Тема 12 МЕНЕДЖМЕНТ ПРОГРАМНИХ РОЗРОБОК

12.1. УПРАВЛІННЯ РОЗРОБКОЮ ПРОГРАМНИХ СИСТЕМ

Управління розробкою програмних систем (software management)- Це діяльність, спрямована на забезпечення необхідних умов для роботи колективу розробників програмного забезпечення (ПЗ), на планування та контроль діяльності цього колективу з метою забезпечення необхідної якості ПЗ, виконання термінів та бюджету розробки ПЗ. Часто цю діяльність називають управлінням програмним проектом (software project management). Тут під програмним проектом (software project) розуміють всю сукупність робіт, що з розробкою ПЗ, а хід виконання цих робіт називають розвитком програмного проекту (software project progress).
До необхідних умов роботи колективу відносять приміщення, апаратно-програмні засоби розробки, документацію та матеріально-фінансове забезпечення. Планування та контроль передбачають розбиття всього процесу розробки ПЗ на окремі конкретні роботи (завдання), підбір та розстановку виконавців, встановлення строків та порядку виконання цих робіт, оцінку якості виконання кожної роботи. Фінальною частиною цієї діяльності є організація та проведення атестації (сертифікації) ПЗ, якою завершується стадія розробки ПЗ.
Хоча види діяльності з управління розробкою ПЗ можуть бути досить різноманітними, залежно від специфіки ПЗ, що розробляється, і організації робіт з його створення можна виділити деякі загальні процеси (види діяльності) з управління розробкою ПЗ:
- Складання плану-проспекту з розробки ПЗ;
- Планування та складання розкладів з розробки ПЗ;
- Управління витратами з розробки ПЗ;
- поточний контроль та документування діяльності колективу з розробки ПЗ;
- Підбір та оцінка персоналу колективу розробників ПЗ.

Складання плану-проспекту з розробки ПЗвключає формулювання пропозицій у тому, як виконувати розробку ПЗ. Насамперед має бути зафіксовано, для кого розробляється ПЗ:
• для зовнішнього замовника;
• для інших підрозділів тієї самої організації;
• є ініціативною внутрішньою розробкою.

У плані-проспекті повинні бути встановлені загальні обриси робіт зі створення ПЗ та оцінена вартість розробки, а також матеріально-фінансові ресурси та тимчасові обмеження, що надаються для розробки ПЗ. Крім того, він повинен включати обґрунтування, якого роду колективом має розроблятися ПЗ (спеціальною організацією, окремою бригадою тощо). І, нарешті, мають бути сформульовані необхідні технологічні вимоги (включаючи, можливо, і вибір відповідної технології програмування).
Планування та складання розкладів з розробки ПЗ— це діяльність, пов'язана з розподілом робіт між виконавцями та за часом їх виконання у межах намічених термінів та наявних ресурсів.
Управління витратами розробки ПЗ- Це діяльність, спрямована на забезпечення відповідної вартості розробки в рамках виділеного бюджету. Вона включає оцінювання вартості розробки проекту в цілому

або окремих його частин, контроль за виконанням бюджету, вибір відповідних варіантів його витрачання. Ця діяльність тісно пов'язана з плануванням та складанням розкладів протягом усього періоду виконання проекту. Основними джерелами витрат є витрати на апаратне обладнання (hardware), вербування та навчання персоналу, оплату праці розробників.
Поточний контроль та документування діяльності колективу з розробки ПЗ- Це безперервний процес стеження за ходом розвитку проекту, порівняння дійсних стану та витрат із запланованими, а також документування різних аспектів розвитку проекту. Цей процес допомагає вчасно виявити труднощі та передбачити можливі проблеми у розвитку проекту.
Підбір та оцінка персоналу колективу розробників ПЗ- Це діяльність, пов'язана з формуванням колективу розробників ПЗ. Наявний штат розробників далеко не завжди буде відповідним за кваліфікацією та досвідом роботи для даного проекту. Тому доводиться частково вербувати відповідний персонал та частково організовувати додаткове навчання існуючих розробників. У будь-якому випадку в колективі, що формується, хоча б один його член повинен мати досвід розробки програмних засобів (систем), порівнянних з ПЗ, яке потрібно розробити. Це допоможе уникнути багатьох простих помилок у розвитку проекту.
Забезпечення якості.Якість ПЗ формується поступово в процесі всієї розробки ПЗ у кожній окремій роботі, що виконується за програмним проектом. Не можна вносити зміни щодо покращення якості у вже створену програму.
Для керівництва цією діяльністю призначається спеціальний менеджер, підпорядкований безпосередньо директору, менеджер з якості. Йому безпосередньо підпорядковані бригади, що формуються з контролю якості. Для кожної роботи організується огляд відповідною бригадою. Дивлюся підлягають всі програмні компоненти та документи, що включаються до ПЗ, а також процеси їх розробки. Огляд контролю якості є функцією управління розробкою та пов'язаний з оцінкою того, наскільки результати цієї роботи узгоджуються з декларованими вимогами щодо якості ПЗ.
Для оцінки є програмні стандарти. Вони фіксують вдалий досвід висококваліфікованих фахівців з розробки програмного забезпечення для різних його класів та для різних моделей якості.
Розрізняють два види таких стандартів:
• стандарти ПЗ (програмного продукту);
• стандарти процесу створення та використання ПЗ.
Стандарти ПЗвизначають деякі властивості, якими повинні мати програми або документи ПЗ, тобто визначають якоюсь мірою якість ПЗ. До стандартів ПЗ належать передусім стандарти мови програмування, склад документації, структуру різних документів, різні формати та інших.
Стандарти процесу створення та використання ПЗвизначають, як має проводитися цей процес, тобто підхід до розробки ПЗ, структуру життєвого циклу ПЗ та його технологічні процеси. Хоча ці стандарти безпосередньо не визначають якості ПЗ, проте вважається, що якість ПЗ істотно залежить від якості процесу його розробки.

