НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“Київський політехнічний інститут”
Факультет електроніки
Кафедра систем автоматизованого проектування

Unknown 12/11/2002 16:44
**Примечание [1]:**

**Інструктивно - методичні матеріали**

**Щодо лекцій**

**з дисципліни ”Технології програмування та створення програмних продуктів”**

**для студентів напряму підготовки 6.0804**

**“комп’ютерні науки”**

**спеціальності – 7.080402 – інформаційні технології проектування.**

**Методичні вказівки знаходяться в електронному вигляді на сервереу кафедри САПР:**

**\\DRAGON2\Techmat\kiselev\TP\KONSPECT_TP.doc**

**Затверджено**
**на засіданні кафедри**
**систем автоматизованого проектування**
**Протокол № 1 від 30.08.2005р.**
**Зав. кафедри**
**Петренко А. І**

**Київ 2005р.**

**Викладач : Кисельов Г. Д .**

ЗМІСТ



## АВТОМАТИЗОВАНЕ ПРОГРАМУВАННЯ .

Під технологією програмування розуміють сукупність систематизованих та узагальнених знань ( наука ) про оптимальні засоби та прийоми проведення процесу програмування . Процес програмування повинен забезпечувати при заданих умовах отримання продукції із заданими властивостями . Технологія програмування визначає професійну культуру програміста і людей . які забезпечують створення програмного продукту.

Продуктивність праці - фактор , заради якого розвивається технологія програмування .

Технологія програмування охоплює зміст процесу програмування від появи потреб у створенні деякої програми до її виготовлення, і далі - до припинення користування нею в результаті морального старіння.

Аналіз відомих і широко поширених технологій та методів програмування дозволяє сформулювати такі основні вимоги :

1 . Технологія програмування повинна забезпечувати відторгнення програмного виробу від його розробника ( тобто людський фактор повинен бути зведений до мінімуму ) .

2 . Технологія програмування та засоби її підтримки повинні підтримувати цільоспрямовану працю колективу розробників - це означає , що інструментальні засоби підтримки технології програмування повинні блокувати будь-якиі несанкціоновані дії , більш того , програмні апаратні засоби підтримки повинні включати в себе засоби мережевого планування та експлуатації .

3 . Технологія програмування повинна бути "безпаперовою" , тобто весь процес виготовлення програмного виробу та управління колективом програмістів повинен бути максимально механізованим .

4 . Інструментні засоби підтримки технології повинні охоплювати всі етапи роботи колективу програмістів .

5. Технологія програмування не повинна жорстко прив'язуватись до мови програмування . За сучасними уявленнями мова програмування не є головною керую-чою ланкою в програмуванні .

6 . Технологія програмування повинна буті простою за побудовою , інструментарій повинен містити у собі засоби підказок і навчання , крім того , основою повинен бути метод контекстної допомоги розробників програмного продукту .

7 . Технологія програмування повинна фіксувати в хронологічному порядку всі дії колективу програмістів .

Сучасні вимоги примушують програмістів основний час розробки приділяти не створенню алгоритму , не структуруванню данних , а створенню інтерфейсу користувача .

Під програмним продуктом розуміють вироб , представлений в формі машинних команд тої чи іншої ЕОМ , створений колективом програмістів , і який має споживну вартість . Програмний продукт , на відміну від програми , передається службовими повідомленнями і експлуатується без участі розробника . Цикл життя будь-якого програмного продукту складається з

- аналізу ;
- розробки ;
- супроводження .

На етапі **аналізу** виконуються :

1) Розробка ТЗ; 2) Розробка технічних пропозицій; 3) Розробка ескізного проекту; 4) Розробка технічного проекту .

Етап **розробки** містить в собі розробку робочого проекту; робочий проект закінчується іспитами і тестуванням .

На етапі **супроводження** виконуються впровадження , модифікація і безпосередньо супроводження .

Розглянемо кожний етап аналізу окремо .

1 . **Технічне завдання (ТЗ):** розробляється організацією замовників і узгоджується з виконавцем ( в процесі створення продукту можливі зміни ТЗ ) . В ТЗ виділяють такі розділи :

1) Призначення технічного продукту і його основні функції ;

2) Перелік техніко-економічних вимог, які містять очікувану економічну ефективність , вартість розробки , терміни розробки ;

3) Склад і характеристики потоку вхідної та вихідної інформації ( вимоги до інтерфейсу користувача ) ;

4) Вимоги до технічного і програмного забезпечення ;

5) Функціональні характеристики програмного продукту ;

6) Пристосовність до внесення доробок ;

7) Склад конструкторських і експлуатаційних документацій і вимоги щодо їх оформлення .

2 . **Технічні умови (ТУ):** умови , в яких визначаються вимоги щодо оформлення прийому і постачання готового програмного продукту . Розробляється на основі аналізу ТЗ і містить обгрунтування доцільності запропонованих варіантів структури програмного забезпечення . В технічних вимогах дається оцінка можливості розширення .

3 . **Ескізний проект (ЕП):** містить принципові конструкторські рішення по структурі програмного продукту , в яких визначаються всі розроблювані програмні модулі , організація взаємодії між ними , формулюється угода про розподіл ресурсів . Головна ціль ЕП - вибір оптимального проекту рішення на функціональному рівні , інша ціль - сформулювати угоду про прийомо-здавальні випробування . ЕП обов'язково затверджується замовником , і є основним документом , яким керується розробник в процесі розробки технічного проекту .

4 . **Технічний проект (ТП):** розробляє комплект конструкторських документів , в яких відображене остаточне рішення програмного продукту ( специфікації , комплект програм , технічних засобів , що підлягають розробці ) .

1 . Метод розширення ядра ;

2 . Метод програмування зверху-вниз ( метод структурного проектування ) ;

3 . Метод програмування знизу-вверх ( зворотній до попереднього ) застосовуються для контролю проектування .

## ВИДИ І МОДЕЛІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .

Всі існуючі прикладні програмні забезпечення ( ППЗ ) відносяться до одного з наступних видів:

1. Незалежні програми ;
2. Бібліотеки підпрограм ;
3. Мовні процесори ;
4. Багатофункціональні програми ;
5. Пакети прикладних програм ( ППП ) .

**1 . Незалежні програми** мають найпростішу структуру :

$$P = <G , P>$$

( головна програма і деяка підмножина процедур та функцій ) . Головна характеристика незалежних програм : для введення вихідних даних і виводу результату використовуються оператори мови програмування .

**2 . Бібліотеки підпрограм** ( допоміжний елемент ПЗ ) :

$$P = < P >$$

**3 . Мовні процесори** містять деяку мову і компілятор з цієї мови (мова - символьна) :

$$C = < L , C >$$

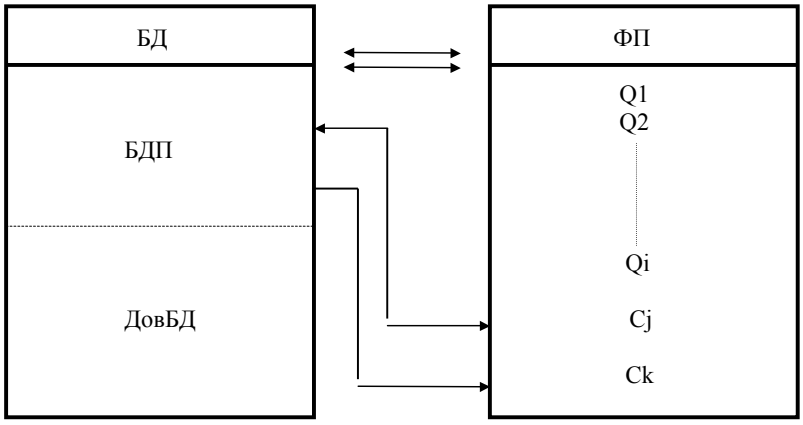
**4 . Багатофункціональні програми** складаються з мовного процесора і однієї або декількох незалежних програм :

$$P = < C , P >$$

**5 . ППП** складаються з мовного процесора + незалежні програми + деяка база даних + керуючий пакет :

$$P = < C , P , M , F >$$

## СТРУКТУРА ППП


**ФП** - функціональні програми . ФП можуть бути різними , але серед них обов'язково повинен бути один або декілька компіляторів . Декілька компіляторів потрібні для того , щоб забезпечити роботу декількох мов . Для однієї складної мови треба всього один компілятор , але це незручно для користувача . Для більш простіших мов , які простіше вивчити , треба декілька простіших компіляторів .

Структурні програми зводяться в базі даних . В БД можна виділити дві частини : поточну БД (або БД проекту) і довідкову .

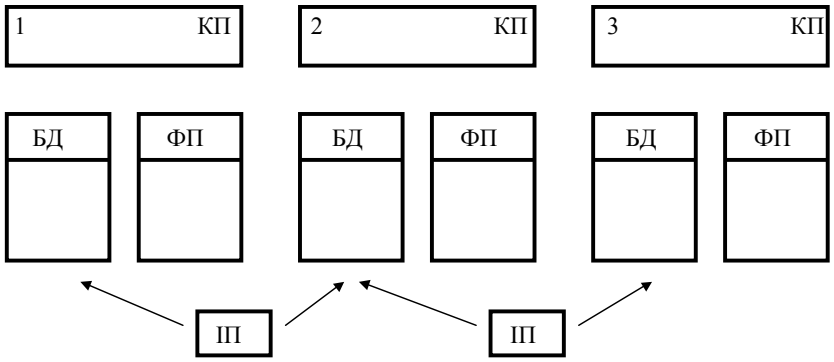
Всі ФП обмінюються між собою тільки через БДП . Зв'язок здійснюється через Керуючу програму через ФП . Запуск здійснюється через Загружчик , до якого поступає вхідна мова . Загружчик передає управляючі команди мови до Планувальника , який створює алгоритм рішення поточної задачі . Планувальник створює розрахункові схеми .

Інтерпретатор виконує розрахункову схему і керує ФП за даною схемою . Але ця робота схеми може керуватися людиною через діалоговий режим , а може бути автоматичною . ФП за схемою видає Вихідну мову (або дані) .Також необхідне Інформатор для того , щоб Користувач бачив , що його схема (прикладна програма) працює .

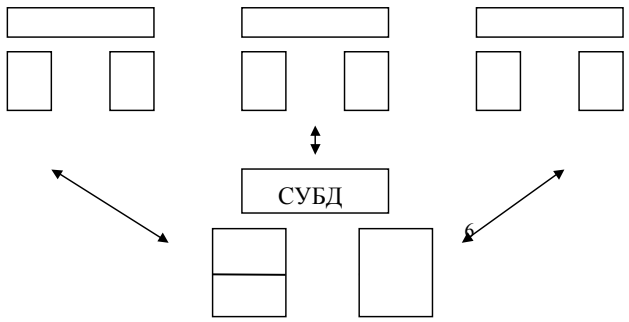
Така структура є зручною , бо ФП незалежні . Це дозволяє розробляти їх різним людям за умови стандартного обміну даними і модифікацій . БД також є динамічно змінними і підстроюється пакет (або структура файлів або СУБД) .

Використовування САПР не завжди зручно застосовувати , тому що для великих схем необхідно ввести велику кількість вхідних даних , в яких може бути багато помилок. Можна використовувати графіку, але дисплей обмежений. Вихід з цього - модульна графіка. Це також повільно, але вже полегшує роботу користувача. З цього бачимо, що використовувати прикладні пакети окремо не вигідно, тому зараз йде інтеграція ППП .

Edif - універсальна мова проектування.




Об'єднання здійснюється через інтеграційну програму , тому що дані кожної з них можуть дуже відрізнятись . Скільки ми приєднаємо пакетів - стільки буде ІП . Така схема пакетів зветься **інтегрованою САПР з послідовною структурою** (або не САПР , а просто схема) . Але можна взяти одну СУБД і приєднати до неї всі інші пакети (**САПР з паралельною структурою** ) . Можна не використовувати СУБД , а просто структури файлів зі спільним інтерфейсом .



БД ФП

Використання єдиної БД забезпечує наступні переваги :

1. Інформаційні структури стають більш інтегрованими тому що об'єктами зберігання є не тільки дані , але й структури , в яких вони організовані .
2. За рахунок структурування даних скорочується дублювання , підвищується надійність роботи .
3. Полегшується можливість організації захисту даних від несанкціонованого або неавторизованого доступу .
4. Прикладні програми становляться незалежними від опису даних і навпаки .
5. Забезпечується інтеграція інформації з забезпеченням з різним ступенем доступу різних груп користувачів .

В такому інтегрованому середовищі після закінчення отримаємо повну документацію , яка створюється поетапно .

Будь-яка велика програма не може бути написана одразу , а є похідною від малої програмної системи .

## ХАРАКТЕРИСТИКИ ППП

1. **Специфіка середовища застосування :**
   - регулярно ;
   - чи одноразово (рідко) використовувані програми .

Одноразово використовуваний пакет - це , наприклад , пакет логічного проектування .

2. **Орієнтація ППП :**
   - методоорієнтовані програми вирішують вузькоспрямовану задачу ;
   - поблемноорієнтовані програми дозволяють вирішувати закінчену задачу .
3. **Кваліфікація користувача :**
   - чим більш вузькоспрямована задача , тим вищою повинна бути кваліфікація користувача .
4. **Ступінь автоматизиції розрахункового процесу**
   - аналогічно , чим більш вузькоспрямована задача , тим менш необхіден більш автоматизований процес .

Якість програмного продукту дуже сильно залежить від характеристик програміста та його здібностей . Але в наш час намагаються звести роботу (участь) програміста до мінімуму . Добра програма - це розтяжне поняття і залежить від багатьох характеристик і причин .

Розробляти програму дорого і довго . Більше всього часу уходить на відладку і тестування програми . І зараз вже є погані прийоми програмування , від яких залежить якість програми :

1) Написання програми без коментаріїв ;
2) Використання мов низького рівня ;
3) Використання багатоцільових процедур і функцій ;
4) Створення програм , що самі себе модифікують ;
5) Використання імен , в яких мало літер .

Друга група проблем - це супроводження :

1) Наявність помилок в програмі , переданій на супроводження ;
2) Проблеми модернізації , пов'язані з системного програмного забезпечення ;

3) За необхідністю внести зміни до програми , користувачеві , як правило , важко знайти розробника ;

4) Важко знайти спеціалістів , які хотіли б супроводжувати програмне забезпечення ;

5) Програмна документація звичайно дуже погана .

