# Операційні системи

## Поняття операційної системи, її призначення

Операцио́нная систе́ма, сокр. ОС (англ. operating system, OS) — комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

## Види операційних систем

Это список известных операционных систем. Операционные системы могут быть классифицированы по базовой технологии (UNIX-подобные, пост-UNIX/потомки UΝΙΧ), типу лицензии (проприетарная или открытая), развивается ли в настоящее время (устаревшие или современные), по назначению (универсальные, ОС встроенных систем, ОС PDA, ОС реального времени, для рабочих станций или серверов), а также по множеству других признаков.

## Структура операційної системи

ОС состоит из:

– загрузчика

– ядра,

– базовой системы ввода-вывода,

– оболочки,

– сервисных программ.

Загрузчик операционной системы

Загрузчик операционной системы — программа, обеспечивающая запуск операционной системы из начального состояния (после включения питания, сброса), инициализацию основных её компонентов.

Ядро операционной системы

Ядро операционной системы — часть ОС, выполняющаяся при максимальном уровне привилегий. Как правило, в ядро помещаются процедуры, выполняющие манипуляции с основными ресурсами системы и уровнями привилегий процессов, а также критичные процедуры, выполнение которых с пользовательскими привилегиями невыгодно. Ядро считается необходимым компонентом ОС, по крайней мере в системах с распределением привилегий, впрочем заявлено создание и безъядерных ОС[1].

Базовая система ввода-вывода

Базовая система ввода-вывода (БСВВ, BIOS) — набор программных средств, обеспечивающих взаимодействие ОС и приложений с аппаратными средствами. Обычно БСВВ представляет набор компонент — драйверов. Также в БСВВ входит уровень аппаратных абстракций, минимальный набор аппаратно-зависимых процедур ввода-вывода, необходимый для запуска и функционирования ОС. Нередко функции БСВВ включаются в ядро.

ОболочкаОболочка операционной системы — часть ОС, задачей которой является взаимодействия ОС с пользователем, заключающееся в предоставлении ему управления и контроля за работой вычислительной системе. Это может быть командный интерпретатор — обеспечивающий управление системой посредством ввода текстовых команд (с клавиатуры, через порт или сеть), графическая оболочка — обеспечивающая работу при помощи координатных устройств ввода (принцип «укажи и нажми»). Операционные системы, не предназначенные для интерактивной работы часто его не имеют.

Служебные программы

Сервисное программное обеспечение (утилиты) – пакет программ, включённый в состав ОС, реализующий действия по управлению и обслуживанию ОС.

## Системні виклики

Систе́мный вы́зов (англ. system call) в программировании и вычислительной технике — обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции.

Современные операционные системы (ОС) предусматривают разделение времени между выполняющимися вычислительными процессами (многозадачность) и разделение полномочий, препятствующее обращению исполняемых программ к данным других программ и оборудованию. Ядро ОС исполняется в привилегированном режиме работы процессора. Для выполнения межпроцессной операции или операции, требующей доступа к оборудованию, программа обращается к ядру, которое, в зависимости от полномочий вызывающего процесса, исполняет либо отказывает в исполнении такого вызова.

С точки зрения программиста, системный вызов обычно выглядит как вызов подпрограммы или функции из системной библиотеки. Однако системный вызов, как частный случай вызова такой функции или подпрограммы, следует отличать от более общего обращения к системной библиотеке, поскольку последнее может и не требовать выполнения привилегированных операций.

## Процеси, потоки та методи їх синхронізації. Критичні області, семафори, м’ютекси, події.

* Процесс – **исполняемая** программа, с которой связаны:
  + Системные информационные ресурсы;
  + Область основной памяти (адресное пространство);
  + Файлы, устройства ввода-вывода.
* Задачи подсистемы управления процессами:
  + Создание и уничтожение процессов;
  + Планирование выполнения процессов;
  + Выделение ресурсов процессам;
  + Защита ресурсов процесса;
  + Синхронизация и взаимодействие процессов.
* Потоки – последовательности команд, совместно использующих данные
* Процесс – совокупность
  + потоков (не менее одного) и
  + защищенного адресного пространства

Средства обмена информацией:

Сигнальные (сигналы UNIX)

минимальный объем передаваемой информации, минимальное влияние. Cпособ информирования процесса со стороны ядра о происшествии некоторого события:

Канальные (FIFO, pipes, sockets)

объем передаваемой информации ограничен пропускной способностью линий связи.

Разделяемая память (shmem)

наиболее быстрый способ взаимодействия процессов в одной вычислительной системе.

main(){

int var = 5;

if(fork()) wait(&var);

else exit(var);

var++;

printf("%d\n",var);

return var;

}

В среде, позволяющей исполнять несколько процессов (потоков) одновременно, очень важно синхронизировать их деятельность. Для этого операционные системы, предлагают несколько синхронизирующих объектов: критические разделы (critical sections), объекты mutex (сокращение от mutual exclusion), семафоры и события.

Критические разделы (только WIN32)

*Критический раздел* (critical section) – объект, к которому должен обратиться поток перед получением эксклюзивного доступа к каким-либо общим данным. Среди синхронизирующих объектов критические разделы наиболее просты, но применимы для синхронизации потоков лишь в пределах одного процесса. Они дают возможность сделать так, чтобы единовременно только один поток получил доступ к определенному региону данных.

При необходимости синхронизировать приложения со специфическими событиями, возникающими в системе, или с операциями, выполняемыми в других процессах, критические разделы непригодны. Например, при создании дочернего процесса, возможно, придется сделать так, чтобы родительский процесс ожидал его завершения и только потом продолжал свою работу.

Каждый объект синхронизации может находиться в одном из двух состояний: *свободном* (signaled) или *занятом* (nonsignaled). Потоки могут остановиться и ждать, пока какой-либо объект не освободится. Если поток родительского процесса должен ждать завершения дочернего, его можно отправить "в спячку" до освобождения объекта ядра, идентифицирующего дочерний процесс. Объекты "процесс" получают статус свободных в момент завершения соответствующих процессов. Это относится и к объектам "поток". Когда поток создан и начинает исполнение своего кода, сопоставленный с ним объект ядра "поток" получает статус занятого. При завершении потока соответствующий объект ядра освобождается.

Объекты Mutex

Эти объекты аналогичны критическим разделам, но с их помощью можно синхронизировать доступ к данным со стороны не только нескольких потоков одного процесса, но и нескольких процессов.

Семафоры

Объекты ядра "семафор" используются для учета ресурсов. При запросе у семафора ресурса операционная система проверяет, свободен ли данный ресурс, и — если свободен — уменьшает счетчик доступных ресурсов, не давая вмешиваться другому потоку. Только после этого система разрешает другому потоку запрашивать какой-либо ресурс.

Для мониторинга занятости ресурсов необходимо создать семафор со счетчиком. При этом нужно учитывать, что семафор считается свободным, если его счетчик ресурсов больше нуля, и занятым, если счетчик равен нулю. При каждом вызове из потока специальной функции с передачей ей описателя семафора система проверяет: больше ли нуля счетчик ресурсов у данного семафора. Если да, уменьшает счетчик на единицу и "будит" поток. Если при вызове функции счетчик семафора оказался обнулен, система оставляет поток неактивным до того, как другой поток освободит семафор (т. е. увеличит его счетчик ресурсов).

Поскольку на счетчик ресурсов семафора могут влиять несколько потоков (или процессов), семафоры — в отличие от критических разделов и объектов mutex — не передаются во владение какому-либо потоку. А значит, один поток может ждать объект "семафор" (уменьшив его счетчик ресурсов), а другой поток освободить семафор (и тем самым увеличить его счетчик ресурсов).

События

События — самая примитивная разновидность синхронизирующих объектов, отличающаяся от семафоров и объектов mutex. Последние обычно применяются для контроля за доступом к данным, а события просто оповещают об окончании какой-либо операции. Существуют два разных типа объектов "событие": со *сбросом вручную* (manual-reset events) и *с автоматическим сбросом* (auto-reset events). Первые используются для оповещения об окончании операции сразу нескольких потоков, вторые — для оповещения единственного потока.

К событиям обычно прибегают в том случае, когда один поток выполняет какую-либо инициализацию, а затем сигнализирует другому потоку, что тот может работать дальше. Инициализирующий поток переводит объект "событие" в занятое (non-signaled) состояние и приступает к своим операциям. По окончании инициализации поток возвращает событие в свободное (signaled) состояние. В то же время рабочий поток приостанавливает свое исполнение и ждет перехода события в свободное состояние. Как только инициализирующий поток просигнализирует событие (т.е. освободит его), рабочий поток "проснется" и продолжит работу.

Например, в процессе исполняются два потока. Первый считывает данные из файла в буферную память и оповещает второй поток, что можно заняться обработкой данных. Закончив обработку, второй поток сигнализирует первому, чтобы тот загрузил новый блок данных из файла и т.д.

## Класичні проблеми синхронізації та їх вирішення

Проблема производителя и потребителя (проблема ограниченного буфера)

Два процесса совместно используют буфер ограниченного размера. Один из них, производитель, помещает данные в этот буфер, а другой, потребитель, считывает их оттуда. Задача может быть обобщена на случай ***т*** производителей и ***п***потребителей.

Проблема обедающих философов

Сформулирована в 1965г. Дейкстрой.

Условия задачи:

Имеется N философов, сидящих за круглым столом. Перед каждым из них находится тарелка со спагетти. Спагетти настолько скользкие, что каждому философу необходимо две вилки , чтобы с ними управиться. Между каждыми двумя тарелками лежит одна вилка.

Жизнь философа состоит из чередующихся периодов поглощения пищи и размышлений (остальные процессы жизнедеятельности в данном случае несущественны). Когда философ голоден, он пытается получить две вилки, левую и правую, в любом порядке. Если ему удалось получить две вилки, он некоторое время ест, затем кладет вилки и продолжает размышления.

Данная проблема полезна для моделирования процессов, соревнующихся за монопольный доступ к ограниченному количеству ресурсов (например, к устройствам ввода-вывода).

Проблема читателей и писателей

Проблема моделирует доступ к базе данных.

Условие задачи: необходимо обеспечить монопольный доступ к базе данных процессу, выполняющему запись данных, и множественный доступ процессам, выполняющим чтение данных. Задача может быть сформулирована с предоставлением приоритета как процессу-писателю, так и процессам-читателям.

Проблема спящего брадобрея

Условие задачи: имеется один процесс-«брадобрей», его кресло и N стульев для процессов-«клиентов». Если «клиентов нет», «брадобрей» спит в своем кресле. Если в парикмахерскую приходит «клиент», он должен разбудить «брадобрея». Если клиент приходит, и видит, что брадобрей занят, он либо занимает один из стульев (если есть место), либо уходит (если места нет). Необходимо запрограммировать брадобрея и клиентов так, чтобы избежать состояния состязания.

## Планування

Задача планирования возникает при наличии нескольких потребителей ограниченного набора ресурсов. Выбор алгоритма определяется классом решаемых задач и поставленными целями.

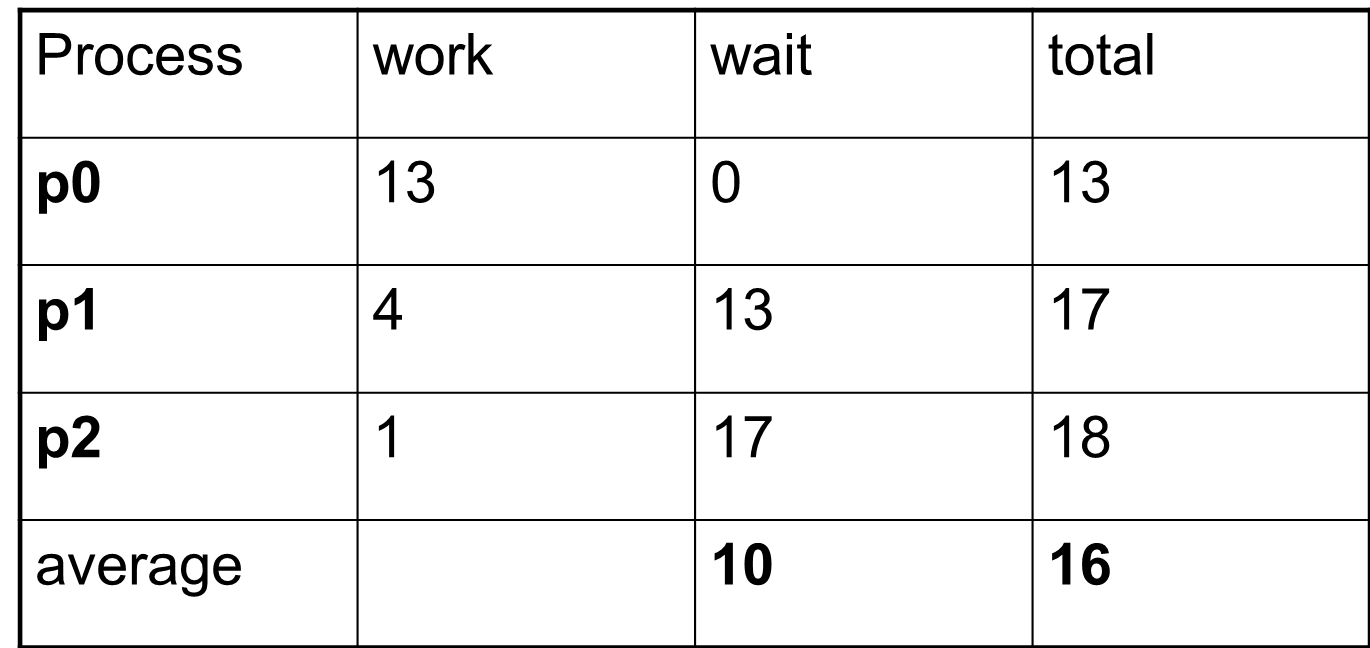
Параметры планирования

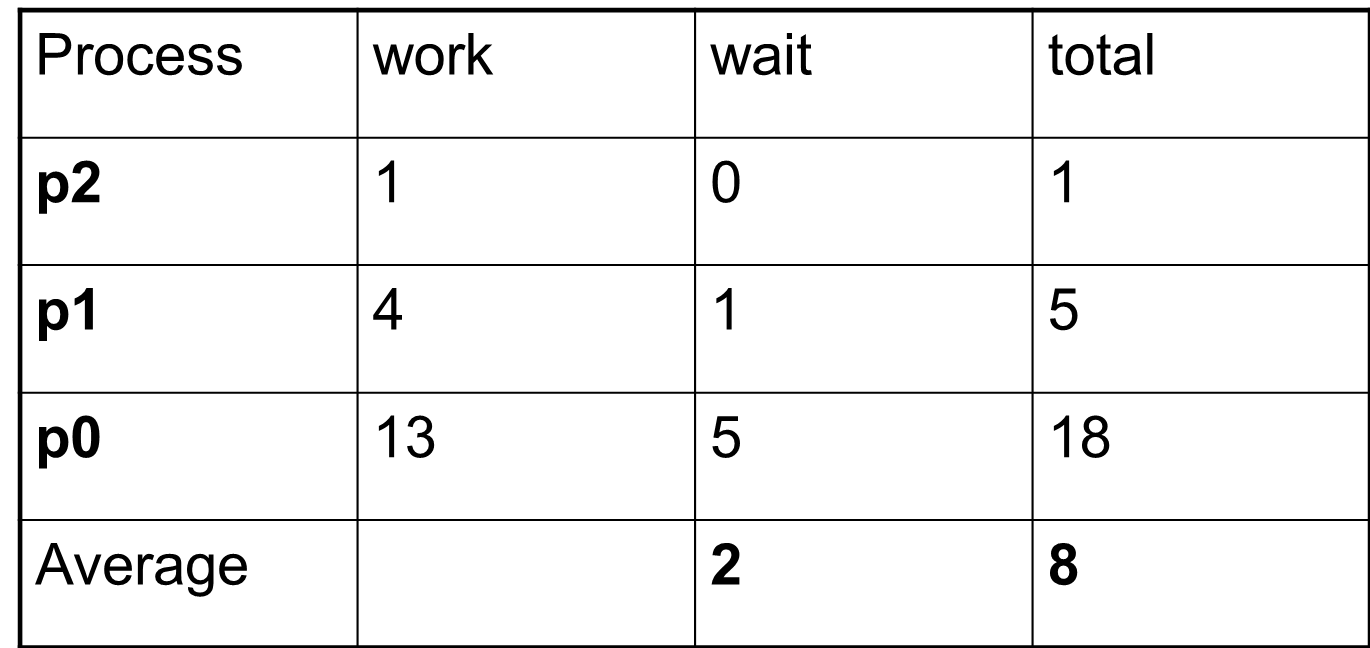
* Динамические параметры системы – количество свободных ресурсов на данный момент времени.
* Статические параметры системы – предельные значения ресурсов:
  + объем оперативной памяти
  + количество подключенных устройств ввода-вывода
  + объем своп-раздела

## Алгоритми планування. Політика та механізм планування

### First-Come, First-Served (FCFS)

* Невытесняющее планирование
* Процессы, находящиеся в состоянии готовности, помещаются в очередь
* Простота реализации
* Большое среднее время отклика в интерактивных процессах (неприменим для систем разделения времени)



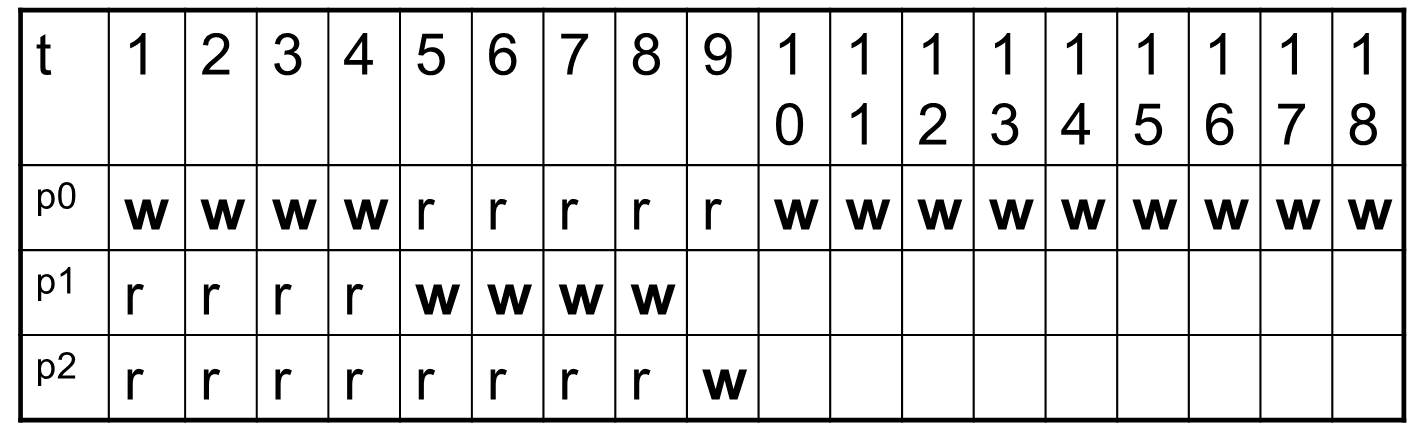


### Round Robin (RR)

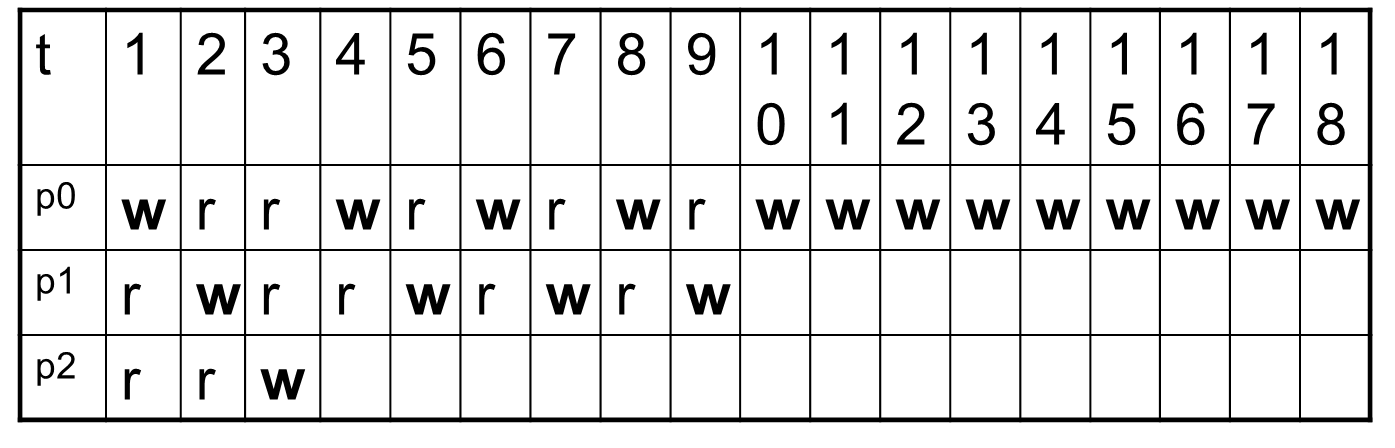
* Вытесняющее планирование
* Циклический список процессов, находящихся в состоянии готовности
* Каждому процессу выделяется определенный квант времени
* Производительность зависит от величины кванта времени

