УДК 004.415

д.т.н. проф. **Лєнков С.В.** (ВІКНУ)
к.т.н. доц. **Грищак О.М.** (НУОУ)
к.т.н. с.н.с. **Охрамович М.М.** (ВІКНУ)
Сотніков Є.О. (ВІКНУ)

DOI: http://doi.org/10.17721/2519-481X/2024/83-08

АНАЛІЗ СУЧАСНИХ ТА ТРАДИЦІЙНИХ МЕТОДІВ ПРОЕКТУВАННЯ І ВИРОБНИЦТВА ВІЙСЬКОВОЇ ТЕХНІКИ З ВИКОРИСТАННЯМ АВТОМАТИЗОВАНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

В статті авторами виконано один із видів аналізу існуючих та перспективних методів проектування і виробництва військової техніки з використанням автоматизованих інформаційних систем. Відомо, що програмне забезпечення АІС має характерні ознаки складної системи і підпорядковується законам науки про складні системи. Сучасний етап розвитку складних інформаційних систем характеризується тим, що тенденція до різкого збільшення складності завдань управління все в більшому ступені виявляється в об'ємності програмного забезпечення систем управління технічними об'єктами, технологічними процесами і виробництвом. В роботі зроблено класифікацію програм, відповідно до SPE, яка, являє собою «модель деякої іншої моделі в рамках теорії абстрактного уявлення окремих об'єктів». Вона може бути віднесена до одного з трьох класів: S-клас − це програми, функції яких формально визначаються специфікаціями і випливають із них. Відповідність між входом і виходом є визначальною і будь-яка зміна, створює нову програму. P-клас − складають програми, прийнятність рішення яких оцінюється шляхом порівняння з реальною обстановкою, тобто програми цього класу повинні постійно змінюватися залежно від зміни навколишнього середовища, уточнення даних тощо. Програми E-класу самі стають частиною реалізації обстановки, яку вони моделюють, тобто в ще більшому ступені, ніж P-програми схильні до змін.

Розглянуті основні фази життєвого циклу АІС і визначені їх особливості: визначення, проектування, реалізації, впровадження, експлуатації.

Визначено концептуальну єдність підходів, що використані на етапах отримання системних специфікацій, проектування модульної структури та реалізації модулів, дозволила об‘єднати прийняття окремих рішень в єдину методику проектування коректного спеціального програмного забезпечення, основними відмінностями якої є: пошаровість, яка досягається використанням принципу абстрагування (ієрархії); об‘єднання процесу проектування з процесом моделювання, що досягається застосуванням графових моделей, які отримуються в процесі проектування, та можливістю використання кожного рівня ієрархії в якості моделі СПЗ; безперервність процесу проектування, яка забезпечується сумісністю моделей, які отримуються на різних етапах, участю системного програміста в розробці з начального етапу і можливістю початку кодування з першого етапу.

Ключові слова: автоматизовані інформаційні системи, системи управління технічними об'єктами, концептуальна єдність підходів, безперервність процесу проектування.

Вступ та аналіз останніх досліджень. Коли мова йде про якість програмного забезпечення автоматизованої інформаційної системи (АІС), об'єктом дослідження стають не стільки власне програми (тексти, що написані мовою, яка сприймається комп'ютером), скільки саме *математичне забезпечення* АІС, тобто сукупність методів, моделей і алгоритмів для розв`язання задач і опрацювання інформації з застосуванням обчислювальної техніки. Тому в

результаті розроблення математичного забезпечення в остаточному підсумку і з'являється сукупність програм, які реалізують задум проектувальника: сукупність функцій АІС, забезпечуючих функціонування комплексу технічних засобів і бажаний розвиток системи, тобто програмне забезпечення.

Програмне забезпечення АІС має характерні ознаки складної системи і підпорядковується законам науки про складні системи [1,2]. Сучасний етап розвитку складних інформаційних систем характеризується тим, що тенденція до різкого збільшення складності завдань управління все в більшому ступені виявляється в об'ємності програмного забезпечення систем управління технічними об'єктами, технологічними процесами і виробництвом. Обсяг опрацьованої інформації в системах цього типу зростає більш інтенсивно, ніж збільшення обсягів самого виробництва або підвищення його техніко-економічних показників [3-5].

Аналіз підходів до проектування АІС військового призначення. Розв'язання задач прискорення створення і використання вимог по підвищенню ефективності АІС на базі розробки пакетів практичних прикладних програм (типових алгоритмічних модулів), моделювання і управління технологічними процесами розроблення типових та уніфікованих елементів забезпечення, способів і методик їх розробки.

Задачі оптимізації вирішуються за допомогою моделей, тому їх ієрархія задач відповідає ієрархії моделей керованих підсистем відповідних комплексів. Автори виділили основні рівні ієрархії таких складних технологічних комплексів [5,6]:

- побудова моделей керування;
- вирішення задач керування, по рівню складності рішення;
- побудова ієрархії технічних засобів.

Перший вид визначає обробку інформації з метою ідентифікації керованих об'єктів.

Другий – характеризує процес аналізу рішень в системі керування і відповідає древу цілей системи.

При побудові системи керування найважливішою проблемою є синтез ієрархічної структури. Ці проблеми зводяться до вибору алгоритмів розв'язання задач, оцінки побудови структури, і розроблення правил їх формування.

Саме формалізація методів і алгоритмів побудови АІС є основою застосування сучасної теорії складних систем до проектування математичного забезпечення АІС[7-9].

Класифікація програм, відповідно до SPE, що була запропонована В.Турським [10], являє собою «модель деякої іншої моделі в рамках теорії абстрактного уявлення окремих об'єктів». Вона може бути віднесена до одного з трьох класів:

S-клас – це програми, функції яких формально визначаються специфікаціями і випливають із них. Відповідність між входом і виходом є визначальною і будь-яка зміна, створює нову програму;

Р-клас – складають програми, прийнятність рішення яких оцінюється шляхом порівняння з реальною обстановкою, тобто програми цього класу повинні постійно змінюватися залежно від зміни навколишнього середовища, уточнення даних тощо.

Програми Е-класу самі стають частиною реалізації обстановки, яку вони моделюють, тобто в ще більшому ступені, ніж Р-програми схильні до змін.

Звісно, що спеціальне програмне забезпечення СПЗ АІС відбиває деяку модель або піддається постійним змінам, або згодом утрачає свою корисність. При його синтезі практичні задачі управління включають екстраполяцію наслідків упровадження на перспективи розвитку системи. При конструюванні СПЗ в роботі [11] стверджується, що до програм типів Р та Е доцільно додати наступну структуру, що всі їх елементи будуть S-програмами. Тобто проектування СПЗ АІС в цьому можна уявити як розвиток специфікацій.

Традиційними технологіями виготовлення СПЗ систем реального часу тривалий час були інструментально-орієнтовані технології, засновані на розвинутих засобах для написання програм, але недостатньо обґрунтовані методологічно. До них варто віднести використання

процедурно-орієнтованих мов реального часу і проблемно-орієнтованих пакетів (пакетів прикладних програм).

Сукупність мови, програмних засобів трансляції, налагодження, документування та обслуговування програм в процесі їхньої розробки й експлуатації складають систему програмування – основу технології при використанні процедурно-орієнтованих мов. Мови виступають і як засіб формалізованого опису і як інструментарій для одержання кінцевого програмного продукту.

Основною якістю програм мов реального часу можна вважати наступні:

- доцільність управління поведінкою операційної системи розвинутими механізмами рівнобіжних обчислень і синхронізації гілок апарату синхронізації обчислювального процесу з зовнішніми подіями, тощо;
- використання досить потужного обчислювального апарата, широкого набору типів і спеціалізованих структур даних;
- використання достатня гнучких різноманітних методологій проектування і підтримку модульного і структурного програмування.

Пакети прикладних програм служать для заміни програмування конструюванням СПЗ з готових елементів і підрозділяються на: бібліотеки програм; системи програмування з використанням спеціалізованих мов; мета системи, що дозволяють генерувати ППП для конкретних застосувань по їх описах.