У зв'язку з цим потрібно :

1) Використання нових технологій (сучасне ООП) ;

2) Писати документацію на ПЗ , призначене іншим , так , як ви би хотіли для себе ;

3) Уникати хитрих прийомів : чим простіше - тим краще .

Існує цілий ряд характеристик якості ПЗ . Всі вони якісні , і кількісних тільки дві : розмір і швидкість роботи , але вони не актуальні, тому що техніка дуже розвинена.

Всі характеристики якості виходять із властивості корисності програми . При цьому розрізняють :

1. **Вихідну корисність** - вона характеризується належністю програми для роботи у вихідному вигляді на потрібному комп'ютері .

2. **Загальна корисність** - показує , в якому вигляді програма дозволяє вносити зміни і використовуватись в умовах , відмінних від вихідних .

Тоді отримаємо наступні характеристики :

1. **Зрозумілість** .

Програмне забезпечення (ПЗ) володіє властивістю зрозумілості в тому ступені , в якому воно дозволяє оцінюваючий особі зрозуміти призначення програмного середовища .

2. **Свідомість** .

ПЗ володіє властивістю свідомості , якщо його документація не містить надлишкової інформації .

3. **Завершеність** .

ПЗ володіє властивістю завершеності , якщо в ньому присутні всі необхідні компоненти , кожний з яких розглянутий всебічно .

4. **Мобільність** .

ПЗ володіє властивістю мобільності , якщо воно може ефективно працювати на комп'ютері іншого типу , ніж ті , на яких воно було розроблене .

5. **Оцінюваємість** .

ПЗ володіє властивістю оцінюваємості , якщо для нього можуть бути сформульовані критерії оцінки застосовності для конкретного інформаційного забезпечення і проведена оцінка за цими критеріями .

6. **Надійність** .

Надійність - це ймовірність роботи програмного продукту без відказу протягом певного проміжку часу з урахуванням вартості кожного виду відказу .Сьогодні найбільш прораховується оцінка відказу в банківських системах .

7. **Точність** .

Точність характеризується оцінкою подання результату роботи програми .

## СЕРЕДОВИЩЕ ДЛЯ РОЗРОБКИ ПРОГРАМНИХ ЗАСТОСУВАНЬ.

У зв'язку зі зростаючою складністю ПЗ пропонуються різні засоби тестування , тощо.

System Application Architecture - **SAA** - Архітектура середовища для розробки програмних додатків .

SAA складається з трьох компонентів :

1. Система інтерфейсів користувача ( Common User Access ) ;
2. Система комунікації ( Common Communication Support ) ;
3. Система програмних інтерфейсів ( Common Programming Interface ) .

Всі ці компоненти забезпечують програмні інтерфейси , протокол та угоду про взаємні дії людини з ПП , ОС . Крім того , тут визначені засоби комунікацій і міжпрограмні інтерфейси . Існують специфікації цих компонентів , відкриті для всіх розробників . Існує ще один компонент :

4.Common Application - Угода про розробку прикладних програм (ПП) і систем . Розробка цього стандарту - найважливіший етап у розвитку ПЗ . Стандарт SAA визначає вимоги до ОС , які розробляються і використовуються сьогодні . Вимоги, які реалізовані в сучасних ОС такі :

1) Збільшення продуктивності у програмістів і кінцевих користувачів ;

2) Значне полегшення використання і супроводження прикладних систем за ра-хунок модифікації системних угод ;

3) Покращення засобів комунікації , що дозволяє широке застосування мережених технологій ;

4) Збільшення віддачі інвестицій на розробку інформаційних систем за рахунок кращого використання програмних ресурсів та досвіду користувача .

## СТРУКТУРА SAA .

Наріжним каменем структури SAA є наступна структура ПЗ :

1. На вищому рівні ( доступному кінцевому користувачеві ) знаходяться прикладні системи (ПС) .
2. Рівень , який підтримує ПС , забезпечує їх розробку , включає мови програмування , генератори додатків , генератори звітів ( інструментальні програмні засоби , які дозволяють формувати текстові засоби ) .
3. На третьому рівні знаходяться програмні засоби , які забезпечують перший і другий рівень : керування трансакціями , керування комунікаціями , керування базами даних , мережеві засоби , керування файлами , керування діалогом , керування завданнями , захист даних , керування представленнями (тобто формою видачі інформації на екран ) .
4. Найбільш віддалені від програмного користувача - системні засоби .

Найнижчий ,четвертий рівень , в архітектурі SAA використовує специфічні функції конкретної ОС , враховує архітектурні особливості апаратури і , отже , архітектура залежить від розрахункового середовища . На цьому рівні виконується керування пам’яттю ЗЗП (зовнішній запам’ятовуючий пристрій - "ВЗУ") , процесором .

Наступний , третій рівень , представляє програмні засоби комунікації , які забезпечують передачу даних між розподіленими ОС та ПС .

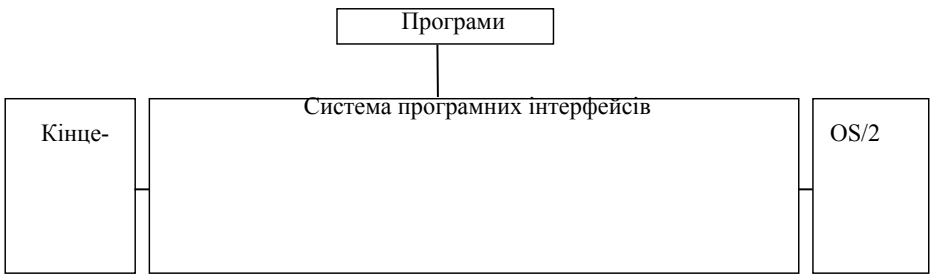
Другий рівень - рівень базових функцій , які використовуються для розробки конкретних додатків .

Перший рівень - прикладні системи ( сапровські системи , банківські , офісні , …) .

Таке ділення ПЗ на рівні дозволяє створювати сучасне мобільне ПЗ , яке може переноситися на різні платформи .

Введена архітектура ПЗ є організаційним засобом при виробництві ПС і реалізована в усіх операційних середовищах фірми IBM . На основі цієї архітектури розроблені операційні середовища Windows 95 фірми Microsoft .

Всі інтерфейси мають наступну структуру :



| | | | |
|---|---|---|---|
| вий користу-вач | Система інтерфейсів користувача | Керування комунікаціями | WIN NT MVS |

**Система інтерфейсів користувача** забезпечує інтерфейс кінцевого користувача з терміналом або АРМ . Уніфікація інтерфейсів користувача забезпечує їх застосування в різних обчислювальних середовищах .

**Система комунікацій** керує протоколами комунікацій , забезпечуючи сумісне виконання завдань різними системами . Завдяки системі комунікацій можна створювати мережі різної конфігурації з використанням різних апаратних плат-форм . Протоколи системи комунікацій забезпечують :

1. Керування потоками даних в мережах ;
2. Керування додатками ;
3. Керування сеансами в мережах ;
4. Керування мережею на фізичному рівні мережі ;
5. Керування зв'язком даних .

**Система програмних інтерфейсів** визначає такі способи програмування нових додатків , щоб вони могли виконуватись на будь-яких технічних засобах
IBM-овської платформи ( тобто незалежність ПС від архітектури конкретної системи апаратури і надання користувачеві певного набору інструментів , до якого входить мова високого рівня (СИ , Паскаль , Делфі), генератор ПП й екстерні системи).

## МЕТОДИ ПРОЕКТУВАННЯ ПС .

За визначення **метод -** це послідовний процес створення ряду моделей , які описують певними засобами різні сторони розроблюваних ПС . На основі методу створена так звана **методологія -** або сукупність механізмів , що застосовуються в процесі розробки ПЗ і об'єднуються одним спільним філософським підходом до елементів . При цьому механізми методології впорядковують процес створення ПС, являючись спільними для всіх розробників і дозволяють менеджерам проекту контролювати його розвиток .

## ОСНОВНІ ПРИНЦИПИ ПРОЕКТУВАННЯ ПС.

1. **Принцип системної єдності .**

Під час створення , функціонування і розвитку прикладного додатку зв'язки між компонентами повинні забезпечувати його цілісність .

2. **Принцип розвитку .**

Програмний додаток повинен створюватись і функціонувати з урахуванням доповнення , удосконалення та оновлення його компонентів .

3.. **Принцип сумісності .**

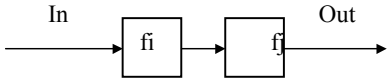
Мови , символи , коди , інформації та зв'язки між компонентами повинні забезпечувати їх сумісне функціонування і зберігати відкриту структуру системи вцілому .

4. **Принцип стандартизації .**

Під час проектування програмних додатків необхідно уніфікувати і стандартизувати компоненти , інваріантні до різних задач .

В 60 - 70 ті рокі отримав розвиток **метод структурного програмування** (**метод програмування зверху - вниз**) . Недоліки : складна програма не повинна перевищувати 100 000 строк тексту . Для більш громіздких програм застосовується **метод об'єктно - орієнтованого програмування** або **метод організації потоків даних** .Метод організації

потоків даних застосовується при рішенні задач інформаційного забезпечення , в яких існує прямий зв'язок між вхідним і вихідним потоком даних , тобто структура будь-якої програмної системи лінійна і включає послідовні блоки перетворень між вхідними і вихідними даними .

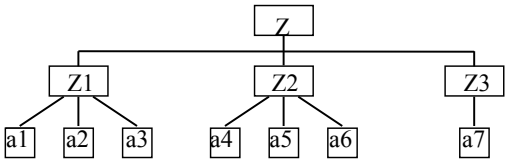



В основі структурного програмування лежить поняття форми і дисципліни , тобто програміст створює програму із базових логічних структур . Базові поняття структурного програмування :

1. Низхідна розробка ;
2. Структурне кодування ;
3. Наскрізний структурний контроль .

**1. Низхідне проектування .**

Полягає в тому , що формулюєтья вихідна задача , яка розбивається на підзадачі , кожна з яких деталізується , тобто ми реалізуємо алгоритмічну декомпозицію



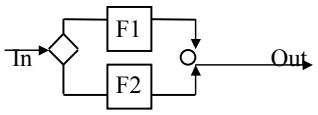

вихідної задачі . Вона розвивається до виходу на рівень програмних процедур і функцій. Структурне програмування полегшує розробку , тому що ми рухаємося зверху-вниз . В результаті поступово замінюючи заглушки або імітатори закінчення програмного продукту , ми можемо бути впевнені , що задача розв'язана правильно , тому що ми рухаємося зверху-вниз . Проблеми виникають , якщо вихідна задача неточно сформульована або постановка її застаріла . Другим недоліком є те , що ми отримуємо працездатний продукт лише на кінцевому рівні , тобто внести будь-які зміни в програму дуже складно до її закінчення , також , як і оцінити результат .

**2. Структурне кодування .**

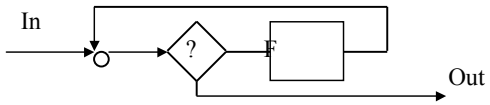
При написанні програм допускається обмежене написання програмних структур .

- слідування ;


- галуження ;




- цикл або повторення .




Також допускаються можливі комбінації цих структур . Кожна програмна структура не повинна перевищувати 100 операторів мови програмування . Використання GOTO дозволене тільки у виключних випадках ( і треба передавати керування нижче строчки з

GOTO) . Структурне програмування - це програмування , в якому потік керування спрямований зверху-вниз .

**3. Наскрізний структурний контроль .**

Ідея полягає в тому , що при низхідній розробці необхідно постійно контролювати всі системні угоди . Будь-яке їх порушення призведе до того , що в кінці контроль не збігається .

## НАСКРІЗНИЙ СТРУКТУРНИЙ КОНТРОЛЬ .

Прийоми структурної декомпозиції .

1. **Пошагова деталізація** .

Вона полягає в тому , що процес передбачає первісний опис логіки програми в термінах деякої гіпотетичної мови дуже високого рівня з наступною деталізацією кожного речення в термінах мови нижчого рівня і так далі , поки не буде достигнутий рівень мови програмування . При цьому на кожному етапі логіка програми виражається тільки в основних конструкціях структурного кодування .

Наприклад :

< вказати дії > ;

< прочитати ім'я > ;

While < не вичерпан список > Do

< чи є контролюємий параметр > ;

If < є відповідність > Then < включити флаг > ;

2. **Захисне програмування .**

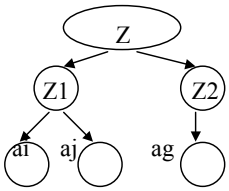
Основане на тому , що всі вихідні дані , які поступають до програмного модуля і читаються з нього , перевіряються на достовірність ( наприклад , на ОДЗ , перевіряється допустимість значень індексів , масивів , керуючих змінних , операторів вводу / виводу) .

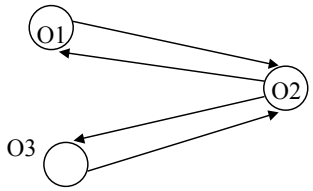
## ОБ'ЄКТНО - ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ .

Будь-який метод програмування будується , перш за все , на використанні відповідних механізмів мови програмування . ООП - це методологія , яка поєднує в собі як процес об'єктної декомпозиції , так і прийоми представлення логічної та фізичної моделі проектуємо програмної системи . В цьому означенні є дві складові частини :

1. ООП - це є отримання об'єктно - орієнтованої декомпозиції .
2. Існують засоби різноманітного відображення логічної структури моделей системи , а також різноманітність моделей , які відображають фізичну структуру програмної системи (архітектуру модулів і процесу) .

З точки зору формального програмування , основна відміна ООП від структурного заключається в підтримці об'єктно - орієнтованої декомпозиції .



Структурне Об'єктно - орієнтоване (самостійні об'єкти)

Результатом об'єктно - орієнтованого аналізу (ООА) є основний критерій якості подальшого програмування . Об'єктно - орієнтований аналіз спрямований на створення моделей предметної області , максимально близьких до реальності . Вимоги до моделей формулюються з використанням понять "класів" та "об'єктів" , що складають словник предметної області . Таким чином , технологія ООП включає три етапу :

1. За результатами ООА будуються моделі предметної області (тобто визначається сукупність об'єктів та їх властивості) .
2. На основі отриманих моделей виконується об'єктно - орієнтоване проектування (визначається система класів та об'єктів в термінах вибраної мови програмування) .
3. Виконується програмування з використанням прийомів ООП .