1)

* Длительность кванта = 4;
* wait = {5,4,8}, wait\_a = **5,7**;
* work = {18,8,9}, work\_a = **11,7**;



* 2)
* Длительность кванта = 1;
* wait = {5,5,2}, wait\_a = **4**;
* work = {18,9,3}, work\_a = **10**;

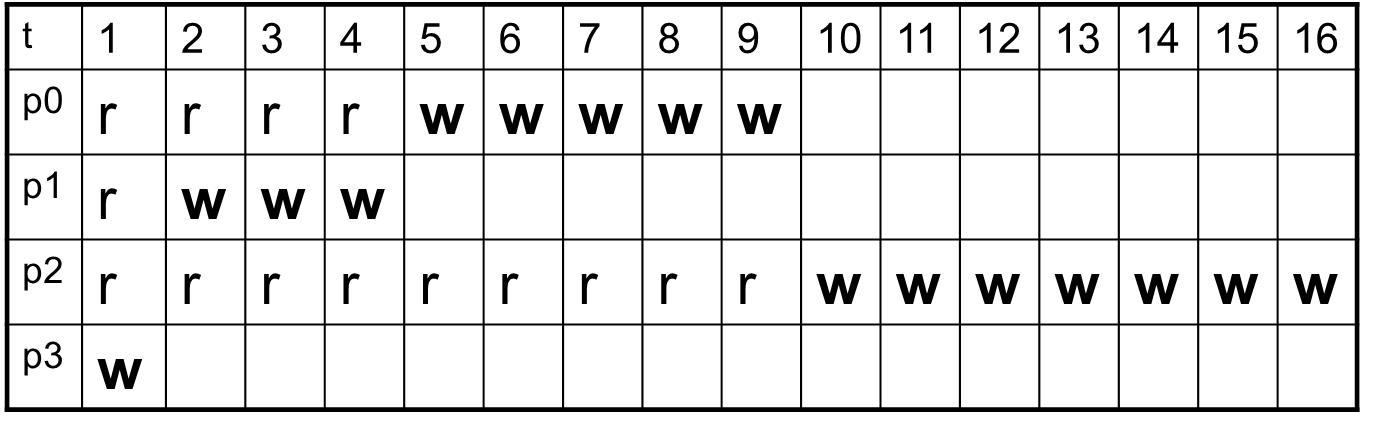


### Shortest Job(Task) First (SJF)

* Выбирает из очереди процесс с минимальным CPU burst
* Квантование не применяется
* Может быть как невытесняющим, так и вытесняющим (учитывается появление новых процессов)

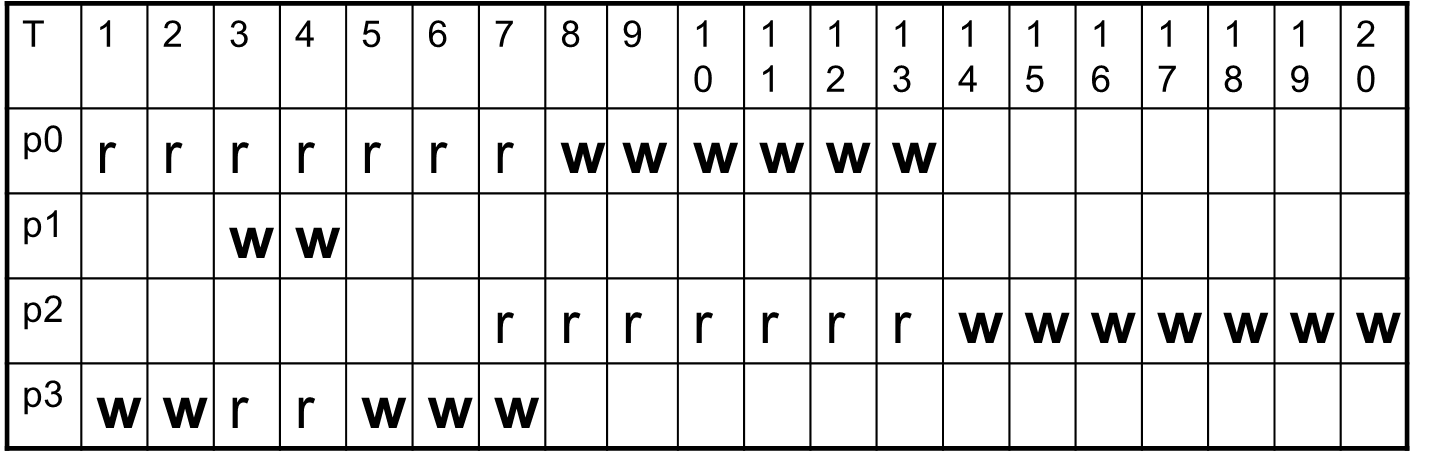
Невытесняющий SJF

* CPU burst = {5,3,7,1}
* wait = {4,1,9,0}, wait\_a = **3,5**;
* work = {9,16,4,1}; work\_a = **15**;



Вытесняющий SJF

* CPU burst = {6,2,7,5};
* T\_start = {0,2,6,0};
* wait = {7,0,7,2}, wait\_a = **4**;
* work = {13,2,14,7}; work\_a = **9**;



* Оценка CPU burst:
  + Задается пользователем (пакетные системы);
  + Вычисление с учетом предыстории:

sr3-f1

### Гарантированное планирование

* Очередной квант времени предоставляется процессу с минимальным значением коэффициента справедливости
* Невозможность прогнозирования поведения пользователей
* N – число пользователей, T – время нахождения пользователя в системе

sr3-f7

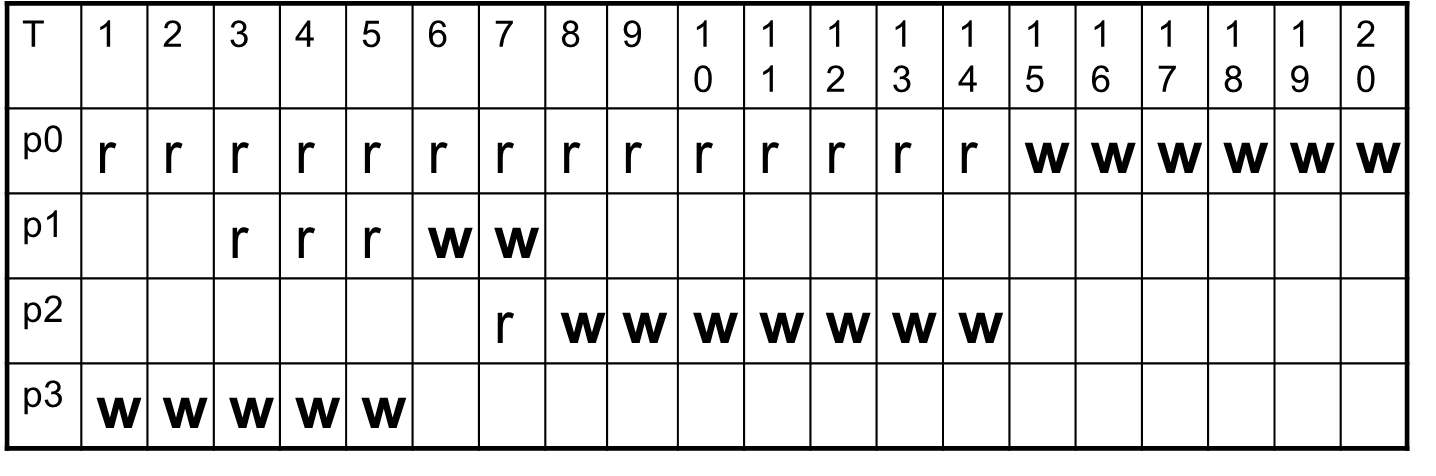
### Приоритетное планирование

Гарантированное планирование и SJF – частные случаи приоритетного планирования

Вытесняющее и невытесняющее планирование.

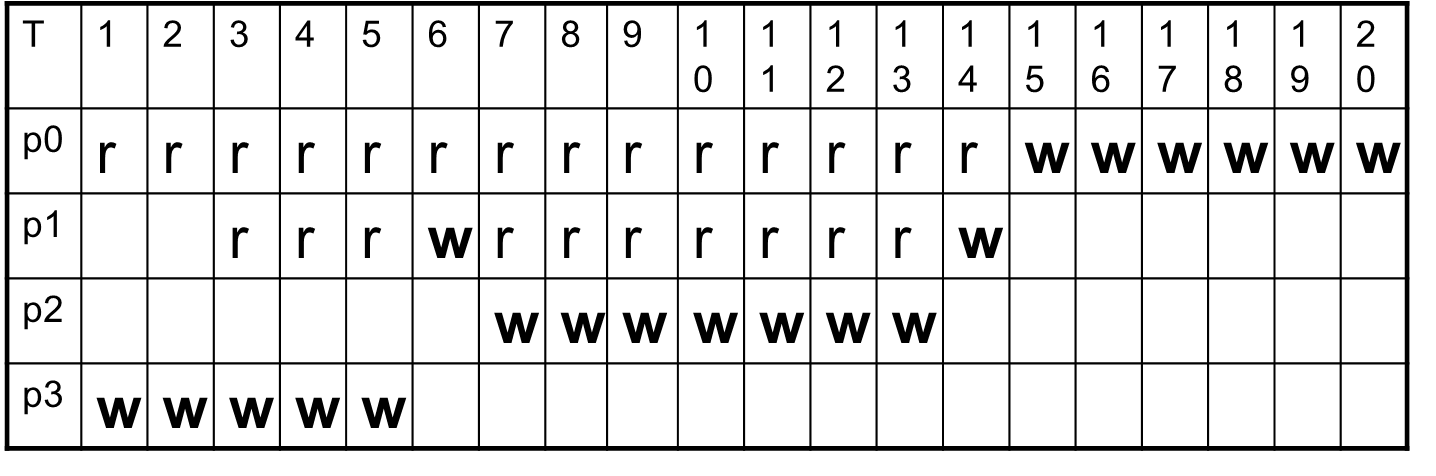
Приоритетное планирование невытесняющее

* CPU burst = {6,2,7,5};
* T\_start = {0,2,6,0};
* Priority = {4,3,2,1};



Приоритетное планирование вытесняющее

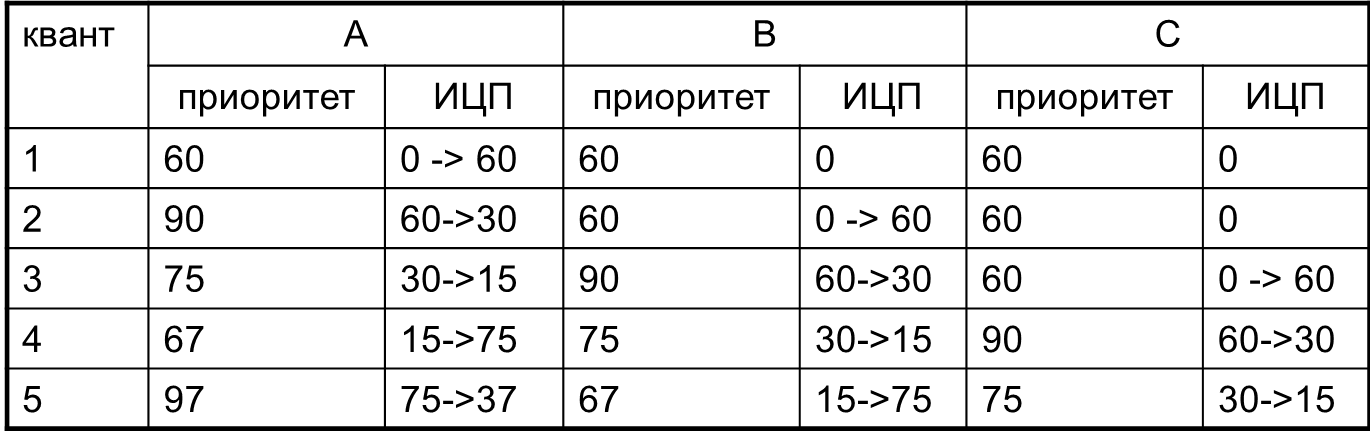
* CPU burst = {6,2,7,5};
* T\_start = {0,2,6,0};
* Priority = {4,3,2,1};



### Диспетчеризация

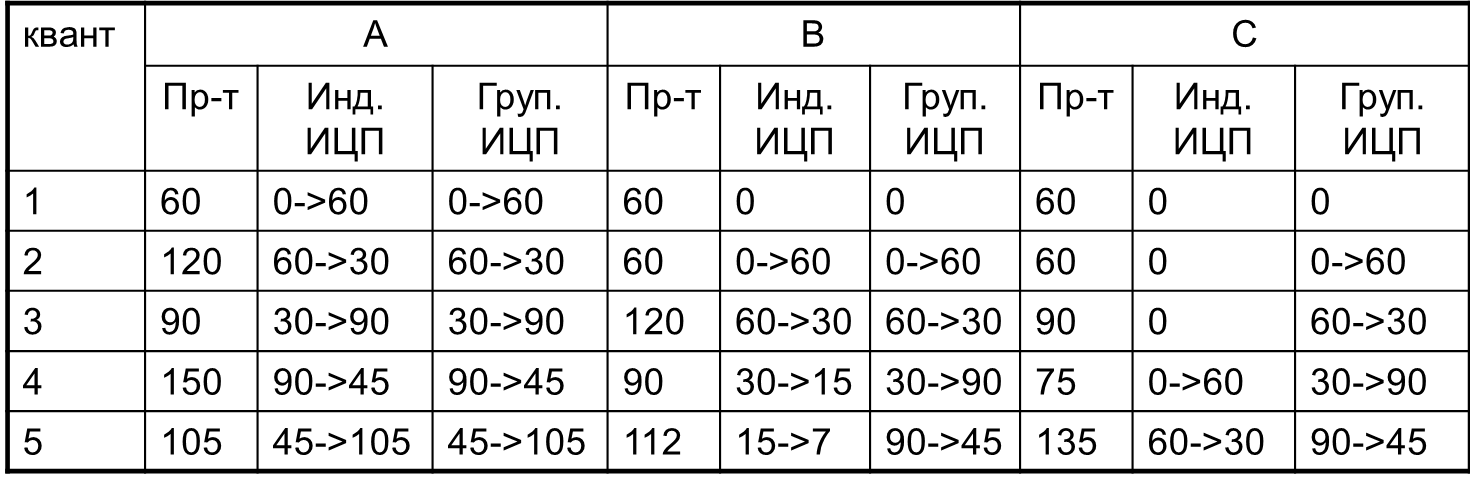
1)

* Три процесса (A, B, C);
* base = 60 - базовый приоритет;
* ИЦП = ИЦП/2;
* Приоритет = ИЦП/2 + base;
* Наращивание ИЦП – 60 раз в сек.



2)

* Три процесса (A, B, C);
* base = 60 - базовый приоритет;
* ИЦП = ИЦП/2; ИЦПГ = ИЦПГ/2;
* Приоритет = ИЦП/2 + ИЦПГ/2 + base;
* Наращивание ИЦП – 60 раз в сек.



## Системи керування пам’яттю

Память – ресурс вычислительной системы, предназначенный для хранения программного кода и даннях. Модуль управления памятью или менеджер памяти – часть операционной системы, отвечающая за управление памятью.

## Підкачка та віртуальна пам’ять

Виртуальная память

* **схема адресации памяти**, при которой память представляется программному обеспечению **непрерывной** и однородной,
* в то время как в реальности для фактического хранения данных используются **отдельные** (разрывные) **области** **различных** **видов** памяти, включая кратковременную (оперативную) и долговременную (жесткие диски, твердотельные накопители)
* **упрощает** адресацию памяти в программном обеспечении
* позволяет **рационально** управлять основной памятью компьютера (хранить в ней только активно используемые области памяти)
* позволяет **изолировать** процессы друг от друга

## Алгоритми заміщення сторінок. Моделювання алгоритмів заміщення сторінок.

Алгоритмы замещения страниц^

* Бит модификации
* Бит обращения
* Локальные алгоритмы - распределяют определенное количество страниц для каждого процесса
* Глобальные алгоритмы - распределяют все страницы между всеми процессами

### FIFO

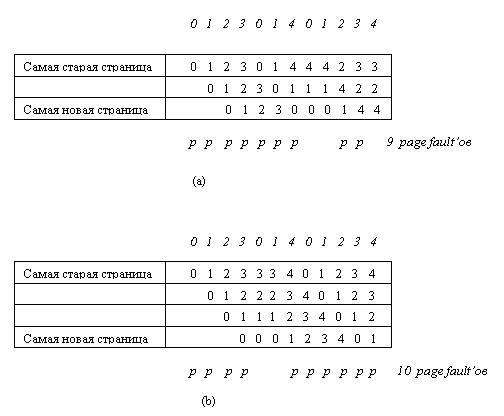
* Замещается самая «старая» страница

Операционная система поддерживает список всех страниц, находящихся в данный момент в памяти, в котором первая страница является старейшей, а страницы в хвосте списка попали в него совсем недавно. Когда происходит страничное прерывание, выгружается из памяти страница в голове списка, а новая страница добавляется в его конец.

Этот алгоритм не требует аппаратной поддержки, однако он не учитывает информацию об использовании страницы.

* Аномалия FIFO – c увеличением числа фреймов растет число страничных прерываний

(“Самая старая страница” <-> “Самая новая страница” -- reverse)



### NRU (Not Recently Used)

Использует R(eferenced) и M(odified) биты для классификации страниц по 4-м классам:

R=0, M=0 – не было обращений и изменений

R=0, M=1 – не было обращений, страница изменена

R=1, M=0 – было обращение, страница не изменена

R=1, M=1 – было обращение и изменение

The not recently used (NRU) page replacement algorithm is an algorithm that favours keeping pages in memory that have been recently used. This algorithm works on the following principle: when a page is referenced, a referenced bit is set for that page, marking it as referenced. Similarly, when a page is modified (written to), a modified bit is set. The setting of the bits is usually done by the hardware, although it is possible to do so on the software level as well.

At a certain fixed time interval, a timer interrupt triggers and clears the referenced bit of all the pages, so only pages referenced within the current timer interval are marked with a referenced bit. When a page needs to be replaced, the operating system divides the pages into four classes:

3. referenced, modified

2. referenced, not modified

1. not referenced, modified

0. not referenced, not modified

Although it does not seem possible for a page to be not referenced yet modified, this happens when a class 3 page has its referenced bit cleared by the timer interrupt. The NRU algorithm picks a random page from the lowest category for removal. So out of the above four pages, the NRU algorithm will replace the not referenced, not modified. Note that this algorithm implies that a modified but not referenced (within last timer interval) page is less important than a not modified page that is intensely referenced.

### LRU (The Least Recently Used)

Замещает страницу, которая не использовалась в течение долгого времени

Для реализации необходим список всех страниц в памяти, отсортированный по частоте обращений и обновляемый при каждом обращении

Требует аппаратной реализации

### NFU (Not Frequently Used)

The least recently used (LRU) page replacement algorithm, though similar in name to NRU, differs in the fact that LRU keeps track of page usage over a short period of time, while NRU just looks at the usage in the last clock interval. LRU works on the idea that pages that have been most heavily used in the past few instructions are most likely to be used heavily in the next few instructions too.

### Second Chance

Данный алгоритм является версией алгоритма FIFO, который позволяет избежать проблемы вытеснения из памяти часто используемых страниц. У самой старейшей страницы изучается бит *R.* Если он равен 0, страница находится в памяти долго и не используется, следовательно, она может быть заменена новой. Если же бит *R* равен 1, то ему присваивается значение 0, страница переносится в конец списка, а время ее загрузки обновляется, то есть считается, что страница только что попала в память. Затем процедура продолжается.

Работа данного алгоритма показана на рис. 3.1, *а.* Здесь изображены страницы от Адо H, хранящиеся в связанном списке и отсортированные по времени их поступления в память. Числа над страницами обозначают их время загрузки в память.