12.2. СТРУКТУРА УПРАВЛІННЯ РОЗРОБКИ ПРОГРАМНИХ ЗАСОБІВ

Розробка ПЗ зазвичай проводиться в організації, в якій одночасно можуть вестися розробки ряду інших програмних засобів. Для управління цими програмними проектами використовується ієрархічна структура управління (рис. 12.1).

*Мал. 12.1.*Структура управління розробкою програмних засобів

На чолі ієрархії знаходиться директор програмістської організації, який відповідає за управління усіма розробками програмних засобів. Йому безпосередньо підпорядковані кілька менеджерів сфери розробок та один менеджер з якості програмних засобів. В результаті спілкування з потенційними замовниками директор ухвалює рішення про початок виконання будь-якого програмного проекту, доручаючи його одному з менеджерів сфери розробок, а також рішення про припинення того чи іншого проекту. Він бере участь в обговоренні загальних організаційних вимог до програмного проекту та проблем, що виникають, вирішення яких вимагає використання загальних ресурсів програмістської організації або зміни замовником загальних вимог.

Менеджер сфери розробок відповідає за управління розробками програмних засобів (систем) певного типу, наприклад програмні системи у сфері бізнесу, експертні системи, програмні інструменти та інструментальні системи, що підтримують процеси розробки програмних засобів та ін. Йому безпосередньо підпорядковані менеджери проектів, що належать до його сфери. Отримавши доручення директора з виконання деякого проекту, організує формування колективу виконавців з цього проекту, бере участь у обговоренні плану-проспекту програмного проекту, що належить до сфери розробок, яку він відповідає, соціальній та обговоренні та вирішенні виникаючих проблем у розвитку цього проекту. Він організує узагальнення досвіду розробок програмних засобів у його сфері та накопичення програмних засобів та документів для повторного використання.

За кожним програмним проектом призначається свій менеджер, який керує розвитком цього проекту. Йому безпосередньо підпорядковані лідери бригад розробників.

Менеджер проекту здійснює планування та складання розкладів роботи бригад щодо розробки відповідного програмного засобу.

Вважається вкрай недоцільним розробка великої програмної системи однією великою єдиною бригадою розробників. Для цього є низка серйозних причин. Зокрема, у великій бригаді час, що витрачається на спілкування між її членами, може бути більшим за час, що витрачається на власне розробку. Негативний вплив має велика бригада на будову ПЗ і інтерфейс між окремими його частинами. Усе це призводить до зниження надійності ПЗ. Тому зазвичай великий проект розбивається на кілька відносно незалежних підпроектів таким чином, щоб кожен підпроект міг бути виконаний окремою невеликою бригадою розробників (зазвичай вважається, що в бригаді не повинно бути більше 8—10 членів). При цьому архітектура ПЗ повинна бути такою, щоб між програмними підсистемами, які розробляють незалежні бригади, був досить простий і добре визначений системний інтерфейс.

Найбільш часто застосовуються чотири підходи до організації бригад розробників: звичайні бригади; неформальні демократичні бригади; бригади провідного програміста; бригада з контролю за якістю.

У звичайній бригаді старший програміст (лідер бригади) безпосередньо керує роботою молодших програмістів. Недоліки такої організації безпосередньо пов'язані зі специфікою розробки: програмісти розробляють сильно пов'язані частини програмної підсистеми; сам процес розробки складається з багатьох етапів, кожен із яких вимагає особливих здібностей від програміста; помилки окремого програміста можуть перешкоджати роботі інших програмістів. Успіх роботи такої бригади досягається у тому випадку, коли її керівник є компетентним програмістом, здатним пред'являти до членів бригади розумні вимоги та вміє заохочувати добру роботу.
У неформальній демократичній бригаді доручена їй робота обговорюється разом усіма її членами, а завдання між її членами розподіляються узгоджено, залежно від здібностей та досвіду членів бригади. Один із членів такої бригади є керівником бригади. Він також виконує деякі завдання, що розподіляються між членами бригади. Неформальні демократичні бригади можуть успішно справлятися з дорученою їм роботою, якщо більшість членів бригади є досвідченими і компетентними фахівцями. Якщо ж неформальна демократична бригада складається з недосвідчених і некомпетентних членів, у діяльності бригади можуть бути великі труднощі. Без наявності в бригаді хоча б одного кваліфікованого та авторитетного члена, здатного координувати та спрямовувати роботу членів бригади, ці труднощі можуть призвести до невдачі проекту.
У бригаді провідного програміста за розробку дорученої програмної підсистеми несе повну відповідальність одна людина — провідний програміст (chief programmer), який є лідером бригади: він сам конструює цю підсистему, складає та налагоджує необхідні програми, пише документацію до підсистеми. Провідний програміст вибирається з-поміж досвідчених і обдарованих програмістів. Решта членів такої бригади в основному створюють умови для найбільш продуктивної роботи провідного програміста. Організацію такої бригади зазвичай порівнюють із хірургічною бригадою. Ядро бригади провідного програміста складають три члени бригади: крім провідного програміста до нього входить дублер провідного програміста та адміністратор бази даних розробки.
Дублер провідного програміста також є кваліфікованим та досвідченим програмістом, здатним виконати будь-яку роботу провідного програміста, але сам він цю роботу не робить. Головний його обов'язок — знати все, що робить провідний програміст. Він виступає в ролі опонента провідного програміста при обговоренні його ідей та пропозицій, але рішення з усіх питань, що обговорюються, приймає одноосібно провідний програміст.
Адміністратор бази даних розробки (librarian) відповідає за супровід усієї документації (включаючи версії програм), що виникає в процесі розробки програмної підсистеми, та забезпечує членів бригади інформацією про поточний стан розробки. Ця робота виконується за допомогою відповідної інструментальної підтримки комп'ютера. Залежно від обсягу та характеру дорученої роботи до бригади можуть бути включені додаткові члени:
• розпорядник бригади, який виконує адміністративні функції;
• технічний редактор, який здійснює доопрацювання та технічне редагування документів, написаних провідним програмістом;
• інструментальщик, який відповідає за підбір та функціонування програмних засобів, що підтримують розробку програмної підсистеми;
• тестувальник, який готує відповідний набір тестів для налагодження програмної підсистеми, що розробляється;
• один або кілька молодших програмістів, які здійснюють кодування окремих програмних компонентів за специфікаціями, розробленими провідним програмістом.
Крім того, до роботи бригади може залучатись для консультації експерт з мови програмування.
Бригада з контролю якості складається з помічників (рецензентів) за якістю ПЗ. До її обов'язків входять огляди тих чи інших частин ПЗ або всього ПЗ в цілому з метою пошуку проблем, що виникають у процесі його розробки. Дивлюся підлягають всі програмні компоненти та документи, що включаються до ПЗ, а також процеси їх розробки. У процесі огляду враховуються вимоги, сформульовані у специфікації якості ПЗ, зокрема, перевіряється відповідність досліджуваного документа чи технологічного процесу стандартам, зазначеним у цій специфікації. В результаті огляду формулюються зауваження, які можуть фіксуватися письмово або передаватися розробникам усно.

