# МЕТОДИ ЕВОЛЮЦІЇ ПРОГРАМНИХ КОМПОНЕНТІВ ДЛЯ ЇХ ПОВТОРНОГО ЗАСТОСУВАННЯ

*Грищенко В.М.*

Інститут програмних систем НАН України, просп. Глушкова, 40, Київ, 03187, Тел.: (044) 266 34–70, E-mail: grish@public.icyb.kiev.ua

Визначаються методи та операції еволюції компонентів для їх поліпшення, відновлення та повторного застосування, тобто рефакторінг (зміна інтерфейсів та реалізацій), реінжінірінг (реструктуризація, перепрограмування,) та реверсний інжінірінг (відновлення початковій структури на основі коду компоненту).

Methods and operation to components evolution for improving, ennewing and reusing, such as refactoring (interfaces and realizations), reengineering (reprogramming, restructuring) and reverse engineering (rehabilitation of structure on base code).

## Вступ

У останні роки швидко розвивається компонентне програмування, яке є складовою частиною програмної інженерії. Воно базується на застосуванні компонентів повторного використання та нових компонентів, що створюються.

Компонентами для повторного їх використання у інших розробках можуть бути: методи об'єктів та їх інтерфейси, діючі програми та програмні системи для різних середовищ та платформ, а також компоненти, що створені по сучасним технологіям (CORBA, COM, JAVA). Компонент є самостійний програмний елемент, що задовольняє визначеним функціональним вимогам, вимогам архітектури, структури й організації взаємодії в компонентній програмі для деякого середовища. Для повторного застосування таких різноманітних компонентів для нових умов їх функціонування необхідні різні методі їх перетворення з метою їх щодо пристосування до визначених умов і середовищ. В цілому повторне застосування компонентів дає значні спрощення процесів розробки нових як окремих компонентів, так і компонентних систем за їх композицією, а також супроводу за рахунок [1-3]:

- більшої формалізації та впорядкованого процесу розробки окремих компонентів;
- повторного застосування компонентів, яке дозволяє зменшити час процесу розробки і одночасно збільшити його якість та надійність;
- визначених правил та механізмів компонентних моделей для виконання компонентів у інтегрованому середовищі, регламентації процесів подання компонентів, опису їх інтерфейсів, обробки компонентів та використання загальносистемних сервісів та ін.;
- спрощення процесів побудови програмної системи з компонентів та їх вдосконалення.

Зараз в теорії та практиці компонентного програмування не досліджені з теоретичної точці зору і не вирішені усі аспекти наукові проблеми щодо перетворення (еволюції) компонентів для їх повторного застосування. С прикладної точки зору інтуїтивні склались способи, за допомогою яких багата років виконувалися зміни одних компонентів на інші. К більш системним методам перетворення компонентів задля нових цілій їх застосування відносяться: рефакторінг (зміни інтерфейсів та реалізацій), реінжінірінг (перепрограмування, реструктуризація) та реверсний інжінірінг (відновлення початковій структури на основі коду компонента). Розробці теоретичних основ проблем перетворення компонентів для подальшого їх застосування при створенні компонентних програмних систем і присвячена дана робота.

## 1. Алгебра обробки та еволюції компонентів

Побудова програмних систем з використанням компонентів повторного використання, що змінюються потребує нових формальних методів еволюції цих компонентів. На цей час у компонентному програмуванні склалися формальні методи, які широко застосовуються в сучасних дослідженнях, є значний обсяг теоретичних результатів, щодо композиції різних компонентів, серед яких можна виділити важливі:

- мови і моделі опису інтерфейсів та взаємодії компонентів;
- компонентні моделі;
- мови опису компонентних архітектур і ін.;

Ці результати спрямовані на розгляд компонента як цільного, формально представленого об'єкта з визначеними характеристиками і типовою структурою. Це відповідає ідеалізованої точки зору, що компоненти є базовими елементами (функціями, модулями і т.д.), які мають атомарну структуру і для них визначені та наперед задані властивості і характеристики, на основі яких будується нові компонентні системи. Однак практично у багатьох випадках базові компоненти піддаються деякій

попередній обробці – зміні інтерфейсів і деяких аспектів реалізації, конфігурації, налагодження і т.д. При цьому отримані об'єкти розглядалися як нові компоненти, які можуть використовуватися як елементи компонентних програм. Операції над базовими компонентами носять не довільний характер, а підкоряються визначеним правилам і умовам, а також потребують, щоб отриманий у результаті новий компонент теж був із усіма необхідними атрибутами і властивостями. Це необхідно для подальшого функціонування структур програмних систем у сучасних середовищах.

