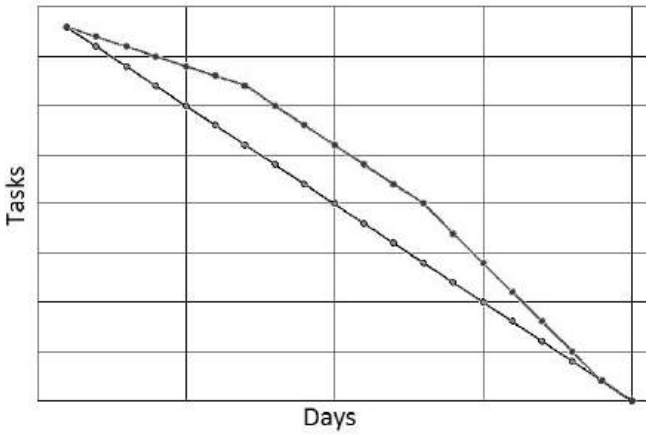


Рисунок 11.8 – Подолання труднощів

Дострокове виконання (рисунок 11.9). Цей графік показує команду, яка завищила час, який знадобиться для виконання завдань.

Якщо ця команда підтримує свою поточну швидкість, вони закінчать усі завдання задовго до дати закінчення спринту. Власнику продукту потрібно додати більше завдань до спринту (показаного стрімким збільшенням кошторису), щоб утримувати команду протягом усієї тривалості спринту.

Непослідовна продуктивність (рисунок 11.10). У цьому прикладі команда проекту починає сильно випереджати базову лінію (див. стрілку ліворуч). Однак робота потім сповільнюється і з четвертого дня команда відстає від графіка. Заходи вживаються у точці, на яку вказує стрілка праворуч, і команда повертається до початкового темпу, щоб закінчити роботу.

Це може свідчити про непослідовність дій у колективі. Керівник проекту повинен дослідити, щоб зрозуміти причину раптових змін.

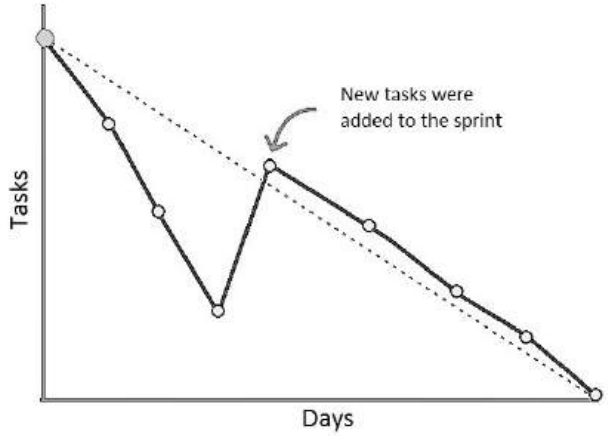


Рисунок 11.9 – Занадто швидке виконання робіт

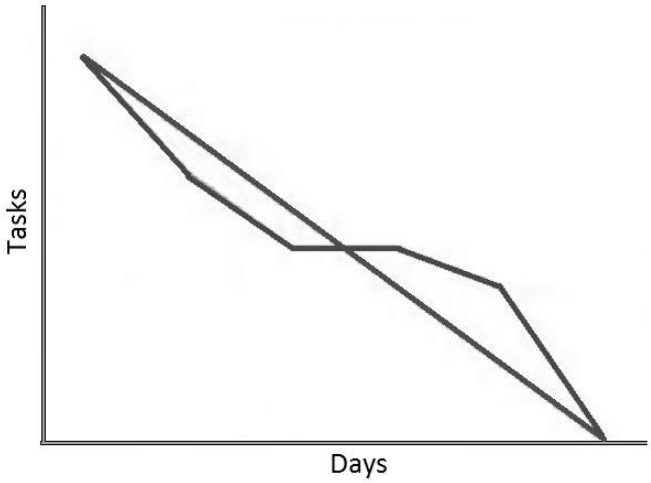


Рисунок 11.10 – Зміни у продуктивності команди

Діаграма зростає протягом спринту (рисунок 11.11). Така діаграма може вказувати на некомпетентність власника продукту або на велику невизначеність, або на інші зміни у роботі. Яка б не була причина згоряння діаграм цього типу – це може бути дуже демотивуючим для команди: не тільки пропущена мета спринту, але, схоже, Ви навіть подорожували далі від неї у всьому спринті.

Один з найбільш неприємних типів діаграм: фактична лінія робить протилежне тому, що вона повинна робити.

Набуття досвіду (рисунок 11.12). Насправді досвідчена група, це така група, яка після початку роботи відразу долає усі виникаючі труднощі і вдосконалюється так, що різко переходить до активного «спалювання» завдань.

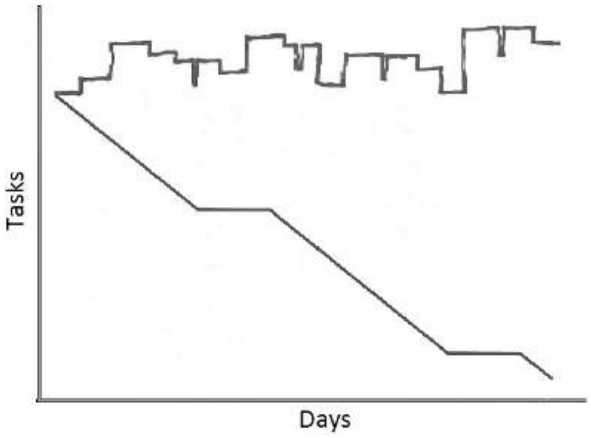


Рисунок 11.11 – Невизначеність обсягу робіт

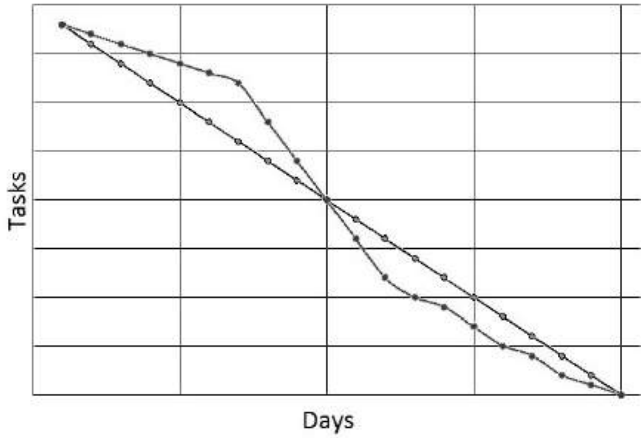


Рисунок 11.12 – Набуття досвіду командою

11.6 Обмеження і проблеми в Scrum

Scrum був розроблений, щоб посилити співпрацю у проектах. Як наслідок – середовище, де співпраця не складається, неідеальна для Scrum. Наприклад, Щоденний Scrum неможливий для деяких членів міжнародних команд. Якщо така важлива частина Scrum, як щоденний захід, починає дратувати, а не допомагати, це знак того, що Scrum може не спрацювати.

Ще одне обмеження полягає у тому, що Scrum команда повинна бути різноманітною і гнучкою. В ідеалі – будь-який член команди розробки має бути здатний виконувати завдання інших членів команди. Це – ще один аргумент на користь Щоденного Scrum, на якому збирається вся команда.

Важливі особливості, що часто забуваються

Часто можна почути, що Scrum не працює або працює гірше, ніж очікувалося. Необхідно зауважити, що найчастіше так відбувається з однієї з таких причин:

1. Scrum застосовується неправильно або неповністю.

На думку авторів Scrum, емпіричний досвід є головним джерелом достовірної інформації. Необхідність повного і точного виконання Scrum вказана в The Scrum Guide і зумовлена нетиповою організацією процесу, відсутністю формального лідера і керівника.

2. Недооціненна важливість роботи із забезпечення мотивації команди.

Одним з основних принципів Scrum є багатофункціональні команди, які самоорганізуються. Згідно з дослідженнями соціологів кількість самовмотивованих співробітників, здатних до самоорганізації, не перевищує 15 % від працездатного населення.

Таким чином, лише невелика частина співробітників здатна ефективно працювати в Scrum без істотних змін у ролях Scrum master і Product Owner, що суперечить ідеології Scrum, і потенційно призводить до неправильного або неповного використання Scrum.

3. Scrum застосовується для продукту, вимоги до якого суперечать ідеології Scrum.

Scrum належить до групи Agile. Таким чином Scrum вітає зміни у вимогах в будь-який момент (Product backlog може бути змінений в будь-який момент). Це ускладнює використання Scrum у fixed-cost/fixed-time проектах. Ідеологія Scrum стверджує, що заздалегідь неможливо передбачити всі зміни, таким чином немає сенсу заздалегідь планувати весь проект, обмежившись лише just-in-time плануванням, тобто планувати тільки ту роботу, яка повинна бути виконана в поточному Sprint. Існують й інші обмеження.

Переваги і недоліки

Scrum має досить привабливе достоїнство. Scrum орієнтований на клієнта, адаптивний. Scrum дає клієнтові можливість робити зміни у вимогах у будь-який момент часу (але не гарантує того, що ці зміни будуть виконані). Можливість зміни вимог приваблива для багатьох замовників ПЗ.

Scrum досить простий у вивченні, дозволяє економити час за рахунок виключення некритичних активностей. Scrum дозволяє отримати потенційно робочий продукт у кінці кожного спринту.

Scrum робить упор на самоорганізування і багатофункціональну команду, здатну вирішити необхідні завдання з мінімальною координацією. Це особливо привабливо для малих компаній і стартапів, оскільки позбавляє необхідності наймати або навчати спеціалізований персонал керівників.

Звичайно, у Scrum є недоліки. З огляду на простоту і мінімалістичність Scrum задає невелику кількість досить жорстких правил. Однак це вступає в конфлікт з ідеєю клієнтоорієнтованості в принципі, тому що клієнтові неважливі внутрішні правила команди розробки, особливо якщо вони обмежують клієнта. Наприклад, у разі необхідності, за рішенням клієнта Sprint backlog може бути змінений, не зважаючи на явне протиріччя з правилами Scrum.

Проблема є більшою, ніж здається. Оскільки Scrum належить до групи Agile, в Scrum не прийнято, наприклад, створення плану комунікацій та реагування на ризики.

Іншою слабкою особливістю Scrum є упор на самоорганізовану, багатофункціональну команду. При уявному зниженні витрат на координацію команди це призводить до підвищення витрат на відбір персоналу, його мотивацію, навчання. При певних умовах ринку праці формування повноцінної, ефективної Scrum команди може бути неможливим.

12 KANBAN

Kanban – японський термін, який почали використовувати у виробництві в 60-х роках в компанії Toyota. В основі цього принципу – конвеєрний метод виробництва, а також різні швидкості виконання окремих технологічних операцій на виробництві.

Kanban на заводах компанії Тойоти – це бережливе виробництво, визначальним принципом якого є поняття «точно в термін». У перекладі з японської слово Kanban означає «сигнал» або «картка». На автомобільних заводах такі картки використовувалися, щоб передати інформацію з одного етапу на попередні про те, скільки і яких деталей потрібно.

Kanban відрізняється від Scrum у першу чергу орієнтацією на завдання. Якщо в Scrum основна орієнтація команди – це успішне виконання спринтів (треба визнати, що це так), то в Kanban на першому місці – задачі. Спринтів ніяких немає, команда працює над завданням з самого початку і до завершення.

