**INT**

***Theoretical concepts of testing***

***Теоретичне розуміння:***

* **\*Розуміє поняття «якість».**
* **\*Вміє навести приклади застосування видів та типів тестування, розуміє коли їх доцільно застосовувати.(Навести приклади з досвіду роботи на проекті).**

**Види тестування програмного забезпечення:**

Існує кілька ознак, за якими прийнято робити класифікацію видів тестування. Зазвичай виділяють такі:

*За ступенем автоматизації:*

* Ручне тестування (manual testing)
* Автоматизоване тестування (automated testing)
* Напівавтоматизоване тестування (semiautomated testing)

*За ступенем підготовленості до тестування:*

* Тестування по документації (formal testing)
* Тестування ad hoc або інтуїтивне тестування (ad hoc testing) — тестування без тест плану та документації, що базується на методиці передбачення помилки та власному досвіді тестера.

*За знанням системи:*

* Тестування чорного ящика (black box)
* Тестування білого ящика (white box)
* Тестування сірого ящика (grey box)

*За ступенем ізольованості компонентів:*

* Компонентне (модульне) тестування (component/unit testing)
* Інтеграційне тестування (integration testing)
* Системне тестування (system/end-to-end testing)

*За часом проведення тестування:*

* Альфа-тестування (alpha testing)
* smoke testing
* Тестування нової функціональності (new feature testing)
* regression testing
* acceptance testing
* Бета-тестування(beta testing)

*За об'єктом тестування:*

* Функціональне тестування (functional testing)
* Тестування продуктивності (performance testing)
* Навантажувальне тестування (load testing)
* Стрес-тестування (stress testing)
* Тестування стабільності (stability/endurance/soak testing)
* Тестування зручності використання або Юзабіліті-тестування (usability testing)
* Тестування інтерфейсу користувача (UI testing)
* Тестування безпеки (security testing)
* Тестування локалізації (localization testing)
* Тестування сумісності (compatibility testing)

*За ознакою позитивності сценаріїв:*

* Позитивне тестування (positive testing)
* Негативне тестування (negative testing)

**Рівні тестування:**

Модульне тестування тестує мінімальний компонент програми, або модуля. Кожний модуль тестується для перевірки правильності його реалізації.

Інтеграційне тестування виявляє дефекти в інтерфейсах та у взаємодії між компонентами (модулями).

Системне тестування тестує інтегровану систему для перевірки відповідності всім вимогам.

Системне інтеграційне тестування перевіряє, чи система інтегрується в будь-яку зовнішню систему (або системи) відповідно до системних вимог.

Приймальне тестування може проводитись кінцевим користувачем, замовником, або клієнтом для перевірки, чи може продукт бути прийнятий до використання.

альфа-тестування — це симульоване або реальне операційне тестування потенційними користувачами/замовником або командою тестувальників на боці розробника.

бета-тестування йде після альфа-тестування. Версії програмного забезпечення, відомі як бета-версії, надаються у користування обмеженій кількості людей поза компанією для того, щоб упевнитись, що програма не містить великої кількості помилок.

* **\*Розуміє принципи тестування** (за ISTQB, Foundation Level 2011, page 14).

Семь принципов тестирования (K2) 35 минут

Принципы

Эти принципы тестирования были предложены в последние 40 лет и являются

общим руководством для тестирования в целом.

Принцип 1 - **Тестування демонструє наявність дефектів** Тестування може показати, що дефекти присутні, але не може довести, що їх немає. Тестування знижує ймовірність наявності дефектів, що знаходяться в програмному забезпеченні, але, навіть якщо дефекти не були виявлені, це не доводить його коректності.  
Принцип 2 – **Вичерпне (повне) тестування недосяжно** Повне тестування з використанням усіх комбінацій вводів і передумов фізично нездійсненно, за винятком тривіальних випадків. Замість вичерпного тестування повинні використовуватися аналіз ризиків і розстановка пріоритетів, щоб більш точно сфокусувати зусилля по тестуванню.  
Принцип 3 - **Раннє тестування** Щоб знайти дефекти якомога раніше, активності по тестуванню повинні бути розпочаті як можна раніше в життєвому циклі розробки програмного забезпечення або системи, і повинні бути сфокусовані на певних цілях.  
Принцип 4 - **Скупчення дефектів** Зусилля тестування повинні бути зосереджені пропорційно очікуваній, а пізніше реальної щільності дефектів по модулях. Як правило, більша частина дефектів, виявлених при тестуванні або спричинили основна кількість збоїв системи, міститься в невеликій кількості модулів.  
Принцип 5 - **Парадокс пестициду** Якщо одні й ті ж тести будуть проганяти багато разів, в кінцевому рахунку цей набір тестових сценаріїв більше не буде знаходити нових дефектів. Щоб подолати цей "парадокс пестициду", тестові сценарії повинні регулярно рецензуватися і коригуватися, нові тести повинні бути різнобічними, щоб охопити всі компоненти програмного забезпечення, або системи, і знайти якомога більше дефектів.  
Принцип 6 - **Тестування залежить від контексту** Тестування виконується по-різному в залежності від контексту. Наприклад, програмне забезпечення, в якому критично важлива безпека, тестується інакше, ніж сайт електронної комерції. Сертифікований тестировщик Програма навчання Базового рівня  
Принцип 7 - **Помилка про відсутність помилок**. Виявлення і виправлення дефектів не допоможуть, якщо створена система не підходить користувачеві і не задовольняє його очікуванням і потребам.

* **\*Розуміє задачі тестування на різних етапах проекту.**
* **\*Може пояснити причини виникнення помилок.**
* **\*Розуміє розподіл обов’язків між керівником проекту, розробниками, керівником тестування, іншими тестувальниками та відповідальними за конфігурацію.**
* Може пояснити згідно з яким теоретичним підходом діє (exploratory testing, ad-hoc testing, risk-based testing, методики функціонального тестування). Вміє пристосувати теоретичну методику до потреб проекту.

*exploratory testing –*

Дослідницьке тестування є підхід до тестування програмного забезпечення, яке коротко описати як одночасне навчання, проектування тестів та проведення тестування. Сем Канер, який ввів цей термін в 1983 році [1] визначає тепер дослідне тестування як "стиль тестування програмного забезпечення, яке підкреслює особиста свобода і відповідальність особистості тестер постійно оптимізувати якість його / її роботи шляхом обробки тесту пов'язаних навчання, проведення дослідів, виконання тесту, і результат тесту інтерпретації як взаємодоповнюючі заходи, які виконуються паралельно протягом всього проекту ». [2]  
 У той час як програма проходить тестування, тестер дізнається речі, які разом з досвідом і творчість породжує нові хороші тести для запуску. Дослідницьке тестування часто сприймається як чорний ящик тестування техніки. Замість цього, ті, хто вчився він вважає це випробування підхід, який може бути застосований до будь тест техніка, на будь-якій стадії процесу розвитку. Ключовим є не техніка тесту, ні пункту проходять випробування або відгуки, ключовим є когнітивна участь тестер, і відповідальність тестера для управління своїм часом [3].

Дослідницьке тестування намагається з'ясувати, яким чином програма дійсно працює, і задавати питання про те, як він буде обробляти важко і легко випадків. Якість тестування залежить від уміння тестера винаходити тестів і пошуку дефектів. Чим більше тестер знає про продукт і різних методів тестування, краще тестування буде.  
 Для подальшого пояснити, порівняння може бути зроблене тестування вільної пошукових своєю протилежністю сценарієм тестування. У цій вправі тестів розроблені заздалегідь. Це включає в себе як окремі етапи й очікувані результати. Ці тести виконуються пізніше тестер, який порівнює фактичний результат з очікуваним. При виконанні дослідного тестування, очікування є відкритими. Деякі результати можна передбачити і очікувалося, інші не можуть. Тестер налаштовує працює, спостерігає і оцінює продукт і його поведінку, критично розслідування результат, і звітної інформації, яка, здається, ймовірно, буде помилка (яка загрожує вартість продукту на декілька осіб) або випуску (яка загрожує Якість робіт по тестуванню).  
 Насправді, тестування майже завжди являє собою поєднання пошукових і сценаріїв тестування, але з тенденцією до одного, в залежності від контексту.  
 За Сем Канер і Джеймс Маркус Бах, дослідне тестування більш мислення або "... спосіб мислення про тестування", ніж методологія. [5] Вони також говорять, що вона перетинає кілька континуум від пошукового (трохи неоднозначні або нечітко сценаріями тестування) до дуже пошуковий (фрістайл дослідне тестування). [6]  
 У документації розвідувальних полігонів з документуванням всіх тестів просто документування помилок. Під час тестування пари, дві людини створюють тестів разом, один виконує їх, та інші документи. Сесія на основі тестування є метод спеціально розроблений, щоб зробити дослідне тестування аудиту і вимірні в більш широкому масштабі.  
 Пошукове тестери часто використовують інструменти, в тому числі захоплення екрану або відео інструменти як запис пошукових сесії, або інструменти, щоб швидко допомогти створити ситуаціях, що представляють інтерес, наприклад, Джеймс Бах Perlclip.

Простейшее определение исследовательского тестирования — это разработка и выполнения тестов в одно и то же время. Что является противоположностью сценарного подхода (с его предопределенными процедурами тестирования, неважно ручными или автоматизированными). Исследовательские тесты, в отличие от сценарных тестов, не определены заранее и не выполняются в точном соответствии с планом. Звучит это просто, но на практике все весьма туманно. Если каждый следующий тест, который мы выполняем, выбирается по результатам предыдущего теста, это означает, что мы используем исследовательское тестирование.Мы начинаем заниматься поисками и исследованиями, когда мы не можем сказать, какие тесты должны быть выполнены, или когда мы еще не имели возможности эти тесты создать, то есть мысль об их написании даже не приходила нам в голову

<http://www.pvsm.ru/testirovanie/12059>

<http://www.pvsm.ru/testirovanie/12059>

*ad-hoc testing* –

тестирование приложений без соблюдения каких-либо правил или тест кейсов, используя собственный опыт и здравый смысл

ЮТУБ:

адхок є підвидом експораторі. Адхок є спрощеною формою експоротері. Ад хок це як хаос.

**Exploratory (одне за одним)**

- планування

- виконання запланованого

- дослідження результатів які отримали з виконанняі алануємо на основі цього щось нове

**Ad Hock (все одночасно)**

- планування

- виконання запланованого

- дослідження результатів які отримали з виконання

*risk-based testing* –

тестирование, основанное на рисках (risk-based testing): Подход к тестированию с целью минимизирования уровня проектных рисков и информирования заинтересованных лиц о текущем состоянии рисков с начальных стадий проекта. Подразумевает под собой управление процессом тестирования, исходя из идентифицированных рисков продукта и использования уровней риска.

Risk-based testing (RBT) is a type of software testing that prioritizes the tests of features and functions based on the risk of their failure - a function of their importance and likelihood or impact of failure.[1][2][3][4] In theory, since there is an infinite number of possible tests, any set of tests must be a subset of all possible tests. Test techniques such as boundary value analysis and state transition testing aim to find the areas most likely to be defective.

когда тестирование направлено на области наибольшего риска.

Риски используются для определения того, где начинать тестирование и каким аспектам уделить особое внимание; тестирование используется для того, чтобы уменьшить риск возникновения неблагоприятных результатов или уменьшить их воздействие. Риски продукта – это особенный тип рисков, который влияет на успех проекта. Тестирование это процесс, направленный на предоставление данных по оставшимся рискам путем измерения эффективности устранения критических дефектов и планов на случай возникновения непредвиденных дополнительных обстоятельств.

Подход к тестированию, основанный на рисках предоставляет превентивные возможности для уменьшения уровня рисков продукта, начиная с первоначальных этапов проекта. Он включает в себя идентификацию рисков продукта и их использование в управлении планированием тестов, спецификациями, приготовлением и выполнением тестов. В подходе, основанном на рисках – установленные риски могут быть использованы для:

· Определения методики тестирования для использования.

· Определения объема тестирования, которое должно быть выполнено.

· Установления приоритетов тестирования для того, чтобы обнаружить критические дефекты какможно раньше.

· Определения необходимости деятельности не связанной с тестированием, которая может быть предпринята для уменьшения рисков (например, предоставление тренинга для неопытных проектировщиков).

Тестирование, основанное на рисках, использует коллективное знание и понимание участников проекта для определения рисков и уровней тестирования, необходимых чтобы работать с этими рисками. Для того чтобы быть уверенным в том, что сбой в продукте минимизирован, действия по управлению рисками обеспечивают строгий порядок подхода к:

· Оценке (и переоценке на регулярной основе) того, что может пойти неверно (риски).

· Определению, какие риски наиболее важны для решения.

· Выполнению действий по работе с этими рисками.

В дополнение, тестирование может поддерживать определение новых рисков, может помогать определять, какие риски должны быть понижены и могут понижать неопределенность в отношении рисков.

**Оцінка ризиків**  
 Зміни між двома версіями або версіями є ключовим для оцінки ризику. Оцінка критичних модулів бізнесу є першим кроком у визначенні пріоритетів тестів, але вона не включає в себе поняття еволюційного ризику [роз'яснення необхідності] Цей потім розширюється за допомогою двох методів:. Зміни на основі тестування та регресійного тестування.  
     Зміна тестування на основі тестів дозволяє командам оцінити зміни, внесені в реліз, а потім пріоритети тести на зміну модулів. [Розпливчасто]  
     Регрес тестування гарантує, що зміни, такі, як виправити помилку, не вводити нові недоліки в програмному забезпеченні в стадії тестування. Однією з головних причин для регресійного тестування полягає у визначенні зміни в одній частині програмного забезпечення впливає на інші частини програмного забезпечення.  
Ці два методи дозволяють тестів команд пріоритети тестів, заснованих на ризику, зміни та критичності бізнес-модулів. Деякі технології [який?] Може зробити такої стратегії тестування дуже проста в налаштуванні і підтримувати з програмним забезпеченням зміни. [Розпливчасто]

Види ризиків

Ризик можна визначити як імовірність того, що непоміченим помилка програмного забезпечення може зробити негативний вплив на користувачів системи  
Методи оцінки ризиків за різними розміри:  
Бізнес або оперативного  
• Висока використання підсистеми, функції або  
• Критичність підсистеми, функції або функції, в тому числі витрати на провал  
технічний  
• Географічне розподіл команди розробників  
• Складність підсистеми або функції  
зовнішній  
• спонсора або виконавчої переваги  
• Нормативні вимоги  
E-Business Відмова режиму, що відносяться  
• статичні дефекти змісту  
• Веб-сторінка дефекти інтеграції  
• Функціональні, пов'язані з поведінкою недостатності  
• Обслуговування (наявність і ефективність), пов'язані відмови  
• зручність і доступність пов'язаних недостатності  
• уразливість  
• Великий провал інтеграції Масштаб

Risk can be identified as the probability that an undetected [software bug](http://en.wikipedia.org/wiki/Software_bug) may have a negative impact on the user of a system

**Business or Operational**

* High use of a subsystem, function or feature
* Criticality of a subsystem, function or feature, including the cost of failure

**Technical**

* Geographic distribution of development team
* Complexity of a subsystem or function

**External**

* Sponsor or executive preference
* Regulatory requirements

**E-Business Failure-Mode Related**

* Static content defects
* Web page integration defects
* Functional behavior-related failure
* Service (Availability and Performance) related failure
* Usability and Accessibility-related failure
* Security vulnerability
* Large Scale Integration failure

[httpHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"://HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"wwwHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY".HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"youtubeHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY".HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"comHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"/HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"watchHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"?HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"featureHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"=HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"playerHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"\_HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"detailpageHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"&HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"vHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"=HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"WmZBTuHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"5HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"wwIYHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"#HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"tHYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"=1609HYPERLINK "http://www.youtube.com/watch?feature=player\_detailpage&v=WmZBTu5wwIY"s](http://www.youtube.com/watch?feature=player_detailpage&v=WmZBTu5wwIY) (27:00)+(43:50)

* Розуміє теоретичні моделі розробки ПЗ (V-подібна діаграма, waterfall, agile (scrum, kanban, MSF).

*V-Model* (или VEE модель) является моделью разработки информационных систем (ИС), направленной на упрощение понимания сложностей, связанных с разработкой систем. Она используется для определения единой процедуры разработки программных продуктов, аппаратного обеспечения и человеко-машинных интерфейсов.

Основной принцип V-образной модели заключается в том, что детализация проекта возрастает при движении слева направо, одновременно с течением времени, и ни то, ни другое не может повернуть вспять. Итерации в проекте производятся по горизонтали, между левой и правой сторонами буквы.