При перетворенні необхідно забезпечити цілісність отриманого компонента шляхом виконання сукупності правил, враховуючі деякі обмеження і залежності, які є складовими елементами компонента зі збереженням цільності структури, зовнішніх властивостей і характеристик.

Методи перетворення компонентів з метою отримання нових компонентів можливо поділити на зовнішні та внутрішні. До зовнішніх методів будимо відносити ті, що забезпечують обробку компонентів з урахуванням середовища та сервісу (розміщення, додавання тощо). Ці методи базуються на відповідних операціях O(mani) маніпулювання компонентами та їх інтерфейсами у середовищі. До внутрішніх методів будемо відносити методи рефакторінгу, реінжінірінгу та реверсного інжінірінгу. Вони виконують внутрішні перетворення компонента не залежно від середовища, а залежно від мети, що ставиться. Кожен з методів базується на трьох множинах операцій відповідно до методів, тобто O(Refac), O(Reing), O(rever). Усі ці множини операцій призначені для перебудови компонентів множини Comp = $\{Comp^i\}_{i=1,...,N}$ і разом складають алгебру, так звану компонентну алгебру, що складається із *зовнішній та внутрішній алгебр*:

$\Omega$ = {{Comp, O(mani)}, {Comp, O(Refac), O(Reing), O(rever)}},

де {Comp, O(mani)} – зовнішня алгебра;

{Comp, O(Refac), O(Reing), O(rever)} – внутрішня алгебра перетворення компонентів до нового призначення.

Серед множини компонентів Comp існують особливі компоненти-шаблони, для яких CInt = ∅ і CImp = ∅, тобто множини інтерфейсів і реалізацій – суть порожні множини:

TComp = (Template, ∅, CFact, ∅, CServ).

Умова цілісності компонента для шаблона повинна виконуватися, тому що множини вхідних інтерфейсів є порожня множина, незалежна від наявності або відсутності реалізацій, вираження має істинне значення.

Множини компонентів та операцій внутрішньої компонентної алгебри {Comp, O(Refac), O(Reing), O(rever)} пов'язані з множиною відповідних інтерфейсів, які забезпечують взаємодію компонентів із середовищем, де компоненти розташовані..

Далі визначаються методи та операції наведеної компонентної алгебри.

## 2. Метод та операції маніпулювання компонентами

Нехай OldComp позначає базовий компонент, для якого застосовуються операції із множини O(main), а NewComp відповідає компоненту після виконання операцій:

OldComp = (OldCName,OldCInt,CFact,OldCImp,CServ),

NewComp = (NewCName,NewCInt,CFact,NewCImp,CServ).

До операції маніпулювання відносяться операції:

- Sum – додавання компонентів;
- Locate **–** пошук і визначення находження необхідного екземпляра компонента;
- Create – створення екземпляра компонента;
- Remove – видалення екземпляра компонента;
- Link – об'єднання компонентів $Comp_3 = Comp_1 \cup Comp_2$

Тобто O(main) = {Sum, Locate, Create, Remove, Link }.

Ці операції в основному дуже прості і не потребують їх роз'яснення, крім першой.

**Операцію Sum –** додавання нової реалізації для компонента, суть якої полягає в тому, що в результаті множина інтерфейсів компонента може розширитися за рахунок додавання нових вихідних інтерфейсів, якщо реалізація, що додається, вимагає додаткової функціональності, наданої іншими компонентами. Позначимо додаткову множину вихідних інтерфейсів через $NewCIntO^s = \{NewCIntO^{sq}\}$. В окремому випадку $NewCIntO^s = \varnothing$, додаткова функціональність для реалізації, що додається, не потрібна.

Відповідно до моделі компонента, яка є абстрактним його представленням та відображає його основні властивості, характеристики, типову структуру, а також здатність до взаємодії з іншими елементами компонентного середовища, кожен компонент має декількох параметрів. Модель компонента в загальному випадку представляється так:

Comp = (CName, CInt, CFact, CImp, CServ), (1)

де, CName - унікальне ім'я компонента;

CInt = $\{CInt^i\}$ - множина інтерфейсів, зв'язаних з компонентом;

CFact - інтерфейс керування екземплярами компонента;

CImp = $\{CImp^j\}$ – множина реалізацій компонента;

CServ = {CServ$^r$} – інтерфейс, що визначає множину системних сервісів, необхідних для підтримки функціонування компонента і взаємодії з компонентним середовищем.

Ім'я компонента повинне бути унікальним у будь-якому просторі імен для компонентного середовища. Це означає, що в нього повинний входити єдиний компонент із заданим ім'ям. Для уникнення колізій застосовується метод кваліфікованих імен, при якому кожне ім'я входить у деякий простір імен. Керування такими просторами забезпечується спеціальними алгоритмами побудови просторів імен у виді ієрархічного дерева, де кожна вершина однозначно визначається маршрутом (послідовністю вершин) від кореня дерева.