Деплоймент завдання робиться тоді, коли воно готове. Презентація виконаної роботи – теж.

Команда не повинна оцінювати час на виконання завдання, бо це має мало сенсу і майже завжди помилкове спочатку.

Команда для роботи використовує Kanban-дошку. Дошка повинна повністю відображувати процес створення цінності, який у Kanban називають потоком. Однак

Канбан ≠ доска.

Якщо відсутні певна межа незавершеної роботи і сигнал для проведення нової роботи за системою, це не Kanban.

12.1 Призначення Kanban

Kanban у виробничій системі обмежує обсяг незавершеного виробництва. На всіх стадіях виробничого процесу, крім оброблюваних деталей, знаходиться величезна кількість вузлів, деталей і заготовок у вигляді запасів незавершеного виробництва. Необхідно організувати їх зберігання, облік і контроль, інакше можливі псування, втрата або розкрадання деталей. Зі зберіганням запасів незавершеного виробництва пов'язані значні витрати.

Крім того, запаси незавершеного виробництва можуть стати баластом, що знижує гнучкість системи і заважає переходу на нові види продукції.

Системи управління за принципом «Just-in-Time» (точно вчасно) обмежують незавершене виробництво шляхом точного планування потреб на кожному етапі виробничого процесу. Kanban використовує сигнальні картки для визначення потреб у деталях. Картки передаються із завершальних етапів на більш ранні. Кількість замовлень обмежена, що не дозволяє запускати нові завдання, поки не виконані наявні, або замовляти деталі про запас. Така система називається витягаючою: завдання «витягуються» (pull) з кінця конвеєра, а не «затовкуються» (push) в його початок.

Для візуалізації процесу використовується дошка завдань (Kanban-дошка), проте сама собою дошка не поліпшить виробничий процес. Важливо правильно визначити обмеження, що забезпечують баланс між різними етапами виробничого процесу. Дошка завдань, надаючи наочну інформацію, дозволяє виявляти «вузькі місця», де утворюються затори

невиконаних завдань, а також етапи, де через дефіцит деталей виникають простої.

Користь від обмеження незавершеного виробництва в розробленні ПЗ не так очевидна, як при складанні автомобілів. Ніхто не навчить програмістів програмувати швидше за допомогою цих методів. Але більшу частину часу завдання знаходиться в черзі або блокується з тих чи інших причин, а іноді просто є забутими в гущі недороблених справ. З наведенням ладу в задачах Kanban впоратися цілком може.

Kanban (рисунок 12.1) – це «витягаюча» система. У ній створюється баланс між постійним потоком, який усуває витрати на очікування, і мінімальною кількістю роботи в процесі (WIP – Work in Progress), що знижує ризики перевиробництва.

WIP регулюється за допомогою карток: їх кількість зафіксована, а інструкції в них направляють виконавців нижнього потоку.

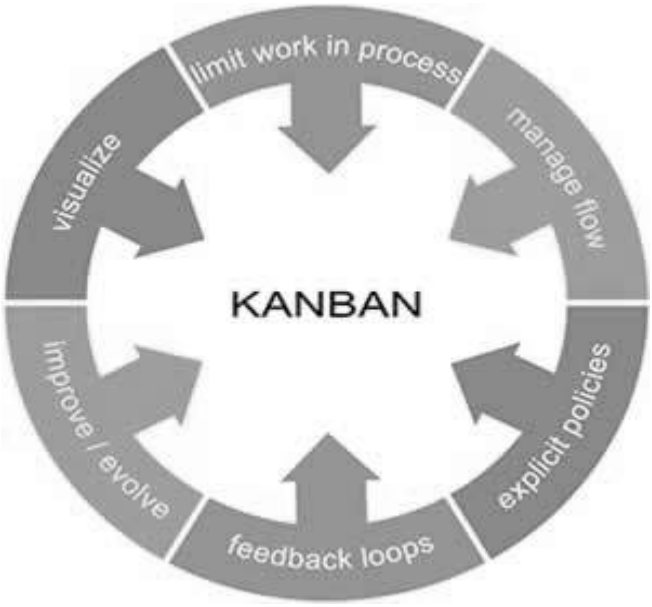


Рисунок 12.1 – Kanban

12.2 Правила Kanban

У Kanban усього шість правил, вони вводяться поступово. Нові не додаються, доки попередні зміни не стали звичними для більшості співробітників. Спочатку Kanban пропонує щадити стару структуру та ієрархію, тому зміни будуть еволюційними.

Правило 1. Візуалізуйте потік завдань

Kanban спирається на візуалізацію (рисунок 12.2). Усі завдання записуються на загальнодоступному місці, щоб у будь-який момент знати, як йдуть справи.

Візуалізація буває різною: дошка зі стікерами, стіл з картками, таблиця в Excel або програми типу Trello і Jira. Фізична дошка має перевагу в тому, що вона завжди перед очима. Не потрібно включати комп'ютер, відкривати браузер і заходити на сайт, щоб дізнатися, як йде робота. Команда відразу «бачить» актуальну картину. На ній легко вносити зміни і вона забезпечує тактильну і соціальну взаємодію між учасниками команди. Зате для аналітики або підрахунку метрик зручнішою є програма.

Запишіть усі завдання. Щоб створити візуалізацію, потрібно записати усі завдання, які Ви виконуєте зараз і збираєтеся зробити в найближчі дні. Після цього стане зрозуміло, скільки у Вас реальної роботи, а скільки в планах.

Визначте статуси завдань. Статуси завдань – це колонки на дошці. Колонки можна використовувати різні, конкретних правил немає. У найпростішому випадку використовуються три колонки: «Зробити», «У роботі» і «Готово». У реальних дошках, як правило, колонок більше, що дозволяє отримати більш детальну інформацію про стан завдань.

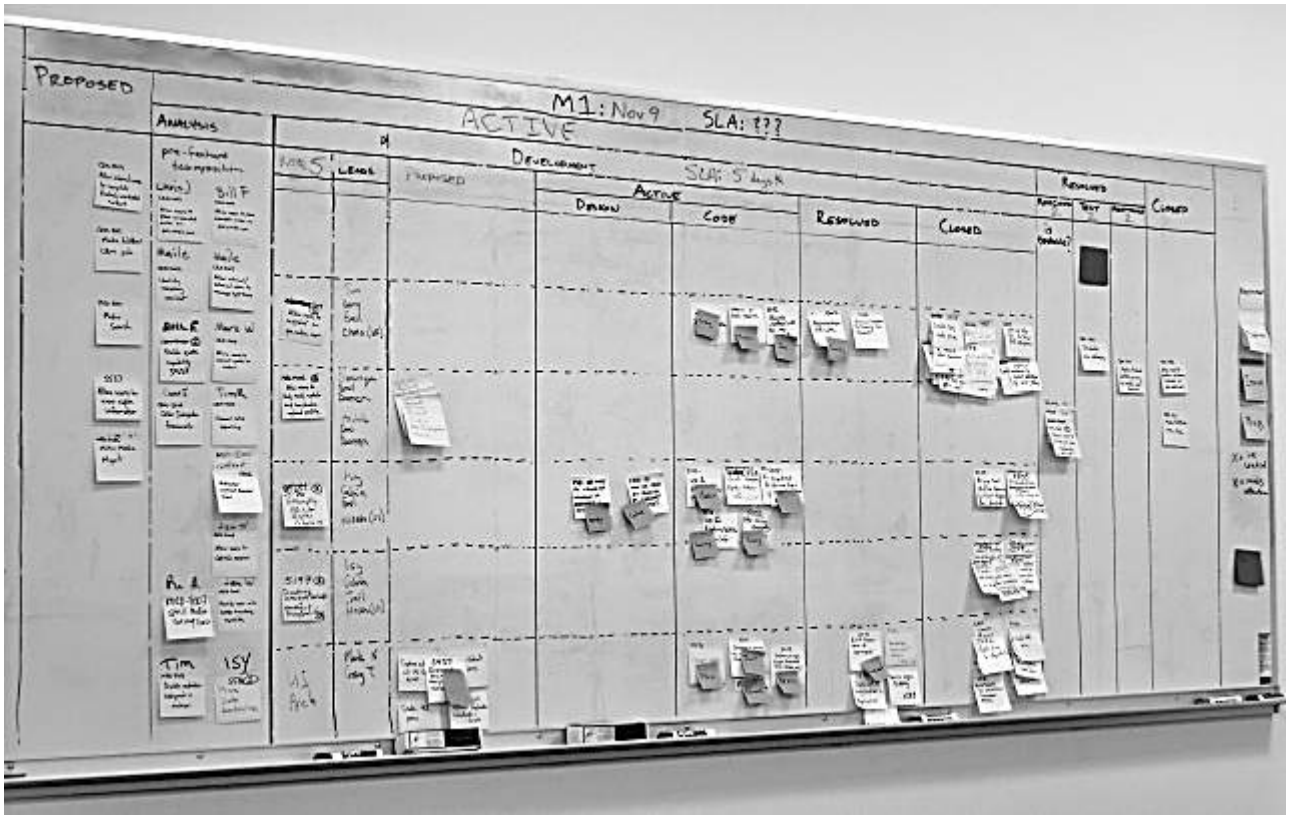


Рисунок 12.2 – Візуалізація потоку завдань

Усі завдання повинні бути на дошці. Не можна працювати над тим, чого немає у візуалізації.

Правило 2. Обмежуйте кількість одночасної роботи

Нерегульована багатозадачність – це одна з причин, чому проекти розтягуються: сили витрачаються не на завдання, а на перемикання між ними.

Kanban пропонує обмежити кількість одночасної роботи. Таким чином Ви підвищите свою ефективність і прискорите просування карток від статусу «Зробити» до статусу «Готово». Рекомендується не більше двох задач на члена команди. Для початку можна визначити фактичну кількість поточних завдань і прийняти за початкове обмеження. Далі ліміт потрібно поступово зменшувати. Обмеження по стовпцях підбираються дослідним шляхом.

Після обмеження кількості одночасної роботи в колонці «Зробити» виявиться багато карток. Щоб їх упорядкувати, потрібна пріоритизація. Можна помітити картки кольором, розташувати в певному порядку або скласти рейтинг з балами. Головне, щоб всі однозначно розуміли, які завдання потрібно зробити зараз, а які можна відкласти на пару днів.

Закінчуйте розпочаті справи, а не беріться за кілька нових паралельно. Одна виконана робота краще декількох розпочатих.

Обмеження багатозадачності – найважливіше і одночасно саме спірне правило Kanban. Це суперечить деяким менеджерським методикам, які рекомендують переривати тривалі завдання для виконання коротких. Це дозволяє швидше передавати роботу на наступні етапи. Однак Kanban потребує обмежити одночасне виконання завдань, щоб прискорити завершення кожної з них. В екстрених випадках допускається призупинити або заблокувати поточне завдання, щоб виконати більш термінове та пріоритетне.

Правило 3. Контролюйте потік завдань

Візуалізація допомагає стежити за швидкістю просування карток і рівномірним завантаженням співробітників. Якщо щось не так, на дошці це відразу видно. Дивлячись на дошку, в першу чергу приділяйте увагу тим завданням, які «підвисають» у тому чи іншому стовпці. Якщо у Вас якийсь із етапів займає найбільше часу, то спробуйте перерозподілити ресурси або ж додати людей, якщо є така можливість.