Прийоми і способи програмування визначаються мовою програмування . Всі мови програмування можна класифікувати за стилем програмування , який вони допускають . Виділяється п'ять критеріїв класифікації мов :

1. **Процедурно - орієнтований стиль програмування .**

Абстракції , що визначають результативну програму , - алгоритми (Фортран) .

2. **Об'єктно - орієнтований .**

Основні абстракції програм - класи та об'єкти (С++ , Паскаль) .

3. **Логічно - орієнтований .**

Абстракції виражені в здійсненні перекатів (Проглог) .

4. **Стиль , орієнтований на правила .**

Підтримується в мовах програмування експертних систем . Абстрації :
якщо < умова > тоді < дія > .

5. **Стиль , орієнтований на обмеження .**

Абстракції - інваріантні співвідношення , які характерні для вузьких систем .

Кожний стиль програмування має свою концептуальну основу . В об'єктно - орієнтованому стилі програмування вона базується на чотирьох головних елементах :

1. Абстрагування .
2. Обмеження доступу .
3. Модульність .
4. Ієрархія .

Виключення будь-якого з перерахованих елементів порушує об'єктно - орієнтований підхід до програмування . Крім того , є ще три додаткових елемента :

1. Типізація .
2. Паралелізм .
3. Стійкість .

Відсутність або нерозуміння програмістом концептуальної основи призведе до того , що програма , написана на об'єктно - орієнтованій мові , практично не буде відрізнятись від програм , написаних не на об'єктно - орієнтованій мові , але , головне, при цьому практично неможливе вирішення дуже складної задачі .

Розглянемо основні елементи об'єктно - орієнтованого стиля програмування .

## 1. АБСТРАГУВАННЯ.

Абстракції - це такі істотні характеристики деякого об'єкта , які відрізняють його від інших об'єктів і таким чином точно визначають його особливості з точки зору подальшого розгляду і аналізу . Головна проблема ООП - вибір достатньої множини абстракцій для заданої предметної області . В абстрагуванні є виділення об'єктів . Об'єкт , який використовує ресурси інших об'єктів , називається **клієнтом** . Опис поведінки об'єкта включає опис операцій , які можуть бути виконані над ним , і операцій , які об'єкт виконує над іншими об'єктами . Такий підхід концентрує увагу на зовнішніх особливостях об'єктів .

Повний набір операцій , які об'єкт може здійснити над іншими об'єктами , називається **протоколом** . Всі абстракції володіють як статичними , так і динамічними властивостями . Наприклад , об'єкт "файл" має ім'я , певний об'єм пам'яті і деяку інформацію , що записана в цій пам'яті . Всі ці параметри статичні . Конкретні значення параметрів змінюються , тобто вони динамічні . Об'єктно - орієнтований стиль програмування пов'язаний з впливом на об'єкти (впливає деяка множина подій) . Вплив на об'єкт визиває реакцію об'єкта , яку можна виконати по відношенню до даного об'єкта або іншого об'єкта . Це відрізняє об'єктний стиль від процедурного . В процедурному стилі зміна параметрів - суть самого програмування . В ООП зміна параметрів міститься у властивостях самого об'єкта , а суть - керування потоком інформації .

## 2.ОБМЕЖЕННЯ ДОСТУПУ .

Створенню абстракцій деякого об'єкта повинні передувати обмеження на спосіб і реалізацію . Вибраний спосіб реалізації повинен бути схованим і захищеним для більшості об'єктних користувачів . Ніяка частина складної системи не по-винна знаходитись в залежності від подробиців побудови інших частин . Обмеження доступу дозволяють вносити до програми зміни , зберігаючи її надійність і мінімізуючи затрати на програмування . Абстрагування і обмеження доступу є взаємо-доповнюючими операціями . Абстрагування фіксує увагу на зовнішніх особливостях об'єкта , а обмеження доступу - це є захист інформації , яка визначає явні бар'єри між різними абстракціями . Практично , це є різне написання двох частин інтерфейсу і внутрішньої реалізації . Інтерфейс створює абстракцію поведінки всіх об'єктів даного класу , тобто він відображає зовнішню прояву об'єкта . Внутрішня реалізація описує механізми бажаної поведінки об’єкта . Таким чином , обмеження доступу - це процес захисту окремих елементів об'єкта , який не зачепає істотних характеристик об'єкта як цілого . На практиці захищається як структура об'єкта , так і реалізація його метода . Наприклад , в мові Smaltolk реалізується повний захист від доступу до змінних об'єкта іншого класу ще під час компіляції . Object Pascal взагалі не здійснює цей контроль . Це зрозуміло з точки зору ООП . Захист здійснює програміст . В C++ більш гнучке керування доступом , тобто деякі об'єкти відносяться до загальнодоступних , а деякі до часних .

## 3. МОДУЛЬНІСТЬ .

Класи і об'єкти складають логічну структуру програми . Якщо ці абстракції реалізовані в модулі , створюється фізична структура програми . Модульність - це розділення програм на незалежно - компільовані фрагменти , що мають між собою засоби повідомлення . В більшості мов , які підтримують принципи модульнго програмування , здійснюється розділення на інтерфейсну частину і реалізаційну. Особливість ООП полягає в тому , що при формуванні модулів необхідно фізично розділяти класи і об'єкти , які складають логічну структуру програми і явно відрізняються від простого набору процедур і функцій . Кінцевою метою декомпозиції програми на модулі є зниження затрат на програмування за рахунок незалежних розробок і тестування окремих модулей .

Принцип модульного програмування або модульного структурування по-лягає в розділенні програми на функціонально - самостійні частини (або модулі) , що забезпечує замінність , модифікацію , видалення або доповнення складових час-тин або модуль-операндів . Переваги використання модульного принципу програмування :

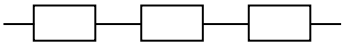
1. Спрощується відладка програм , скорочується об'єм програм .
2. Підвищується надійність .
3. Виключається взаємовплив помилок .
4. Забезпечується можливість організації сумісної роботи великих колективів програмістів .

Модульна структура програм формується на етапі технічного проектування . Результат - структура модулей і звязки між ними .

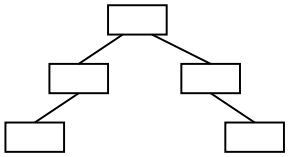
Структурна організація модулей може бути побудоване по-різному . Виділяють декілька основних типів :

1. **Монолітно-модульна структура .**

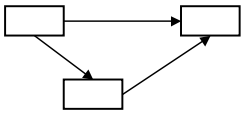

2. **Модульно-послідовна структура .**



3. **Модульно-ієрархічна структура .**



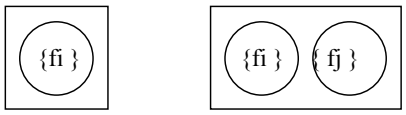
4. **Модульно-хаотична структура** (впорядкованості першого і третього типу немає) .



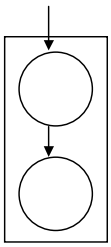
Виділяється таке поняття , як **цілісність модулей** . Модулі можуть мати

1. **Функціональну цілісність .**

Модуль з функціональною цілісністю має один об'єкт , який реалізує набір функцій . Якщо в модуль зведені декілька об'єктів , незалежних один від одного , то це не порушує функціональної цілісності .




2. **Послідовно-функціональна цілісність .**



{fi }

{ fj }

3. **Модулі з алгоритмічною цілісністю .**
Містять об'єкти , об'єднані за принципом алгоритмічної спільності .
4. **Модулі з часовою цілісністю .**
Об'єднують об'єкти , що виконуються на одному часовому етапі обробки даних .

## 4. ІЄРАРХІЯ .

Для складної предметної області збільшення кількості абстракцій призводить до втрати контролю за ними . В деякому ступені , контроль забезпечується обмеженням доступу . Другий спосіб - побудова систем-ієрархій .

Ієрархія - це ранжирована або впорядкована система абстракцій . Основними видами ієрархічної структури є **структура класів** ( або **ієрархія за номенклату-рою**) і **структура об'єктів** (або ієрархія за складом ). Ієрархія створюється через механізм наслідування . Розрізняють просте (використовується в Паскалі) і множинне (використовується в С ) наслідування :

Просте Множинне

Якщо ієрархія за номенклатурою задає відношення узагальненості , то ієрархія за складом задає відношення агрегірования (це відношення об'єднання декількох елементів структури).

Додаткові елементи об'єктно - орієнтованого стиля програмування .

## 1. ТИПІЗАЦІЯ .

Тип - це точне означення властивостей , побудови або поведінки , яке властиве деякій сукупності об'єктів . Тип і клас - еквівалентні поняття . Клас - це результат логічного проектування . Тип формує поняття класу в програмі . Типізація дозволяє виконати опис абстракцій таким чином , щоб вони підтримувалися на рівні мови програмування . Об'єктно - орієнтовані мови програмування можуть бути строго , нестрого типізовані або зовсім не типізовані . Це означає , що в строго типізованій мові всі елементи класу повинні бути описаними (За умовчанням не можна) . В нестрого типізованій мові деякі елементи формуються за умовчанням . Застосування слаботипізованих мов призводить до появи помилок в програмах . Наприклад ,

1. Декларування типів даних спрощує документування програм .
2. Компілятор генерує більш ефективний об'єктний код при явному об'явленні типів даних .
3. Якщо типи даних не контролюються іноді під час компіляції , то відладка програм сильно ускладнюється , тому що можливі дуже загадкові збої в роботі .

## 2. ПАРАЛЕЛІЗМ .

Це властивість об'єктів знаходитися в активному або пасивному стані .

## 3. СТІЙКІСТЬ .

Це властивість об'єкта існувати в часі та просторі достатньо тривалий час , незалежно від процесу , що спричинив це явище .

Процес розробки включає два основних етапу :

1. Архітектурний етап програмування (ескізне , технічне) .
2. Етап детального програмування (робоче програмування) .

На архітектурному етапі розробляються специфікації . Зовнішні специфікації визначають структуру програмних модулей .

## РОЗРОБКА СПЕЦИФІКАЦІЙ .

Під специфікацією розуміють достатньо повний і точний опис задачі , яку необхідно вирішити . Специфікація являє собою опис в термінах предметної області . Розрізняють функціональні та експлуатаційні специфікації .

**Функціональні специфікації** описують об'єкти , які беруть участь в задачі, ділення задачі на підзадачі (домени) , вхідні та вихідні дані , процеси та дії , реакцію на виключні ситуації , тощо . Функціональні специфікації включають такі розділи :

1. Призначення програми .
2. Граничні умови .
3. Опис функцій .
4. Опис вхідних та вихідних даних .
5. Верифікаційні вимоги (описуються всі тести по контролю програми) .
6. Тип та кількість документів , які складають специфікацію .

**Експлуатаційні специфікації** визначають вимоги до швидкості роботи програми , використовуваним ресурсам , спеціальні вимоги до надійності та безпеки .

Специфікації поділяються на два класи :

1. Графічні .
2. Текстові .

Графічні мови представляють специфікації у вигляді графів та діаграм (найпростіший - блок-схема , SA , HIPO) . До графічних мов відноситься мова керуючих графів (ЯПГ - ярусно-паралельних графів) .

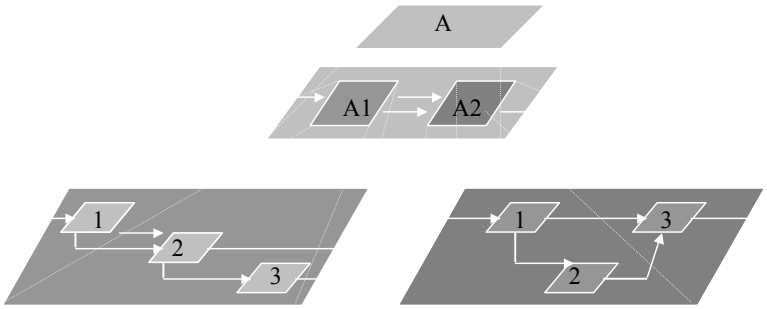
Текстові мови представляють специфікації у вигляді текстів в псевдокодах (PDL , PSL , мови пошагової деталізації) . Текстова мова описує специфікацію архітектурного рівня , формує набір специфікацій на програмні модулі і включає наступні розділи :

1. Ім'я та мета .
2. Неформальний опис (містить загальний огляд дій) .
3. Посилання (описує всі посилання даного модуля на інші і всіх інших на даний модуль) .
4. Ввод / вивод (описує структуру вхідних і вихідних даних) .
5. Алгоритм (описує всі дії модуля , задає структуру класів і структуру об'єктів) .
6. Примітки .

## РОЗРОБКА СПЕЦИФІКАЦІЙ МЕТОДОМ СТРУКТУРНОГО АНАЛІЗУ SA

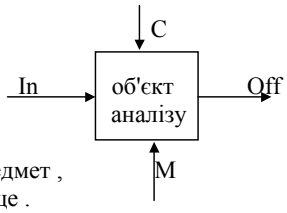
Метод структурного аналізу являє собою послідовну деталізацію проектуємої системи зверху-вниз . Деталізований розклад на кожному рівні повинне бути повністю еквівалентним до попереднього рівня . Документований результат розкладу представляється у вигляді набору графічних схем і пояснень до них . Такий набір схем

називається **моделлю системи** . Модель системи - це є ієрархічно зв'язаний набор схем , в якому кожна схема є деталізацією якогось об'єкта та його оточення зі схеми попереднього , більш високого рівня . Аналізуємий об'єкт представляється на схемі у вигляді набору елементів , зображених прямокутниками , і зв'язку між ними , зображеного стрілками . Елементи , на які розкладений вихідний об'єкт в сукупності повинні точно представляти цей об'єкт . Сукупність стрілок, які входять до системи і виходять з неї , створює середовище схеми . Середовище по-винне точно співпадати з середовищем деталізованого об'єкта .




Середовище об'єкта , зображене стрілками , може складатися з чотирьох типів стрілок :

- входу ;
- виходу ;
- керування ;
- механізму .



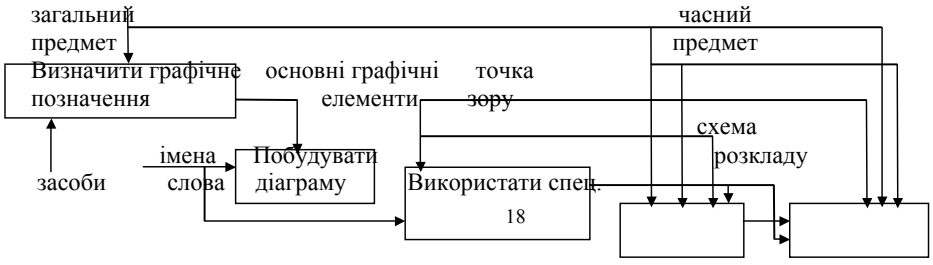

Об'єкти аналізу - це або предмети або операції . Якщо об'єкт аналізу предмет , то операції - його зовнішнє середовище .