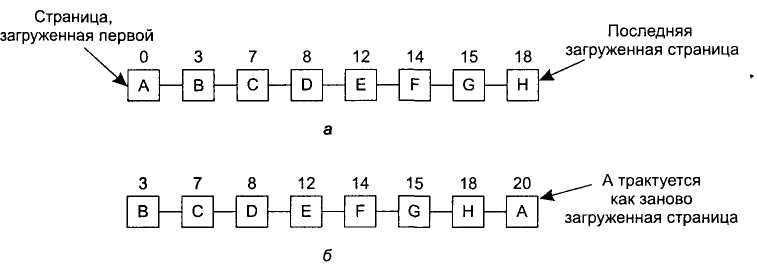


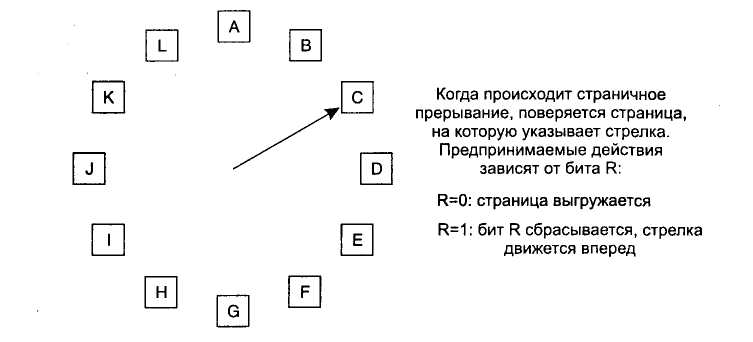
Рис. 3.1. Действие алгоритма «вторая попытка»: страницы, отсортированные в порядке очереди (FIFO) (а); список страниц, если страничное прерывание произошло во время 20, а страница *А* имеет бит R, равный 0 (б)

Алгоритм «вторая попытка» ищет в списке самую старую страницу, к которой не было обращений в предыдущем временном интервале. Если же происходили ссылки на все страницы, то «вторая попытка» превращается в обычный алгоритм FIFO.

### Часы

Данный алгоритм отличается от алгоритма «вторая попытка» только своей реализацией. Алгоритм «вторая попытка» является корректным, но не эффективным, поскольку постоянно передвигает страницы по списку. Поэтому лучше хранить все страничные блоки в кольцевом списке, как показано рис. 3.2. Стрелка указывает на старейшую страницу.

Когда происходит страничное прерывание, проверяется та страница, на которую направлена стрелка. Если ее бит R равен 0, страница выгружается, на ее место в круг встает новая страница, а стрелка сдвигается вперед на одну позицию. Если бит R равен 1, то он сбрасывается, стрелка перемещается к следующей странице. Этот процесс повторяется до тех пор, пока не находится та страница, у которой бит R = 0.



### Рабочий набор

Идея алгоритма рабочих наборов – ОС в каждый момент времени должна обеспечивать наличие в основной памяти текущих рабочих наборов всех процессов, которым разрешена конкуренция за доступ к процесору.

«Сборщик страниц» – выполняет откачку страниц, не входящих в рабочие наборы процессов. Начинает работать при достижении некоторого минимального порога числа свободных страниц. Аналог в Windows -- менеджер балансного набора (Working set manager)

Свопинг – выполняет выгрузку всех страниц процесса в ситуации, если свободных страниц нет

## Принципи роботи апаратури введення-виводу

* Устройство ввода-вывода
  + Механическая часть (в самом устройстве)
  + Электронная часть (контроллер, адаптер)
* Низкоуровневый интерфейс между устройством и контроллером (битовый поток)
* Контроллер формирует из потока битов блоки (с проверкой контрольной суммы) и генерирует команды устройству
* Контроллер содержит управляющие регистры и буферы данных

## Принципи побудови програмного забезпечення введення-виводу

Принципы организации ввода-вывода

* Программируемый ввод-вывод Процессор посылает необходимые команды контроллеру ввода-вывода. Процесс, запросивший ввод-вывод, находится в состоянии ожидания завершения операции ввода-вывода
* Управляемый прерываниями ввод-вывод. Для реализации необходима дополнительная линия – линия (шина) прерываний. По завершении выполнения операции устройство выставляет на линию прерываний сигнал. Для реализации необходима дополнительная линия – линия (шина) прерываний. По завершении выполнения операции устройство выставляет на линию прерываний сигнал.
* Ввод-вывод с использованием DMA (Direct Memory Access). Режим обмена данными без участия центрального процесора. DMA-контроллер получает доступ к системной шине независимо от ЦП. Содержит регистры, доступные ЦП для чтения и записи. Регистры задают порт, направление переноса данных (чтение/запись), единицу переноса (побайтно/пословно), число байтов, которое следует перенести. ЦП программирует контроллер DMA, устанавливая его регистры. По окончании записи устройство посылает сигнал подтверждения контроллеру DMA.

Программные уровни ввода-вывода:

## Переривання. Обробники переривань

При возникновении прерывания инициируется его обработка, включающая следующие шаги:

* Сохранение всех регистров (включая PSW), не сохраненных аппаратно
* Установка контекста для процедуры обработки прерываний
* Установка указателя стека для процедуры обработки прерываний
* Выдача подтверждения контроллеру прерываний. Если централизованного контроллера прерываний нет, разрешение прерываний
* Копирование содержимого регистров в таблицу процессов
* Запуск процедуры обработки прерываний, которая извлечет информацию из регистров контроллера устройства, инициировавшего прерывание
* Выбор процесса, которому передается управление
* Установка контекста для выбранного процесса
* Загрузка регистров нового процесса, включая его PSW
* Выполнение нового процесса

## Драйвери пристроїв

Программа для управления устройствами определенного вида (класса), принимающая стандартные запросы от ОС и преобразующая их в команды устройства. Для получения доступа к аппаратной части устройства драйвер должен быть частью ядра Классификация драйверов – по типам устройств (символьные, блочные).

## Типи пристроїв введення-виводу. Диски, таймери, алфавітно-цифрові термінали, графічні інтерфейси користувача, мережні термінали.

* Диски, дорожки, сектора
* *Время поиска* (seek time) – время перемещения манипулятора для позиционирования на дорожке (5-10 мс)
* *Задержка вращения* (rotational delay) – время ожидания поворота пластины так, чтобы сектор оказался под головкой (3 мс)
* *Пропускная способность передачи данных* (200 Мб/сек).

Время передачи даннях^

* T = b/(r\*N)
* b – количество передаваемых данных
* r – скорость вращения (об/с)
* N – число байтов на дорожке

RAID – набор физических дисков, «видимый» ОС как единый логический диск

Таймеры управляют устройствами, передающими в систему информацию о времени (текущее время суток, учет процессорного времени). Используют отдельный набор системных вызовов.

Таймеры отложенного выполнения используют для оповещения процесса об окончании некоторого промежутка времени Хранят число прерываний, оставшееся до сработки таймера (данное число уменьшается при каждом прерывании).

* Терминальный ввод-вывод
* Аппаратный терминал (дисплей + клавиатура) используется для организации работы множества пользователей
* Подключается через последовательный интерфейс
* Работает в символьном режиме (обмен данными и их отображение выполняется посимвольно)
* Символы делятся на отображаемые и управляющие (ESC-последовательности)

Каждому терминалу соответствует файл символьного устройства.

## Файли

Правила именования

* ограничения на длину имени (поддержка длинных имен и генерация коротких синонимов)
* ограничения на используемые символы
* чувствительность к регистру символов
* понятие расширения файла
* уникальность имени и символьные ссылки

Типы файлов

* Обычные (регулярные)
  + Текстовые
  + Двоичные
* Специальные (связанные с устройствами ввода-вывода)
  + Устройства
  + Сокеты
  + Каналы

Выделение дискового пространства файлу:

* Непрерывная последовательность блоков
* Связный список
* Связный список с использованием индекса (file allocation table – FAT)

## Каталоги

* Специальный файл, предназначенный для организации многоуровневой структуры хранения файлов
* Содержит список хранящихся в нем файлов
* Каждая запись имеет фиксированную структуру и длину
* Полное и относительное имя файла
* Монтируемые файловые системы
* Текущий каталог (.)
* Родительский каталог (..)

## Реалізації та приклади файлових систем

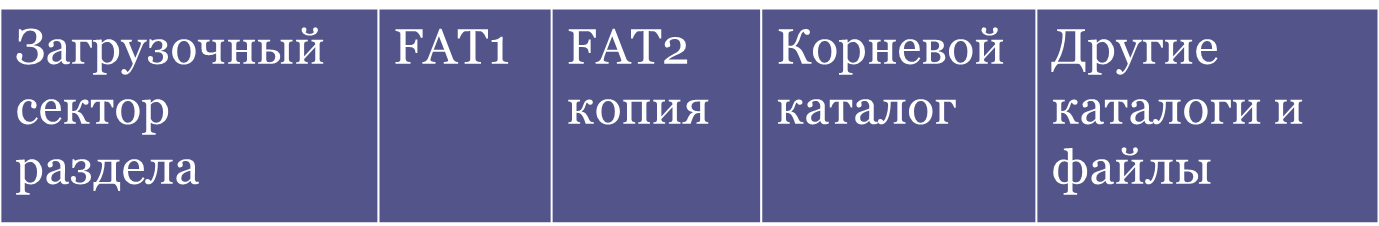
Файловые системы семейства Windows:

* FAT (FAT-12, FAT-16, FAT-32) – ориентирована на небольшие диски и простые структуры каталогов
* exFAT (FAT-64)
* NTFS

Термины

* **Сектор** – наименьшая единица физического хранения на диске (степень 2 и почти всегда 512 байт)
* **Кластер** – один или несколько (степень 2) последовательных секторов на одной дорожке
* **Том** – пространство, занимаемое файловой системой. Может занимать весь диск, его часть или несколько дисков

FAT – структура тома:



FAT32:

Позволяет создавать разделы более 2Гб (до 2 Тб). Загрузочная запись содержит резервные копии наиболее важных структур данных.

exFAT – extended FAT:

Ориентирована на мобильные устройства хранения. Предельный размер файла 264 байт.

NTFS:

Возможность установки прав доступа к отдельным файлам и каталогам. Журналируемая файловая система – восстановление после сбоев. Поддержка жестких и символьных ссылок.

## Определить время загрузки с диска программы определенного размера, если известно его среднее время поиска, время вращения, вместимость одной дорожки диска и размер страницы.

* *Время поиска* (seek time) – время перемещения манипулятора для позиционирования на дорожке (5-10 мс)
* *Задержка вращения* (rotational delay) – время ожидания поворота пластины так, чтобы сектор оказался под головкой (3 мс)
* *Пропускная способность передачи данных* (200 Мб/сек).

Время передачи данных: T = b/(r\*N)

* b – количество передаваемых данных
* r – скорость вращения (об/с)
* N – число байтов на дорожке

Размер программы на диске – 64 Кб. Каждая дорожка – 32 Кб. Среднее время поиска – 10 мс. Время вращения – 10 мс. Размер страницы 2 Кб. =>

=> 32 Кб 10 мс (вращ.)

2 Кб (1 страница) x мс.

x= (2\*10)/32 = 0.625 (мс)

Пусть страницы – на разных дорожках. Тогда общее время T=Nстр. \* (Tпоиск. + Tсчит.)= (64/2)\* (10+0,625).

## Вычислить время считывания файла, если известны параметры размещения файла на диске и параметры доступа и считывания данных с диска

Есть гибкий диск с шагом чередования 2. На каждой дорожке по 8 секторов по 512 б.Скорость вращения диска 300 об/мин. На первоначальное позиционирование необходимо ½ оборота. =>

=> на считывание одной дорожки – (0,5 (нач. поз.) + 1 + 1 (шаг чередования – 2))/(300/60)=0,5 с. Скорость считывания 1 дорожки – (8\*512)/0,5.

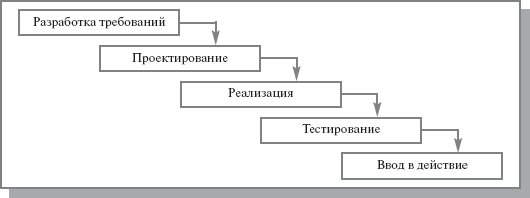
# Технологія створення програмних продуктів

## Життєві цикли програмного забезпечення систем обробки інформації

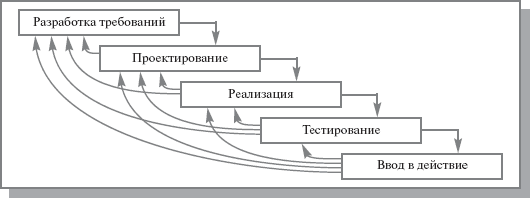
Жизненный цикл программного обеспечения (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Извесны и используются следующие модели жизненного цикла:

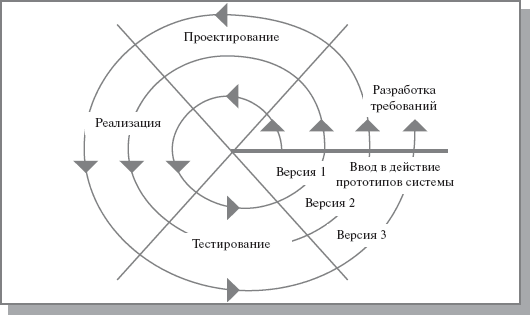
*Каскадная* модель предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе.



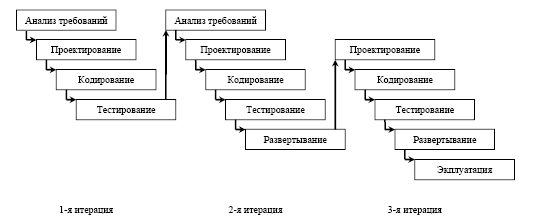
*Поэтапная* модель с промежуточным контролем. Разработка ИС ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных этапах; время жизни каждого из этапов растягивается на весь период разработки.



*Спиральная* модель. На каждом витке спирали выполняется создание очередной версии продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка. Особое внимание уделяется начальным этапам разработки - анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования).

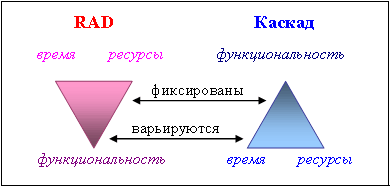


*Ітеративні* або *инкрементальные* моделі припускають розбиття що розробляється ПО на набір частин, які розробляються послідовно, один за одним так, щоб на самому початку — на першій ітерації — можна було розробити частину, не залежну від інших, пройшовши повний цикл робіт на ній, потім оцінити результати і на наступній ітерації або переробити першу частину, або розробити наступну частину, яка може залежати від першої, або якось сумістити доопрацювання першої частини з додаванням нових функцій.

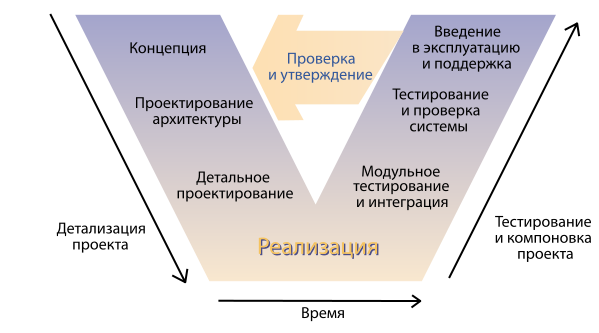


*Rapid Application Development* (RAD) – это жизненный цикл процесса проектирования, созданный для достижения более высоких скорости разработки и качества ПО, чем это возможно при традиционном подходе к проектированию.

RAD предполагает, что разработка ПО осуществляется небольшой командой разработчиков за срок порядка трех-четырех месяцев путем использования инкрементного прототипирования с применением инструментальных средств визуального моделирования и разработки.



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.



## Мери і метрики програмного забезпечення

При планировании программного проекта надо оценить людские ресурсы (в человеко-месяцах), продолжительность (в календарных датах), стоимость (в тысячах долларов). Обычно исходят из прошлого опыта. Если новый проект по размеру и функциям похож на предыдущий проект, вполне вероятно, что потребуются такие же ресурсы, время и деньги.

### Размерно-ориентированные метрики

Размерно-ориентированные метрики прямо измеряют программный продукт и процесс его разработки. Основываются размерно-ориентированные метрики на LOC-оценках (Lines Of Code). LOC-оценка — это количество строк в программном продукте.

;

;

;

.

*Достоинства размерно-ориентированных метрик:*

1) широко распространены;

2) просты и легко вычисляются.

*Недостатки размерно-ориентированных метрик:*

1) зависимы от языка программирования;

2) требуют исходных данных, которые трудно получить на начальной стадии проекта;

3) не приспособлены к непроцедурным языкам программирования.

### Функционально-ориентированные метрики

Функционально-ориентированные метрики косвенно измеряют программный продукт и процесс его разработки. Вместо подсчета LOC-оценки при этом рассматривается не размер, а функциональность или полезность продукта.

Используется 5 информационных характеристик.

1. *Количество внешних вводов.* Подсчитываются все вводы пользователя, по которым поступают разные прикладные данные. Вводы должны быть отделены от запросов, которые подсчитываются отдельно.

2. *Количество внешних выводов.* Подсчитываются все выводы, по которым к пользователю поступают результаты, вычисленные программным приложением. В этом контексте выводы означают отчеты, экраны, распечатки, сообщения об ошибках. Индивидуальные единицы данных внутри отчета отдельно не подсчитываются.

3. *Количество внешних запросов.* Под запросом понимается диалоговый ввод, который приводит к немедленному программному ответу в форме диалогового вывода. При этом диалоговый ввод в приложении не сохраняется, а диалоговый вывод не требует выполнения вычислений. Подсчитываются все запросы — каждый учитывается отдельно.

4. *Количество внутренних логических файлов.* Подсчитываются все логические файлы (то есть логические группы данных, которые могут быть частью базы данных или отдельным файлом).

5. *Количество внешних интерфейсных файлов.* Подсчитываются все логические файлы из других приложений, на которые ссылается данное приложение.

После сбора всей необходимой информации приступают к расчету метрики — *количества функциональных указателей FP* (Function Points).

Количество функциональных указателей вычисляется по формуле

FP = Общее количество х (0,65+ 0,01 x),

где *Fi* — коэффициенты регулировки сложности.

Каждый коэффициент может принимать следующие значения: 0 — нет влияния, 1 — случайное, 2 — небольшое, 3 — среднее, 4 — важное, 5 — основное.

Значения выбираются эмпирически

После вычисления FP на его основе формируются метрики производительности, качества и т. д.:

;

;

;

.

Область применения метода функциональных указателей — коммерческие информационные системы. Для продуктов с высокой алгоритмической сложностью используются метрики *указателей свойств* (Features Points). Они применимы к системному и инженерному ПО, ПО реального времени и встроенному ПО.

Для вычисления указателя свойств добавляется одна характеристика — *количество алгоритмов.*

*Достоинства функционально-ориентированных метрик:*

1. Не зависят от языка программирования.

2. Легко вычисляются на любой стадии проекта.

*Недостаток функционально-ориентированных метрик:* результаты основаны на субъективных данных, используются не прямые, а косвенные измерения. FP-оценки легко пересчитать в LOC-оценки.

## Інженерія вимог до програмних систем

Разделим требования на две большие группы – функциональные и нефункциональные.

*Функциональные* требования являются детальным описанием поведения и *сервисов* системы, ее функционала. Они определяют то, что система должна уметь делать.

*Нефункциональные* требования не являются описанием функций системы. Этот вид требований описывает такие характеристики системы, как *надежность*, особенности поставки (наличие инсталлятора, документации), определенный уровень качества (например, для новой Java-машины это будет означать, что она удовлетворяет набору тестов, поддерживаемому компанией *Sun*). Сюда же могут относиться требования на средства и процесс разработки системы, требования к *переносимости*, соответствию стандартам и т.д. Требования этого вида часто относятся ко всей системе в целом. На практике, особенно начинающие специалисты, часто забывают про некоторые важные нефункциональные требования.

Сформулируем ряд важных свойств требований.

– ясность, недвусмысленность — однозначность понимания требований заказчиком и разработчиками;

– полнота и непротиворечивость;

– необходимый уровень детализации;

– прослеживаемость — важно видеть то или иное требование в различных моделях, документах, наконец, в коде системы;

– тестируемость и проверяемость — необходимо, чтобы существовали способы оттестировать и проверить данное требование;

– модифицируемость.

Определяют следующие виды *деятельности* при работе с требованиями.