Дошка покаже, як йде робота. Допоможіть Команді закінчити її якомога швидше.

Правило 4. Зробіть домовленості й очікування явними

Правила, за якими працює Команда, мають бути відомі кожному і при цьому регулярно змінюватися. Рекомендується повісити найважливіші правила біля дошки або всередині колонок. Наприклад, над кожною з колонок можуть бути зафіксовані критерії, при виконанні яких задача може бути поміщена в дану колонку.

Домовленості допомагають Команді працювати злагоджено. Зробіть їх явними.

Правило 5. Аналізуйте роботу і вводьте зворотні зв'язки

Регулярні планерки й аналіз – обов'язкова вимога KANBAN. Вони потрібні, щоб бути впевненим: Команда рухається у правильному напряму і не вибивається із термінів і бюджетів.

Наприклад, якщо код, який пишуть розробники, періодично не проходить тестування і повертається на доопрацювання, то можливо є варіанти поліпшити якість розроблення, щоб у тест потрапляв більш якісний продукт.

Обмежень за форматом немає. Це можуть бути зустрічі, телефонні дзвінки або просто анкети. Agile віддає перевагу живому спілкуванню, тому рекомендується збиратися біля дошки. Планерки-летючки – кожен день і кожен тиждень, аналіз – раз на місяць. Регулярні зустрічі, які ще є «петлями» зворотного зв'язку, називають каденціями.

Будьте ініціативні: спілкуйтеся з колегами і пропонуйте ідеї.

Правило 6. Еволюціонуйте завдяки спільним експериментам

Kanban-команда завжди знаходиться в пошуку ідеальної системи, де картки рухаються по дошці максимально швидко.

Для цього Команда проводить експерименти: змінює кількість одночасної роботи або, по-іншому, пріоритизує завдання. Щоб система еволюціонувала, експерименти повинні бути загальними. Потрібно регулярно пробувати нове. Щоб точно знати, який ефект дав нововведення, не проводьте кілька експериментів відразу. Краще пробувати одну ідею за іншою, і залишати в роботі найвдаліші.

Експерименти допомагають команді розвиватися. Не бійтеся нового!

12.3 Класи обслуговування

Kanban використовує класи обслуговування для того, щоб підвищити пріоритет певним типам робіт, замовникам або нівелювати такий вплив на бізнес, як вартість затримки. Вартість затримки – це недоотриманий прибуток або понесені витрати через невчасно надані послуги. Розглянемо вплив вартості затримки і відповідний клас обслуговування на прикладах:

1. Прискорений клас – невідкладна швидка допомога. Немає часу відкладати вирішення проблеми і ставити його в чергу завдань. Треба діяти якомога швидше. Поточні завдання можуть бути заблоковані задля вирішення нагальної проблеми.
2. Клас з фіксованою датою – вартість затримки різко зростає після певного періоду, наприклад, через введення в дію нового закону. Не встигнемо вчасно, є ризик втратити ліцензію або потрапити під штрафні санкції.
3. Стандартний клас – вартість затримки зростає пропорційно часу. Якщо робимо відразу, отримуємо прибуток відразу. Якщо робимо довго, отримуємо прибуток нескоро.
4. Нематеріальний клас – робота потрібна, але явного прибутку не несе, вартість затримки зростає повільно. Наприклад, прибирання в будинку. Можна і не робити прибирання регулярно, але через півроку доведеться робити генеральне прибирання.

SLA – Service level agreement, чи угода про рівень сервісу.

Незважаючи на те що назва перекладається як угода, насправді це – елемент статистики, який свідчить про те, як швидко в середньому раніше виконувалися завдання, і дозволяє у зв'язку з цим побудувати реалістичні очікування. Але на відміну від статистики цей рівень може включати невеликий запас при повідомленні його внутрішнім або зовнішнім замовникам.

12.4 Каденції

Каденція – термін з музики. У контексті Kanban-методу вона означає ритм. Каденціями називають регулярні зустрічі, які ще є «петлями» зворотного зв'язку. Регулярність задає ритм, з яким тече потік роботи. Існує сім каденцій:

– Kanban-мітинг (щоденна). Мета зустрічі – збільшити швидкість потоку завдань. Ці летючки зручно проводити в формі «щоденної прогулянки вздовж дошки». Команда переглядає дошку справа наліво (тобто починаючи з завершальних етапів), знаходить проблемні місця і шукає рішення, як швидше завершити поточні завдання. Кожен може внести пропозицію, і команда до цього прислухається.

- Зустріч з наповнення черги (зазвичай раз на два тижні). Беремо на себе таке зобов'язання, як сервіс.
- Зустріч з планування поставки (зазвичай раз на два тижні). Повертаємо виконані зобов'язання.
- Зустріч з огляду сервісу (зазвичай раз на два тижні). З метриками обговорюємо якість сервісу і як його поліпшити, якщо потрібно.
- Операційна зустріч (зазвичай раз на місяць). З метриками обговорюємо якість взаємодії пов'язаних сервісів.
- Зустріч з огляду ризиків (зазвичай раз на місяць). З метрик обговорюємо вплив заблокованих завдань на роботу сервісу.
- Зустріч з огляду стратегії (зазвичай раз у квартал). З метриками обговорюємо зміни в стратегії. Збираються всі команди, які працюють у компанії. Керівництво розповідає про фінанси, і кожен співробітник розуміє, скільки заробив його відділ, що робить компанія в цілому і в якому стані знаходяться завдання. Команди розповідають, які ресурси їм необхідні.

Над завданням може працювати кілька вузькопрофільних команд. Наприклад, спочатку працюють аналітики, потім дизайнери малюють прототип, а на третьому етапі включаються розробники. При цьому універсальні (кросфункціональні) команди також можуть використовуватися.

12.5 Ролі

Роль Менеджера запитів (SRM)

У матеріалах з Kanbany автори зрідка чесно кажуть, що ця роль потрібна швидше для того, щоб не звільняти Product Owner при переході зі Scrum, але і в цій позиції (SRM) немає особливої потреби, якщо немає проблем з отриманням завдань від замовника. Ця роль може бути корисною тим командам, які страждають від відсутності «єдиної точки входу» у продуктових компаніях. Тоді ця людина може встановлювати формальні політики і розбиратися із вхідною чергою, але це точно не передбачається як робота на повну ставку. Як варіант, можна мати одного реквест менеджера на кілька команд і перетворити в нього колишнього менеджера проектів або керівника групи розробки (у загальному того, хто в новому процесі залишився без зайвої роботи).

Роль Менеджера поставки (SDM)

Service delivery manager – набагато сильніша роль, і на неї найчастіше ставлять саме менеджера проектів, як такого, хто потрібен спочатку, щоб справи доводилися до кінця. За наявності деливери менеджера в більшості випадків роль реквест менеджера практично не потрібна, хоча формально вони разом забезпечують двовладдя, реквест

менеджер займається тими завданнями, якими ще не вирішили, потрібно їх робити чи ні, а деливери менеджер тими, які вже точно потрібно робити і доводити до кінця.

12.6 Cumulative Flow Chart – діаграма сукупного потоку

Cumulative Flow Chart являє собою зведений графік (рисунок 12.3), по суті візуалізує стан Kanban-дошки в часі. Показує співвідношення завдань з різними статусами «в плані», «в роботі», «на контролі», «зроблено» – у кожен момент часу. На горизонтальній осі – час, на вертикальній – кількість завдань.

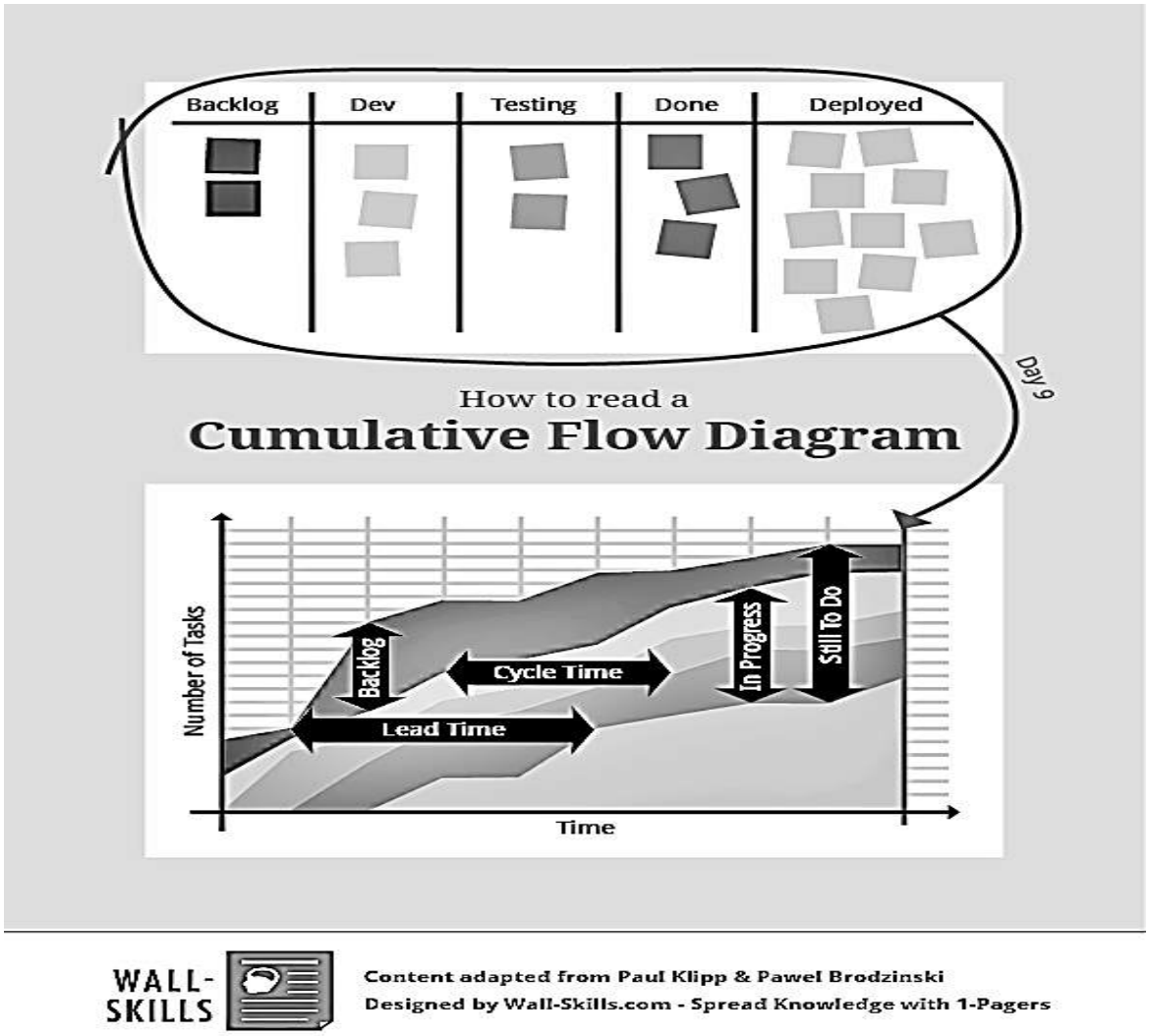


Рисунок 12.3 – Cumulative Flow Chart – діаграма сукупного потоку

Метрики Flow Chart

Метрики знімаються з усього потоку завдань. Ось короткий список метрик:

- **Cycle Time** – час, який задача перебувала в розробленні від моменту, коли нею почали займатися, до моменту, коли вона пройшла фазу кінцевої поставки.
- **WIP** – кількість завдань, які одночасно знаходяться в роботі. Розділяються за різними стадіями роботи.
- **Lead Time** – час від появи завдання до її кінцевої поставки. Містить Cycle Time і час очікування в черзі на реалізацію.
- **Wasted Time** – час, який задача проводить у різних чергах, а не безпосередньо в роботі.
- **Effectiveness** – відсоток часу, який витрачається безпосередньо на роботу із завданням, а не на очікування в різних чергах.
- **Throughput** – кількість завдань, яку може виконувати команда в одиницю часу (день, тиждень, місяць).