Якщо об'єкт аналізу предмет, то стрілка входу зображує операції , які створюють цей предмет , виходу - операції , які використовують цей предмет , керування - умови існування предмета (або даного) , а механізму - засоби реалізації даного .

Якщо об'єкт аналізу операція , то стрілка входу зображує дані , які обробляє операція , виходу - отримані в результаті операції , керування - умови , при яких виконується операція , а механізму - засоби реалізації операції .

Для опису зв'язків і реалізації посилань між елементами моделей їм надаються наступні позначення . Якщо об'єкт аналізу операція - він позначається символом A , якщо дані - Д . Літери всіх елементів супроводжується номером (нумерація всіх елементів схеми виконуєтья зліва-направо , зверху-вниз) . Існує сувора регламентація формування схем розкладу . Їх креслять на бланках , в нижній та верхній частині виділяють область ідентифікатора , де розміщують позначення схеми , назву , відомості про автора , номер листа , тощо . Зміст моделі включає число схем , їх назву і кількість сторінок . Створюється тлумачний словник використаних термінів . Розглянемо приклад такої схеми :

Назва : Модель



| Позначення | Забезпеч. посилання | Організувати матеріал |
|---|---|---|

## ОПЕРАТИВНО - ГРАФІЧНИЙ МЕТОД .

Специфікація являє собою альбом зв'язаних між собою схем (за HIPO) , кожна з яких описує деяку частину задачі або системи , яка підлягає розробці . Щоб скласти специфікацію за методом HIPO вихідна задача ділиться на частини . Декомпозиція задачі виконується аналогічно до технології SA , тобто отримаємо деревовидну систему розкладу , в якій коренем дерева є система вцілому , а на наступних рівнях - все більш деталізовані її частини . На відміну від SA метод HIPO має дві особливості :

1. Кожна частина системи і розклад цієї частини на наступному рівні - не обов’язково еквівалентні . Зрозумілі деталі можуть не підлягати подальшому розкладу .
2. Розклад кожної частини об'єкта не залежить від розкладу інших частин . Це виконується за функціональним признаком . Це означає , що кожна виділена частина повинна описувати деяку закінчену змістовну функцію . Механізм реалізації цієї функції при цьому не враховується .

Добрий розклад дозволяє отримати повне уявлення про систему з будь-яким ступенем деталізації . По закінченню розкладу розробник готує комплект документацій, які включають схему кожного з елементів розкладу і схему составу розкладу . Схема составу розкладу являє собою деревовидний граф з коренем у верхній частині листа . Верхівками графа є прямокутники , що зображують елементи розкладу , кожний з яких вписаний в функціональне найменування елемента . Крім цього , кожний елемент містить цифрове або літерно-цифрове позначення . Воно складене таким чином , що несе інформацію про номер рівня відповідного дерева (зліва-направо) . Лінії , які з'єднують прямокутники , показують входження елементу в елемент верхнього рівня .

Крім графа , який зображує структурний опис задачі , складають зміст альбома документів , який включає позначення всіх елементів та їх найменування і номери листів в альбомі .

Таким чином , метод HIPO не накладає ніяких обмежень на методику ієрархічного розкладу і може застосовуватись на будь-якому етапі розробки .

Наприклад , візьмемо задачу коректировки информації і складену для неї схему составу розкладу .

| Програма КОР. | Схема составу розкладу | Лист 1.0<br>Дата 18.09.97 |
|---|---|---|

Коректувати розділ бібліотеки
Схема 1.1

Виконати контроль даних
Схема 2 .1

Коректувати черговий запис розділу
Схема 2 .2

| Позначення | Найменування | Номер листа | Примітки |
|---|---|---|---|
| 0 . 0 | Схема составу розкладу | 1 . 0 | |
| 1 . 1 | Коректувати розділ бібліотеки | 2 . 0 | |
| 2 . 1 | Виконати контроль даних | 3 . 0 | |
| 2 . 2 | Коректувати черговий запис розділу | 4 .0 | |


Опис кожного елементу розкладу виконується у вигляді схеми , яка показує кожний елемент як функцію обробки інформації . Схема містить заголовок , вміст і коментарій . Коментарій - це таблиця , яка показує всі вхідні та вихідні файли з найменуванням , призначенням , тощо . Наприклад , І КОР - вихідни дані для коректировки . Вихідні дані для схеми 2 .1 зустрічаються на листі № 1 і листі № 3 . Вхідні та вихідні дані поєднуються з елементами перетворення даних стрілками . Тонка стрілка - зв'язок за керуванням , інші - зв'язок за даними . Коло зі стрілкою - зв'язок за керуванням всередині листа . Зв'язок за керуванням між листами - 4 .0
( № листа , № пункта звідки / куди ) .

4.

Схема повинна бути наглядною , читабельною і зрозумілою для програміста .Бажано так проводити розклад , щоб кожний елемент розкладу вміщувався на сторінці для спрощення роботи . Допускається включення до альбому також інших документів (форми вихідних звітів , опис вхідних даних , програм , тощо) .

## РОЗРОБКА СПЕЦИФІКАЦІЙ З ВИКОРИСТАННЯМ ПСЕВДОКОДА .

Псевдокод - це частково формалізований запис для наглядного текстового представлення схем алгоритмів і програм . В загальному випадку , псевдокод має принципово неформальний характер і не розрахований на автоматичну трансляцію написаних на ньому алгоритмів . В результаті розробник свободний від додержання формальних мовних правил і може приділити основну увагу змістовному рішенню логічних основ алгоритму . Мови специфікації виконуються у відповідності до технології пошагової деталізації . Зокрема , мова ПДП (проектування і документації програм) , як будь-яка мова , включає в себе основні елементи мови :

1. Алфавіт (прописні і строчні літери рус. / лат . алфавіту , цифри , спеціальні математичні символи) .
2. Лексеми мови : ідентифікатори (будь-яка строка символів , яка складається з літер , цифр і починається з літери) .
3. Ключеві слова (якщо , то , тощо) .
4. Константи .

До лексем мови відносяться ще коментарії і оператори мови . Для запису оператора вводяться обмеження :

- в одній строці - один оператор ;
- в кінці оператора - " ; " ;
- вложені оператори повинні зсуватися впрво на декілька позицій по відно шенню до попереднього рівня вложеності .

Типи операторів :

1. **Оператор дії .**

Це неформалізований запис деякого предписання ("Знайти кінець файла" , …) .

2. **Оператор присвоєння .**

< ліва частина > ::= < вираз > ;

Вираз може бути у вигляді математичного запису або оператора дії . В принципі , вираз може бути деталізованим до запису на конкретній мові програмування .

3. **Оператор керування .**

Будується на основі ключевих слів , які схожі на ключеві слова Паскаля .

Розглянемо приклад :

```
/1VET - процедура (X , N , M , V , E)
Параметри
вхідні
X - масив (1..N) , у відповідний _ _ вихідний масив чисел
N  - скаляр , цілий _ _ розмір масиву X
вихідні
Тіло
C2 := C2 := 0 ;
Цикл по
I := 1  до  N  шаг 1
повторити
C1 := C1 + X(I) ; і  т . д .
```

Недолік метода - він орієнтований на отримання структурованих програм , тобто для об'єктних практично не підходить .

## ПРОЕКТУВАННЯ З ВИКОРИСТАННЯМ КЕРУЮЧИХ ЯРУСНО - ПАРАЛЕЛЬНИХ ГРАФІВ .

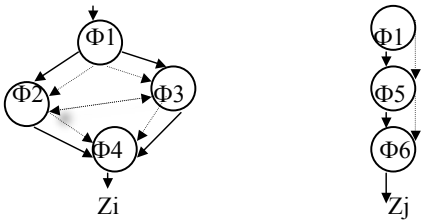
Проектування починається з визначення множини задач , які реалізуються в проектуємій системі Z = { Zi , i = 1 , … , | Z |} . Кожній задачі можна поставити у відповідність один або декілька алгоритмів , за допомогою яких вона вирішується . Множина алгоритмів A = { aj , j= 1 , … , | A | } , | A | > | Z | . Алгоритм - це є впорядкована множина функціональних операторів , послідовне виконання яких призводить до вирішення поставленої задачі .

a j = < Ф1 j (X ,Y) , Ф2 j (X ,Y) , … ,Ф n j (X ,Y) >

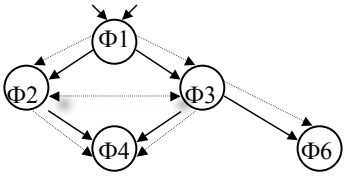
Задача розробника спроектувати ПЗ таким чином , щоб мінімізувати його об'єм . Якщо вважати , що кожний функціональний оператор це є програмний модуль , об'єкт або просто процедура чи функція , тоді задача : для множини Z вибрати таку структуру ПЗ , яка дозволяє з мінімальними затратами ресурсів вирішити всі алгоритми множини A .

Задача вирішується таким чином .

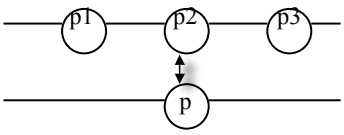
Для кожного алгоритму множини A будується граф , верхівкою якого є функціональний оператор Ф(X ,Y) , де X - множина вхідних даних ,Y - множина результатів . Керуючі дуги показують , в якій послідовності треба рухатися за алгоритмом . Інформаційні дуги показують , що реалізується процедура склеювання еквівалентних верхівок .



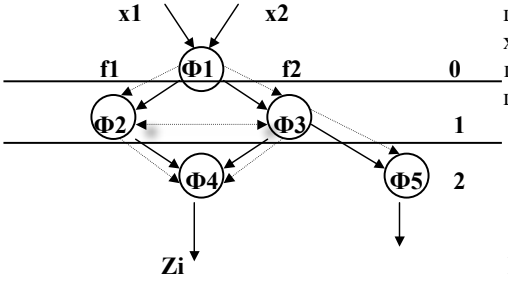

Дуги також нумеруються . Подібні графи рисуються для всієї множини алгоритмів . Після цього визначаються еквівалентні верхівки графа (для них вхідні і вихідні інформаційні дуги співпадають) . Потім реалізується процедура склеювання еквівалентних верхівок . Т . ч . , отримаємо новий граф , в якому кількість верхівок є кількістю реалізованих задач .




Використовуючи формальний апарат графів , ми намагаємося побудувати оптимальну структуру . Задача - впорядкувати послідовність функціональних операторів , які зведені до мінімальної кількості програмних операторів . Створюється один спільний граф для множини алгоритму шляхом склеювання . Редукція - коли декілька послідовних складних операторів поєднуються в один складний оператор , таким чином в результуючему графі виключаються лінійні послідовності операторів , тобто після кожної верхівки потрібно вирішувати , по якій гілці ми будемо рухатися . Таким чином , отримаємо **ярусно - паралельний граф .**



0 - й ярус - дуги вхідних даних ; 1 - й ярус - оператори , зв'язані з дугами вхідних і вихідних даних ; i-1 - ярус - дуги вхідних і вихідних даних . Розрахунковою схемою алгоритму задачі **Zi** на ярусно - паралельному графі (ЯПГ) будемо називати всі



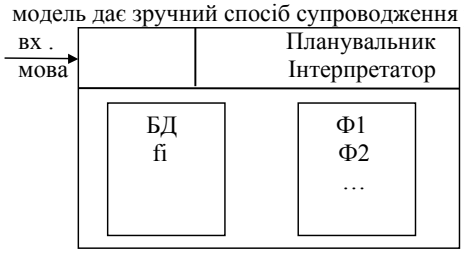

шляхи , що ведуть від початку вер хівки до її кінця . Верхівці **Zi** в на-шому випадку відповідають два шляхи :

Qi 1 = < Ф1 , Ф2 , Ф4 >
Qi 2 = < Ф1 , Ф3 , Ф4>

В результаті цих шляхів отримаємо розрахункову схему :

Qi = < Ф1 , < Ф2 , Ф3 >, Ф4 >.

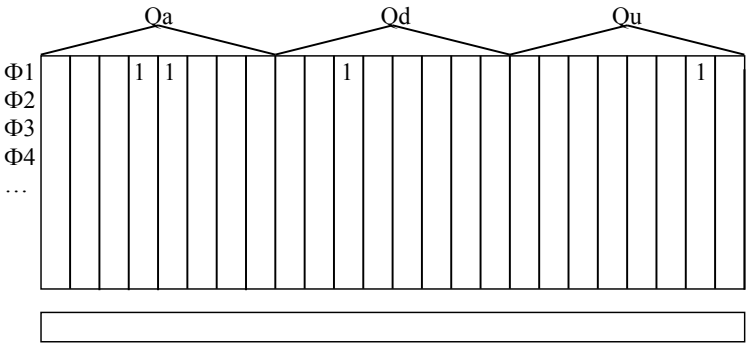
Немає різниці - виконувати Ф2, Ф3 чи Ф3, Ф2 , тому що вони входять до одного яруса . Таким чином , оператори , які входять до одного яруса або до однієї розрахункової схеми , можуть виконуватись в будь-якому порядку . Яруси ЯПГ визначають стан розрахункового процесу . Кожному функціональному оператору ми можемо поставити свою програму виконання (планувальник планує схему , інтерпретатор її виконує) . Якщо система не є інтерактивною , потрібно вказати тільки кінцеву мету , і планувальник будує свою схему . Але користувачеві простіше самому виконати схему , додаючи вхідні дані . Вся множина команд деякого яруса є активною , інших ярусів - пасивною . База даних - інформаційні дуги , це файли fi в БД . ЯПГ - це математична модель пакета прикладних програм . Всі мови - множина дуг Xi . Маючи таку структуру , можна керувати програмним продуктом . Ця модель дає зручний спосіб супроводження розра-




хункових процесів - реалізація роз-рахункової схеми алгоритму (впорядкована послідовність операторів і мультіоператорів) . Виконання оператора супроводжу-ється повідомленнями :

- інформаційні повідомлення Qu = { U1, U2, … , | Qu | };
- аварійні повідомлення Qa = { U1, U2, … , | Qa | };
- діагностичні повідомлення Qd = { U1, U2, … , | Qd | } .