Добре знайомі традиційні методи проектування ПЗ виділяють два наступних: "униз" (аналітичні підходи) і "нагору" (синтетичний підхід) [10-13]. Проектування методом "униз" полягає у постановленні задачі і розвитку шляхом розбивки задачі на підзадачі. Розробка методом "нагору" починається з оцінки рівня базових понять і подалі розвиває формування більш абстрактних понять, у кінцевому рахунку приводячи до термінів яких легко висловити рішення всієї задачі.

Такі підходи не дають вирішення на кожному кроці, проте забезпечують повний спектр можливостей, які подаються базовим прошарком. Такий підхід до проектування орієнтований скоріше на обслуговування об’єкту, ніж на розв’язання конкретних задач. Цей підхід використовується при побудові операційних систем та інструментальних засобів. Разом з тим, проектування алгоритмічної структури звичайно велося методом «униз».

Одним із різновидів аналітичних підходів є покрокова розробка (ПР) [14] і проектування вертикальними прошарками (ПВС) [15].

Серед відомих методів важливу групу складають методи структурного проектування, які засновані на аналізі потоку інформації. Вони з одного боку, відображають потік даних, а з іншого − перетворення вхідних даних у вихідні. Такі методи описані Джексоном [16] та Уорнером [17]. Вони ефективні при розробці інформаційних систем, тому що орієнтовані на виявлення відповідності вхідних файлів вихідним.

Всі розглянуті традиційні методи лягли в основу нових і дають можливість боротися зі складністю − декомпозицією у рівнях абстракції, що по суті є єдиним способом одержання надійного тобто коректного ПЗ. Слід зазначити, що як аналітичний, так і синтетичний підходи призводять до поняття модульності.

Основна аналітична частина дослідження. Аналіз та розуміння того або іншого методу програмування неможливо поза розглядом поняття моделі життєвого циклу програми − системи процедур, правил і інструментальних засобів, використовуваних для розроблення програмної системи. Життєвий цикл програмної системи спершу визначає вимоги до неї користувачів і закінчується аналізом експлуатації системи цими користувачами в рамках їхнього власного оточення.

Як правило, в життєвому циклі є наступні фази:

1. Визначення:
   а) вироблення вимог;
   б) розроблення специфікацій.

2. Проектування:
    а) проектування архітектури;
    б) детальне проектування.
3. Реалізація (конструювання):
    а) кодування;
    б) інтеграція;
    в) тестування (сертифікація).
4. Впровадження.
5. Експлуатація (супровід).

Фаза визначення

Під час фази визначення виявляються властивості, які повинна мати система в остаточному варіанті, тобто створюється задум системи. Розроблювач визначає детальну картину зовнішніх властивостей готового виробу: описуються функції, що буде виконувати система; характеристики інтерфейсу, забезпечуваного системою. Під час цієї фази приймаються рішення щодо розмірів, часу відгуку, продуктивності та інших параметрів системи.

Визначення здійснюється за допомогою двох основних категорій − вимог і специфікацій. При цьому розрізняють опис необхідних властивостей (вимоги) і опис об'єкта, що володіє цими властивостями (специфікації).

Вимога – це властивість, необхідна для рішення проблеми або досягнення цілі. При описі використовуються такі поняття, як якість, можливості або інші характеристики системи, гаданого її використання або середовища. Як правило, вимоги розробляються з метою виявлення місця утворюваної системи в рамках раніше визначеної організації або середовища.

Це середовище використання може не мати формального опису, а робота організації користувача майже завжди ґрунтується на інтуїтивному розумінні робочого середовища робітниками даної організації. Тому найкращою мовою опису вимог, що забезпечують суворе, точне і всеосяжне експертне їхнє висловлення, є професійно-природна мова експертів у сфері діяльності організації користувача, тобто вимоги можуть бути розроблені, зрозумілі і перевірені тільки експертами в даній області застосування.

Специфікація − опис на мові розроблювача зовнішньо відомих характерних рис поводження системи.

Функціональна специфікація програми містить у собі:

- мету програми;
- граничні умови;
- опис функції (що програма повинна робити і, що, можливо, буде робити);
- специфікацію вхідних і вихідних даних;
- верифікаційні вимоги.

Фаза проектування

Вхідною інформацією для проектування є специфікації, написані на підставі вимог користувача. Базуючись на цілях проекту, вимогах до поведінки, планованої документації, специфікаціях дуже високого рівня, різноманітних вихідних формах для користувача, проектувальник здійснює архітектурне і детальне проектування.

Проектування архітектури (первинна або загальна стадія проектування) закінчується декомпозицією специфікацій у структуру системи. При цьому забезпечується подання на модульному рівні, що припускає оцінку, уточнення, модифікацію на самому ранньому етапі розвитку програми. Поряд з описом загальної структури, результатом етапу є специфікації на модуль, що складаються, як правило, з імені/мети, неформального опису, посилань, входу/виходу, алгоритму, приміток.

Ім'я/мета. У цій секції дається ім'я модуля й одне речення, яке описує, що має робити модуль. З ім'ям звичайно дають формальні параметри, асоційовані з модулем.Формальний параметр це параметр, що вказується в полі операндів процедури або макрокоманди. При

виконанні процедури формальний параметр заміщується реальним параметром.

Неформальний опис. Секція містить загальний огляд дій модуля. Потрібно старанно підходити до глибини деталізації опису, приведеного в цій частині. Легко скотитися до поверхневої багатослівності, через що секція стане марною. З іншого боку, легко написати так мало, що стане неясним, що повинен робити модуль.

Посилання. Існують дві категорії посилань: модулі, що прямо посилаються на даний модуль, і модулі, на які посилається даний модуль. Обидві категорії важливі, тому що якщо модуль буде змінюватися, стає ясним, на яких модулях це відіб'ється. Ці посилання також забезпечують можливість перехресних перевірок. З'являється гарантія того, що модулі, на які є посилання, дійсно знають, що на них посилаються. Можна також перевірити, що якщо передбачається, що деякі модулі посилаються на даний модуль, вони дійсно це роблять.

Вхід/вихід. Якщо є формальні параметри, вони повинні бути описані в цій секції. Те саме стосується значення функції, класів змінних і системних констант. Використовуються три специфічних класи змінних:

а) глобальні змінні, до яких мають доступ усі модулі;

б) локальні змінні, відомі тільки даному модулю;

в) змінні, що відомі тільки деяким модулям. Цей останній клас змінних може йти під різними іменами, але важливо відокремити його від перших двох. Інший метод класифікації змінних використовує права доступу. Системні константи − це фіксовані значення, до яких звертаються за символічними іменами.

Алгоритм. Ця частина модуля описує дії модуля. Вони повинні бути зрозумілими розроблювачу. Алгоритм повинний бути структурований. Тут утримуються корисні коментарі. Наприклад, тимчасові характеристики, незвичні ситуації, що призводять до помилкових умов тощо.

На цьому етапі системна структура трансформується в процедурний опис (логіку) програми. Відбувається аналіз, та оцінюється алгоритм для запуску даного модуля. Всі деталі і рішення стосовно кожного модуля повинні бути добре визначені.

Фаза реалізації

Під час цієї фази реалізується переклад проекту у форму програми для конкретної ЕОМ, складання системи та її перевірка в умовах тестових і реальних умовах роботи з метою визначення відповідності експлуатаційних характеристик заявленим у специфікації.

Фаза впровадження

Під час цієї фази провадиться повне, упорядковане, координоване користувачем залучення до реального середовища нової або зміненої системи. Варто одержати підтвердження відповідності висунутим раніше вимогам до системи.

Фаза експлуатації

Під час цієї фази життєвого циклу програмної системи здійснюється оцінка працюючої системи і підтримка її працездатності в заданих межах. При випуску програми як товарної продукції для загального користування завершальною фазою є супровід, що полягає в таких діях:

- знаходження і виправлення помилок;
- додання нових функцій і модифікація існуючих;
- включення програми в нову систему;
- поліпшення показників роботи.

Супровід великої програми може коштувати в 2−3 рази дорожче, ніж її розробка. Найбільші витрати йдуть на своєчасне (синхронне) коригування документації і носіїв у користувача.

Під час проектування архітектури специфікації перетворюються в структуру системи. Засоби, що працюють на цій стадії, розділяють на дві групи: ті, що здійснюють опрацювання й ті, що орієнтовані на дані. Методології, спрямовані на опрацювання, надають особливого значення процесу і структурі в створенні архітектури програми. Методології, орієнтовані на

дані, роблять основний акцент на даних.