* *Выделение требований (requirements elicitation)*, нацеленное на выявление всех возможных источников требований и ограничений на работу системы и *извлечение тренований* из этих источников.
* *Анализ требований (requirements analysis)*, целью которого является обнаружение и устранение противоречий и неоднозначностей в требованиях, их уточнение и систематизация.
* *Описание требований (requirements specification)*. В результате этой *деятельности* требования должны быть оформлены в виде структурированного набора документов и моделей, который может *систематически* анализироваться.
* *Валидация требований (requirements validation)*, которая решает задачу оценки понятности сформулированных требований и их характеристик, необходимых, чтобы разрабатывать ПО на их основе, в первую очередь, непротиворечивости и полноты, а также соответствия корпоративным стандартам на техническую документацию.

## Функціональне тестування програмних систем

Функціональне тестування, воно ж тестування чорного ящика — тести для нього, а також використовувані критерії повноти проведеного тестування визначають на основі вимог до функціональності.

**Функціональне тестування** – один з основних видів незалежного тестування програмного забезпечення, спрямований на перевірку реалізованості функціональних вимог.  Інакше кажучи, фахівці з функціонального тестування визначають, чи вирішує розроблене ПЗ завдання, заради якого воно було створено, чи задовольняє воно потреби замовника / користувача.

Основні критерії функціонального тестування:

* Функціональна придатність продукту.
* Точність виконання поставлених завдань.
* Відповідність стандартам розробки.
* Відповідність кінцевої реалізації продукту пред’явленим функціональним вимогам.
* Захищеність від несанкціонованого використання і некоректних дій користувачів.

Рівні тестування:

* Модульне (компонентне) (Unit testing)
* Інтеграційне (Integration Testing)
* Системне (System testing)
* Регресійне (Regression testing)
* Приймальне (Acceptance testing)

**Переваги функціонального тестування:**

* імітує фактичне використання системи.

**Недоліки функціонального тестування:**

* можливість упущення логічних помилок в програмному забезпеченні;
* ймовірність надмірного тестування.

## Структурне тестування програмних систем

Структурне тестування, воно ж тестування білого ящика — тести створюються на основі знань про структуру самої системи і про те, як вона працює. Критерії повноти засновані на відсотку елементів коду, які відпрацювали в ході виконання тестів (покриття коду тестами). Для оцінки ступеня відповідності вимогам можуть притягуватися додаткові знання з дослідження вимог.

Структурне тестування програмного забезпечення може бути реалізоване такими методами:

* тестуванням маршрутів;
* тестуванням циклів;
* тестуванням обробки даних.

**Тестування маршрутів**

Тестування структури маршрутів програмних модулів виконують шляхом перевірки коректності виділених маршрутів виконання програм і виявлення логічних помилок формування маршрутів. На практиці за відсутності впорядкованого аналізу потоків управління деякі маршрути у програмі (до 50%) виявляються пропущеними під час тестування.

**Тестування циклів**

Наявність циклів у програмних модулях здатне різко збільшувати складність їхнього тестування. На складність тестування циклу впливає його структура та два параметри: число маршрутів у тілі циклу та число ітерацій циклу. За умови зростання кожного з цих параметрів пропорційно зростає їхній добуток, а отже, і складність тестування. Тому вичерпне тестування реальних складних програм із циклами практично неможливе.

**Тестування обробки даних**

Функціонування будь-якої програми можна розглядати як обробку потоку даних, переданих від входу у програму до її виходу. Вхідні дані послідовно використовуються для визначення ряду проміжних результатів аж до одержання необхідного набору вихідних даних.

## Види архітектури програмних систем. Межі використання різних архітектур.

Архитектура ПО обычно содержит несколько видов, которые аналогичны различным типам чертежей в строительстве зданий.

Архитектурный вид состоит из 2 компонентов:

* Элементы
* Отношения между элементами

Архитектурные виды можно поделить на 3 основных типа:

1. Модульные виды (англ. *module views*) — показывают систему как структуру из различных программных блоков.
2. Компоненты-и-коннекторы (англ. *component-and-connector views*) — показывают систему как структуру из параллельно запущенных элементов (компонентов) и способов их взаимодействия (коннекторов).
3. Размещение (англ. *allocation views*) — показывает размещение элементов системы во внешних средах.

Примеры модульных видов:

* Декомпозиция (англ. *decomposition view*) — состоит из модулей в контексте отношения «является подмодулем»
* Использование (англ. *uses view*) — состоит из модулей в контексте отношения «использует» (т.е. один модуль использует сервисы другого модуля)
* Вид уровней (англ. *layered view*) — показывает структуру, в которой связанные по функциональности модули объединены в группы (уровни)
* Вид классов/обобщений (англ. c*lass/generalization view*) — состоит из классов, связанные через отношения «наследуется от» и «является экземпляром»

Примеры видов компонентов-и-коннекторов:

* Процессный вид (англ. *process view*) — состоит из процессов, соединённых операциями коммуникации, синхронизации и/или исключения
* Параллельный вид (англ. *concurrency view*) — состоит из компонентов и коннекторов, где коннекторы представляют собой «логические потоки»
* Вид обмена данными (англ. *shared-data (repository) view*) — состоит из компонентов и коннекторов, которые создают, сохраняют и получают постоянные данные
* Вид клиент-сервер (англ. *client-server view*) — состоит из взаимодействующих клиентов и серверов и коннектором между ними (например, протоколов и общих сообщений)

Примеры видов размещения:

* Развертывание (англ. *deployment view*) — состоит из программных элементов, их размещения на физических носителях и коммуникационных элементов
* Внедрение (англ. *implementation view*) — состоит из программных элементов и их соответствия файловым структурам в различных средах (разработческой, интеграционной и т.д.)
* Распределение работы (англ. *work assignment view*) — состоит из модулей и описания того, кто ответственен за внедрение каждого из них

Хотя было разработано несколько языков для описания архитектуры программного обеспечения, в настоящий момент нет согласия по поводу того, какой набор видов должен быть принят в качестве эталона. В качестве стандарта «для моделирования программных систем (и не только)» был создан язык UML.

## Моделі управління програмних систем

### Адміністративна модель (теорія X)

Традиційна ієрархічна модель.

Характерні риси моделі:

* Владна піраміда.
* Чіткий розподіл ролей і обов'язків.
* Чіткий розподіл відповідальності.
* Проходження інструкціям, процедурам, технологіям.
* Роль менеджера: планування, контроль, ухвалення основних рішень.

Переваги моделі: ясність, простота, прогнозованість. Модель добре поєднується з каскадною моделлю життєвого циклу і застосовна в тих же випадках, що і каскадна модель. Модель ефективна у разі сталого процесу.

Недоліки моделі пов'язані з тим, що адміністративна система прагне самозбереженню (стабільності) і погано сприйнятлива до зміни ситуації – нові типи проектів, застосування нових технологій, оперативна реакція на зміну ринку. Крім того, в адміністративній моделі погано уживаються індивідуалісти і генератори ідей.

### Модель хаосу (теорія Y)

У основі моделі хаосу лежить Теорія Y, яка є повною протилежністю Теорії X. Основна теза Теорії Y: робота — природна і приємна діяльність і більшість людей, насправді, дуже відповідальні і не ухиляються від роботи.

Характерними рисами моделі хаосу є:

* Відсутність явно виражених ознак влади
* Роль менеджера – поставити завдання, забезпечити ресурсами, не заважати і стежити, щоб не заважали інші
* Відсутність інструкцій і регламентованих процедур
* Індивідуальна ініціатива – рішення з проблеми приймається там, де проблема виявлена
* Процес нагадує творчу гру учасників.

Переваги такої моделі в тому, що творча ініціатива учасників нічим не зв'язана і потенціал учасників розкривається повною мірою. Це буває особливо ефективно у разі, коли для вирішення проблеми потрібний пошук нових підходів, методів, ідей і засобів.

### Відкрита архітектура (теорія Z)

Адміністративна і хаотична моделі є двома «крайнощами», між якими знаходяться безліч моделей, що поєднують переваги «крайніх» моделей. Однією з таких моделей є модель відкритої архітектури, заснована на Теорії Z.

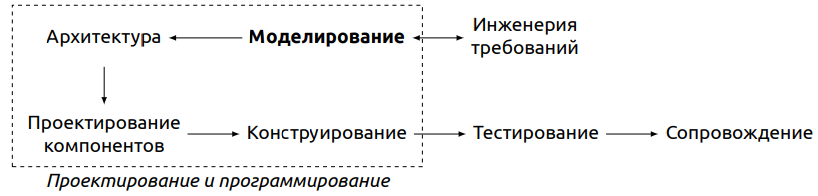
Особливостями цієї моделі є:

* Адаптація до умов роботи – якщо робимо незалежні модулі, то розходимося і робимо, якщо потрібна архітектура бази даних, то збираємося разом і обговорюємо ідеї.
* Колективне обговорення проблем, вироблення консенсусу і ухвалення рішення – не всі можуть погодиться, але ухвалене рішення є колективним і через це – обов'язковим для всіх.
* Розподілена відповідальність – відповідають всі, хто обговорював, виробляв, приймав.
* Динаміка складу робочих груп залежно від поточних завдань.
* Відсутність спеціалізації – учасники міняються ролями і функціями і можуть при необхідності замінити один одного.
* Завдання менеджера – активна участь в процесі, контроль конструктивності обговорень, забезпечення можливості активної участі всіх.

Відкрита архітектура є гнучкішою, такою, що адаптується, настроюється на ситуацію. Вона дає можливість проявити себе всім членам команди – в ній можуть уживатися і індивідуалісти і колективісти. Колективне обговорення висловлених ідей дозволяє залишати тільки прагматичні ідеї.

## Моделювання програмних систем. Типи і призначення моделей структурного моделювання.

https://softandware.org.ua/wp-content/uploads/2014/11/07-modeling.pdf



Представление – абстрактная модель системы, выделяющая ее характеристики, соответствующие определенному аспекту ее функционирования.

Основные представления:

▶ контекстное представление модель окружения, в котором выполняется система;

▶ взаимодействия связи системы с окружением, а также элементов в системе;

▶ структурное представление организация системы и данных для обработки;

▶ поведение модель реагирования системы в ответ на внешние события.

### Контекстные модели

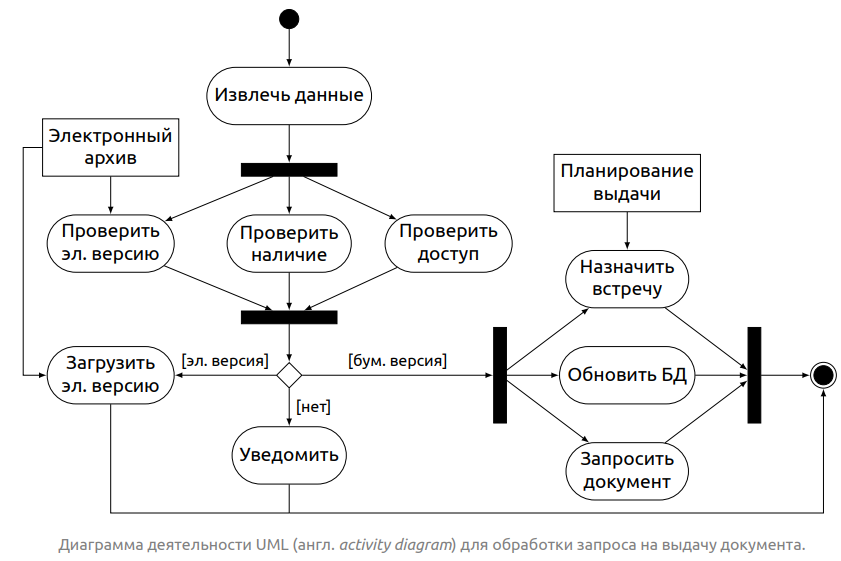
Цель:

▶ разграничение функций системы и ее окружения;

▶ определение компонентов, которые надо имплементировать, и используемых интерфейсов.



Модель процесса:



### Модели взаимодействия

Цели:

▶ отладка взаимодействия с пользователями;

▶ выработка пользовательских требований;

▶ определение возможных «узких мест» и проблем коммуникации;

▶ отработка производительности (англ. performance) и надежности (англ. dependability).

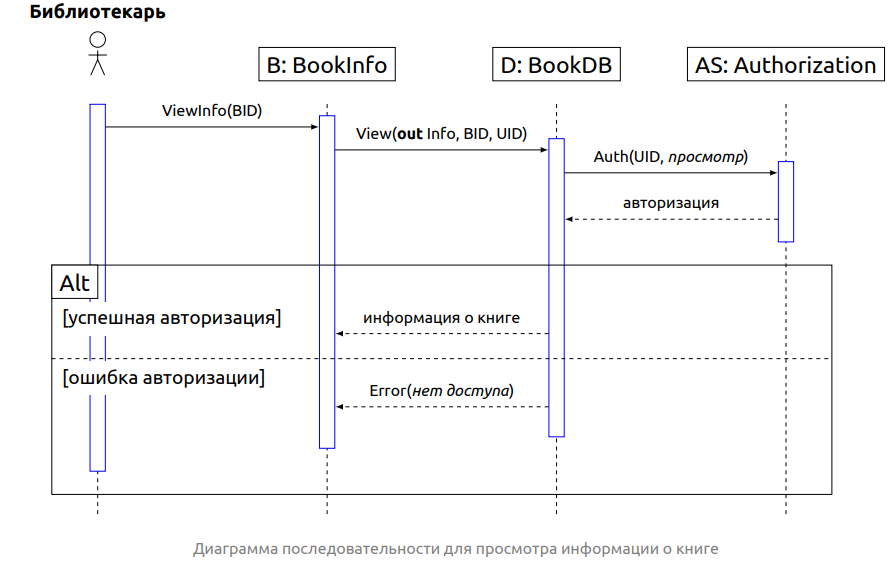
Типы моделей:

▶ варианты использования моделирование взаимодействия системы с актерами

(пользователями или другими системами); (use case diagram, діаграма прецидентів)

▶ диаграммы последовательностей моделирование взаимодействия компонентов

системы. 



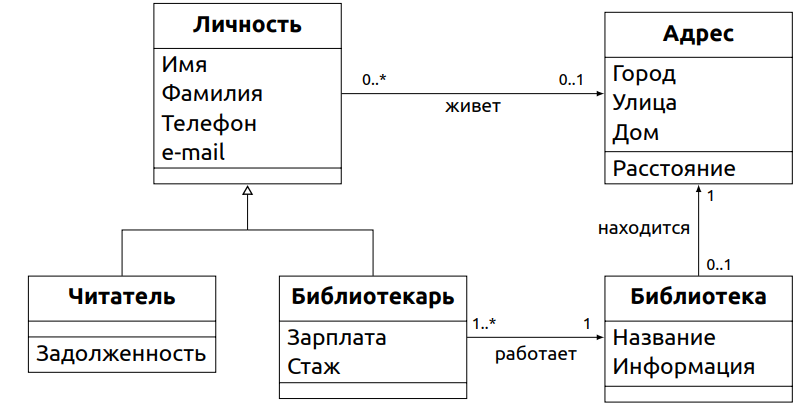
### Структурные модели

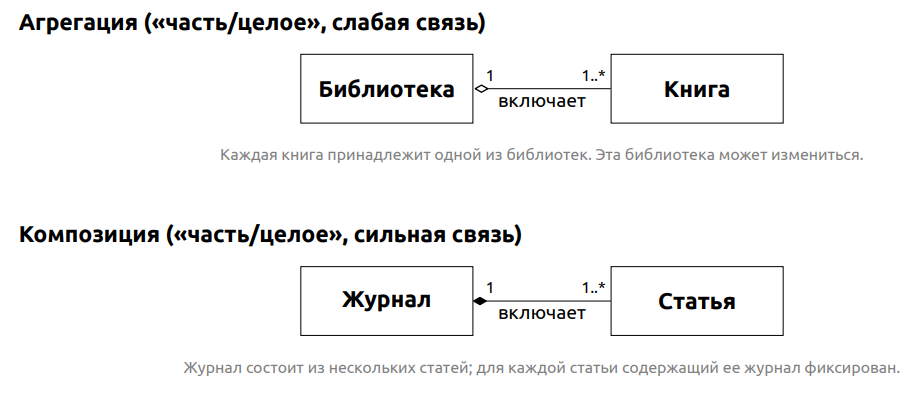
Цели:

▶ определение архитектуры системы;

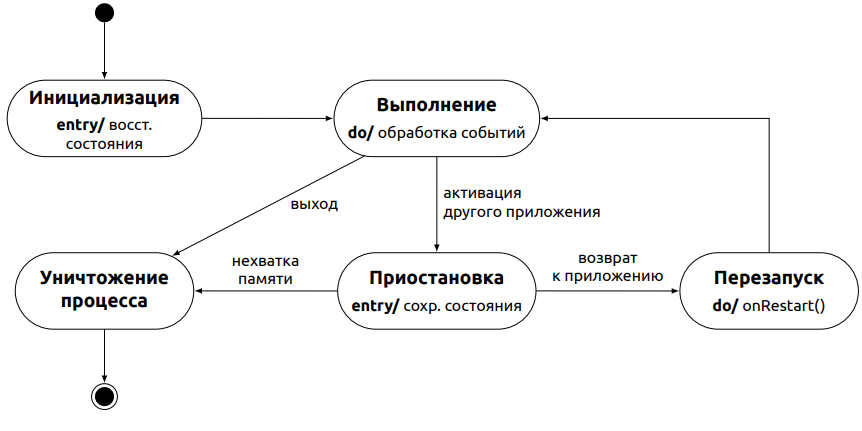
▶ определение структур хранения данных.





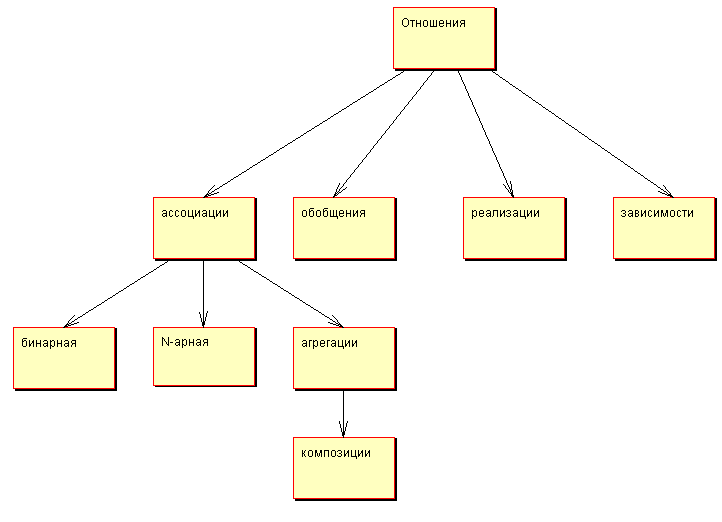


### Модели поведения

Цель: определение реакции системы на внешние и внутренние входные сигналы. 

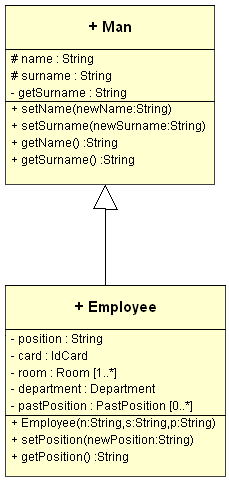
## Уніфікована мова моделювання (UML), значення UML в процесі проектування програмного забезпечення.

Диаграмма классов UML позволяет обозначать отношения между классами и их экземплярами.



### Обобщения

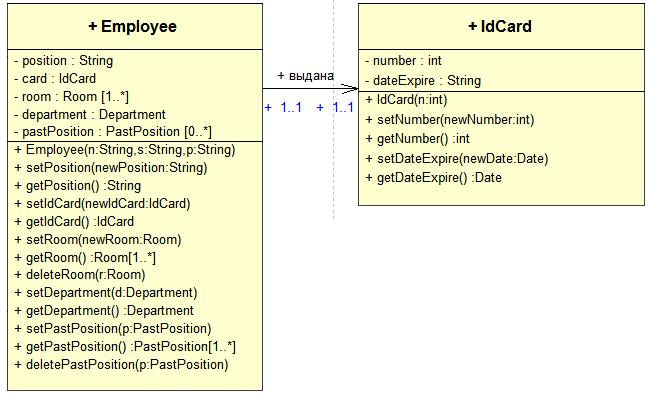
Отношение обобщения — это наследование



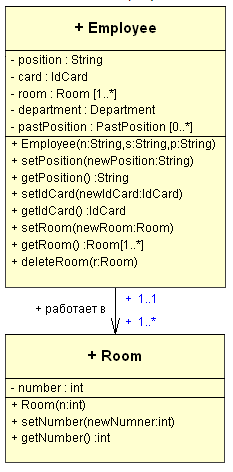
### Ассоциация

Ассоциация показывает отношения между объектами-экземплярами класса.