Якщо взяти конкретний момент часу, то можна побачити, що лінії, які ілюструють різні статуси або наближаються одна на одну, або віддаляються. Звідси можна зробити висновок: якщо лінія «план» злетіла вгору, а кількість «готових» зростає повільними темпами, – значить, Ваші виконавці не встигають справлятися з потоком завдань.

Якщо на графіку утворилося «вузьке місце» (його ще називають «пляшковим горлечком», рисунок 12.4), то це свідчить про те, що хтось в технологічному ланцюжку гальмує процес, завдання не йдуть з одного етапу на наступний, утворюються простої.

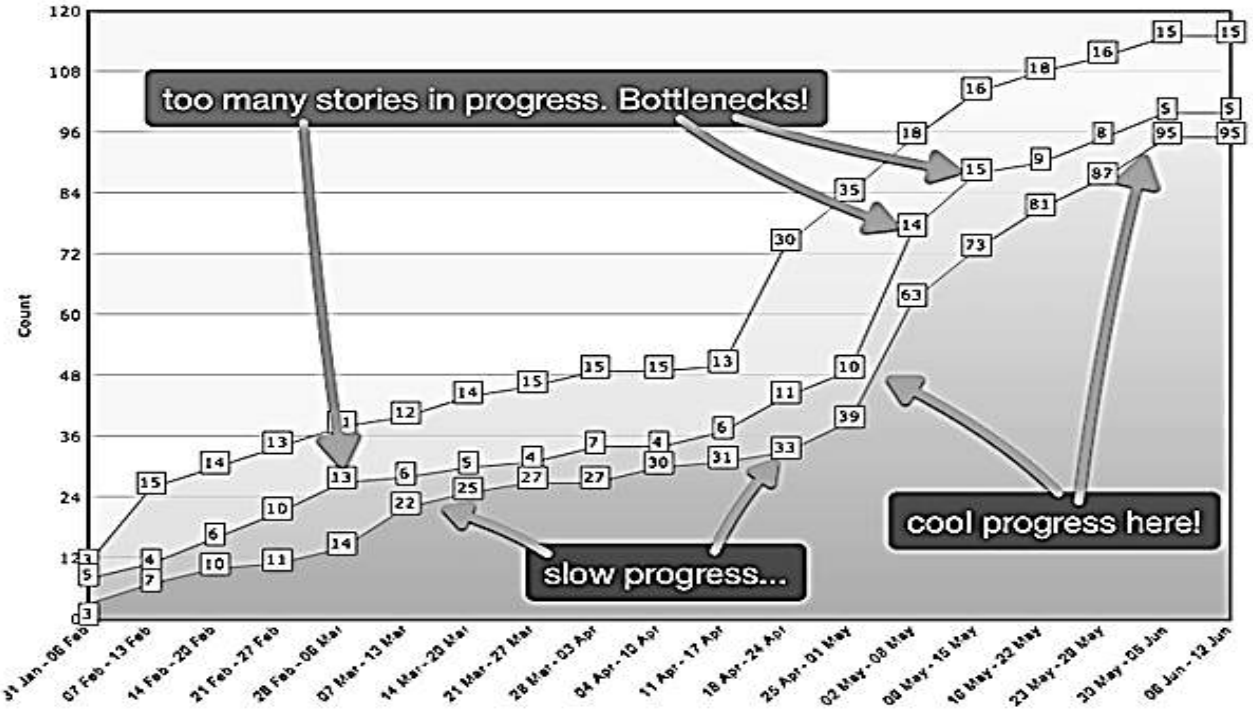


Рисунок 12.4 – Пошук «вузьких місць»

Наприклад, виробництво не встигає забезпечити продукцією відділ контролю якості. Керівник повинен прийняти рішення: збільшити кількість ресурсів на цьому етапі, підвищити продуктивність поточних виконавців, скоротити потік нових завдань – на свій розсуд.

12.7 Переваги та недоліки Kanban

Kanban має такі переваги:

- Гнучкість планування. Команда концентрується тільки на поточну роботу, пріоритет завдання виставляється менеджером.
- Високе залучення команди до процесу розроблення. Завдяки постійним зборам, прозорості процесів і можливостям самоорганізації працівники згуртовуються і проявляють щирий інтерес.
- Менша тривалість циклу. Якщо кілька людей має схожі навички, тривалість скорочується, якщо ж тільки один – з'являється вузьке місце. Тому співробітники повинні ділитися знаннями і тим самим оптимізувати тривалість циклу. Тоді вся команда зможе взятися за роботу, яку забуксувала, і відновити плавний потік.
- Менше вузьких місць. Ліміти дозволяють швидко знаходити вузькі і проблемні місця, які з'явилися через дефіцит уваги, людей або навичок.
- Наочність. Коли всі виконавці мають доступ до даних, то вузькі місця легше помітити. Kanban-команди, крім самих карток, зазвичай використовують два загальних звіти: графіки управління і сукупного потоку.

На практиці система відмінно себе показує в сферах неосновного виробництва – групи підтримки програмного забезпечення або служби підтримки.

Kanban добре працює при управлінні стартапами без чіткого плану, але де активно просувається розроблення.

Kanban має і недоліки:

- система погано працює з командами кількістю понад п’ять осіб;
- він не призначений для довгострокового планування.

12.8 Відмінності Kanban від Scrum

Існує багато спільного, наприклад, наявність Backloga і дошки завдань в обох підходах.

Тим не менш, є важливі відмінності (таблиця 12.1).

Таблиця 12.1 – Відмінності Kanban від Scrum

	Scrum	Kanban
Темп	Повторювані спринти фіксованої тривалості	Безперервний процес
Випуск релізу	У кінці кожного спринту після схвалення проектним менеджером (власником товару)	Потік триває без перерв або на розсуд команди
Ролі	Власник продукту, Scrum майстер, команда розробників	Команда під керівництвом ПМ, в деяких випадках залучаються тренери з agile kanban
Головні показники	Швидкість команди	Проведений час
Прийнятність змін	У ході спринту зміни небажані, оскільки можуть призвести до неправильної оцінки завдань	Зміни можуть трапитися в будь-який момент

13 ЖУРНАЛ (BACKLOG) ПРОДУКТУ

Журнал або Product Backlog (Product Backlog) – це пріоритезувати набір призначених для користувача історій. В процесі виконання цей список може оновлюватися і доповнюватися.

Журнал Продукту – єдиний впорядкований список вимог, тобто перелік усіх можливостей і функцій, які зацікавлені люди хочуть побачити в Продукті.

Є різні способи формування Backloga. Але спільним є такий принцип:

У проекті не повинно бути активності, не відображеної в Backlog

Журнал (backlog) проекту гнучкої розробки містить список запитів на роботу, яку повинна виконати команда (IIBA, 2013). Журнали проектів зазвичай складаються з користувацьких історій, але в деяких командах їх наповнюють іншими вимогами, бізнес-процесами та дефектами, які потрібно усунути. У кожному проекті має бути лише один журнал. Тому може знадобитися представляти в журналі дефекти, щоб можна було визначати їх пріоритети в одному ряду з новими для користувача історіями. У деяких командах дефекти скасовують як нові призначені для користувача історії або варіанти старих історій. Журнали можуть реалізовуватися у вигляді карт історій або в спеціалізованих інструментах. Пуристи від гнучкої розробки наполягають на використанні карт, але вони незручні для роботи у великих проектах або в розподілених командах.

Розстановка пріоритетів у журналі виконується регулярно – одні завдання вибираються на наступні ітерації, а інші відкидаються.

Пріоритети, які призначаються елементами журналу, не фіксуються раз і назавжди і можуть змінюватися в наступній ітерації. Відстеження зв'язків між елементами журналу і бізнес-вимогами полегшує пріоритизації. В усіх проектах, і не тільки в тих, де застосовується гнучке розроблення, потрібно управляти пріоритетами завдань, які залишаються в журналі.

Звичайний Product Backlog містить такі пункти (робочі елементи):

- функції продукту (наприклад, форми для користувача історій – опису бажаної функціональності);
- виявлені дефекти (баги);
- технічні (архітектурні) роботи.

Зазвичай команда додає потрібні деталі й оцінки в елементи Backloga під час спеціального проекту, який називається backlog grooming, або refinement. Backlog refinement (поліпшення, оптимізація, «чистка») – це захід, під час якого команда додає деталі, оцінки та порядок в елементи продукту. Цей постійний процес означає співпрацю власника продукту і розробників, коли ними розглядаються і переглядаються всі елементи продукту.

13.1 Призначені для користувача історії (User Story)

User Story – це коротке формулювання наміру, що описує щось таке, що система повинна робити для користувача.

Призначена для користувача історія – це коротке твердження, яке виражає потребу користувача і служить вихідною точкою для спілкування з метою з'ясування деталей. Призначені для користувача історії спеціально створені для обслуговування потреб «гнучких» розробників. При вивченні призначених для користувача вимог Ви маєте право задіяти назви варіантів використання, функцій або потоків процесів, але саме призначені для користувача історії найчастіше використовуються в проектах гнучкого розроблення. Обсяг користувальницької історії визначають так, щоб її реалізація могла би укластися в одну ітерацію.

User Story мають низку нюансів:

- вони не є детальним описом, а являють собою скоріше обговорюване подання намірів (потрібно зробити щось на зразок цього);
- вони є короткими і легко читаються;
- вони являють собою невеликі інкременти цінної функціональності, яка може бути реалізована в рамках декількох днів або тижнів;
- вони відносно легко піддаються оцінюванню;
- вони організовані в списки, які легко впорядкувати;
- вони не деталізуються на самому початку проекту, а більш опрацьовуються при надходженні нової інформації;

– вони потребують мінімуму супроводу і можуть бути безпечно скасовані.

Історію можна оцінити за критеріями «INVEST»:

– ***I**ndependent (Незалежний)*. Історії можуть опрацьовуватися в будь-якій послідовності і зміни в одній користувальницької історії не впливають на інші;
– ***N**egotiable (Договірної, здатність домовлятися)*. Уникайте надмірної деталізації; зберігайте їх, щоб команда могла вирішити, яку частину історії реалізувати;
– ***V**aluable (Цінний)*. Користувачі або клієнти отримують деяку цінність з історії;
– ***E**stimable (Оцінний)*. Команда повинна мати змогу оцінювати історії і використовувати оцінки для планування;
– ***S**mall (Маленький)*. Великі історії складніше оцінити і спланувати. Історія повинна бути спроектована, закодована і протестована в рамках ітерації;
– ***T**estable (тестується)*. Повинні бути визначені критерії готовності, на основі яких розробляються тестові приклади (test cases).

Структура user story

Текст user story повинен пояснювати роль/дії користувача, його потребу і цінність.