Інформаційні повідомлення : " Де знаходиться користувач і наступні кроки " .
Аварійні : " Задача може бути виконаною за деякими обмеженнями системи " .
Діагностичні : " Помилка від помилки у вихідних даних і місце її виникненя " .
Всі ці типи можуть формуватися в кожному функціональному операторі . Таблиця повідомлень : (строки - функціональні оператори , стовбці - повідомлення) .



Фn

F = 1 1 1

При виконанні кожного функціонального оператора заповнюється маса повідомлень . Це робить функціональний оператор , потім інтерпретатор переглядає цю строку , і , якщо вони є , виводить їх . Тобто можна легко настроюватись на будь-яку мову . Як мінімум повинно бути одне інформаційне повідомлення .

## РОБОЧЕ ПРОЕКТУВАННЯ .

Після розробки і відладки програм виникає питання ефективності проектування . Стиль проектування - основний фактор , який визначає ефективність програмування . Найгіршим стилем є прагнення до багаточисельних покрашень працюючої програми . Оптимізація програмного коду - задача компілятора , а не програміста .

Правила :

1. Не реалізовувати всі припущення по підвищенню ефективності до моменту , поки програма не буде працювати правильно .
2. Не жертвувати легкістю читання заради ефективності .
3. Не оптимізувати , якщо немає необхідності .
4. Якщо необхідно підвищити ефективність (робочої програми в реальному часі) , то добиватися ефективності можна відповідно після вимірювання часу роботи кожного програмного модуля (не більше 5 % процедур займає 90% часу) .
5. Для складних систем , як правило , найпростіші алгоритми .
6. Рекомендації по використанню коментаріїв :
   - уникайте великої кількості коментаріїв ;
   - коментуйте так , наче ви даєте відповідь на питання читача ;
   - фізично висувайте всі коментарії з тексту своєї програми ;
   - прокоментуйте всі змінні .

## ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .

Тестування - це єдиний спосіб перевірки правильності роботи програми , воно сприяє підвищенню надійності програми . Надійність неможливо внести до програми в результаті тестування , вона визначається тільки правильностю етапів проектування . Ймовірність того , що ми зможемо правильно спроектувати велику програму нескінченно мала . Роль тестування полягає в тому , щоб визначити місце знаходження небагатьох помилок , які залишилися в добре спроектованій програмі . Тестуванням неможливо виправити погано спроектовану програму .

Види роботи :

1. Testing (тестування) - процес виконання програми або її частини з метою знаходження помилок .
2. Proof (доказ) - спроба знайти помилки в програмі безвідносно до зовнішнього для програми середовища . Це формально математичний метод доказу правильності програми (1 - виведення стверджень , 2 - їх доказ) .
3. Vereficatijn (контроль) - спроба знайти помилки , виконав програму в тестовому або моделюємому середовищі .

4. Validation (іспит) - спроба знайти помилки , виконав програму в заданому реальному середовищі .
5. Sertification (атестація) - авторитетне підтвердження правильності програми , аналогічно до атестації приборів та обладнання .При тестуванні з метою атестації виконується порівняння з деяким раніше визначеним стандартом .
6. Debugging (відладка) - спрямована на встановлені природою відомі помилки та на їх виправлення .

Види тестування :

1. Тестування модуля або автономне тестування (контроль окремих програмних модулей в ізольованому середовищі) .
2. Тестування сполучень (контроль інтерфейсів між окремими модулями або підсистемами) .
3. Тестування зовнішніх функцій (контроль зовнішньої поведінки системи , яка визначається зовнішніми специфікаціями) .
4. Комплексне тестування (контроль або іспит системи по відношенню до вихідних задач тестування) .
5. Тестування прийнятності (перевірка відповідності програми вимогам користувача) .

## ЛІНГВІСТИЧНЕ ЗАБЕЗПЕЧЕННЯ ПРОЕКТУ .

Лінгвістичне забезпечення , мова - невід'ємна частина програмного продукту (тому що призначений для роботи користувача , а не програміста) . В САПР - системах - інженерно - орієнтоване спілкування . В 80-ті роки сформувалося поняття вхідної мови , зараз - інтерфейс користувача (це замінимо , тому що інтерфейс користувача - це графічна вхідна мова) . Мова (інтерфейс) - засіб спілкування між людиною і машиною .

Мови поділяються на види :

1. **Процедурно - орієнтовані мови** (послідовність дій визначається порядком слідування речень мови) . Командні мови або алгоритмічні мови - це процедурно - орієнтовані мови (наприклад , в текстовому редакторі існує меню) .
2. **Непроцедурні мови** (порядок обробки мови не визначений) . Це різні мови топологічного кодування .
3. **Компілятивні мови** (характеризуються перетворенням вхідних речень в машинний код , який потім виконується) .
4. **Інтерпретуючі мови** (припускають зчитування вхідних речень і виконання ви-значених в них дій) .

При створенні нового програмного продукту стоять питання і в створенні нових мов (інтерфейсів) . Це складне і трудоємке питання , тому що користувачі звикають до них , але вони застарівають і орієнтуватися на них вже складно , а вхідна мова , як правило , пов'язана зі структурою даних (результат компіляції - це формування нової математичної моделі) . Таким чином , якщо створювати нову систему , потрібно робити новий компілятор , тому що в нас нові структури даних , бо ми модифікуємо нашу мову . Мовами IHDL , ELF користуються тільки комп'ютери (передача даних мережею) , людина не може на них спілкуватися .

Форма запису вхідних мов

1. Повинна скорочувати або повністю виключати необхідність будь-яких знань користувача в області програмування .
2. Повинна пропонувати користувачеві способи запису даних у вигляді , найбільш звичному для нього .
3. Повинна сприяти скороченню часу підготовки даних .

МОДЕЛІ ПРОЕКТУВАННЯ ДАНИХ І ЗНАНЬ.

Відповідно до аспекту семантичності розрізняють:

- синтаксичні моделі;
- семантичні моделі;

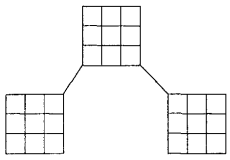
*Синтаксичні моделі* - це такі моделі, які характеризуються тим, що їх можливості обмежуються засобами зовнішньої синтаксичної структурізації даних. Цій структурізації підкоряється кожна реалізація або екземпляр того чи іншого типу даних Семантичні властивості даних представлені лише в тому розумінні, в якому вони можуть бути ассоційовані з синтаксичними структурами.

Відповідно до особливостей синтаксичної структурізації розрізняють

- табличні моделі;
- графові моделі;

*Табличні моделі* - це такі моделі, всі дані в яких представляються однаково. Тому екземпляр кожного типу представляється у вигляді рядка в таблиці.

В *графовій моделі* розрізняють власне дані, що мають табличне представлення і які розглядають як вершини графу, і зв'язки, що зв'язують таблиці в деякий граф:



Серед класичних моделей даних до табличних відноситься базово-реаліційна модель, а до графових - ієрархічна і сітьова.

В сітьовій моделі на зв'язки не накладаються ніякі обмеження.

В сучасній СУБД підтримуються ієрархічні і сітьові моделі, в вузлах яких створюються таблиці за базово-реаляційним методом.

В синтаксичних моделях можуть реалізуватись різні методи маніпуляції, даними. Наприклад, навігаційні методи передбачають детальне управління пошуком або селекцією необхідних даних.

Результатом виконання кожної команди є єдиний екземпляр шуканого типу або типу, що лежить на шляху шукомого.

Методи пошуку - це специфікаційні методи. Вони передбачають завдання критерію селекції даних

. Результатом виконання специфікаційного критерію вибору є множина всіх екземплярів у відповідному критерії.

*Семантичні моделі* - це такі моделі, що характеризуються наявністю засобів специфікації семантичних властивостей об’єкту. Семантичні властивості об'єкту визначаються через відношення:

- відношення агрегації;
- відношення асоціації;
- відношення узагальнення і т.д.

Наслідком специфікації відношень є обмеження від цілісності по включенню й усуненню елементів із баз даних, а також певні правила конструювання взаємозв'язаних типів.

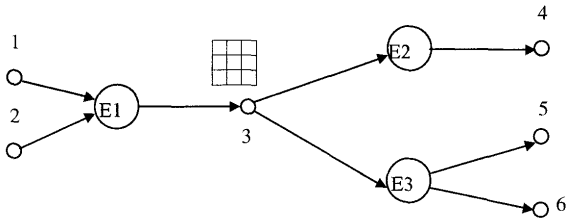
Поява семантичних моделей вслід за синтаксичними відображає тенденцію інтелектуалізації комп'ютерних систем і забезпечує природний перехід від баз даних до баз знань

бази знань, будучи логічним продовженням баз даних, активно використовують механізми семантичної інтерпретації

Перехід від даних до знань призводить до ускладнення інформаційних структур, які в результаті набувають таких характеристик:

1. *Інтерпретування* - коли дані можуть змістовно інтерпретуватись лише відповідними програмами Знання відрізняються тим, що можливість змістовної інтерпретації є не завжди

Якщо ввести відношення зв'язності між елементами множини, тоді ми вводимо семантику




Кожному вузлу схеми можна поставити у відповідність таблицю, що являє собою стани цього вузла у відповідні моменти часу. Тобто можна в модель вкласти семантичні властивості, і семантика буде накопичуватись в міру розвитку модельного експерименту

2. *Наявність класифіцируючих відношень* - коли в базах даних, не дивлячись на розмаїття форм зберігання даних, можливості компактного опису можливих зв'язків між різними типами даних обмежені. При переході до знань в базах знань встановлюються багаточисельні відношення між одиницями знань, такі як елемент-множина, тип-підтип, ситуація-підситуація Цей принцип називається принципом *наслідування,* тобто наслідування властивостей окремих інформаційних елементів скорочує об'єм інформації і ускладнює процес завдання відношень між окремими елементами

3. *Ситуативні зв'язки* - визначають ситуативну сумісність окремих подій, факторів, що зберігаються в пам'яті і дозволяють будувати процедури часового аналізу знань

Науковий напрям, в рамках якого вирішуються наукові і практичні задачі організації баз знань називається штучним інтелектом Дослідження, що ведуться в області штучного інтелекту зводяться до таких чотирьох напрямів.

1) Представлення знань і робота з ними Тобто створення моделей та мов представлення знань, а також програмних і апаратних засобів їхнього перетворення

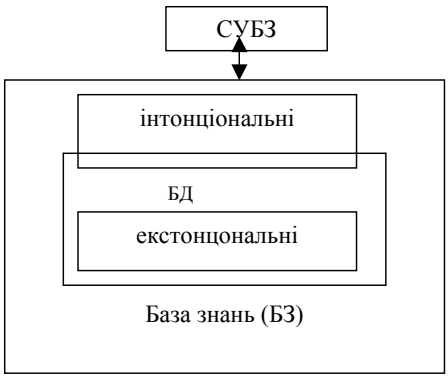
2) Планування доцільної поведінки Тобто створення методів формування цілей та вирішення задач планування дії автоматичного обладнання, функціонуючий в складному зовнішньому середовищі (задачі управління роботами)

3) Спілкування людини з комп'ютером Тобто створення так званого "дружественного пользовательского интерфейса".

4) Розпізнавання образів і навчання, яке включає сприйняття зорових, звукових та інших видів інформації і її оброблення, і адоптацію штучних технічних систем до середовища шляхом навчання.

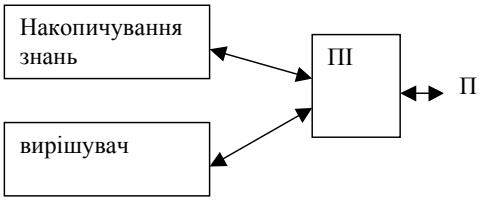
Для вирішення цих задач ми повинні сформувати базу знань Вона повинна

включати екстонціональні знання (дані) та інтонціональні знання Повний об'єм ектонціональних та інтонціональних знань утворюють базу даних




Система штучного інтелекту при всіх відмінностях, пов'язаних з різним призначенням, архітектурою, містять три обов'язкових елемента (блока)

- блок накопичення знань,
- вирішувач,
- інтерфейс користувача і користувач;




Блок накопичування знань містить засоби структурізації і управління знаннями

Вирішувач - моделює інтелектуальну діяльність людини

Інтерфейс користувача забезпечує доступ користувача до цього середовища штучного інтелекту

**Типи знань.**

В системах штучного інтелекту представлення знань здійснюється за допомогою знакових або семіотичних систем


Екстонціональні знання являють собою дані, що характеризують конкретні об' єкти, тобто їх стани і значення параметрів у конкретні моменти часу (екстонціоналом інтенціонала комп'ютера е його марка).

В характеристиках сіміотичних систем виділяють три аспекти.

1 синтаксичний,

2 семантичний;

3 прагматичний;

*Синтаксис* описує внутрішню будову знакової системи, тобто правила побудови і перетворення знакових виразів.

*Симантика* визначає відношення між знаками та їхніми концептами, тобто задає смисл конкретних знаків

*Прагматика* визначає знак з точки зору конкретної сфери його застосування Наприклад, кожен символ може бути елементом алфавіту, але ще може бути використай як орнамент зображення складних графічних систем

Відповідно до перелічених аспектів симіотичних систем виділяють три типи знань

1. Синтаксичний,

2. Семантичний;

З Прагматичний;

*Синтаксичні знання* характеризують синтаксичну структуру описуємого об'єкта аба явища, яке не залежить від змісту і смислу використаних при цьому понять

*Семантичні знання* містять інформацію, безпосередньо пов'язану зі значеннями та смислом описуємих об' єктів або явищ

*Прагматичні знання* описують об'єкти та явища з точки зору вирішуваної задачі, тобто з урахуванням діючих в ній специфічних критеріїв

**Моделі представлення знань.**

Моделі представлення знань розбиваються на

- декларативні;
- процедуальні

Декларативні моделі передбачають представлення знань як деяких сукупностей загальнозначимих тверджень.

В процедуальних моделях знання представляються у вигляді сукупності процедур над станами в предметній області.

З точки зору моделей представлення знань найбільше розповсюдження отримали моделі, що базуються на теорії Фрейма, семантичних сітей і логічних моделей

Вцілому виділяють чотири класи моделей представлення знань

1. Семантичні сіті.

2. Система Фрейма.