Методології, що орієнтовані на опрацювання

Найбільше відомі такі методології, орієнтовані на опрацювання: модульне програмування, метод функціональної декомпозиції, метод проектування потоку даних, метод проектування структур даних, метод HIPO [5,7].

Модульне програмування. Основні концепції модульного програмування:

кожний модуль розроблено до єдиної незалежної функції;

кожний модуль має єдину систему вхід/вихід;

розмір модуля повинен бути мінімальним;

кожний модуль може бути спроектований і закодований різноманітними програмістами і може бути окремо протестований;

При такому підході система розділяється на декілька частин, одночасно утворюваних різноманітними програмістами. Кожний модуль реалізує єдину функцію. Розмір модуля невеликий, тому тестування кероване і може бути проведено ретельно. Після кодування і тестування всіх модулів відбувається їхня інтеграція і тестується вся система. При супроводі тестується й налагоджується тільки той модуль, що погано працює. Очевидні переваги в полегшенні написання і тестування програм, зменшується вартість їх супроводу.

Модуль не повинний змінювати команди іншого модуля і, як правило, не повинний зберігати історію своїх викликів (хоча остання властивість істотно спрощує трасування при збої). Відомі позитивні якості модульних програм:

легкість упорядкування і налагодження, функціональні компоненти такої програми пишуться й налагоджуються порізно, можливе добре структуроване налагодження як “нагору”, так і “униз”;

легкість супроводу і модифікації;

можливість розподілу модулів між програмістами різноманітної кваліфікації відповідно до рівня складності;

можливість створювати бібліотеки найбільше вживаних програм;

спрощення процедури завантаження в оперативну пам'ять великої задачі, що потребує сегментації

виникнення численних природних контрольних точок для спостереження за просуванням програми, використовуваних для підвищення усталеності СПЗ.

Передача управління в модульній структурі відбуваються лише по вертикальних лініях, що з'єднує модулі в схемі ієрархії.

Будь-який модуль може активізувати підпорядкований модуль і одержати керування після завершення його роботи. Модуль більш низького прошарку не може викликати модуль більш високого прошарку. Модуль зобов'язаний повернути управління тому модулю, що його викликав. Прийняття рішень модулями нижчого рівня за модулі вищого рівня не припускається. Надмірна кількість аргументів, що подаються модулю, указує на необхідність поділу функції (основна мета при цьому − скорочення числа аргументів).

Функціональна декомпозиція. Функціональна декомпозиція істотно базується на стратегії "розділяй і керуй". Відомий фахівець в галузі методології програмування Парнас, формалізував процедуру функціональної декомпозиції у формі покрокової деталізації, у якості критерію декомпозиції системи запропонував концепцію приховування інформації. При використанні цього параметру досліджуваний модуль визначається автором суб'єктивно. Другою важливою ідеєю є проектування програмної системи у вигляді набору віртуальних машин замість традиційного підходу, що використовує блок-схеми. Перевага функціональної декомпозиції в її придатності. Хиби − непередбачуваність і мінливість.

Проектування з використанням потоку даних. Ці методи використовують потік даних як рушійну силу процесу проектування програми. При цьому використовуються різноманітні функції відображення, що перетворюють потік інформації в структуру програми.

Структурне проектування засноване на концепції, яка висунута в роботах Йодана і

Майерса. Метод намагається боротися з хибою, властивою методу функціональної декомпозиції, при використанні якого не можна управляти якістю декомпозиції функції. Підхід полягає у відображенні потоку даних проблеми в структуру програми з використанням деяких способів аналізу проекту. Зазвичай прийнята така процедура:

- ідентифікується потік даних і відтворюється граф потоку даних;
- ідентифікуються вхідні, центральні і вихідні перетворюючі елементи;
- формується ієрархічна структура програми, що використовує ці елементи. Після цього оптимізується структура програми.

Цей підхід застосовується при відсутності яскраво виражених структур даних.

Технологія структурного аналізу проекту SADT, заснована на структурному аналізі, запропонованому Россом.

SADT застосовується в різноманітних практичних і військових галузях. Метод особливо ефективний на стадіях розвитку системи.

Метод перетворення графів. Метод заснований на одержанні системної специфікації у вигляді графа керування задачами з графа інформаційного зв'язку між ними У основі методу − використання причинно-наслідкових зв'язків між задачами системи, заснованої на їхній взаємній залежності за даними. Основна мета − одержати схему виконання задач, вільну від конфліктів, породжених відсутністю синхронізації даних.

Методологія Уорнера [4,5]. Вона подібна методології Джексона в тому, що ключем до проекту програми є структури даних. Проте процедура проектування більш деталізована. Використовуються чотири види подання проекту: діаграми організації даних, діаграми логічного проходження, список інструкцій, псевдокод. Діаграма організації даних описує вхідні і вихідні дані. Діаграми логічного проходження подають логічний потік цих даних. Список інструкцій містить команди, що використовуються в проекті. Псевдокод потрібний при описі кінцевих результатів проектування. Методологія Уорнера може бути узагальнена в такий спосіб:

ідентифікувати усі вхідні дані системи;

організувати вхідні дані в ієрархічну форму;

визначити детальний формат кожного елемента вхідного файлу і зафіксувати число їх появ;

повторити кроки 1−3 для вихідних даних;

специфікувати деталі програми, ідентифікуючи типи команд, що містяться в проекті, в такому порядку: читання, розгалуження, обчислення, входи, виходи, виклик підпрограм;

використовувати діаграми типу блок−схем для показу логічної послідовності інструкцій, використовуючи спеціальні символи для подання початку процесу, кінця процесу, розгалуження і вкладення;

пронумерувати елементи логічної послідовності і розкрити їх за допомогою інструкцій, записаних у кроку 5.

HIPO (Ієрархія плюс Вхід, Опрацювання, Вихід). Метод ієрархічних діаграм, розвинений фірмою IBM. Основні характеристики:

спроможність подавати зв'язок між вхідними/вихідними даними і процесом опрацювання;

можливість декомпозувати систему ієрархічно, не залучаючи зайві дрібні деталі;

використання трьох елементів: вхід, опрацювання, вихід.

Опрацювання (процес) специфікується як центральний блок діаграми, вона сполучена з елементами, що відображують вхід і вихід.

Основна процедура проектування з використанням HIPO:

почати з найвищого рівня абстракції;

ідентифікувати вхід, вихід і опрацювання;

з'єднати кожний елемент входу і виходу з відповідним опрацюванням;

документувати кожний елемент системи, використовуючи HIPO діаграми;

деталізувати діаграму, використовуючи кроки 1−4.

Методології, орієнтовані на дані. У цих методологіях виділяються компоненти проекту, засновані на даних. Це так звані об'єктно-орієнтована методологія проектування і методологія проектування концептуальних баз даних. Оскільки обидві технології належать до методу формалізації специфікацій, спочатку розглянемо концепцію методів формальних специфікацій.

Програми можуть бути побудовані методично (систематично), виходячи з формальних специфікацій на дані, з якими вони працюють. Базуючись на формальних специфікаціях, можна розробити засоби автоматичного програмування і докази слушності програм. Особлива увага в літературі приділяється абстракціям даних. Відомі математичні основи доказу коректності з використанням алгебраїчних і аксіоматичних специфікацій. У багатьох роботах формальні специфікації в сполученні з технікою верифікації використані для виявлення помилок на ранніх стадіях життєвого циклу ПЗ.

Об’єктно-орієнтована методологія проектування. Заснована на концепціях приховування інформації й абстрактних типів даних. Такий підхід розглядає всі такі ресурси, як дані, модулі і системи в якості об'єктів.

Кожний об'єкт містить деяку структуру даних (або тип даних), обрамлену набором процедур, що знають як маніпулювати з цими даними. Використовуючи цю методологію, проектант може створити свій власний абстрактний тип і відобразити проблемну частину у ці створені проектантом абстракції замість традиційного відображення проблемної частини у визначені керуючі структури і структури даних мови реалізації. Подібний підхід рекламується як більш природний, чим методології, орієнтовані на опрацювання (на процес).