12.3. ПІДБІР КОМАНДИ

Кожна розробка збирає довкола себе команду проекту. Ця команда проекту складається з осіб кількох типів: - кінцеві користувачі; розробники; начальник відділу; начальник відділу інформаційних систем; відповідальний за гарантію якості; група, відповідальна за бета-тестування.
*Кінцеві користувачі*здійснюють введення тестів у систему, що розробляється, забезпечують зворотний зв'язок у проекті інтерфейсу, проводять бета-тестування та допомагають керувати визначенням досягнення кінцевої мети.
*Розробники*відповідають за дослідження, проект та створення програмного забезпечення, включаючи альфа-тестування своєї роботи. Один із розробників є керівником команди проекту, який регулює потік інформації між членами команди.
Начальник відділу відповідає за достовірність даних, що видаються цим відділом інформаційної системи. Крім того, начальник відділу відповідає за те, чи відповідає закінчений додаток поставленим у проекті завданням.
Начальник відділу інформаційних систем визначає цілі (плани) для розробників, засновані на інформації, отриману від менеджерів інших відділів та головного управління.
Крім того, встановлює пріоритети між проектами та працює як джерело інформації між відділами та між розробниками, розподіляє обсяги ресурсів, необхідних для виконання кожного проекту.
Відповідальним за гарантію якості є одна людина, але стежать за якістю усі члени команди проекту. Відповідальний стежить, щоб проект програми досяг намічених цілей; проект програми відповідав опису системи; план тестування та план перетворення даних відповідали вимогам; стандарти розробки інформаційних систем дотримувалися.
Група відповідальних за бета-тестування складається з програмістів та можливих кінцевих користувачів та здійснює два типи тестування. Перший тип використовує плани спеціального тестування, які розроблені відповідальним за гарантію якості. При другому типі використовуються тести, які розробили відповідальні за бета-тестування, застосовуючи критерії відповідального за гарантію якості.
Незалежні консультанти зазвичай концентрують увагу до вартості проекту. Часто вони не беруть до уваги витрати на проведення системного аналізу та розроблення проекту та дають неправильну оцінку часу реалізації даного проекту. Хоча відомо, що необхідно виконати детальний аналіз завдання перед тим, як проект буде затверджений, користувачі не схильні витрачати додаткові кошти на дослідження. На жаль, це часто призводить до великої кількості труднощів у процесі розробки, інколи ж до розвалу всього проекту.

12.4. МЕТОДОЛОГІЯ УПРАВЛІННЯ ПРОЕКТОМ

Взаємодія у команді. Відповідальність за проект несе розробник, а чи не начальник відділу. Начальники є членами команди — останнє слово у деяких важливих питаннях належить їм. Проте насправді проектом керує розробник. Коли розробники виконують «власний» проект, їхній інтерес в успіху цього проекту і, отже, ймовірність його успішного виконання збільшується в геометричній прогресії.
Якщо розробники повністю вивчать типи зв'язків між усіма частинами проекту (на відміну від його частини), їх кваліфікація підвищиться. Це забезпечує тип перехресного навчання у дуже важливих чинниках успішної розробки програмного забезпечення. Чи означає це, що розробник виконує менше власне програмістської роботи? Звісно. Коли розробнику потрібна допомога у програмуванні для конкретного проекту, залучається відділ інформаційних систем та сторонніх розробників, які співпрацюють за контрактом.
Усі члени команди точно знають свої обов'язки. Це досягається з допомогою зустрічей (сесій), у яких обговорюються окремі частини проекту. Кожного моменту часу на певній стадії проекту всі члени команди знають точно, що вони мають робити. Це досягається за допомогою постановки завдань та визначення локальних завдань.
Певна методологія розробки програмне забезпечення усуває розбіжності та відсутність зв'язку між членами команди розробників та між програмістами та кінцевими користувачами.
Нові люди «безболісно» підключаються до проекту на будь-якій стадії розробки.
Кінцева програма базується на фундаментальному аналізі завдання, що дозволяє звести до мінімуму витрати на подальше доопрацювання, модифікацію та супровід продукту.

У процесі розробки створюється потужний пакет документації, що дозволяє спростити неминучий супровід і доповнення програмного продукту.