3 Продукціонні системи

4 Логічні моделі або мови

В основі всіх моделей представлення знань лежить математична логіка

*Математична логіка* - це засіб міркування та аналізу знань

*Міркування* - це ланцюжок взаємопов'язаних умовиводів

В залежності від характеру взаємозв'язку міркування бувають

- індуктивні,
- дедуктивні.

Якщо взаємозв'язок умовиводів побудована на індукції, то міркування - індуктивні

*Індукція* - це умовивід від конкретних фактів до деякої гіпотези. Тобто, до утвердження узагальнюючих абстракцій типових для всіх разом і кожного окремо факту

Індуктивний умовивід передбачає обговорення або введення критеріїв для виправдання достовірності гіпотези. Наприклад:

x+2y-5=0 - це факт, що говорить про те, що це рівняння - пряма.

2x-3y+10=0 - це другий факт.

Можна сформулювати гіпотезу:

ax+by+c=0 - це рівняння прямої.

Доведемо правильність даного твердження. Пошук доказу істинності висунутої гіпотези - найскладніша задача.

При дедуктивному міркуванні (обчисленні) взаємозв'язок міркувань базується на дедукції. При цьому ланки в ланцюгу умовиводів пов'язані відношенням логічного слідування.

*Дедукція* - це умовивід від узагальнюючих абстракцій до фактів або до проміжних абстракцій меншого рівня.

Початком або посилками дедукції є деяка сукупність узагальнюючих абстракцій, а *кінцем* - висновування, яке представляється як питання-факт або як питання-абстракція. Наприклад:

всі люди - смертні - це узагальнююча абстракція;

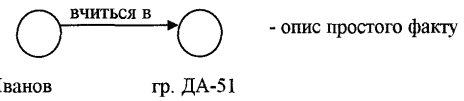
я людина - абстракція меншого рівня;

висновок, отже я - смертний.

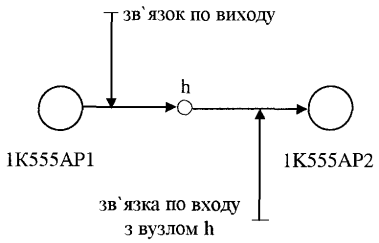
Сукупність логічного слідування полягає в тім, що істинність висновування випливає тільки з одночасної істинності всіх і тільки всіх посилок при всіх можливих замінах абстракцій факторами. Істинність всіх посилок при всіх можливих варіантах заміни останніх фактами встановити практично неможливо. В зв'язку з цим дедуктивне міркування завжди доповнюється формальними правилами виводу, які дозволяють спростити дедуктивний вивід, не порушуючи поняття логічного слідування, і ці правила називаються *інструментарієм аналізу достовірності міркування.*

**Семантична мережна модель.**

Поняття семантичної сіті базується на ідеї про те, що людська пам'ять формується через асоціації між поняттями. В базових функціональним компонентом семантичної сіті служить структура з двох вузлів і зв'язуючих їх дуг. Кожен вузел представляє деяке поняття, а дуги є відношенням між парами понять. Можна вважати, що кожна з таких пар представляє собою факт. Розглянемо приклад семантичної сіті:



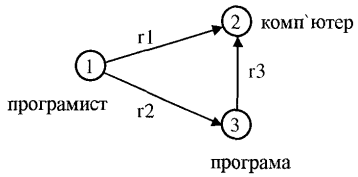
Другий приклад: схема електрична принципова -




Відношення між вершинами одного типу ми забороняємо Дуга в семантичній сіті має напрям. Завдяки цьому між поняттями в рамках певного факту формується відношення суб'єкт-об'єкт. Так як вершини семантичної сіті (або вузли симантичної сіті) з'єднуються з будь-якою кількістю інших вершин чи вузлів, то формується сіть фактів. Семантичні моделі є об'єктноорієнтованими (кожна вершина є об'єкт зі своїми атрибутами, дуги сіті формуються у вигляді покажчиків в структурі об'єкта). Дуги формують відношення зв'язності між об'єктами в явному вигляді, входячи в перелік атрибутів.
Виділяють чотири види зв'язку між об'єктами:

- класифікація;
- агрегірування;
- узагальнення;
- асоціація.

Основна ідея моделювання за допомогою семантичних сітей полягає в тому,- що модель описує реальні об'єкти і зв'язки між ними прямим способом, а це суттєво спрощує доступ до знань. Пересуваючись по дугам семантичної сіті від поняття до поняття, можна отримати детальну інформацію про будь-який об'єкт моделюємого об'єкту реального миру. Розглянемо приклад семантичної системи:




Є три об'єкта. Вводимо відношення між цими об'єктами. r1 -
відношення - сів за комп'ютер;
r2 - відлагодити;
r3 - загружений в (комп'ютер).
Використання семантичних сітей дозволяє представити в базі знань знання про будь-яку предметну область.

Позитивні якості семантичної сіті це:

- наявність засобів для опису обмежень (сам спосіб побудови задає обмеження);
- опис зв'язків між об'єктами;
- визначення операцій над об'єктом. Негативні якості семантичної сіті це:

- довільна структура сіті і різні типи вершин і зв'язків ускладнюють постановку модельного експерименту;

- семантична сіть в явному вигляді - такий атрибут як час. Тому для прив'язки різних подій, виникаючих в цій сіті, до часу створюються власні механізми, які в сіть безпосередньо не підключені.

Сіті бувають:

- простими;
- ієрархічними, коли окремі вершини розкладаються на підсіті. Можемо формувати сіті з множини вкладених підсітей. В цьому випадку виникають відношення не тільки між об'єктами сіті, але й між просторовими об'єктами.

**Фреймова модель.**

*Фрейм* - це фрагмент знання, призначений для опису стандартних ситуацій.

**[< >, < >, < >]**

Фрейми мають вигляд структурійованих наборів компонентів, які називаються об'єктами або слотами.

Кожен слот визначається іменем і набором правил:

F[< N,P>, <>,<>]

Параметри слота або об'єкта називаються атрибутами слота або об'єкта. Як атрибути можуть використовуватись посилання на інші об'єкти.

Фреймова модель - це модель низького інформаційного рівня. Вони використовуються в економіці, в електротехніці (накладаються на семантичні сіті, тобто створюються два типи моделей для різних операцій).

Позитивні якості фрейма:

- в нього можна включати інформацію по умовчанню, тобто атрибути окремих об' єктів можуть мати фіксовані початкові значення;
- забезпечують вимоги структурійованості і зв'язності.

Фрейм є єдиним способом представлення знань для деяких областей.

**Продукційні системи.**

Система продукцій утворюється множиною правил продукції. Ці правила задають певні дії при виконанні заданих умов. Так як водночас може виконуватись декілька умов, то повинна бути певна стратегія їхнього вибору. Правила продукції близькі за смислом до аплікації. Якщо виконується умова *a,* то виконується /?:

$$\alpha \xrightarrow{\text{якщо-то}} \beta$$

P1AP2AP3...PN^J8

/З - висновування або дія, які мають місце при істинності даної функції такої моделі представлення для використання в експертних моделях.

Представлення знань у вигляді продукційних систем використовується, коли знання приймають форму: <причина> —><наслідок>.

В будь-яких продукційних системах база даних складається з множини правил продукції (бази правил) і бази фактів. Тобто:

П={p1,p2,...,pт}

А={a1,a2,...,aп}

При цьому в розвинутих продукційних системах можливий розвиток поповнення або модифікаційної бази фактів (в результаті останнього формування ланцюжка фактів p=al,a2, ,aj можемо породити новий факт am), тобто при формуванні деякого ланцюжка фактів породжується новий факт

Можливі випадки, коли правило продукції пов'язано з виконанням будь-якої процедури на am повідомленні про закінчення цієї процедури В цьому випадку умови і дії є утвердженням даних, тобто продукційна система дозволяє не тільки визначити умови виконання деяких фактів, але й задавати деякі дії, де це необхідно

Недоліками продукційних систем є відсутність внутрішньої структури, тобто завдання заданих правил упорядкування кроків. Тобто в результаті довести логічні висновки, зроблених на основі продукційних систем неможливо Тому системи, побудовані на основі продукційних систем, основані на висновках експертів ( експертні системи)

Позитивні якості продукційної системи

- модульність або дискретність організації знань,
- незалежність правил продукції, тобто легкість модифікації знань . шляхом добавлення і вилучення нових правил;
- можливість використання різних стратегій управління послідовного логічного висновку

**Логічні (мовні) моделі**

Будуються на основі язика логічного висновку, тобто може бути представлена множиною базових символів (алфавітів язика), множиною нетерменірованих символів, множиною породжуваних правил (правил побудови ланцюжків символів у цьому язику), правилами висновку, певною множиною відношень між правильно побудованими формами

L={T,N,P,F}.

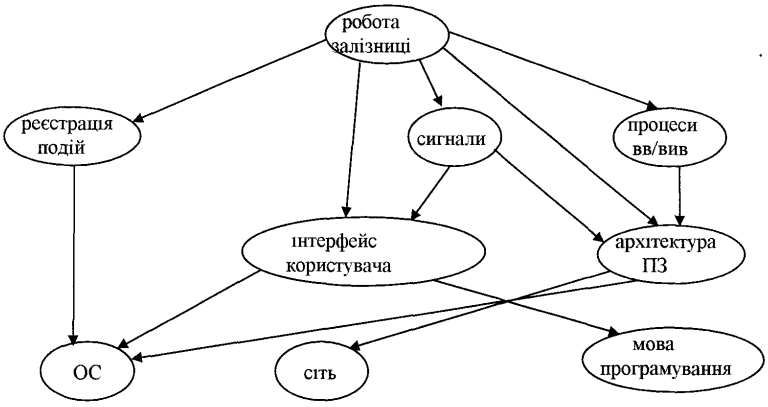
ОБ'ЄКТНО-ОРІЄНТОВАНИЙ АНАЛІЗ.

Об'єктно-орієнтований аналіз - це перший крок на шляху об'єктно-орієнтованого проектування.

Об'єктно-орієнтоване проектування - це об'єктно-орієнтований аналіз, на основі якого ми формуємо об'єктно-орієнтований опис об'єктної області. Сформувавши об'єктно-орієнтований опис ми формуємо об'єктно-орієнтоване проектування'


Вихідні дані для аналізу:

Приступаючи до розв'язання задачі об'єктно-орієнтованого аналізу, системний аналітик повинен виділити в вихідній предметній області декілька чітко визначених областей або доменів. Приступаючи до роботи, аналітик бачить:




Ми визначили зв'язки між підзадачами Кожен домен повинен розглядатися як самостійний інформаційний простір населений власними концептуальними сутностями або об'єктами. Тому в автоматизованій системі управління залізницею домен "роботи залізниці" населений такими концептуальними сутностями' вагон, поїзда, стрілочники Кожен домен може існувати більш чи менш незалежно від інших (наприклад, поїзда, вагони можуть існувати бедз вікон) Деякі домени малі, тобто кількість об'єктів в них невелика, і вони можуть розглядатись як єдине ціле Крупні домени (велика кількість об'єктів) розбиваються на підсистеми.

| | робота залізниці | | | реєстрація подій | інтерфейс користувача | |
|---|---|---|---|---|---|---|
| | відправлення поїздів | рух поїздів | управління коліями | | управ. вивод. | управ вводом |
| інформаційна модель | | | | | | |
| модель станів | | | | | | |
| модель процесів | | | | | | |

Для кожної підсистеми створюється 3 моделі
- інформаційна модель
- модель станів
- модель процесів

Після того, як система розбита на домени і підсистеми, починається аналіз Кожна система або підсистема аналізується незалежно одна від іншої в три етапи: 1. Інформаційне моделювання

2 Моделювання станів

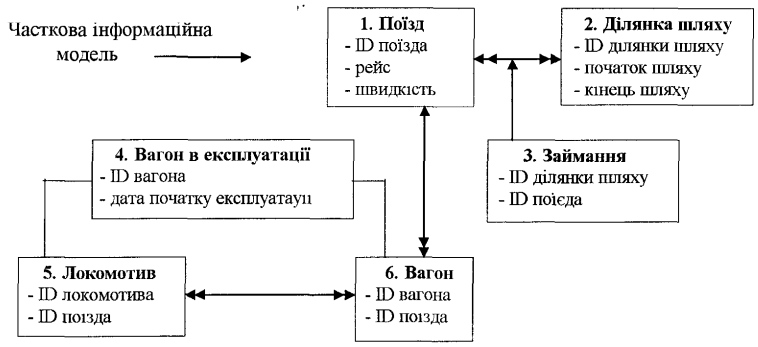
3 Моделювання процесів

CASE - програмні системи для автоматизації програмування

## Інформаційні моделі

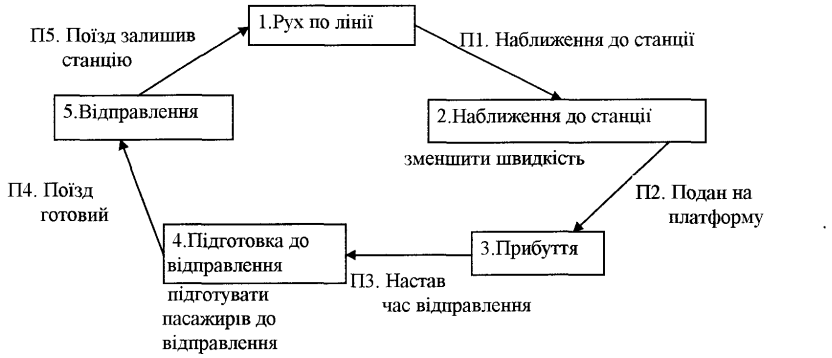
Мета етапу інформаційного моделювання полягає в тому, щоб ідентифікувати концептуальні сутності або об' єкти, які складають підсистеми аналізу.

Беремо підсистему руху поїздів:




## Моделі станів

Модель станів описує поведінку об'єктів в створених інформаційних моделях в часі. Кожен об'єкт і зв'язок між об'єктами можуть мати життєвий цикл або організовану систему поведінки (об'єкт поїзд рухається по залізниці, під'їжджаючи до станції, сповільнює рух, зупиняється, відкриває двері вагона і т.д., коли він готовий до відправлення, то подає




сигнали). Тобто є ряд ситуацій, через які повинен пройти кожен об' єкт

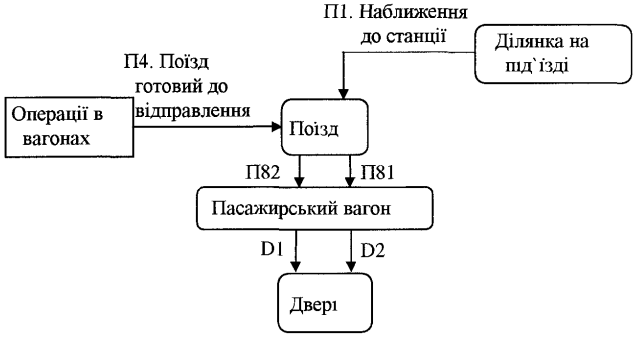
Життєвий цикл об єкта формалізується у вигляді множини станів і подій.