Каждому сотруднику может соответствовать только одна идентификационная карточка, мощность связи 1 к 1.

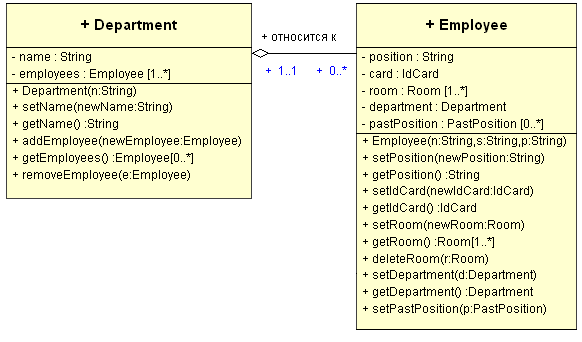


Каждому объекты работник(Employee) может соответствовать несколько рабочих помещений.



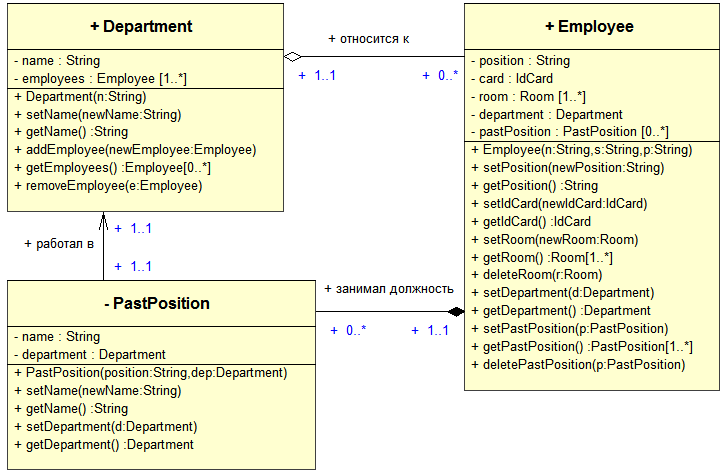
### Агрегация

В каждом отделе может работать один или более человек. Можно сказать, что отдел включает в себя одного или более сотрудников и таким образом их агрегирует. На предприятии могут быть сотрудники, которые не принадлежат ни одному отделу, например, директор предприятия.



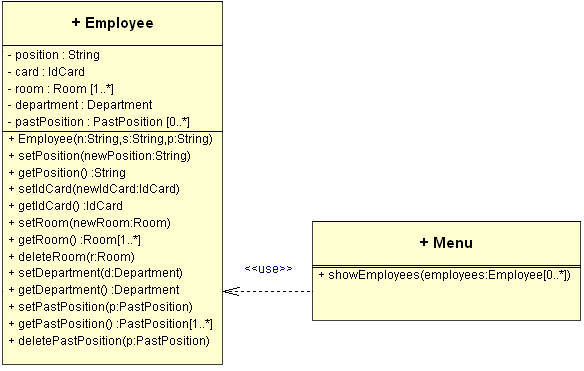
### Композиция

Данные о прошлых занимаемых должностях являются частью данных о сотруднике, таким образом между ними связь целое-часть и в то же время, данные о прошлых должностях не могут существовать без объекта типа «Employee». Уничтожение объекта «Employee» должно привести к уничтожению объектов «pastPosition».



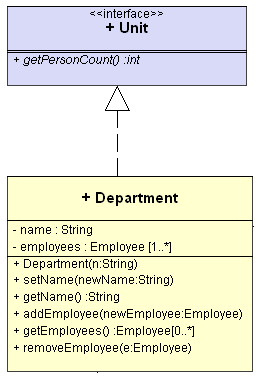
### Зависимость

Параметром для метода является массив объектов «Employee». Таким образом, изменения внесенные в класс «Employee» могут потребовать и изменения класса «Menu».



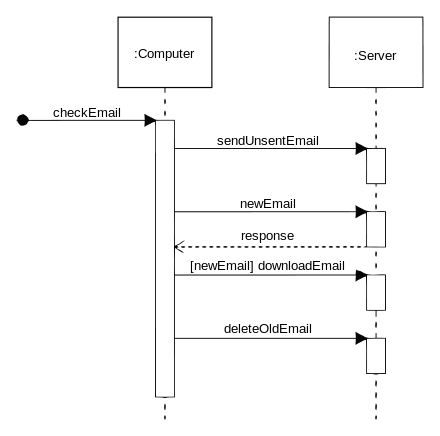
### Реализация

Если представить, что организация может делиться не только на отделы, а например, на цеха, филиалы и т.д. Интерфейс «Unit» представляет собой самую абстрактную единицу деления.



## Диаграмма последовательности

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни» (англ. lifeline), отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

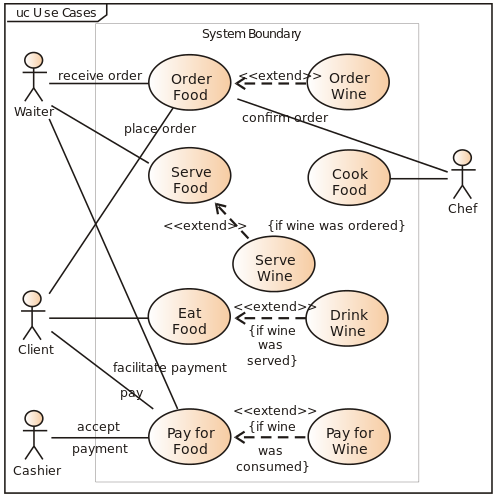


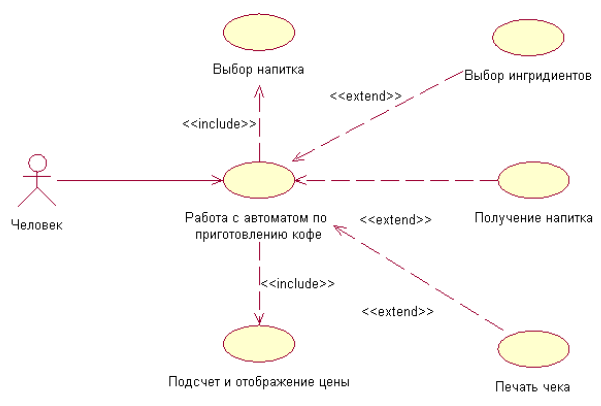


## Діаграма прецедентів

Расширение (англ. Extend) — разновидность отношения зависимости между базовым вариантом использования и его специальным случаем.

Включение (англ. Include) — определяет взаимосвязь базового варианта использования с другим вариантом использования, функциональное поведение которого всегда задействуется базовым вариантом использования.





# Технології захисту інформації

## Погрози інформайійної безпеки

Угроза – это потенциальная возможность определенным образом нарушить целостность информации.

Атака – попытка реализовать угрозу.

Источники угрозы – потенциальные злоумышленники.

Окно опасности – промежуток времени, когда появляется возможность использовать слабое место, до того момента, когда оно будет ликвидировано.

*Классификация угроз:*

- по источнику (его местонахождению) – на внутренние (возникают непосредственно на объекте и обусловлены взаимодействием между его элементами или субъектами) и внешние (возникают вследствие его взаимодействия с внешними объектами);

- по вероятности реализации – на потенциальные и реальные;

- по размерам наносимого ущерба - на общие (наносят вред объекту безопасности в целом, оказывая существенное негативное воздействия на условия его деятельности), локальные (затрагивают условия существования отдельных элементов объекта безопасности) и частные (наносят вред отдельным свойствам элементов объекта или отдельным направлениям его деятельности);

- по природе происхождения - на случайные (не связанные с действиями персонала, состоянием и функционированием объекта информационной безопасности, такие как отказы, сбои и ошибки в работе средств автоматизации, стихийные бедствия и другие чрезвычайные обстоятельства) и преднамеренные (обусловлены злоумышленными действиями людей);

- по предпосылкам возникновения – на объективные (вызваны недостатком системы информационной безопасности объекта, например, несовершенством разработанных нормативно-методических и организационно-плановых документов, отсутствием подготовленных специалистов по защите информации и т.п.) и субъективные (обусловлены деятельностью персонала объекта безопасности, например, ошибками в работе, низким уровнем подготовки в вопросах защиты информации, злоумышленными действиями или намерениями посторонних лиц);

- по видам объектов безопасности – на угрозы собственно информации (обусловлены попытками получения защищаемой информации различными способами и методами независимо от ее местонахождения), угрозы персоналу объекта (связаны с уменьшением влияния персонала на ситуацию в сфере обеспечения информационной безопасности, с попытками получения конфиденциальной информации от допущенного к ней персонала), угрозы деятельности по обеспечению информационной безопасности (направлены на снижение эффективности или нейтрализацию усилий, предпринимаемых руководством и персоналом объекта безопасности для исключения утечки защищаемых сведений, утраты или хищения носителей, ослабление системы защиты информации).

К наиболее важным свойствам угрозы относятся избирательность, предсказуемость и вредоносность. *Избирательность* характеризует нацеленность угрозы на нанесение вреда тем или иным конкретным свойствам объекта безопасности. *Предсказуемость* характеризует наличие признаков возникновения угрозы, позволяющих заранее прогнозировать возможность появления угрозы и определять конкретные объекты безопасности, на которые она будет направлена. *Вредоносность* характеризует возможность нанесения вреда различной тяжести объекту безопасности. Вред, как правило, может быть оценен стоимостью затрат на ликвидацию последствий проявления угрозы либо на предотвращение ее появления.

Классификация мер обеспечения компьютерной безопасности по способам осуществления:

1. правовые (законодательные) - действующие в стране законы, указы и нормативные акты, регламентирующие правила обращения с информацией, закрепляющие права и обязанности участников информационных отношений в процессе ее обработки и использования, а также устанавливающие ответственность за нарушения этих правил, препятствуя тем самым неправомерному использованию информации и являющиеся сдерживающим фактором для потенциальных нарушителей

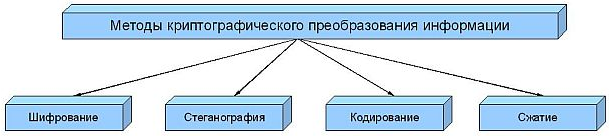
2. морально-этические - нормы поведения, которые традиционно сложились или складываются по мере распространения ЭВМ в стране или обществе.

3. организационные (административные) - это меры организационного характера, регламентирующие процессы функционирования системы обработки данных, использование ее ресурсов, деятельность персонала, а также порядок взаимодействия пользователей с системой таким образом, чтобы в наибольшей степени затруднить или исключить возможность реализации угроз безопасности.

4. физические - основаны на применении разного рода механических, электро- или электронно-механических устройств и сооружений, специально предназначенных для создания физических препятствий на возможных путях проникновения и доступа потенциальных нарушителей к компонентам системы и защищаемой информации, а также технических средств визуального наблюдения, связи и охранной сигнализации

5. технические (аппаратурные и программные) - основаны на использовании различных электронных устройств и специальных программ, выполняющих (самостоятельно или в комплексе с другими средствами) функции защиты (идентификацию и аутентификацию пользователей, разграничение доступа к ресурсам, регистрацию событий, криптографическое закрытие информации и т.д.).

## Криптографічні методи закриття інформації



Процесс шифрования заключается в проведении обратимых математических, логических, комбинаторных и других преобразований исходной информации, в результате которых зашифрованная информация представляет собой хаотический набор букв, цифр, других символов и двоичных кодов.

Для шифрования информации используются алгоритм преобразования и ключ. Как правило, алгоритм для определенного метода шифрования является неизменным. Исходными данными для алгоритма шифрования служит информация, подлежащая зашифрованию, и ключ шифрования. Ключ содержит управляющую информацию, которая определяет выбор преобразования на определенных шагах алгоритма и величины операндов, используемых при реализации алгоритма шифрования. Операнд – это константа, переменная, функция, выражение и другой объект языка программирования, над которым производятся операции.

В отличие от других методов криптографического преобразования информации, методы стеганографии позволяют скрыть не только смысл хранящейся или передаваемой информации, но и сам факт хранения или передачи закрытой информации. В основе всех методов стеганографии лежит маскирование закрытой информации среди открытых файлов, т.е. скрываются секретные данные, при этом создаются реалистичные данные, которые невозможно отличить от настоящих. Обработка мультимедийных файлов в информационных системах открыла практически неограниченные возможности перед стеганографией.

Графическая и звуковая информация представляются в числовом виде. Так, в графических объектах наименьший элемент изображения может кодироваться одним байтом. В младшие разряды определенных байтов изображения в соответствии с алгоритмом криптографического преобразования помещаются биты скрытого файла. Если правильно подобрать алгоритм преобразования и изображение, на фоне которого помещается скрытый файл, то человеческому глазу практически невозможно отличить полученное изображение от исходного. С помощью средств стеганографии могут маскироваться текст, изображение, речь, цифровая подпись, зашифрованное сообщение.

Скрытый файл также может быть зашифрован. Если кто-то случайно обнаружит скрытый файл, то зашифрованная информация будет воспринята как сбой в работе системы. Комплексное использование стеганографии и шифрования многократно повышает сложность решения задачи обнаружения и раскрытия конфиденциальной информации.

Содержанием процесса кодирование информации является замена исходного смысла сообщения (слов, предложений) кодами. В качестве кодов могут использоваться сочетания букв, цифр, знаков. При кодировании и обратном преобразовании используются специальные таблицы или словари. В информационных сетях кодирование исходного сообщения (или сигнала) программно-аппаратными средствами применяется для повышения достоверности передаваемой информации.

Часто кодирование и шифрование ошибочно принимают за одно и тоже, забыв о том, что для восстановления закодированного сообщения, достаточно знать правило замены, в то время как для расшифровки сообщения помимо знания правил шифрования, требуется ключ к шифру.

Сжатие информации может быть отнесено к методам криптографического преобразования информации с определенными оговорками. Целью сжатия является сокращение объема информации. В то же время сжатая информация не может быть прочитана или использована без обратного преобразования. Учитывая доступность средств сжатия и обратного преобразования, эти методы нельзя рассматривать как надежные средства криптографического преобразования информации. Даже если держать в секрете алгоритмы, то они могут быть сравнительно легко раскрыты статистическими методами обработки. Поэтому сжатые файлы конфиденциальной информации подвергаются последующему шифрованию. Для сокращения времени передачи данных целесообразно совмещать процесс сжатия и шифрования информации.

## Симетричні криптосистеми

Работа симметричных шифров включает в себя два преобразования:

*C = Ek{m)* и *m = Dk{C),*

где *т* — открытый текст, *Е* — шифруюш;ая функция, *D* — расшифровываюш;ая функция, *к* — секретный ключ, *С* — шифротекст.

Следует отметить, что как шифрующая, так и расшифровывающая функции обшщеизвестны, и тайна сообщения при известном шифротексте зависит только от секретности ключа *к.*

Алгоритм называется криптосистемой с симметричным ключом если обе стороны, кторые обменивающиеся шифрованной информацией, применяют один и тот же секретный ключ. Иногда симметричные криптосистемы используют два ключа: один дляшифрования, а другой дляобратного процесса. В этом случае мы будем предполагать, что шифрующий ключ легко восстанавливается по расшифровывающему и наоборот.

Число возможных ключей должно быть очень велико. Если количество возможных ключей мало, то атакующий имеет возможность взломать шифр простым перебором вариантов. Принято считать, что вычисления, состоящие из 280 шагов, в ближайшие несколько лет будут неосуществимы. Поэтому ключ, исключаюыщй взлом простым перебором, должен насчитывать по крайней мере 80 битов.

Большинство симметричных шифров можно разделить на две больших группы. Первая — поточные шифры, где за один раз обрабатывается один элемент данных (бит или буква), а вторая — блочные шифры, в которых за один шаг обрабатывается группа элементов данных (например, 64 бита).

Однако не следует считать это разделение закостенелым. Так, например, при использовании некоторых ухищрений получают из блочного шифра - поточный и наоборот. А, например, блочный шифр с размером выходного блока 8 бит (один символ) можно считать поточным.

Блочные шифры оперируют с блоками открытого текста. К ним предъявляются следующие требования:

• достаточная криптостойкость;

• простота процедур зашифрования и расшифрования;

• приемлимая надежность.

Под криптостойкостью понимают время, необходимое для раскрытия шифра при использовании наилучшего метода криптоанализа. Надежность - доля информации, дешифруемая при помощи какого-то криптоаналитического алгоритма.

Само преобразование шифра должно использовать следующие принципы:

• Рассеивание (diffusion) - т.е изменение любого знака открытого текста или ключа влияет на большое число знаков шифротекста, что скрывает статистические свойства открытого текста;

• Перемешивание (confusion) - использование преобразований, затрудняющих получение статистических зависимостей между шифротектстом и открытым текстом.

Практически все современные блочные шифры являются композиционными - т.е состоят из композиции простых преобразований или F=F1oF2oF3oF4o..oFn, где F-преобразование шифра, Fi-простое преобразование, называемое также i-ым циклом шифрования. Само по себе преобразование может и не обеспечивать нужных свойств, но их цепочка позволяет получить необходимый результат.

## Асиметричні криптосистеми

В асимметричных криптосистемах используются два ключа: открытый и секретный. Открытый ключ может быть опубликован в справочнике наряду с именем пользователя. В результате любой желающий может зашифровать с его помощью свое письмо и послать закрытую информацию владельцу соответствующего секретного ключа. Расшифровать посланное сообщение сможет только тот, у кого есть секретный ключ. Более точно, имеют место преобразования:

сообщение + ОТКРЫТЫЙ КЛЮЧ АЛИСЫ = ШИФРОТЕКСТ

ШИФРОТЕКСТ + секретный ключ Алисы = сообщение.

Таким образом, каждый может послать Алисе секретную информацию, воспользовавшись ее открытым ключом. Но только Алиса в состоянии расшифровать сообщение, поскольку лишь у нее есть соответствующий секретный ключ.

Причина работоспособности таких криптосистем заключается в односторонней математической связи, существующей между двумя ключами, при которой информация об открытом ключе никак не помогает восстановить секретный, но владение секретным ключом обеспечивает возможность расшифровывать сообщения, зашифрованные открытым.

Необходимо найти какое-нибудь легко осуществимое на стадии шифрования математическое преобразование, которое сложно было бы обратить (без знания специальной секретной информации) для реализации второй стадии алгоритма, т. е. расшифрования. Преобразование, обладающее указанным свойством, называется односторонней функцией или функцией-ловушкой, поскольку в ее дверь войти легко (зашифровать данные), а вот выйти без ключа довольно проблематично.

Придумать такую функцию на пустом месте довольно непросто, но, к счастью, есть некоторый набор широко известных и всесторонне изученных односторонних функций. К ним относятся задача разложения целых чисел на множители, проблема вычисления дискретних логарифмов или вычисление квадратных корней по модулю составного числа.

## Криптографічні хеш-функції

Хеширование (иногда хэширование, англ. hashing) — преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования называются хеш-функциями или функциями свёртки, а их результаты называют хешем, хеш-кодом или дайджестом сообщения (англ. message digest).

Криптографическая хэш-функция H – это функция, определенная на битовых строках произвольной длины со значениями в строках битов фиксированной длины и удовлетворяющая ряду требований специфичных для криптографических приложений. Ее значение часто называют хэш-кодом или хэш-значением.

Для того, чтобы хеш-функция H считалась криптографически стойкой, она должна удовлетворять трем основным требованиям, на которых основано большинство применений хеш-функций в криптографии:

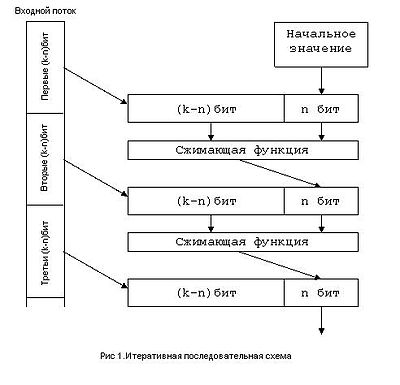
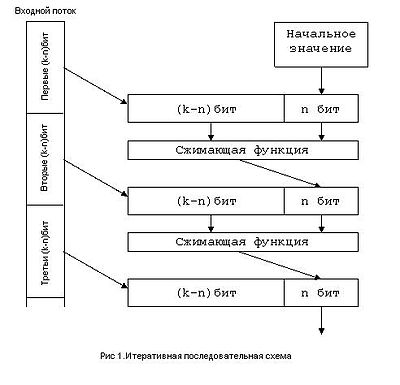
* Необратимость или стойкость к восстановлению прообраза: для заданного значения хеш-функции m должно быть вычислительно невозможно найти блок данных X, для которого H(X) = m.
* Стойкость к коллизиям первого рода или восстановлению вторых прообразов: для заданного сообщения M должно быть вычислительно невозможно подобрать другое сообщение N, для которого H(N) = H(M).
* Стойкость к коллизиям второго рода: должно быть вычислительно невозможно подобрать пару сообщений (М,М’), имеющих одинаковый хеш.