Через можливість створювати в процесі проектування різноманітні види абстракції типів даних на цьому шляху проектант може сконцентруватися на проекті системи, не турбуючись про деталі інформаційних об'єктів, використовуваних у системі. Повний поділ між специфікаціями і реалізацією в мовах АДА і МОДУЛА−2 роблять їх найбільше пристосованими для використання в даній методології. Основні дії, реалізовані методологією:

визначити проблему;

розвити неформальну стратегію, що подає загальну послідовність кроків, яка задовольняє вимогам до системи;

формалізувати стратегію (ідентифікувати об'єкти і їхні атрибути; ідентифікувати операції над об'єктами; встановити інтерфейси; реалізувати операції).

Методологія, що заснована на проектуванні концептуальних баз даних. Належить до класу методологій, орієнтованих на дані, і покликана дати проектувальнику методичні вказівки в процесі трансляції специфікацій у концептуальну схему бази даних. Цей підхід ставить своєю метою встановити уніфіковану концептуальну модель з більш багатим семантичним значенням і використовувати концепцію абстракцій даних для спрощення проектування. Процес проектування розглядається як процес побудови моделі. Запропоновано методи конструювання концептуальної моделі, засновані на способах узагальнення/спеціалізації. Передбачається, що проектант починає з визначення найбільше загальних природно виникаючих класів об'єктів і подій предметної галузі. Подальші деталі програмної системи вводяться послідовними ітераціями описом підкласів уже поданих класів і спеціалізації взаємодій у цих класах. Проте, ця галузь, що торкається взаємозв'язку між способами концептуального моделювання, поданням знань і механізмом абстрагування потребує подальших досліджень.

Основи методології проектування якісного спеціального програмного забезпечення автоматизованих інформаційних систем воєнного призначення. Вказаній методології присвячено багато наукових джерел, наприклад [5-9, 19-22] та інші. Базою методології створення якісного СПЗ є використання наступних етапів, на кожному з яких розроблення ведеться пошарово на обмеженому наборі допустимих структур.

I − етап проектування системних зв'язків (системна специфікація). Змістом цього етапу є тривимірна функціонально-подійно-режимна декомпозиція СПЗ на задачі. При цьому у доповнення до класичної функціональної декомпозиції здійснюється декомпозиція за принципом реакції на події з об‘єкту та за режимами роботи системи.

II − етап виділення функціонального ядра кожної задачі. Особливістю такого виділення є забезпечення незалежності даної частини задачі від обраної операційної системи і обраного методу міжзадачного зв'язку за даними. При цьому всі функції зв'язку покладаються на частину задачі, що залишилася − інтерфейс. Задача–інтерфейс − основний об'єкт операційної системи, за допомогою якої здійснюється зв'язок функціональних ядер за управлінням та інформацією. Починаючи з даного етапу, розроблення ведеться за двома зустрічними напрямками: проектування ядер ведеться “згори донизу” (аналітичний підхід), остаточна інтеграція - збирання системи за допомогою інтерфейсів − “знизу вгору” (синтетичний підхід).

III − етап полягає в пошаровому модульному проектуванні, який є подальшою декомпозицією функціональних ядер.

IV – етап виробляється деталізація модулів з використанням структурних примітивів структурного програмування.

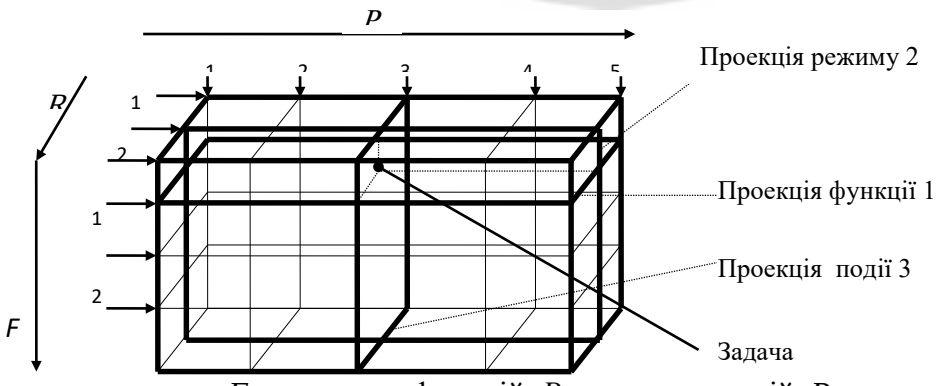
Слід відзначити, що на першому етапі проектування здійснюється функціонально-подійно-режимна декомпозиція (ФПР-декомпозиція) СПЗ на задачі - виділення окремих задач, які реалізують певну часткову функцію системи управління, що виконується при виникненні конкретної події в одному з режимів роботи системи.

Оскільки будь-яка зміна ситуації у часі, що змінює поточний стан потоку управління в системі, називається подією, до початку декомпозиції необхідно мати ситуаційну модель – відображення множини ситуацій, які можуть виникнути на об‘єкті, у множину зовнішніх подій для системи управління. Прикладом зовнішньої події є активізація ініціативної кнопки оператором системи. Режим розуміється як регламент роботи системи, що зберігає структуру управління. Введення системи в роботу, “програмний відступ” при виявленні непрацездатності елементів СПЗ, закінчення роботи відбуваються при переході з одного режиму на інший. Функція - сукупність дій СПЗ, спрямованих на досягнення певної мети (проведення розрахунків для отримання сукупності результатів).

Після розбиття СПЗ на задачі здійснюється опис вхідних та вихідних даних для кожної з отриманих задач, що реалізують певний режим.

Подання задачі z_j у вигляді двох наборів даних (вхідних $\{Ind\}$ і вихідних $\{Od\}$) дозволяє побудувати орієнтований граф інформаційних зв'язків між задачами системи $\boldsymbol{L}_I$, в якому множина вершин графа збігається з множиною задач, а дуга, що зв'язує вершини z_i та z_j, існує тоді, коли

$$\{Od_i\} \cap \{Ind_j\} \neq \varnothing \ . \qquad (1)$$



F – множина функцій, *P* - множина подій, *R* - множина режимів

Рисунок 1 - Схема функціонально-подійно-режимної декомпозиції СПЗ на задачі

Наявність в інформаційному графі сильно зв’язаних компонент свідчить про нерозумну декомпозицію на першому етапі проектування. З точки зору ефективності організації обчислювального процесу більш раціональним є об'єднання задач, що утворюють сильно зв’язану компоненту, в одну задачу. Дана процедура відповідає розкладанню графа на максимально сильно зв’язані підграфи.