Стани являють собою положення або ситуацію об'єкта, в яких використовуються певні фізичні закони, правила і лінії поведінки.

Події являють собою інцидент, який примушує об' єкт переходити з одного стану в інший. Моделі станів формуються для кожного об'єкта і зв'язку, які мають інтересуючу нас динамічну поведінку. З кожним станом пов'язана деяка діяльність. Ця діяльність, яку називають дією, відбувається в той час, коли об'єкт досягає стану.

Для того, щоб досягти согласованої поведінки різних об'єктів, моделі станів взаємодіють між собою за допомогою подій.

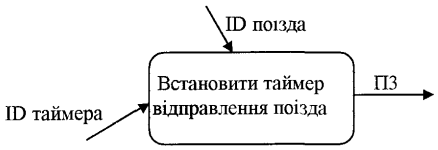
Наприклад.
модель стану поїзда може породжувати подію для об'єктів, з яких складається вагон:




П81,П82 - відкривання та зачинення дверей.
Події, які описують поведінку кожного об' єкта, приймають участь і в синхронізації поведінки об' єктів
В моделі взаємодії об'єктів створюються взаємодії між підсистемами і деякі події, що беруть участь в синхронізації взаємодії між підсистемами

**Моделі процесів**

Сформувавши моделі станів кожного об'єкта, для кожної дії, визначеної в різних станах дається його повний функціональний опис. Цей опис має такий вигляд.




Для кожної дії, визначеної в моделі станів, визначається звідки беруться дані для реалізації даної дії. Тобто в результаті об'єктно-орієнтованого аналізу ми формуємо такий документ

1 Опис і обгрунтування виділеного домена для досліджуваної предметної області 2. Опис і обгрунтування виділених підсистем для складних доменів
3 Інформаційні моделі для кожної підсистеми і нескладного домена
4 Опис об' єктів та їх атрибутів для кожної інформаційної моделі, включаючи опис зв'язків кожного об'єкту
5 Моделі станів для кожного об'єкта
6 Модель взаємодії об'єктів
7 Модель взаємодії підсистем
8 Моделі процесів для кожної дії в моделі станів об' єктів

## Концепції інформаційного моделювання

### Поняття об'єкта

В об'єктно-орієнтованому аналізі *об'єкт* - це така абстракція множини предметів реального світу, в якій:

1 Всі предмети в цій множині або екземпляр мають одні й ті ж характеристики..

2 Всі екземпляри підкорені і узгоджуються з одним й тим самим набором правил поведінки

Кожен об'єкт в моделі повинен бути забезпечений унікальним і значимим ім'ям, а також унікальним ключовим ітералом ( короткою формою ім'я )

У великій моделі для організації документації об'єкти, крім того, повинні бути пронумеровані


Спеціального терміна для визначення сукупності екземплярів деякого об'єкта в об' єктно- орієнтованому програмуванні не існує

Об'єкти ідентифікуються шляхом розглядання концептуальної сутності, пов'язаної з аналізированим управлінням.

Ряд задач припускають створення об'єктів, що мають фізичний характер. Інші задачі породжують породжують абстрактні об'єкти.

Якщо класифікувати, то більшість об'єктів створених в об'єктно-орієнтованому аналізі відносяться до таких категорій:

1. Реальні об' єкти
2 Ролі
3 Інциденти
4 Взаємодія
5. Специфікація

*Реальні об'єкти* - це абстракції фактичного існування деяких об'єктів у фізичному світі.

*Ролі* - це абстракції мети або призначення людини, частини обладнання, організації

*Інциденти* - це абстракція чогось, що зчинилося або сталося

*Об'єкти взаємодії* - це об'єкти, що одержуються із співвідношень між іншими об' єктами

*Специфікації* - використовуються для представлення правил, стандарту або критерію якості

### Опис об'єктів

Опис - це коротке інформативне твердження, яке дозволяє встановити чи є реальний предмет екземпляром або ні.

Опис об'єкта повинен бути основан на абстракції, яка точно пояснює подібність предметів реального світу. Наприклад, є об'єкт, що знімає показання приладу, - це штатний

службовець (електрик), який може прочитати значення, що відображаються на електричних приладах і в цей час виконує це завдання.

### Атрибути об'єкта

Атрибут - це абстракція однієї характеристики, яку мають всі абстраговані як об'єкт сутності. Кожен атрибут забезпечується іменем унікальним в межах об'єкту.

Щоб звернутися до атрибута пишуть:

<ім'я об'єкта>.<ім'я атрибута>

Наприклад:

<літак>. <тип>

<студент>.<прізвище>

Атрибутів у об'єкта може бути декілька. Для будь-якого екземпляра об'єкту атрибут приймає значення. Для атрибута може визначатись діапазон можливих значень.

Ідентифікатор об'єкта - це множина з одного чи більш атрибутів, значення яких однозначно визначає кожен екземпляр об'єкта.

Об'єкт може мати декілька ідентифікаторів При цьому кожен ідентифікатор може бути побудован з одного чи декількох атрибутів. Якщо об'єкт має декілька ідентифікаторів, то один з них вибирається привілейованим.

При розробленні моделей об'єктно- орієнтованого аналізу об'єкт зображується графічно у вигляді прямокутника, що містить ім'я, номер і ключовий ітерал об'єкта, а також імена атрибутів. Атрибути, що складають привілейований ідентифікатор об'єкта відмічаються * або ніяк. А решта відмічаються • зліва. Можливо текстове представлення об'єкта. Привілейований ідентифікатор в текстовому представленні об'єктів підкреслюється.

Літак (ID літака, розмах крила)

### Типи атрибутів

1. Описові
2. Вказуючі
3. Допоміжні

Описові атрибути представляють факти, внутрішньо притаманні кожному екземпляру об'єктів. Якщо значення описового атрибута змінюється, то це говорить-про те, що деякий аспект екземпляра змінився, але сам екземпляр залишився старим.

Вказуючі атрибути використовуються для ідентифікації екземплярів. Вказуючі атрибути використовуються, як ідентифікатори, але бувають випадки

Допоміжні атрибути використовуються для зв'язку екземпляра одного об'єкта з екземпляром іншого.

### Опис атрибутів

Опис атрибутів залежить від типа атрибута.

Для описових атрибутів опис повинен встановлювати реальну характеристику, яка абстрагується, як атрибут.

Опис діапазону значень надається одним з декількох способів:

1. Переліченням всіх можливих значень, які атрибут може приймати.
2. Посиланням на документ, що приводить можливі значення
3. Формулюванням правила, яке визначає які значення допустимі

Для вказуючих атрибутів опис встановлює форму указання, організацію, яка призначає указання та степінь, в якій ім'я атрибута може використовуватись, як частина ідентифікатора

Для допоміжних атрибутів опис повинен вказувати реальне відношення, що встановлюється атрибутами.

**Правила атрибутів**

Інформаційна модель грунтується на реалідійному представленні даних, тобто представленні даних у вигляді відношень між ними. Тому вводяться наступні правила для атрибутів об'єктів інформаційних технологій:

1 Забороняється створення об' єктів, в яких маються атрибути, що повторюються

2 Атрибут не повинен мати ніякої внутрішньої структури, тобто його неможна розкласти на атрибути більш низького рівня

3 Кожен атрибут, що не е частиною ідентифікатора характеризує весь об' єкт, а не його частиною

4 Кожен атрибут, що не є частиною ідентифікатора характеризує екземпляр об' єкта, який вказаний ідентифікатором, а не визначає інші атрибути ідентифікатора

**Зв'язки між об'єктами**

Зв'язок - це абстракція відношень, які виникають між різними предметами в реальному світі

Реальні предмети, що беруть участь у відношенні, повинні бути абстрагованими, як об'єкти Кожен зв'язок в моделі задається парою імен, які описують цей зв'язок із перспективи кожного об'єкта, який бере участь в зв'язку. Наприклад, поїзд включає вагони, вагони належать поїзду.

Існують три функціональні види зв'язку:

1 Один до одного

2. Один до багатьох

3 Багато до багатьох

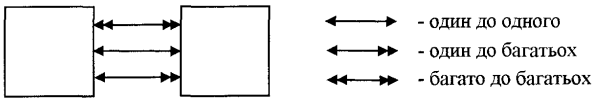
Ці функціональні види зв'язку називаються безумовними формами

Зв'язок один до одного існує коли один екземпляр одного об'єкта пов'язан з єдиним екземпляром іншого

Зв'язок один до багатьох існує коли один екземпляр деякого об'єкта пов'язан з одним чи більш екземплярами іншого, і кожен екземпляр другого об'єкта пов'язан тільки з одним екземпляром першого

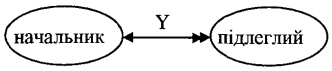
Зв'язок багато до багатьох існує коли один екземпляр деякого об'єкта пов'язан з одним чи більшою кількістю екземплярів іншого, і кожен екземпляр другого об'єкта пов'язан з одним чи більш екземплярами першого

Графічно це виглядає так



### Умовні зв'язки

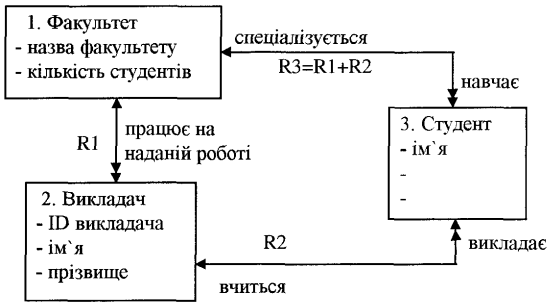
Основний зв'язок визначає, що можуть існувати такі об'єкти, які не зв'язані Ці моделі помічаються літерою Y після відповідної стрілки зв'язку:




### Опис зв'язків

Опис кожного зв'язка включає:
- ідентифікатор зв'язку;
-вид зв'язку;
- формулювання того, як зв'язок був реалізован;

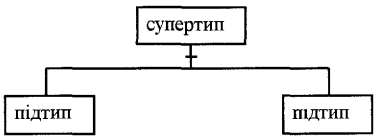
Можливе формулювання імені зв'язку з точки зору кожного об'єкта, що бере участь.
Деякі зв'язки утворюються, як наслідки існування інших зв'язків. Наприклад:




Відношення існують між викладачем і студентом, але це викликає відношення між студентом і факультетом.

### Підтипи і супертипи

Поняття підтипа і супертипа аналогічно до поняття ієрархії. Підтип - це об' єкт, що має спільні атрибути з об' єктом супер типа. Атрибути, спільні для всіх об'єктів підтипу, знаходяться в об'єкті супертипа Об'єкти підтипу можуть мати свої додаткові атрибути.



**Життєві цикли об'єкта**

Закінчивши розробку інформаційної моделі, ми отримали її структуру, опис об'єктів і атрибутів, і опис зв'язків. Після закінчення розробки інформаційної моделі розробляються життєві цикли об'єктів. Будь-які об'єкти реального світу динамічні в часі (мають динамічну поведінку моделі). Можна казати, що кожен об'єкт має свій строк життя. Можуть бути запропоновані наступні параметри опису поведінки різних предметів:

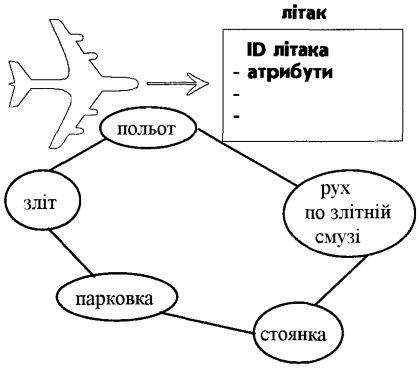
1. Об' єкти реального світу на протязі їх строка життя проходять різні стадії
2. Порядок еволюціонування об'єкта формує характерний відмітний признак поведінки цього об' єкта. Тобто будь-який об' єкт має свій порядок поведінки

З В будь-який сучасний момент часу реальний об'єкт знаходиться в одній єдиній стадії

4. Об'єкти еволюціонують від однієї стадії до іншої стрибкоподібне. Це залежить від рівня абстрагування процесу поведінки літака
5. В схемі поведінки об'єкта дозволені не всі еволюції між стадіями (деякі еволюції забороняються законами фізики)
6. Об'єкти еволюціонують в наслідок інциденту, тобто перехід від стадії до стадії діється в результаті інциденту, тобто інцидент є деяка подія.

Так як всі екземпляри об'єктів підкоряються одним й тим самим правилам поведінки, то, абстрагуючи об'єкт, ми абстрагуємо і спільну модель поведінки всіх його екземплярів.

Життєвим циклом об'єкта називається формалізований опис моделі поведінки об' єкта, що спільно використовується всіма екземплярами.

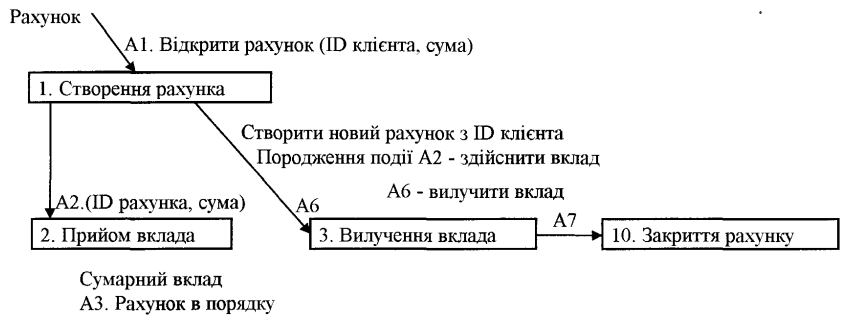



Життєвий цикл об' єкта - модель стану об' єкта.