Множина CInt = $Cint_1 \cup CInt_2$ складається з інтерфейсів двох типів. До першого типу $Cnt_1$ відносяться інтерфейси, що реалізуються в середовищі даного компонента, тобто мають відповідні реалізації методів. До другого типу $Cint_2$ відносяться інтерфейси, реалізовані в інших компонентах, але функціональність яких потрібно для виконання методів даного компонента.

**Операція розширення інтерфейсу Cfact** є комплексною, її реалізація зв'язана з існуванням нового типу компонента або контейнера. Отже, відповідно до класифікації операція розширення інтерфейсу керування екземплярами компонентів відноситься до зовнішній алгебри.

Кожен інтерфейс компонента представлений у такому вигляді:

$CInt^i = ( IntName^i , IntFunc^i , IntSpec^i )$

Де $IntName^i$ – ім'я інтерфейсу;

$IntFunc^i$ – функціональність (сукупність методів), що реалізовано даним інтерфейсом;

$IntSpec^i$ – специфікація інтерфейсу, що включає опис типів, констант, інших елементів даних, а також сигнатур методів і т.д.

Реалізація методів позначається через операцію Provide($CInt^i$), обумовлена інтерфейсом $CInt^i$ і надається деякою компонентною реалізацією.

Інтерфейс CFact реалізується за допомогою методів керування екземплярами компонента, в тому числі операціями, що входять до зовнішній алгебри, тобто

CFact = {Locate, Create, Remove}.

Кожну реалізацію компонента задамо так:

$CImp^j = ( ImpName^j , ImpFunc^j , ImpSpec^j )$,

де $ImpName^j$ – ідентифікатор ім'я реалізації компонента;

$ImpFunc^j$ – функціональність, реалізована даною реалізацією (або сукупністю реалізацій методів);

$ImpSpec^j$ – специфікація реалізації (опис умов виконання, опис параметрів настроювання реалізації і т.д.).

Реалізація компонента являє собою сукупність методів визначеної сигнатури і типами даних для переданих або параметрів, що повертаються після виконання методу. По цих сигнатурах і типам даних відбувається зіставлення реалізацій і інтерфейсів, що містять опис власних методів.

Необхідною вимогою для представлення змінного компонента є умова цілісності:

$(\forall CInt^i \in CInt)\ (\exists CImp^j \in CImp)\ Provide(CInt^i) \subseteq CImp^j$ ,

де знак $\in$ означає включення і забезпечення підтримки не тільки необхідного інтерфейсу, але й інших.

Операції роботи з середовищем, включають операції: інсталяція, видалення та заміщення компонентів. Розглянемо їх.

Операція інсталяції ins (розгорнення) компонента в компонентному середовищі – CE має вид:

$CE_2 = Comp \oplus CE_1$.

Операція видалення компонента з компонентного середовища це:

$CE_2 = CE_1 \setminus Comp$

Операцію заміщення deput компоненти $Comp_1$ компонентом $Comp_2$ можна виразити через операції $\oplus$ і \, тоді маємо $CE_2 = Comp_2 \oplus (CE_1 \setminus Comp_1)$.

Нехай $\Psi$ позначає множину операцій роботи з середовищем $\Psi = \{\oplus, \setminus, \cup\}$, тоді $\Omega_1$= {CSet, CESet, $\Psi$} визначає частину зовнішній компонентної алгебри, яка включає множини компонентів, компонентних середовищ і операції над їх елементами.

## 3. Метод та операції рефакторінга компонентів

До методів еволюції компонентів відноситься *рефакторінг* компонентів – сукупність моделей, методів і засобів перетворення, а також зміни структурних і якісних характеристик базових об'єктів (компонентів, програм) для отримання нових компонентів, найбільш придатних для побудови конкретної компонентної системи. Пропонується підхід до формального представлення методів рефакторінга компонентів та побудови відповідної алгебраїчної внутрішній моделі або внутрішньої алгебри операцій еволюції компонентів, тобто рефакторінга. Розглянемо основні визначення понять у проблемі рефакторінгу.

Підхід до рефакторінгу компонентів одержав розвитку у об'єктно-орієнтованому програмуванні [4] у зв'язку з застосуванням шаблонів проектування [5] і методами поліпшення коду і структури

об’єктно-орієнтованих програм. Як результат розроблені цілі бібліотеки типових трансформацій шуканих об'єктів (класів), що поліпшують ті чи інші характеристики [4].