12.5. СКЛАДНІ МЕТОДОЛОГІЇ РОЗРОБКИ

Отримавши певне уявлення необхідність розгляду методології управління проектом, розглянемо окремі її складові: попередній аналіз; чітке формулювання мети; складені моделі даних та словники; вихідні форми; безпека системи та даних; платформа та оточення; контингент майбутніх користувачів.
Попередній аналізє дуже важливим етапом. Ви повинні бути впевнені, що маєте всю необхідну інформацію про клієнта перед тим, як візьметеся за реалізацію проекту.
Чітке формулювання мети має відповідати на запитання: «Що система має робити?»; «Чи було чітко сформульовано мету створення системи?»; Чи знає кінцевий користувач, що система дійсно повинна робити? Звичайно, дуже важливо знайти справжню мету програми, щоб мати можливість визначити межі проекту. Це необхідно зробити настільки швидко, наскільки це можливо.
Моделі даних та словники необхідні для того, щоб дані, що обробляються в додатку, були виділені та визначені у поняттях, доступних як кінцевим користувачам, так і команді розробників. Часто трапляється, що заздалегідь не існує будь-якої моделі даних, що сформувалася, і проектувальник повинен створити словник і модель даних, а потім повернутися до користувача і обговорити з ним розроблену схему, щоб користувач розумів її.
Вихідні форми. При попередньому опитуванні користувача необхідно зробити начерки всіх вихідних форм, оскільки може знадобитися додаткове нарощування словника задля забезпечення реалізації тієї чи іншої.
Безпека системи та даних. Перш ніж розпочати розробку, кінцевий користувач повинен визначити необхідність забезпечення безпеки системи та даних. Включення системи забезпечення безпеки має розглядатися на ранній стадії проектування.
Платформа та оточення. Важливо оцінити оточення, у якому працюватиме система. Клієнти витрачають великі кошти на придбання апаратних засобів до того, як звертаються до вас. Ви повинні з'ясувати всі деталі: про мережні апаратні та програмні ресурси; про типи комп'ютерів; про операційну систему; типи принтерів, моніторів, дисководів, інших периферійних пристроїв.
Контингент майбутніх користувачів. Часто поняття «хто» значно важливіше за поняття «що». Хороше розуміння категорій кінцевих користувачів може дати вам важливу стартову інформацію для створення проекту. Ви повинні постійно вивчати, що хочуть ваші кінцеві користувачі. Різні типи груп користувачів мають різні вимоги, які повинні бути враховані при проектуванні програмного забезпечення.
Якщо ви робите програми для спільного ринку, то можете створити лише дуже грубе уявлення про кінцевого користувача. Якщо ви пишете будь-яку загальну програму обліку, то можете лише припустити, що мій клієнт має загальне уявлення про комп'ютер і він має необхідність що-небудь враховувати.
Якщо ви пишите програму підтримки офісу бюро подорожей та екскурсій, то знаєте, що кінцеві користувачі будуть використовувати цей додаток саме в цій галузі. Таким чином, ви можете оптимізувати систему обліку та розрахунків з огляду на конкретну специфіку. Однак ви не знаєте ні досвіду роботи кінцевих користувачів з обчислювальною технікою, ні рівня їхнього професіоналізму в їхньому власному бізнесі.
Якщо ви пишете додаток користувача, наприклад офісну систему для офісу «Іванів і сини», то можете безпосередньо спілкуватися з кінцевими користувачами і з'ясовувати їх рівень пізнання обчислювальної техніки та досвід роботи у власному бізнесі, що дасть можливість розробнику заздалегідь передбачити більшість конфліктних ситуацій між вашим додатком та кінцевими користувачами.
Що чекають на вас кінцеві користувачі?
Кожна група кінцевих користувачів має різні вимоги та очікування від вашої системи. Перед проектуванням системи необхідно з'ясувати, на що розраховує кінцевий користувач. Необхідно звернути увагу на такі аспекти: початкове обстеження та складання технічного завдання, інсталяція, навчання, підтримка, допомога в експлуатації.
Резюме Як проектувальник системи, ви повинні повернутися на рівень попереднього аналізу завдання та переконатися, що вся необхідна інформація отримана. У разі недотримання цієї вимоги ви можете

значно уповільнити реалізацію проекту внаслідок багаторазового повторного звернення до користувача за уточненням неправильно трактованих деталей та необговорених умов.

12.6. АНАЛІЗ ПОБАЖАНЬ І ВИМОГ ЗАМОВНИКА

Існує величезна прірва між ідеями користувачів та уявленням про можливі способи реалізації цих ідей конкретними розробниками. Містом між цими двома поняттями має бути первинний етап обстеження проекту та складання технічного завдання на даний проект. Це завдання ділиться на три стадії: вивчення вимог замовника, уточнення функціональної специфіки задачі та технічне проектування задачі.

Аналіз вимог та побажань замовникапочинається з отримання замовлення на нову розробку (або модифікацію існуючої) і закінчується складанням документа, в деталях описує цю розробку. Це повинен бути інтерактивний процес, в результаті якого з'являється документ, що повністю описує завдання і задовольняє обидві сторони, що включає розгляд всіх проблем і задач, що вирішуються, безліч аркушів з вимогами і побажаннями замовника та іншу необхідну інформацію.

Найважливіша мета, якої необхідно досягти на цьому першому етапі, — знайти і зрозуміти, що ж НА САМОМ СПРАВІ ХОЧЕ КОРИСТУВАЧ. Іноді зробити це не так просто, оскільки користувач не завжди точно уявляє, ЩО він дійсно хоче отримати. Банальним прикладом можуть бути користувачі, які замовляють, наприклад, одночасно кілька великих завдань типу «Облік заробітної плати», «Ведення складського обліку», «Складання табеля» тощо, називаючи все це «Бухгалтерією». Якщо проігнорувати даний етап, то проект може зрештою бути засуджений на велику кількість доопрацювань, добудовування коду «на коліні» та неодмінне сидіння програмістів у вихідні, щоб зробити клієнту дійсно те, що він хоче і що не було заздалегідь обумовлено.