Применительно к разработке информационных систем V-Model — вариация каскадной модели, в которой задачи разработки идут сверху вниз по левой стороне буквы V, а задачи тестирования — вверх по правой стороне буквы V. Внутри V проводятся горизонтальные линии, показывающие, как результаты каждой из фаз разработки влияют на развитие системы тестирования на каждой из фаз тестирования. Модель базируется на том, что приемо-сдаточные испытания основываются, прежде всего, на требованиях, системное тестирование — на требованиях и архитектуре, комплексное тестирование — на требованиях, архитектуре и интерфейсах, а компонентное тестирование — на требованиях, архитектуре, интерфейсах и алгоритмах.[4]

*Цели*

V-модель обеспечивает поддержку в планировании и реализации проекта. В ходе проекта ставятся следующие задачи:

Минимизация рисков: V-образная модель делает проект более прозрачным и повышает качество контроля проекта путём стандартизации промежуточных целей и описания соответствующих им результатов и ответственных лиц. Это позволяет выявлять отклонения в проекте и риски на ранних стадиях и улучшает качество управления проектов, уменьшая риски.

Повышение и гарантии качества: V-Model — стандартизованная модель разработки, что позволяет добиться от проекта результатов желаемого качества. Промежуточные результаты могут быть проверены на ранних стадиях. Универсальное документирование облегчает читаемость, понятность и проверяемость.

Уменьшение общей стоимости проекта: Ресурсы на разработку, производство, управление и поддержку могут быть заранее просчитаны и проконтролированы. Получаемые результаты также универсальны и легко прогнозируются. Это уменьшает затраты на последующие стадии и проекты.

Повышение качества коммуникации между участниками проекта: Универсальное описание всех элементов и условий облегчает взаимопонимание всех участников проекта. Таким образом, уменьшаются неточности в понимании между пользователем, покупателем, поставщиком и разработчиком.[5]

*Достоинства*

Пользователи V-Model участвуют в разработке и поддержке V-модели. Комитет по контролю за изменениями поддерживает проект и собирается раз в год для обработки всех полученных запросов на внесение изменений в V-Model.[6]

На старте любого проекта V-образная модель может быть адаптирована под этот проект, так как эта модель не зависит от типов организаций и проектов.[7]

V-model позволяет разбить деятельность на отдельные шаги, каждый из которых будет включать в себя необходимые для него действия, инструкции к ним, рекомендации и подробное объяснение деятельности.[8]



The **V-Model** is a term applied to a range of models, from a conceptual model designed to produce a simplified understanding of the [complexity](http://en.wikipedia.org/wiki/Complexity) associated with [systems development](http://en.wikipedia.org/wiki/Systems_development) to detailed, rigorous development lifecycle models and project management models.

There are several radically different forms of the V-Model, and this creates considerable confusion. The V-Model falls into three broad categories. [[2]](http://en.wikipedia.org/wiki/V-Model)

Firstly there is the German V-Model "Das V-Modell", the official project management methodology of the German government. It is roughly equivalent to 2["PRINCE2](%22PRINCE2), but more directly relevant to software development. [[3]](http://en.wikipedia.org/wiki/V-Model)

The US also has a government standard V-Model, which dates back about 20 years, like its German counterpart. Its scope is rather narrower, being a systems development lifecycle model, but still more far more detailed and more rigorous than most UK practitioners and testers would understand by the V-Model. [[4]](http://en.wikipedia.org/wiki/V-Model) [[5]](http://en.wikipedia.org/wiki/V-Model) [[6]](http://en.wikipedia.org/wiki/V-Model) [[7]](http://en.wikipedia.org/wiki/V-Model)

In the UK, and throughout the testing community worldwide, the V-Model is widely seen as a vaguer, illustrative depiction of the software development process, as described in the ISTQB Foundation Syllabus for software testers. [[8]](http://en.wikipedia.org/wiki/V-Model) There is no single, accepted definition of this model, which is more directly covered in the alternative article on the [V-Model (software development)](http://en.wikipedia.org/wiki/V-Model_%28software_development%29). There are therefore multiple variations of this version. This problem must be borne in mind when discussing the V-Model.

|  |
| --- |
|  |

**Overview**

The V-model is a graphical representation of the [systems development lifecycle](http://en.wikipedia.org/wiki/Systems_development_lifecycle). It summarizes the main steps to be taken in conjunction with the corresponding deliverables within [computerized system validation](http://en.wikipedia.org/wiki/Computerized_system_validation) framework.

The *V* represents the sequence of steps in a project life cycle development. It describes the activities to be performed and the results that have to be produced during product development. The left side of the "V" represents the decomposition of requirements, and creation of system specifications. The right side of the *V* represents integration of parts and their validation.[[6]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[7]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[9]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[10]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[11]](http://en.wikipedia.org/wiki/V-Model)

It is sometimes said that validation can be expressed by the query "Are you building the right thing?" and verification by "Are you building it right?"

In practice, the usage of these terms varies. Sometimes they are even used interchangeably.

[The PMBOK guide](http://en.wikipedia.org/wiki/A_Guide_to_the_Project_Management_Body_of_Knowledge), an [IEEE](http://en.wikipedia.org/wiki/IEEE) standard, defines them as follows in its 4th edition:[[12]](http://en.wikipedia.org/wiki/V-Model)

* "**Validation.** The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with *verification*."
* "**Verification**. The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with *validation*."

**Objectives**

The V-Model provides guidance for the planning and realization of projects. The following objectives are intended to be achieved by a project execution:

* **Minimization of Project Risks**: The V-Model improves project transparency and project control by specifying standardized approaches and describing the corresponding results and responsible roles. It permits an early recognition of planning deviations and risks and improves process management, thus reducing the project risk.
* **Improvement and Guarantee of Quality**: As a standardized process model, the V-Model ensures that the results to be provided are complete and have the desired quality. Defined interim results can be checked at an early stage. Uniform product contents will improve readability, understandability and verifiability.
* **Reduction of Total Cost over the Entire Project and System Life Cycle**: The effort for the development, production, operation and maintenance of a system can be calculated, estimated and controlled in a transparent manner by applying a standardized process model. The results obtained are uniform and easily retraced. This reduces the acquirers dependency on the supplier and the effort for subsequent activities and projects.
* **Improvement of Communication between all Stakeholders**: The standardized and uniform description of all relevant elements and terms is the basis for the mutual understanding between all stakeholders. Thus, the frictional loss between user, acquirer, supplier and developer is reduced.

[**V Model topics**](http://www.testinganswers.com/2012/06/v-model-is-basis-of-structured-testing.html)





Systems engineering and verification.[[13]](http://en.wikipedia.org/wiki/V-Model)

**Systems Engineering and verification**

The Systems Engineering Process (SEP) provides a path for improving the cost effectiveness of complex systems as experienced by the system owner over the entire life of the system, from conception to retirement.[[1]](http://en.wikipedia.org/wiki/V-Model)

It involved early and comprehensive identification of goals, a concept of operations that describes user needs and the operating environment, thorough and testable system requirements, detailed design, implementation, rigorous acceptance testing of the implemented system to ensure it meets the stated requirements (system verification), measuring its effectiveness in addressing goals (system validation), on-going operation and maintenance, system upgrades over time, and eventual retirement.[[1]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[6]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[7]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[11]](http://en.wikipedia.org/wiki/V-Model)

The process emphasizes requirements-driven design and testing. All design elements and acceptance tests must be traceable to one or more system requirements and every requirement must be addressed by at least one design element and acceptance test. Such rigor ensures nothing is done unnecessarily and everything that is necessary is accomplished.[[1]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[6]](http://en.wikipedia.org/wiki/V-Model)

**The specification stream**

The specification stream mainly consists of:

* User Requirement Specifications
* Functional Requirement Specifications
* Design Specifications

The testing stream generally consists of:

* Installation Qualification (IQ)
* Operational Qualification (OQ)
* Performance Qualification (PQ)

The development stream can consist (depending on the system type and the development scope) of customization, configuration or coding.

**Applications**





Off-Core alternatives (illustrating upward and downward iterations and Time and Maturity dimension). Source - K. Forsberg and H. Mooz 2004[[6]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[11]](http://en.wikipedia.org/wiki/V-Model)

The V-model is used to regulate the software development process within the German federal administration. Nowadays it is still the standard for [German](http://en.wikipedia.org/wiki/Germany) federal administration and defense projects, as well as software developers within the region.

The concept of the V-Model was developed simultaneously, but independently, in Germany and in the United States in the late 1980s:

* The German V-Model was originally developed by IABG in Ottobrunn, near Munich, in cooperation with the Federal Office for Defense Technology and Procurement in Koblenz, for the Federal Ministry of Defense. It was taken over by the Federal Ministry of the Interior for the civilian public authorities domain in summer 1992.[[14]](http://en.wikipedia.org/wiki/V-Model)
* The US V-Model, as documented in the 1991 proceedings for the [National Council on Systems Engineering](http://en.wikipedia.org/wiki/International_Council_on_Systems_Engineering) (NCOSE; now INCOSE as of 1995),[[11]](http://en.wikipedia.org/wiki/V-Model) was developed for satellite systems involving hardware, software, and human interaction.
* The V-Model first appeared at [Hughes](http://en.wikipedia.org/wiki/Hughes) Aircraft circa 1982 as part of the pre-proposal effort for the FAA Advanced Automation System (AAS) program. It eventually formed the test strategy for the Hughes AAS Design Competition Phase (DCP) proposal. It was created to show the test and integration approach which was driven by new challenges to surface latent defects in the software. The need for this new level of latent defect detection was driven by the goal to start automating the thinking and planning processes of the air traffic controller as envisioned by the Automated Enroute Air Traffic Control (AERA) program. The reason the V is so powerful comes from the Hughes culture of coupling all text and analysis to multi dimensional images. It was the foundation of Sequential Thematic Organization of Publications (STOP) [[15]](http://en.wikipedia.org/wiki/V-Model) created by Hughes in 1963 and used until Hughes was divested by the [Howard Hughes Medical Institute](http://en.wikipedia.org/wiki/Howard_Hughes_Medical_Institute) in 1985.[[16]](http://en.wikipedia.org/wiki/V-Model)

It has now found widespread application in commercial as well as defense programs. Its primary use is in Project Management[[6]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[7]](http://en.wikipedia.org/wiki/V-Model) and throughout the project lifecycle.

One fundamental characteristic of the US V-Model is that time and maturity move from left to right and one cannot move back in time. All iteration is along a vertical line to higher or lower levels in the system hierarchy, as shown in the figure.[[6]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[7]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[11]](http://en.wikipedia.org/wiki/V-Model) This has proven to be an important aspect of the model. The expansion of the model to a dual-Vee concept is treated in reference.[[6]](http://en.wikipedia.org/wiki/V-Model)

As the V-model is publicly available many companies also use it. In project management it is a method comparable to 2["PRINCE2](%22PRINCE2) and describes methods for project management as well as methods for [system HYPERLINK "http://en.wikipedia.org/wiki/Systems\_development\_lifecycle"development](http://en.wikipedia.org/wiki/Systems_development_lifecycle). The V-Model while rigid in process, can be very flexible in application, especially as it pertains to the scope outside of the realm of the System Development Lifecycle normal parameters.

**Advantages**

These are the advantages V-Model offers in front of other systems development models:

* The users of the V-Model participate in the development and maintenance of The V-Model. A change control board publicly maintains the V-Model. The change control board meets anywhere from every day to weekly and processes all change requests received during system development and test.[[17]](http://en.wikipedia.org/wiki/V-Model)
* The V-Model provides concrete assistance on how to implement an activity and its work steps, defining explicitly the events needed to complete a work step: each activity schema contains instructions, recommendations and detailed explanations of the activity.[[18]](http://en.wikipedia.org/wiki/V-Model)

**Limits**

The following aspects are not covered by the V-Model, they must be regulated in addition, or the V-Model must be adapted accordingly [[19]HYPERLINK "http://en.wikipedia.org/wiki/V-Model"[20]](http://en.wikipedia.org/wiki/V-Model):

* The placing of contracts for services is not regulated.
* The organization and execution of operation, maintenance, repair and disposal of the system are not covered by the V-Model. However, planning and preparation of a concept for these tasks are regulated in the V-Model.
* The V-Model addresses software development within a project rather than a whole organization.

V-модель являє собою термін, який застосовується до цілого ряду моделей, від концептуальної моделі призначені для виробництва спрощеного розуміння складності, пов'язані з розвитком системи докладним, строгий життєвий цикл розробки і моделі управління проектами.  
Є кілька радикально різними формами V-модель, і це створює значну плутанину. V-модель розпадається на три основні категорії. [2]  
По-перше, це німецька V-модель "Das V-Modell», офіційний методології управління проектами уряду Німеччини. Це приблизно еквівалентно PRINCE2, але більш безпосереднє відношення до розробки програмного забезпечення. [3]  
У США також державного стандарту V-модель, яка сходить близько 20 років, як і його німецький колега. Його можливості досить вузьке, будучи розвитку систем життєвого циклу моделі, але ще набагато більш детальну і більш суворими, ніж більшість практикуючих Великобританії і тестери б розуміти під V-модель. [4] [5] [6] [7]  
У Великобританії і у всьому тестування співтовариством у всьому світі, V-модель широко розглядається як розпливчасті, ілюстративний опис процесу розробки програмного забезпечення, як описано в навчальному Фонду ISTQB для тестувальників програмного забезпечення. [8] Існує не одна, прийнятого визначення цієї моделі, які більш безпосередньо розглядаються в альтернативну статтю на V-моделі (розробка програмного забезпечення). Є тому кілька варіантів цієї версії. Ця проблема повинна мати на увазі при обговоренні V-модель.  
•  
Огляд  
V-модель являє собою графічне представлення життєвого циклу розробки систем. У ньому коротко викладаються основні кроки, які необхідно прийняти у зв'язку з відповідними результатами в рамках комп'ютеризованою системою перевірки.  
V являє собою послідовність кроків життєвого циклу розробки проекту. У ньому описуються заходи, які будуть проведені і результати, які повинні бути зроблені при розробці продукту. У лівій частині "V" являє собою розкладання вимогам, а також створення специфікацій системи. У правій частині V являє інтеграція частин і їх перевірки. [6] [7] [9] [10] [11]  
Іноді кажуть, що перевірка може бути виражений запит "Ви будуєте правильну річ?" і перевірка на "Ви будуєте це правильно?"  
На практиці, використання цих термінів змінюється. Іноді вони навіть використовуються як взаємозамінні.  
Керівництво PMBOK, IEEE стандарт, що визначає їх як слід у своїй 4-е видання: [12]  
• "Validation. Гарантія того, що продукт, послуга чи система задовольняє потреби клієнтів та інших виявлених зацікавлених сторін. Часто включає в себе прийняття та сумісність із зовнішніми клієнтами. Контраст з перевіркою".  
• «Контроль. Оцінки чи ні продукт, послугу або система відповідає регулювання, вимоги, специфікації, або нав'язаних умов. Часто внутрішній процес. Контраст з перевіркою".  
Цілі  
V-модель забезпечує керівництво з планування та реалізації проектів. Наступні мети передбачається досягти шляхом виконання проекту:  
• Мінімізація ризиків проекту: V-модель підвищує прозорість проекту та управління проектом, вказавши стандартизованих підходів і опис відповідних результатів і відповідальна роль. Це дозволяє раннє виявлення відхилень і планування ризиків і покращує процес управління, тим самим зменшуючи ризик проекту.  
• вдосконалення та гарантії якості: в якості стандартизованої моделі процесу, V-модель гарантує, що результати мають бути надані повні і мати бажаної якості. Певні проміжні результати можуть бути перевірені на ранній стадії. Рівномірний зміст продукту поліпшення читабельності, зрозумілості та перевірюваність.  
• Зниження сукупної вартості протягом усього проекту і життєвий цикл системи: зусилля по розробці, виробництву, експлуатації та технічного обслуговування системи можна обчислити, оцінюються і контролюються в прозорій манері, застосовуючи стандартну модель процесу. Отримані результати є однорідним і легко простежується. Це знижує набувачів залежності від постачальника і зусиль для подальших заходів та проектів.  
• покращення комунікації між усіма зацікавленими сторонами: стандартизовані і єдине опис всіх відповідних елементів і умов є основою для взаєморозуміння між усіма зацікавленими сторонами. Таким чином, втрати при терті між користувачем, замовник, постачальник і розробник знижується.  
Тим V моделі  
 Інженерних систем та перевірці. [13]  
Інженерних систем та перевірці  
Process Systems Engineering (SEP) забезпечує шлях для підвищення економічної ефективності складних систем з досвіду власника системи протягом усього терміну служби системи, від концепції до виходу на пенсію. [1]  
У ньому взяли участь якнайшвидшого і всеосяжного визначення мети, концепції операцій, що описують потреби користувача і операційного середовища, ретельна і перевіряються вимоги до системи, детальне проектування, впровадження, строгі приймальні випробування впровадженої системи для забезпечення відповідності встановленим вимогам (система перевірки ), виміру її ефективності у вирішенні цілей (система перевірки), поточної експлуатації та технічного обслуговування, модернізації системи в часі, і в остаточному підсумку вихід на пенсію. [1] [6] [7] [11]  
Цей процес підкреслює вимоги керованої розробки і тестування. Всі елементи дизайну і приймально-здавальні випробування повинні проводитись відповідно до одним або декількома системними вимогами і кожну вимогу повинні бути вирішені, принаймні один елемент дизайну та приймально-здавальні випробування. Така строгість гарантує, нічого не робиться без необхідності і все, що потрібно буде зробити. [1] [6]  
Специфікація потоку  
Специфікація потоку в основному складається з:  
• Технічні вимоги користувача  
• Функціональні характеристики вимог  
• Технічні характеристики Design  
Тестування потік зазвичай складається з:  
• Установка кваліфікація (IQ)  
• Операційна кваліфікація (OQ)  
• Продуктивність кваліфікація (PQ)  
Розвиток потік може складатися (в залежності від типу системи і розвитку сфери), налаштування, конфігурації або кодування.  
Застосування  
    