**Метод рефакторінга** компоненту – це цільовий спосіб одержання нового компонента на базі існуючого, який включає операції модифікації (зміни, заміщення, розширення) елементів, що входять до складу компонента. Кожен метод характеризується метою і можливими послідовностями операцій перетворення складових компоненту. Він виконується за допомогою процесу зміни існуючого компонента з метою додання йому нових функціональних і структурних характеристик, що найбільше повно задовольняють вимоги компонентної конфігурації, до складу якої шуканий компонент повинний увійти. Фактично це означає таку зміну існуючого компонента, при якому він найбільше повно відповідає результатам проектування компонентної програмної системи.

Методи рефакторінга у компонентному програмуванні базуються на типовій структурі, моделі компонента та поділяється на зміни інтерфейсів компонентів (додавання, розширення і т.д) або реалізацій (додавання, розширення і т.д) та зміни інтерфейсів керування екземплярами компонентів (додавання нових функцій) або системними сервісами (шляхом додавання нових функцій).

**Операції рефакторінга компонентів.** Кожна операція є базовою, атомарною функцією перетворення, що зберігає цілісність компонента.

Розглядаються такі операції над компонентами, котрі задовольняють наступним умовам:

– об'єкт, отриманий у результаті рефакторінга є компонент з відповідними властивостями, характеристиками та типовою структурою;

– операція рефакторінгу компонента не утрачує функціональність, тобто новий компонент може застосуватися в раніше побудованих компонентних системах.

Існує досить складна і важлива проблема, що позв'язана з іменуванням та ідентифікацією нових компонентів. Методи рефакторінга можуть бути різними і, відповідно до визначення, отримані нові компоненти повинні однозначно ідентифікуватися. До операцій рефакторінга компонентів відносяться:

– додавання нової реалізації для існуючого та нового інтерфейсу;
– заміна існуючої реалізації нової з еквівалентною функціональністю;
– додавання нового інтерфейсу (за умови наявності відповідної реалізації);
– розширення існуючого інтерфейсу та інтерфейсу **Cfact;**
– розширення інтерфейсу **CServ (** для нових системних сервісів у компонентному середовищі).

**Модель рефакторінгу компонентів.** Для визначення моделі спочатку розглянемо множину операцій O(Refac) рефакторінгу:

**O(Refac) = {AddOImp, ReplImp, AddInt , Cfact},**

де AddOImp – додавання операції для існуючого інтерфейсу і нового;
ReplImp – операція заміщення існуючої реалізації новою;
AddInt – операція додавання нового вхідного інтерфейсу;
Cfact – операція розширення інтерфейсу.

Далі дамо визначення кожної операції.

**Операція AddOImp** додавання операції для існуючого інтерфейсу і нового має вигляд:

NewComp = AddOImp(OldComp, $NewCImp^s$,$NewCIntO^s$).

Умова цілісності компонента при цьому виконується автоматично, тому що множина вхідних інтерфейсів залишається колишнім і з цілісності старого компонента випливає цілісність нового. Операція AddOImp є асоціативною і комутативною операцією. Доказ цьогов випливає з аналізу множин інтерфейсів і реалізацій, що входять до складу відповідних компонентів та використовуються асоціативні і комутативні властивості операцій над множинами.

Різновидом цієї операції є AddNImp з формою запису

NewComp = AddNImp(OldComp,$NewCImp^s$,$NewCIntO^s$).

Для цей операції аналогічно доводиться й асоціативність і комутативність.

**Операція ReplImp** - заміщення існуючої реалізації нової має вигляд

NewComp=ReplImp(OldComp, $NewCImp^s$, $NewCIntO^s$, $OldCImp^r$, $OldCIntO^r$),

де $NewCImp^s$ - реалізація, що додається;
$NewCIntO^s$ – множина додаткових вихідних інтерфейсів для реалізації, що додається;
$OldCImp^r$ – реалізація, що заміщається;
$OldCIntO^r$ – множина вихідних інтерфейсів, зв'язаних з реалізацією, що заміщається.

**Операція AddInt** додавання нового вхідного інтерфейсу має вигляд

NewComp = AddInt(OldComp, $NewCIntI^q$).

Як видно із семантики операція має умовний характер та є частковою операцією.

У відмінності від операції розширення існуючої реалізації, для якого немає обов'язкової вимоги її схоронності в структурі компонента, всі існуючі вхідні інтерфейси повинні зберігатися. Ця вимога випливає з умов допустимості методів рефакторінга, розглянутих вище. Тому суть операції зводиться до додавання розширеного інтерфейсу як нового із семантикою, аналогічної попередній операції.

**Операція розширення інтерфейсу Cfact** є комплексною, її реалізація зв'язана з існуванням нового типу контейнера. Отже, відповідно до класифікації методів рефакторінга, що наведено вище, операція розширення інтерфейсу керування екземплярами компонентів відноситься до внутрішній алгебри.