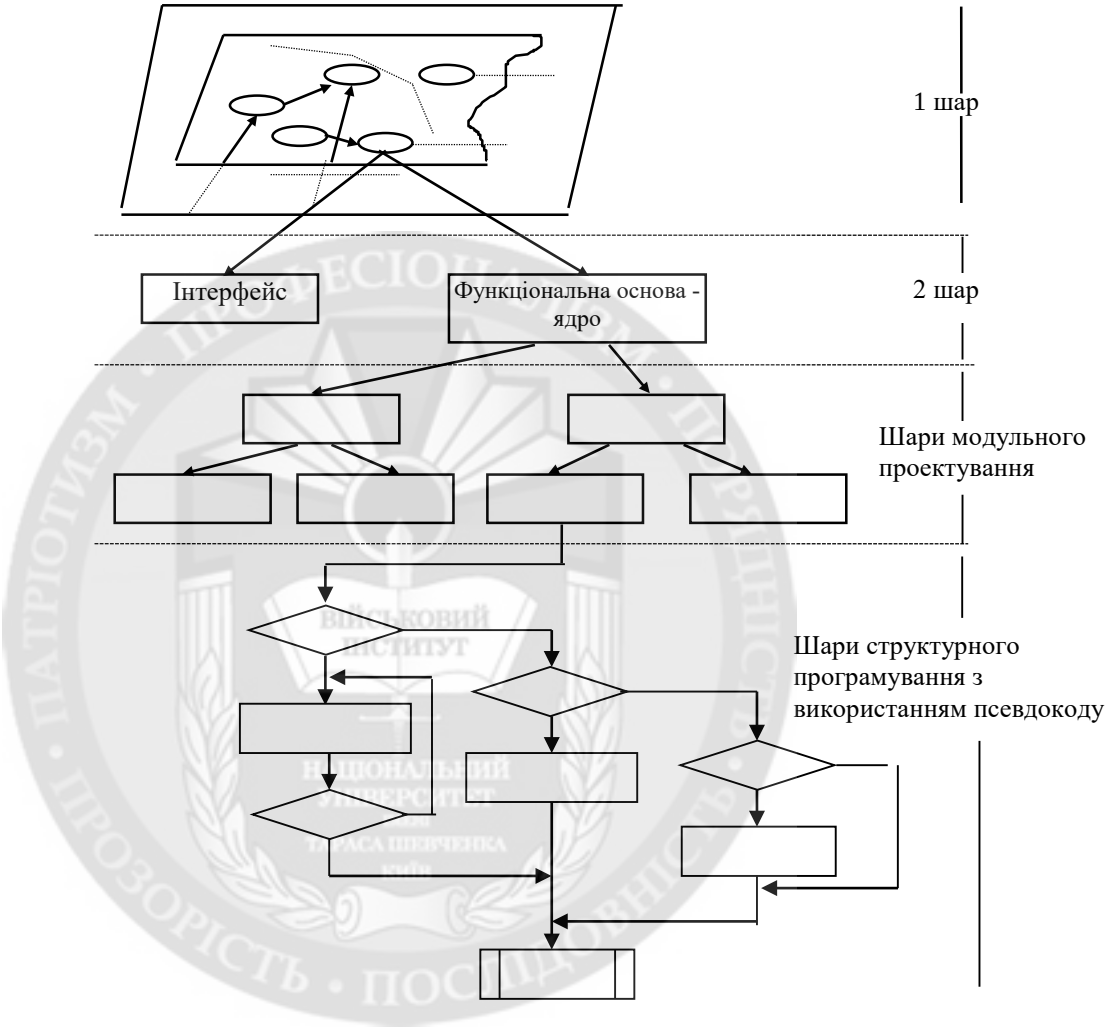


Рисунок 2 – Схема пошарового структурного проектування СПЗ для одного режиму

У графі управління, що буде отриманий з перетвореного графа інформаційних зв'язків, міжзадачних циклів не буде, а циклічність виконання окремих задач буде виражена петлею при вершині. На рис. 3 показано приклад перетворення графа та об'єднання задач. Задачі z_5 , z_6 , z_7 об'єднані в нову задачу Z_5', z_{11} , z_{12} , z_{13} — в Z_6', z_3 , z_8 , z_9 — в Z_3’.

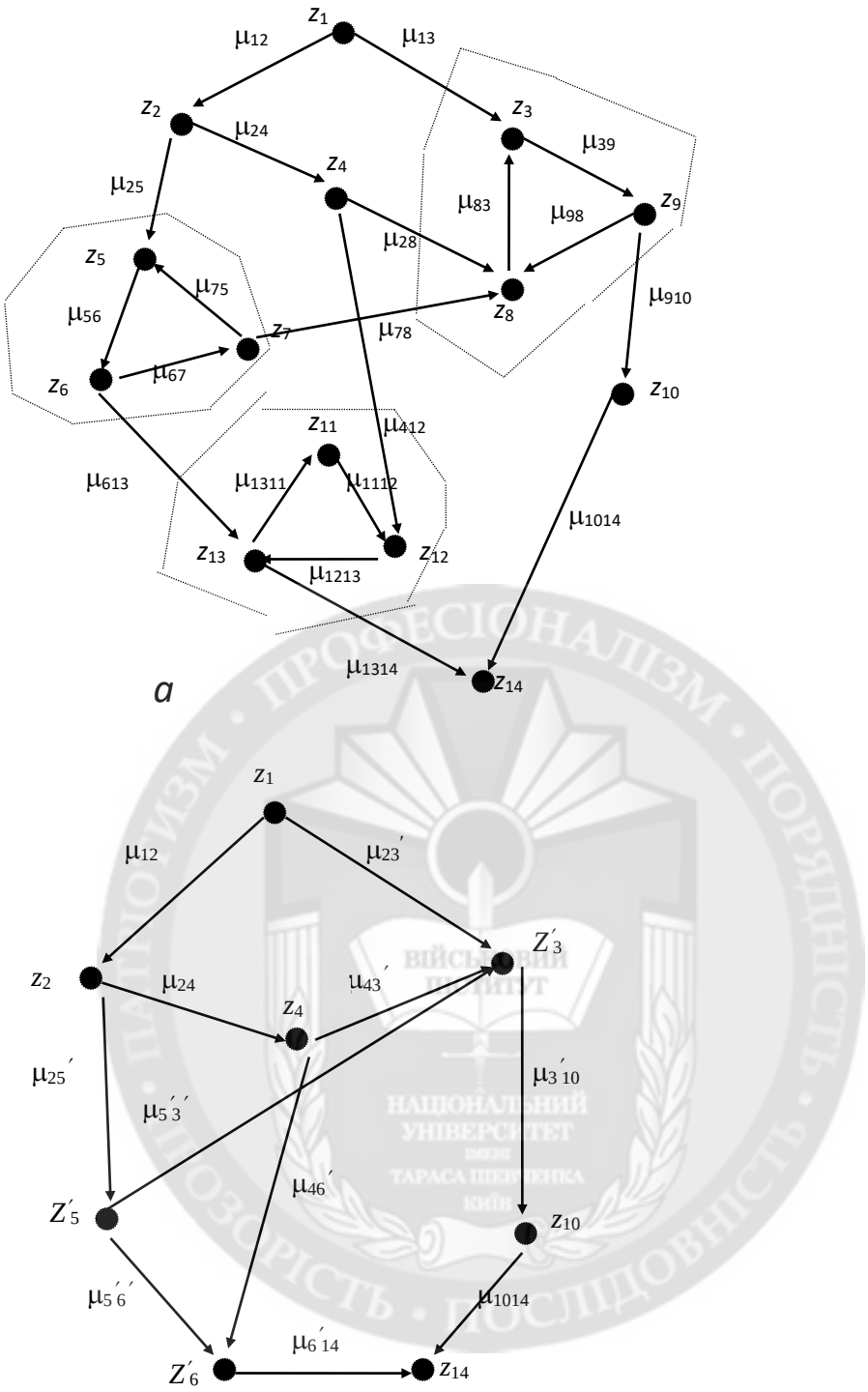


Рисунок 3 – Перетворення графа інформаційних зв’язків (об’єднання задач):
а – вихідний граф; *б* –перетворений граф

Етап модульного проектування по суті є етапом функціональної декомпозиції. Рівні абстракції або шари готового продукту на цьому етапі проектування набувають статус рівней модулів у створюваній програмі, відбиваючи майбутню ієрархію системи. До цього часу існують різночитання поняття модульності. У роботі прийнято визначення, згідно з яким модульним називається таке програмне забезпечення, в якому можна змінити будь-яку частину логічної структури, не викликаючи змін в інших частинах. Під модулем розуміється послідовність логічно зв'язаних фрагментів, які оформлені як окрема частина програми.

Визначальна властивість модуля - незалежність відносно логічної структури СПЗ, аргументів (параметрів) модуля, констант, тощо. Будемо розрізняти Ф-модулі (акції – функціональні дії), І-модулі (інтерфейси) і ОСД-модулі (операції над структурами даних). Введення ОСД-модулів пояснюється тим, що правильне використання структур даних грає дуже важливу роль при проектуванні та визначає ефективність СПЗ.

На конструкцію модуля накладаються такі обмеження:

модуль повинен мати один вхід і один вихід;

модуль не повинен змінювати команди іншого модуля;

модуль не повинен зберігати історію своїх викликів (хоча її наявність спрощує трасування при збоях роботи СПЗ);

в ідеалі кожний модуль повинен реалізувати одну функцію цілком.

Програми, які розроблені за модульним принципом, мають такі переваги:

легкі у складанні і налагодженні а, отже, в супроводі і модифікації;

існує можливість розподілу модулів для програмування між програмістами різної кваліфікації;

можливість створення бібліотек найбільш вживаних програм;

спрощення процедури завантаження в оперативну пам'ять великої задачі, яка вимагає сегментації;

можливість підвищення стійкості СПЗ за рахунок виникнення численних природних контрольних точок для спостереження за роботою програми.