Off-основної альтернативи (иллюстрирующая вгору і вниз ітерацій і часу і погашення розмірності). Джерело - К. Форсберг і H. Mooz 2004 [6] [11]  
V-модель використовується для регулювання процесу розробки програмного забезпечення в німецькій федеральній адміністрації. В даний час він як і раніше стандарт для німецької федеральної адміністрації та оборонних проектів, а також розробників програмного забезпечення в регіоні.  
Концепція V-модель була розроблена одночасно, але незалежно один від одного, в Німеччині і в Сполучених Штатах в кінці 1980-х років:  
• Німецький V-модель спочатку була розроблена IABG в Ottobrunn, недалеко від Мюнхена, у співпраці з Федеральним відомством з технології обороні і закупівель в Кобленці, для федерального міністерства оборони. Вона була передана Федеральному міністерству внутрішніх справ для цивільного суспільним надбанням влади влітку 1992 року. [14]  
• США V-модель, як зафіксовано в 1991 році розгляду до Національної ради з питань інженерних систем (NCOSE, тепер INCOSE за станом на 1995), [11] був розроблений для супутникових систем, що включають устаткування, програмне забезпечення та людської взаємодії.  
• V-модель вперше з'явилася на літак Хьюза близько 1982 як частина попередню пропозицію зусилля для автоматизації FAA Advanced System (AAS) програми. Це в кінцевому підсумку сформували стратегію тестування для дизайну Hughes AAS Конкурс фази (DCP) пропозиції. Він був створений, щоб показати, випробування і інтеграція підходу, який був викликаний новим викликам на поверхню приховані дефекти в програмному забезпеченні. Потреба в цей новий рівень прихованого виявлення дефектів був обумовлений метою, щоб почати автоматизацію мислення і планування процесів авіадиспетчера, як це передбачено в автоматизованих Маршрутна управління повітряним рухом (AERA) програми. Причина, по якій V настільки потужним, походить від Hughes культури зчеплення весь текст і аналізу багатовимірних образів. Це було основою послідовного Тематичні організації Публікації (STOP) [15] створені Hughes в 1963 році і використовувалася до Хьюз був позбавлений по Медичного інституту Говарда Хьюза у 1985 році. [16]  
В даний час знайшли широке застосування в комерційних, так і військових програм. Його первинне використання в управлінні проектами [6] [7] і протягом всього життєвого циклу проекту.  
Одна з основних характерних для США V-моделі є те, що час і зрілість рух зліва направо, і ніхто не може рухатися назад в часі. Всі ітерації по вертикальній лінії вище або нижче рівня в ієрархії системи, як показано на малюнку. [6] [7] [11] Це виявилося важливим аспектом моделі. Розширення моделі подвійного Vee поняття трактується у посилання. [6]  
Як V-модель є загальнодоступною багато компаній також використовують його. В управлінні проектами це метод порівняємо з PRINCE2 і описує методи управління проектами, а також методи для розвитку системи. V-модель в той час як жорстка в процесі, може бути дуже гнучкою у застосуванні, особливо, що стосується сфери за межами області системи Development Lifecycle нормальні параметри.  
Переваги  
Ці переваги V-модель пропонується в присутності інших систем розробки моделей:  
• користувачі V-модель участі в розвитку та підтримці V-модель. Плата управління змінами публічно стверджує, V-модель. Плата управління змінами зустрічається скрізь від кожного дня щотижневого і обробляє всі запити на зміну, отримані в ході розробки системи і випробування. [17]  
• V-модель надає конкретну допомогу з питань здійснення діяльності та її етапи роботи, що визначають явно подій, необхідних для завершення робіт кроком. Кожного виду діяльності схема містить інструкції, рекомендації та детальні роз'яснення про діяльність [18]  
Рамки  
Наступні аспекти не поширюється на V-моделі, вони мають бути врегульовані в доповнення або V-моделі повинні бути адаптовані відповідно [19] [20]:  
• розміщення контрактів на надання послуг не регулюється.  
• організація та здійснення експлуатації, технічного обслуговування, ремонту та утилізації системи не поширюється на V-модель. Тим не менш, планування і підготовка концепції для вирішення цих завдань регулюється в V-модель.  
• V-Model адреси розробки програмного забезпечення в рамках проекту, а не всієї організації.

*Каскадная модель* (англ. waterfall model) — модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки. В качестве источника названия часто указывают статью, опубликованную У. У. Ройсом (W. W. Royce) в 1970 году; забавно, что сам Ройс использовал итеративную модель разработки.

 

**Водоспадна модель** ( 0<%90%D>0<%BD%D>0<%B>3<%D>0<%BB%D>1<%96%D>0<%B>9<%D>1<%81%D>1<%>8<C%D>0<%BA%D>0<%B>0<_%D>0<%BC%D>0<%BE%D>0<%B>2<%D>0<%B>0["англ.](%22англ.) *waterfall model* ) - послідовний метод розробки програмного забезпечення, названий так через діаграму схожу на водоспад (як на ілюстрації справа).

Цей метод з'явився раніше ніж перше 0<%>9<F%D>1<%80%D>0<%BE%D>0<%B>3<%D>1<%80%D>0<%B>0<%D>0<%BC%D>0<%BD%D>0<%B>5<_%D>0<%B>7<%D>0<%B>0<%D>0<%B>1<%D>0<%B>5<%D>0<%B>7<%D>0<%BF%D>0<%B>5<%D>1<%87%D>0<%B>5<%D>0<%BD%D>0<%BD%D>1<%>8[F"програмне забезпечення](F%22програмне%20забезпечення). Його застосовували для створення складних інженерних конструкцій (0<%>9<B%D>1<%96%D>1<%82%D>0<%B>0<%D>0[%BA"літаків](%BA%22літаків), 0<%>9<C%D>1<%96%D>1<%81%D>1[%82"мостів](%82%22мостів) і подібного). Зрозуміло, що зміни в проекті після того, як вже натягнуті розтяжки коштують дуже дорого, тому метод передбачає перфекціонізм на кожному етапі. А так, як колись ще не існувало формальних методів розробки ПЗ, для неї просто перейняли цю модель.

Перший формальний опис водоспадної моделі, після якої вона стала популярною був здійснений В. В. Ройсом у 19700<%92%D>0<%BE%D>0<%B>4<%D>0<%BE%D>1<%81%D>0<%BF%D>0<%B>0<%D>0<%B>4<%D>0<%BD%D>0<%B>0<_%D>0<%BC%D>0<%BE%D>0<%B>4<%D>0<%B>5<%D>0<%BB%D>1<%>8[C"[1]](C%22%5B1%5D). Попри те, що стаття містить переважно критику методу, на неї часто посилаються.

|  |
| --- |
|  |

**Плюси методу**

* Ніяких переробок
* Гарна специфікація перетікає в гарну документацію
* Зрозуміла модель
* Кодери можуть мати низьку кваліфікацію.

**Мінуси**

* Необхідний перфекціонізм на кожному етапі.
* Важко вносити зміни (якщо взагалі можливо)
* Надлишкове проектування
* Поділ розробників на "perfect" та "code monkeys"

**Модифікації**

Так як цей метод слабо підходить для розробки саме ПЗ, частіше використовують його модифікації.

Найвідоміша модифікація - Sashimi. Названа так через японську страву 0<%A>1<%D>0<%B>0<%D>1<%88%D>0<%B>8<%D>0<%BC%D>1[%96&action=edit&redlink=](%96%26action=edit%26redlink=)1["сашимі](%22сашимі). Це 0<%A>1<%D>1<%83%D>1<%88%D>1[%96"суші](%96%22суші) нарізане і сервіроване так, що складені рядочком кусочки накладаються один на одного.

В моделі розробки перекриваються фази. Тому водоспад стає більш схожим на сашімі.

Содержание модели

В 1970 году в своей статье Ройс описал в виде концепции то, что сейчас принято называть «каскадная модель», и обсуждал недостатки этой модели. Там же он показал как эта модель может быть доработана до итеративной модели.

В оригинальной каскадной модели Ройса, следующие фазы шли в таком порядке:

-Определение требований

-Проектирование

-Конструирование (также «реализация» либо «кодирование»)

-Воплощение

-Тестирование и отладка (также «верификация»)

-Инсталляция

-Поддержка

Переход от одной фазы к другой происходит только после полного и успешного завершения предыдущей

Следуя каскадной модели, разработчик переходит от одной стадии к другой строго последовательно. Сначала полностью завершается этап «определение требований», в результате чего получается список требований к ПО. После того как требования полностью определены, происходит переход к проектированию, в ходе которого создаются документы, подробно описывающие для программистов способ и план реализации указанных требований. После того как проектирование полностью выполнено, программистами выполняется реализация полученного проекта. На следующей стадии процесса происходит интеграция отдельных компонентов, разрабатываемых различными командами программистов. После того как реализация и интеграция завершены, производится тестирование и отладка продукта; на этой стадии устраняются все недочёты, появившиеся на предыдущих стадиях разработки. После этого программный продукт внедряется и обеспечивается его поддержка — внесение новой функциональности и устранение ошибок.

Тем самым, каскадная модель подразумевает, что переход от одной фазы разработки к другой происходит только после полного и успешного завершения предыдущей фазы, и что переходов назад либо вперёд или перекрытия фаз — не происходит.

Тем не менее, существуют модифицированные каскадные модели (включая модель самого Ройса), имеющие небольшие или даже значительные вариации описанного процесса.

**Гнучка́ розро́бка програ́много забезпе́чення** (0<%90%D>0<%BD%D>0<%B>3<%D>0<%BB%D>1<%96%D>0<%B>9<%D>1<%81%D>1<%>8<C%D>0<%BA%D>0<%B>0<_%D>0<%BC%D>0<%BE%D>0<%B>2<%D>0<%B>0["англ.](%22англ.) *Agile software development*) — клас методологій розробки програмного забезпечення, що базується на 0<%86%D>1<%82%D>0<%B>5<%D>1<%80%D>0<%B>0<%D>1<%82%D>0<%B>8<%D>0<%B>2<%D>0<%BD%D>0<%B>0<_%D>1<%82%D>0<%B>0<_%D>1<%96%D>0<%BD%D>0<%BA%D>1<%80%D>0<%B>5<%D>0<%BC%D>0<%B>5<%D>0<%BD%D>1<%82%D>0<%BD%D>0<%B>0<_%D>1<%80%D>0<%BE%D>0<%B>7<%D>1<%80%D>0<%BE%D>0<%B>1<%D>0<%BA%D>0<%B>0["ітеративній розробці](%22ітеративній%20розробці), в якій вимоги та розв'язки еволюціонують через співпрацю між самоорганізовуваними багатофункціональними командами.

Термін з'явився у 2001 році, коли був написаний 0<%>9<C%D>0<%B>0<%D>0<%BD%D>1<%96%D>1<%84%D>0<%B>5<%D>1<%81%D>1<%82_%D>0<%B>3<%D>0<%BD%D>1<%83%D>1<%87%D>0<%BA%D>0<%BE%D>1<%97_%D>1<%80%D>0<%BE%D>0<%B>7<%D>1<%80%D>0<%BE%D>0<%B>1<%D>0<%BA%D>0<%B>8["Маніфест гнучкої розробки](%22Маніфест%20гнучкої%20розробки).

Більшість гнучких методологій націлені на мінімізацію ризиків, шляхом зведення розробки до серії коротких циклів, що мають назву 0<%86%D>1<%82%D>0<%B>5<%D>1<%80%D>0<%B>0<%D>1<%86%D>1<%96%D>1<%>8[F"ітерацій](F%22ітерацій), які зазвичай тривають один-два тижні. Кожна ітерація сама по собі виглядає як програмний проект в мініатюрі, і включає всі завдання, необхідні для видачі мінімального приросту за функціональністю: планування, аналіз вимог, проектування, кодування, тестування і документування. Хоча окрема ітерація, як правило, недостатня для випуску нової версії продукту, мається на увазі те, що гнучкий програмний проект готовий до випуску в кінці кожної ітерації. Після закінчення кожної ітерації, команда виконує переоцінку пріоритетів розробки.

Agile акцентує увагу на безпосередньому спілкуванні «віч-на-віч». Більшість agile команд розташовані в одному офісі, його іноді називають *bullpen*. Як мінімум вона включає і «замовників» (замовники, які визначають продукт, також це можуть бути менеджери продукту, бізнес аналітики або клієнти). Офіс може також включати тестувальників, дизайнерів інтерфейсу, 0<%A>2<%D>0<%B>5<%D>1<%85%D>0<%BD%D>1<%96%D>1<%87%D>0<%BD%D>0<%B>8<%D>0<%B>9<_%D>0<%B>0<%D>0<%B>2<%D>1<%82%D>0<%BE%D>1[%80"технічних авторів](%80%22технічних%20авторів) і менеджерів.

Основною метрикою agile методів є робочий продукт. Віддаючи перевагу безпосередньому спілкуванню agile методи зменшують обсяг письмової документації в порівнянні з іншими методами. Це привело до критики цих методів як не дисциплінованих.



*Принципы гибкой методологии*

В манифесте описано 12 основополагающих принципов Agile. Вот некоторые из них:

-удовлетворение потребностей заказчика;

-внесение изменений даже не поздних стадиях разработки;

-часто выпускается готовый продукт;

-разработчики и представители бизнеса работают вместе на протяжении всего проекта;

-показатель прогресса — работающий продукт;

-грамотное проектирование и гибкость;

-простота — минимизировать лишнюю работу;

-анализ и корректировка эффективности работы команды.

Преимущества применения Agile

-частые релизы — требования не успевают устаревать, частью функционала уже можно пользоваться;

-фиксированная длина итераций — можно предсказывать скорость работы команды с учетом рисков;

-команда сама оценивает задачи — оценки реалистичны, команда мотивирована выполнить свои обязательства;

-команда самоуправляемая — 10 голов учтут больше чем одна очень умная;

-в конце каждой итерации процесс работы оценивается и вносятся улучшения;

-команда кроссфункциональная — границы отделов компании не являются препятствием при сотрудничестве, разнообразные навыки сочетаются и происходит синергия.

*Scrum* — методология управления разработкой информационных систем для гибкой разработки программного обеспечения. Scrum чётко делает акцент на качественном контроле процесса разработки. Кроме управления проектами по разработке ПО Scrum может также использоваться в работе команд поддержки программного обеспечения (software support teams), или как подход управления разработкой и сопровождением программ: Scrum of Scrums.

**Принципы**

Agile — семейство процессов разработки, а не единственный подход в разработке программного обеспечения, и определяется Agile Manifesto0<%93%D>0<%B>8<%D>0<%B>1<%D>0<%BA%D>0<%B>0<%D>1<%>8<F_%D>0<%BC%D>0<%B>5<%D>1<%82%D>0<%BE%D>0<%B>4<%D>0<%BE%D>0<%BB%D>0<%BE%D>0<%B>3<%D>0<%B>8<%D>1<%>8<F_%D>1<%80%D>0<%B>0<%D>0<%B>7<%D>1<%80%D>0<%B>0<%D>0<%B>1<%D>0<%BE%D>1<%82%D>0<%BA%D>0<%B>8["[1]](%22%5B1%5D). Agile не включает практик, а определяет ценности и принципы, которыми руководствуются успешные команды.

Agile Manifesto разработан и принят 11-13 февраля 2001 года на лыжном курорте The Lodge at Snowbird в горах Юты. Манифест подписали представители следующих методологий [Extreme programming](http://ru.wikipedia.org/wiki/Extreme_programming), [Scrum](http://ru.wikipedia.org/wiki/Scrum), [DSDM](http://ru.wikipedia.org/wiki/DSDM), 1["Adaptive software development](%22Adaptive%20software%20development), 1["Crystal Clear](%22Crystal%20Clear), 1["Feature-driven development](%22Feature-driven%20development), 1["Pragmatic Programming](%22Pragmatic%20Programming). Agile Manifesto содержит 4 основные идеи и 12 принципов. Примечательно, что Agile Manifesto не содержит практических советов.

Основные идеи:

* Личности и их взаимодействия важнее, чем процессы и инструменты;
* Работающее программное обеспечение важнее, чем полная документация;
* Сотрудничество с заказчиком важнее, чем контрактные обязательства;
* Реакция на изменения важнее, чем следование плану.