Очевидно, що будь-який проект розпочинається з ідеї. Як тільки з'являється ідея, одна чи кілька людей починають її розвивати. Ці люди — замовники чи потенційні користувачі. Вони визначають початкові вимоги та приймають рішення про створення того чи іншого програмного продукту. Таким чином, необхідно з'ясувати, що ці люди хочуть отримати від програмного продукту.

Перед початком обговорення майбутнього проекту дуже важливо переконатися, що по обидва боки столу переговорів сидять саме ті люди, які потрібні для спільного обговорення проекту. Три найбільш поширені помилки припускаються на даному етапі.

Помилка 1. Користувачі, які починають обговорення проекту, не є людьми, які прийматимуть остаточне рішення про вимоги до обговорюваної системи (тобто вони не є людьми, які мають повне уявлення про завдання, що ними описується).

Помилка 2. Учасники обговорення з боку розробників не є людьми, які стосуються технічної розробки проекту.

Помилка 3. Технічні фахівці не розуміють користувачів (або не докладають зусиль до розуміння), або розробники погано знаються на діловодстві та бізнесі, або вони останню частину життя провели не відходячи від монітора і можуть розмовляти тільки мовою бітів і байтів.

У першому випадку користувачі, які беруть участь в обговоренні проекту, описують усі, з їхньої точки зору, деталі майбутнього завдання і залишаються задоволеними думкою, що вони точно виклали всі вимоги та побажання до завдання. На жаль, якщо користувачі, які брали участь в обговоренні, не є кінцевими користувачами даної системи, то після складання кінцевого документа, який в деталях описує задачу, що вирішується, у людей, які підписують цей документ, виникають питання і будь-які нові пропозиції щодо вдосконалення окремих деталей або їхню зміну. Ця ситуація виникає у більшості подібних випадків. Така ситуація відкидає процес розробки програми стадію аналізу майбутнього проекту. Наявна втрата часу та коштів.

Аналогічна проблема виникає за участю у складанні проекту людей, які ніколи не використовуватимуть створювану програму. Це загальна проблема проектування програмного забезпечення. Коли весь проект розроблений, реалізований, відтестований і представлений замовнику, кінцеві користувачі, ті, хто дійсно використовуватиме створений додаток, з'ясовують, що він, скоріше, перешкода, ніж допомога в їх роботі.

Третя з описаних вище проблем полягає в тому, що користувачі, що пред'явили мінімальні вимоги до системи на стадії системного проектування і розробку проекту на розгляд виробника, починають

обурюватися, що продукт не задовольняє тим чи іншим вимогам, а тому працює некоректно і вимагає переробки.
Ще однією поширеною помилкою є вибір керівника проекту, який не має відповідних технічних знань для реалізації даного проекту. Ця проблема зазвичай зустрічається при розробці великих проектів, де потрібна велика команда програмістів. Часто існує технічний лідер, який може керувати проектом так само добре, як вирішувати технічні питання. Використання його як менеджера проекту краще, ніж використання простого адміністратора.
У процесі аналізу вимог замовника важливо, щоб у переговорах взяв участь один із членів команди розробників, а в кращому разі провідний технічний спеціаліст або технічний керівник проекту. На жаль, досить важко зібрати в одній кімнаті та одночасно всіх людей, яким необхідно брати участь в обговоренні проекту.
Якщо у процесі обговорення бере участь лише адміністративна особа або керівник проекту, далекий від проблем безпосереднього кодування, виникає безліч проблем і питань, пов'язаних з можливою оптимізацією окремих операцій, створенням словника баз даних, системними вимогами до створюваного програмного забезпечення і т. д. у разі окремим членам доводиться повторно спілкуватися з кінцевими користувачами для з'ясування неврахованих чи погано продуманих питань, що врешті-решт може зіпсувати відносини між командою розробників та кінцевими користувачами. З іншого боку, участь в обговоренні проекту технічних фахівців може призвести до помітного спрощення проекту за рахунок приведення окремих вимог користувача до існуючих і раніше розроблених технологій задоволення цих вимог. Коли необхідний технічний персонал просто не може бути присутнім на всіх засіданнях обговорення проекту або технічний фахівець повинен тимчасово перейти на інші дії в процесі обговорення, може допомогти так званий протокол засідань. Цей протокол містить нотатки про всі питання, що обговорюються на цьому засіданні, а також імена, посади та телефони учасників обговорення як з одного, так і з іншого боку. Цей протокол також має містити інформацію про прийняті рішення, обумовлені нюанси та будь-які деталі, обговорення яких уже проводилося. Зрештою, дані нотатки повинні перерости в кінцевий документ, який описує результат, отриманий у процесі обговорення всіх частин і деталей проекту.
Якщо зазначені рекомендації будуть дотримані, то технічна сторона розробки буде розглянута повніше, що дозволить згодом уникнути багатьох помилок, пов'язані з нерозумінням тій чи іншій стороною технічних особливостей проекту.
Уникайте програмістів, які можуть просидіти кілька діб над створенням функції, яка фактично не потрібна кінцевому користувачеві. Програмісти, які не вміють працювати з користувачами, не розуміють питань, пов'язаних зі специфікою роботи кінцевого користувача, або не вміють викласти більш менш складні технічні питання простою російською мовою, не повинні брати участь у процесі обговорення проекту. Вони можуть створити додаткові труднощі технічного та тимчасового характеру, починаючи детально з'ясовувати несуттєві питання.
На жаль, багато програмістів не дуже добре розуміються на навколишньому діловому світі. Їхня спеціалізація — комп'ютери та програми, а не створення кінофільмів або управління госпітальним господарством. Виникає питання: «Чи справді необхідно команді розробників детально розбиратися у діловодстві та специфіці бізнесу кінцевих користувачів?» Недосвідчений програміст подумає: «Користувачі - професіонали у своїй галузі, я - професіонал у своїй; якщо ми почнемо навчати один одного нашим професіям, чи знадобимося ми один одному врешті-решт?»
Неправильно. Професійні знання у тій чи іншій галузі не набуваються у процесі спільного обговорення будь-якого проекту. Не купуються вони і в процесі написання програми на задану тематику. Професійні знання часто набуваються в процесі багатьох років навчання, що йдуть за не менш тривалим періодом спроб і помилок.
Коли користувачі намагаються описати, що вони хочуть від окремих частин програми, не треба відразу переводити їхні побажання до коду. Необхідно зрозуміти, що хоче користувач, а потім постаратися зробити це найбільш оптимальним способом.
Оптимальний варіант, коли користувач має уявлення про технічну сторону обговорюваного завдання, а команда програмістів має досвід у сфері діяльності користувача. Коли поєднуються такі якості користувача та розробника, приблизно половина питань одразу знімається з обговорення за непотрібністю.