Рекомендується дотримуватися такої структури історії:

як <роль / персонаж>, Я <щось хочу отримати>, <з метою>. As a <user type>, I want to <function> So that <benefit / value>

Іншими словами:

хто – що – навіщо

Наприклад:

як споживачеві мені зручно шукати книги за жанрами, щоб швидко знайти ті, які я люблю читати.

як керуючий з випуску нової продукції я хочу мати можливість відслідковувати покупки наших клієнтів, щоб бути в курсі, які книги їм можна пропонувати.

У користувальницькій історії:

– є один actor (персонаж);
– є одна дія;
– є одна цінність benefit / value / impact.

Actor

Рекомендується використовувати такі прийоми:

- розділіть усіх акторів на групи: цільова група, важлива група, менш важлива група тощо;
- дайте унікальні назви акторам у цих групах (Покупець, Менеджер, Водій, Доктор тощо), навіть якщо в системі у них будуть однакові ролі «Користувача системи»;
- пишіть історії з точки зору цих акторів, указуючи їх унікальні назви.

Внаслідок цього Ви зможете правильно вибудувати пріоритет, оскільки історії акторів цільової групи для нас важливіші.

Призначені для користувача історії повинні висловлювати думку тих, хто буде купувати і користуватися продуктом (не забудьте про те, що це не тільки кінцеві користувачі, а й ті, хто впливають на здійснення покупки. Наприклад, кінцевими користувачами ігор часто є діти, але купують їх батьки).

Дія

Це суть історії, «що потрібно зробити». Важливо описувати історію на рівні «ЩО?», а не «ЯК?». Опишіть проблему, а не її вирішення. Краще Ви потім з командою це обговоріть і знайдете більш оптимальне «ЯК» - рішення.

При розробленні історій необхідно дотримуватися симетричності дій: якщо заплановано додавання якогось елементу, повинна бути можливість його видалення, скасування і т. п.

Цінність

Має відображувати бізнес цінності для користувача. Історія повинна впливати на актора. А вже цей вплив веде до мети, яка має цінність. Якщо такого зв'язку немає, – значить, Ви робите щось марне.

Приклади невдалих користувальницьких історій:

- *Як користувач я хочу керувати рекламними оголошеннями, щоб видаляти застарілі або помилкові оголошення.*

Це історії різних акторів: адміністратора, рекламодавця. І у них різні потреби і різні бізнес-цінності. Доцільно замінити цю історію двома історіями різних користувачів (це саме заміна, а не декомпозиція)

- *Як рекламодавець я можу керувати своєю рекламою в системі, орієнтованій на користувачів.*

Як адміністратор я можу управляти оголошеннями користувачів, так щоб контент сайту був легальним.

– *Як власник продукту, я хочу, щоб у системі була можливість видаляти оголошення, аби користувачі могли видаляти оголошення.*

Тут відразу дві помилки: невірний актор і відсутність мети.

Product Owner (власник товару) написав історію для себе, а не для користувачів. Product Owner відповідає за роботу з однією історією. Але він повинен описувати потреби і цілі користувачів. У цьому випадку цілі взагалі немає: хочу видаляти, щоб видаляти.

Нерідко замість роботи з користувачами розробники і дизайнери починають самі писати історії від їх імені. Такі історії не відображують потреби реальних користувачів. Якщо немає можливості працювати з користувачами, краще обійтися без історій і скласти традиційну специфікацію вимог.

13.2 Деталізація користувальницьких історій. Завдання і критерії готовності

Деталізація потрібна, але деталі історії не опрацьовуються заздалегідь. Замовник і команда під час обговорень історій спільно приходять до розуміння рівня деталізації, який необхідний на поточній фазі, і приймають спільні рішення, поповнюючи історії все більшою кількістю інформації.

Завдання/підзадача (task / sub task) – використовується для більш детального поділу користувальницької історії розробниками і QA. Чим досвідченіший розробник, тим детальніше опис завдань і більша їх кількість під конкретною користувальницькою історією. Завдання потрібні в першу чергу самому розробнику, щоб через тиждень він міг згадати, що потрібно зробити в певній користувальницькій історії.

Початкові призначені для користувача історії можуть бути неточними, суперечити один одному або частково дублюватися. Прийнято говорити, що користувач сам не знає, що йому потрібно. Насправді користувач не може описати свої потреби з тим ступенем деталізації, яка необхідна програмістам. Безліч деталей повинна уточнюватися в процесі обговорення користувальницьких історій за участю команди і користувачів.

Таким чином історії поповнюються деталями в міру необхідності, еволюціонуючи від коротких висловлювань до деталізованих і узгоджених вимог із вбудованими критеріями готовності.

Часто деталізація історій відбувається під час вироблення критеріїв готовності (acceptance criteria). Команда в процесі обговорення повинна вирішити, які деталі мають бути опрацьовані, щоб для користувача історія уважалася реалізованою. Критерії готовності лягають в основу тестів.

Критерії готовності доповнюють історію, роблять її вимірною, а також дозволяють продемонструвати історію, випустити її для користувачів або інвесторів.

Необхідно знати, що історії погано визначають технічні (нефункціональні) вимоги. Якщо необхідно пояснити, що архітектурний елемент повинен робити як компонент або сервіс, опишіть технічну історію, використовуючи мову моделювання (наприклад, UML).

13.3 Дефекти (bugs) як завдання

У багатьох agile-методологіях виявлені при тестуванні дефекти (bugs) включаються в backlog поряд з одними історіями та іншими робочими елементами. Це дозволяє одноманітно управляти роботою як зі створення нових функцій, так і з удосконалення або виправлення вже наявних.

Баг включається в backlog якщо:

– дефект був виявлений при регресійному тестуванні раніше написаного коду;
– дефект виявлений в новому коді, але його виправлення перенесено на наступну ітерацію (спринт).

Якщо дефект виявлений під час тестування нового коду, як правило, нове завдання-баг у backlog не заводиться. Замість цього пов'язане з ним завдання повертається до списоку To-Do з відповідним коментарем і виконується його переоцінка. Таким чином, завдання не буде завершене, доки не будуть усунуті всі пов'язані з нею баги.

У деяких методиках рекомендується заводити в backlog всі виявлені баги для їх пріоритизації. Це дозволяє уникнути витрат часу на «вилизування» фонових завдань. Однак треба розуміти, якщо виправлення непріоритетного бага було відкладено на наступну ітерацію, досить імовірно, що він уже ніколи не буде виправлений. Це дозволить зосередити зусилля на якнайшвидшому розробленні пріоритетних функцій, але може призвести до поступової деградації ПЗ через накопичення малозначних дефектів.

13.4 Епік (epic)

Епік (epic, епопея) – це великий обсяг робіт, який може бути розбитий на більш дрібні обсяги. Зазвичай епік використовують для групування і структуризації користувальницьких історій. Іншими словами, епік – це колекція історій. У епік можуть бути згруповані ті, що відносяться до однієї категорії користувачів (епік продавця), до однієї підсистеми (епік мобільної версії) або до однієї групи пов'язаних функцій різних користувачів (епік управління обліковими даними). Майк Кон (Cohn, 2010) визначає епік (epic) як призначену для користувача історію, яка дуже

велика для реалізації в одній ітерації. З цієї причини епіки потрібно розбивати на набори дрібніших історій. Іноді епіки такі великі, що їх доводиться розбивати на ряд дрібніших епік, кожна з яких своєю чергою розбивається на історії, доки не будуть отримані історії, які можна з упевненістю оцінити, реалізувати і протестувати в одній ітерації.

Структура опису епік (рисунок 13.1) може бути такою ж, як і у користувальницької історії. Іноді крім епік уводять ще й теми, що являють собою верхній рівень угруповання вимог, пов'язаний з якоюсь із цілей проекту.

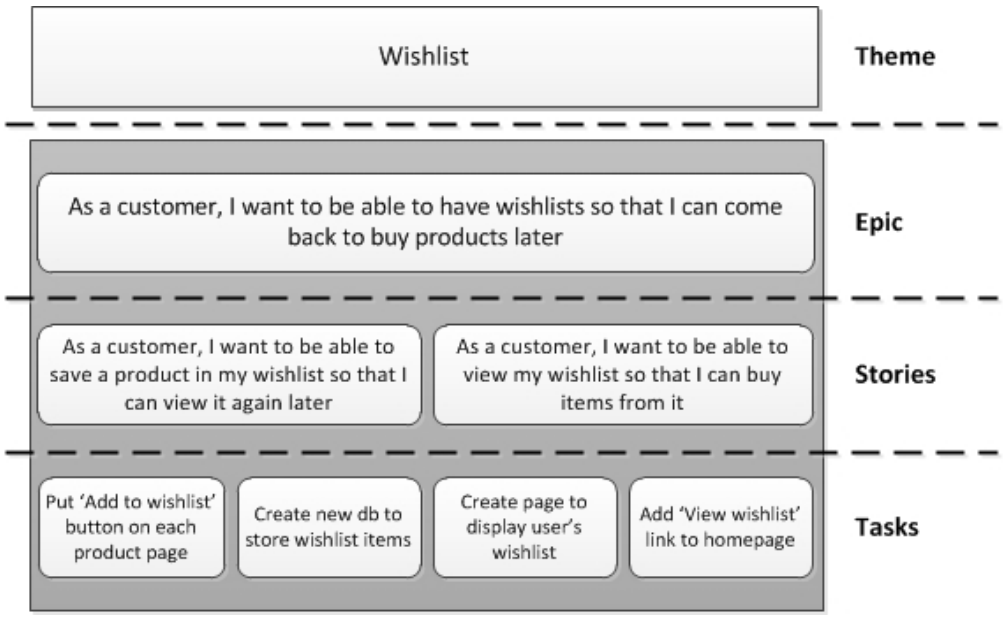


Рисунок 13.1 – Структура опису епік

13.5 Функції (Features)

Функція – це група можливостей системи, яка являє собою цінність для користувача. У контексті проекту гнучкого розроблення функції можуть поширюватися на одну призначену для користувача історію, кілька таких історій, окрему епіку або навіть на кілька епік.

Наприклад, функція збільшення масштабу в фотокамері телефону може розроблятися для реалізації двох не пов'язаних між собою користувальницьких історій:

- *• Будучи матір'ю я хочу мати можливість робити на шкільних заходах знімки, на яких чітко видно мою дочку, щоб я могла ділитися ними з дідусем і бабусею.*
- *• Будучи орнітологом я хочу мати можливість на відстані отримувати чіткі фотографії птахів, де їх можна розрізняти.*

Визначення історій самого нижнього рівня, що відповідають бізнес-вимогам, дозволяє визначати мінімальний набір функціональності, який здатна надати команда і який одночасно є певною цінністю для клієнта.

13.6 Приклади користувальницьких історій

Як водій, у якого загорілася лампочка бензину, я хочу швидко знайти найближчу хорошу заправку, щоб заправитися якісним бензином.

Критерії приймання:
Як водій, у якого загорілася лампочка, я можу переглянути всі найближчі заправки.
Як ... я можу вибрати заправки відомих брендів АЗС.
Як ... я можу бачити найближчі заправки обраних брендів списком.
Як ... я можу бачити найближчі заправки, обрані на карті.

Оброблення помилок:
1. При вимкненій геолокації користувача необхідно дати йому інформацію про те, де її ввімкнути.

Технічні замітки:
1. Заправки в списку повинні оновлюватися при зміні місця розташування користувача на 100 метрів.