Принципы, которые разъясняет Agile Manifesto0<%93%D>0<%B>8<%D>0<%B>1<%D>0<%BA%D>0<%B>0<%D>1<%>8<F_%D>0<%BC%D>0<%B>5<%D>1<%82%D>0<%BE%D>0<%B>4<%D>0<%BE%D>0<%BB%D>0<%BE%D>0<%B>3<%D>0<%B>8<%D>1<%>8<F_%D>1<%80%D>0<%B>0<%D>0<%B>7<%D>1<%80%D>0<%B>0<%D>0<%B>1<%D>0<%BE%D>1<%82%D>0<%BA%D>0<%B>8["[2]](%22%5B2%5D):

* удовлетворение клиента за счёт ранней и бесперебойной поставки ценного программного обеспечения;
* приветствие изменений требований даже в конце разработки (это может повысить конкурентоспособность полученного продукта);
* частая поставка рабочего программного обеспечения (каждый месяц или неделю или ещё чаще);
* тесное, ежедневное общение заказчика с разработчиками на протяжении всего проекта;
* проектом занимаются мотивированные личности, которые обеспечены нужными условиями работы, поддержкой и доверием;
* рекомендуемый метод передачи информации — личный разговор (лицом к лицу);
* работающее программное обеспечение — лучший измеритель прогресса;
* спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок;
* постоянное внимание улучшению технического мастерства и удобному дизайну;
* простота — искусство не делать лишней работы;
* лучшие технические требования, дизайн и архитектура получаются у самоорганизованной команды;
* постоянная адаптация к изменяющимся обстоятельствам.

**Scrum** — методологія 0<%A>3<%D>0<%BF%D>1<%80%D>0<%B>0<%D>0<%B>2<%D>0<%BB%D>1<%96%D>0<%BD%D>0<%BD%D>1<%>8<F_%D>0<%BF%D>1<%80%D>0<%BE%D>0<%B>5<%D>0<%BA%D>1<%82%D>0<%B>0<%D>0<%BC%D>0<%B>8["управління проектами](%22управління%20проектами) для 0<%93%D>0<%BD%D>1<%83%D>1<%87%D>0<%BA%D>0<%B>0<_%D>1<%80%D>0<%BE%D>0<%B>7<%D>1<%80%D>0<%BE%D>0<%B>1<%D>0<%BA%D>0<%B>0<_%D>0<%BF%D>1<%80%D>0<%BE%D>0<%B>3<%D>1<%80%D>0<%B>0<%D>0<%BC%D>0<%BD%D>0<%BE%D>0<%B>3<%D>0<%BE_%D>0<%B>7<%D>0<%B>0<%D>0<%B>1<%D>0<%B>5<%D>0<%B>7<%D>0<%BF%D>0<%B>5<%D>1<%87%D>0<%B>5<%D>0<%BD%D>0<%BD%D>1<%>8[F"гнучкої розробки програмного забезпечення](F%22гнучкої%20розробки%20програмного%20забезпечення). Скрам чітко робить акцент на якісному контролі процесу розробки.

Підхід вперше описали Гіротака Такеучі та Ікуджіро Нонака[[1]](http://uk.wikipedia.org/wiki/Scrum) в статті *The New New Product Development Game* (*Гарвардський Діловий Огляд* [[2]](http://uk.wikipedia.org/wiki/Scrum), *січ-лют 1986*). Вони відзначили, що проекти, над якими працюють невеликі, крос-функціональні команди, зазвичай систематично продукують кращі результати, і пояснили це, як «підхід регбі». У 1991 році ДеҐрейс та Шталь у книжці *Злі проблеми, справедливі рішення*[[3]](http://uk.wikipedia.org/wiki/Scrum) послалися на цей підхід, як на Scrum (*штовханина; сутичка навколо м'яча (у регбі)*), спортивний термін, згаданий в статті Такеучі і Нонака. 0<%>9<A%D>0<%B>5<%D>0<%BD_%D>0<%A>8<%D>0<%B>2<%D>0<%B>0<%D>0<%B>1<%D>0<%B>5<%D>1[%80"Кен Швабер](%80%22Кен%20Швабер) на початку 1990-х використовував підхід який привів Scrum в його компанію. Вперше метод Scrum було представлено на загальний огляд задокументованим, чітко сформульованим та описаним спільно Сазерлендом та Швабером на OOPSLA'96 в Остіні. Швабер та Сазерленд протягом наступних років працювали разом щоб обробити та описати весь їхній досвід та найкращі практичні зразки для індустрії в одне ціле, в ту методологію, що відома сьогодні як Scrum. Швабер об'єднав зусилля з Майком Бідлом[[4]](http://uk.wikipedia.org/wiki/Scrum) в 2001, щоб детально описати метод в книжці Agile Software Development with SCRUM. Не зважаючи на те, що для Scrum нарікли долю управління проектами з розробки ПЗ, він може також використовуватися в роботі команд обслуговувань програмного забезпечення (*software maintenance teams*), або як підхід управління розробкою і супроводом програм: *Scrum of Scrums*.

|  |
| --- |
|  |

**Визначення**





Скрам процеси

Scrum — це кістяк процесу, який включає набір методів і попередньо визначених ролей. Головні дійові особи — *ScrumMaster*, той хто опікується процесами, веде їх і працює як керівник проекту, *Власник Продукту*, людина, що представляє інтереси кінцевих користувачів та інших зацікавлених в продукті сторін, та *Команду*, яка включає розробників.

Протягом кожного *спринту* [[5]](http://uk.wikipedia.org/wiki/Scrum), 15-30 денного періоду (тривалість визначається командою), працівники створюють функціональний ріст програмного забезпечення.

Набір можливостей, які імплементуються кожного спринту, приходять з етапу, що має назву *product backlog* (документація запитів на виконання робіт), який має найвищу пріоритетність за рівнем вимог до роботи, що повинна бути виконана. Запити на виконання робіт (*backlog items*) що визначені протягом *наради з планування спринту* (*sprint planning meeting*) переміщуються в етап спринту. Протягом цієї наради Власник Продукту інформує про завдання, які він хоче, аби були виконані. Тоді Команда визначає, скільки з бажаного вони можуть зробити, щоб завершити необхідні частини протягом наступного спринту[[6]](http://uk.wikipedia.org/wiki/Scrum). Протягом спринту команда виконує визначений фіксований список завдань (т.з. *backlog items*). Впродовж цього періоду ніхто не має права змінювати перелік запитів на виконання робіт, що слід розуміти, як заморожування вимог (*requirements*) протягом спринту.

**Дійові особи**

За методикою Scrum у виробничому процесі є визначені ролі, що розбиті на дві групи — «свиней» та «курей». Ці назви використані через жарт про свиню та курку.[[6]](http://uk.wikipedia.org/wiki/Scrum)

Свиня і курка йшли собі по дорозі. Курка дивиться на свиню і питає «Егей, а давай відкриємо ресторан!» Свиня дивиться на курку і відповідає «Добра ідея, а що ми будемо подавати на стіл?» Курка думає і каже: «Чому б не подавати шинку та яйця?». «Я не згодна», відповідає свиня, «тоді я буду повністю зобов’язанна (досл. повністю приготована, *committed*), а ти лише задіяна (*involved*).»

Отже *свині* використовуються для побудови продукту регулярно і часто (повністю задіяні), тоді як будь-які інші — *кури*, ті, що зацікавлені (і задіяні) в проекті, але не мають прямого стосунку до приготування *страви*. Потреби, бажання, ідеї та вплив *курей* беруться до уваги, але їм не завжди дозволяють прямо впливати, видозмінювати або включатися в хід Scrum проекту.

**«Свині»**

Свині цілком задіяні в проекті, у Скрам процес, так би мовити вони єдині з «власним беконом» на виробничій лінії.

* **Власник Продукту (*Product Owner*)**
* **Керівник (*ScrumMaster*)**
* **Команда (*Scrum Team*)**

**«Кури»**

* **Користувачі (*Users*)**
* **Клієнти, Продавці (*Stakeholders*)**
* **Експерти-консультанти (*Consulting Experts*)**

**Артефакти**

**Product backlog (беклог)**

**Product backlog** – це документ, який має список вимог до функціональності, які упорядковані згідно зі ступенем важливості. Product backlog представляє список того, що повинно бути реалізовано. Елементи цього списку називається «історіями» (*user story*) або елементами backlog-у (*backlog items*). Product backlog відкритий для редагування усім учасникам Scrum-процесу.

***Обов’язкові поля***

* **ID** – унікальний ідентифікатор, порядковий номер, який використовується для ідентифікації історій у разі їх перейменування.
* **Назва (Name)** – стислий опис історії. Він повинен бути однозначним, щоб і розробники і product owner могли зрозуміти, про що йде мова і відрізнити одну історію від іншої.
* **Важливість (Importance)** – ступінь важливості даної історії на погляд product owner ’a. Зазвичай представляє собою натуральне число, іноді для цієї цілі використовуються 0<%A>7<%D>0<%B>8<%D>1<%81%D>0<%BB%D>0<%B>0<_%D>0<%A>4<%D>1<%96%D>0<%B>1<%D>0<%BE%D>0<%BD%D>0<%B>0<%D>1<%87%D>1<%87%D>1[%96"числа Фібоначчі](%96%22числа%20Фібоначчі). Чим більше значення, тим більше пріоритет.
* **Попередня оцінка (initial estimate)** – початкова оцінка об’єму робіт, необхідного для реалізаціє історії порівняно з іншими історіями. Вимірюється у story point’ах. Приблизно відповідає числу «ідеальних людино-днів».
* **Як продемонструвати (how to demo)** – стисле пояснення того, як завершена задача буде продемонстрована у кінці спринта. Дане поле може представляти собою код автоматизованого тесту для прийомо-сдаточного іспиту.

***Додаткові поля***

Іноді, також, використовуються додаткові поля у *product backlog*, в основному для того, щоб допомогти product owner’у визначитися з його пріоритетами.

* **Категорія (track)**. Наприклад, «панель управління» чи «оптимізація». За допомогою цього поля *product owner* може легко вибрати усі пункти категорії «оптимізація» і задати їм низький пріоритет.
* **Компоненти (components)** – указує, які компоненти (наприклад, база даних, сервер, клієнт) будуть зачеплені при реалізації історії. Дане поле складається з групи checkbox’ів, які відмічаються, якщо відповідні компоненти потребують змін.
* **Ініціатор запиту (requestor)**. *Product owner* може захотіти зберігати інформацію про усіх замовників, зацікавлених у даній задачі. Це потрібно для того, щоб тримати їх у курсі діла про хід виконання робот.
* **ID у системі обліку помилок (bug tracking ID)** – якщо ви використовуєте окрему 0<%91%D>0<%B>0<%D>0<%B>3<%D>1<%82%D>1<%80%D>0<%B>5<%D>0<%BA%D>0<%B>5<%D>1[%80"систему обліку помилок](%80%22систему%20обліку%20помилок), тоді у описі історії корисно зберігати посилання на всі дефекти, які до неї відносяться.

**Sprint backlog**

**Sprint backlog** – містить функціональність, обранню *Product Owner* із *Product Backlog*. Всі функції розбиті по задачам, кожна з яких оцінюється командою. Кожен день команда оцінює об’єм роботи, який необхідно провести для завершення задачі.

**Burndown chart**

**Burndown chart** – показує, скільки вже виконано і скільки ще залишається зробити.

**Зустрічі**

**Планування спринта (Sprint Planning Meeting)**

Проходить на початку нової ітерації Спринта.

* Із Product Backlog обираються задачі, зобов’язання по виконанню яких за спринт приймає на себе команда;
* На основі обраних задач створюється Sprint Backlog. Кожна задача оцінюється у ідеальних людино-годинах;
* Рішення задачі не повинно займати більше 12 годин або одного дня. При необхідності задача розбивається на підзадачі;
* Обговорюється та визначається, яким чином буде реалізоване цей об’єм робіт;
* Тривалість наради обмежена зверху 4-8 годинами в залежності від тривалості ітерації, досвіду команди і т.п.;
* (перша частина наради) Беруть участь Product Owner + Команда: обирають задачі із Product Backlog;
* (друга частина наради) Бере участь лише команда: обговорюють технічні деталі реалізації, наповнюють Sprint Backlog

**Щоденна нарада (Daily Scrum meeting)**

Відбувається кожен день протягом спринта. Є «пульсом» ходу спринта. Нараді властиві наступні обмеження:

* починається точно вчасно;
* всі можуть спостерігати, але говорять тільки «свині»;
* триває не більш ніж 15 хвилин;
* проводиться в одному і тому ж місці протягом одного спринта.

Протягом наради кожен член команди відповідає на 3 запитання:

* Що зроблено з моменту попередньої щоденної наради?
* Що буде зроблено з моменту поточної наради до наступної?
* Які проблеми заважають досягненню цілей спринта? (Над рішенням цих проблем працює ScrumMaster. Зазвичай це рішення проходить за рамками щоденної наради і у складі осіб, що безпосередньо займаються даною перешкодою.

**Демонстрація (Demo Meeting)**

* Проходить у кінці ітерації (спринта).
* Команда демонструє внесок функціональності до продукту всім зацікавленим особам.
* Залучається максимальна кількість глядачів.
* Усі члени команди беруть участь у демонстрації (одна людина на демонстрацію або кожен показує, що зробив за спринт).
* Обмежена 4-ма годинами в залежності від тривалості ітерації і змін у продукті.

**Ретроспектива (Retrospective Meeting)**

* Члени команди висловлюють свою думку про минулий спринт.
* Відповідають на два основних запитання:
* Що було зроблено добре у минулому спринті?
* Що потрібно покращити в наступному?
* Виконують покращення процесу розробки (вирішують питання та фіксують вдалі рішення).
* Обмежена 1-3ма годинами.

Скрам (Scrum) — это набор принципов, на которых строится процесс разработки, позволяющий в жёстко фиксированные и небольшие по времени итерации, называемые спринтами (sprints), предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет. Возможности ПО к реализации в очередном спринте определяются в начале спринта на этапе планирования и не могут изменяться на всём его протяжении. При этом строго фиксированная небольшая длительность спринта придаёт процессу разработки предсказуемость и гибкость

Спринт

Спринт[10] — итерация в скрам, в ходе которой создаётся функциональный рост программного обеспечения. Жёстко фиксирован по времени. Длительность одного спринта от 2 до 4 недель. В отдельных случаях, к примеру согласно Scrum стандарту Nokia, длительность спринта должна быть не более 6 недель. Тем не менее, считается, что чем короче спринт, тем более гибким является процесс разработки, релизы выходят чаще, быстрее поступают отзывы от потребителя, меньше времени тратится на работу в неправильном направлении. С другой стороны, при более длительных спринтах команда имеет больше времени на решение возникших в процессе проблем, а владелец проекта уменьшает издержки на совещания, демонстрации продукта и т. п. Разные команды подбирают длину спринта согласно специфике своей работы, составу команд и требований, часто методом проб и ошибок. Для оценки объема работ в спринте можно использовать предварительную оценку, измеряемую в очках истории, где 1 очко истории приблизительно равно 1 человекодню. Предварительная оценка фиксируется в резерве проекта. На протяжении спринта никто не имеет права менять список требований к работе, внесенном в резерв проекта.

*Коротко о Scrum*

Scrum — это методология управления проектом, основные особенности которой: над проектами работают небольшие кроссфункциональные команды, которые в идеале являются самоорганизующимися; работа разбивается на маленькие компоненты, которые отсортированы по приоритетам и оценены по времени; время разработки делится на итерации фиксированной длины, которые называют спринтами; в начале каждого спринта определяют минимальный функционал продукта, который будет выпущен в конце этой итерации;

После каждого спринта проводится демонстрация готового к использованию кода проводятся ежедневные короткие митинги для всей команды; постоянно анализируется и оптимизируется процесс разработки на ретроспективах.

*Что такое Kanban*

Kanban — это система организации производства, которая пришла к нам с фирмы “Тойота”, и характеризуется тремя принципами: визуализация процесса работы — работу необходимо выписать на карточки и прикрепить их на стене, подписав столбцы, чтобы видеть этапы работы; ограничение количества незавершенной работы (НЗР) — необходимо определить максимальное кол-во работы на каждом этапе; измерение времени выполнения задачи.

**Канбан в IT (Kanban Development)**

[Разработка](http://habrahabr.ru/hub/development/)\*

Я собираюсь написать несколько статей про новую методологию гибкой разработки Канбан (Kanban Development) в целях подготовки к [Scandinavian Agile Conference 2009](http://www.scan-agile.org/), где я буду делать один из докладов (кстати, заодно приглашаю всех на конференцию).   
Сегодня публикую первую из статей.  
Основная задача первой статьи — это как можно проще описать основы Канбан: что это такое, в чем отличие от других гибких методологий и зачем это нужно.  
Также я хотел бы собрать как можно больше вопросов и сомнений в комментариях, чтобы ответить на них в следующих статьях, так что пишите всё, что вам непонятно, или что ещё вы хотели бы узнать про Канбан.  
Я не то, чтобы большой специалист по этой новой методологии, но мы внутри команды пришли к Канбану самостоятельно и последовательно прошли все этапы мутации от SCRUM до Канбан, так что практический опыт есть.  
  