12.7. АНАЛІЗ ВИМОГ ДО ПРОЕКТУ

Одне підключення до процесу розробки необхідних осіб з обох сторін не спричинить створення повноцінного документа, що описує завдання. Важливо зберегти простоту процесу аналізу вимог та уникати обмірковування, як буде реалізована та чи інша функція чи процедура. Необхідно пам'ятати, що аналіз вимог замовника може тривати від двох годин до кількох тижнів, залежно від складності завдання.
Може існувати велика кількість способів розпочати та проводити аналіз вимог, але всі вони повинні призводити до одного й того самого результату - складання документа, що описує всі вимоги та побажання користувача.
Найпростіший спосіб – почати обстеження зверху вниз. Що головна мета системи? Визначення основних компонентів системи може бути корисним для введення користувача в потрібне русло обговорення проблеми. Майже всі системи вимагають введення певної інформації та виведення якихось звітних форм (у вигляді звітів та запитів), деякого виду конфігурації, можливості імпорту та експорту даних, архівування та, можливо, сервісний розділ. Виходячи з цих даних, можете отримати інформацію про те, що має перебувати в головному меню програми, та обміркувати деякі деталі розробки ще до повного визначення проекту.
Незалежно від прийнятого підходу до розгляду вимог користувача результатом аналізу має бути ясне розуміння того, що вимагає користувач і що хоче. Тонка відмінність між цими двома поняттями є важливою. Вимоги користувача обмежуються його уявленням про запропоноване їм завдання. Ці вимоги користувач явно обговорює у процесі дискусії. Побажання користувача нерідко залишаються за кадром не тому, що користувач не обговорює їх спеціально, а тому, що він підсвідомо вважає деякі вимоги природними і не вимагають спеціального виділення.
Головна мета цього етапу — переконатися, що ви розумієте потреби користувача та пріоритети напрямів розробки.
Далі слідує функціональна специфікація - це міст між початковим оглядом вимог і технічною специфікацією, що розробляється пізніше. Документ повинен складатися з логічних розділів типу короткого огляду системи, що супроводжується коротким описом основних фрагментів або функціональних об'єктів. Демонстрація планованих екранних форм має показувати основні напрями дій із головними функціональними об'єктами та модулями програми. Розділ опису звітів повинен містити всі звітні форми, які ви плануєте створювати. У великих системах основні модулі можуть бути розбиті на простіші з описом того, що ці, простіші модулі будуть робити.
Плануйте цей документ таким чином, щоб користувач, який не зацікавлений у розгляді детальних особливостей системи, міг би прочитати лише першу частину документа з описом основних функцій системи. Користувачі, зацікавлені у розгляді більш детальних деталей, можуть читати документ далі.

12.8. ВИМОГИ КОРИСТУВАЧА

Ваші специфікації мають відповідати всім вимогам користувачів. Переконайтеся, що знову зверніться до початкового аналізу перед завершенням специфікації, що враховані всі вимоги та запити користувачів. Якщо вимога користувача не може бути задоволена, поясніть чому, а не просто виключіть її зі специфікації.
Ви також повинні обговорити з користувачем обмежені ресурси, які є у користувача. Дев'яносто дев'ять відсотків проблем, що виникають під час програмування, можуть бути вирішені шляхом використання специфічних зовнішніх пристроїв, драйверів та сторонніх програм.
Припустимо, функціональна специфікація розроблена, підписана та покладена на полицю. Але вона може виявитися цілком марною з низки причин. При неправильному ставленні до розробки функціональної специфікації вона може бути погано написана, погано організована або, що найімовірніше, обтяжена томами опису непотрібних технічних подробиць. Іншими словами, працювати з таким документом буде неможливо.
Однією з найнебезпечніших хвороб розробки програм є синдром «повзуючого проекту», або «зсуву ґрунту». Він проявляється, коли функціональна специфікація неповно розглядає окремі аспекти проекту. У цьому випадку, в міру створення системи, користувачі, розглядаючи окремі готові модулі, будуть просити внести деякі вдосконалення, посилаючись на неясні описи цього модуля