Данные требования не являются независимыми:

* Обратимая функция нестойка к коллизиям первого и второго рода.
* Функция, нестойкая к коллизиям первого рода, нестойка к коллизиям второго рода; обратное неверно.

Не доказано существование необратимых хеш-функций, для которых вычисление какого-либо прообраза заданного значения хеш-функции теоретически невозможно. Обычно нахождение обратного значения является лишь вычислительно сложной задачей. n-битная хеш-функция считается криптостойкой, если вычислительная сложность нахождения коллизий для неё близка к 2n/2.

Для криптографических хеш-функций также важно, чтобы при малейшем изменении аргумента значение функции сильно изменялось (лавинный эффект). В частности, значение хеша не должно давать утечки информации даже об отдельных битах аргумента. Это требование является залогом криптостойкости алгоритмов хеширования пользовательских паролей для получения ключей.

Принципы построения

### Итеративная последовательная схема

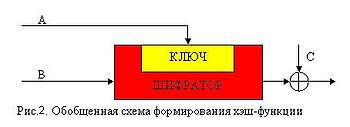
Ядром алгоритма является сжимающая функция — преобразование k входных в n выходных бит, где n — разрядность хеш-функции, а k – произвольное число большее n.

Входной поток разбивается на блоки по (k-n)бит. Алгоритм использует временную переменную размером в n бит, в качестве начального значения которой берется некое, общеизвестное число. Каждый следующий блок данных объединяется с выходным значением сжимающей функции на предыдущей итерации. Значением хеш-функции являются выходные n бит последней итерации. Каждый бит выходного значения хеш-функции зависит от всего входного потока данных и начального значения. Таким образом достигается лавинный эффект.

Размер входного потока данных должен быть кратен (k-n). Как правило, перед началом алгоритма данные расширяются неким, заранее известным, способом.

### Сжимающая функция на основе симметричного блочного алгоритма

В качестве сжимающей функции можно использовать симметричный блочный алгоритм шифрования. Для обеспечения большей безопасности можно использовать в качестве ключа блок данных предназначенный к хешированию на данной итерации, а результат предыдущей сжимающей функции в качестве входа.

**

*A, B и C могут принимать значения Mi, Hi − 1,*  *или быть константой, где Mi — i-ый блок входного потока,*  *— сложение по модулю 2, Hi — результат i-ой итерации.*

Основным недостатком хеш-функций, спроектированных на основе блочных алгоритмов, является низкая скорость работы.

## Цифрові підписи в системах електронного документообігу

Применение криптографических хэш-функций позволяет создать схему подписи RSA без восстановления сообщения, что намного эффективней для длинных сообщений. Предположим, нам дано длинное сообщение М для визирования. Сначала вычисляется h(M) и потом применяется преобразование подписи RSA к хэш-значению h(M)d  т.е . подпись получается как

S = h(M)d (mod N).

Наконец, подпись и само сообщение передаются вместе в виде пары (М, S). Проверка пары (М, S) состоит из трех этапов:

* «Шифрование» S с помощью шифрующей экспоненты RSA для получения Н': Н' = SE (mod N).
* Вычисление h(M) по М.
* Проверка равенства Н' h(M), Если оно верно, то подпись законна. В противном случае — незаконна.

Производится шифрование не самого сообщения, а значение хеш-функции взятой от сообщения. Данный метод обеспечивает следующие преимущества:

* Понижение вычислительной сложности. Как правило, документ значительно больше его хеша.
* Повышение криптостойкости. Криптоаналитик не может, используя открытый ключ, подобрать подпись под сообщение, а только под его хеш.
* Обеспечение совместимости. Большинство алгоритмов оперирует со строками бит данных, но некоторые используют другие представления. Хеш-функцию можно использовать для преобразования произвольного входного текста в подходящий формат.

*Схема цифровой подписи с использованием RSA:*

1. Генерация ключевой пары. Выбирается закрытый ключ, вычисляется соответствующий ему открытый ключ:
   1. Выбираются два различных случайных простых числа p и q заданного размера.
   2. Вычисляется их произведение n = pq, которое называется модулем.
   3. Вычисляется значение функции Эйлера от числа n:

φ(n) = (p -1)(q - 1).

* 1. Выбирается целое число e (1 < e < φ(n)), взаимно простое со значением функции φ(n) - открытая экспонента
  2. Вычисляется число d, обратное к числу e по модулю φ(n), то есть число, удовлетворяющее условию:  Число d называется секретной экспонентой.
  3. Пара e, n публикуется в качестве открытого ключа RSA.
  4. Пара d, n играет роль секретного ключа RSA.

1. Формирование подписи. Для заданного электронного документа с помощью закрытого ключа вычисляется подпись.
   1. Взять открытый текст 
   2. Создать цифровую подпись  с помощью своего секретного ключа : =h(M’)d(mod n)
   3. Передать пару , состоящую из сообщения и подписи.
2. Проверка (верификация) подписи. Для данных документа и подписи с помощью открытого ключа определяется действительность подписи.
   1. Принять пару 
   2. Взять открытый ключ  стороны A
   3. Проверить подлинность подписи: h' = e (mod n) h(M’) => подпись верная

n=p\*q; phi(n)=(p-1)(q-1); d=e-1mod(phi(n)).

ri=ri-2-qi-1\*ri-1

## Комп’ютерна стеганографія в системах безпеки інформаційних технологій

Стеганографія - це наука про приховану передачу інформації шляхом збереження в таємниці самого факту передачі.

На відміну від криптографії, яка приховує вміст секретного повідомлення, стеганографія приховує саме його існування. Стеганографію зазвичай використовують спільно з методами криптографії, таким чином, доповнюючи її.

● Використання зарезервованих полів комп'ютерних форматів файлів - суть методу полягає в тому, що частина поля розширень, що не заповнена інформацією про розширення, за замовчуванням заповнюється нулями. Відповідно ми можемо використовувати цю «нульову» частину для запису своїх даних. Недоліком цього методу є низький ступінь прихованості(скритності) і малий обсяг інформації, що передається.

● Метод приховування інформації в місцях гнучких дисків, що не використовуються - при використанні цього методу інформація записується в частини диска, що не використовуються, наприклад, на нульову доріжку. Недоліки: низька продуктивність, передача невеликих за обсягом повідомлень.

● Метод використання особливих властивостей полів форматів, які не відображаються на екрані - цей метод заснований на спеціальних «невидимих» полях для отримання зносок, покажчиків. Наприклад, написання чорним шрифтом на чорному тлі. Недоліки: низька продуктивність, невеликий обсяг інформації, що передається.

● Використання особливостей файлових систем - при зберіганні на жорсткому диску файл завжди займає ціле число кластерів (мінімальних обсягів інформації, що адресуються). Наприклад, в файловій системі FAT32 стандартний розмір кластера- 4 КБ. Відповідно для зберігання 1 КБ інформації на диску виділяється 4 КБ, з яких 1КБ потрібен для зберігання файлу, а решта 3 ні на що не використовуються - відповідно їх можна використовувати для зберігання інформації. Недолік даного методу: легкість виявлення.

●Цифрова стеганографія - напрям класичної стеганографії, заснований на приховуванні або впровадженні додаткової інформації в цифрові об'єкти, викликаючи при цьому деякі спотворення цих об'єктів. Але, як правило, дані об'єкти є мультимедіа-об'єктами (зображення, відео, аудіо, текстури 3D-об'єктів) та внесення спотворень, які знаходяться нижче порога чутливості середньостатистичної людини, не приводить до помітних змін цих об'єктів. Крім того, у відцифрованих об'єктах, що спочатку мали аналогову природу, завжди присутній шум квантування; далі, при відтворенні цих об'єктів з'являється додатковий аналоговий шум і нелінійні спотворення апаратури, все це сприяє кращому прихованню інформації.

## Аутентифікація об’єктів в системах безпеки інформаційних технологій

***Идентификация*** позволяет субъекту (пользователю, процессу, действующему от имени определенного пользователя, или иному аппаратно-программному компоненту) назвать себя (сообщить свое имя). Посредством ***аутентификации*** вторая сторона убеждается, что субъект действительно тот, за кого он себя выдает. В качестве синонима слова "*аутентификация*" иногда используют словосочетание "проверка подлинности"

Аутентификация бывает ***односторонней*** (обычно клиент доказывает свою подлинность серверу) и ***двусторонней*** (***взаимной***). Пример *односторонней аутентификации* - процедура входа пользователя в систему.

Выделяют следующие **методы аутентификации** пользователей:

-статические и одноразовые пароли;

-биометрия – это метод автоматизированного распознавания человека по его уникальным физиологическим или поведенческим характеристикам.

Вариантов биометрических методов более 600, среди них можно выделить следующие:

- по отпечаткам пальцев.

- по форме ладони – метод, основанный на уникальности трехмерной геометрии кисти руки;

- по сетчатке глаза – метод идентификации по рисунку кровеносных сосудов глазного дна;

- по радужной оболочке глаза. Уникальный для каждого человека рисунок радужной оболочки глаза сканируется камерой, в которую инсталлировано специальное программное обеспечение;

- по форме лица - определяется положение основных точек на лице (глаза, нос, губы), вычисляется расстояние между ними и конфигурируется множество вариантов на случай поворота или наклона головы, изменения выражения лица;

# Проектування інформаційних систем

## Основи проектування програмних комплексів ІС

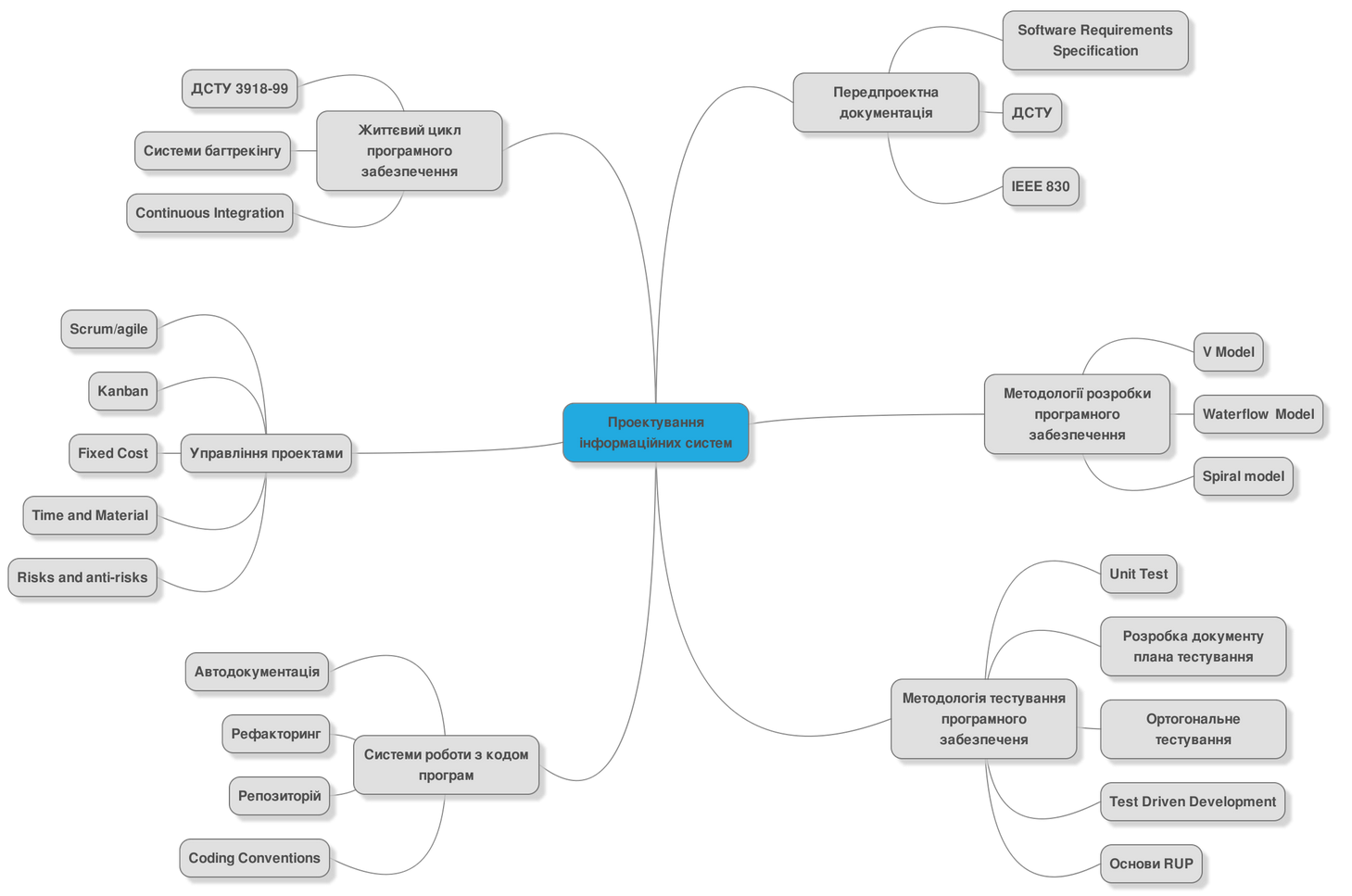
Інформаційна система (скор. ІС) - система обробки інформації і відповідні організаційні ресурси (людські, технічні, фінансові і т. д.), які забезпечують і поширюють інформацію.

* **автоматизовані**: інформаційні системи, в яких автоматизація може бути неповною (тобто потрібне постійне втручання персоналу);
* **автоматичні**: інформаційні системи, в яких автоматизація є повною, тобто втручання персоналу не потрібно або потрібно тільки епізодично.
* технічні засоби обробки даних
* програмне забезпечення
* відповідний персонал

За функціональним призначенням:

* Керувальні (АСКТП, АСКВ);
* Проектувальні (САП);
* Наукового пошуку (АСНД, експертні системи);
* Діагностичні, моделювальні;
* Систем підготовки прийняття рішення (СППР).

## Модель комплексу ІС



## Основні проблеми сучасних ІС та їх вирішення

*Чувствительность системы к неправильным действиям.*

Все бизнес-процессы должны быть идеально отлажены. Любая неправильная информация или ее отсутствие ведет к кардинальным ошибкам в работе системы и, как следствие, высокому риску принятия неверного решения.

*Проблемы управления.*

Отсутствие формализованной системы управления компанией. В частности:

отсутствие правил принятия решений

отсутствие способов контроля качества работы как предприятия в целом, так и его подразделений

отсутствие понятных инструментов для принятия управленческих решений

*Проблемы поддержки программного обеспечения*, автоматизирующего информационноую систему, связаны с:

техническими сбоями и поломками ПО

потребностью предприятия в обновлении версий ПО

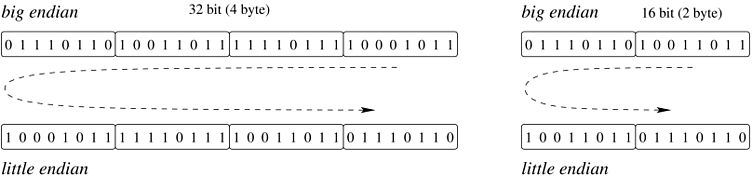
необходимостью постоянного обучения новых сотрудников – пользователей ПО

необходимостью регулярной модернизации ПО с целью более полного удовлетворения нужд принятой системы управления.

## Основні проблеми програмування сучасних ІС

Порядок байтів може змінюватися на різних типах процесорів

Розмір основних типів даних може змінюватися на різних аппаратних платформах



* Різні процесори можуть працювати і в порядку від молодшого до старшого, і в зворотному, наприклад, ARM, PowerPC (але не PowerPC 970), DEC Alpha, MIPS, PA-RISC і IA-64. Зазвичай порядок байтів вибирається програмно під час ініціалізації операційної системи, але може бути вибраний і апаратними перемичками на материнській платі. У цьому випадку правильніше говорити про порядок байтів операційної системи. Змінюваний порядок байтів іноді називають англ. bi-endian.
* Змішаний порядок байтів (англ. middle-endian ) іноді використовується при роботі з числами, довжина яких перевищує машинне слово. Число представляється послідовністю машинних слів, які записуються у форматі, природному для даної архітектури, але самі слова слідують у зворотному порядку.

Класичний приклад middle-endian - представлення 4-байтних цілих чисел на 16-бітних процесорах сімейства PDP- 11 ( відомий як PDP-endian ) . Для представлення двохбайтних значень (слів) використовувався порядок little-endian , але 4-х байтне подвійне слово записувалося від старшого слова до молодшого.

Для перетворення між мережним порядком байтів (англ. network byte order), який завжди у форматі big-endian, і порядком байтів, що використовується на машині (англ. host byte order), стандарт POSIX передбачає функції htonl (), htons (), ntohl () , ntohs ():

uint32\_t htonl (uint32\_t hostlong); - конвертує 32-бітну беззнакову величину з локального порядку байтів в мережевій;

uint16\_t htons (uint16\_t hostshort); - конвертує 16-бітну беззнакову величину з локального порядку байтів в мережевій;

uint32\_t ntohl (uint32\_t netlong); - конвертує 32-бітну беззнакову величину з мережевого порядку байтів в локальний;

uint16\_t ntohs (uint16\_t netshort); - конвертує 16-бітну беззнакову величину з мережевого порядку байтів в локальний.

Різні операційні системи – різні системні виклики.

## Огляд методів та засобів програмування ІС. Application Programm Interface

An application programming interface (API) це інтерфейс на рівні вихідних кодів, який може бути реалізований комп'ютерною системою чи бібліотекою програмування для підтримки запитів від іншої комп'ютерної системи чи бібліотеки.

Application Binary Interface відрізняється від API наступним чином: API визначає інтерфейс між програмним кодом і бібліотекою, тобто одинакові вихідні тексти будуть компілюватися на будь-якій системі, що підтримує API.

ABI дозволяє використовувати вже відкомпільований бінарный код (об'єктний код), який забезпечує доступ до функцій бібліотеки тільки в рамках визначеної апаратної платформи.

### Основні принципи розробки API

1) Do One Thing

2) Design by Sample Re-coding

3) Evolve By Subtraction (розвивайтесь шляхом вирахування, наприклад, заснованому на відгуках користувачів інтерфейсу).

4) Declarative Beats Imperative. Императивное программирование описывает процесс вычисления в виде заданной последовательности инструкций, изменяющих состояние программы. Декларативное программирование описывает процесс каково нечто, а не как его создать. Порядок выполнения и способ достижения не важен. Базовой операцией при декларативном подходе является операция над множеством или над всеми элементами множества.

5) Preserve the Users Metadata

6) Build on the Familiar

7) The Best Code is No Code At All

## Розробка технічного завдання до типової інформаційної системи. Оформлення технічного завдання згідно ГОСТ 19.201-78, ГОСТ 34-602-89 таінших

В ТЗ більше букв ніж в SRSОсновна мета ТЗ — врегулювати технічні відносини між замовниками та виконавцями.

19.201-78:

Правила оформлення

Лист затвердження і титульний лист

Зміни та доповнення

Склад розділів технічного завдання

Зміст розділів

Вступ

Підстави для розробки

Призначення розробки

Вимоги до програми або програмного виробу

Вимоги до програмної документації

Техніко-економічні показники

Стадії та етапи розробки

Порядок контролю та приймання

Додатки

Висновки

Розробниками стандарту передбачений розділ «Спеціальні вимоги», можливість додавання нових розділів(припустимо, розділів «Додаткові вимоги» або «Вимоги до інтерфейсу»).

Не можна захоплюватися «тематичними» ГОСТ, містять конкретні вимоги до компонентів системи.

Характерна помилка початківців - «канали зв'язку повинні задовольняти вимогам ГОСТ-у такому-то». Відомо, що прийманню-здачі робіт по створенню системи, виробів, програмного продукту завжди передує проведення випробувань.

## Архітектурні рішення побудови ІС

Архитектура ИС – концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы.



Слой представления – взаимодействие с пользователем

Бизнес-логика – правила обработки даннях.