Таке формулювання завдання допомагає розробникам і тестувальникам не намагатися порівнювати готову задачу з UI вимогами, які швидко застарівають, а дивитися на основні проблеми, які повинні бути вирішені в рамках завдання.

Правилько структурована користувалььницька історія містить багато додаткової інформації, корисної для розробників (рисунок 13.2).

Епік з історіями:

An Epic: Managing profiles		
A Story: As an app user, I want to add profile photos so that more people write to me about how awesome I am	A Story: As an admin, I want to delete / block photos from users 'profiles so that they do not scare off other people with their nude pics (or violate community rules)	A Story: As an app user, I want to have a separate field where I can tell more about myself so that people fall in love with my personality and not with my penthouse in the center of New York

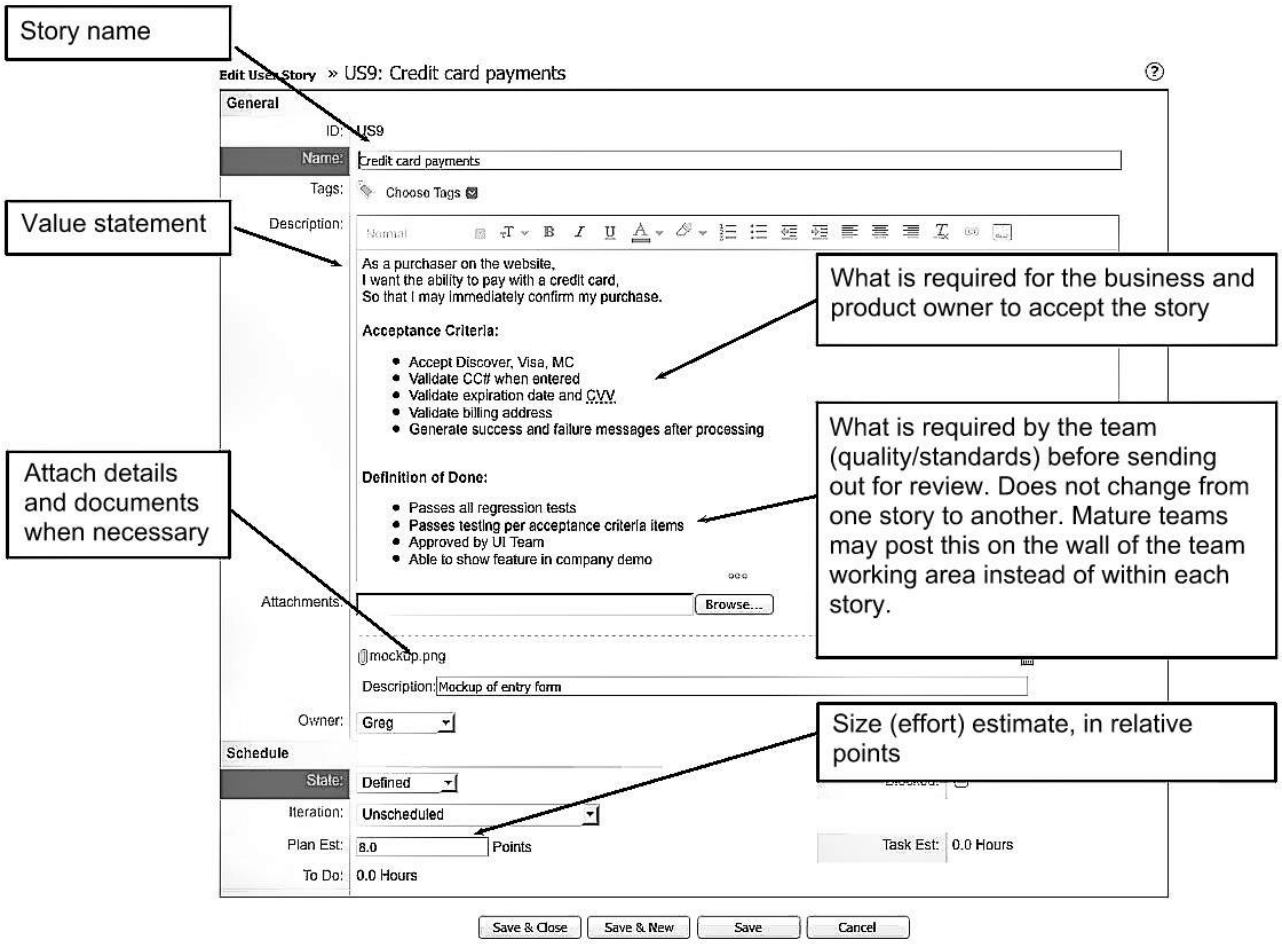


Рисунок 13.2 – Приклад структуризації користувальницької історії

14 ЗАГАЛЬНІ ПРИЙОМИ ТА ІНСТРУМЕНТИ AGILE-ПІДХОДІВ

14.1 Дошка завдань (Kanban-дошка)

Kanban-дошка є невід'ємною частиною методології Kanban, але застосовується і в багатьох інших підходах, і не тільки в Agile.

Класичний Kanban – дошка завдань розбивається на стовпці, що відповідають етапам роботи. Завдання пишуть на паперових стікерах. Потім їх приклеюють на дошку в той стовпець-етап, якому відповідло б завдання. Базова Kanban-дошка складається просто з трьох колонок: «не розпочато», «у процесі» і «виконано».

Для кожної картки можна призначити наступний статус, той, в який вона повинна потрапляти після проходження поточного. Це дозволяє побудувати автоматизацію за Push і Pull методологіями (Push – для карток встановлені конкретні відповідальні особи, а Pull – на наступну стадію

завдання буде потрапляти без виконавця і будь-який учасник проекту зможе взяти її собі в роботу).

Незважаючи на те що у Kanban-дошок дуже проста структура, вони в той же час є надзвичайно потужним інструментом.

Оскільки всі робочі елементи додаються на дошку і підтримуються в актуальному вигляді, будь-яка людина може з одного погляду побачити, де зараз знаходиться його робота, а також отримати в загальних рисах уявлення, скільки вже зроблено, скільки залишилося, а над чим працюють прямо зараз.

Інформація, розміщена на Kanban-дошці, може бути подана і у вигляді діаграми Гантта. Однак робота з Kanban-дошкою простіша і не потребує спеціальних знань і підготовки в галузі управління проектами. Тому Kanban-дошка краща, якщо плануванням та управлінням розроблення буде займатися не тільки менеджер, але і команда розробників.

Вона досить доступна для команди, щоб її прийняти, досить інформативна для менеджменту та ключових осіб, щоб бути їм корисною. Її просто підтримувати в фізичному і в електронному виглядах, вона досить гнучка, щоб адаптувати її під потреби різних проектів.

Дошка може бути як фізичною, так і віртуальною. Віртуальні дошки зручні для ведення обліку та автоматичного підрахунку різних метрик. Однак фізична дошка наочніша.

Велику дошку на стіні офісу видно завжди і відразу всім співробітникам і, крім планування, вона служить також мотивацією для команди. Багато agile-методів передбачають проведення щоденних зборів стоячи перед дошкою.

Роль дошки може виконувати стіл або просто стіна офісу.

Дошку можна організувати відповідно до особливостей робочого процесу, створивши колонки відповідно до етапів виконання робіт. Також на дошці позначаються ліміти незавершених робіт, що обмежують кількість одночасно виконуваних робіт (рисунок 14.1) .

Рисунок 14.1 – Дошка з лімітами незавершених робіт

Іноді замість одного стовпчика To Do вводять декілька стовпців, що відповідають різним рівням планування: Future, This Week, Today, ... (рисунок 14.2).

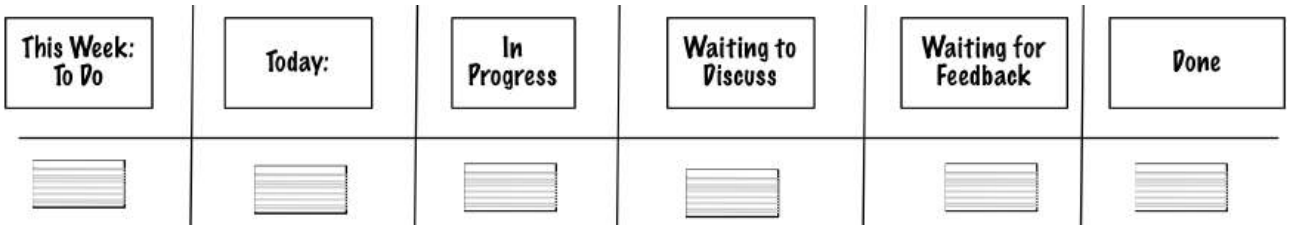


Рисунок 14.2 – Дошка з різними рівнями планування

Можна використовувати різнокольорові картки для позначення різних видів завдань (рисунок 14.3).

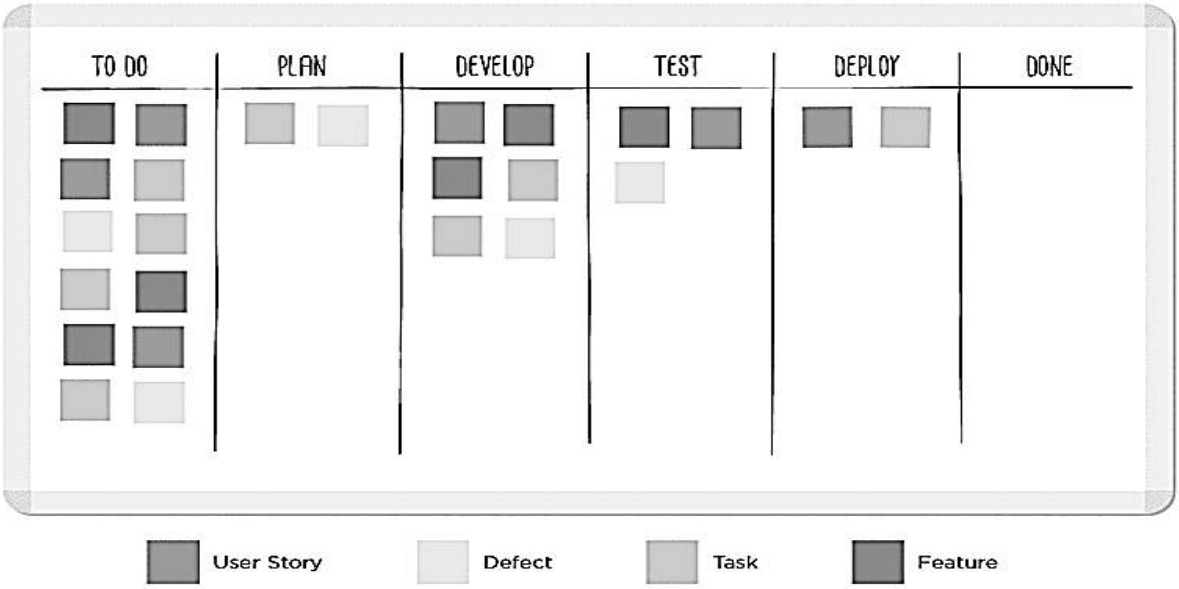


Рисунок 14.3 – Кольорові позначки різних видів завдань

На дошках можуть виконуватися спеціальні підколонки-буфери. Проміжні колонки Done служать буферами між етапами робочого процесу. Наприклад, у такому буфері можуть знаходитися завдання, кодування яких уже закінчилося, а тестування ще не почалося. Накопичення завдань у буфері понад установлений ліміт говорить про проблеми з продуктивністю на наступному етапі (рисунок 14.4).