  
Для начала напишу про происхождение термина **Канбан**.  
  
Этот термин пришёл к нам из Японии благодаря широко известной в узких кругах [производственной системе Тойота](http://en.wikipedia.org/wiki/Toyota_Production_System). Хотелось бы, чтобы как можно больше людей прочитало про эту систему и основные принципы, заложенные в неё — бережливое производство, постоянное развитие, ориентацию на клиента и т.п. Все эти принципы описаны в книге Тайити Оно [Производственная система Тойоты](http://www.ozon.ru/context/detail/id/2142749/), которая переведена на русский.  
  
Термин Канбан имеет дословный перевод: “Кан” значит видимый, визуальный, и “бан” значит карточка или доска.  
На заводах Тойота карточки Канбан используются повсеместно для того, чтобы не загромождать склады и рабочие места заранее созданными запчастями. Например, представьте, что вы ставите двери на Тойоты Короллы. У вас около рабочего места находится пачка из 10 дверей. Вы их ставите одну за другой на новые машины и, когда в пачке остается 5 дверей, то вы знаете, что пора заказать новые двери. Вы берете карточку Канбан, пишете на ней заказ на 10 дверей и относите ее тому, кто делает двери. Вы знаете, что он их сделает как раз к тому моменту, как у вас закончатся оставшиеся 5 дверей. И именно так и происходит — когда вы ставите последнюю дверь, прибывает пачка из 10 новых дверей. И так постоянно — вы заказываете новые двери только тогда, когда они вам нужны.  
А теперь представьте, что такая система действует на всём заводе. Нигде нет складов, где запчасти лежат неделями и месяцами. Все работают только по запросу и производят именно столько запчастей, сколько запрошено. Если вдруг заказов стало больше или меньше — система сама легко подстраивается под изменения.  
  
**Основная задача карт Канбан в этой системе — это уменьшать количество «выполняющейся в данный момент работы» (work in progress).**  
Например, на всю производственную линию может быть выделено ровно 10 карточек для дверей. Это значит, что в каждый момент времени на линии не будет больше 10 готовых дверей. Когда заказывать новые двери и сколько — это задача для того, кто их устанавливает. Только он знает свои потребности, и только он может помещать заказы производителю дверей, но он всегда ограничен числом 10.  
Этот метод Бережливого производства (Lean manufacturing) был придуман в Тойоте и сейчас многие производственные компании по всему миру его внедряют или уже внедрили.  
  
Но это всё относится к производству, а не к разработке программного обеспечения.  
**А что же такое Канбан разработка применительно к ПО, и чем она отличается от других гибких методологий, буть то SCRUM или XP?**  
  
Во-первых, нужно сразу понять, что Канбан — это не конкретный процесс, а система ценностей. Как, впрочем, и SCRUM с XP. Это значит, что никто вам не скажет что и как делать по шагам.  
Во-вторых, весь Канбан можно описать одной простой фразой — **«Уменьшение выполняющейся в данный момент работы (work in progress)»**.  
В-третьих, Канбан — это даже еще более «гибкая» методология, чем SCRUM и XP. Это значит, что она не подойдет всем командам и для всех проектов. И это также значит, что команда должна быть еще более готовой к гибкой работе, чем даже команды, использующие SCRUM и XP.  
  
**Разница между Канбан и SCRUM:**  
— В Канбан нет таймбоксов ни на что (ни на задачи, ни на спринты)  
— В Канбан задачи больше и их меньше  
— В Канбан оценки сроков на задачу опциональные или вообще их нет  
— В Канбан «скорость работы команды» отсутствует и считается только среднее время на полную реализацию задачи  
  
А теперь посмотрите на этот список и задумайтесь — что остается от гибкой методологии, если мы удаляем спринты, увеличиваем размеры задач и перестаем мерять скорость работы команды? Ничего?  
Как вообще можно говорить о контроле за разработкой, если мы убираем основные инструменты контроля — сроки, скорость работы и спринты? Для меня этот вопрос является чуть ли не самым важным.  
менеджеры всегда думают о контроле и пытаются его получить, хотя на самом деле никогда его не имеют. Контроль разработки со стороны менеджера — это фикция. Если команда не хочет работать, то как ее не контролируй, она провалит проект.  
Если команда получает фан от работы и работает с полной отдачей, то никакой контроль и не нужен, а только мешает, увеличивает издержки.  
Например, общеизвестная проблема SCRUM — это большие издержки от обсуждений, встреч и большие потери времени на стыках спринтов (когда как минимум день уходит на закрытие одного спринта, а потом день на открытие нового. И если спринт — 2 недели, то 2 дня из 2 недель — это 20%, чертовски много). В итоге чуть ли не 30-40% времени при применении SCRUM тратится на поддержание самого процесса — на ежедневные митинги, на 5% workshop, на спринт ретроспектив и т.п. 30%!  
  
Канбан разработка отличается от SCRUM в первую очередь ориентацией на задачи. Если в SCRUM основная ориентация команды — это успешное выполнение спринтов (надо признать, что это так), то в Канбан на первом месте задачи.  
Спринтов никаких нет, команда работает над задачей с самого начала и до завершения. Деплоймент задачи делается тогда, когда она готова. Презентация выполненной работы — тоже. Команда не должна оценивать время на выполнение задачи, ибо это имеет мало смысла и почти всегда ошибочно вначале.  
Если менеджер верит команде, то зачем иметь оценку времени? Задача менеджера — это создать приоритизированный пул задач, а задача команды — выполнить как можно больше задач из этого пула. Всё. Никакого контроля не нужно. Всё, что нужно от менеджера — это добавлять задачи в этот пул или менять им приоритет. Именно так он управляет проектом.  
  
Команда для работы использует Канбан-доску. Например, она может выглядеть так (взял [тут](http://www.agileproductdesign.com/blog/2009/kanban_over_simplified.html)):  
  
Столбцы слева направо:  
  
**Цели проекта**:  
Необязательный, но полезный столбец. Сюда можно поместить высокоуровневые цели проекта, чтобы команда их видела и все про них знали. Например, «Увеличить скорость работы на 20%» или «Добавить поддержку Windows 7».  
  
**Очередь задач**:  
Тут хранятся задачи, которые готовы к тому, чтобы начать их выполнять. Всегда для выполнения берется верхняя, самая приоритетная задача и ее карточка перемещается в следующий столбец.  
  
**Проработка дизайна**:  
этот и остальные столбцы до «Закончено» могут меняться, т.к. именно команда решает, какие шаги проходит задача до состояния «Закончено».  
Например, в этом столбце могут находиться задачи, для которых дизайн кода или интерфейса еще не ясен и обсуждается. Когда обсуждения закончены, задача передвигается в следующий столбец.  
  
**Разработка**:  
Тут задача висит до тех пор, пока разработка фичи не завершена. После завершения она передвигается в следующий столбец.  
Или, если архитектура не верна или не точна — задачу можно вернуть в предыдущий столбец.  
  
**Тестирование**:  
В этом столбце задача находится, пока она тестируется. Если найдены ошибки — возвращается в Разработку. Если нет — передвигается дальше.  
  
**Деплоймент**:  
У всех проектов свой деплоймент. У кого-то это значит выложить новую версию продукта на сервер, а у кого-то — просто закомитить код в репозиторий.  
  
**Закончено**:  
Сюда стикер попадает только тогда, когда все работы по задаче закончены полностью.  
  
В любой работе случаются срочные задачи. Запланированные или нет, но такие, которые надо сделать прямо сейчас. Для таких можно выделить специальное место (на картинке отмечено, как «Expedite»). В Expedite можно поместить одну срочную задачу и команда должна начать ее выполнять немедленно и завершить как можно быстрее. Но может быть только одна такая задача! Если появляется еще одна — она должна быть добавлена в «Очередь задач».  
  
А теперь самое важное. Видите цифры под каждым столбцом? Это число задач, которые могут быть одновременно в этих столбцах. Цифры подбираются экспериментально, но считается, что они должны зависеть от числа разработчиков в команде.  
Например, если вы имеете 8 программистов в команде, то в строку «Разработка» вы можете поместить цифру 4. Это значит, что одновременно программисты будут делать не более 4-х задач, а значит у них будет много причин для общения и обмена опытом. Если вы поставите туда цифру 2, то 8 программистов, занимающихся двумя задачами, могут заскучать или терять слишком много времени на обсуждениях. Если поставить 8, то каждый будет заниматься своей задачей и некоторые задачи будут задерживаться на доске надолго, а ведь главная задача Канбан — это уменьшение времени прохождения задачи от начала до стадии готовности.  
Никто не даст точный ответ, какие должны быть эти лимиты, но попробуйте для начала разделить число разработчиков на 2 и посмотреть, как это работает в вашей команде. Потом эти числа можно подогнать под вашу команду.  
Под «разработчиками» я понимаю не только программистов, но и других специалистов. Например, для столбца «Тестирование» разработчики — это тестеры, т.к. тестирование — это их обязаность.  
  
Задачи на такой доске — это не просто задачи, а то, что называется Минимальной Маркетинговой Фичей, то есть фича, которую можно «продать» клиентам.  
Хорошая проверка для ММФ — это вопрос себе «А стал бы я писать про эту фичу в блоге компании?». Если нет — это не ММФ.  
  
**Что нового и полезного дает такая доска с лимитами?**  
  
Во-первых, **уменьшение числа параллельно выполняемых задач сильно уменьшает время выполнения каждой отдельной задачи.** Нет нужды переключать контекст между задачами, отслеживать разные сущности, планировать их и т.д. — делается только то, что нужно. Нет нужды устраивать спринт планнинги и 5% воркшопы, т.к. планирование уже сделано в столбце «очередь задач», а детальная проработка задачи начинается ТОЛЬКО тогда, когда задача начинает выполняться.  
  
Во-вторых, **сразу видны затыки.** Например, если тестеры не справляются с тестированием, то они очень скоро заполнят весь свой столбец и программисты, закончившие новую задачу, уже не смогут переместить ее в столбец тестирования, т.к. он заполнен. Что делать? Тут время вспомнить, что «мы — команда» и решить эту проблему. Например, программисты могут помочь тестерам завершить одну из задач тестирования и только тогда передвинуть новую задачу на освободившееся место. Это позволит выполнить обе задачи быстрее.  
  
В-третьих, можно вычислить время на выполнение усредненной задачи. Мы можем помечать на карточке дату, когда она попала в очередь задач, потом дату, когда ее взяли в работу и дату, когда ее завершили. По этим трем точкам для хотя бы 10 задач можно уже посчитать среднее время ожидания в очередь задач и среднее время выполнения задачи. А из этих цифр менеджер или product owner может уже рассчитывать всё, что ему угодно.  
  
Весь Канбан можно описать всего тремя основными правилами:  
1. **Визуализируйте производство**  
— Разделите работу на задачи, каждую задачу напишите на карточке и поместите на стену или доску.  
— Используйте названные столбцы, чтобы показать положение задачи в производстве.  
2. **Ограничивайте WIP** (work in progress или работу, выполняемую одновременно) **на каждом** этапе производства.  
3. **Измеряйте время цикла** (среднее время на выполнение одной задачи) и **оптимизируйте постоянно процесс**, чтобы уменьшить это время.  
  
**Всего 3 правила!**  
Например, в SCRUM — 9 базовых правил. В XP — 13, а в классическом RUP — аж более 120. Почувствуйте разницу.  
  
На этом я закончу первую статью про Канбан.

*Разница между Канбан и SCRUM:*

— В Канбан нет таймбоксов ни на что (ни на задачи, ни на спринты)

— В Канбан задачи больше и их меньше

— В Канбан оценки сроков на задачу опциональные или вообще их нет

— В Канбан «скорость работы команды» отсутствует и считается только среднее время на полную реализацию задачи

*Microsoft Solutions Framework (MSF)* — методология разработки программного обеспечения, предложенная корпорацией Microsoft. MSF опирается на практический опыт Microsoft и описывает управление людьми и рабочими процессами в процессе разработки решения.

Модель проектной группы MSF

Модель проектной группы MSF (MSF Team Model) описывает подход Майкрософт к организации работающего над проектом персонала и его деятельности в целях максимизации успешности проекта. Данная модель определяет ролевые кластеры, их области компетенции и зоны ответственности, а также рекомендации членам проектной группы, позволяющие им успешно осуществить свою миссию по воплощению проекта в жизнь.

Модель проектной группы MSF разрабатывалась в течение нескольких лет и возникла в результате осмысления недостатков пирамидальной, иерархической структуры традиционных проектных групп.

В соответствии с моделью MSF проектные группы строятся как небольшие многопрофильные команды, члены которых распределяют между собой ответственность и дополняют области компетенций друг друга. Это дает возможность четко сфокусировать внимание на нуждах проекта. Проектную группу объединяет единое видение проекта, стремление к воплощению его в жизнь, высокие требования к качеству работы и желание самосовершенствоваться.

Ниже описываются основные принципы, ключевые идеи и испытанные методики MSF в применении к модели проектной группы.

MSF включает в себя ряд основных принципов. Вот те из них, которые имеют отношение к успешной работе команды:

Распределение ответственности при фиксации отчетности

Наделяйте членов команды полномочиями

Концентрируйтесь на бизнес-приоритетах

Единое видение проекта

Проявляйте гибкость — будьте готовы к переменам

Поощряйте свободное общение

Успешное использование модели проектной группы MSF основывается на ряде ключевых концепций (key concepts):

Команда соратников

Сфокусированность на нуждах заказчика

Нацеленность на конечный результат

Установка на отсутствие дефектов

Стремление к самосовершенствованию

Заинтересованные команды работают эффективно

MSF основан на постулате о шести качественных целях, достижение которых определяет успешность проекта. Эти цели обуславливают модель проектной группы. В то время как за успех проекта ответственна вся команда, каждый из её ролевых кластеров, определяемых моделью, ассоциирован с одной из упомянутых шести целей и работает над её достижением.

В проектную группу входят такие ролевые кластеры:

управление программой

управление продуктом

разработка

тестирование

управление релизом

удовлетворение потребителя

Они ответственны за различные области компетенции (functional areas) и связанные с ними цели и задачи. Иногда ролевые кластеры называются просто ролями. Но в любом случае суть концепции остается той же — построить основу производственных отношений и связанную с ней модель команды такими, чтобы они были приспосабливаемыми (масштабируемыми) для удовлетворения нужд любого проекта.

Как уже было сказано выше, проектная группа по MSF состоит из шести ролевых кластеров, каждый из которых отвечает за:

управление программой (program manager) — разработку архитектуры решения, административные службы;

разработку (developer) — разработку приложений и инфраструктуры, технологические консультации;

тестирование (QAE) — планирование, разработку тестов и отчетность по тестам;

управление выпуском (release manager) — инфраструктуру, сопровождение, бизнес-процессы, выпуск готового продукта;

удовлетворение заказчика (user experience) — обучение, эргономику, графический дизайн, техническую поддержку;

управление продуктом (product manager) — бизнес-приоритеты, маркетинг, представительство интересов заказчика.

Наличие шести ролевых кластеров не означает, что количество членов команды должно быть кратным шести — один человек может совмещать несколько ролей и наоборот, ролевой кластер может состоять из нескольких лиц в зависимости от размера проекта, его сложности и профессиональных навыков, требуемых для реализации всех областей компетенции кластера. Минимальный коллектив по MSF может состоять всего из трех человек. Модель не требует назначения отдельного сотрудника на каждый ролевой кластер. Смысл состоит в том, что в команде должны быть представлены все шесть качественных целей. Обычно, выделение как минимум одного человека на каждый ролевой кластер обеспечивает полноценное внимание к интересам каждой из ролей, но это экономически оправданно не для всех проектов. Зачастую члены проектной группы могут объединять роли.

В малых проектных группах объединение ролей является необходимым. При этом должны соблюдаться два принципа:

Роль команды разработчиков не может быть объединена ни с какой другой ролью.

Избежание сочетания ролей, имеющих предопределенные конфликты интересов.

Как и в любой другой командной деятельности, подходящая комбинация ролей зависит от самих членов команды, их опыта и профессиональных навыков. На практике совмещение ролей встречается нередко. И если проектная группа производит его обдуманно и управляет связанными с таким объединением рисками, возникающие проблемы будут минимальными.

MSF не предоставляет конкретных рецептов управления проектами и не содержит объяснений разнообразных методов работы, которые применяют опытные менеджеры. Принципы MSF формируют такой подход к управлению проектами, при котором:

ответственность за управление проектом распределенная между лидерами ролевых кластеров внутри команды — каждый член проектной группы отвечает за общий успех проекта и качество создаваемого продукта.

профессиональные менеджеры выступают в качестве консультантов и наставников команды, а не выполняют функции контроля над ней — в эффективно работающей команде каждый её член имеет необходимые полномочия для выполнения своих обязанностей и уверен, что получит от коллег все необходимое.

Как следует из вышесказанного, одна из характерных особенностей MSF — отсутствие должности менеджера проекта!