Таким чином, отримали можливість визначити модель рефакторінгу

$M_{Refac}$ = {$O_{Refac}$ , {CSet = {$NewComp^{nь}$}} (2).

Для CSet = {$Comp_n$} – множин компонентів, кожний елемент, що входить до моделі (1), тоді $\Omega_1$= {CSet, $O_{Refac}$} визначає частину внутрішній компонентної алгебри, яка включає множини компонентів і операцій рефакторінга над їх елементами. Пари (Refac, CSet ) визначають елементи цієї компонентної алгебри щодо методу рефакторінга.

## 4. Метод та операції метода реінжінірінгу компонентів

Питанню реінжінірінга компонентів зараз приділяється ще мало уваги. Під **процесом реінжінірінгу** будимо розуміти повторну реалізацію компонента, програми або системи за компонентами (зробленої раніше без використання компонентної технології) шляхом зміни окремих функцій, додавання деяких нових властивостей, перепрограмування, підвищення зручності експлуатації або супроводу. Реінжінірінг включає такі процеси:

– реорганізація і реструктуризація компонента;
– переклад мови компонента на одну з більш сучасних мов програмування;
– модифікація або модернізація структури компонента і його даних.

При цьому архітектура системи за компонетами може залишаться незмінною.

З технічної точки зору реінжінірінг – це рішення проблеми еволюції системи за компонентами. Якщо врахувати, що архітектура такої системи не змінюється, те зробити централізовану систему розподіленої представляється справою досить складною. Важко також змінити мову програмування старої системи на нову (наприклад, Java на C++).

Однак з комерційної точки зору реінжінірінг приймають часто за єдиний спосіб збереження наслідуваних систем в експлуатації. Підходи до еволюції системи, при яких змінюється всі, є дорогими або ризикованими, тому що програмних систем є багато, а повна або радикальна заміна, реструктуризація їх у більшості випадках ускладнена.

За допомогою реінжінірінга удосконалюється системна структура, створюється нова документація і полегшується супровід системи. Супровід старих систем за компонентами дійсно коштує дорого, однак реінжінірінг може продовжити час їхнього існування.

У порівнянні з більш радикальними підходами до удосконалювання систем реінжінірінг має такі переваги:

– зниження ризиків за рахунок адаптації діючій системи із компонентів, а не повторна розробка при якої є висока імовірність помилок, наприклад у системній специфікації або при с розробки системи.

**– з**ниження витра**т** за рахунок реінжінірінгу компонентів, ніж повторна розробка нової системи. Згідно даним різних комерційних структур він в чотири рази дешевше, ніж повторна розробка системи за компонентами.

Реінжінірінг тісно пов'язаний зі зміною ділових процесів, орієнтованих на зниження кількості зайвих видів діяльності і підвищення ефективності ділового процесу. Ці обставини припускають упровадження нових компонентів або модифікацію існуючих. Залежність ділового процесу від наслідуваних систем необхідно виявляти заздалегідь до планування яких-небудь змін у самому бізнесі-процесі. Тому рішення про реінжінірінг виникає, коли наслідувану систему не вдається адаптувати до нових ділових процесів або до середовища його функціонування.

Основне розходження між реінжінірінгом і новою розробкою системи полягає в тому, що написання системної специфікації починається не з «нуля», а зі старої системи, на основі якої розробляється специфікація нової системи.

**Метод реінжінірінгу –** цільовий засіб отримання нового компоненту за рахунок послідовності операцій внесення змін, модернізації або модифікації, а також перепрограмування або адаптація компонентів.

**Реінжінірінг** компонентів – це сукупність моделей, методів та процесів зміни структури і можливостей компонентів з метою отримання компонента з новими можливостями.

До методів реінженерії програмних компонентів і систем з метою еволюції існуючих компонентів для продовження їхнього використання відносяться:

– аналіз вихідного коду для визначення логіки компоненту та внесення змін;
– настроювання компонентів системи на зміни в архітектурі або платформі, а також їх модифікація;
– кодування і декодування даних компонентів при переході з однієї платформи на іншу;
– модернізація системи за рахунок зміни окремих функцій системи за компонентами;
– розширення можливостей (функцій, сервісу й ін.) компонентів;
– перетворення структури системи або окремих її компонентів.

При виконані операцій реінжінірінгу необхідно щоб вони задовольняли наступним властивостям компонентів.

- Об’єкт, отриманий у результаті реінжінірінгу, є компонент з типовою структурою та зі своїми особливостями і характеристиками.