Життєві цикли можуть описуватись як:

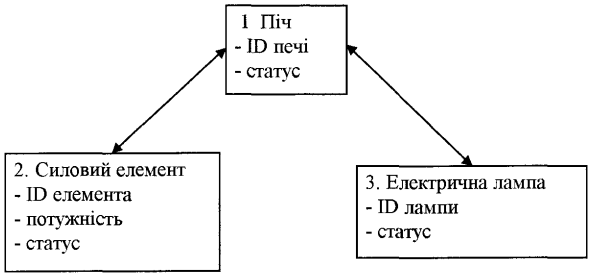
- діаграма станів
- у вигляді таблиці кінцевого автомата

Модель станів для об'єкта рахунка:




**Координація життєвих циклів**

Життєві цикли різних об' єктів координуються один з одним.

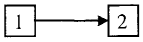



Кожен з цих об'єктів має свій життєвий цикл.

Множина станів, в яких може знаходитись кожен об' єкт, - це множина значень, що приймає статус ( його атрибутів). Поточний стан атрибута статус є поточний стан об'єкта. Стан - положення об'єкта, в якому виконується певний набір правил приписів або фізичних законів.

Кожному стану присвоюється унікальний номер. В моделі станів виділяються стани, в яких екземпляр об'єкта з'являється вперше і активізується. Такі стани називаються станами створення.

Перехід в стан створення позначається стрілкою, яка зв'язує якісь елементи станів:




Крім того, в моделі станів виділяються так звані заключні стани - стани, в яких екземпляр об'єкта припиняє своє існування (існувань в життєвому циклі може бути одне чи декілька). Заключність стана може представляти одну з таких ситуацій:

1. Екземпляр стає нерухомим (тобто він залишається в межах моделі, але не активен)
2. Екземпляр вилучається з програмної моделі

## Поточний стан

В будь-який момент часу екземпляри об' єктів можуть знаходитись в різних станах Ідентифікація поточного стану здійснюється через атрибут об'єкта, який називається статусом.

## Подія

Подія - це абстракція інциденту або деякого сигналу в реальному світі, який інформує про переміщення екземпляра об'єкта в новий стан. Абстрактна подія має чотири інтерпритації

- значення
- призначення
- мітка

- дані (тобто подія може нести якісь дані)

Значення міститься в короткій фразі, яка повідомляє, що діється в реальному світі
Призначення несе інформацію про моделі станів, яка ініціалізується або приймає дані подіє Використовується для того, щоб синхронізувати життєві цикли різних об'єктів (зв'язати один з одним).
Мітки спеціального призначення не мають Служать для ідентифікації самої події і сполучаються або значенням або призначенням
Дані служать для передачи повідомлення разом із самою подією і сполучаються зі значеннями або призначеннями.
Мітки подій повинні визначатися так, щоб всі події, які приймаються даним об' єктом, починалось з одного й того ж ітерала

## Дані подій

В програмній моделі події можна інтерпретувати, як сигнал повідомлень, який можна переносити Дані бувають:

1 Ідентифікуючі
2 Додаткові

Ідентифікуючі - атрибути, що визначають якісь параметри об' єкта.
Додаткові - дані, які треба перенести з одного стану в інший або передавати в якийсь поточний стан об' екта.
Існує ряд правил, які зв'язують дані, що передаються з подіями, з поточними станами об' єкта:

1 Правило тих самих даних

Всі події, що визивають перехід в певний стан, повинні нести одні й тіж типи даних Гарантує, що яка б подія не перевела об'єкт в стан, функціонувати він буде правильно.

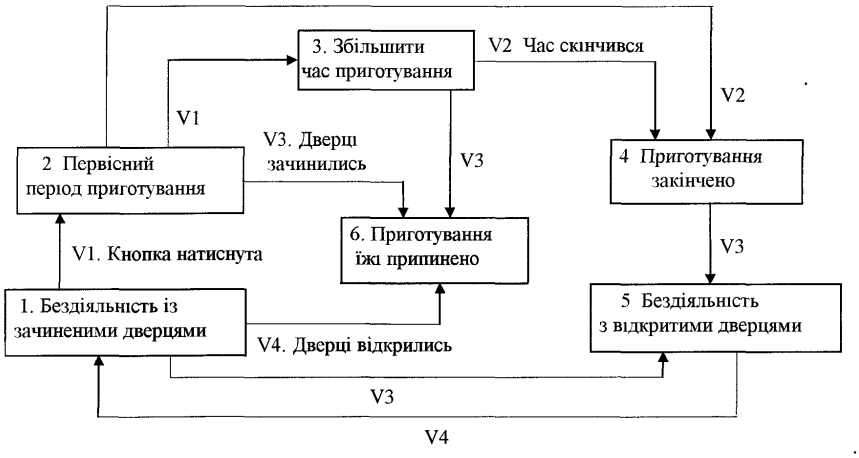
2 Правило стану не створень

Якщо подія викликає перехід стану, який не є станом створення об'єкту, то ідентифікатор екземпляра, до якого події використовуються, може бути відсутньою, але якщо програмні моделі активні декілька екземплярів одного й того ж об' єкта, то передача ідентифікатора екземпляра обов'язкова

3 Правило стану створень

Якщо подія викликає перехід стану створення нового екземпляра об'єкту, воно може не нести ідентифікатор екземпляра тільки в тому випадку, якщо цей ідентифікатор генерується в стан створення.

Один інцидент в реальному світі може бути абстрагрован в більш ніж одну подію Модель станів мікрохвильової печі.




Цей життєвий цикл відноситься тільки до

печі **Дія**

Дія - це діяльність або операція, яка повинна бути виконана екземпляром об'єкта при досягнені ним поточного стану.

З кожним станом обов'язково повинна бути пов'язана одна дія Використовуючи одну й ту саму модель станів, дії повинні бути визначені так, щоб вони могли бути виконані будь-яким екземпляром

Дії дозволяють виконати такі операції

1 Виконати будь-які обчислення

2 Породжувати події для будь-якого екземпляра об'єкта

3 Породжувати події для будь-якої зовнішньої системи або підсистеми, включаючи аппаратну частину

4. Створювати, вилучати, встановлювати або обнуляти таймер

5 Читати і записувати атрибути інших екземплярів власного об'єкта

6 Читати і записувати атрибути інших екземплярів інших об' єктів

Щоб гарантувати не суперечність моделі станів, на дії накладаються обмеження

- будь-яка дія може гарантувати не суперечність даних власного екземпляра об'єкта, тобто якщо модифікуються атрибути власного екземпляра, то атрибути всіх екземплярів, що залежать від нього, повинні бути перелічені
- якщо дія створює або вилучає екземпляр об'єкта, то вона може гарантувати, що будь-які зв'язки, включаючи ці екземпляри, або вилучаються або перебудовуються (перепризначуються)
- дія може гарантувати не суперечність екземплярів підтипів і супертипів (якщо виличається підтип, то можна перевірити можливі вилучення супертипа)

**Опис дій**

В процесі об'єктно-орієнтованого аналізу описуються в текстовій формі або на деякому псевдокоді, який вибирається розробником.

В процесі виконання об'єктно-орієнтованого аналізу треба зв'язати дії, події і час Для цього виконуються певні припущення про час

1 Тільки одна дія в моделі станів може виконуватись в конкретний час Якщо дія почата - вона повинна закінчитися раніше ніж обробиться нова подія.

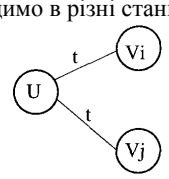
2 Дії в різних моделях станів можуть виконуватись одночасно.

3 Події ніколи не губляться.

4 Якщо подія породжена для екземпляра, який виконує дії, то подія не буде прийнята до завершення дії

5 Після обробки події зникають

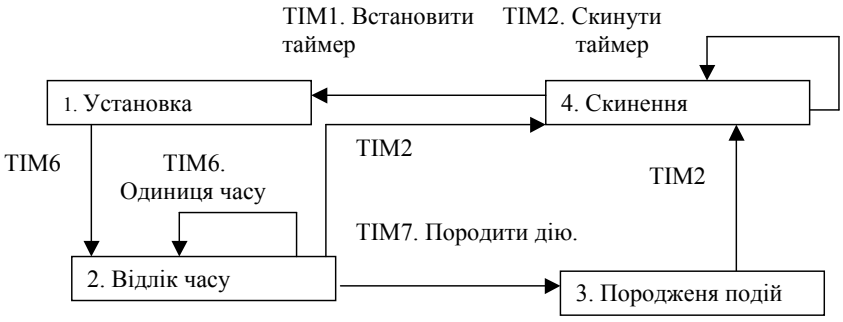
Якщо по одній і тій ж події переходимо в різні стани, то отримаємо диаграму




**Таймери**

Таймер - це механізм, який використовується дією для створення події через певний наперед заданий час Атрибутами таймера є:

1 Ідентифікатор таймера
2 Час до моменту формування події
3 Мітка події
4 Ідентифікатор екземпляра таймера.




**Загальні форми життєвого циклу**

Структура діаграми станів може бути будь-
яка Виділяють дві загальні форми.

1 Циркулярні життєві цикли (наприклад, життєвий цикл таймера) Об'єкти з циркулярним життєвим циклом можуть існувати на протязі всього часу існування роботи системи.

2 Життєвий цикл народження і смерті.

Екземпляри об'єкта, що підкоряються такому життєвому циклу, створюються або знищуються в межах часу життя програмної системи. Життєві цикли типа народження і смерті мають один чи більш станів створення екземпляра об'єкта і аналогічно один чи більш станів завершення екземпляра об'єкта.

**Формування життєвого цикла**

Зовнішня поведінка СВЧ зображена такою диаграмою станів:

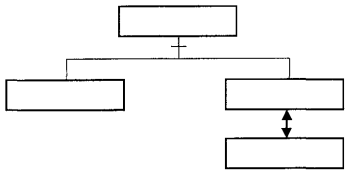

Точність опису поведінки того чи іншого об'єкта описується тими чи іншими задачами програмних прикладних систем.

При описі функціонування об'єкта треба дотримуватись наступних рекомендацій.

1. Визначити мету включення в діаграму станів кожного нового стану.
2. Кожен стан має контекст, тобто дії, які виконуються в ньому, можуть залежити від того, який стан був попереднім і який стан буде наступним.
3. Повний знімок функціонування об'єкта (докладна діаграма станів об'єкта) розглядається тільки в тому випадку, якщо від проміжних станів даного об' єкта залежать стани пов'язаних з ним інших об'єктів.

Правила об'єктно-орієнтованого аналіза передбачають, щоб різних станах не виконувались одні й ті ж дії.

**Життєві цикли для підтипів і супертипів**



При описі життєвих циклів окремо описуються життєві цикли для супертипів і для кожного підтипу. Причому успадкування признака викликає міграцію життєвих циклів При формуванні життєвих циклів дотримуються таких правил:

1 Якщо екземпляри об'єкта створюються або знищуються під час виконання програми, то життєвий цикл такого об'єкта обов'язково формалізується ідеталізується
2. Якщо екземпляри об'єкта набувають значення своїх атрибутів в різних станах життєвого цикла, то такий життєвий цикл теж обов'язково формалізується і деталізується.
3. Обов'язково формалізуються життєві цикли об'єктів типу обладнання. .
4. Обов'язково формалізуються життєві цикли об'єктів, які виконують розв'язання деякої математичної задачі.
5 Обов'язково формалізуються життєві цикли об'єктів, що вступають в зв'язки з іншими об' єктами в різних станах свого життєвого циклу.
6. Об'єкти специфікації як правило не формалізуються.
7. Життєві цикли об'єктів, що представляють собою засоби обслуговування, як правило не формалізуються.

### Облік відказів або помилкової поведінки об'єктів в процесі об'єктно-орієнтованого аналізу

Для більшості програмних систем необхідно передбачати не тільки правильну поведінку об'єктів, але й аномальну поведінку Аномальна поведінка повинна бути формалізована в тій самій моделі станів Для цього в моделі станів додають стани, події і дії, які відповідають аномальній поведінці об'єкта. Це робиться для можливості обліку якихось подій, що виникають з-за, наприклад, збою обладнання

При цьому враховуються такі помилкові події.

- відказ або збій обладнання (втрата потужності, заїдання перемикача, зростання температури і т д.);
- помилки персоналу (подача помилкових команд),
- відкази датчика для різних програмне управляючих систем,
- відкази силових приводів,
- відкази, пов'язані з часом, тобто порушення часових інтервалів

### Аналіз зв'язків між об'єктами

Відношення між об'єктами (предметами) в реальному світі розвиваються в часі Відношення між об'єктами формалізуються у вигляді зв'язків. Формалізація відношення в інформаційній моделі представляє собою її статичні описи, які не відображають факта існування цього зв'язку в часі, тобто зв'язок як таковий між об'єктами може бути динамічним. Інформаційна модель не відображає того, які екземпляри об'єктів беруть участь в зв'язку

Так як зв'язок може бути динамічним, виникає поняття життєвого циклу зв'язку Життєвий цикл зв'язку або модель станів зв'язку описує поведінку типового невизначеного екземпляра зв'язку. Якщо зв'язок не має життєвого цикла, то він може обладати динамічною поведінкою за умови, що екземпляри зв'язку створюються і знищуються в межах повного життєвого циклу програмної системи, що аналізується

### Конкуруючі зв'язки

Вони виникають в тих випадках, коли декілька об'єктів претендують на встановлення зв'язків один з одним в один й той же момент часу

Проблеми конкуренції такого типу вирішуються за допомогою перетворення конкуруючих запитів в послідовний ланцюг запитів. В об'єктно-орієнтованому аналізі розробляться спеціальна модель станів зв'язків, яка враховує різні стратегії упорядкування конкуруючих запитів.

Основою моделі станів є так званий визначник стану зв'язку Визначник стану зв'язку аналізується монітором або керуючим обладнанням моделі станів Принцип упорядкування такий: кожен запит постачається атрибутом статусу. Отже відповідно до цього будується черга.

Існує загальний алгоритм упорядкування запитів в міру находження їх на монітор - взятий з загальної теорії програмування.

Поради для побудови моделей станів об'єктів та їх зв'язків

1 Досвід показує, що всі проблеми, пов'язані з діаграмами станів, пов'язані з помилками формування предметної області.

2. Мітки обов'язково повинні бути узгодженні. Для цього формується загальний список всіх подій в програмній системі і описується область дій кожні події.

3 Опис конкуруючих зв'язків
4. Крім виділення моделі життєвого циклу, для кожного об'єкта корисно будувати модель взаємодії об'єктів Із загального списку виділяють об'єкти, тобто на інформаційну модель треба накласти схему подій, що викликають активізацію цих об'єктів під час їх функціонування.