Суттєві обмеження на глибину модульності програм накладає підвищення витрат обчислювальних ресурсів системи за часом роботи центрального процесора й обсягом пам'яті. Додаткові обмеження на реалізацію модульних програм полягають в наступному. Передачі управління між модулями відбуваються по вертикальних лініях, що з'єднують модулі у схемі ієрархії. Модуль, що активізує підлеглий модуль, отримує управління після завершення його роботи (зворотне не вірно — модуль більш низького рівня ієрархії не може викликати модуль більш високого рівня). Модуль завжди повертає управління тому модулю, що його викликав. Прийняття рішень модулями нижчого рівня за модулі вищого рівня не допускається. Надто велика кількість аргументів, що надаються модулю, тягне необхідність розподілу функцій. Пропонується обчислювати кількість модулів, виходячи з залежності $M=\eta/6$, де M — кількість модулів, η — кількість єдиних за змістом вхідних і вихідних параметрів, які включені в алгоритм. Дана залежність базується на припущенні, що кількість параметрів ідеального модуля дорівнює 6. Введення певних правил модульного проектування, опис їх у термінах формальної теорії дозволяє побудувати формальні моделі ієрархічних модульних структур і провести їх верифікацію.

Структурне програмування припускає використання певних принципів розроблення програм, яким відповідає набір жорстких правил для полегшення тестування, поліпшення читаності програм і підвищення продуктивності праці проектувальників. Методи структурного програмування базуються на відмові від використання оператора GO TO в явному вигляді і реалізації сукупності цілком певних структурованих операторів передачі управління (керуючих структур) у поєднанні з принципами спадного (зверху вниз) проектування.

Розрізняють декілька класів керуючих структур. Основний (базовий) клас утворюють *D*-структури, що являють собою програми, які складені з структурних примітивів з одним входом і одним виходом (оператори призначення, виклики процедур, оператори вводу-виводу, умовні конструкції форми IF *P* THEN S_1 ELSE S_2, цикли форми WHILE *P* DO *S*). Наступний клас керуючих структур − *D'*-структури (оператори IF *P* THEN *S* з однією гілкою, цикли REPEAT *S* UNTIL *P*, оператори CASE з гілками).

Наступні класи являють собою узагальнення розглянутих вище структур, які пов'язані з наявністю багатьох входів і виходів.

Серед них є BJ_n-структури, що складені з основних дій, композицій, структур IF *P* THEN

S_1 ELSE S_2 і одновхідних-одновихідних структур Ω_k ($k \leq n$).

У Ω_k-структурі є k послідовних предикатів і дій з k виходами, по одному на кожний предикат.

Наступний клас складають RE_n, REC_n, DRE_n, $DREC_n$-структури.

RE_n-структура включає основні дії, композиції, структури IF_THEN_ELSE, команди виходу форми EXIT(c) (c − позитивна ціла константа) і конструкції REPEAT_END.

REC_n-структура подібна RE_n-структурі з тією відзнакою, що у неї є додаткова команда формування циклу CYCLE(c).

DRE_n-структура являє собою сукупність RE_n-структури з доданням структур WHILE_DO.

$DREC_n$-структури визначаються як DRE_n-структури з доданням операторів CYCLE(c).

У цілому RE_n-структурам і їх варіантам відповідають програми, в яких передача управління може здійснюватися тільки у початок або у кінець вкладеного циклу управління.

GP_n- і L-структури складають наступний клас структур. При цьому GP_n-структура − така структура, в якій всі підструктури з одним входом і одним виходом мають не більш n різноманітних предикатів. L-структура — будь-яка структура без обмежень на кількість або конфігурацію предикатів, дій і передач управління. Дана структура відповідає програмі з вільним використанням міток і операторів GO TO.

Основними результатами теорії структурного програмування є результати досліджень з теоретичної повноти D-структур [107] і результати щодо ієрархії структурних класів [106].

Сенс першого результату зводиться до того, що доведена можливість перетворення будь-якої L-структури у функціонально еквівалентну D-структуру, тобто для проектування програм достатньо D-структур (оператор GO TO виявляється теоретично непотрібним): послідовної композиції, IF_THEN_ELSE та WHILE_DO. При цьому L-структура може бути зведена до D-структури без введення нових бульових перемінних або зміни конкретних дій і предикатів програм тільки тоді, коли вона не містить циклів з двома роздільними виходами.

Другий результат показує, які структури і в якому ступені можуть бути зведені до інших (рис. 4). Безперервна спрямована вгору лінія на даному рисунку означає суворе зведення структур нижнього класу до структур верхнього класу (наприклад, клас C_1 зведений до класу C_2, якщо кожна структура в C_1 може бути перетворена у структури з C_2 , але не навпаки). З'єднання штриховою лінією означає, що нижній клас зведений до верхнього, при цьому кожна структура нижнього класу може бути перетворена до структури верхнього, але не обов'язково навпаки. Еквівалентність класів ($C_1 \sim C_2$) означає, що кожна структура C_1 може бути перетворена в структуру C_2 і навпаки.

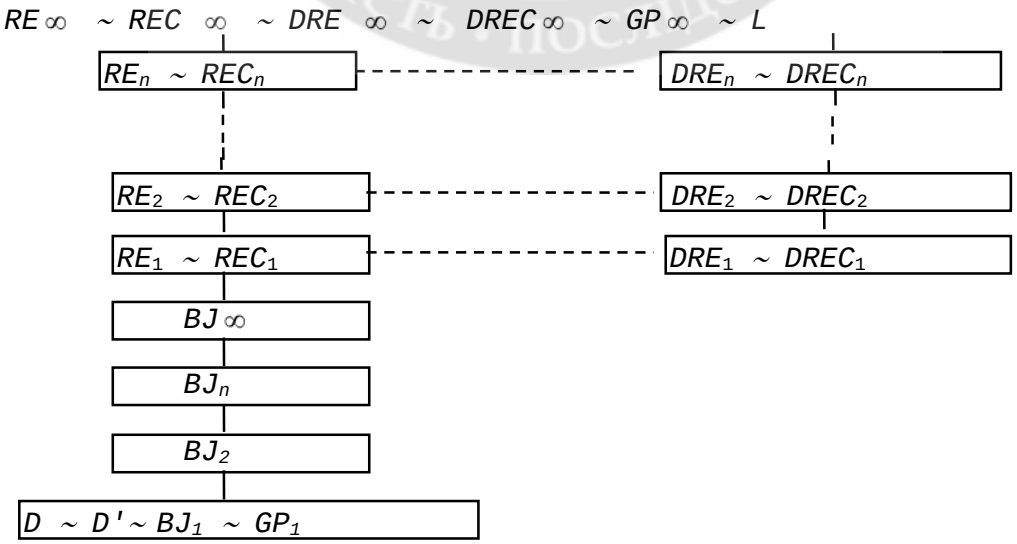


Рисунок 4 – Ієрархія класів управляючих структур

Аналіз практичного застосування структур при використанні принципів структурного програмування показує, що для потреб практики виявляється достатнім використання класів *D* і *D'*. Застосування більш високих структур суперечить принципу ясності. При цьому *D'*-структури часто виявляються більш корисними, ніж *D*-структури. Окрім основних переваг структурного програмування (ієрархічність, видимість, наочність, скорочення термінів проектування), його реалізація дозволяє ввести контроль правильності програм на рівні керуючих структур. Дотримання принципу незалежності (структури нижнього рівня не можуть користуватися інформацією про верхні рівні) дозволяє не тільки пошарово проектувати і модифікувати ППЗ, але й побудувати ієрархічну схему локалізації помилок: ситуація − система задач − задача − функціональне ядро − модуль − керуючі структури.

Для зменшення ймовірності виникнення помилок в СПЗ технологія проектування повинна бути прозорою, простою для супроводження та дозволяти ефективне управління розробкою. Крім того, вона повинна враховувати особливості предметної галузі, для якої розробляються програми.

З цією метою технологічний процес проектування може бути диференційований на ряд етапів-процедур. Ці процедури упорядковані та застосовуються у певній послідовності і дозволяють вести проектування крок за кроком. При цьому інформація, що отримується в результаті виконання певної процедури, є вхідною для наступних.

Висновки. В роботі проведено аналіз сучасних та традиційних методів проектування і виробництва військової техніки з використанням автоматизованих інформаційних систем.