Слой доступа к данным – хранение, выборка, модификация и удаление даннях

Классификация архитектур:

По степени распределенности:

Настольные(desktop) – все данные (БД, СУБД, клиентские приложения) хранятся на одном компьютере

Распределенные (distributed) – компоненты распределены по нескольким компьютерам

Файл-серверные

Клиент-серверные

Двузвенные

Многозвенные

### Файл-серверная архітектура

Плюсы:

• Многопользовательский режим работы с данными

• Централизованное управление доступом

• Низкая стоимость и высокая скорость разработки

Минусы:

• Низкая производительность; сильная перегрузка ЛВС

• Низкая надежность

• Слабая возможность расширения

### **Клиент-серверная архитектура**

Плюсы:

• Многопользовательский режим работы

• Гарантия целостности данных

Минусы:

• Бизнес-логика на клиенте, сложности изменения алгоритмов

• Слабая защита данных от взлома

• Высокие требования к пропускной способности, клиентским машинам

• Высокая сложность администрирования и разработки

## Основи UML. Використання діаграми класів та діаграми прецедентів у проектуванні ІС

Див. [Уніфікована мова моделювання (UML), значення UML в процесі проектування програмного забезпечення](#_Уніфікована_мова_моделювання)

## Вимоги до побудови архітектурно-програмної моделі ІС

Див. [Інженерія вимог до програмних систем](#_Інженерія_вимог_до)

## Загальні підходи до побудови архітектури ІС

Проектування архітектури програмного забезпечення — це процес розроблення, що виконується після етапу аналізу і формування вимог. Задача такого проектування — перетворення вимог до системи у вимоги до ПЗ і побудова на їхній основі архітектури системи.

Архітектура системи — це структурна схема компонентів системи, взаємодіючих між собою через інтерфейси. Компоненти можуть складатися з послідовності більш дрібних компонентів та інтерфейсів.

Побудова архітектури системи здійснюється шляхом визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури.

Архітектуру системи визначають також як множину представлень, кожне з яких відбиває деякий аспект, що цікавить групу учасників проекту — аналітиків, проектувальників, кінцевих користувачів та ін. Представлення фіксують проектні рішення щодо структури і поділу ІС на окремі компоненти та визначення зв'язків між ними. На ці рішення впливають вимоги до функцій і середовища.

Проектування архітектури системи може проводитися різними методами (об'єктно-орієнтованим, компонентним і ін.), кожний з яких пропонує свій шлях побудови архітектури, а саме, визначення концептуальної, об'єктної й інших моделей за допомогою відповідних конструктивних елементів (блок-схем, графів, структурних діаграм тощо).

При застосуванні об'єктно-орієнтованого підходу компонентами є окремі об'єкти, а процес конструювання об'єктної структури перетворюється в процес виявлення наявних у предметній області об'єктів і визначення сценарію їхнього виконання і взаємодії.

## Методології розробки програмного забезпечення ІС

Методология разработки программного обеспечения — совокупность методов, применяемых на различных стадиях жизненного цикла программного обеспечения и имеющих общий философский подход.

### Agile

Основополагающие принципы Agile-манифеста:

1) Наивысшим приоритетом для нас является удовлетворение потребностей заказчика, благодаря регулярной и ранней поставке программного обеспечения.

2) Изменение требований приветствуется, даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.

3) Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.

4) На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.

5) Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.

6) Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.

7) Работающий продукт – основной показатель прогресса.

8) Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно.

9) Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.

10) Простота – искусство минимизации лишней работы – крайне необходима.

11) Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.

12) Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

Люди и взаимодействие важнее процессов и инструментов

Работающий продукт важнее исчерпывающей документации

Сотрудничество с заказчиком важнее согласования условий контракта

Готовность к изменениям важнее следования первоначальному плану

Переваги в тому, що він є економічно ефективним і економить багато часу для програмного забезпечення розробки продуктів та послуг. Його мінус в тому, що є вискоий ризик неуспішного результату.

### Scrum

В скрам используется *четыре артефакта*:

Product Backlog. Это список всех требований, которые нужно сделать по проекту. Когда в Backlog’e нет требований, проект считается завершенным.

Sprint Backlog. Это список всех требований, которые нужно сделать в ближайший спринт. В течение спринта, новые требования не могут появится в Sprint backlog.

Sprint Goal. Это краткое описание того, ради чего выполняется данный спринт.

Sprint Burndown Chart. Диаграмма обновляется каждый раз, когда завершается какая-либо задача.

В скрам используется *три роли*:

Product Owner. Product Owner — это представитель подразделения, которое владеет разрабатываемым продуктом. Несет персональную ответственность за ценность требований для рынка/пользователей

Scrum Master. Организует работу команды и обеспечивает её всем необходимым. Скрам мастер не назначает людей на задачи — это делает сама команда.

Team. 4-10 человек. Взаимозаменяемая. С фиксированным составом (в ходе спринта).

В скрам *четыре ритуала*:

Sprint Planning Meeting. Команда выбирает требования из Product Backlog и формирует Sprint Backlog. Встреча, которая проводится перед началом каждого спринта.

Daily Meeting. встреча проходит только стоя, поэтому длительность встречи не более 15 минут;

Каждый должен ответить всего на 3 вопроса: что я делал вчера, чем я занимаюсь сегодня, какие есть проблемы.

Sprint Review. По завершению каждого спринта команда обязана провести демонстрацию полученного результата.

Retrospective. Ритуал, который направлен на обмен опытом внутри команды. Встреча проводится после Sprint Review.

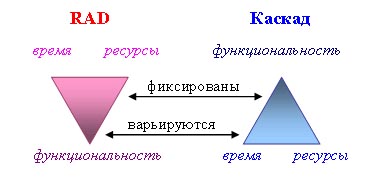
### RAD

Технологию RAD целесообразно применять, когда четко определены некоторые приоритетные направления разработки проекта.

1. Необходимо выполнение проекта в сжатые сроки. Быстрое выполнение проекта позволяет создать систему, отвечающую требованиям сегодняшнего дня. Если система проектируется долго, то весьма высока вероятность, что за это время существенно изменятся фундаментальные положения, регламентирующие деятельность организации, то есть, система морально устареет ещё до завершения её проектирования.

2. Нечетко определены требования к ПО. В большинстве случаев заказчик весьма приблизительно представляет себе работу будущего программного продукта и не может четко сформулировать все требования к ПО. Требования могут быть вообще не определены к началу проекта либо могут изменяться по ходу его выполнения.

3. Проект выполняется в условиях ограниченности бюджета. Разработка ведётся небольшими RAD-группами в короткие сроки, что обеспечивает минимум трудозатрат и позволяет вписаться в бюджетные ограничения.

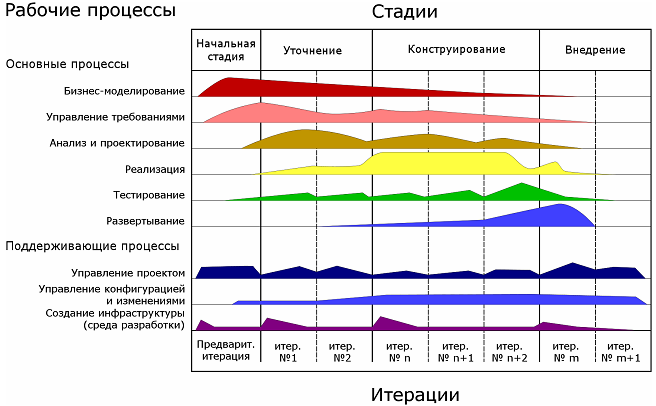


Технология *быстрой разработки приложений* (RAD) позволяет обеспечить:

* быстроту продвижения программного продукта на рынок;
* интерфейс, устраивающий пользователя;
* лёгкую адаптируемость проекта к изменяющимся требованиям;
* простоту развития функциональности системы.

### RUP

RUP использует итеративную модель разработки. В конце каждой итерации проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать проектные артефакты и получить промежуточную, но функциональную версию конечного продукта.



Достоинства RUP:

- можно использовать как основу для водопадного стиля разработки, так и в качестве гибкого процесса;

- снижение основных рисков заказчика и разработчика;

- улучшение качества ПО за счет многократных проверок изменений;

Недостатком является недостаточный уровень формализма, который может приводить к несогласованности решений, принимаемых участниками проекта, к непродуктивным затратам ресурсов на переработку кода и на повторное решение типовых проблем.

### Test Driven Development

Розробка через тестування – процес розробки програмного забезпечення, що передбачає написання й автоматизацію модульних тестів до момента написання відповідних класів або модулів. Це гарантує, що всі обов'язки будь-якого елемента програмного забезпечення визначаються ще до того, як вони будуть закодовані.

Понаття:

1) Мок-об'єкти (MockObjects, or Dummy) – заглушки, що автоматично генеруються, які можуть виступати у ролі реальних об'єктів. Поведінкою моків можна керувати безпосередньо в тесті. Моки можуть виконувати додаткові перевірки.

2) Модульний тест – тест, котрий перевіряє пведінку невеликої частини додатка. Ця частина може бути одним класом, одним методом або набором класів, що реалізують якісь архітектурні розв'язки, і це рішення необхідно перевірити на працездатність.

3) Тест TestCase – набір тестових методів, призначених для тестування одного класа. Зазвичай TestCase складається з методів, чиє ім'я починається з приставки test. Кожен такий метод тестує будь-який один момент класа, що тестується. В приймальному тестуванні TestCase – це набір команд, які тестують одну значущу для замовника функціональність.

4) Набір тестів TestSuite – набір тестів, призначений для тестування будь-якої збільшеної сутності програмної системи. Іноді TestSuite використовують у значенні “всі тести, які є для додатка”.

5) Фікстура Fixture – стан середовища тестування, який потрібен для успішного виконання тестового методу. Це може бути набір будь-яких об'єктів, стан бази даних, наявність певних файлів и т.д. Фікстура створюється в методі setUp() перед кожним викликом методу виду testSomethng() тесту (TestCase) і видаляється в testDown() після закінчення виконання тестового методу.

6) Перевірка Assert метод класу який призначена для звірення реального стану коду, що тестується, з очікуваним.

При Автоматичних тестах замість програміста стимулюванням коду і перевіркою результатів займається комп'ютер, який відображає на екрані результат виконання тесту: код працездатний або код непрацездатний.

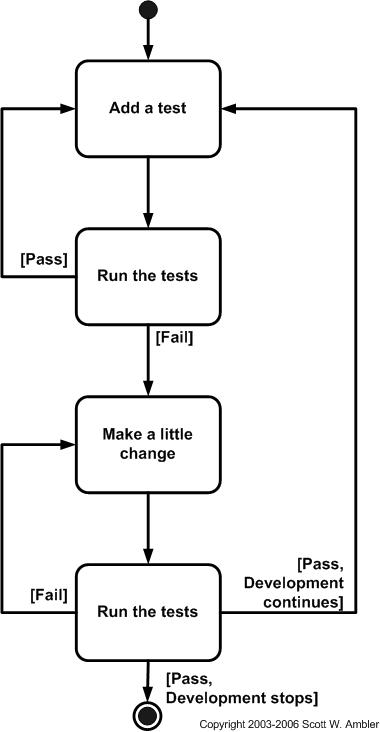
Процедура розробки: пишемо новий код тоді, коли автоматично код не спрацював; видаляємо дублювання.

Порядок етапів програмування:

1) Червоний – напишіть невеликий тест, який не працює, а можливо, навіть не компілюється.

2) Зелений – змусьте тест працювати як можна швидше, при цьому не думайте про правильності дизайну і чистоті коду. Напишіть рівно стільки коду, щоб тест спрацював.

3) Рефакторинг – видаліть з написаного вами коду будь-яке дублювання.



## Waterflow та Model Driven Development

Див. Каскадная модель в [Життєві цикли програмного забезпечення систем обробки інформації](#_Життєві_цикли_програмного)

Разработка, управляемая моделями, (англ. model-driven development) — это стиль разработки программного обеспечения, когда модели становятся основными артефактами разработки, из которых генерируется код и другие артефакты.

Модель — это абстрактное описание программного обеспечения, которое скрывает информацию о некоторых аспектах с целью представления упрощенного описания остальных. Модель может быть исходным артефактом в разработке, если она фиксирует информацию в форме, пригодной для интерпретаций людьми и обработки инструментами. Модель определяет нотацию и метамодель. Нотация представляет собой совокупность графических элементов, которые применяются в модели и могут быть интерпретированы людьми. Метамодель описывает понятия используемые в модели и фиксирует информацию в виде метаданных, которые могут быть обработаны инструментами.

## Основи проектного менеджменту розробки програмного забезпечення

Проектне управління – дисципліна по плануванню, організації і управлінню ресурсами для успішного управління конкретними цілями та задачами.

Проект – це унікальна (на відміну від операцій) діяльність, що має початок і кінець у часі, направлена на досягнення певного результату/цілі, створення певного унікального продукту чи послуги при заданих обмеженнях по ресурсам та срокам, а також вимогам до якості і допустимого рівня риску.

Ознаки того, що діяльність не є проектом:

Якщо ціль з самого початку не визначена або сильно міняється у процесі виконання робіт;

Якщо діяльність не є керованою, тобто зовнішні залежності перебільшуюсть можливості впливу на діяльність;

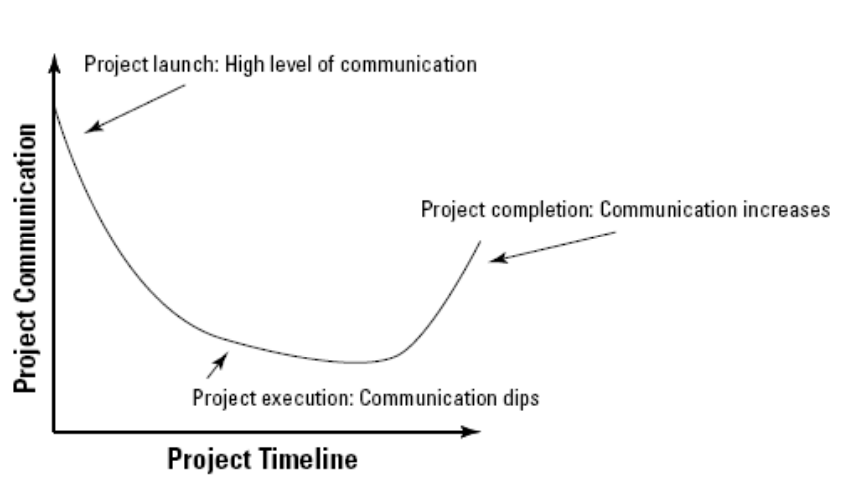
Якщо обмеження діяльності з самого почтаку не задані (строки, ресурси, час, якість, допустимий рівень ризику);

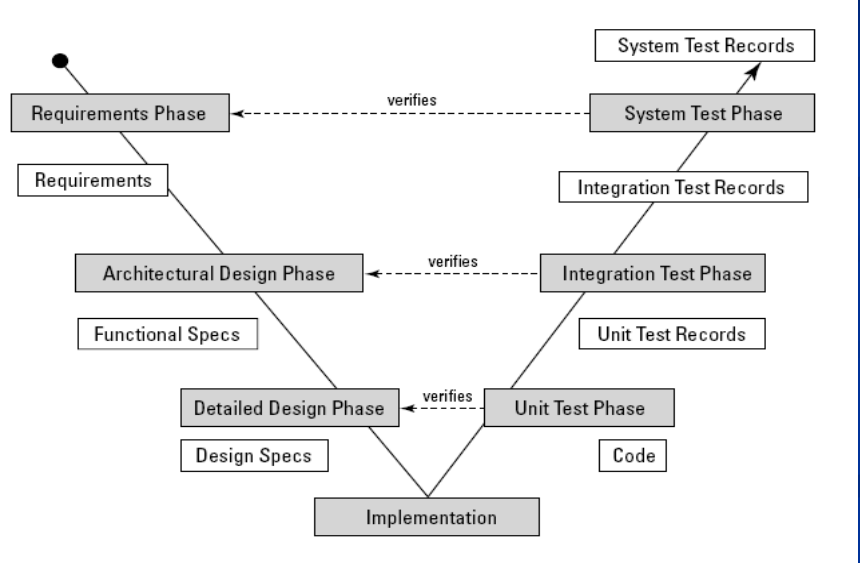
Якщо результат не унікальний (наприклад, приготування обіду кожного дня).

### Планування проекту

* Створення технічного завдання
* Визначення етапів проекту
* Визначення цілей по завершенню кожного етапу проекту
* Визначення доступних ресурсів (людські, обладнання, бюджет, час)
* Створення діаграми Ганта
* Визначення ризиків по кожному етапу проекту





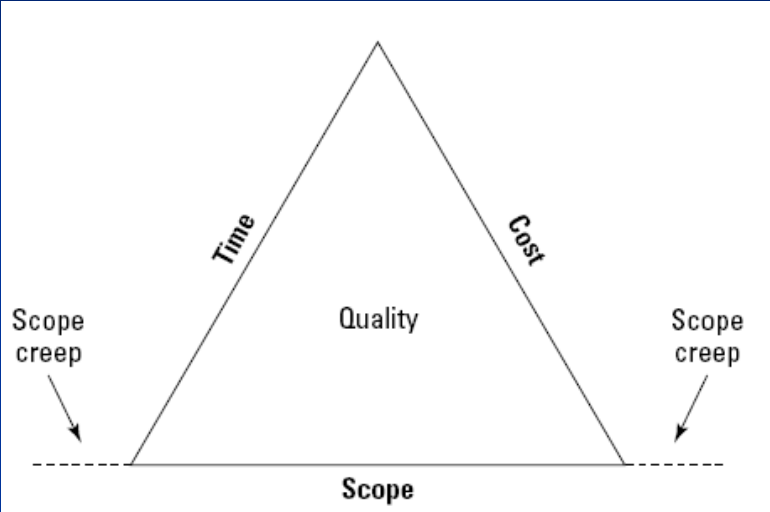


## Методика оцінки ризиків

Існують декілька видів аутсортінгу:

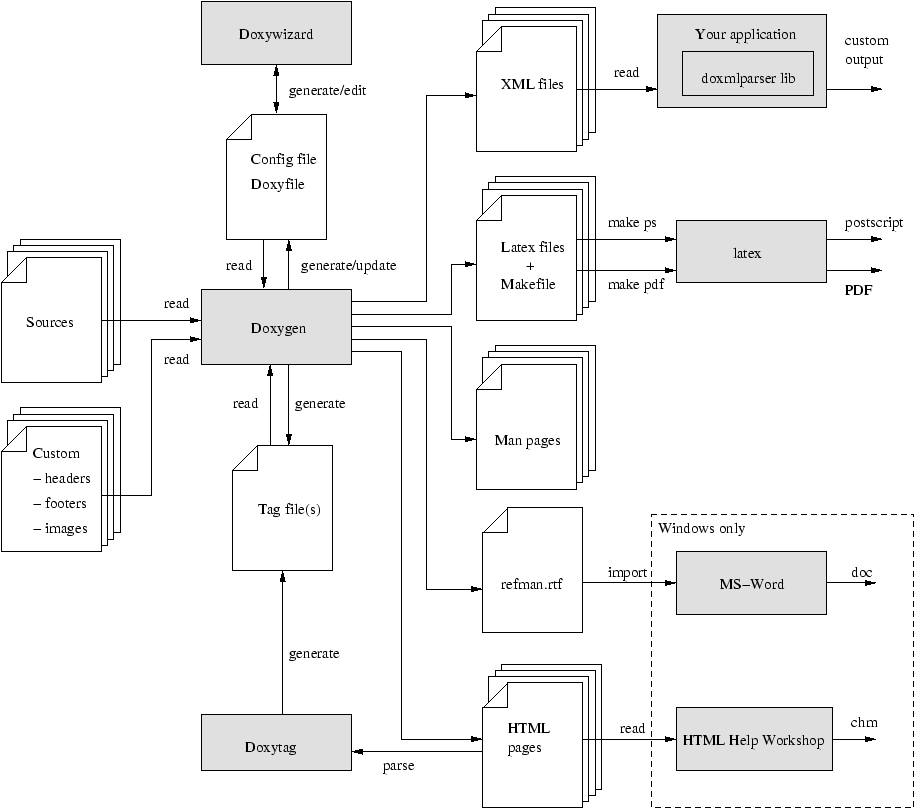
* Повний аутсортінг, включаючи технічне завдання. Найвищий ризик, в якому доля проекту повністю переноситься на виконавця.
* Частковий аутсортінг (спеціалізовані модулі проекту). Меньший ступінь ризику, так як модулі можуть бути швидко передані на розробку в інші організації та розмір збитків меньший, ніж в повному аутсортінгу.
* Team Extention Model (використання контрактних розробників). Ризик мінімальний, так як контрактні розробники можуть використовуватися на загальних умовах з усіма учасниками проекту

Найбільш небезпечний вид ризику – зміна цілей проекту.