Модель проектной группы MSF предлагает разбиение больших команд (более 10 человек) на малые многопрофильные группы направлений (feature teams). Эти малые коллективы работают параллельно, регулярно синхронизируя свои усилия. Кроме того, когда ролевому кластеру требуется много ресурсов, формируются т. н. функциональные группы (functional teams), которые затем объединяются в ролевые кластеры.

Использование ролевых кластеров не подразумевает и не навязывает никакой специальной структуры организации или обязательных должностей. Административный состав ролей может широко варьироваться в разных организациях и проектных группах. Чаще всего роли распределяются среди различных подразделений одной организации, но иногда часть их отводится сообществу потребителей или внешним по отношению к организации консультантам и партнерам. Ключевым моментом является четкое определение работников, ответственных за каждый ролевой кластер, их функций, ответственности и ожидаемого вклада в конечный результат.

Модель проектной группы MSF не обеспечивает успех сама по себе. Есть много других факторов, определяющих успех или неудачу проекта, но структура проектной группы, безусловно, вносит существенный вклад.

Подходящая структура команды является фундаментом успеха, и реализация модели MSF с использованием лежащих в её основе принципов поможет сделать проектные группы более эффективными и, как следствие, более успешными.

Модель процессов MSF

Модель процессов MSF (MSF process model) представляет общую методологию разработки и внедрения IT решений. Особенность этой модели состоит в том, что благодаря своей гибкости и отсутствию жестко навязываемых процедур она может быть применена при разработке весьма широкого круга IT проектов. Эта модель сочетает в себе свойства двух стандартных производственных моделей: каскадной (waterfall) и спиральной (spiral). Модель процессов в MSF 3.0 была дополнена ещё одним инновационным аспектом: она покрывает весь жизненный цикл создания решения, начиная с его отправной точки и заканчивая непосредственно внедрением. Такой подход помогает проектным группам сфокусировать свое внимание на бизнес-отдаче (business value) решения, поскльку эта отдача становится реальной лишь после завершения внедрения и начала использования продукта.

Процесс MSF ориентирован на «вехи» (milestones) — ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата. Этот результат может быть оценен и проанализирован, что подразумевает ответы на вопросы: «Пришла ли проектная группа к однозначному пониманию целей и рамок проекта?», «В достаточной ли степени готов план действий?», «Соответствует ли продукт утвержденной спецификации?», «Удовлетворяет ли решение нужды заказчика?» и т. д.

Модель процессов MSF учитывает постоянные изменения проектных требований. Она исходит из того, что разработка решения должна состоять из коротких циклов, создающих поступательное движение от простейших версий решения к его окончательному виду.

Модель процессов MSF тесно связана с базовыми принципами MSF, рассмотренными выше. Вообще говоря, тремя особенностями модели процессов MSF являются:

Подход, основанный на фазах и вехах.

Итеративный подход.

Интегрированный подход к созданию и внедрению решений.

Модель процессов включает такие основные фазы процесса разработки:

Выработка концепции (Envisioning)

Планирование (Planning)

Разработка (Developing)

Стабилизация (Stabilizing)

Внедрение (Deploying)

Кроме этого существует большое количество промежуточных вех, которые показывают достижение в ходе проекта определенного прогресса и расчленяют большие сегменты работы на меньшие, обозримые участки. Для каждой фазы модели процессов MSF определяет:

что (какие артефакты) является результатом этой фазы

над чем работает каждый из ролевых кластеров на этой фазе

В рамках MSF программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами. MSF рекомендует начинать разработку решения с построения, тестирования и внедрения его базовой функциональности. Затем к решению добавляются все новые и новые возможности. Такая стратегия именуется стратегией версионирования. Несмотря на то, что для малых проектов может быть достаточным выпуск одной версии, рекомендуется не упускать возможности создания для одного решения ряда версий. С созданием новых версий эволюционирует функциональность решения.

Итеративный подход к процессу разработки требует использования гибкого способа ведения документации. «Живые» документы (living documents) должны изменяться по мере эволюции проекта вместе с изменениями требований к конечному продукту. В рамках MSF предлагается ряд шаблонов стандартных документов, которые являются артефактами каждой стадии разработки продукта и могут быть использованы для планирования и контроля процесса разработки.

Решение не представляет бизнес-ценности, пока оно не внедрено. Именно по этой причине модель процессов MSF содержит весь жизненный цикл создания решения, включая его внедрение — вплоть до момента, когда решение начинает давать отдачу.

* Розуміє та практично застосовує equivalence partitioning, boundary value analysis, decision tables, state transition diagrams для планування та оцінки виконання функціонального тестування.

3.2.1 Эквивалентное разделение <приложения> (Equivalence partitioning)

Рассматриваемая область приложения разделяется на коллекцию наборов или эквивалентных классов, которые считаются эквивалентными с точки зрения рассматриваемых связей и характеристик <спецификации>. Репрезентативный набор тестов (иногда – только один тест) формируется из тестов эквивалентных классов (или наборов классов).

3.2.2 Анализ граничных значений (Boundary-value analysis)

Тесты строятся с ориентацией на использование тех величин, которые определяют предельные характеристики тестируемой системы. Расширением этой техники являются тесты оценки живучести (robustness testing) системы, проводимые с величинами, выходящими за рамки специфицированных пределов значений.

3.2.3 Таблицы принятия решений (Decision table)

Такие таблицы представляют логические связи между условиями (могут рассматриваться в качестве “входов”) и действиями (могут рассматриваться как “выходы”). Набор тестов строится последовательным рассмотрением всех возможных кросс-связей в такой таблице.

3.2.4 Тесты на основе конечного автомата (Finite-state machine-based)

Строятся как комбинация тестов для всех состояний и переходов между состояниями, представленных в соответствующей модели (переходов и состояний приложения).

3.2.5 Тестирование на основе формальной спецификации (Testing from formal specification)

Для спецификации, определенных с использованием формального языка, возможно автоматически создавать и тесты для функциональных требований. В ряде случаев могут строится на основе модели, являющейся частью спецификации, не использующей формального языка описания.

3.2.6 Случайное тестирование (Random testing)

В отличие от статистического тестирования (будет рассматриваться в 3.5.1 “Operational profile”), сами тесты генерируются случайным образом по списку заданного набора специфицированных характеристик.

Техники тест дизайна (Test Design Technics)

Многие люди тестируют и пишут тестовые случаи (test cases), но не многие пользуются специальными техниками тест дизайна. Постепенно, набираясь опыта они осознают, что постоянно делают одну и ту же работу, поддающуюся конкретным правилам. И тогда они находят, что все эти правила уже описаны.

Предлагаем вам ознакомиться с кратким описанием наиболее распространенных техник тест дизайна:

Эквивалентное Разделение (Equivalence Partitioning - EP). Как пример, у вас есть диапазон допустимых значений от 1 до 10, вы должны выбрать одно верное значение внутри интервала, скажем, 5, и одно неверное значение вне интервала - 0.

Анализ Граничных Значений (Boundary Value Analysis - BVA). Если взять пример выше, в качестве значений для позитивного тестирования выберем минимальную и максимальную границы (1 и 10), и значения больше и меньше границ (0 и 11). Анализ Граничный значений может быть применен к полям, записям, файлам, или к любого рода сущностям имеющим ограничения.

Причина / Следствие (Cause/Effect - CE). Это, как правило, ввод комбинаций условий (причин), для получения ответа от системы (Следствие). Например, вы проверяете возможность добавлять клиента, используя определенную экранную форму. Для этого вам необходимо будет ввести несколько полей, таких как "Имя", "Адрес", "Номер Телефона" а затем, нажать кнопку "Добавить" - эта "Причина". После нажатия кнопки "Добавить", система добавляет клиента в базу данных и показывает его номер на экране - это "Следствие".

Предугадывание ошибки (Error Guessing - EG). Это когда тест аналитик использует свои знания системы и способность к интерпретации спецификации на предмет того, чтобы "предугадать" при каких входных условиях система может выдать ошибку. Например, спецификация говорит: "пользователь должен ввести код". Тест аналитик, будет думать: "Что, если я не введу код?", "Что, если я введу неправильный код? ", и так далее. Это и есть предугадывание ошибки.

Исчерпывающее тестирование (Exhaustive Testing - ET) - это крайний случай. В пределах этой техники вы должны проверить все возможные комбинации входных значений, и в принципе, это должно найти все проблемы. На практике применение этого метода не представляется возможным, из-за огромного количества входных значений.

*эквивалентное разбиение (equivalence partitioning):* Разработка тестов методом черного ящика, в которой тестовые сценарии создаются для проверки элементов эквивалентной области. Как правило, тестовые сценарии разрабатываются для покрытия каждой области как минимум один раз.

эквивалентная область (equivalence partition): Часть области входных или выходных данных, для которой поведение компонента или системы, основываясь на спецификации, считается одинаковым.

раздел эквивалентности - 39 (при тестировании ПО - раздел входных данных, эквивалентных в том смысле, что результат тестирования программы (приложения) с использованием любого одного значения этого диапазона будет репрезентативным для всего диапазона. Благодаря такому подходу можно существенно уменьшить объём тестирования без ущерба для качества. Различают действительные и недействительные диапазоны эквивалентных значений; например, для номеров месяцев в году действительный диапазон - от 1 до 12, а диапазоны 0, ..., -k и 13, ..., m являются недействительными. Выбор подобных разделов производится обычно по спецификации приложения и призван обеспечить уменьшение числа используемых контрольных примеров (test case) до необходимого минимума, при этом примеры должны охватывать все возможные сценарии тестирования

*Эквивалентное Разделение* (Equivalence Partitioning - EP). Как пример, у вас есть диапазон допустимых значений от 1 до 10, вы должны выбрать одно верное значение внутри интервала, скажем, 5, и одно неверное значение вне интервала - 0.

*Анализ Граничных Значений* (Boundary Value Analysis - BVA). Если взять пример выше, в качестве значений для позитивного тестирования выберем минимальную и максимальную границы (1 и 10), и значения больше и меньше границ (0 и 11). Анализ Граничный значений может быть применен к полям, записям, файлам, или к любого рода сущностям имеющим ограничения.

*Таблица принятия решений (таблица решений)* — способ компактного представления модели со сложной логикой. Аналогично условным операторам в языках программирования, они устанавливают связь между условиями и действиями. Но, в отличие от традиционных языков программирования, таблицы решений в простой форме могут представлять связь между множеством независимых условий и действий.

Таблицы принятия решений, как правило, разделяются на четыре квадранта, как показано ниже.

Условия Варианты выполнения условий

Действия Необходимость действий

В простейшем случае здесь Условия — список возможных условий, Варианты выполнения условий — комбинация из выполнения и/или невыполнения условий из этого списка. Действия — список возможных действий, Необходимость действий — указание надо или не надо выполнять соответствующее действие для каждой из комбинаций условий. Например для ситуации «свет погас»Свет в соседней комнате горит Да Нет Нет

Свет у соседей горит - Да Нет

Поменять лампочку Х

Проверить пробки Х

Позвонить электрику Х Х

Позвонить диспетчеру Х

Вариантов выполнения условия может быть не два: да или нет, а несколько, например цвет может быть красным, оранжевым, синим. В более сложных таблицах может применяться нечёткая логика.

Действия могут быть элементарными или ссылаться на другие таблицы принятия решений. Необходимость выполнения действий может быть неупорядоченной, как в данном примере, или упорядоченной. В последнем случае если при определённой комбинации выполнения условий возможно выполнение нескольких действий, то в таблице решений указывается их приоритет.

*Диагра́мма состоя́ний* — ориентированный граф для конечного автомата, в котором

-вершины обозначают состояния

-дуги показывают переходы между двумя состояниями

На практике вершины обычно изображаются в виде окружностей и, если нужно, двойных окружностей. В нотации UML состояния изображаются прямоугольниками с закругленными углами[1].

 S1 и S2 — состояния. Дуги помечены входными данными.

* Може пояснити переваги та можливі ризики незалежного / залежного тестування.

**\*Розуміє на яких конфігураціях (або в яких середовищах) має проводитись тестування.**

**What is Quality?**

·         Customer satisfaction? Subjective term. It will depend on who the ‘customer’ is. Each type of customer will have their own view on ‘quality’

**What is Software Quality?**

·         Measurement of how close is actual software product to the expected (intended) product

·         Customer satisfaction (to who?)

·    Quality Software: reasonably bug-free, delivered on time and within budget, meets requirements and/or expectations, and is maintainable

**What is Software Quality Assurance?**

·    Software QA is the process of monitoring and improving all activities associated with software development, from requirements gathering, design and reviews to coding, testing and implementation.

**What is the difference between Software Testing and Software QA?**

·    Testing is mainly an error detection process

·    Software QA is preventative . It aims to ensure quality in the methods & processes. (“Quality Assurance”  measures the quality of processes used to create a quality product)

**What is Software Testing?**

·    Software Testing is the process of analyzing the software in order to detect the differences between existing and required conditions and to evaluate the features of the software.  It involves the entire software development process:

            - monitoring and improving the process

            - making sure that any agreed-upon standards and procedures are followed

            - ensuring that problems are found and dealt with, at the earliest possible stage

·    The purpose of testing is verification, validation and error detection (in order to find and fix the problems)  
- Verification is checking for conformance and consistency by evaluating the results against pre-specified requirements. (Verification: Are we building the system right?)  
- Validation is the process of checking that what has been specified is what the user actually wanted. (Validation: Are we building the right system?)  
- Error Detection: finding if things happen when they shouldn’t or things don’t happen when they should.

**Is it possible to find/fix all the bugs in a software product before it goes to the customers? Why test?**

·         To establish and to enforce business systems of the QA Organization (Test planning, bug tracking, bug reporting, test automation, release certification, and others)

**What is black/white box testing?**

·         Black box software testing is done without access to the source code.

·         White box testing is done with access to the code. Bugs are reported at the source code level, not behavioral.

**Describe a bug?**

·         Mismatch between actual behavior of a software application and its intended (expected) behavior. We learn about expected behavior from requirements, specifications, other technical documentation.

**What is use case?**

·         Use cases are used by Business Analysts as a format for specifying system requirements. Each use case represents completed business operation performed by user.  From the QA prospective we would need to execute End-To-End test to make sure the requirement is implemented.

·         Find more here: <http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci334062,00.html>

**What is the most important impact QA can have on a product development process?**

·         Clarifying requirements

·         Bringing down percentage of code re-written due to the change in requirements

**What is Negative testing? Positive?**

·   Positive testing aimed at showing software works as intended when user does what he/she does correct actions.

·   Negative testing aimed at showing that software handles properly situations in which user acts not as user is supposed to act (invalid inputs, unreasonable selections of settings, etc.)

**Which type of testing results in highest number of bugs found?**

·         Negative testing (versus Positive testing of same type)

**What is the software development life cycle?**

·         The software development life cycle (SDLC) is a conceptual model used in project management that describes the stages involved in an information system development project, from an initial feasibility study through maintenance of the completed application. Read more - <http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci755068,00.html>

·         And more - <http://portnov.readyhosting.com/RU/sdlc.html>

**What is a Test Case?**

* Set of conditions and/or variables under which a tester will determine if a requirement upon an application is satisfied

**What does Test Case include?**

When planning for testing the test case:

* Test case ID
* The purpose (Title, Description) of the test case
* An instruction on how to get from the **application base state** **to a verifiable application output or expected result**
* Expected result

When execute test cases we need two more columns:

* Actual result
* PASS/FAIL indication

**What is a test plan?**

* **Document that describes the objectives, scope, approach, and focus of a software testing effort.**
* The process of preparing a test plan is a useful way to think through the efforts needed to validate the acceptability of a software product. The completed document will help people outside the test group understand the 'why' and 'how' of product validation. It should be thorough enough to be useful but not so thorough that no one outside the test group will read it.

**What does Test Plan include?**

The following are some of the items that might be included in a test plan, depending on the particular project:

\* Title

\* Identification of software including version/release numbers

\* Revision history of document including authors, dates, approvals

\* Table of Contents

\* Purpose of document, intended audience

\* Objective of testing effort

**\* Software product overview**

\* Relevant related document list, such as requirements, design documents, other test plans, etc.

\* Relevant standards or legal requirements

\* Traceability requirements

\* Relevant naming conventions and identifier conventions

\* Overall software project organization and personnel/contact-info/responsibilities

\* Test organization and personnel/contact-info/responsibilities

\* Assumptions and dependencies

\* Project risk analysis

**\* Testing priorities and focus**

**\* Scope and limitations of testing**

\* Test outline - a decomposition of the test approach by test type, feature, functionality, process, system, module, etc. as applicable

\* Outline of data input equivalence classes, boundary value analysis, error classes

\* Test environment - hardware, operating systems, other required software, data configurations, interfaces to other systems

\* Test environment validity analysis - differences between the test and production systems and their impact on test validity.