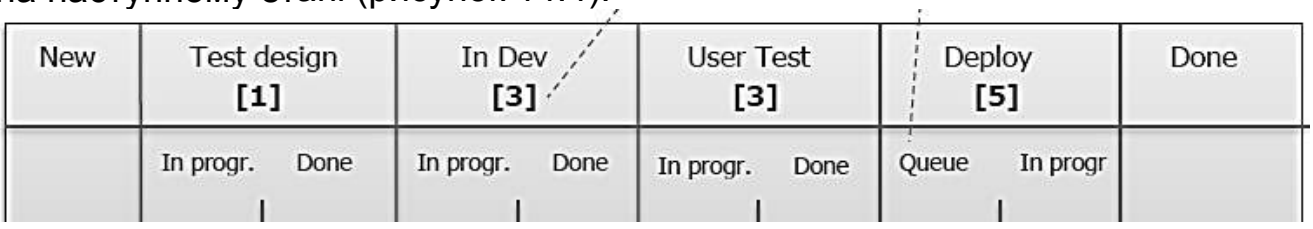


Рисунок 14.4 – Дошка з накопичувальними буферами

На дошці можна позначити очікуваний термін завершення кожної з задач, прийнятих у роботу (рисунок 14.5).

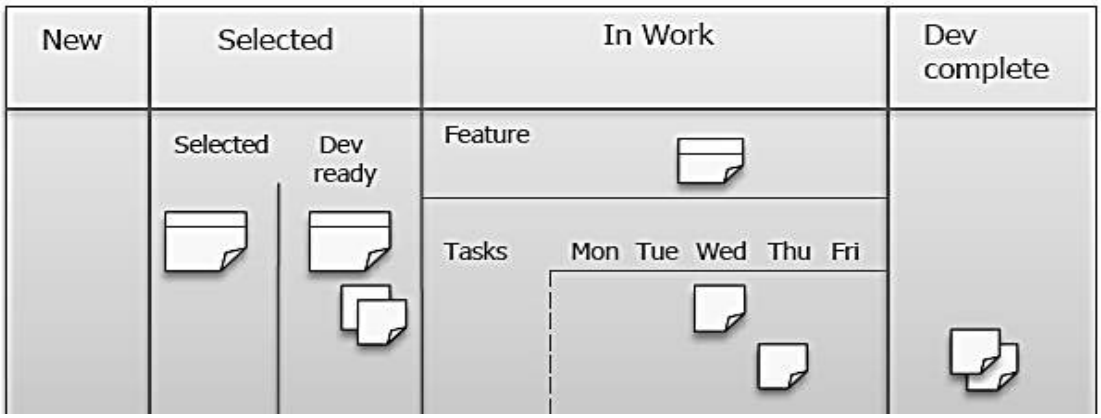


Рисунок 14.5 – Дошка з позначками очікуваних термінів

Якщо важлива послідовність завдань, можна використовувати гібрид Kanban-дошки і діаграми Гантта (рисунок 14.6).

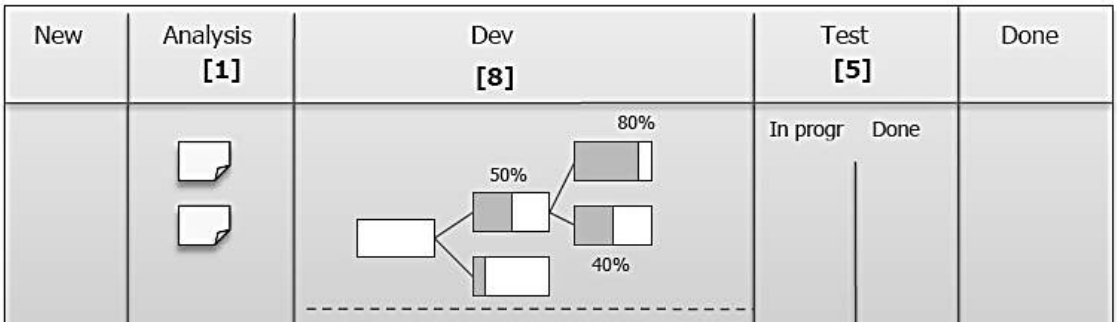


Рисунок 14.6 – Комбінування Kanban-дошки з діаграмою Гантта

Багатошарова Kanban-дошка дозволяє одночасно відстежувати прогрес завдань бізнесу, архітектури та розроблення (рисунок 14.7).

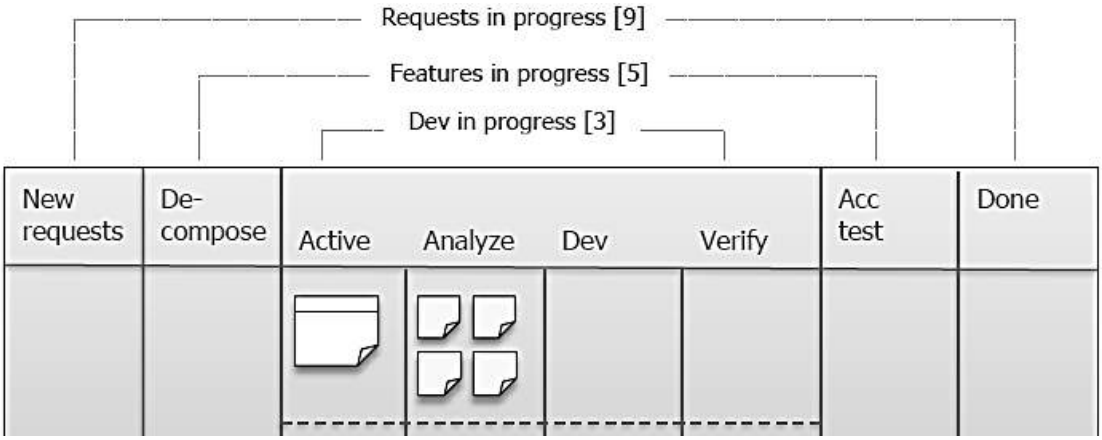


Рисунок 14.7 – Багатошарова дошка для різних рівнів управління

Поділ дошки на зони може бути не тільки вертикальним. Наприклад, можуть бути позначені зони, в які потрапляють завдання, що потребують консультації з клієнтом або з кимось із експертів (рисунок 14.8).

Рисунок 14.8 – Дошка з додатковими зонами

Крім того, можуть бути виділені зони Sometime і Cancelled, куди потрапляють завдання, виконання яких відкладено на невизначений термін або скасовано.

Для позначення нагальних проблем можуть бути введені блокувальні картки (зазвичай це червона картка). Поява на дошці такої картки означає, що треба перервати поточну роботу і розбиратися з проблемою, що виникла (рисунок 14.9).

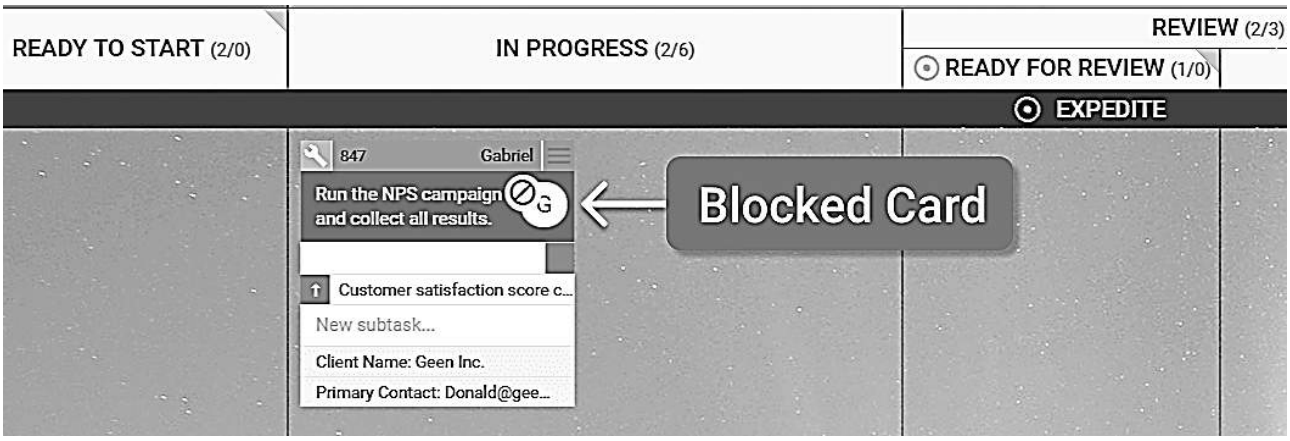


Рисунок 14.9 – Дошка з блокувалною карткою

Картки (Cards, Tickets) для Kanban-дошки в найпростішому випадку можуть являти собою паперові кольорові стікери, на яких від руки пишуться назва роботи та ім'я відповідального. На віртуальних картках може бути розміщено більше інформації: оцінювання складності, пріоритет, зв'язок з іншими завданнями і т. д. Для стильного і гламурного оформлення

фізичної дошки картки друкують на бланках, а аватарки виконавців кріплять на магнітики.

14.2 Покер планування

Покер планування (Planning Poker, Scrum Poker) являє собою спрощену адаптацію методу Делфі.

Метод Делфі або Дельфі (названий на честь Дельфійського оракула, як і СУБД ORACLE і IDE Delphi) запропонований в 1950 роках корпорацією RAND, США для прогнозування впливу майбутніх наукових розробок на методи ведення війни.

Метод Делфі передбачає багатотурові експертні опитування. Експерти відповідають на запитання анкети в письмовому вигляді, даючи свої оцінки їх обгрунтування. Опитування може проводиться поштою. Експерти не зустрічаються один з одним і нічого не обговорюють. Це зроблено, щоб виключити вплив авторитетів, тому що з якогось вузького питання найбільш кваліфікованим експертом може бути не найтитульованіший фахівець.

Потім організаційна група узагальнює отримані відповіді і готує нову анкету, в яку включаються варіанти відповідей, отримані в першому турі, і зазначаються фактори, прийняті до уваги експертами при обгрунтуванні оцінок. При цьому не повідомляється, хто саме дав конкретну оцінку. У деяких варіантах крайні оцінки: найвищу і найнижчу – з подальшого розгляду виключають як аномалії. Експерти дають відповіді повторно. Опитування повторюється кілька разів (турів) і з кожним туром опитування розкид оцінок звужується.

Метод дозволяє отримати узгоджені і обґрунтовані оцінки щодо складних проблем. Однак на проведення опитувань потрібно багато часу (до декількох місяців). Метод Делфі виправданий при дослідженні складних проблем, що потребують великих капітальних вкладень.

Покер планування являє собою просту і швидку процедуру багатотурового опитування. Замість анкет використовуються картки із заздалегідь визначеними значеннями. У кожного своя колода. Члени команди вибирають карти і кладуть їх сорочкою вгору. Потім одночасно відкривають і обговорюють свої оцінки. При необхідності проводиться другий тур оцінювання того ж завдання. Досягнення консенсусу необов'язково, достатньо того, якщо оцінки будуть досить близькими.

Правила гри:

- Грає вся команда. Якщо хтось не прийшов, краще не обговорювати завдання, що потребують його участі.
- Обов'язково участь особи, яка відповідає за вимоги: представник замовника, власник продукту або product manager.