- Новий компонент після реінжінірінгу зберігає свої функції та можливість застосування в раніше побудованих системах за компонентами.
- Компонент, отриманий у результаті реінжінірінгу, відповідає вимогам і обмеженням моделі системи за компонентами.

Відтоді операції реінжінірінгу зміни, реструктуризації і адаптації складових елементів компонента, виконуються при умовах збереження існуючої функції або розширення функцій компонента у рамках тій же моделі системи за компонентами, де був розташовано компонент до реінжінірінгу.

До додаткових операцій реінжінірінгу будемо відносити:

- іменування компонентів та ідентифікація компонентів;
- розширення функцій існуючої реалізації компонента;
- перекладу мови компонента на нову сучасну мову програмування;
- реструктуризації структури компонента;
- модифікації опису компонента і його даних.

Нові компоненти необхідно ідентифікувати за визначеними діючими правилами іменування компонентів і методів їхнього застосування у системах за компонентами. Рішення проблеми ідентифікації пов'язано також з необхідністю створення компонентних конфігурацій та каркасів системи за компонентами.

**Операції реінжінірінгу компонентів O(Reing).**

Множина операцій реінжінірінгу має вигляд:

O(Reing) = {rewrite, restruc, adop, supp, conver},

де rewrite – операція перекладу мови компонента на одну з більш сучасних мов програмування;
restruc – операція реорганізації і реструктуризації компонента;
adop – операція адаптації до нових умов функціонування компонента;
supp – операція додавання до множини нових компонентів з новими функціями;
conver – операція конвертування даних компонента на іншу платформу.

Ці операції змінюють тільки компонент без зміни інтерфейсу, оскільки передбачається, що у якості інтерфейсу використовується запит-повідомлення (request) до середовища, де виготовлені компоненти розташовуються і виконуються. Запит до компонента формується у середовищі клієнта і відправляється через мережу серверу для виконання відповідного компонента, що викликається.

Нехай OldComp = (OldCName, OldCImp) позначає і'мя та реалізацію компонента до застосування операції реінжінірінгу, NewComp = (NewCName, NewCImp) після виконання цієї операції, а

OldCInt, CFact, OldCImp, Cserv = ∅.

Таким чином, **NewComp** отримують шляхом виконання операції rewrite, restruc, adop, supp, conver. В результаті маємо набір нових компонентів, які одержують шляхом виконання запропонованих операцій реінжінірінгу, у вигляді:

$NewComp^1$ = rewrite (OldComp, NewCImp),
$NewComp^2$ = restruc (OldComp, NewCImp),
$NewComp^3$ = adop (OldComp, NewCImp),
$NewComp^4$ = supp (OldComp, NewCImp),
$NewComp^5$ = conver (OldComp, NewCImp). (3).

Ці нові компоненти утворюють множину компонентів

Cset = { $NewComp^n$ }, де n = 1…5. (4).

Усі попередньої операції реінжінірінгу є асоціативними і комутативними. Вони забезпечують цілісність нового компонента. Тобто

$NewComp=OldComp \cup NewComp^n$

Аналогічно, як при рефакторінгу, має місце лема1 і лема 2 з тою різницію, що розглядається замість інтерфейсу повідомлення - requests, що розробляються для компонента відповідно середовища, де цей компонент розташовано. При цьому може бути декілька нових повідомлень – requests, які співставляються у відповідність до нових компонентів, що виникають у результаті виконання операцій реінжінірінгу. Вони розміщуються у тій множині компонентів і разом з компонентом створюють пару (NewComp, request). Для запитів можуть розглядатися операції їх розширення з вимогами схоронності їх окремо від структури компонента, наприклад у репозитарії інтерфейсів. Ця вимога випливає з умов допустимості методів реінжінірінгу до інтерфейсів та умов сучасних середовищ (CORBA, JAVA тощо). Суть операції додавання та розширення інтерфейсу є операцією аналогічною поповнення репозитарію.

Таким чином, розгляд операцій реінжінірінгу дозволяє додати внутрішню компонентну алгебру множиною компонентів CSet = {$NewComp^{nb}$} та операціями реінжінірінгу

$O_{Reing}$= {rewrite, restruc, adop, supp, conver }.

Виходячи з (3- 4) маємо модель реінжінірінгу компонентів у наступному вигляді

$M_{Reing}$ = { $O_{Reing}$ , {CSet = {$NewComp^{nb}$}}, (5)

де пара ($O_{Reing}$, CSet, ) визначає додатковий елемент внутрішній компонентної алгебри, тоді $\Omega_2$= {CSet, $O_{Reing}$} визначає частину цієї компонентної алгебри, що містить множини компонентів і операцій щодо реінжінірінгу.