Програмне забезпечення АІС має характерні ознаки складної системи і підпорядковується законам науки про складні системи. Сучасний етап розвитку складних інформаційних систем характеризується тим, що тенденція до різкого збільшення складності завдань управління все в більшому ступені виявляється в об'ємності програмного забезпечення систем управління технічними об'єктами, технологічними процесами і виробництвами. Обсяг опрацьованої інформації в системах цього типу зростає більш інтенсивно, чим збільшення обсягів самого виробництва, або підвищення його техніко-економічних показників.

Базою методології створення якісного СПЗ є використання етапів, на кожному з яких розроблення ведеться пошарово на обмеженому наборі допустимих структур.

Перший етап − етап проектування системних зв'язків (системна специфікація). Змістом цього етапу є тривимірна функціонально-подійно-режимна декомпозиція СПЗ на задачі. При цьому у доповнення до класичної функціональної декомпозиції здійснюється декомпозиція за принципом реакції на події з об‘єкту та за режимами роботи системи.

Другий етап − виділення функціонального ядра кожної задачі. Особливістю такого виділення є забезпечення незалежності даної частини задачі від обраної операційної системи і обраного методу міжзадачного зв'язку за даними. При цьому всі функції зв'язку покладаються на частину задачі, що залишилася - інтерфейс. Задача- інтерфейс - основний об'єкт операційної системи, за допомогою якої здійснюється зв'язок функціональних ядер за управлінням та інформацією. Починаючи з даного етапу, розроблення ведеться за двома зустрічними напрямками: проектування ядер ведеться “згори донизу” (аналітичний підхід), остаточна інтеграція - збирання системи за допомогою інтерфейсів - “знизу вгору” (синтетичний підхід).

Третій етап полягає в пошаровому модульному проектуванні, який є подальшою декомпозицією функціональних ядер.

На четвертому етапі виробляється деталізація модулів з використанням структурних примітивів структурного програмування.

Таким чином, концептуальна єдність підходів, що використані на етапах отримання системних специфікацій, проектування модульної структури та реалізації модулів, дозволила об‘єднати прийняття окремих рішень в єдину методику проектування коректного спеціального програмного забезпечення, основними відмінностями якої є:

пошаровість, яка досягається використанням принципу абстрагування (ієрархії);

об‘єднання процесу проектування з процесом моделювання, що досягається застосуванням графових моделей, які отримуються в процесі проектування, та можливістю використання кожного рівня ієрархії в якості моделі СПЗ;

безперервність процесу проектування, яка забезпечується сумісністю моделей, які отримуються на різних етапах, участю системного програміста в розробці з начального етапу і можливістю початку кодування з першого етапу.

З метою виявлення та діагностування помилок при передачі управління між структурними елементами системи за допомогою "методу паролів" найбільша ефективність застосування паролів досягається їх рівномірним розподілом по всій шарах СПЗ.

Метод "тестових інтерфейсів" дозволяє удосконалити процес проектування програмного забезпечення за рахунок застосування діагностичної системи контролю за цілісністю програм.

Тестування є одним з найважливіших методів контролю якості СПЗ, і практично єдиним з них, що може бути доступним кінцевим користувачам. Жоден з інших відомих методів, жодна будь-яка їх комбінація не можуть повністю замінити тестування, оскільки тільки метод тестування дозволяє отримати оцінки якості СПЗ по коду що виконується.

ЛІТЕРАТУРА:

1. Основы инженерии качества программных систем / Ф.И. Андон, Г.И. Коваль, Г.М. Коротун, В.Ю. Суслов: НАНУ, Институт программных систем. – К.: Академпериодика, 2002, 504 с.

2. Де Марко Т., Листер Т. Человеческий фактор. Успешные проекты и команды. – М.: Символ-Плюс, 2012, 288 с.

3. Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. – Офіційне видання. – К.: Держспоживачстандарт, 2001, (Національний стандарт України).

4. Толубко В.Б., Сбітнєв А.І., Пермяков О.Ю., Методологічні основи проектування прикладного забезпечення до АСУ воєнного призначення. Монографія. – К.: НАОУ 2004 р. 248 с.

5. Грищак О.М. Проектування спеціального програмного забезпечення авто-матизованих інформаційних систем (аналітичний огляд)// Комп'ютерні інформаційні технології та системи. Вісник ЧДТУ. Черкаси - 2008. №2 - 075 с. 3-5

6. Лєнков С.В., Грищак О.М., Жиров Г.Б., Пампуха І.В. Оцінка "практичності" та "коректності" спеціального прогрманого забезпечення автоматизованих інформаційних систем воєнного призначення // Збірник наукових праць Військового інституту Київського університету імені Тараса Шевченка. – Київ, - 2022. - №74. – С.83 – 89. DOI: 10.17721/2519-481X/2022/74-09.

7. Лєнков С.В., Сєлюков О.В., Гусак Ю.А., Пампуха І.В., Солодєєва Л.В. Оцінка "здатності до супроводу" та "здатності до переміщення" спеціального програмного забезпечення автоматизованих інформаційних систем воєнного призначення // Збірник наукових праць Військового інституту Київського університету імені Тараса Шевченка. – Київ, - 2022. - №75. – С. 23 – 30. DOI: 10.17721/2519-481X/2022/75-04.

8. Лєнков С.В., Бойченко О.В. Оцінка вірогідності даних автоматизованої системи управління // Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка. – К., 2014. – № 45. – С. 46 – 51.

9. Лєнков С.В., Джулій В.М., Муляр І.В. Динамічні показники оцінки рівня функціональної безпеки інформаційної системи // Журна «Сучасна спеціальна техніка», м.Київ, 2016. – 2(45). – С.59 – 67.

10. Турский В. Методология проектирования программ. - М.: Мир, 1981. -265с.;

11. Lehman M. Programs, Life Cycle and Laws of Software Evolution// Proc. of IEEE. - 1980. - Vol. 68. - pp. 26-46.

12. Йодан Э. Структурное проектирование и конструирование программ. - М.: Мир, 1979. -415с.

13. Миллс Х. Программирование больших систем по принципу "сверху-вниз"// Средства отладки больших систем. - М.:Статистика. - 1977.

14. Wirth N. Program Development by Step-Wise Refinment//Comm. ACM. - 1971. - Vol. 14. - pp. 221-229.

15. Фуксман А.Л. Технологические аспекты создания программных систем. - М.: Статистика, 1979. - 184с.

16. Jakson M.A. Principles of Program Design. - N.Y.: Academic Press. - 1975. - p.238.

17. Liskov B.H. A Design Methodology for Reliable Software Systems // Fall Joint Computer Conf., AFIPS. – 1972. – Vol. 41. – P. 191-199.

18. В.А. Кашканова, А.А. Кашканова, Б.П. Кукец Інформаційні системи і технології на автомобільному транспорті: Навчальний посібник. – Вінниця: ВНТУ, 2020-104с.

19. Семчак О.М., Богданова О.Г. Проблеми та перспективи розвитку автоматизованих систем управління цілевказання // Збірник наукових праць Військового інституту Київського університету імені Тараса Шевченка. – Київ, - 2024. - №82. – С. 94 – 102. DOI: DOI:https://doi.org/10.17721/2519-481X/2024/82-11.

20. Кацалап В.О., Омельянчук А.В., Сивак О.В. Обгрунтування показників відмовостійкості автоматизованої системи управління центру оперативного керівництва Збройних Сил України // Збірник наукових праць Військового інституту Київського університету імені Тараса Шевченка. – Київ, - 2027. - №77. – С. 45 – 100. DOI: https://doi.org/10.17721/2519-481X/2022/77-04.

21. Фесьоха В.В., Нерознак Є.І., Сова О.Я. Удосконалена модель оптимального використання серверних ресурсів кластерної системи військового призначення на основі рівноваги неша // Збірник наукових праць Військового інституту Київського університету імені Тараса Шевченка. – Київ, - 2023. - №79. – С. 159 – 171. DOI: https://doi.org/10.17721/2519-481X/2023/79-15.

22. Khlaponin Y.I., Qasim N. H., Tarasiuk D.M.// Збірник наукових праць Військового інституту Київського університету імені Тараса Шевченка. – Київ, - 2023. - №80. – С. 91 – 97. DOI: http://doi.org/10.17721/2519-481X/2023/80-10.