– Власник продукту вибирає елемент для оцінювання з Backloga продукту і читає його опис.
– Члени команди розробки обговорюють елемент (але не його оцінку) і задають уточнюючі питання власнику продукту, який відповідає на них.
– Кожен оцінювач таємно обирає карту, що відповідає його оцінці.
– Усі одночасно відкривають свої карти.
– Якщо всі оцінювачі виберуть однакову карту, значить, уже досягнуто порозуміння, і узгоджене число стає оцінкою даного елемента Backloga.
– Якщо ж оцінки розходяться незначно, вибирається середня оцінка.
– Якщо ж оцінки розходяться істотно, члени команди приступають до обговорення, після чого голосують повторно.
– Вибирається наступний елемент з Backloga – і процедура оцінювання повторюється.

Як правило, обговорення починається з прохання до тих оцінювачів, які поставили найвищу і найнижчу оцінки, обгрунтувати свої оцінки. Можливо, саме вони взяли до уваги важливі фактори, упущені іншими членами команди. Якщо команда грає вперше (взагалі або в даному складі), рекомендується промовляти і обговорювати всі оцінки. Це дозволяє переконатися, що всі користуються однаковими критеріями оцінювання.

Шкала оцінювання

Традиційно як оцінки завдань використовуються числа Фібоначчі. Це дозволяє оцінювати масштаби завдань без високої точності: 8 набагато більше, ніж 3, а ось між 108 і 103 різниця не така істотна, тому для великих значень точність не потрібна. Крім того, числа Фібоначчі, що є сумами один одного, легко декомпозуються.

Як значення в колоді карт найчастіше використовуються:
– послідовність чисел Фібоначчі: 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89;
– видозмінена послідовність, основана на числах Фібоначчі, але містить заокруглені значення (шкала Майка Кона): 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100;
– ступені двійки: 0, 1, 2, 4, 8, 16, 32, 64, 128;
– розміри футболок: M, S, L, XL, XXL;
– оцінка «на пальцях»: кулак, 1, 2, 3, 4 пальці, всі пальці розчепірені, долоня відкрита;
– породи собак: завдання-болонка, завдання-спанієль, завдання-вівчарка, завдання-бульдог і т. д .;
– взагалі без карт; багато розробників розуміють важливість участі команди в плануванні, але самі карти вважають безглуздим

ритуалом; у цьому випадку краще просто обговорити завдання на зборах команди.

Багато наборів карт містять особливі значення:

0 – завдання надто просте або майже закінчене;

∞ – завдання надто складне і повинно бути декомпозованим;

? – недостатньо даних, завдання не підлягає оцінюванню;

«Чашка кави» – потрібна перерва.

Існують web-додатки для гри у покер у розподіленій команді, а також додатки для смартфона: замість паперових карт можна вибрати зображення на екрані і перевернути телефон (рисунок 14.10).

Рисунок 14.10 – Web-додатки, а також додатки для смартфона для гри у покер у розподіленій команді

Рекомендується робити оцінювання не в конкретних одиницях часу, а в абстрактних одиницях роботи – story points. Команда знає або може оцінити з досвіду, скільки story points може бути виконано протягом тижня або місяця. Але індивідуальна продуктивність членів команди може істотно відрізнятися. Тому оцінки «в цілому» досить точні, а «зокрема» – можуть мати дуже великі розбіжності. Саме з цієї причини в деяких випадках використовуються некількісні шкали: футболки, собаки, відра, пальці тощо. Це дозволяє ранжувати завдання за складністю, не маючи точного оцінювання. На практиці числам часто надається конкретний кількісний сенс: години робіт або «ідеальні дні».

14.3 Magic Estimation

Сесія оцінювання складається з таких кроків:

- Закладіть один набір номерів Фібоначчі карт для покеру планування у порядку, на яких є знак запитання. Залиште достатньо місця між картками.
- Дайте кожному члену команди набір приблизно однакової кількості смужок унікальних PBI.
- Поясніть команді процес і правила. Не дозволяється розмовляти або практикувати невербальне спілкування, якщо про це не сказано.
- Якщо продукт не відомий колективу, нехай власник продукту поділиться своїм баченням продукту.
- Нехай кожен член команди, своєю чергою, поміщує кожен PBI свого набору на номер, за яким він оцінює його так, як на рисунку 14.11. Ви можете дозволити члену команди вперше вголос прочитати заголовок PBI. Це коштує додаткового часу, але перевага полягає в тому, що кожен член команди чув про кожен предмет продукту «Блокування продукту». Таким чином команда отримує більше відчуття того, що там є. Усім іншим не дозволяється проявляти будь-яку реакцію. Запишіть число на смужку паперу.
- Якщо хтось не знає, що означає ІПП, його можна поставити під знаком запитання. Це буде розглянуто пізніше на наступному кроці.
- Напишіть також знак запитання на смужку PBI. Можливо, власник продукту хоче переформулювати заголовок після цього, щоб зробити його більш зрозумілим.
- Власники продукції, розміщені на знаку запитання, тепер пояснюються власником продукту. Якщо намір зрозумілий, їх можна розділити між групами, які мають бути розміщені по порядку. Це можна зробити негайно або у другому раунді (див. нижче).
- Тепер розпочинається другий раунд. Це робиться в повній тиші і без невербального спілкування. Кожен член команди може одночасно забрати PBI та замінити його. Він може підтвердити магічну оцінку,

поставивши її на одне число або дати іншу магічну оцінку, поставивши її під іншим числом. Він також записує (нове) число після попередньої магічної оцінки. Таким чином зрозуміло, які PBI вже пройшли другий раунд. Це робиться доти, доки всі PBI не отримають додаткової кількості. Кожен може це зробити зі своєю швидкістю, тому деякі замінять більше PBI, ніж інші. Але сеанс може продовжуватися швидше таким чином.

– Тепер зробіть ще один раунд. Ви можете зробити більше раундів, якщо хочете, але для нас – три роботи добре для отримання достатньо магічних оцінок на PBI, щоб зробити висновок і щоб сесія не була занадто довгою.
– Зверніть увагу і зробіть висновки в два етапи:

Наприклад, для PBI, які незначно змінилися у діапазоні трьох наступних чисел Фібоначчі, обчисліть середнє значення і запишіть це як остаточну магічну оцінку. Цей діапазон може бути [1, 2, 3] або [3, 5, 8]. Якщо Ви вважаєте, що відстань між 3 і 8 занадто велика, Ви також можете зробити діапазон максимум у 3 сюжетні точки, наприклад [2–5].

PBI, які змінилися більше, ніж на попередньому кроці, необхідно обговорити централізовано. Нехай кожен член команди, який написав на ній номер, пояснить причину цього одним реченням. Потім спробуйте обговорити загальну магічну оцінку. Якщо неможливо досягти домовленості, візьміть до уваги магічні оцінки, які залишаються незмінними.

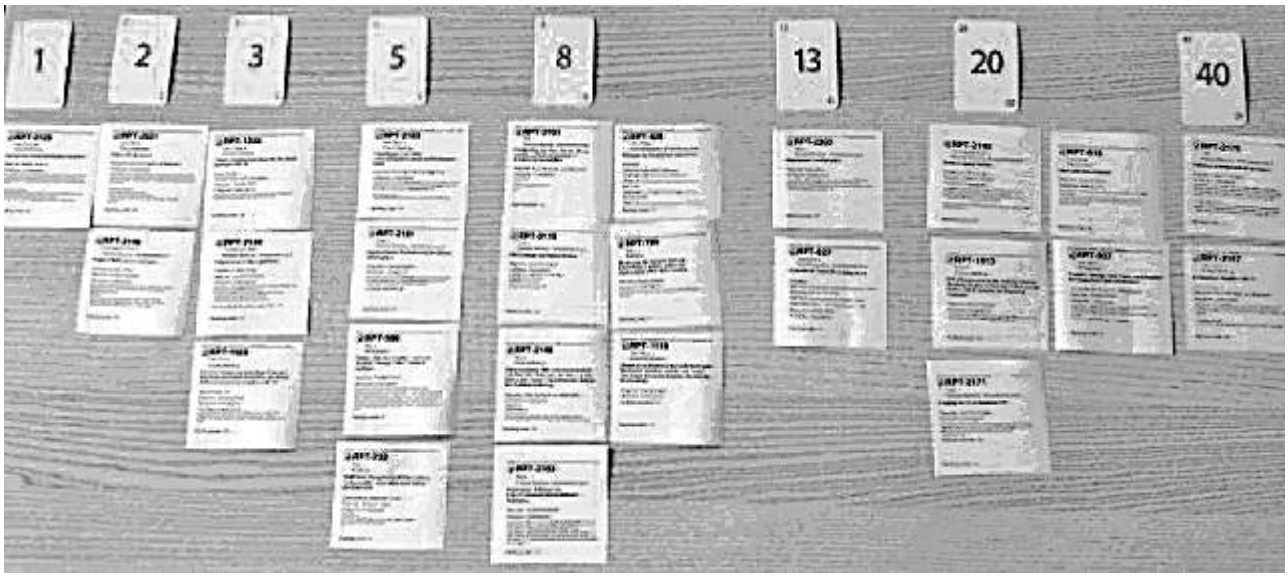


Рисунок 14.11 – Оцінювання за методом Magic Estimation

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Катренко, А. В. Управління IT-проектами : підручник / А. В. Катренко. Кн. 1 : Стандарти, моделі та методи управління проектами / наук. ред. В. В. Пасічник. – Львів : Новий Світ-2000, 2013. – 550 с.
2. Строкань, О. В. Управління IT-проектами : консп. лекцій / О. В. Строкань. – Мелітополь, 2017. – 120 с
3. Руководство к своду знаний по управлению проектами (Руководство PMBOK 4). – 4-е изд. – М. : PMI, 2010. – 496 с.
4. Астахов, А. М. Искусство управления информационными рисками / А. М. Астахов. – М. : ДМК Пресс, 2010. – 312 с.
5. Просницкий, А. Изучение Microsoft Project 2010 за один день методом сквозного примера / А. Просницкий, В. Иванов. – М. : LeoConsulting, 2011. – 35 с.
6. Ройс, У. Управление проектами по созданию программного обеспечения / У. Ройс. – М. : ЛОРИ, 2002 – 424 с.
7. Джекобсон, А. Унифицированный процесс разработки программного обеспечения / А. Джекобсон, Г. Буч, Д. Рамбо. – СПб. : Питер, 2002. – 496 с.
8. Брукс, Ф. Мифический человеко-месяц или как создаются программные системы / Ф. Брукс. – СПб. : Символ-Плюс, 2006. – 171 с.
9. Книберг, Х. Scrum и Kanban: Выжимаем максимум / Х. Книберг, М. Скарин. – 2010 C4Media Inc.
10. Scrum Guide by Ken Schwaber and Jeff Sutherland. https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Ukrainian.pdf

ЗМІСТ

Навчальне видання

Яшина Олена Сергіївна

Пісклова Тетяна Сергіївна

УПРАВЛІННЯ ІТ-ПРОЕКТАМИ

Редактор С. П. Гевло

Зв. план, 2020
Підписано до друку 20.07.2020
Формат 60x84 1/16. Папір офс. № 2. Офс. друк
Ум. друк. арк. 7,1. Обл.-вид. арк. 8. Наклад 50 пр.
Замовлення 179. Ціна вільна

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17
izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001