Таким чином, запропоновані основні положення, методи, операції та модель реінжінірінгу компонентного програмування. Все це дає змогу з загальних теоретичних позицій розглядати проблему реінжінірінгу та вирішувати задачі внесення різного виду змін до компонентів та їх інтерфейсів.

## 5. Метод та операції реверсного інжінірінгу компонентів

Реверсний інжінірінг – це процес аналізу програмної системи з метою визначення її компонентів, їх відношень і структури кожної, а також побудови нового поліпшенного представлення системи на более високому мовному рівні або абстракції [ 7, 8]. Тобто після встановлення структури і логіки компонента провадиться перетворення його вихідного коду шляхом прямого перепрограмування або реконструювання початковій структури компонента та опис його у нову мову. Підтримка та еволюція діючих застарілих великих програмних систем залежить від їх розміру, складності та зрілості, оскільки в їх розробке приймають участь різні спеціалісти на протязі тривалого терміну інвестування їх замовниками. В[9] також стверджується, що такі системи мають декількох типових проблем: застарілі методи розробки, мови програмування і недороблену документацію. Тому їх еволюція є технічно дуже важкою і потребує забагато зусиль.

Склались і підходи автоматичного або автоматизованого перекладу застарілого вихідного коду з однієї мови програмування на іншій, більш сучасний. Це виконується, як правило, за допомогою спеціально створених програм конвертування коду з однієї мови на іншу з урахуванням наявності різних типів даних, або системи зіставлення зі зразком, представленим у виді списку команд, необхідного для перекладу із одного мовного представлення в інше. Переклад реалізується порівняним співставленням зі зразками і має місце на практиці у таких випадках:

– нові апаратні засоби, на яких не може виконуватися компілятор з вихідної мови компонента;
– недоліки в кваліфікації окремого персоналу для супроводу компонентів, написаних на специфічних мовах, що вийшли з уживання;
– зміни політики організації супроводу програмних систем в зв'язку з переходом на загальні сучасні ОС та мови програмування.

Автоматизований переклад стає неможливим, якщо структурні компоненти вихідного коду не мають відповідності в новій мові (наприклад, вихідний опис містить убудовані умовні команди компіляції, що не підтримуються в новій мові). В такому випадку провадиться ручне перетворення системи або окремих її компонентів, на що витрачаються значні технічні, людські та фінансові ресурси.

Таким чином, процес реверсного інжінірінгу програмній системи складається з двох шляхів:

1) аналіз опису компонента з метою отримання його структури та логіки опису;
2) перетворення діючій логіки опису компоненту у новий опис можливо зі змінами.

Перший шлях слугує здобуванню фактів про кожний компонент із його програмного тексту і встановлення відповідності його проектним рішенням для подальшого відновлювання компонента.

Другий шлях являє собою перетворення цього програмного коду у нову реалізацію. У випадку відновлення існуючої застарілої архітектури, виконується перепроектування відповідно новим вимогам. Здійснюється переоцінка нової архітектури для досягнення певних характеристик якості системи і обмежень різного виду. При цьому може виникнути необхідність переміщення компонентів попередньої системи в нове середовище, або їх повна заміна іншими, що перетворені.

В останні роки досліджені методи реверсного інжінірінгу відносно об’єктно-орієнтованого програмування [8]. Склался новий підход до реверсного інжінірінгу, що базується на виконанні базових операцій візуалізації (visual) та вимірювання поліметрик (metric) програмних систем в рамках ментальної моделі, яка пропонує наступні цілі:

– забезпечення високої якості системи і переогляд її в термінах розміру, скдадності та структури;
– пошук ієрархії классів та атритутів програмних об’єктів з метою наслідування їх у ядрі системи;
– ідентифікація класів об’єктів з визначенням розміру і /або складності усіх класів системи;
– пощук патернів, їх ідентифікація, а також фіксація їх місця та ролі у структурі системи.

Цей підхід орієнтовано на індустріальні системи у млн. строк коду з використанням метрик для оцінки характеристик системи. Він дозволяє генерацію тестів для перевірки кодів, а також проведення метричного аналізу системи для отримання фактичних значень внутрішніх та зовнішніх характеристик системи.

В результаті огляду системи будується модель, яка містить список проблематичних класів та патернів системи, які можуть модифікуватися і перепроектуватися, а також коло яких можливо організувати процес еволюції системи. Коли деякий клас погано спроектований (наприклад, багата методів і є пусті коди) або система не виконує потрібну роботу, то виконується збір інформації для метальній моделі.