функціональної специфікації. Поступово система набуватиме вигляду величезного динозавра в латках, оскільки глобальні зміни розроблених структур програми проводити вже не можна, а зміни та вдосконалення необхідно вносити. Це може призвести до перевитрати тимчасового ліміту на створення окремих модулів та нестабільності роботи системи через випадання окремих функціональних шматків програми із суворої загальної схеми усієї системи.
Швидке макетування – метод проектування, розробки та зміни інтерфейсів користувача «на льоту». Кінцеві користувачі повинні активно включатися в цей процес, оскільки розробка інтерфейсу разом з користувачем відбувається значно швидше, ніж без нього. Спільна технологія дає можливість «підігнати» інтерфейс під користувача за кілька коротких сесій. Для цього є спеціальні засоби, зокрема CASE-засоби. З потужними CASE-засобами процес розробки програм помітно спрощується. Проектувальник використовує програмні засоби для створення та компонування словників даних, потоків даних та діаграм об'єктів, а в деяких випадках прототипів процесів обробки даних та функціонального коду.
Однак використання CASE-засобів розробки програм не дуже поширене у сфері розробки промислових програм. Це відбувається з двох причин. По-перше, це обмеженість можливостей CASE-систем. По-друге, якщо CASE-система досить потужна і багатофункціональна, вона вимагає великих тимчасових витрат за її освоєння.
Наприкінці цього етапу, якщо було написано хорошу, легко розуміємо, неперевантажену і непусту функціональну специфікацію, системний аналітик чи технічна група зможе перейти до наступного етапу — створення технічної специфікації, — ґрунтуючись на інформації, отриманій на всіх попередніх етапах.

12.9. ТЕХНІЧНЕ ПРОЕКТУВАННЯ

Технічне проектування - це міст між функціональною специфікацією та фактичною стадією кодування. Часто команда розробників намагається скоротити та об'єднати стадію розробки функціональної специфікації та технічне проектування та розробити один документ. Це є помилка.
По-перше, користувач читатиме документ із великою кількістю незрозумілих йому технічних подробиць. У цьому випадку користувач відкине ваш документ, що може призвести до недостатньої закінченості цього документа.
По-друге, якщо функціональна специфікація концентрує увагу вимогах і побажаннях користувача, то технічне проектування має орієнтуватися створення методів реалізації даних вимог. Тільки після того, як обидві ці фази завершені та акценти розставлені, програміст може приступати до безпосереднього кодування.
Коли обидві ці стадії об'єднані, розробник неспроможна сконцентруватися на якомусь напрямі мислення й у результаті виходить неясний і погано відпрацьований документ. Або ще гірше, програміст починає реалізовувати ідею, яка ще не визначена до кінця користувачем.
Середовище розробкидозволяє всім членам команди знати розміщення всіх файлів проекту, бібліотек, документів та іншої пов'язаної з проектом інформації. Вона повинна бути створена таким чином, щоб усі члени команди розробників з мінімальними витратами часу могли звернутися до будь-якої інформації щодо проекту.
Створення тимчасової діаграми проекту є найважливішим етапом робіт, на якому необхідно скласти детальний розклад термінів початку та закінчення розробки кожного модуля, частин проекту та всього проекту загалом. Необхідно враховувати час, який витрачається на додаткові контакти із замовником, розробку специфічних інструментальних засобів, а також можливі проблеми, пов'язані з непередбаченими обставинами (наприклад, хвороба співробітника або часткова втрата даних внаслідок збоїв апаратного забезпечення).

12.10. РЕАЛІЗАЦІЯ

Зазвичай на етапі кодування спливають усі неприємні проблеми, які тільки можна собі уявити. Чим більший проект, тим більше проблем. Ось чому перші три кроки такі важливі.
Якщо всі з описаних вище кроків повністю пройдені, то реалізація програми значно спрощується. В ідеалі всі потенційні вузли та пастки мають бути передбачені та обійдені. Технічна специфікація може і

повинна бути передана команді програмістів, які виконують безпосереднє кодування, щоб вони могли записувати код згідно з детальним проектом завдання. Будь-які проблеми, що виникають на цьому етапі, повинні відстежуватися програмістами і поміщатися у документи, що стосуються проблеми, щоб відображати всі зміни, що виникають.
Огляд кодуробиться програмістами - кодувальниками програм на спеціальній сесії (зустрічі). Як і на етапі огляду проекту, під час огляду коду «відловлюється» велика кількість неточностей та помилок, виявляються неоптимальні ділянки програми. Огляд коду дозволяє побачити різним членам групи розробників фактичний код, виконаний колегами за проектом. Оскільки програмування є творчим процесом, кожен член команди представляє бачення однієї й тієї проблеми по-різному. Хтось вирішує це питання краще, хтось гірше. Огляд коду дозволяє виявити гірше написані ділянки програми та за необхідності переписати їх, скориставшись порадою досвідченішого члена команди. Також розгляд різних прийомів, технологій та підходів до програмування дозволяє скористатися ними для вирішення майбутніх проблем у подальших проектах. Особливо це корисно для новачків команди, хоча, як відомо, «навіть старого собаку можна навчити новим трюкам».