\* Test environment setup and configuration issues

\* Software migration processes

\* Software CM processes

\* Test data setup requirements

\* Database setup requirements

\* Outline of system-logging/error-logging/other capabilities, and tools such as screen capture software, that will be used to help describe and report bugs

\* Discussion of any specialized software or hardware tools that will be used by testers to help track the cause or source of bugs

\* Test automation - justification and overview

**\* Test tools to be used, including versions, patches, etc.**

\* Test script/test code maintenance processes and version control

\* Problem tracking and resolution - tools and processes

\* Project test metrics to be used

\* Reporting requirements and testing deliverables

\* Software entrance and exit criteria

\* Initial sanity testing period and criteria

\* Test suspension and restart criteria

**\* Personnel allocation**

\* Personnel pre-training needs

\* Test site/location

\* Outside test organizations to be utilized and their purpose, responsibilities, deliverables, contact persons, and coordination issues

\* Relevant proprietary, classified, security, and licensing issues

\* Open issues

\* Appendix - glossary, acronyms, etc.

**Write test cases for a text field?**

* 5 test cases for capacity including 2 for each boundary and one for the class between boundaries
* 3 test cases for valid/invalid input of letters, digits, special characters
* One test cases for each allowed special character (email field as an example)
* Functionality testing if there is any functionality (validation of input as an example, case sensitivity, required field, etc.)

**What is Test matrix**

Data collection mechanism. It provides a structure for testing the effect of combining two or more variables, circumstances, types of hardware, or events. Row and column headings identify the test conditions. Cells keep the results of test execution.

**If there are so many settings/options to choose, how to write test cases?**

* Test cases should be developed for all most common potential scenarios
* They should cover most of the positive input

**Beside test case & test plan, what documents are required to write?**

·        Check Lists

·        Test matrices

·        Test design specs

·        End-to-end tests

·        Test summary reports

·        Bug reports

**Describe risk analysis**

Risk analysis means the actions taken to avoid things going wrong on a software development project, things that might negatively impact the scope, quality, timeliness, or cost of a project. This is, of course, a shared responsibility among everyone involved in a project. However, there needs to be a 'buck stops here' person who can consider the relevant tradeoffs when decisions are required, and who can ensure that everyone is handling their risk management responsibilities.

**How will you write test cases for testing fields LOGIN & PASSOWRD, positive and negative testing?**

**Testing boundary conditions? Why? How?**

* **Boundary value analysis is a methodology for designing test cases that concentrates software testing effort on cases near the limits of valid ranges.**
* Boundary value analysis is a method which refines equivalence partitioning. It generates test cases that highlight errors better than equivalence partitioning. The trick is to concentrate software testing efforts at the extreme ends of the equivalence classes. At those points when input values change from valid to invalid errors are most likely to occur. As well, boundary value analysis broadens the portions of the business requirement document used to generate tests.

For example, if a valid range of quantity on hand is -9,999 through 9,999, write test cases that include:

1. the valid test case quantity on hand is -9,999,

2. the valid test case quantity on hand is 9,999,

3. the invalid test case quantity on hand is -10,000 and

4. the invalid test case quantity on hand is 10,000

**What is the difference between a test case and a test plan?**

* Test plan is the most comprehensive Software Testing document that describes the **objectives, scope, approach, and focus of a software testing effort**
* Test case is the smallest Software Testing document that describes both typical and atypical situation (set of conditions and/or variables) that may occur in the **use** of an application (under which a tester will determine if a requirement upon an application is satisfied).

**Which documents would you refer to when creating Test Cases?**  
All business and technical documentation available:

- PRD - Product Requirements Document

- BRD - Business Requirements Document

- Functional Specifications

- Manuals and Help

- Use Cases

- Test Design

- Third party publications (books, published by independent authors)

**What is Business Requirements Document (BRD)?**  
BRD is written by the Business Analysts. It details the business solution for a project including the documentation of customer needs and expectations.

The most common objectives of the BRD are:  
- To gain agreement with stakeholders   
- To provide a foundation to communicate to a technology service provider what the solution needs to do to satisfy the customer's and business’ needs   
- To provide input into the next phase for this project   
- To describe what not how the customer/business needs will be met by the solution

***Testing documentation***

***Практичне застосування:***

* **\*Розробляє функціональні тест кейси згідно специфікацій продукту, розробка сценаріїв тестування, що покривають логічні шляхи використання продукту.**
* **\*Здійснює періодичну корекцію, оновлення та вдосконалення тест-кейсів.**
* **\*Здатен оцінити повноту тестового покриття окремої функціональної області, проаналізувати вимоги до функціональності на чіткість, несуперечливість, повноту.**
* **\*Здатен розподілити пріоритети серед набору тестових задач.  
  Заповнює PL**
* Знайомий із набором тестової документації згідно із стандартом IEEE 829. (Розповісти)

IEEE 829-2008, also known as the 829 Standard for Software and System Test Documentation, is an IEEE standard that specifies the form of a set of documents for use in eight defined stages of software testing, each stage potentially producing its own separate type of document. The standard specifies the format of these documents but does not stipulate whether they all must be produced, nor does it include any criteria regarding adequate content for these documents. These are a matter of judgment outside the purview of the standard. The documents are:

-Test Plan: a management planning document that shows:

- How the testing will be done (including SUT (system under test) configurations).

- Who will do it

- What will be tested

- How long it will take (although this may vary, depending upon resource availability).

- What the test coverage will be, i.e. what quality level is required

-Test Design Specification: detailing test conditions and the expected results as well as test pass criteria.

-Test Case Specification: specifying the test data for use in running the test conditions identified in the Test Design Specification

-Test Procedure Specification: detailing how to run each test, including any set-up preconditions and the steps that need to be followed

-Test Item Transmittal Report: reporting on when tested software components have progressed from one stage of testing to the next

-Test Log: recording which tests cases were run, who ran them, in what order, and whether each test passed or failed

-Test Incident Report: detailing, for any test that failed, the actual versus expected result, and other information intended to throw light on why a test has failed. This document is deliberately named as an incident report, and not a fault report. The reason is that a discrepancy between expected and actual results can occur for a number of reasons other than a fault in the system. These include the expected results being wrong, the test being run incorrectly, or inconsistency in the requirements meaning that more than one interpretation could be made. The report consists of all details of the incident such as actual and expected results, when it failed, and any supporting evidence that will help in its resolution. The report will also include, if possible, an assessment of the impact of an incident upon testing.

-Test Summary Report: A management report providing any important information uncovered by the tests accomplished, and including assessments of the quality of the testing effort, the quality of the software system under test, and statistics derived from Incident Reports. The report also records what testing was done and how long it took, in order to improve any future test planning. This final document is used to indicate whether the software system under test is fit for purpose according to whether or not it has met acceptance criteria defined by project stakeholders.

Relationship with other standards

Other standards that may be referred to when documenting according to IEEE 829 include:

IEEE 1008, a standard for unit testing

IEEE 1012, a standard for Software Verification and Validation

IEEE 1028, a standard for software inspections

IEEE 1044, a standard for the classification of software anomalies

IEEE 1044-1, a guide to the classification of software anomalies

IEEE 830, a guide for developing system requirements specifications

IEEE 730, a standard for software quality assurance plans

IEEE 1061, a standard for software quality metrics and methodology

IEEE 12207, a standard for software life cycle processes and life cycle data

BS 7925-1, a vocabulary of terms used in software testing

BS 7925-2, a standard for software component testing

Use of IEEE 829

The standard forms part of the training syllabus of the ISEB Foundation and Practitioner Certificates in Software Testing promoted by the British Computer Society. ISTQB, following the formation of its own syllabus based on ISEB's and Germany's ASQF syllabi, also adopted IEEE 829 as the reference standard for software and system test documentation.

IEEE 829-2008, також відомий як стандартний 829 Документація тестів програмного забезпечення і системи, є стандартним IEEE, який визначає вид комплекту документів для використання у восьми певних етапів тестування програмного забезпечення, кожен етап потенційно виробництво власного окремого виду документа. Стандарт визначає формат цих документів, але не передбачають Чи вони всі повинні бути зроблені, і не включають будь-які критерії, що стосуються адекватного змісту цих документів. Це питання здорового глузду поза увагою стандарту. Документи:  
-**Test Plan:** управління документами планування, який показує:  
         - Як тестування буде зроблено (в тому числі SUT (тестованої системи) конфігурації).  
         - Хто буде робити це  
         - Що буде перевірена  
         - Скільки часу це займе (хоча це може варіюватися, залежно від наявності ресурсів).  
          - Що тестового покриття буде, тобто те, що рівень якості необхідно  
**Дизайн-Test Specification**: докладні умови випробувань і очікувані результати, а також критерії пройти тест.  
**-Test Case Специфікація:** визначення тестових даних для використання в виконанні умов випробувань, зазначених у специфікації проектування тестів  
**-Процедура тестування** Специфікація: докладно, як запустити кожного тесту, в тому числі будь-які налаштування передумови і заходи, які мають бути виконані  
**Випробувань Пункт Передача звіту:** звіт про те, коли перевірені програмні компоненти пройшли шлях від однієї стадії тестування до наступного  
**-Test Log:** запис, яка перевіряє випадки були запущені, який керував ним, в якому порядку, і чи є кожен тест пройдено чи ні  
**-Test Звіт про інцидент:** деталізація, для будь-якого тесту, який не вдалося, фактичних і очікуваних результатів, а також інші відомості, що дозволяють пролити світло на те, чому тест не вдався. Цей документ навмисно названий звіт про інцидент, і не вина доповіді. Причина в тому, що розбіжності між очікуваними і фактичними результатами може виникнути з ряду причин, крім несправності в системі. Вони включають в себе очікувані результати помилитися, тест час працювати неправильно, або непослідовність у вимогах означає, що більше одного тлумачення може бути зроблено. Звіт складається з усіх деталей інциденту, таких як фактичних і очікуваних результатів, коли це не вдалося, і будь-які підтверджуючі докази, які допоможуть у його вирішенні. У доповіді буде також включати, якщо це можливо, оцінку впливу падаючих на тестування.  
**-Test Зведений звіт**: Управління доповідь, що містить будь-яку важливу інформацію розкриті тести виконані, і в тому числі оцінки якості робіт з тестування, якість програмного забезпечення системи тестування і статистичних даних, отриманих зі звітів про інциденти. У доповіді також записує те, що тестування було зроблено, і скільки часу знадобилося для того, щоб поліпшити майбутні планування тестування. Цей підсумковий документ використовується для позначення того, програмне забезпечення тестованої системи, придатний для цілей залежно від того, чи не він зустрівся з прийняттям критеріїв, визначених зацікавлених сторін проекту.  
Взаємозв'язок з іншими стандартами

***Following established processes***

***Теоретичне розуміння:***

* Самостійно та адекватно встановлює пріоритети у власній роботі***.***

***Практичне застосування:***

* **\*Здійснює тестування, застосовуючи декілька видів тестування.**
* **\*Реєструє знайдені дефекти згідно правил.**
* **\*Вміє визначати закономірності у знайдених дефектах.**

**\*Може перевірити тестове середовище на правильність конфігурування.**(Уточнюючі питання відповідно до специфіки проекту).  
**Відгуки PL**

***Evaluating test results/decision making***

***Теоретичне розуміння:***

* **\*Вміє (на основі специфікації чи власного досвіду) пояснити різницю між дефектом та запланованим функціоналом (фіча).**

Программным дефектом называется ошибка в программном продукте, вследствие которой продукт ведет себя непредвиденно (неверно). Большинство дефектов возникают из-за допущенной ошибки в программном коде или логической ошибки, допущенной во время проектирования. Гораздо меньшее количество - вследствие ошибок работы инструментальных средств (компилятора, генератора кода).

Дефекты делят на следующие категории

Арифметические ошибки (деление на 0, переполнение)

Логические ошибки (бесконечные циклы, бесконечная рекурсия)

Ошибки работы с ресурсами (null pointer, access violation, переполнение буфера)

Ошибки синхронизации многопоточности (дедлок, race condition)

Поскольку количество дефектов в большинстве программных продуктом достаточно велико, для учета дефектов применяют специальный класс систем - систему учета запросов (Issue Tracking System). Данная система кроме учета дефектов ведет еще учет требований продукта, релизов и т.д.

**Навести приклади/Відгук PL**

* **\*Вміє зробити висновок/оцінку про готовність функціональності, що тестувалася до здачі.  
  Відгук PL**
* **\*Вміє підсумувати виконану тестову роботу та дати висновок про стан тестованого функціоналу (що і як тестував, що не тестував, і які висновки можна зробити в результаті)**

**Уточнюючі питання/ PL**

***Improving processes, innovating***

***Практичне застосування:***

Висуває пропозиції щодо покращення процесу тестування, шаблонів документів, процесу обміну інформацією в середині команди.  
**Відгук PL**

***Performing testing***

***Практичне застосування:***

* **\*Вміє визначити пріоритет виконання завдань для себе.**  
  **Відгук PL**
* **\*Вміє узагальнити шляхи до знайдення дефекту.** **Відгук PL**
* При повторному тестуванні виявляє специфічні прояви більш загального дефекту. **Відгук PL**
* **\*Здатен знайти дефекти, породжені виправленням попереднього.**  
  **Відгук PL**
* Здатен визначити набори тестових даних для виконання тестів. **Відгук PL**

***Test management (Communicating in team/with client)***

**\*Спілкується з розробниками та керівником, аргументовує свою позицію (листами/ усно/ трекінг систему)**.  
**Відгук PL**

***Практичні навички роботи з ПК***

***Загальне:***

* **\*Bug tracking tool**

Система отслеживания ошибок (англ. bug tracking system) — прикладная программа, разработанная с целью помочь разработчикам программного обеспечения (программистам, тестировщикам и др.) учитывать и контролировать ошибки (баги), найденные в программах, пожелания пользователей, а также следить за процессом устранения этих ошибок и выполнения или невыполнения пожеланий.

Redmine, BUGS - the Bug Genie, Bugzilla, eTraxis, GNATS, Mantis bug tracking system, Trac, EmForge, Picket, Flyspray, DEVPROM, YouTrack

* Code version control tool

Система управления версиями (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).

Программное обеспечение Википедии ведёт историю изменений для всех её статей, используя методы, аналогичные тем, которые применяются в системах управления версиями.

* **\*Test management tool (в тому числі Exсel)**

Test management tools are used to structure automated tests and manual test processes, and to easily manage multiple environments. Quality assurance teams use these types of tools as a single application for managing test cases, environments, automated tests, defects and project tasks. Test management tools offer the prospect of streamlining the testing process and allow quick access to data analysis, collaborative tools and easy communication across multiple project teams. Tracking of bugs, defects and project tasks are done within one application, simplifying processes and saving teams time and money.

**Structuring the test process**

Test management tools give teams the ability to consolidate and structure the test process using one test management tool, instead of installing multiple applications that are designed to manage only one step of the process. Test management tools allow teams to manage test case environments, automated tests, defects and project tasks. Some applications include advanced dashboards and detailed tracking of key metrics, allowing for easy tracking of progress and bug management.

**Implementation**

A test management tool that includes everything needed to manage the test process can save testers the hassle of installing separate applications that are necessary for the testing process. They can be implemented with minimal programming ability, allowing for easy installation and monitoring of the test process across multiple project groups. Once installed, teams have instant access to a user interface and can immediately start running and recording test cases. These types of applications are designed to simplify the test management process with high levels of automation and tracking built in, yet don’t require advanced programming skills or knowledge to implement. They are useful for teams who manage a variety of test cases and for larger teams who need an all-inclusive application for project management.

**Using**

Once a project has kicked off, a test management tool tracks bug status, defects and projects tasks, and allows for collaboration across the team. When administering test cases, users can access a variety of dashboards to gain access to data instantly, making the test process efficient and accurate. The type of dashboard used is determined by the scope of the project and the information and data that needs to be extracted during the testing process. Data can be shared and accessed across multiple project teams, allowing for effective communication and collaboration throughout the testing process.

Розуміння технологій (загальне):

* Active directory

Active Directory — LDAP-сумісна реалізація інтелектуальної служби каталогів корпорації Microsoft для операційних систем родини Windows NT. Active Directory дозволяє адміністраторам використовувати групові політики (GPO) для забезпечення подібного налаштування користувацького робочого середовища, розгортати ПЗ на великій кількості комп'ютерів (через групові політики або за допомогою Microsoft Systems Management Server 2003 (або System Center Configuration Manager)), встановлювати оновлення ОС, прикладного та серверного ПЗ на всіх комп'ютерах в мережі (із використанням Windows Server Update Services (WSUS); Software Update Services (SUS) раніше). Active Directory зберігає дані і налаштування середовища в централізованій базі данних. Мережі Active Directory можуть бути різного розміру: від кількох сотень до кількох мільйонів об'єктів.



* DHCP server

DHCP (англ. Dynamic Host Configuration Protocol — протокол динамічної конфігурації вузла) — це мережний протокол, що дозволяє комп'ютерам автоматично одержувати IP-адресу й інші параметри, необхідні для роботи в мережі TCP/IP. Для цього комп'ютер звертається до спеціального серверу, під назвою сервер DHCP. Мережевий адміністратор може задати діапазон адрес, що розподіляють серед комп'ютерів. Це дозволяє уникнути ручного налаштування комп'ютерів мережі й зменшує кількість помилок. Протокол DHCP використовується в більшості великих мереж TCP/IP.