У даному підході дії по візуалізації системи висвітлюється на двохрозмірному екрані у вигляді ієрархічного дерева. В ньому вузли відображають об’єкти і їх властивості, а відношення між ними задаються контурами команд фрагментів програм. При цьому застосовується таблиця метрик, в якої знаходяться свідомості про метрики класів об’єктів (число класів, методів, атрибутів, підкласів та строк

коду), метрики методів об'єктів (число параметрів, викликів, повідомлень тощо), метрики атрибутів об'єктів (час доступу, число доступів в класі та підкласі тощо).

В процесі візуалізації провадиться збір метричних даних про систему. Коли реально визначені усі дані о різних фактичних метриках програмної системи виконується оцінка якості та розробка плану перебудови застарілої системи на нову з отриманням тих же можливостей або нових додаткових.

**Операції реверсного інжінірінгу O(rever).**

Множина цих операцій включає:

O(Rever) = {visual, metric, restruc, design, rewrite}, (6)

де visual – візуальна операція представлення графу на екрані дисплея та інформації про структуру системи (число компонентів, відношень між ними тощо);

metric – операція оцінки стану системи за отриманими візуальними даними та метриками, що використовується при послідовної реструктуризації системи;

restruc – операція реорганізації і реструктуризації системи, як при операції реінжінірінгу відповідно метричному вимірюванню;

design – побудова нової компоненти системи по застарілій, яка не задовольняла новим умовам еволюції системи;

rewrite – операція перекладу мови окремих об'єктів–компонентів в іншу мову програмування відповідно до нової структури системи.

Виходячи із того, що запропоновані операції (6) мають відношення до елементів об'єктно-орієнтованих програмних систем, якими є об'єкти, їх методи та інтерфейси, що створюють множини {CSet = {NewObj}}, модель реверсного реінжінірінгу приймає наступний вигляд:

$M_{Rever}$ = { $O_{Rever}$, {CSet = {NewObj}}},

де пара ($O_{Rever}$, Cset) визначає додатковий елемент внутрішній компонентної алгебри, тоді $\Omega_3$= {CSet, $O_{Rever}$} визначає частину цієї алгебри, яка включає множини компонентів і операцій щодо реверсного інжінірінгу. Розглянутий підхід дає змогу вирішувати задачі еволюційного розвитку та перетворення застарілих систем по їх коду.

Таким чином, запропоновано внутрішня компонентна алгебра, що складається із трьох частин $\Omega = \{\Omega_1, \Omega_2, \Omega_3\}$, яким відповідають методи, операції та моделі еволюції програмних компонентів. Дано загальний погляд на усі три проблеми сучасного перетворення програмних систем

## Висновки

Запропоновані методи еволюції програмних компонентів, до яких відносяться рефакторінг, реінжінірінг та реверсний інжінірінг компонентів. Розглянуті їх операції. Побудовано компонентна алгебра, яка складається із зовнішній та внутрішній алгебр. До зовнішній алгебри віднесені множини компонентів та операцій обробки і маніпулювання компонентами. До внутрішній – множини компонентів та множини операцій рефакторінга, реінжінірінгу та реверсного інжінірінгу. Усі операції формально визначені та сформульовані умови цілісності компонентів при їх застосуванні. Запропоновано конструктивний підхід до встановлення попередній структури компонента по його вихідному коду. Отримані результати визначають необхідний теоретичний базис компонентного програмування, що може використовуватися при практичному створенні методик і методологій проектування компонентних систем із змінних компонентів та компонентів повторного застосування.

## Література:


1. *Грищенко В.Н., Лаврищева Е.М.* Методы и средства компонентного программирования // Кибернетика и системный анализ. - № 1, 2003. – С. 39-55.
2. *Грищенко В.Н., Лаврищева Е.М.* Компонентно-орентированное программирование. Состояние, направления и перспективы развития. // Проблемы программирования.- Киев, 2002, N 1-2. – С.80-90.
3. *Грищенко В.Н.* Систематизований підхід до визначення програмних компонентів // Проблемы программирования.- Киев, 2001, N 3-4, – С.23-30.
4. *Фаулер М.* Рефакторинг: улучшение соответствующего кода. – СПб.: Символ-Плюс, 2003. – 432 с.
5. *Грищенко В.Н.* Формальные модели компонентного программирования // Проблемы программирования. – 2003. - №2. - С.42-57.
6. *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Патерны проектирования. – СПб: Питер, 2001. – 368 с.
7. *E.Chikofky and I. Cross.* Reverse Engineering and Design Recovery. A Taxonomy, Software Engineering.– Jan., 1990, (ISSN 0098–5589).– p.324 –335.
8. *M.Lanza and S.Ducasse.* Polimetric Views –A lightweight Visual Approach to Reverse Engineering.– IEEETransaction on Software Engineering.– Sept., 2003, №3 (ISSN 0098–5589).– p.782-796.