REFERENCES:

1. Osnovy inzhenerii kachestva programnykh system / F.I. Andon, H.I. Koval, H.M. Korotun, V.Yu. Suslov: NANU, Instytut prohramnykh system. – K.: Akademperiodyka, 2002, 504 s.

2. De Marko T., Lister T. Chelovecheskiy faktor. Uspeshnye proekty i komandy. – M.: Simvol-Plyus, 2012, 288 s.

3. Informatsiyni tekhnolohiyi. Otsinyuvannya protsesiv zhyttyevoho tsyklu prohramnykh zasobiv. – Ofitsiine vydannya. – K.: Derzhspozhyvachstandart, 2001, (Natsionalnyy standart Ukrayiny).

4. Tolubko V.B., Sbitniev A.I., Permyakov O.Yu., Metodolohichni osnovy proektuvannya prykladnoho zabezpechennya do ASU voyennoho pryznachennya. Monohrafiya. – K.: NAOU 2004 r. 248 s.

5. Hryshchak O.M. Proektuvannya spetsialnoho prohramnoho zabezpechennya avtomatyzovanykh informatsiynykh system (analitychnyy ohlyad)// Komp'yuterni informatsiyni tekhnolohiyi ta systemy. Visnyk ChDTU. Cherkasy - 2008. №2 - 075 s. 3-5.

6. Lenkov S.V., Hryshchak O.M., Zhyrov H.B., Pampukha I.V. Otsinka "praktychnosti" ta "korektnosti" spetsialnoho prohrmanoho zabezpechennya avtomatyzovanykh informatsiynykh system voyennoho pryznachennya // Zbirnyk naukovykh prats Viys'kovoho instytutu Kyyivs'koho universytetu imeni Tarasa Shevchenka. – Kyyiv, - 2022. - №74. – S.83 – 89. DOI: 10.17721/2519-481X/2022/74-09.

7. Lenkov S.V., Selyukov O.V., Husak Yu.A., Pampukha I.V., Solodyeyeva L.V. Otsinka "zdatnosti do suprovodu" ta "zdatnosti do peremishchennya" spetsialnoho prohramnoho zabezpechennya avtomatyzovanykh informatsiynykh system voyennoho pryznachennya // Zbirnyk naukovykh prats Viys'kovoho instytutu Kyyivs'koho universytetu imeni Tarasa Shevchenka. – Kyyiv, - 2022. - №75. – S. 23 – 30. DOI: 10.17721/2519-481X/2022/75-04.

8. Lenkov S.V., Boychenko O.V. Otsinka virohidnosti danykh avtomatyzovanoyi systemy upravlinnya // Zbirnyk naukovykh prats Viys'kovoho instytutu Kyyivs'koho natsionalnoho universytetu imeni Tarasa Shevchenka. – K., 2014. – № 45. – S. 46 – 51.

9. Lenkov S.V., Dzhuliy V.M., Mulyar I.V. Dynamichni pokaznyky otsinky rivnya funktsionalnoyi bezpeky informatsiynykh system.

10. Turskyi V. Metodolohyia proektyrovanyia prohramm. - M.: Myr, 1981. -265s.

11. Lehman M. Programs, Life Cycle and Laws of Software Evolution// Proc. of IEEE. - 1980. - Vol. 68. - pp. 26-46.

12. Yodan Э. Strukturnoe proektyrovanye y konstruyrovanye prohramm. - M.: Myr, 1979. -415s.

13. Mylls X. Prohrammyrovanye bolshykh system po pryntsypu "sverkhu-vnyz"// Sredstva otladky bolshykh system. - M.:Statystyka. - 1977.

14. Wirth N. Program Development by Step-Wise Refinment//Comm. ACM. - 1971. - Vol. 14. - pp. 221-229.

15. Fuksman A.L. Tekhnolohycheskye aspekty sozdanyia prohrammnыkh system. - M.: Statystyka, 1979. - 184s.

16. Jakson M.A. Principles of Program Design. - N.Y.: Academic Press. - 1975. - p.238.

17. Liskov B.H. A Design Methodology for Reliable Software Systems // Fall Joint Computer Conf., AFIPS. – 1972. – Vol. 41. – R. 191-199.

18. V.A. Kashkanova, A.A. Kashkanova, B.P. Kukets Informatsiini systemy i tekhnolohii na avtomobilnomu transporti: Navchalnyi posibnyk. – Vinnytsia: VNTU, 2020-104s.

19. Semchak O.M., Bohdanova O.H. Problemy ta perspektyvy rozvytku avtomatyzovanykh system upravlinnia tsilevkazannia // Zbirnyk naukovykh prats Viiskovoho instytutu Kyivskoho universytetu imeni Tarasa Shevchenka. – Kyiv, - 2024. - №82. – S. 94 – 102. DOI: DOI:https://doi.org/10.17721/2519-481X/2024/82-11.

20. Katsalap V.O., Omelianchuk A.V., Syvak O.V. Obhruntuvannia pokaznykiv vidmovostiikosti avtomatyzovanoi systemy upravlinnia tsentru operatyvnoho kerivnytstva Zbroinykh Syl Ukrainy // Zbirnyk naukovykh prats Viiskovoho instytutu Kyivskoho universytetu imeni Tarasa Shevchenka. – Kyiv, - 2027. - №77. – S. 45 – 100. DOI: https://doi.org/10.17721/2519-481X/2022/77-04.

21. Fesokha V.V., Neroznak Ye.I., Sova O.Ia. Udoskonalena model optymalnoho vykorystannia servernykh resursiv klasternoi systemy viiskovoho pryznachennia na osnovi rivnovahy nesha // Zbirnyk naukovykh prats Viiskovoho instytutu Kyivskoho universytetu imeni Tarasa Shevchenka. – Kyiv, - 2023. - №79. – S. 159 – 171. DOI: https://doi.org/10.17721/2519-481X/2023/79-15.

22. Khlaponin Y.I., Qasim N. H., Tarasiuk D.M.// Zbirnyk naukovykh prats Viiskovoho instytutu Kyivskoho universytetu imeni Tarasa Shevchenka. – Kyiv, - 2023. - №80. – S. 91 – 97. DOI: http://doi.org/10.17721/2519-481X/2023/80-10.

Dr. Tech. Sci., prof. Lienkov S.V., Ph.D. Grishchak O.M.
Ph. D. Okhramovych M.M., Sotnikov Y.O.

ANALYSIS OF MODERN AND TRADITIONAL METHODS OF DESIGNING AND MANUFACTURING MILITARY EQUIPMENT USING AUTOMATED INFORMATION SYSTEMS

In the article, the authors perform one type of analysis of existing and prospective methods of designing and manufacturing military equipment using automated information systems. It is known that AIS software has the characteristic features of a complex system and is subject to the laws of complex systems science. The current stage of development of complex information systems is characterized by a sharp increase in the complexity of management tasks, which is increasingly reflected in the volume of software for managing technical objects, technological processes, and production. The work classifies programs according to SPE, which represents "a model of another model within the theory of abstract representation of individual objects." It can be classified into one of three classes: S-class - these are programs whose functions are formally defined by specifications and follow from them. The correspondence between input and output is decisive, and any change creates a new program. P-class - these are programs whose solution acceptability is assessed by comparison with the real situation, meaning that programs of this class must constantly change depending on the environment, data refinement, etc. E-class programs themselves become part of the implementation of the situation they model, meaning they are even more prone to changes than P-programs.

The main phases of the AIS life cycle are considered, and their features are defined: definition, design, implementation, deployment, and operation.

The conceptual unity of approaches used at the stages of obtaining system specifications, designing the modular structure, and implementing modules allowed for the integration of individual decisions into a single methodology for designing correct special software, the main differences of which are: layering achieved through the use of the abstraction principle (hierarchy); combining the design process with the modeling process achieved through the application of graph models obtained during design, and the possibility of using each level of the hierarchy as a model of the special software; continuity of the design process ensured by the compatibility of models obtained at different stages, the participation of the system programmer from the initial stage, and the possibility of starting coding from the first stage.

Keywords: automated information systems, technical object management systems, conceptual unity of approaches, continuity of the design process.