12.11. СИСТЕМНЕ ТЕСТУВАННЯ

Стадія тестування системи починається після того, як більшість модулів системи вже завершено. Тестування може складатися з трьох окремих фаз:
- Системний тест, або лабораторні випробування;
- Досвідчена експлуатація;
- приймальний тест.
Альфа-тест(Лабораторні випробування). Ця фаза тестування має дві мети. По-перше, цей тест повинен підтвердити, що всі фрагменти правильно інтегровані у систему. Це дозволяє групі тестування розпочати повне тестування усієї системи. Зазвичай використовується деяка однорідна технологія тестування всім компонент системи, що дозволяє визначити відповідність всіх частин певним, заздалегідь передбаченим параметрам. Один із шляхів створення сценаріїв тестування — створювати методи тестування у процесі безпосереднього кодування.
*Лабораторне тестування*остання можливість розробників виправити всі виявлені помилки, перш ніж система буде передана кінцевим користувачам. Бета-тестування — це не та стадія, на якій програмісти хотіли б виявляти серйозні збої розробленої системи, тому лабораторне тестування має проходити максимально повно. Якщо альфа-тестування проведено неякісно, загальний процес тестування може зайняти тривалий час, оскільки виправлення помилок, виявлених на наступних стадіях тестування, займає значно більше часу через неможливість їх виправлення «на льоту». Будь-які виявлені проблеми повинні протоколюватися, щоб хронологія проблем та їх усунення була доступна при виникненні наступних питань про проблеми, що існували раніше.
Бажано, щоб програмне забезпечення не передавалося для дослідної експлуатації, доки всі відомі проблеми не будуть вирішені. Насправді програмне забезпечення часто випускається для бета-тестування з вже знайденими, але ще не вирішеними проблемами через брак часу і закінчення термінів розробки проекту. Це недопущення призводить до значних непередбачених затримок, пов'язаних з труднощами подальшого тестування через наявність будь-яких помилок або неточностей.
Бета-тестування-це наступна фаза загального тестування, коли програмне забезпечення поставляється обмеженому колу кінцевих користувачів більш жорсткого тестування. Добре відомо, що користувачі іноді використовують програмне забезпечення не для тих цілей, для яких воно призначалося. Тому вони часто можуть знаходити помилки в тих місцях програми, над якими протягом цього часу проводилися лабораторні випробування, які не знайшли жодних порушень. Це слід очікувати і не заперечувати можливості повернення до попередньої фази — лабораторного тестування. У цих випадках часто допомагають протоколи виявлених та фіксованих помилок.

12.12. ПРИЙМАЛЬНИЙ ТЕСТ

Приймальний тест.Приймальний тест стає простою формальністю, якщо попередні стадії тестування успішно завершено. Використовуючи інформацію про те, що всі виявлені помилки вже виправлені, приймальний тест просто підтверджує, що нових проблем не виявлено і програмне забезпечення готове для випуску. Очевидно, що чим більше існує реальних чи потенційних користувачів вашого продукту, тим важливішим є приймальний тест. Коли виробляється промислове тиражування і розсилання більше трьох сотень тисяч дисків, хочеться подвійно переконатися, що програма написана без помилок і всі явні та неявні проблеми було вирішено. Звичайно, якщо попередні стадії не пройшли успішно, то приймальний тест є єдиною можливістю запобігти витратам на зміну та доповнення продукту, що поставляється.

12.13. ПІСЛЯРЕАЛІЗАЦІЙНИЙ ОГЛЯД

Даний етап – найкраща можливість здійснити огляд створеного програмного забезпечення, перш ніж буде розпочато новий проект. Типові питання, що виникають після здачі програмного проекту користувачу-замовнику:
• Що ми робили правильно?
• Що ми робили неправильно?
• Які етапи були найкориснішими, а які непотрібними?
• Чи не було що-небудь на якомусь етапі розробки, щоб допомогло вдосконалити програмний продукт?

12.14. СУПРОВІД ПРОГРАМ

Супровід програм - "ложка дьогтю" для кожного програміста. Це завжди перешкода на початку розробки будь-якого нового проекту, що змушує відволікатися від розробки проекту та повертатися до старих програм та старих проблем. Ніщо не робить супровід настільки непривабливим, як погано документований код, недостатньо повне початкове проектування та відсутність зовнішньої документації.
Якщо більшість кроків розробки виконано правильно, супровід не викликатиме серйозних проблем, а буде елементарною технічною підтримкою та модифікацією програмного забезпечення.

ВИСНОВКИ

• Розробка програмних систем – складний захід. Можна виділити такі загальні процеси з управління розробкою ПЗ: складання плану-проспекту з розробки ПЗ - планування та складання розкладів з розробки ПЗ; управління витратами розробки ПЗ; поточний контроль та документування діяльності колективу з розробки ПЗ; підбір та оцінка персоналу колективу розробників ПЗ.
• Зазвичай, в організації одночасно розробляється кілька програмних проектів. Для оптимальної якості та швидкості роботи необхідно правильно структурувати управління організацією.
• Кожна розробка проекту збирає довкола себе команду фахівців (команду проекту), що складається з кінцевого користувача; розробників; начальника відділу; начальник відділу інформаційних систем; відповідального за гарантію якості; групи, відповідальної за бета-тестування
• Необхідно дотримуватись методології управління проектом, яка ділиться на складові: попередній аналіз; чітке формулювання мети; складені моделі даних та словники; вихідні форми; безпека системи та даних; платформу та оточення; контингент майбутніх користувачів.
• Різниця між поняттями «бажання замовника» та «кінцевий продукт» зазвичай дуже велика. Мостом для їх з'єднання має бути первинний етап обстеження проекту та складання технічного завдання на цей проект. Це завдання ділиться на три стадії: вивчення вимог замовника, уточнення функціональної специфіки задачі та технічне проектування задачі. Якщо говорити про вимоги користувача, то їх необхідно дотримуватися неухильно.
• Технічне проектування — своєрідний міст між функціональною специфікацією та фактичною стадією кодування. Це вкрай важлива стадія і недбало до неї ставитися не можна.
• Системне тестування може складатися із трьох окремих фаз: системний тест або лабораторні випробування; дослідна експлуатація; приймальний тест.
• Супровід - нелюбима програмістами, але необхідна частина, що дає можливість для вдосконалення продукту.

Контрольні питання

1. Що таке програмний проект?
2. Що включає складання плану-проспекту з розробки ПЗ?
3. Назвіть основні джерела витрат розробки ПЗ.
4. Перерахуйте обов'язки членів ядра бригади програмістів.
5. Чим займаються незалежні консультанти?
6. Назвіть складові методології розробки.
7. У чому полягає аналіз вимог та побажань замовника?
8. Що таке швидке макетування?
9. У чому полягає технічне проектування?
10. Назвіть три фази тестування. І. Навіщо потрібен приймальний тест?
12. Назвіть фактор, який ускладнює супровід найбільшою мірою.