* HTML, XML

HTML (англ. HyperText Markup Language — Мова розмітки гіпертексту) — стандартна мова розмітки веб-сторінок в Інтернеті. Більшість веб-сторінок створюються за допомогою мови HTML (або XHTML). Документ HTML оброблюється браузером та відтворюється на екрані у звичному для людини вигляді.

Розши́рювана мо́ва розмі́тки (англ. Extensible Markup Language, скорочено XML) — запропонований консорціумом World Wide Web (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет.[1] Є спрощеною підмножиною мови розмітки SGML. XML документ складається із текстових знаків, і придатний до читання людиною.

* **\*SharePoint(на рівні користувача)**

Microsoft SharePoint Products and Technologies — це колекція програмних продуктів і компонентів від корпорації Microsoft, яка включає в себе:

-Набір веб-застосунків для організації спільної роботи

-Функціональність для створення веб-порталів

-Модуль пошуку інформації в документах і інформаційних системах

-Функціональність управління робочими процесами і систему керування вмістом масштабу підприємства

-Модуль створення форм для введення інформації

-Функціональність для бізнес-аналізу

SharePoint може бути використаний для створення сайтів, що надають користувачам можливість для спільної роботи. Створювані на платформі SharePoint сайти можуть бути використані в якості сховища інформації, знань і документів, а також використовуватися для виконання веб-застосунків, таких як вікі і блоги, що полегшують взаємодію. Користувачі можуть керувати і взаємодіяти з інформацією в списках і бібліотеках документів використовуючи елементи управління, що зваться веб-частини (SharePoint WebParts)

SharePoint представлен в виде двух основных продуктов — Microsoft SharePoint Foundation (ранее назывался WSS) и Microsoft SharePoint Server (ранее назывался MOSS). Помимо этого, предлагается инструментальное средство Microsoft Office SharePoint Designer (SPD).

Microsoft SharePoint Foundation — бесплатное приложение к Windows Server. Microsoft SharePoint Foundation предоставляет базовую инфраструктуру для совместной работы — редактирование, хранение документов, контроль версий и т. д. Также он включает в себя такую функциональность, как «маршруты» движения документов (платформа для документооборота), списки заданий, напоминания, онлайн-дискуссии.

Microsoft SharePoint Server 2010 — платный компонент для интеграции функциональности SharePoint в работу приложений MS Office. Он является надстройкой над Microsoft SharePoint Foundation и расширяет его возможности. Microsoft Project Server теперь является дополнением, которое устанавливается на Microsoft SharePoint Server 2010.

Microsoft Office SharePoint Server (MOSS) — платный компонент для интеграции функциональности SharePoint в работу приложений MS Office. Он являлся надстройкой над WSS и расширял его возможности. Включал в себя инструменты для бизнес аналитики — Excel Services, Business Data Catalog. MOSS позволял получить доступ к Microsoft Project Server и к формам Microsoft Office InfoPath через браузер, централизованно, в соответствии с концепцией многомодульного портала. Поддерживал специальные библиотеки, такие как PowerPoint Template Libraries. MOSS ранее был известен как SharePoint Server и SharePoint Portal Server.

Microsoft Office SharePoint Designer (SPD) — HTML-редактор в стиле WYSIWYG (ранее — FrontPage), оптимизированный для создания SharePoint-страниц и управления документами для WSS сайтов. SPD дает возможность доступа к функциональности своего рендер-движка через Microsoft Expression Web и через среду разработки Microsoft Visual Studio. Весной 2009 года стал бесплатным продуктом.

* Exchange

Microsoft Exchange Server — серверний програмний продукт для обміну повідомленнями і спільної роботи. Є частиною Windows Server System. Основні функції Microsoft Exchange:

-Обробка і пересилка поштових повідомлень

-Сумісний доступ до календарів і завдань

-Підтримка мобільних пристроїв і веб-доступ

облікових записів користувачів і поштових скриньок, списки контактів). Окремо від Active Directory зберігаються тільки самі поштові скриньки (у зв'язку з істотним розміром). Завдяки механізму реплікації Active Directory у разі використання кількох серверів Microsoft Exchange Server зберігається актуальність даних на всіх серверах. Так само «автоматично» підтримується ієрархічна система довірчих відносин між доменами в лісі.

**Особливості**

Головна особливість сервера — тісна інтеграція з Active Directory: велика частина призначених для користувача даних зберігається в Active Directory (зв'язок облікових записів користувачів і поштових скриньок, списки контактів). Окремо від Active Directory зберігаються тільки самі поштові скриньки (у зв'язку з істотним розміром). Завдяки механізму реплікації Active Directory у разі використання кількох серверів Microsoft Exchange Server зберігається актуальність даних на всіх серверах. Так само «автоматично» підтримується ієрархічна система довірчих відносин між доменами в лісі.

Для роботи з OMA/OWA використовуються можливості IIS. **Підтримувані протоколи і клієнти**

Microsoft Exchange Server може працювати разом з наступнити клієнтами:

Microsoft Outlook (зі складу Microsoft Office) — основний клієнт для роботи з сервером з робочих станцій.

Outlook Web Access (OWA) — веб-інтерфейс. В останній версії Exchange 2007 OWA підтримується основна функціональність Outlook за винятком можливості редагувати завдання з планувальника, спам-фільтра

Outlook mobile access — вкрай спрощений інтерфейс для доступу з мобільних пристроїв (інтерфейс споживає мінімальний трафік і оптимізований під екрани низької роздільної здатності)

Active Sync у складі Windows CE (працює через Active Sync на робочій станції або безпосередньо з exchange server через https з'єднання з OWA).

Довільним поштовим клієнтом по протоколах IMAP/POP3. Клієнт може бути несумісним з такими функціями Exchange Server, як доступ до календарю, публічні папки, тощо

**Резервне копіювання**

Microsoft Exchange Server при установці доповнює стандартний інструмент архівації Windows — NTBackup — підтримкою сховищ Exchange. Окрім цього, Microsoft також публікує список серверних застосунків для архівації виробництва компаній-партнерів, для архівації сховищ Exchange Server. Резервне копіювання сховищ в «пофайловому вигляді», тільки за умови відключення сховищ на час резервного копіювання — украй не рекомендується. Механізм тіньового копіювання підтримується, і можливість його застосування залежить від вибраного продукту для архівації.

* File Server

Файл-сервер — это выделенный сервер, предназначенный для выполнения файловых операций ввода-вывода и хранящий файлы любого типа. Как правило, обладает большим объемом дискового пространства, реализованном в форме RAID-массива для обеспечения бесперебойной работы и повышенной скорости записи и чтения данных.

Программная архитектура «файл-сервер»

Файл-серверные приложения — приложения, схожие по своей структуре с локальными приложениями и использующие сетевой ресурс для хранения данных в виде отдельных файлов. Функции сервера в таком случае обычно ограничиваются хранением данных (возможно также хранение исполняемых файлов), а обработка данных происходит исключительно на стороне клиента. Количество клиентов ограничено десятками ввиду невозможности одновременного доступа на запись к одному файлу. Однако клиентов может быть в разы больше, если они обращаются к файлам исключительно в режиме чтения.

Достоинства:

-низкая стоимость разработки;

-высокая скорость разработки;

-невысокая стоимость обновления и изменения ПО.

Недостатки:

рост числа клиентов резко увеличивает объём трафика и нагрузку на сети передачи данных;

высокие затраты на модернизацию и сопровождение сервисов бизнес-логики на каждой клиентской рабочей станции;

низкая надёжность системы.

* Загальні Поглиблені знання ОС: Windows, Linux, MAC
* Mobile platforms

[Debugging in Android](http://www.droidnova.com/debugging-in-android-using-eclipse,541.html)

Опціонально: знає мови скриптування; знає специфічні технології , що використовуються на проекті (HTML тощо)

Сцена́рный язы́к или скри́птовый язы́к (англ. scripting language, в русской литературе принято название язык сценариев) — высокоуровневый язык программирования для написания сценариев — кратких описаний выполняемых системой действий. Разница между программами и сценариями довольно размыта. Сценарий — это программа, имеющая дело с готовыми программными компонентами[1].

Согласно Джону Устерхауту, автору языка Tcl, высокоуровневые языки можно разделить на языки системного программирования (англ. system programming languages) и сценарные языки(англ. scripting languages). Последние он также назвал склеивающими языками(англ. glue languages) или языками системной интеграции(англ. system integration languages). Сценарии обычно интерпретируются, а не компилируются[2], хотя сценарные языки программирования один за другим обзаводятся JIT-компиляторами[3][4][5].

PL/SQL Developer

SQL (англ. Structured query language — мова структурованих запитів) — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних. Сам по собі SQL не є ні системою керування базами даних, ні окремим програмним продуктом. Не будучи мовою програмування в тому розумінні, як C або Pascal, SQL може формувати інтерактивні запити або, будучи вбудованою в прикладні програми, виступати в якості інструкцій для керування даними. Стандарт SQL, крім того, вміщує функції для визначення зміни, перевірки і захисту даних.

SQL — це діалогова мова програмування для здійснення запиту і внесення змін до бази даних, а також управління базами даних. Багато баз даних підтримує SQL з розширеннями до стандартної мови. Ядро SQL формує командна мова, яка дозволяє здійснювати пошук, вставку, обновлення, і вилучення даних, використовуючи систему управління і адміністративні функції. SQL також включає CLI (Call Level Interface) для доступу і управління базами даних дистанційно.

SELECT список\_вибірки

[ INTO нова\_табличка ]

FROM таблиця

[ WHERE умови\_пошуку ]

[ GROUP BY групувати\_по\_умові ]

[ HAVING умови\_пошуку ]

[ ORDER BY сортувати\_по\_умові [ ASC | DESC ] ]

* SQL Manager Studio

SQL Server Management Studio — утилита из Microsoft SQL Server 2005 и более поздних версий для конфигурирования, управления и администрирования всех компонентов Microsoft SQL Server. Утилита включает скриптовый редактор и графическую программу, которая работает с объектами и настройками сервера.[1]

Главным инструментом SQL Server Management Studio является Object Explorer, который позволяет пользователю просматривать, извлекать объекты сервера, а также полностью ими управлять.

Также есть SQL Server Management Studio Express для Express версии сервера, которая является бесплатной. Однако в ней нет поддержки ряда компонентов (Analysis Services, Integration Services, Notification Services, Reporting Services) и SQL Server 2005 Mobile Edition.

* Вміє писати прості SQL запити

***Локалізаційне:***

* Basic compile knowledge
* **\* Вміє працювати з інструментами Administrative Tools.**



**\* Процеси, сесії, івент менеджер і т.п.**

Проце́сс — выполнение пассивных инструкций компьютерной программы на процессоре ЭВМ. Стандарт ISO 9000:2000 Definitions определяет процесс как совокупность взаимосвязанных и взаимодействующих действий, преобразующих входящие данные в исходящие.

Компьютерная программа сама по себе это только пассивная совокупность инструкций, в то время как процесс — это непосредственное выполнение этих инструкций.

Часто процессом называют выполняющуюся программу и все её элементы: адресное пространство, глобальные переменные, регистры, стек, открытые файлы и т. д.

Поддержка сессий в PHP - это способ сохранения определённых данных в течение последовательно выполняемых доступов. Это даёт возможность создать более специализированные приложения и увеличить привлекательность вашего web-сайта.

**FAT NTFS**

**Что такое файловая система?**

ФС управляет хранением и доступом к файлам на компьютере - и этим все сказано. Одним из базовых понятий ФС является кластер - минимальный размер данных на диске. Например, если ваш файл "весит" всего 1 байт, а размер кластера на винчестере - 8 Кб, то в итоге на жестком диске размер файла будет тоже 8 Кб (один кластер). Если же файл реально занимает 8,1 Кб, на диске он будет "весить" все 16 Кб (два кластера). Теперь попробуйте прикинуть, сколько сотен файлов хранится на винчестере, и вам покажется, что все прошедшие годы вы просто недооценивали роль кластера в вашей жизни.

Помимо размера кластера (он может меняться в зависимости от ФС), немаловажно, каким образом файловая система заполняет свободную область на диске, плохой алгоритм ведет к фрагментации данных (когда части одного файла на диске находятся далеко друг от друга). Скажу, забегая вперед, что в NTFS алгоритм заполнения не очень хорош. Но обо всем по порядку...

**FAT32**

Теоретически размер логического диска FAT32 ограничен 8 Тб. На практике же встроенные в Windows 2000/XP средства администрирования дисков не позволят создать раздел размером более 32 Гб. Но даже этого для сегодняшних ПК хватает с лихвой.

Имена файлов в FAT32 могут содержать до 255 символов. Максимально возможный размер одного файла составляет 4 Гб.

Пожалуй, одной из важнейших характеристик ФС является стабильность, то есть устойчивость к ошибкам. У FAT32 с этим дела обстоят, прямо говоря, не важно. Наиболее распространенная ошибка, которая знакома всем обладателям Windows 98, - неправильно записанные данные о количестве свободного места. Она возникает по причине того, что данные о количестве свободного места не вычисляются, как это происходило в FAT16, а просто записываются в загрузочную область. И когда возникает сбой в процессе копирования (удаления, перемещения) файла, ОС не успевает записать обновленные данные о свободном месте на диске, хотя на самом деле оно изменилось. В результате возникает ошибка, исправить которую можно лишь полной проверкой винчестера специальной программой.

К тому же FAT32 довольно сильно подвержен фрагментации (особенно при заполнении диска более чем на 80%) - это существенно замедляет работу. В особо запущенных случаях фрагментация может привести даже к "падению" всей ФС.

**NTFS**

Ограничения на размер жесткого диска, выставляемые NTFS, сегодня недостижимы - 2 000 000 Гб, так что, можно сказать, ограничений попросту нет. Первые 12% диска под управлением NTFS отводятся под основную таблицу файлов MFT (Master File Table). Она представляет собой каталог всех имеющихся файлов, причем файлы небольшого размера (100 байт) хранятся прямо в MFT - это заметно ускоряет доступ к ним. Для работы ФС очень важны первые 16 элементов MFT (указатели на системные файлы) и поэтому на диске хранится копия этих записей. В результате "снести" NTFS довольно непросто: система в состоянии обойти серьезные неисправности поверхности диска и пережить даже повреждение MFT (аналогичная ситуация для FAT закончилась бы фатально).

Каталог в NTFS представляет собой специфический файл, хранящий ссылки на другие файлы и каталоги. Его внутренняя структура подобна бинарному дереву, что позволяет в десятки раз сократить время поиска нужного файла (так называемый метод деления пополам). Чем больше файлов в каталоге, тем больше преимущество перед FAT32 при поиске.

Отказоустойчивость NTFS связана с тем, что работа с данными производится на основе транзакций - действий, совершаемых целиком и корректно или не совершаемых вообще.

Представим себе, что осуществляется запись данных на диск, и вдруг на середине процесса выясняется, что в том месте, куда мы решили записать очередную порцию данных - физическое повреждение поверхности. В этом случае транзакция записи откатывается целиком (отменить изменения можно благодаря журналу транзакций). Место помечается как сбойное, а данные записываются в другое место - начинается новая транзакция.

Разграничение прав доступа к файлам в NTFS далеко не всегда защищает от несанкционированного доступа к секретным данным. Злоумышленник может подключить ваш жесткий диск к другому компьютеру и без труда прочесть нужные файлы, поскольку ограничение прав доступа не распространяется за пределы вашей ОС. Поэтому в NTFS была введена дополнительная мера - шифрование данных на уровне файловой системы, что позволяет скрывать информацию более успешно.

Сжатие данных в NTFS тоже производится на уровне ФС, что позволяет работать с ними относительно быстро, распаковывая информацию "налету". Механизм сжатия чрезвычайно гибок - можно упаковывать даже половину файла, другую часть оставляя несжатой.

Имя файла может содержать любые символы, включая полный набор национальных алфавитов, так как данные представлены в Unicode (65535 разных символов).

**Localization**

Інтернаціоналізація (англ. internationalization) — процес адаптації продукту, такого як програмне або апаратне забезпечення, до мовних і культурних особливостей регіону (регіонів), відмінного від того, в якому розроблявся продукт. В англійській мові для слова «internationalization» прийнято скорочення «i18n».

Є важлива відмінність між інтернаціоналізацією та локалізацією. Інтернаціоналізація — це адаптація продукту для потенційного використання практично в будь-якому місці, тоді як локалізація — це додавання спеціальних функцій для використання в деякому визначеному регіоні.

Локализа́ция програ́ммного обеспече́ния — процесс адаптации программного обеспечения к культуре какой-либо страны. Как частность — перевод пользовательского интерфейса, документации и сопутствующих файлов программы с одного языка на другой.

Для локализации в английском языке иногда применяют сокращение «L10n». Где буквы «L» и «n» — начало и окончание слова Localization, а цифра 10 — количество букв между ними.