## Розробка документації Reference Manual на Doxygen

Автоматичне документування програмного коду.



/\*!

\* ... первая строчка ...

\* ... вторая строчка ...

\*/

/\*!

\brief Краткое описание и

его продолжение.

Подробное описание

\*/

//! Краткое описание

//! \details Подробное описание

Во всех предыдущих примерах подразумевалось, что документирующий блок предварял документируемый элемент, но иногда бывают ситуации, когда удобнее разместить его после документируемого элемента. Для этого необходимо в блок добавить маркер "<":

int variable; ///< Краткое описание

Любая команда в Doxygen представляет собой слово на английском языке предваренное символом "\" или "@" (обе записи тождественны). Приведём для примера несколько таких команд:

Команда Значение

\authors Указывает автора или авторов

\version Используется для указания версии

\date Предназначена для указания даты разработки

\bug Перечисление известных ошибок

\warning Предупреждение для использования

\copyright Используемая лицензия

\example Команда, добавляемая в комментарий для указания ссылки на исходник с примером (добавляется после команды)

\todo Команда, используется для описания тех изменений, которые необходимо будет сделать (TODO).

\struct to document a C-struct.

\union to document a union.

\enum to document an enumeration type.

\fn to document a function.

\var to document a variable or typedef or enum value.

\def to document a #define.

\typedef to document a type definition.

\file to document a file.

\namespace to document a namespace.

\package to document a Java package.

\interface to document an IDL interface.

\sa see also

## Оформлення коду згідно Coding Convention

Стандарт оформления кода обычно принимается и используется некоторой группой разработчиков программного обеспечения для единообразного оформления совместно используемого кода. Целью принятия и использования стандарта является упрощение восприятия программного кода человеком, минимизация нагрузки на память и зрение при чтении программы. Образцом для стандарта кодирования может стать набор соглашений, принятых в какой-либо распространённой печатной работе по языку (например, стандарт кодирования на языке Си, получивший сокращённое наименование K&R, происходит из классического описания Си его авторами — Керниганом и Ритчи), широко применяемая библиотека или API.

## Колективна робота над проектом з системами багтрекінгу

Система отслеживания ошибок (англ. bug tracking system) – прикладная программа, разработанная с целью помочь разработчикам программного обеспечения (программистам, тестировщикам и др.) учитывать и контролировать ошибки и неполадки, найденные в программах, пожелания пользователей, а также следить за процессом устранения этих ошибок и выполнения или невыполнения пожеланий. Главный компонент такой системы — база данных, содержащая сведения об обнаруженных дефектах. Эти сведения могут включать в себя: номер (идентификатор) дефекта; короткое описание дефекта; кто сообщил о дефекте; дата и время, когда был обнаружен дефект; версия продукта, в которой обнаружен дефект; серьёзность (критичность) дефекта и приоритет решения; описание шагов для выявления дефекта (воспроизведения неправильного поведения программы); ожидаемый результат и фактический результат; кто ответственен за устранение дефекта; обсуждение возможных решений и их последствий; текущее состояние (статус) дефекта; версия продукта, в которой дефект исправлен.

Типичный жизненный цикл дефекта:

новый – дефект зарегистрирован тестировщиком

назначен – назначен ответственный за исправление дефекта

разрешён – дефект переходит обратно в сферу ответственности тестировщика. Как правило, сопровождается резолюцией, например:

*исправлено* (исправления включены в версию такую-то)

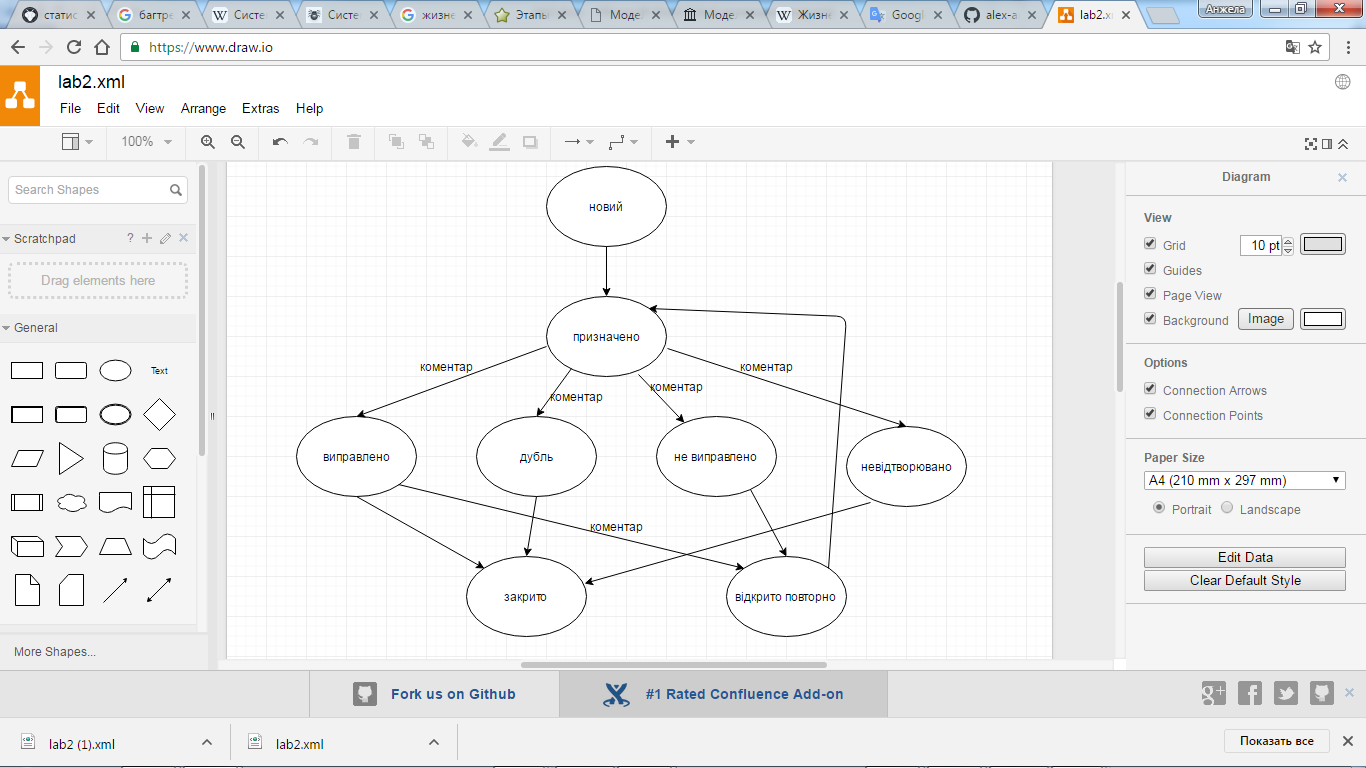
*дубль* (повторяет дефект, уже находящийся в работе)

*не исправлено* (работает в соответствии со спецификацией, имеет слишком низкий приоритет, исправление отложено до следующей версии и т.п.)

*невоспроизводимо* (запрос дополнительной информации об условиях, в которых дефект проявляется).

далее тестировщик проводит проверку исправления, в зависимости от чего дефект либо снова переходит в статус назначен (если он описан как исправленный, но не исправлен), либо в статус закрыт.

открыт повторно – дефект вновь найден в другой версии.



## Колективна робота над проектом ІС в CVS, SVN, GIT

За допомогою систем керування версіями ви стежите за змінами коду вашого програмного продукту в ході його розробки, і можете керувати його різними станами. Версією називають набір вихідних кодів програми та відповідних автоматичних інструкцій по збірці, яка дозволяє отримати працездатний продукт, який успішно компілюється, встановлюється і запускається в роботу. Атомарною структурою в системі управління версією э рядок.

Для роботи в системы контролю версій необхідно:

1) Створити репозиторій

2) Помістити файли в репозиторій

3) Записати зміни в репозиторій

4) Взяти файли з репозиторію

5) Перевірити останні зміни

6) Робота з гілками

git init

Инициализация локального репозитория в рабочей директории. При инициализации репозитория создается скрытая папка .git. В ней будут хранится рабочие файлы СКВ. Там можно будет найти логи, если они будут необходимы.

>> git checkout -b new\_branch

создание новой ветки и переход на нее, является обьеденением двух команд:

>> git branch new\_branch

>> git checkout new\_branch

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда git status. Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда git add. Простейший способ зафиксировать изменения – это набрать git commit.

Для того чтобы удалить файл из Git'а, вам необходимо удалить его из отслеживаемых файлов (точнее, удалить его из вашего индекса) а затем выполнить коммит. Это позволяет сделать команда git rm, которая также удаляет файл из вашего рабочего каталога.

Одна из типичных отмен происходит тогда, когда вы делаете коммит слишком рано, забыв добавить какие-то файлы, или напутали с комментарием к коммиту. Если вам хотелось бы сделать этот коммит ещё раз, вы можете выполнить commit с опцией – amend. Use "git checkout -- <file>..." to discard changes in working directory.

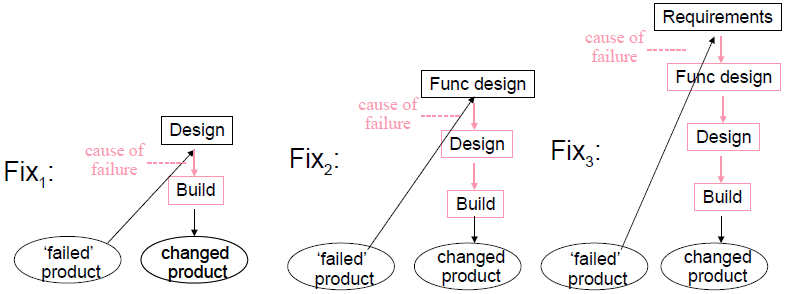
Слить (merge) изменения назад в ветку master:

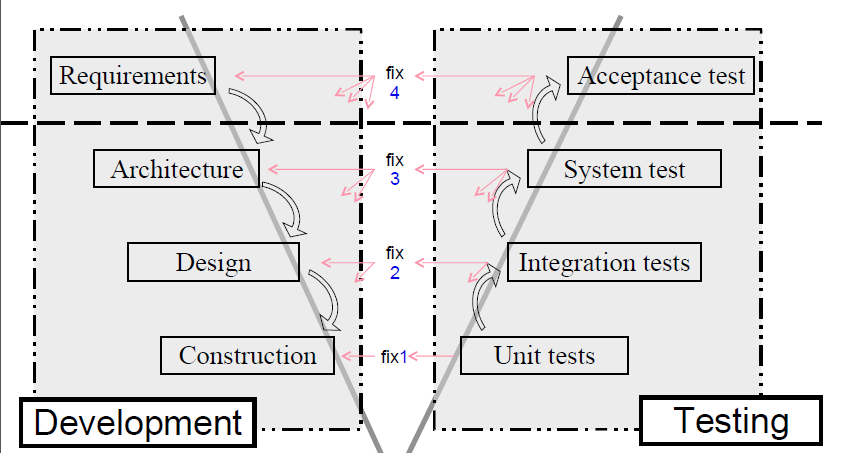
$ git checkout master

$ git merge hotfix

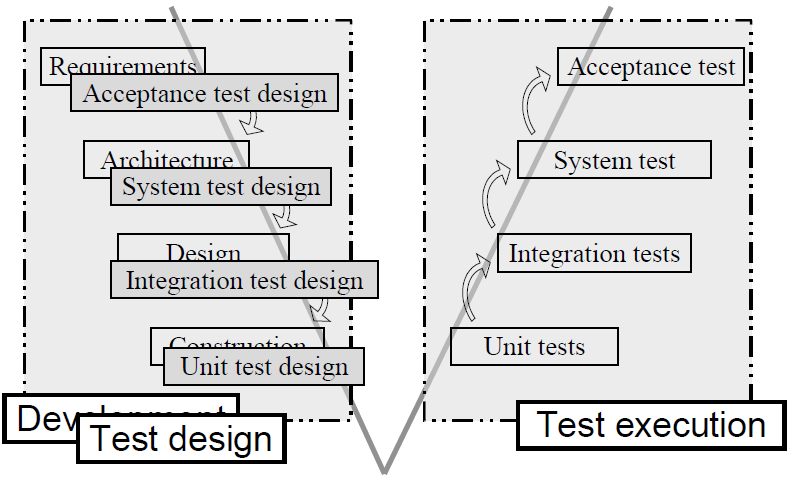
## Тестування програмного забезпечення згідно V-моделі

* Fix1 - вада в коді програми; переробляється тільки код
* Fix2 - вада в специфікаціях (технічний дизайн); змінюються і специфікації, і код
* Fix3 - вада в архітектурі / функціональному дизайні; міняються архітектура, специфікації і код





Тест-проектування у V-моделі:



## Основи RUP. Тестування програмного забезпечення згідно RUP

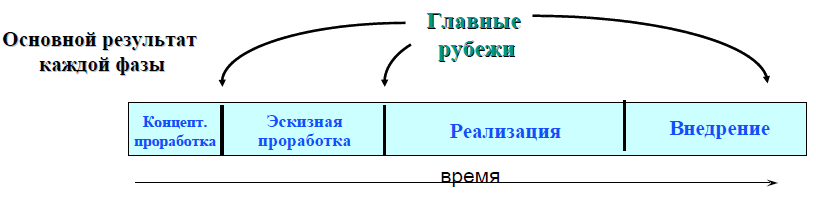
Див. [Методології розробки програмного забезпечення ІС](#_Методології_розробки_програмного) ([RUP](#_RUP)).

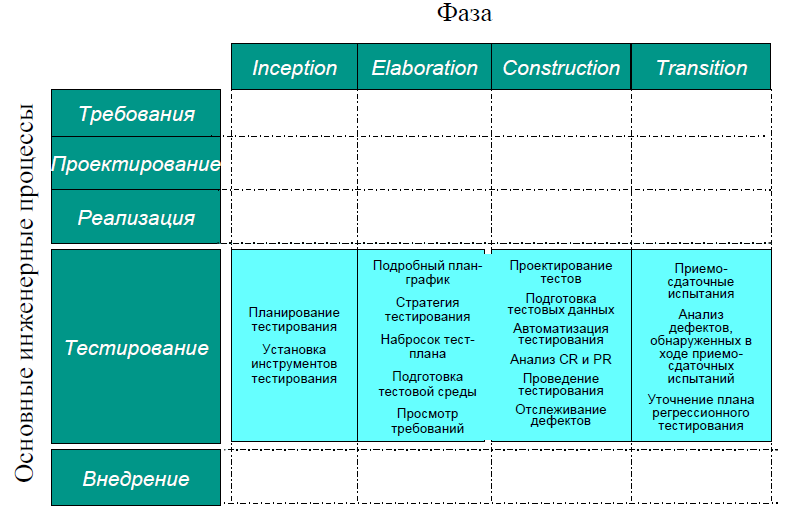
Дві основні стадії ЖЦ:

Інженерна (дослідницька). Завершується, коли є чітке розуміння, ЯК робити продукт (архітектура)

Виробнича. Складається головним чином у виготовленні продукту

Рубіж між стадіями - головний рубіж проекту. Співвідношення термінів і ресурсів між стадіями визначається специфікою проекту.





## Рефакторінг та використання методів зміни та покращення програмного коду

Рефакторінг (англ. refactoring) — процес зміни внутрішньої структури програми, який не зачіпає її зовнішньої поведінки і його ціллю є полегшення розуміння її роботи. В основі рефакторінгу лежить послідовність невеликих еквівалентних (тобто зберігаючи поведінку) перетворень.

Ціль рефакторінгу – зробити код програми більш зрозумілим для людини. Рефакторінг слід відрізняти від оптимізації продуктивності. Як і рефакторінг, оптимізація зазвичай не змінює поведінку програми, а тільки пришвидшує її роботу. З іншої сторони оптимізація часто ускладнює розуміння коду, що є протилежним рефакторінгу. Форматування коду рефакторінгом не є.

Можна виділити найбільш очевидні причини, коли код потрібно піддати рефакторінгу: дублювання коду; довгий метод; великий клас; довгий список параметрів; «заздрісні» функції – метод, який надмірно звертається до даних іншого об'єкту; надлишкові тимчасові змінні; класи даних; незгрупованні дані.

Виділяють наступні методи рефактоиингу:

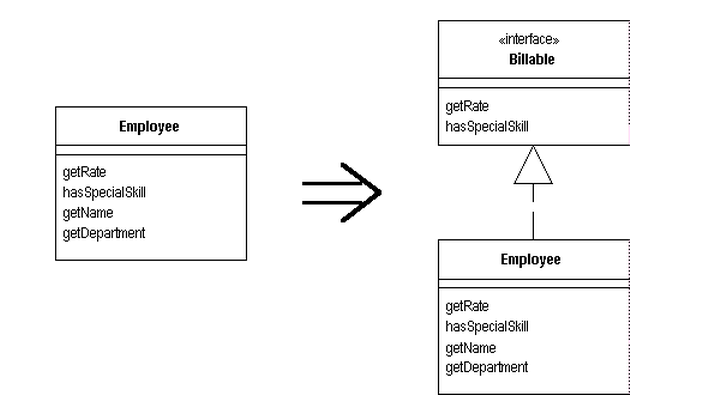
Зміна сигнатури методу (Change Method Signature). Суть зміни сигнатури методу полягає в додаванні, зміні або видаленні параметра методу. Змінивши сигнатуру методу, необхідно скорегувати звернення до нього в коді всіх клієнтів.

Інкапсуляція поля (Encapsulate Field). У випадку, якщо в класу є відкрите поле, необхідно зробити його закритим і забезпечити методи доступу. Після «інкапсуляції поля» часто застосовується «Переміщення методу».

Переміщення методу (Move Method). Переміщення методу застосовується по відношенню до методу, який частіше звертається до іншого класу, ніж до того, в якому сам розташовується.

Виділення класу (Extract Class). Створення нового класу і переміщення відповідних полів і методів зі старого класу в новий клас.

Виділення інтерфейсу (Extract Interface). Деякі клієнти використовують одну і ту ж підмножину інтерфейсу класу або два класи, які мають загальний інтерфейс.



Виділення локальної змінної (Extract Local Variable)

Виділення методу (Extract Method). Виділення методу полягає у виділенні з довгого і/або вимагаючого коментарів коду окремих фрагментів і перетворенні їх в окремі методи, з підстановкою відповідних викликів у місцях використання.

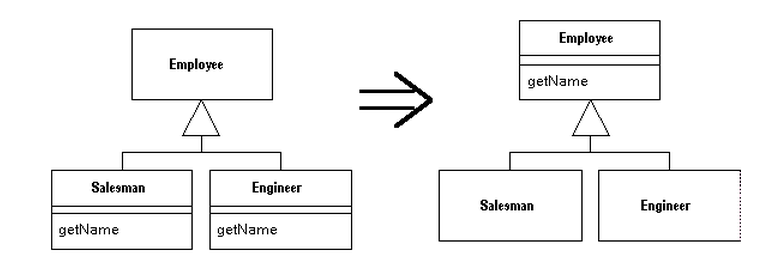
Генералізація типу (Generalize Type)

Вбудовування (Inline)

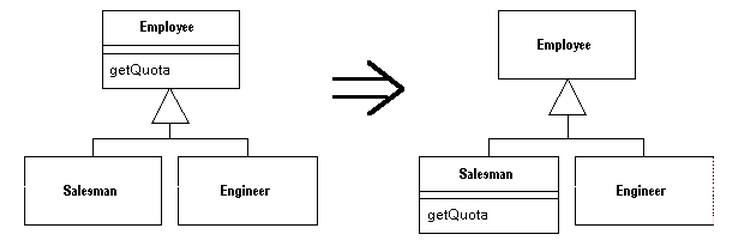
Введення фабрики (Introduce Factory)

Введення параметра (Introduce Parameter)

Підйом поля / методу (Pull Up). Переміщення поля або методу в суперклас.



Спуск поля / методу (Push Down). Переміщення поля методу з суперкласу.



Заміна умовного оператора поліморфізмом (Replace Conditional with Polymorphism). Умовний оператор з декількома гілками замінюється викликом поліморфного методу деякого базового класу, що має підкласи для кожної гілки вихідного оператора. Вибір гілки здійснюється неявно, в залежності від того, екземпляру якого з підкласів виявився адресований виклик.

Проблемы, возникающие при проведении рефакторинга: проблемы, связанные с базами данных; проблемы изменения интерфейсов; трудности при изменении дизайна.